ROCZNIKI POLSKIEGO TOWARZYSTWA MATEMATYCZNEGO
Seria III: MATEMATYKA STOSOWANA XVI (1980)

ms ptm 1973

Ministerstwo Nauki i Szkolnictwa Wyższego

Maciej M. Sysło (Wrocław)

# O złożoności obliczeniowej problemów kombinatoryki i teorii grafów*

(*Praca przyjęta do druku 2.5.1978*)

*To do better*
*What others do well*
Oreste Vaccari

**Spis treści**

1. Wstęp, 53
2. Złożoność obliczeniowa algorytmów, 55
3. Modele obliczeń, 58
4. Sposoby reprezentacji grafów a efektywność algorytmów, 61
5. Dolne oszacowania złożoności obliczeniowej, 62
6. Przykłady algorytmów optymalnych i „dobrych", 66
7. Problemy o złożoności wielomianowej, 78
8. Problemy, dla których niemożliwe jest istnienie algorytmów o złożoności wielomianowej, 78
9. Problemy NP-zupełne, 79
10. Zakończenie. Problemy, 92
11. Podziękowania, 93
Bibliografia, 93

## 1. Wstęp

Ostatnie lata są okresem bardzo intensywnego rozwoju kombinatoryki i teorii grafów oraz ich zastosowań. Powodów tego upatrywać należy między innymi w ogólności grafów jako tworów matematycznych, dzięki czemu są powszechnie używane do modelowania problemów związanych z dyskretnymi układami obiektów.

Zastosowania teorii grafów wymagają wielu algorytmów i, chociaż za najbujniejsze lata rozwoju samej teorii uznaje się ostatnie dwudziestolecie, to dopiero od niedawna coraz więcej uwagi zaczęto poświęcać konstrukcjom i analizie obliczeniowych metod i algorytmów tej teorii.

Powszechnie przyjmuje się, że jeden z pierwszych algorytmów podany został przez Euklidesa. Chociaż od tego czasu pojawiło się bardzo wiele różnych metod

* Jest to rozszerzony konspekt wykładu wygłoszonego na V Konferencji Zastosowań Matematyki, która odbyła się w Bukowcu, w dniach od 1 do 8 grudnia, 1976 roku.

obliczeniowych, to jednak początkowo, gdzieś od pierwszych lat 50-tych, głównym celem analizy algorytmów było wykazywanie poprawności proponowanych metod. Spowodowane było to tym, że obliczenia przeprowadzano ręcznie, a więc niewiele było mowy o ich pracochłonności, a już absolutnie ani słowa o przestrzeni potrzebnej dla zapisu danych, wielkości pomocniczych i ostatecznych wyników. Najbujniejszy okres rozwoju metod obliczeniowych rozpoczął się z chwilą pojawienia się maszyn cyfrowych. Początkowo jednak większy nacisk w analizie algorytmów położony był na zajętość pamięci niż na czas obliczeń, gdyż m.c. były wyposażone w stosunkowo niewielkie pamięci operacyjne i pozbawione były całkowicie pamięci pomocniczych, a z drugiej strony uważano, że już taka szybkość obliczeń, jaką dysponujemy używając m.c. powinna gwarantować możliwość rozwiązywania wszystkich problemów praktycznych. Baczniejszą uwagę na czasową pracochłonność algorytmów zaczęto zwracać z chwilą, gdy pamięci pomocnicze przestały być jakimkolwiek problemem.

Powszechnie uważa się, że Edmonds [26] był pierwszym, który zwrócił uwagę na wyraźne różnice między algorytmami, których czas obliczeń rośnie wielomianowo i wykładniczo ze wzrostem rozmiaru problemu. Od tego czasu jest to podstawowym kryterium klasyfikacji algorytmów kombinatorycznych.

Niniejsza praca nie pretenduje do kompletnego przeglądu wszystkich, czy chociażby najważniejszych algorytmów kombinatoryki i teorii grafów. Algorytmy zamieszczone tutaj ilustrują jedynie ogólniejsze rozważania nad złożonością obliczeniową problemów kombinatorycznych. Zakłada się, że Czytelnik zna podstawowe algorytmy kombinatoryki i teorii grafów.

Problemy związane z analizą algorytmów pojawiają się w naturalny sposób w sytuacji, gdy w przypadku ustalonego problemu dysponujemy już pewną rodziną algorytmów. Powstają wtedy następujące pytania: (a) jak porównywać ze sobą algorytmy, czyli jak oceniać dobroć algorytmów i rozstrzygać, który z dwóch jest lepszy, (b) w jakim kierunku prowadzić poszukiwania nowych, bardziej efektywnych metod, oraz (c) czy istnieją realne szanse znalezienia lepszych metod. Odpowiedzi na te pytania interesują zarówno programistów, a więc bezpośrednich użytkowników algorytmów, jak i teoretyków informatyki.

Przechodząc już do konkretnych rozważań winniśmy przede wszystkim uświadomić sobie znaczenie pojęcia „algorytm" oraz miejsce tych „tworów" w teorii obliczeń. Już na tym pierwszym kroku pojawia się problem, na który trudno jest dać całkowicie wyczerpującą i zadowalającą odpowiedź. W pierwszej części pracy przedstawiono w zarysie podstawowe modele obliczeń stosowane w analizie algorytmów kombinatorycznych. Na uwagę zasługuje fakt, że chociaż takie twory jak RAM i RASP są konstrukcjami teoretycznymi, to jednak do tego stopnia stoją blisko istniejących maszyn cyfrowych, że przy niewielkich założeniach dodatkowych, programy m.c. napisane w dowolnym języku programowania mogą być uznane za programy maszyn RAM i RASP. Na bazie przyjętych modeli obliczeń możemy następnie określić miary złożoności algorytmów, używane do oszacowania pracochłonności obliczeń, wyznaczania oszacowań dolnych złożoności oraz porównywania różnych algorytmów rozwiązywania tego samego problemu.

W dalszej części pracy omówione zostały problemy, dla których istnieją algorytmy optymalne i/lub „dobre". Przedstawione w tej części pracy problemy zaliczane są do klasy ***P***, tj. klasy problemów, które mogą być rozwiązywane za pomocą algorytmów o złożoności wielomianowej. Dalej podana jest bardziej formalna definicja klasy ***P*** oraz klasy ***NP***, do której należą z małymi wyjątkami wszystkie trudne problemy kombinatoryczne, takie jak problem plecakowy, problemy szeregowania, problemy dróg Hamiltona w grafach i w sieciach itd. Pytanie, czy ***P*** = ***NP***, jest obecnie podstawowym problemem badawczym w analizie złożoności obliczeniowej algorytmów kombinatorycznych.

Praca nie zawiera żadnego wprowadzenia podstawowych terminów teorii grafów, kombinatoryki i programowania matematycznego, które Czytelnik znaleźć może odpowiednio w książkach Harary'ego [45], Liu [79] i Lawlera [75]. Bibliografia zawiera także opracowania niecytowane, nie pretenduje jednak do kompletnej bibliografii prac poświęconych złożoności obliczeniowej. Dla uzupełnienia polecamy uwadze Czytelnika opracowania Reingolda [94] oraz Irlanda i Fischera [56]. Bardziej aktualne bibliografie znaleźć można w nowszych opracowaniach książkowych Aho, Hopcroft i Ullman [2], Coffman [12] i Rustin [100] oraz w pracach Karpa [61] i Tarjana [114].

## 2. Złożoność obliczeniowa algorytmów

Dysponując rodziną algorytmów rozwiązujących dany problem powinniśmy umieć porównywać je, a więc umieć odpowiadać na pytania, który z dwóch algorytmów jest lepszy oraz jakie są szanse, że poszukując nowych metod rozwiązywania, znajdziemy metodę jeszcze lepszą. W tym celu musimy najpierw określić miarę *złożoności* i pojęcie *optymalności* algorytmu, a więc co przyjmujemy za ocenę dobroci algorytmu oraz kiedy algorytm możemy uznać za najlepszy.

Najpowszechniejszymi miarami złożoności obliczeniowej algorytmów są czas obliczeń oraz zajętość pamięci wyznaczane — jako funkcje *rozmiaru* problemu. Za *rozmiar* problemu przyjmujemy ten parametr (lub zbiór parametrów), który określa długość układu danych wejściowych. Dla przykładu, rozmiarem problemu polegającego na wyznaczeniu pewnej funkcji macierzy może być jej wymiar, a za rozmiar problemu teorii grafów przyjmujemy zwykle liczbę wierzchołków lub/i liczbę krawędzi. Obie miary złożoności obliczeniowej mogą być wyznaczane na dwa różne sposoby, a mianowicie, jako wielkości oczekiwane lub w wyniku analizy „najgorszych" danych. *Oczekiwana złożoność* oraz *złożoność najgorszego przypadku* jako funkcje rozmiaru problemu są odpowiednio średnią i maksymalną złożonością algorytmu wyznaczonymi na podstawie wszystkich możliwych układów danych. Wyznaczenie oczekiwanej złożoności jest zwykle trudniejsze, gdyż wymaga najpierw określenia i zbadania rozkładu prawdopodobieństwa układów danych, co w konkretnych przypadkach nastręcza wiele kłopotów.

Dalej mówić będziemy tylko o złożoności algorytmów wyznaczanej w wyniku analizy najgorszego przypadku danych. Złożoność obliczeniowa algorytmu określa

jednocześnie rozmiary tych problemów, które mogą być rozwiązywane za jego pomocą i przy użyciu konkretnej maszyny cyfrowej.

Najczęściej złożoność algorytmu jest wyznaczana i podawana z dokładnością do stałego współczynnika proporcjonalności i z uwzględnieniem tylko najistotniejszych członów. Na przykład, jeżeli czas obliczeń za pomocą jakiegoś algorytmu wynosi $cn^2+dn$, gdzie $c$ i $d$ są stałymi, to mówimy, że jego złożoność jest $O(n^2)$ i czytamy „rzędu $n^2$". Ogólnie, funkcja $g(n)$ jest $O(f(n))$, jeżeli istnieje stała $c$ taka, że $g(n) \leqslant cf(n)$ dla prawie wszystkich $n$. Jak łatwo zauważyć, symbol $O(\cdot)$ używany jest dla określania górnych oszacowań. Dla określenia dolnych oszacowań używać będziemy natomiast oznaczenia $\Omega(\cdot)$ o następującym znaczeniu: funkcja $g(n)$ jest $\Omega(f(n))$, jeżeli istnieje stała $c$ taka, że $cf(n) \leqslant g(n)$ dla prawie wszystkich $n$.

W dalszej części tego paragrafu, na przykładzie jednego problemu, zilustrujemy rzeczywisty powód wzrostu szybkości obliczeń za pomocą maszyn cyfrowych. Wydawać by się mogło, że wzrost szybkości obliczeń spowodowany rozwojem maszyn cyfrowych zmniejsza znaczenie badań nad efektywnością algorytmów. Jest jednak inaczej. To właśnie złożoność obliczeniowa algorytmu określa wzrost rozmiaru problemów, które stają się rozwiązywalne ze wzrostem szybkości maszyn cyfrowych.

W tablicy 1 podane są niektóre algorytmy badające, czy dany graf zwykły jest płaski. Do 1963 roku nie istniał żaden specjalny algorytm, a jedyna metoda polegała na bezpośrednim zastosowaniu kryterium Kuratowskiego, co w konsekwencji prowadziło do zbadania wszystkich podgrafów o 5-ciu i 6-ciu wierzchołkach. Dla informacji podajmy, że kryterium Kuratowskiego zostało udowodnione w 1930 roku. Następnie, potrzeba było już tylko niecałej dekady dla otrzymania algorytmu, który jest optymalny z dokładnością do stałego współczynnika proporcjonalności.

W czwartej kolumnie podany jest czas obliczeń za pomocą algorytmów $A_1$–$A_5$ przy założeniu, że stała proporcjonalności $k$ jest taka sama dla wszystkich algorytmów i wynosi $k = 10$ msec, a rozmiar zagadnienia, w tym przypadku liczba wierzchołków grafu $n$, wynosi 100. W następnych dwóch kolumnach podane są maksymalne rozmiary zagadnień, które mogą być rozwiązane w ciągu 1 min i 1 godz., odpowiednio, za pomocą algorytmów $A_1$–$A_5$ przy tym samym założeniu o stałej proporcjonalności. Dodatkowo oznaczmy przez $A_0$ jakikolwiek algorytm o złożoności $O(2^n)$.

**Tablica 1**

| Algorytm | | | Czas obliczeń | $n$ | |
|---|---|---|---|---|---|
| Nazwa | Autor | Złożoność | $k = 10$ msec<br>$n = 100$ | czas obliczeń<br>1 min | czas obliczeń<br>1 h |
| $A_1$ | Kuratowski — do 1963 r. | $O(n^6)$ | 325 lat | 4 | 8 |
| $A_2$ | Goldstein — od 1963 r. | $O(n^3)$ | 2.8 h | 18 | 71 |
| $A_3$ | Lempel i inni — od 1967 r. | $O(n^2)$ | 100 sek | 77 | 600 |
| $A_4$ | Hopcroft i Tarjan — od 1971 r. | $O(n\log n)$ | 7 sek | 643 | 24 673 |
| $A_5$ | Tarjan — od 1971 r. | $O(n)$ | 1 sek | 6 000 | $36 \cdot 10^4$ |

Przypuśćmy teraz, że następna generacja maszyn cyfrowych będzie dziesięć razy szybsza od obecnej. W tablicy 2 przedstawiono, jak wzrosną maksymalne rozmiary zagadnień, które będą mogły być rozwiązywane za pomocą algorytmów $A_0$–$A_5$ dzięki takiemu wzrostowi szybkości obliczeń na m.c. ($s_i$ oznacza maksymalny rozmiar zagadnienia, które może być rozwiązywane obecnie za pomocą algorytmu $A_i$, $i = 0, 1, \ldots, 5$).

Tablica 2

| Algorytm | | Maksymalny rozmiar zagadnienia | |
|---|---|---|---|
| Nazwa | Złożoność | przed wzrostem szybkości m.c. | po 10-krotnym wzroście szybkości m.c. |
| $A_0$ | $O(2^n)$ | $s_0$ | $s_0+3.3$ |
| $A_1$ | $O(n^6)$ | $s_1$ | $1.46s_1$ |
| $A_2$ | $O(n^3)$ | $s_2$ | $2.15s_2$ |
| $A_3$ | $O(n^2)$ | $s_3$ | $3.16s_2$ |
| $A_4$ | $O(n\log n)$ | $s_4$ | $10s_4$ dla dużych $s$ |
| $A_5$ | $O(n)$ | $s_5$ | $10s_5$ |

Zauważmy, że 10-krotny wzrost szybkości m.c. powoduje jedynie wzrost o 3 rozmiaru zagadnień, które mogą być rozwiązywane za pomocą algorytmu $A_0$, a 3-krotny wzrost w przypadku algorytmu $A_3$. Z drugiej zaś strony, na podstawie tablicy 1, w czasie 1 minuty, zastępując algorytm $A_2$ przez $A_3$ możemy rozwiązywać problemy 4 razy większe, a przez $A_4$ — aż 35 razy większe. Porównanie to wywiera o wiele większe wrażenie niż jedynie dwukrotny wzrost rozmiaru zagadnień rozwiązywalnych za pomocą $A_2$ osiągnięty dzięki aż dziesięciokrotnemu zwiększeniu szybkości maszyn cyfrowych.

W tym miejscu należy zwrócić jeszcze uwagę na znaczenie stałej proporcjonalności, która może być różna dla różnych algorytmów, a dla ustalonego problemu jest zwykle większa dla algorytmów o mniejszym rzędzie złożoności. W ten sposób, algorytmy asymptotycznie mniej efektywne mogą być szybsze dla zagadnień o interesujących nas rozmiarach. Dla przykładu załóżmy, że rzeczywiste złożoności algorytmów $A_0$, $A_3$, $A_4$ i $A_5$ wynoszą $2^n$, $100n^2$, $100n\log n$ i $1000n$, odpowiednio. Wtedy, algorytm $A_0$ jest najszybszy, gdy $2 \leqslant n \leqslant 10$, $A_3$ — gdy $10 \leqslant n < 59$, $A_4$ — gdy $59 \leqslant n < 1024$ i wreszcie $A_5$ dopiero, gdy $1024 \leqslant n$.

Istnieje wiele „trudnych" problemów obliczeniowych. W paragrafie 9 przedstawimy klasę problemów kombinatorycznych, z których obecnie żaden nie ma algorytmu o złożoności rzędu $n^c$, gdzie $n$ jest rozmiarem problemu, a $c$ stałą. Rozpatrzmy dla przykładu problem badania, czy dwa grafy są izomorficzne. Złożoność jakiejkolwiek metody rozwiązywania tego problemu zachowuje się asymptotycznie tak, jak $n!$, a więc rośnie szybciej niż $n^c$ dla jakiejkolwiek stałej $c$. Załóżmy, że stosujemy bezpośrednią metodę sprawdzania, czy dwa grafy $G_1$ i $G_2$ są izomorficzne, tj. najpierw zmieniamy porządek wierzchołków grafu $G_1$, a później porównujemy czy

$G_1$ jest identyczny z $G_2$. Niech $10^{-6}n^2$ sec będzie czasem potrzebnym do wykonania tych dwóch czynności dla ustalonego uporządkowania wierzchołków.

Tablica 3

| $n$ | czas |
|---|---|
| 10 | 6 min. |
| 15 | 9 lat |
| 20 | $3 \cdot 10^5$ stulecia |

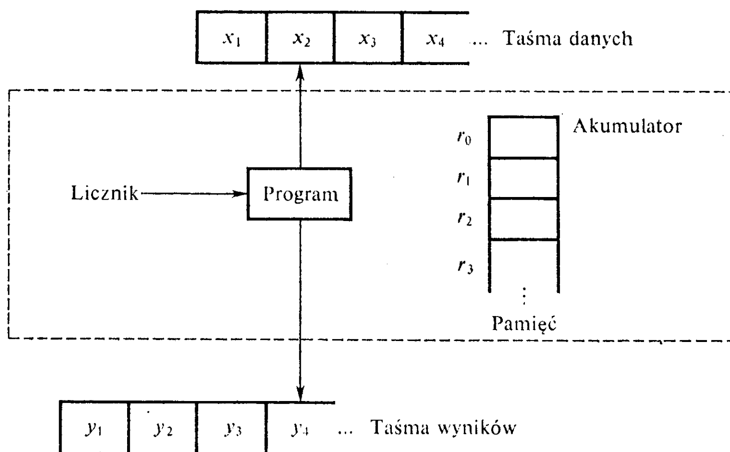
W tablicy 3 podane zostały czasy działania tej bezpośredniej metody w przypadku grafów o 10, 15 i 20 wierzchołkach.

## 3. Modele obliczeń

Aby móc kontynuować analizę algorytmów i określać ich złożoność obliczeniową, musimy najpierw sprecyzować model „urządzenia" wykonującego obliczenia, między innymi po to, aby określić elementarne operacje algorytmu. Jednym z przykładów problemu, którego rozwiązanie wymaga dokładnego określenia podstawowych kroków algorytmu, jest wyznaczanie dolnych oszacowań złożoności (paragraf 5), które z kolei wykorzystywane są w dowodach optymalności algorytmów (paragraf 6). Niestety, nie istnieje żaden model obliczeń, który byłby odpowiedni dla wszystkich rozważań. Tym bardziej żadna z istniejących m.c. nie może być przyjęta za standardową m.c. Jedną z podstawowych trudności pojawiających się przy wyborze modelu obliczeń jest problem ograniczonej i różnej długości słowa maszynowego. Natomiast dokonując wyboru teoretycznego modelu obliczeń, powinniśmy zadbać o to, aby późniejsze rozważania i otrzymane wyniki odnosiły się do możliwie najszerszej klasy istniejących maszyn cyfrowych.

*Maszyny Turinga* i *funkcje rekurencyjne* były jednymi z pierwszych modeli obliczeń i algorytmów. Zwłaszcza maszyny Turinga są bardzo użyteczne w rozważaniach teoretycznych, chociaż odbiegają dość daleko od istniejących maszyn cyfrowych. Podobnie, funkcje rekurencyjne są wygodnym aparatem matematycznym dla analizy algorytmów, ale dla rzeczywistych metod obliczeniowych na istniejących m.c. są modelem dość uciążliwym. Bardziej realistycznymi modelami maszyn cyfrowych zaakceptowanymi dość szeroko w analizie złożoności obliczeniowej są maszyny o jednakowym czasie dostępu do pamięci bez programu i z programem w pamięci, które w skrócie nazywać będziemy odpowiednio RAM (*random access machine*) i RASP (*random access stored program machine*). Bardzo szczegółowy opis maszyn Turinga, RAM i RASP, znaleźć można w książce [2], a tutaj przytoczymy jedynie najistotniejsze dla naszych rozważań cechy RAM i RASP, tych wyidealizowanych maszyn cyfrowych. Te trzy typy modeli są równoważne, jeśli chodzi o możliwości obliczeń, nie są jednak jednakowo szybkie.

RAM składa się z taśmy danych, która może być tylko czytana, taśmy wyników, na której można tylko pisać, pamięci i programu. Podstawowymi symbolami obu taśm są liczby całkowite. Pamięć składa się z ciągu rejestrów $r_0, r_1, \ldots,$ z których każdy zdolny jest przechowywać dowolnie wielką liczbę całkowitą. Nie ma także żadnego ograniczenia nałożonego na liczbę rejestrów. Program dla maszyny RAM nie jest przechowywany w pamięci, zakładamy więc, że nie może się modyfikować. Wszystkie obliczenia odbywają się w wyróżnionym rejestrze $r_0$ nazywanym akumulatorem. Program jest ciągiem instrukcji, które dodatkowo mogą być poprzedzone etykietami. Dokładna postać instrukcji nie jest istotna w rozważaniach, możemy jednak założyć, że mamy do naszej dyspozycji wszystkie instrukcje występujące w istniejących maszynach cyfrowych, a więc instrukcje arytmetyczne i logiczne, instrukcje wejścia i wyjścia, instrukcje skoku itp. Rysunek 1 przedstawia schemat RAM.




Rys. 1

W ogólności, program maszyny RAM definiuje odwzorowanie taśmy danych na taśmę wyników. Okazuje się, że RAM, podobnie jak wiele innych modeli maszyn cyfrowych, może obliczać funkcje częściowo rekurencyjne. Interpretując z kolei program maszyny RAM jako akceptor *języka*, tj. akceptor zbioru łańcuchów złożonych z symboli pewnego skończonego zbioru zwanego *alfabetem*, można udowodnić, że język jest akceptowalny przez program maszyny RAM wtedy i tylko wtedy, gdy jest on językiem rekurencyjnie przeliczalnym.

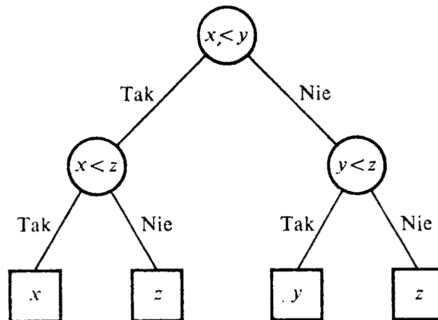
Aby móc wyznaczać złożoność programów maszyny RAM musimy najpierw określić czas potrzebny do wykonania poszczególnych instrukcji oraz pamięć zajmowaną przez każdy z rejestrów. Powszechnie rozważa się dwie funkcje kosztów wykonania instrukcji programu maszyny RAM. Przy założeniu *jednostajnej funkcji kosztów* wykonanie każdej instrukcji przebiega w jednostce czasu, a każdy rejestr

zajmuje jedną jednostkę pamięci. Druga funkcja kosztów, *logarytmiczna*, jest w pewnym sensie bardziej realistyczna, gdyż uwzględnia oczywisty fakt, że długość słowa pamięci jest ograniczona. W tym drugim przypadku zakłada się także, że koszt wykonania dowolnej instrukcji jest proporcjonalny do długości argumentów.

Różnica pomiędzy maszynami RAM i RASP polega na tym, że programy tej drugiej maszyny mogą być przechowywane w pamięci, a więc mogą się modyfikować. Złożoność równoważnych, w sensie funkcjonalnym, programów maszyn RAM i RASP jest taka sama z dokładnością do stałego współczynnika proporcjonalności, bez względu na rodzaj przyjętej funkcji kosztów.

Maszyny RAM i RASP są w wielu przypadkach nazbyt ogólnymi modelami dla rzeczywistych obliczeń. Możemy jednak wprowadzić uproszczone wersje maszyn RAM i RASP, które dla przykładu będą uwzględniały tylko pewne możliwości (instrukcje) najogólniejszych modeli. W ten sposób otrzymujemy maszyny RAM wykonujące tylko programy bez instrukcji skoku, programy złożone z instrukcji, których argumentami i wynikami mogą być tylko bity (tj. liczby 0 lub 1) lub wektory złożone z bitów oraz programy, których podstawę stanowią instrukcje rozgałęziania.

Zatrzymajmy się na chwilę przy modelu RAM z programami ostatniego typu. Istnieje duża klasa problemów obliczeniowych, których podstawowymi instrukcjami są porównania, na przykład problemy porządkowania ciągu liczb czy wyznaczania $k$-tego elementu co do wielkości. Porównanie jest oczywiście instrukcją typu rozgałęziania. Najczęściej tego typu programy przedstawione są w postaci drzewa binarnego zwanego *drzewem decyzyjnym*, którego wierzchołki pośrednie odpowiadają instrukcjom, krawędzie — decyzjom powziętym w wierzchołku, z którego wychodzą, a wierzchołki końcowe — ostatecznym wynikom.




Rys. 2

Rysunek 2 przedstawia drzewo decyzyjne programu, który wyznacza najmniejszą spośród liczb $x$, $y$, $z$.

W § 5 pokażemy, w jaki sposób wyprowadzane są dolne oszacowania złożoności obliczeniowej algorytmów, które mogą być modelowane za pomocą drzew decyzyjnych.

## 4. Sposoby reprezentacji grafów a efektywność algorytmów

Najczęściej stosowane sposoby reprezentacji grafu (np. w pamięci maszyny cyfrowej) można w ogólności podzielić na dwie grupy.

Do pierwszej grupy zaliczamy metody macierzowe, a więc np. reprezentacje grafu za pomocą macierzy sąsiedztwa wierzchołków, macierzy incydencji lub macierzy sąsiedztwa krawędzi. Z jednym tylko wyjątkiem w przypadku ostatniego sposobu (zobacz twierdzenie Junga, [45], str. 72), nieocechowany graf jest określony przez jedną z tych macierzy z dokładnością do izomorfizmu, a więc macierze te są odpowiednimi metodami reprezentacji grafu. Dla wielu zagadnień, takich jak obliczanie długości i generowanie dróg ekstremalnych między każdą parą wierzchołków w grafie czy wyznaczanie tranzytowego domknięcia, metody macierzowe są podstawowymi sposobami zapisu grafu.

Do drugiej grupy zaliczamy wszelkiego rodzaju struktury listowe odzwierciedlające przede wszystkim sąsiedztwo wierzchołków grafu, ale nie tylko. Najprostszym przykładem reprezentacji z tej grupy jest zbiór list $\{A(v)\}$, gdzie $A(v)$ jest zbiorem następników (dla digrafów) lub zbiorem sąsiadów (dla grafów zwykłych) wierzchołka $v$.

Wspomniane reprezentacje grafu różnią się między sobą przede wszystkim zajętością miejsca (np. w pamięci m.c.). W tym sensie, o metodach macierzowych możemy powiedzieć, że w wielu przypadkach są bardzo nieoszczędnymi sposobami reprezentacji. Zauważmy, że jeżeli jakiś algorytm polega na wykonaniu w pętli pewnych czynności kolejno dla wszystkich bezpośrednich połączeń w grafie, to wszystkie zera macierzy sąsiedztwa (a więc elementy, które nie odpowiadają połączeniom) możemy w pewnym sensie uznać za nieistotne. Jak się przekonamy w dalszej części tego paragrafu, reprezentacje grafu mogą mieć także istotny wpływ na efektywność (tj. liczbę kroków) algorytmu. Tak więc, przystępując do poszukiwań i konstrukcji nowych metod rozwiązywania problemów teorii grafów powinniśmy umieć określić jaki wpływ na złożoność obliczeniową algorytmów mogą mieć sposoby reprezentacji grafów.

Rozpatrzmy dla przykładu szczególne podklasy grafów zwykłych, a mianowicie drzewa, grafy płaskie i grafy zewnętrznie płaskie. Dla ustalonej liczby wierzchołków $n$, liczba krawędzi drzewa wynosi dokładnie $m = n-1$, a koniecznymi warunkami płaskości i zewnętrznej płaskości grafu są odpowiednio nierówności $m \leqslant 3n-6$ i $m \leqslant 2n-3$. Jak widać, liczba krawędzi w grafie należącym do którejś z tych klas jest ograniczona przez liniową funkcję liczby wierzchołków, a więc drzewa i grafy płaskie mogą być reprezentowane za pomocą list sąsiadów $\{A(v)\}$ o sumarycznej liczbie elementów $O(n)$, a każdy ze wspomnianych wyżej sposobów macierzowych wymaga co najmniej $\Omega(n^2)$ elementów.

Hopcroft i Tarjan [54] jako jedni z pierwszych, dzięki użyciu listowej reprezentacji grafu zwykłego otrzymali liniowy algorytm badania czy graf jest płaski i dodatkowo wykazali nieformalnie, że wszystkie inne algorytmy badania płaskości grafu korzy-

stające z macierzowych reprezentacji grafu wymagają co najmniej $\Omega(n^2)$ operacji, a więc nie są algorytmami optymalnymi.

Podobnie jest w przypadku badania zewnętrznej płaskości grafu zwykłego (patrz Sysło i Iri [108]).

Jedną z pierwszych prac poświęconych wpływowi sposobu reprezentacji grafu na efektywność metod badania pewnych własności digrafów jest praca: Holt i Reingold [47], w której udowodniono, że jeżeli $n$-wierzchołkowy digraf reprezentowany jest za pomocą macierzy sąsiedztwa wierzchołków, to dowolny algorytm wykrywania konturów lub określania czy digraf jest silnie spójny wymaga sprawdzenia co najmniej $\Omega(n^2)$ elementów macierzy (patrz następny paragraf).

Stąd wnioskujemy, że znane algorytmy wykrywania konturów w digrafie korzystające z macierzowej reprezentacji digrafu są optymalne z dokładnością do stałego współczynnika proporcjonalności, zobacz algorytm Marimonta [80] oraz jego realizację w książce Kucharczyka i Sysły [71]. Istnieje realizacja algorytmu Marimonta korzystająca z listowego zapisu digrafu, która dla grafu o $m$ łukach wymaga $O(m)$ dodawań i porównań.

Mówimy, że $P$ jest *nietrywialną* własnością (di)grafu, jeżeli

(a) dla każdego $n$ istnieją dwa $n$-wierzchołkowe grafy $G_n$ i $\overline{G}_n$, z których jeden ma, a drugi nie ma własności $P$,

(b) spełnienie tej własności nie zależy od istnienia pętli w grafie, oraz

(c) nie zależy od sposobu ponumerowania wierzchołków grafu.

Rosenberg [98] sformułował przypuszczenie, że aby stwierdzić czy $n$-wierzchołkowy graf ma jakąś nietrywialną własność, należy zbadać $\Omega(n^2)$ elementów macierzy sąsiedztwa wierzchołków. Przypuszczenie to następnie zostało obalone przez Aanderaa, który pokazał, że sprawdzenie, czy $n$-wierzchołkowy digraf ma *odpływ*, tj. wierzchołek o $n-1$ łukach wchodzących i bez łuków wychodzących (łatwo sprawdzić, że jest to nietrywialna własność digrafu) wymaga zbadania $3n-3$ elementów macierzy sąsiedztwa wierzchołków. Własność ta przestaje być kontrprzykładem, gdy zażądamy dodatkowo, aby $P$ była własnością *monotoniczną*, tj. taką, że jeżeli $D = (V, E)$ ma własność $P$, to także dowolny (di)graf $D' = (V, E')$ taki, że $E \subseteq E'$ ma własność $P$. Tak zmodyfikowane przypuszczenie Aanderaa-Rosenberga zostało udowodnione przez Rivesta i Vuillemina [97], którzy pokazali, że dowolna nietrywialna i monotoniczna własność grafu zwykłego wymaga w najgorszym przypadku zbadania co najmniej $n^2/9$ elementów macierzy sąsiedztwa, gdzie $n$ jest liczbą wierzchołków grafu.

## 5. Dolne oszacowania złożoności obliczeniowej

Przed przystąpieniem do wyznaczania dolnego oszacowania złożoności obliczeniowej algorytmów musimy ustalić najpierw (a) podstawowe zadanie, które staramy się rozwiązać, (b) typ lub rodzinę branych pod uwagę algorytmów, oraz (c) co składać się będzie na dolne oszacowanie. Zauważmy, że odpowiedzi na dwa ostatnie

pytania są zwykle ściśle związane z odpowiedzią na pierwsze, gdyż sam problem narzuca zwykle rodzaj algorytmów i typ działań podstawowych, które mogą być stosowane do jego rozwiązywania.

Mimo wszystko odpowiedź na pierwsze pytanie, a nawet i na drugie nie określają definitywnie sposobu obliczania złożoności algorytmu, a co za tym idzie, i oszacowań. Idealnym rozwiązaniem byłoby określenie rzeczywistego kosztu algorytmu przy uwzględnieniu wszystkich operacji, jednak takie podejście bardzo utrudnia otrzymanie jakiegokolwiek rozwiązania. Istnieją jednak takie próby, patrz paragraf 6.13, gdzie omówiono realizację metody Strassena, nieklasycznego sposobu mnożenia macierzy. Najczęściej wyodrębnia się operacje podstawowe, które mają największy wpływ na złożoność algorytmu, a co za tym idzie, i na czas obliczeń. Te ogólne uwagi odnoszą się zarówno do wyznaczania oszacowań dolnych, jak i do określania samej złożoności obliczeniowej. Oszacowania dolne mogą być także określane bądź dla przypadku najgorszych danych, bądź dla złożoności oczekiwanej (średniej).

Jak dotychczas, brak jest jakichkolwiek ogólnych metod wyznaczania oszacowań dolnych złożoności obliczeniowej różnego typu obliczeń, nawet w przypadku ograniczenia rozważań do klasy zagadnień kombinatorycznych lub obliczeniowych problemów teorii grafów. Przedstawimy teraz szereg metod o różnym zasięgu stosowania, których powstanie było często związane z wprowadzeniem dość mocnych ograniczeń na typ problemu, rodzinę algorytmów oraz typ operacji podstawowych.

Jedna z najogólniejszych metod otrzymywania oszacowań minimalnej liczby działań oparta jest na następującym rozumowaniu. Załóżmy, że rozpatrywany problem można scharakteryzować za pomocą $n$ wielkości liczbowych, z których każda ma istotny wpływ na wynik obliczeń, oraz że poszukujemy algorytmu w klasie metod wykonujących jedynie elementarne operacje o co najwyżej $k$ argumentach. Można łatwo wykazać, że przy tych założeniach dowolny algorytm dla naszego problemu musi wykonywać co najmniej $n/k$ elementarnych operacji po to tylko, aby wszystkie istotne elementy ciągu danych zostały uwzględnione [57]. Na tej podstawie, jeżeli graf ma $n$ wierzchołków i $m$ łuków, a o rozpatrywanym problemie wiemy skądinąd, że wymaga wzięcia pod uwagę wszystkich łuków, to złożoność obliczeniowa dowolnego algorytmu rozwiązywania tego problemu jest co najmniej $\Omega(n^2)$ lub $\Omega(m)$, w zależności od przyjętej reprezentacji grafu (patrz także § 4 o wpływie reprezentacji grafu na złożoność obliczeniową algorytmów). Załóżmy na moment, że badamy pewne własności grafów, w których liczba krawędzi ograniczona jest przez liniową funkcję liczby wierzchołków (drzewa i grafy płaskie są przykładami takich klas grafów). Na podstawie powyższych rozważań możemy wywnioskować, że w takim przypadku tylko algorytmy wykorzystujące listową reprezentację grafu mogą okazać się optymalnymi lub optymalnymi z dokładnością do stałego współczynnika proporcjonalności.

Przedstawimy teraz jedno z najwcześniejszych oszacowań złożoności obliczeniowej algorytmów, podane przez Holta i Reingolda [47]. Digraf $G$ o $n$ wierzchołkach ponumerowanych kolejnymi liczbami od 1 do $n$ zawiera *kontur*, jeżeli istnieje ciąg wierzchołków $i_1, i_2, \ldots, i_{l-1}, i_l = i_1$ taki, że $(i_j, i_{j+1})$ jest łukiem w $G$ dla

każdego $j = 1, 2, \ldots, l-1$. Niech $A$ będzie macierzą sąsiedztwa wierzchołków digrafu $G$. Stąd $G$ zawiera kontur wtedy i tylko wtedy, gdy istnieje ciąg liczb naturalnych $i_1, i_2, \ldots, i_{l-1}, i_l$ takich, że $a_{i_1 i_2} = a_{i_2 i_3} = \ldots = a_{i_{l-1} i_l} = 1$, $i_l = i_1$. Możemy więc mówić także o konturach w macierzach o elementach 0 lub 1.

TWIERDZENIE [47]. *Jeśli algorytm $\mathscr{A}$ określa, czy macierz $A$ zawiera kontur, to w najgorszym przypadku $\mathscr{A}$ wymaga zbadania co najmniej $n(n+1)/2$ elementów macierzy $A$.*

D o w ó d. Dowód tego twierdzenia jest typowy dla twierdzeń określających dolne oszacowania liczby operacji w algorytmach korzystających z macierzy. Niech $\mathscr{A}$ będzie algorytmem wykrywającym kontury w macierzach o elementach 0 i 1, a $A$ macierzą kwadratową $n \times n$, która nie zawiera konturu. W macierzy $A$ istnieje $\binom{n}{2} = n(n-1)/2$ nieuporządkowanych par różnych wskaźników $i, j$ $(1 \leqslant i, j \leqslant n,\ i \neq j)$, i załóżmy, że algorytm $\mathscr{A}$ bada mniej niż $n(n-1)/2$ spośród odpowiednich elementów $a_{ij}$ $(i \neq j)$ macierzy $A$. Wtedy istnieje para wskaźników $i, j$ taka, że $\mathscr{A}$ nie sprawdza ani $a_{ij}$ ani $a_{ji}$. Niech $A'$ będzie macierzą $A$ z wyjątkiem $a'_{ij} = a'_{ji} = 1$, czyli $A'$ zawiera kontur $(i, j, i)$, który nie zostanie wykryty przez algorytm $\mathscr{A}$. Doszliśmy więc do sprzeczności, a więc $\mathscr{A}$ musi sprawdzać co najmniej $n(n-1)/2$ elementów $a_{ij}$ $(i \neq j)$ macierzy $A$ oraz dodatkowo wszystkie elementy leżące na głównej przekątnej, aby stwierdzić, czy $A$ nie zawiera pętli. Reasumując, algorytm $\mathscr{A}$ wymaga sprawdzenia co najmniej $n(n+1)/2$ elementów macierzy $A$. ∎

Tak więc, jakikolwiek algorytm wykrywający kontury w digrafach reprezentowanych za pomocą macierzy sąsiedztwa wierzchołków ma złożoność co najmniej $\Omega(n^2)$.

Rozpatrzmy teraz problem wyznaczania długości najkrótszych dróg między wszystkimi parami wierzchołków w sieci. Niech $D = (d_{ij})$ oznacza macierz długości bezpośrednich połączeń w sieci, o której zakładamy jedynie, że nie zawiera konturów ujemnej długości. Istnieje wiele różnych algorytmów wyznaczania macierzy $D^*$, macierzy długości najkrótszych dróg (patrz prace przeglądowe Dreyfusa [25], Sysły [105] i Yena [121]), ale tutaj rozpatrzymy jedynie takie metody, które polegają na wielokrotnym wykonaniu tzw. $\Delta$-operacji. Niech $D'$ oznacza macierz otrzymaną z macierzy $D$ w wyniku wykonania $\Delta$-operacji $(i_0, j_0, k_0)$. Elementy macierzy $D'$ mają następującą postać:

$$d'_{ij} = d_{ij} \quad \text{dla} \quad (i, j) \neq (i_0, j_0),$$

$$d'_{i_0 j_0} = \min\{d_{i_0 j_0}, d_{i_0 k_0} + d_{k_0 j_0}\}.$$

Jak widać, pojedyncza $\Delta$-operacja polega na wykonaniu jednego dodawania i jednego porównania.

TWIERDZENIE (Nakamori [84]). *Każdy algorytm wyznaczający długości najkrótszych dróg między wszystkimi parami wierzchołków, który jest realizacją ciągu $\Delta$-operacji, wymaga co najmniej $n(n-1)(n-2)$ różnych $\Delta$-operacji.*

D o w ó d. Załóżmy nie wprost, że dla naszego problemu istnieje algorytm polegający na wykonaniu ciągu $\Delta$-operacji, w którym jednak nie występuje $\Delta$-operacja

$(i_0, j_0, k_0)$ i rozpatrzmy $n$-wierzchołkową sieć, w której długości bezpośrednich połączeń określone są następująco:

$$d_{ij} = \begin{cases} 0, & \text{jeśli } i = j \text{ lub } (i,j) = (i_0, k_0) \text{ lub } (i,j) = (k_0, j_0), \\ \infty, & \text{w przeciwnym przypadku.} \end{cases}$$

Po zastosowaniu tego algorytmu do macierzy $D$, otrzymujemy macierz długości najkrótszych dróg $D^*$ identyczną z macierzą $D$, a więc sprzeczność, bo $d^*_{i_0 j_0} = \infty$, a długość najkrótszej drogi z $i_0$ do $j_0$ wynosi 0. ■

Zauważmy, że twierdzenie to pozostaje prawdziwe także dla macierzy $D$ o nieujemnych elementach. Na podstawie powyższego twierdzenia wnioskujemy, że w przyjętej klasie metod algorytmy Warshalla i Floyda oraz Dantziga są optymalne z dokładnością do stałego współczynnika proporcjonalności.

Rozpatrywana klasa algorytmów nie zawiera algorytmu Dijkstry, składającego się także z $\Delta$-operacji, w której jednak kolejność poszczególnych operacji jest wyznaczana na bieżąco, w zależności od wartości danych i aktualnych wartości parametrów.

Hoffman i Winograd [46] podali teoretycznie najlepszą metodę wyznaczania długości najkrótszych dróg między wszystkimi parami wierzchołków. Ich metoda polega na wyborze specjalnej kolejności wykonywania $\Delta$-operacji dla sieci, która uprzednio zostaje zdekomponowana.

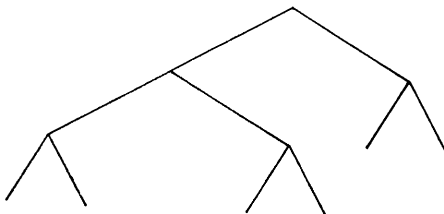
Omówimy teraz oszacowania złożoności obliczeniowej algorytmów, którym mogą być przyporządkowane drzewa decyzyjne. Załóżmy, że podstawowe operacje algorytmów są typu porównania. W ten sposób, rozpatrywane drzewa decyzyjne są drzewami binarnymi, w których wierzchołki pośrednie odpowiadają podstawowym operacjom, a końcowe — możliwym rezultatom. *Korzeń* drzewa jest początkiem algorytmu. Za *wysokość* drzewa przyjmujemy maksymalną liczbę wierzchołków pośrednich leżących na drodze z korzenia do wierzchołka końcowego, a więc wysokość odpowiada liczbie operacji podstawowych w najgorszym przypadku zachowania się algorytmu. *Zewnętrzna długość* drzewa jest sumą długości dróg z korzenia do wszystkich końcowych wierzchołków drzewa. Tak więc, jeżeli $N$ jest liczbą wszystkich wierzchołków końcowych w drzewie, to zewnętrzna długość tego drzewa jest $N$ razy większa od oczekiwanej złożoności algorytmu, przy założeniu, że wszystkie wyniki są jednakowo prawdopodobne. Aby więc wyznaczyć dolne oszacowania złożoności obliczeniowej problemu o $N$ możliwych wynikach, należy oszacować minimalną wysokość i minimalną długość zewnętrzną drzewa o $N$ wierzchołkach końcowych.

LEMAT. *Minimalna wysokość i minimalna długość zewnętrzna drzewa binarnego o $N$ wierzchołkach końcowych wynoszą odpowiednio $k$* i *$Nk+N-2^k$*, gdzie $k = \lceil \log N \rceil$ (1).

D o w ó d (szkic, szczegóły patrz Knuth [68]). Oszacowania podane w lemacie osiągnięte są dla drzewa binarnego, w którym odległości od korzenia kolejnych

(1) Wszystkie logarytmy w tej pracy są o podstawie 2.

wierzchołków pośrednich i końcowych (licząc poziomami i od lewej do prawej) są kolejnymi elementami ciągu 0, 1, 1, 2, 2, 2, 2, 3, 3, 3, 3, 3, 3, 3, 3,..., w którym $l$ występuje $2^l$ razy. W przypadku drzewa binarnego o $N$ wierzchołkach końcowych należy uwzględnić dokładnie $2N-1$ elementów tego ciągu, gdyż drzewo binarne o $N$ wierzchołkach końcowych zawiera $N-1$ wierzchołków pośrednich. Na rysunku 3 pokazane jest takie drzewo o 6-ciu wierzchołkach końcowych.



Rys. 3

Jego wysokość wynosi $3 = \lceil \log 6 \rceil$, a zewnętrzna długość $16 = 3N+N-2^3$. ∎

Pierwsza część lematu jest niejako sformalizowaniem intuicji mówiącej, że aby rozróżnić między sobą $k$ możliwych wyników należy wykonać co najmniej $\lceil \log k \rceil$ operacji binarnych.

Oszacowania podane w ostatnim lemacie użyte zostaną w następnym paragrafie w dowodach optymalności pewnych algorytmów porządkowania i przeszukiwania.

Istnieje jeszcze wiele innych metod wyznaczania oszacowań dolnych, na przykład metody, które korzystają z pewnych geometrycznych własności obszaru poszukiwań, który najczęściej jest wielościanem wypukłym.

## 6. Przykłady algorytmów optymalnych i „dobrych"

To co dotychczas powiedzieliśmy o złożoności obliczeniowej algorytmów sugeruje już pewien sposób dowodzenia ich optymalności.

W przypadku konkretnego problemu, ustalamy najpierw klasę metod, wśród których poszukiwać będziemy najlepszej metody rozwiązywania oraz ustalamy, które z działań uważać będziemy za podstawowe i tylko te uwzględniamy później w złożoności obliczeniowej i w jej dolnych oszacowaniach.

Na dalszych etapach postępowania, z jednej strony staramy się wyprowadzić coraz lepsze dolne oszacowania złożoności, a z drugiej coraz szybsze algorytmy obliczeń. W konsekwencji otrzymać możemy albo *optymalny algorytm*, czyli algorytm wykonujący dokładnie tyle samo działań, ile wynosi dolne oszacowanie złożoności, albo *algorytm optymalny z dokładnością do stałego współczynnika proporcjonalności*, tj. taki, którego złożoność jest tego samego rzędu, co dolne oszacowanie.

Algorytmy, które nie są optymalne w żadnym z przyjętych znaczeń, ale których złożoność obliczeniowa bardzo niewiele odbiega od wartości dolnego oszacowania,

lub które pomimo braku teoretycznych opracowań charakteryzują się dość dużą efektywnością praktyczną, nazywać będziemy algorytmami „dobrymi".

W dalszej części tego paragrafu przedstawimy szereg problemów kombinatoryki i teorii grafów wraz z uwagami o ich złożoności, oszacowaniach i najlepszych algorytmach.

## Algorytmy optymalne

**6.1. Wyznaczanie kolejnych elementów co do wielkości.** Problem *wyboru* (selection problem) polega na wyznaczeniu $k$-tego co do wielkości elementu spośród $n$ danych liczb. Problemu tego nie należy mylić z problemem *porządkowania* (ordering problem), który polega na wyborze i uporządkowaniu $k$ największych elementów.

Podstawowym działaniem w analizie złożoności algorytmów wyboru i porządkowania jest porównanie, wykonywane zwykle na dowolnych dwóch elementach spośród danych.

Dla $k = 1$ problem wyboru polega na wyznaczeniu największego elementu spośród $n$ elementów. Łatwo można wykazać, że w tym przypadku należy wykonać co najmniej $n-1$ porównań, a więc bezpośrednia metoda wyboru jest algorytmem optymalnym.

Naszkicujmy dowód tego faktu. Niech $X = \{x_i\}$ będzie ustalonym zbiorem $n$ elementów, a $A$ i $B$ podzbiorami $X$, o których zakładamy, że przez cały czas obliczeń $A$ zawiera maksymalny element zbioru $X$ i $B = X-A$. Jak łatwo zauważyć, obliczenia kończymy wtedy i tylko wtedy, gdy $|A| = 1$. Na początku obliczeń przyjmujemy $A = X$ i $B = \emptyset$. Przy założeniu, że możemy wykonywać tylko porównania mamy trzy możliwości: albo porównujemy ze sobą dwa elementy z tego samego zbioru, tj. z $A$ lub z $B$, albo elementy, z których jeden należy do $A$, a drugi do $B$. Widać, że tylko wykonanie porównania między dwoma elementami ze zbioru $A$ pozwala zmniejszyć aktualną liczbę elementów w zbiorze $A$ i to tylko o 1. Zatem po $n-1$ krokach otrzymujemy zbiór $A$ zawierający dokładnie jeden, szukany element zbioru $X$.

Okazuje się, że jeżeli wszystkie permutacje elementów zbioru $X$ są jednakowo prawdopodobne, to ten algorytm jest także optymalny w sensie średniej złożoności.

W podobny sposób można wykazać, że dowolny algorytm wyznaczający jednocześnie minimum i maksimum spośród $n$ liczb, a wykorzystujący tylko porównania, wymaga ich co najmniej $\lceil 3n/2 \rceil - 2$. Optymalny algorytm dla tego zadania składa się z dwóch podstawowych kroków, z których pierwszy polega na podziale zbioru $X$ na dwa rozłączne podzbiory $X_1$ i $X_2$ zawierające odpowiednio minimum i maksimum (podział taki może być wyznaczony za pomocą $\lfloor n/2 \rfloor$ porównań między elementami kolejnych par ze zbioru $X$), a drugi — na wyznaczeniu minimum w $X_1$ i maksimum w $X_2$ (patrz dalsze szczegóły w pracy Pohl [86]).

Przypomnijmy w tym miejscu, że dopuszczamy jedynie porównania wykonywane tylko między elementami spośród danych i tylko takie działania brane są pod

uwagę przy wyznaczaniu złożoności obliczeniowej algorytmów wyboru, porządkowania i sortowania (patrz 6.2).

Osłabiając to założenie, Reingold i Rabin [92], [93] pokazali, że wyznaczenie maksimum spośród $n$ liczb wymaga nadal wykonania co najmniej $n-1$ porównań, nawet jeżeli działania te mogą być wykonywane między analitycznymi, a w szczególności liniowymi funkcjami danych. Dopuszczając funkcje wykładnicze o parametrach i zmiennych całkowitych, maksimum może być znalezione za pomocą $\lceil \log n \rceil$ porównań wartości tych funkcji. W tym przypadku korzysta się z następującego faktu. Załóżmy dla wygody, że wszystkie dane liczby $x_1, x_2, \ldots, x_n$ są całkowite i $n = 2^k$. Można wykazać, że jeżeli

$$(n+1)^{x_1} + \ldots + (n+1)^{x_{n/2}} \geqslant (n+1)^{x_{n/2+1}} + \ldots + (n+1)^{x_n},$$

to maksimum znajduje się w podzbiorze $\{x_1, x_2, \ldots, x_{n/2}\}$, a w przeciwnym przypadku, w podzbiorze $\{x_{n/2+1}, \ldots, x_n\}$. Zatem, znalezienie maksymalnej liczby wymaga wykonania $\lceil \log n \rceil$ tego typu porównań.

Problem wyboru pojawia się w naturalny sposób w rozgrywkach turniejowych, gdy chcemy tak zorganizować rozgrywki, aby wybór zwycięzcy wymagał rozegrania jak najmniejszej liczby meczów. Stąd pochodzi nasze pierwotne założenie o typie operacji, jaka może być stosowana w algorytmach dla problemu wyboru, pozbawione jest bowiem sensu porównywanie na przykład odwrotności pierwiastków z bezwzględnych wartości różnic między osiągnięciami różnych zespołów.

Dla $k = 2$, Kislicyn [66] udowodnił, że $n-2+\lceil \log n \rceil$ jest konieczną i wystarczającą liczbą porównań do wyznaczenia drugiego co do wielkości elementu spośród $n$ danych.

Dla $k > 2$ znany jest jedynie algorytm o złożoności $O(n)$, a więc algorytm optymalny z dokładnością do stałego współczynnika proporcjonalności, patrz Blum i inni [2], [7].

**6.2. Sortowanie (porządkowanie).** Niech $S(n)$ oznacza minimalną liczbę porównań potrzebną do uporządkowania $n$ elementów. Udowodnimy teraz następujące nierówności

$$\lceil \log n! \rceil \leqslant S(n) \leqslant 1 + n\lceil \log n \rceil,$$

na podstawie których i przy użyciu wzoru Stirlinga możemy wywnioskować, że $S(n)$ zachowuje się w przybliżeniu jak $n\lceil \log n \rceil$. Dolne oszacowanie wynika z lematu o minimalnej wysokości drzewa binarnego o ustalonej liczbie wierzchołków końcowych, których w przypadku sortowania $n$ liczb jest $n!$ (patrz § 5). Natomiast górne oszacowanie wynika z oceny złożoności algorytmu Steinhausa [102]. Algorytm Steinhausa nazywany *porządkowaniem za pomocą binarnego umieszczania* (binary insertion sort), wyznacza szukane uporządkowanie przez umieszczenie kolejnych elementów w już uporządkowanym podciągu elementów dołączonych w poprzednich krokach tak, aby nowy ciąg był także uporządkowany. Rzeczywista liczba porównań w algorytmie Steinhausa wynosi $S_1(n) = 1 + n\lceil \log n \rceil + n - 2^{1+\lfloor \log n \rfloor}$, i już dla $n \geqslant 5$ zachodzi $\lceil \log n! \rceil < S_1(n)$. Ukazało się wiele prac poprawiających górne oszaco-

wanie liczby $S(n)$. Jedną z metod, o której udowodniono, że jest optymalna dla $n \leqslant 11$ oraz $n = 20, 21$, podali Ford i Johnson [36]. Dla $n = 12$ oszacowanie dolne wynosi 29, natomiast metoda Forda i Johnsona wymaga 30 porównań. Policzono następnie za pomocą maszyny cyfrowej (60 godzin!), że $S(12) = 30$, a więc metoda Forda i Johnsona jest także optymalna dla $n = 12$.

Załóżmy, że dany jest uporządkowany ciąg liczb $x_1 < x_2 < \ldots < x_n$ i oznaczmy przez $s(n)$ minimalną liczbę porównań potrzebną do umieszczenia liczby $y$ w odpowiednim miejscu tego ciągu. Jedną z pierwszych metod umieszczania podał także Steinhaus [102]. Polega ona na przeszukiwaniu danego ciągu za pomocą połowienia; stąd pochodzi jej popularna nazwa — metoda *binarnego przeszukiwania* (binary search). Jak można łatwo zauważyć, binarne przeszukiwania uporządkowanego ciągu o $n$ elementach wymąga $\lfloor \log n \rfloor + 1$ porównań. Z drugiej strony, na podstawie lematu o minimalnej wysokości drzewa binarnego (§ 5), $\lceil \log(n+1) \rceil$ jest dolnym oszacowaniem złożoności problemu umieszczenia, gdyż drzewo decyzyjne dla tego problemu ma $n+1$ wierzchołków końcowych odpowiadających tyluż różnym przedziałom, wyznaczonym przez $-\infty$, $x_i$ $(i = 1, 2, \ldots, n)$ oraz $+\infty$. Ponieważ $\lceil \log(n+1) \rceil = \lfloor \log n \rfloor + 1$, więc algorytm binarnego przeszukiwania jest algorytmem optymalnym. Udowodniono, że jest to także optymalny algorytm w średnim przypadku dla jednostajnego rozkładu danych.

Przedstawiliśmy wyżej szereg problemów, dla których istnieją optymalne algorytmy rozwiązywania. W swej naturze są to problemy zaliczane raczej do czystej kombinatoryki, istnieje jednak wiele algorytmów teorii grafów, w których znajdują zastosowanie. Z drugiej strony trudno jest znaleźć jakiś bardziej zaawansowany problem algorytmiczny teorii grafów, dla którego istnieje optymalny algorytm rozwiązywania. Najlepszą sytuację wśród istniejących algorytmów na grafach ilustrują podane niżej problemy, których metody rozwiązywania można nazwać optymalnymi z dokładnością do stałej proporcjonalności, nazwy tej używamy w przypadku, gdy dolne oszacowanie złożoności oraz złożoność konkretnego algorytmu są tego samego rzędu. Sytuacja taka może zaistnieć w dwóch przypadkach, gdy (a) znana jest dokładna wartość dolnego oszacowania i tylko rząd złożoności algorytmu lub (b) znane są jedynie rzędy dolnego oszacowania i złożoności algorytmu.

**6.3. Drzewa.** Grafy będące drzewami charakteryzują się ustaloną liczbą krawędzi, która jest o jeden mniejsza od liczby wierzchołków. Istnieje wiele różnych sposobów zapisu struktury drzew, patrz np. Knuth [68].

Wybór odpowiedniej reprezentacji w konkretnym przypadku powinien zależeć przede wszystkim od rodzaju rozpatrywanego zagadnienia, a w szczególności od rodzaju informacji o strukturze grafu wykorzystywanych w trakcie rozwiązywania i typu operacji wykonywanych na elementach grafu lub ich podzbiorach. Uwagi te odnoszą się nie tylko do drzew, ale także do innych struktur dyskretnych.

W ogólnym przypadku, najczęściej używaną reprezentacją drzew są listy, które na przykład w jednej z najprostszych postaci, przy ustalonej numeracji wierzchołków, zawierają $n-1$ par wierzchołków odpowiadających krawędziom. Uważa się

dość intuicyjnie, że rozwiązanie większości problemów teorii grafów wymaga dla ustalonego grafu wzięcia pod uwagę wszystkich jego krawędzi, a więc w tych przypadkach za dolne oszacowanie złożoności możemy przyjąć $\Omega(m)$, gdzie $m$ jest liczbą krawędzi grafu. W ten sposób, dla drzew otrzymujemy oszacowanie $\Omega(n)$, gdyż $m = n-1$, a więc starając się otrzymać optymalny lub tylko optymalny z dokładnością do stałego współczynnika proporcjonalności algorytm musimy wybrać taką reprezentację grafu, która w przypadku drzew wymaga $O(n)$ elementów pamięci.

Są to luźne uwagi, których sformalizowanie wymagałoby uściślenia wielu pojęć. (Pewne wyjaśnienia szczegółowe znaleźć można w § 4.) Poprzestaniemy jednak tutaj na tych wyjaśnieniach, a dalej w tym paragrafie dokonamy przeglądu ważniejszych problemów i algorytmów związanych z drzewami.

*6.3.1. Wyznaczanie centralnych wierzchołków w drzewie.* Na podstawie klasycznego twierdzenia Jordana wiadomo, że każde drzewo ma albo jeden wierzchołek centralny, albo dwa sąsiednie wierzchołki centralne. Metoda wyznaczenia tych wierzchołków polega na sukcesywnym odrzucaniu kolejnych wierzchołków wiszących. Aby jednak za każdym razem w poszukiwaniu aktualnych wierzchołków wiszących nie przeglądać całego zbioru wierzchołków, korzystamy z bardzo prostego spostrzeżenia, dzięki któremu w konsekwencji otrzymujemy algorytm o złożoności $O(n)$. Otóż, wierzchołki, które w następnym kroku stają się wierzchołkami wiszącymi, w kroku poprzednim są sąsiadami wierzchołków odrzucanych. Możliwe jest więc zapamiętanie kolejnych wierzchołków wiszących za pomocą listy–kolejki o długości $n$.

Algorytm wyznaczania wierzchołków centralnych w drzewie pojawia się w wielu innych algorytmach teorii grafów, które niekoniecznie odnoszą się do drzew, na przykład w algorytmie badania izomorfizmu i kodowania grafów zewnętrznie płaskich, patrz Sysło [110].

*6.3.2. Izomorfizm drzew.* Badanie, czy dwa $n$-wierzchołkowe drzewa są izomorficzne może być wykonane za pomocą algorytmu o złożności $O(n)$. Podstawowymi etapami tego algorytmu są wyznaczanie wierzchołków centralnych oraz cechowanie i porządkowanie pozostałych wierzchołków, poczynając od wierzchołków najbardziej odległych od wierzchołków centralnych. Pierwszy etap został już omówiony wyżej, natomiast liniowa realizacja drugiego etapu jest możliwa dzięki temu, że cechy nadawane wierzchołkom drzewa są liczbami lub ciągami liczb całkowitych z przedziału $(0, n)$. Można więc w tym przypadku zastosować koszykową metodę porządkowania (patrz szczegóły w Aho, Hopcroft i Ullman [2]).

*6.3.3. Drzewa częściowe i minimalne drzewa częściowe.* Drzewo częściowe grafu zwykłego, który jest zapamiętany w postaci list, może być wyznaczone za pomocą algorytmu o złożoności $O(\max(m, n))$. Szkieletem tego algorytmu jest przeszukiwanie grafu *metodą zgłębiania* (depth-first search), patrz Hopcroft i Tarjan [51]. Przez „przeszukiwanie grafu" rozumiemy tutaj jakąkolwiek metodę trawersowania wszystkich krawędzi grafów w taki sposób, że każdy wierzchołek jest odwiedzany dokładnie raz. Metoda zgłębiania została opisana po raz pierwszy przez Tarjana

[112] i od tego czasu została z powodzeniem zastosowana w konkretnych realizacjach wielu metod, które dzięki temu zyskały wiele na efektywności. Poniższy program napisany w ALGOL-podobnej symbolice jest bardzo zwięzłym opisem przeszukiwania grafu metodą zgłębiania.

```
begin
  procedure dfs (v);
    begin
      oznaczyć wierzchołek v jako osiągnięty;
      for wierzchołka w będącego sąsiadem v do
        if w nie został jeszcze osiągnięty
          then dfs(w)
    end;
  for jeszcze nie osiągniętego wierzchołka s do
    dfs(s)
end
```

Wyznaczenie drzewa częściowego w grafie reprezentowanym za pomocą macierzy wymaga $O(n^2)$ operacji.

Innym, bardzo popularnym zagadnieniem jest wyznaczanie minimalnego lub maksymalnego drzewa częściowego, tj. drzewa częściowego o sumarycznie najmniejszej lub odpowiednio, największej wadze w grafie z obciążonymi krawędziami. Dla grafów o $O(n^2)$ krawędziach metodą optymalną (z dokładnością do stałego współczynnika proporcjonalności) jest $O(n^2)$ algorytm Dijkstry, który jest szczególną wersją ogólnego algorytmu zwanego *algorytmem chciwości* (greedy algorithm), patrz Edmonds [28] i Lawler [75]. Podstawową ideą tej metody jest najlepszy wybór w każdym kroku z jednoczesnym sprawdzaniem dopuszczalności kolejno otrzymywanych rozwiązań częściowych. Podejście to znalazło bardzo szerokie zastosowanie w wielu innych metodach optymalizacji.

Szczegółowe rozważania w tym konkretnym przypadku (patrz [10], [64] i [120]) doprowadziły ostatnio do bardziej efektywnych realizacji strategii chciwości i tak, minimalne drzewo częściowe w (a) dowolnym grafie o $n$ wierzchołkach i $m$ krawędziach może być wyznaczone w czasie $O(m \log\log n)$, (b) w grafie gęstym (tj. takim, w którym liczba krawędzi $m$ jest ograniczona z dołu przez $\Omega(n^{1+\varepsilon})$, gdzie $\varepsilon$ jest stałą) w czasie $O(m)$, i (c) w grafie płaskim w czasie $O(n)$.

*6.3.4. Drzewa binarne.* Bardzo ważną klasę drzew stanowią drzewa binarne znajdujące szerokie zastosowania w analizie algorytmów. Istnieje wiele optymalnych algorytmów konstrukcji różnego typu drzew binarnych. Wyczerpujący opis tych zagadnień znaleźć można w książkach Knutha [68].

**6.4. Grafy płaskie.** Jak już wspomnieliśmy w § 2, sprawdzenie, czy dany graf jest płaski, może być wykonane w czasie $O(n)$, gdzie $n$ jest liczbą wierzchołków grafu. Podobnie, dla testowania zewnętrznej płaskości grafu istnieje liniowy algorytm, patrz Sysło i Iri [108].

W ostatnim okresie wiele uwagi poświęcono znalezieniu optymalnego algorytmu (z dokładnością do stałej proporcjonalności) badania, czy dwa grafy płaskie są izomorficzne, i ostatecznie taki algorytm został zaproponowany przez Hopcrofta i Wonga [53], ale jak piszą sami autorzy „na obecnym etapie realizacji, jest on nieefektywny ze względu na dużą stałą proporcjonalności", patrz także Fontet [35]. Natomiast dla testowania izomorfizmu grafów zewnętrznie płaskich podany został liniowy algorytm, który można zrealizować efektywnie, patrz Sysło [110].

**6.5. Problemy spójności i osiągalności.** Dzięki zastosowaniu metody zgłębiania do przeszukiwania grafu zwykłego i digrafu zapamiętywanych w postaci list sąsiadów otrzymano $O(\max(m, n))$ algorytmy wyznaczania składowych silnej spójności, składowych 2-spójności i składowych 3-spójności oraz mostów i wierzchołków rozspajających [50], [51], [112] i [113]. Odpowiednie algorytmy dla grafów zapamiętanych w postaci macierzy sąsiedztwa wierzchołków są o złożoności $O(n^2)$.

Bardzo ważnym zadaniem jest wyznaczanie macierzy osiągalności (*reachability matrix*) lub macierzy tranzytywnego domknięcia grafu. Dla grafów zwykłych macierz ta może być wyznaczona za pomocą algorytmu o złożoności $O(n^2)$, natomiast najlepszy ze znanych algorytmów dla digrafów, wykorzystujący metodę Strassena mnożenia macierzy wymaga $O(n^{2.81})$ operacji podstawowych i nic nie jest wiadomo o optymalności tego algorytmu. Nie są znane także żadne istotne oszacowania dolne. Patrz algorytmy i prace przeglądowe poświęcone temu problemowi [71], [107] i [121] oraz paragrafy 6.11 i 6.13. Problem ten ma algorytm, dla którego oczekiwana liczba operacji podstawowych wynosi $O(n^2)$, jednak w najgorszym przypadku jest to metoda rzędu $n^3$, patrz [85].

**6.6. Bezkonturowość.** W paragrafie 5 przedstawiliśmy dolne oszacowania złożoności obliczeniowej dla problemu wykrywania konturów w digrafie, które wynoszą odpowiednio $n(n+1)/2$ i $O(m)$ dla macierzowej i listowej reprezentacji digrafu. Z drugiej strony istnieje szereg metod, które mogą być tak zrealizowane, że otrzymane w konsekwencji algorytmy są optymalne z dokładnością do stałego współczynnika proporcjonalności. Dla przykładu wymieńmy tutaj algorytm Marimonta [80] i algorytm oparty na przeszukiwaniu digrafu metodą zgłębiania. Ciekawe byłoby określenie dokładnych wartości stałych proporcjonalności i zbadanie efektywności konkretnych realizacji obu tych algorytmów.

**6.7. Problemy dróg ekstremalnych.** Niech $D_d = \langle V(D), E(D); d\rangle$ oznacza *sieć*, czyli digraf z obciążonymi łukami, gdzie $d$ jest funkcją rzeczywistą opisaną na zbiorze łuków digrafu $D$, tj. $d\colon E \to \boldsymbol{R}$, przy czym, jak widać, dopuszczamy także ujemne wartości funkcji $d$. W zależności od interpretacji, $d(i,j)$ może oznaczać długość, przepustowość lub niezawodność połączenia $(i,j)$. Zatrzymajmy się dla przykładu jedynie na problemach związanych z tą pierwszą interpretacją. Problemy wyznaczania najkrótszych dróg w $D_d$ można w ogólności podzielić na trzy klasy: (i) wyznaczanie najkrótszej drogi między ustaloną parą wierzchołków, (ii) wyznaczanie najkrótszych dróg z ustalonego wierzchołka do wszystkich pozostałych wierzchołków oraz (iii) wyznaczanie najkrótszych dróg między każdą parą wierzchołków. Problem

(ii) jest często rozwiązywany przez iterację (i). W § 5 omówiliśmy dolne oszacowanie złożoności algorytmów rozwiązywania problemu (iii), które bazują na $\Delta$-operacji. W tym miejscu należy jeszcze wspomnieć o bardzo istotnym założeniu, które nakładamy na sieć $D_d$. Otóż, aby zagadnienia powyższe miały skończone rozwiązanie, sieć $D_d$ nie może zawierać zamkniętych dróg o ujemnych długościach. W rzeczywistości problemy (i)–(iii) rozwiązywane są w zmodyfikowanej postaci: jeżeli sieć $D_d$ nie zawiera ujemnego konturu, to wyznaczyć żądane drogi, a w przeciwnym razie sygnalizować niespełnienie założenia. W ten sposób każdy z algorytmów dla problemów (i)–(iii) i dla sieci o dowolnych wartościach funkcji $d$ może służyć do badania, czy sieć $D_d$ zawiera ujemny kontur. Ta modyfikacja ma istotny wpływ na złożoność algorytmów. Dla przykładu, problem (ii) ma $O(n^2)$ algorytm, jeżeli $d$ przyjmuje tylko wartości nieujemne (np. algorytm Dijkstry), natomiast dla funkcji $d$ o dowolnych wartościach, najlepszy algorytm jest o złożoności $O(n^3)$. Fakt ten jednak nie ma wpływu na rząd złożoności algorytmów dla problemu (iii), wynoszący $n^3$, chociaż dla sieci o nieujemnych wartościach funkcji $d$ istniejące algorytmy charakteryzują się mniejszym współczynnikiem proporcjonalności. Algorytm Dijkstry jest jedną z wielu metod rozwiązywania problemu (ii), które dla sieci pełnych są algorytmami optymalnymi z dokładnością do stałych współczynników proporcjonalności, pozostaje więc tylko określić ich dokładne wartości.

Sieci pełne pojawiły się w tej pracy już kilka razy. Czy w rzeczywistości występują na tyle często, aby poświęcać im specjalną uwagę? Argumentacją za może być tutaj pewna umowa przyjmowana w obliczeniach automatycznych. Otóż, struktura digrafu oraz wartości funkcji $d$ pamiętane są zwykle w jednej macierzy kwadratowej stopnia $n$, w której za długość nie istniejącego łuku przyjmuje się liczbę $M$ odpowiadającą $+\infty$. W zagadnieniach najkrótszych dróg wystarcza, jeżeli stała $M$ jest większa od $n \times \max d(i, j)$, gdzie max jest brane po istniejących łukach $(i, j)$. W ten sposób, każda sieć zostaje zastąpiona siecią pełną. Istnieją jednak metody rozwiązywania problemów (i)–(iii), które w obliczeniach uwzględniają jedynie występujące w digrafie łuki i zwykle oszacowania złożoności takich metod zawierają także $m$, liczbę łuków sieci, np. Johnson podał tego typu algorytm dla problemu (ii), którego złożoność wynosi $O(n^2+nm)$. Dalsze, bardziej szczegółowe, rozważania znaleźć można w pracach przeglądowych [25], [105] i [121].

## Algorytmy „dobre”

W poprzednich punktach wymieniliśmy szereg problemów, które bądź doczekały się algorytmów optymalnych, bądź optymalnych z dokładnością do stałego współczynnika proporcjonalności, który w wielu przypadkach może mieć istotny wpływ na efektywność, o czym była już mowa pod koniec § 2. Może się więc okazać, że dla interesujących nas wartości parametrów problemu, bardziej efektywnymi są metody, które nie są optymalne w żadnym sensie. O takich rezultatach mowa jest w wielu pracach poświęconych badaniom efektywności już konkretnych,

maszynowych realizacji algorytmów. Oczywiście, wyniki takich badań zależą dodatkowo od wielu czynników takich, jak język programowania i jego implementacja, użyta konfiguracja m.c., czy wreszcie „sztuka" programisty, które rzadko są uwzględniane w modelach obliczeń. Mimo tego, wyniki eksperymentów maszynowych są bardzo istotnym etapem w badaniach nad efektywnością algorytmów i ich konkretnych realizacji, łączącym w sobie rozważania i wyniki teoretyczne z analizą praktycznych możliwości użycia. Wynikami tego typu badań zainteresowani są przede wszystkim bezpośredni użytkownicy maszyn cyfrowych i algorytmów oczekujący w konkretnym przypadku zdecydowanej odpowiedzi na pytanie, którą z metod wybrać do obliczeń.

Dla uzupełnienia więc powyższej listy problemów, które obecnie mogą być rozwiązywane za pomocą optymalnych algorytmów, w dalszej części tego paragrafu wyszczególnimy szereg problemów, niektóre z nich ponownie, wraz z algorytmami rozwiązania, które nie mogą być uznane za optymalne w żadnym z przyjętych znaczeń. Wymienimy między innymi pewne algorytmy, o których przypuszcza się, że są optymalne.

**6.8. Porządkowanie.** Na podstawie nierówności z 6.2 możemy przyjąć, że każdy algorytm porządkowania o złożoności $\sim n\lfloor \log n \rfloor$ jest algorytmem dobrym.

**6.9. Drzewa i problemy dróg ekstremalnych.** Algorytmy Dijkstry dla problemu minimalnego drzewa częściowego i najkrótszych dróg w sieci, które są optymalne dla sieci pełnych, są także algorytmami dobrymi dla sieci rzadszych uzupełnionych zwykle do sieci pełnych, przez opisanie funkcji długości na wszystkich parach wierzchołków.

**6.10. Grafy płaskie.** W związku z tym, że liniowy algorytm badania izomorfizmu grafów płaskich [53] ma większe znaczenie teoretyczne niż praktyczne, w konkretnych obliczeniach stosować możemy algorytm o złożoności $O(n\log n)$, który może być zrealizowany efektywnie (patrz Hopcroft i Tarjan [49]).

**6.11. Problemy spójności i osiągalności.** Teoretycznie najefektowniejsze metody wyznaczania tranzytywnego domknięcia digrafu nie doczekały się jeszcze żadnej wzmianki o efektywności jakiejś konkretnej realizacji maszynowej. Być może teraz, gdy wiadomo jak efektywnie można zrealizować metodę Strassena, będzie można zbadać efektywność realizacji najlepszych algorytmów wyznaczania tranzytywnego domknięcia.

We wspomnianej już pracy [107] przedstawiono wyniki eksperymentów maszynowych z wieloma algorytmami wyznaczającymi tranzytywne domknięcie. Żadna z przetestowanych metod nie góruje wyraźnie nad pozostałymi, a ich efektywność zależy w dużym stopniu od gęstości digrafu. Tak jak można się było spodziewać, metoda Warshalla jest wyraźnie gorsza od pozostałych, może być jednak w bardzo prosty sposób zrealizowana za pomocą działań logicznych wykonywanych na całych słowach maszynowych, co w konsekwencji prowadzi do zmniejszenia czasu obliczeń oraz pozwala na rozwiązywanie zagadnień o dużych rozmiarach.

**6.12. Przepływy w sieciach.** Do problemów tej grupy zaliczamy zagadnienia powiązane w jakiś sposób z twierdzeniem Forda–Fulkersona o maksymalnym przepływie i minimalnym przekroju w sieci [75].

Przez długie lata podany przez Forda–Fulkersona algorytm oznaczania był jedyną metodą wyznaczania maksymalnego przepływu w sieci. Już sami autorzy tego algorytmu podali przykład, dla którego zła kolejność podstawowych operacji algorytmu może nie prowadzić do wyznaczenia optymalnego rozwiązania nawet po nieskończonej liczbie kroków. Istnieje także prosty przykład sieci, dla której wyznaczenie optymalnego przepływu wymagać może od algorytmu Forda–Fulkersona wykonania liczby operacji proporcjonalnej do wartości rozwiązania (tj. wartości maksymalnego przepływu). Obecnie mamy do dyspozycji dwa dobre algorytmy, których efektywność nie zależy od konkretnych wartości parametrów, różnych od $n$ i $m$. Jeden algorytm podany został przez Dinica [24] i ma złożoność $O(n^2m)$, a drugi zaproponowali Edmonds i Karp i ma złożoność $O(nm^2)$ [29].

**6.13. Mnożenie macierzy.** Złożoność obliczeniowa mnożenia dwóch macierzy, jednego z klasycznych problemów analizy numerycznej, ma także istotny wpływ na efektywność algorytmów kombinatorycznych. Wspomnieliśmy dotychczas o teoretycznie najefektywniejszej metodzie wyznaczania tranzytywnego domknięcia digrafu, której rząd złożoności jest taki sam, jak metody Strassena mnożenia dwóch macierzy. Dalej pokażemy, jaki jest wpływ tego faktu na złożoność obliczeniową mnożenia dwóch macierzy boolowskich.

Szczegółowy opis metody Strassena znaleźć można w pracy Jankowskiego i Woźniakowskiego [57], a tutaj przytoczymy jedynie najistotniejsze uwagi o jednej z jej efektywnych realizacji.

Jest oczywiste, że złożoność mnożenia dwóch macierzy kwadratowych stopnia $n$ wynosi co najmniej $\Omega(n^2)$, gdyż żaden z elementów obu macierzy nie może być pominięty w trakcie obliczeń. Dotychczas najlepsze dolne oszacowanie złożoności tego problemu podał Kirkpatrick i wynosi ono $2n^2-n$ mnożeń lub dzieleń i $2n^2-3n+1$ dodawań lub odejmowań. Z drugiej strony, mamy szereg konkretnych algorytmów, a wśród nich klasyczny algorytm wymagający $n^3$ mnożeń i $n^3-n^2$ dodawań, algorytm Winograda wymagający $n^3/2+n^2$ mnożeń i $2n^3$ dodawań, istotnie szybszy od klasycznego, jeśli mnożenie jest wyraźnie kosztowniejsze od dodawania oraz metodę Strassena, której złożoność jest $O(n^{\log 7})$. Istnieje więc nadal duża rozpiętość między najlepszym dolnym oszacowaniem złożoności a złożonością najlepszego algorytmu. Optymalność metody Strassena udowodniono tylko dla niewielu przypadków szczególnych, np. dla $n = 2, 3$, patrz Hopcroft i Kerr [48]. We wspomnianej już pracy [57] omówiono przypuszczenia i problemy związane z uogólnieniem i polepszeniem metody Strassena.

Przez ponad pięć lat metoda Strassena uważana była jedynie za rezultat teoretyczny. Zwykle, bardzo niechętnie i z dużym pesymizmem wspominano o możliwościach jej efektywnej realizacji na maszynie cyfrowej. Podawano bardzo grube oszacowania wymiaru macierzy, od którego ta metoda jest w praktyce lepsza od

klasycznej, jednocześnie zastrzegając się, że dla dokładnego porównania tych metod nie wystarcza brać pod uwagę jedynie działania arytmetyczne. Praca Cohena i Rotha [13] jest jedną z pierwszych prób opisania maszynowej realizacji metody Strassena. Okazało się, że już dla $n = 40$ metoda Strassena może szybciej mnożyć dwie macierze niż metoda klasyczna (obie metody zaprogramowano w ALGOL-PDP-10). Ale jak sami autorzy zastrzegli się, jest to raczej drugoplanowy wynik ich pracy. Za najważniejsze rezultaty pracy [13] należy uznać:

(a) przeprowadzenie systematycznej (mikro)analizy algorytmu Strassena z podaniem dokładnych wyrażeń na liczby wszystkich istotnych operacji. Uwzględniono mnożenie, dodawanie, odejmowanie, użycie nowych zmiennych dla przechowywania wielkości pomocniczych oraz operacje pobierania i odsyłania zawartości komórki maszyny cyfrowej. W ten sposób, korzystając z podanych wyrażeń można oszacować efektywność zaproponowanej realizacji metody Strassena dla dowolnej, innej m.c.;

(b) zamianę oryginalnego, rekurencyjnego algorytmu na iteracyjny, który pozwolił zmniejszyć do minimum liczbę parametrów i czas potrzebne dla samej organizacji obliczeń. W czasie porównania obu tych realizacji, rekurencyjnej i iteracyjnej, okazało się, że dla $n = 64$ ta pierwsza wymaga 33 minut, zaś ta druga tylko 2 minut;

(c) użycie specjalnego liniowego sposobu pamiętania elementów macierzy, który pozwala na szybszy dostęp do nich, według schematu wymaganego przez sam algorytm. Zaproponowany sposób jest bardzo użyteczny przy podziale macierzy na części i wykonywaniu operacji na całych wektorach.

|  |  |  |  |
|---|---|---|---|
| 0 | 1 | 4 | 5 |
| 2 | 3 | 6 | 7 |
| 8 | 9 | 12 | 13 |
| 10 | 11 | 14 | 15 |

Rys. 4

Na rysunku 4 pokazana jest zaproponowana numeracja elementów macierzy o wymiarach $4 \times 4$.

Zakończymy ten punkt uwagami o mnożeniu dwóch macierzy boolowskich, które pojawia się w wielu algorytmach na grafach reprezentowanych za pomocą macierzy sąsiedztwa wierzchołków lub sąsiedztwa łuków. Przy okazji tego problemu chcielibyśmy zwrócić uwagę na bardzo ważne w teorii algorytmów pojęcie transformacji (redukcji). Transformacja problemu $P_1$ w problem $P_2$ pozwala na przeniesienie algorytmu i dolnych oszacowań z $P_2$ na $P_1$.

Udowodnić można, że mnożenie dwóch macierzy boolowskich może być zredukowane do wyznaczania tranzytywnego domknięcia macierzy, i na odwrót. Załóżmy najpierw, że dysponujemy algorytmem mnożenia macierzy boolowskich, którego złożoność wynosi $M(n)$. Wtedy, tranzytywne domknięcie macierzy boolowskiej $A$ wyznaczamy przez uprzedni jej rozkład na cztery podmacierze o wymiarach $\frac{1}{2}n\times\frac{1}{2}n$

$$A=\begin{bmatrix} B & C \\ D & E \end{bmatrix},$$

a wtedy

$$A^*=\begin{bmatrix} F & FCE^* \\ E^*DF & E^*+E^*DFCE^* \end{bmatrix},$$

gdzie * oznacza operację tranzytywnego domknięcia, a $F=(B+CE^*D)^*$. W ten sposób, aby wyznaczyć tranzytywne domknięcie macierzy boolowskiej stopnia $n$ musimy obliczyć tranzytywne domknięcie dwóch macierzy o wymiarach $\frac{1}{2}n\times\frac{1}{2}n$ oraz wykonać sześć mnożeń i dwa dodawania macierzy o tych samych wymiarach. Niech $T(n)$ oznacza złożoność algorytmu wyznaczania tranzytywnego domknięcia macierzy boolowskiej stopnia $n$. Mamy więc następującą nierówność rekurencyjną

$$T(2^k)\leqslant 2T(2^{k-1})+6M(2^{k-1})+2\cdot 2^{2k-2},$$

z której otrzymujemy nierówność $T(n)\leqslant cM(n)$, gdzie $c$ jest pewną stałą. Podobnie, mnożenie dwóch macierzy boolowskich może być zredukowane do wyznaczania tranzytywnego domknięcia. Ostatecznie otrzymujemy następujące nierówności $M(n)\leqslant cT(n)\leqslant c^2M(n)$ dla pewnej stałej $c$.

Jest to przykład *liniowej transformacji* (*redukcji*) jednego problemu w drugi. Problem $P_1$ jest liniowo transformowalny do problemu $P_2$, jeśli wykorzystując algorytm rozwiązywania problemu $P_2$ można skonstruować algorytm rozwiązywania problemu $P_1$ taki, że czas rozwiązywania $P_1$ jest liniową funkcją czasu rozwiązywania problemu $P_2$.

W ten sposób, w klasie macierzy boolowskich stopnia $n$ iloczyn dwóch macierzy oraz tranzytywne domknięcie mogą być wyznaczane w czasie proporcjonalnym do $n^{2.81}$. Są to oczywiście złożoności najgorszego przypadku. Wspomnieliśmy już o metodzie mnożenia dwóch macierzy boolowskich, której oczekiwana liczba operacji jest rzędu $n^2$, metoda ta jednak wymaga $O(n^3)$ operacji w najgorszym przypadku [85].

Dalsze uwagi o złożoności obliczeniowej mnożenia i wyznaczania tranzytywnego domknięcia macierzy boolowskich znaleźć można w opracowaniach [2] i [33].

Przedstawiona wyżej transformacja liniowa jest szczególnym przypadkiem tzw. transmisji wielomianowej, którą omówimy dokładnie w § 9.

## 7. Problemy o złożoności wielomianowej

W poprzednim paragrafie przedstawiliśmy szereg problemów, których algorytmy rozwiązywania są o *złożoności wielomianowej*, w skrócie takie algorytmy nazywamy *algorytmami wielomianowymi*. Innymi słowy, czas obliczeń za pomocą algorytmu wielomianowego ograniczony jest przez wielomian zmiennej rozmiaru zagadnienia (tj. długości danych wejściowych). Istnienie algorytmu wielomianowego jest obecnie powszechnym kryterium klasyfikacji obliczeniowych problemów kombinatoryki i teorii grafów. Problemy, które mogą być rozwiązywane za pomocą algorytmów wielomianowych nazywamy czasem w skrócie *problemami wielomianowymi*. Przedstawione wyżej problemy i ich algorytmy ilustrują także podstawową tendencję w badaniach nad metodami rozwiązywania problemów wielomianowych. Z jednej strony starania idą w kierunku znalezienia jak najlepszych dolnych oszacowań złożoności, a z drugiej opracowywane są coraz to lepsze algorytmy, tj. wymagające krótszego czasu obliczeń i/lub mniejszego obszaru pamięci maszyny cyfrowej. Te starania mają na celu doprowadzenie w konsekwencji do znalezienia optymalnych algorytmów.

Odrębną klasę zagadnień stanowią problemy, dla których dotychczas nie udało się znaleźć żadnego wielomianowego algorytmu rozwiązywania. Do klasy tej należą między innymi takie problemy, jak problem komiwojażera, badanie izomorfizmu grafów, problemy pokrycia i podziału, kolorowanie grafu itd. W § 9 pokażemy w sposób bardziej sformalizowany, że w większości przypadków problemy z tej klasy są równoważne w tym sensie, że albo dla każdego z tych problemów istnieje wielomianowy algorytm rozwiązywania, albo algorytm taki nie istnieje dla żadnego z nich.

Przed przejściem do omówienia problemów niewielomianowych wspomnijmy jeszcze o problemach, dla których nie mogą istnieć algorytmy wielomianowe.

## 8. Problemy, dla których niemożliwe jest istnienie algorytmów o złożoności wielomianowej

Przypomnijmy, że długość układu danych wejściowych jest określana za pomocą liczby wierzchołków i liczby łuków grafu, odpowiednio $n$ i $m$. Istnieje w teorii grafów wiele problemów obliczeniowych, dla których, ze względu na długość wyjścia (tj. ze względu na ilość wyników), nie są możliwe algorytmy o wielomianowej złożoności. Wszystkie tego typu problemy polegają na generowaniu całej rodziny podgrafów o ustalonych własnościach, których liczba dla pewnych klas grafów może rosnąć wykładniczo lub szybciej ze wzrostem długości danych wejściowych.

Dla przykładu, rozpatrzmy generowanie wszystkich klik grafu i zauważmy, że $k$-dzielny graf o $n = 3k$ wierzchołkach i trójelementowych klasach wierzchołków ma dokładnie $3^k$ klik.

Innymi przykładami tego typu problemów są: generowanie wszystkich drzew częściowych grafu i generowanie wszystkich cykli prostych grafu. Pierwszy z tych problemów znajduje zastosowanie przy wyznaczaniu parametrów dla liniowych sieci elektrycznych, a drugi w analizie grafu programu obliczeń.

Sam charakter zagadnień zmusza nas do przyjęcia w tym przypadku innej miary złożoności obliczeniowej, w której powinniśmy uwzględnić także wpływ ilości wyników. Niech więc dodatkowo $N$ oznacza liczbę podgrafów, które mają być wygenerowane. Najefektywniejsze algorytmy generowania wszystkich drzew częściowych i cykli prostych grafu wymagają $O(n+m+mN)$ operacji i $O(n+m)$ pamięci [91].

## 9. Problemy NP-zupełne

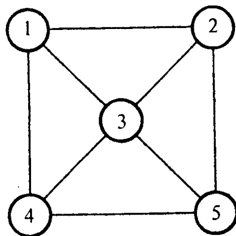
W rozdziale tym omówimy szczegółowo równoważność zachodzącą między problemami kombinatorycznymi, które na obecnym etapie rozwoju algorytmów uważane są za problemy *trudne* (intractable). Całkowicie sformalizowane rozważania tego rozdziału powinny być przeprowadzone w terminach maszyn Turinga (przyjętych tutaj za model obliczeń), tak jak oryginalnie przedstawione zostały przez Cooka [16] i Karpa [59], a później w książce Aho, Hopcroft i Ullman [2]. Aby jednak niepotrzebnie nie zaciemnić najważniejszych rezultatów, postaramy się w wielu miejscach zrezygnować ze zbytniego formalizmu, a zainteresowanych wszystkimi szczegółami odsyłamy do zacytowanych opracowań.

W związku z przyjętym modelem obliczeń ograniczymy swoją uwagę tylko do *problemów decyzyjnych*, tj. do takich, które wymagają jedynie odpowiedzi *tak* lub *nie*. Dla przykładu, *problem kliki* sformułowany jako problem decyzyjny ma następującą postać: dany jest graf zwykły $G$ i liczba naturalna $k$, i pytamy, czy $G$ zawiera *k-klikę* (tj. podgraf pełny o $k$ wierzchołkach)? Często jednak interesuje nas wyznaczenie największej kliki w grafie $G$, a więc wyznaczenie podgrafu pełnego grafu $G$ o największej liczbie wierzchołków. W wielu przypadkach, problemy optymalizacyjne mają wielomianowe algorytmy pozwalające przetransformować je w odpowiednie problemy decyzyjne. Tak jest na przykład w przypadku każdego problemu NP-zupełnego, o których będzie mowa w dalszej części tego paragrafu. W tym konkretnym przypadku problemu kliki możemy postąpić następująco. Niech $n$ będzie liczbą wierzchołków grafu $G$. Dla każdego $k$ od 1 do $n$ sprawdzamy najpierw, czy graf $G$ zawiera $k$-klikę. W ten sposób rozwiązanie $n$ decyzyjnych problemów kliki pozwala znaleźć liczbę $m$, rozmiar największy kliki w rozpatrywanym grafie $G$. Aby znaleźć wierzchołki tworzące $m$-klikę usuwamy kolejno z $G$ wierzchołek po wierzchołku, aż do momentu, gdy usunięcie wierzchołka $v$ rozbija wszystkie pozostałe $m$-kliki. Następnie rozpatrujemy podgraf $G'$ złożony z sąsiadów wierzchołka $v$ i powtarzamy ten proces rekurencyjnie w celu znalezienia $(m-1)$-kliki $C$ w $G'$. Największa klika w grafie $G$ utworzona jest przez $C$

iv. Łatwo można sprawdzić, że złożoność obliczeniowa powyższej metody wyznaczania największej kliki w grafie jest wielomianową funkcją $n$ i złożoności algorytmu rozwiązywania decyzyjnego problemu kliki.

Drugim, istotnym założeniem stawianym przez model obliczeń jest, aby dane dla rozpatrywanych problemów były w jakiś jednolity sposób zakodowane, na przykład w postaci ciągów zer i jedynek. Taki sposób przedstawienia danych nie jest niczym nowym, występuje on implicite w każdej maszynie cyfrowej. W ogólności, *kodowanie* w tym przypadku polega na wyborze *alfabetu*, tj. zbioru symboli podstawowych (liter, cyfr i innych znaków) i odwzorowania rodziny wszystkich możliwych układów danych w słowa nad wybranym alfabetem, czyli w ciągi elementów tego alfabetu. Chociaż dalej nie będziemy zatrzymywać się nad wszystkimi szczegółami kodowania danych, to należy zdawać sobie sprawę z faktu, że wybór metody kodowania może mieć istotny wpływ na złożoność obliczeniową.

Przyjmujemy następujące założenia o reprezentacji wielkości występujących w problemach rozpatrywanych w dalszej części tego rozdziału: (i) wszystkie liczby całkowite przedstawione są w systemie dziesiętnym, (ii) $n$-wierzchołkowy graf $G$ ma wierzchołki ponumerowane kolejnymi liczbami naturalnymi $1, 2, \ldots, n$ zapisanymi w systemie dziesiętnym, a krawędzie reprezentowane są za pomocą par $(i_1, i_2)$, gdzie $i_1$, $i_2$ są numerami końcowych wierzchołków krawędzi, zapisanymi także w systemie dziesiętnym, (iii) wyrażenie boolowskie o $n$ zmiennych prostych przedstawione jest w postaci łańcucha, w którym $\cap$ reprezentuje koniunkcję, $\cup$ reprezentuje alternatywę, $\neg$ reprezentuje negację, a liczby $1, 2, \ldots, n$ reprezentują zmienne. W razie konieczności możemy używać nawiasów, znak $\cap$ może być pominięty, a negacja $\neg(\alpha)$ może być zastąpiona przez $\bar{\alpha}$. Przy tych założeniach, graf $G$, przedstawiony na rysunku 5, który badamy, czy zawiera 4-klikę, może mieć reprezentację 5 4 (1,2) (1,3) (1,4) (2,3) (2,5) (3,4) (3,5) (4,5), gdzie 5 jest liczbą wierzchołków grafu $G$, 4 oznacza wymiar kliki, o której istnienie pytamy, a ciąg par liczb całkowitych ujętych w nawiasy jest ciągiem krawędzi grafu $G$.




Rys. 5

Możemy stosować jednocześnie różne sposoby kodowania danych, np. liczby całkowite mogą być zapisane w systemie dwójkowym, musimy jednak pamiętać o tym, aby dwa różne kodowania miały wielomianowy algorytm transformacji danych z jednej reprezentacji w drugą. Zatem nie musimy się martwić o wszystkie

szczegóły związane z wyborem alfabetu, w którym reprezentowane są rozpatrywane problemy.

Możemy teraz przejść do bardziej formalnego zdefiniowania wspomnianych już dwóch klas problemów.

**9.1. Problemy klasy *P*.** Niech $\{0, 1\}^*$ oznacza zbiór wszystkich skończonych łańcuchów złożonych z zer i jedynek. *Językiem* nazywamy dowolny podzbiór zbioru $\{0, 1\}^*$. Dowolnemu problemowi decyzyjnemu możemy przyporządkować język $L$, złożony z łańcuchów, które są kodami tych danych wejściowych, dla których problem ma rozwiązanie „tak". Oznaczmy przez $\boldsymbol{P}$ klasę wszystkich problemów decyzyjnych, które mogą być rozwiązane w czasie wielomianowym za pomocą deterministycznej maszyny Turinga. Innymi słowy, zbiór łańcuchów $L$ jest w $\boldsymbol{P}$, jeżeli istnieje algorytm $A$ taki, że

(a) odpowiada „tak" dla łańcuchów z $L$ i „nie" dla łańcuchów z $\{0, 1\}^* - L$;

(b) czas działania algorytmu $A$ jest wielomianem długości wejścia, tj. istnieje wielomian $p(\cdot)$ taki, że dla każdego $x$ liczba kroków algorytmu $A$ jest ograniczona przez $p(|x|)$, gdzie $|x|$ oznacza długość wejścia $x$. Możemy mówić, że algorytm *A rozpoznaje* (recognize) zbiór $L$ lub *akceptuje* (accept) łańcuchy z $L$.

Powyższa definicja klasy problemów zawiera dwa pojęcia nadal niesprecyzowane, a mianowicie algorytm oraz jego krok. Kompletną definicję klasy $\boldsymbol{P}$ można podać definiując dodatkowo te dwa pojęcia, na przykład w terminach maszyn Turinga, tak jak to zrobiono w [2] i [59], tutaj jednak pozostaniemy przy powyższym sformułowaniu bez uzupełnień. Pozwala nam na to pewna dowolność w wyborze formalnego modelu maszyny cyfrowej. Dla naszych celów wystarczy przyjąć za klasę $\boldsymbol{P}$ te problemy, które mogą być rozwiązywane w czasie wielomianowym za pomocą maszyny cyfrowej o nieograniczonej pamięci i akceptującej łańcuchy danych o dowolnej długości. Zatem wszystkie problemy omówione przy okazji prezentacji algorytmów optymalnych i dobrych zaliczamy do klasy $\boldsymbol{P}$.

**9.2. Transformacje wielomianowe i problem spełnialności.** Rozwiązanie jednego problemu przez sprowadzenie do rozwiązania innego problemu stało się już powszechną techniką stosowaną zarówno w czystej jak i stosowanej matematyce. Redukcja taka wymaga określenia transformacji danych dla pierwszego w równoważne dane dla drugiego problemu. Już na początku lat 60-tych Dantzig pokazał, że szereg problemów kombinatorycznych można sprowadzić do zagadnień programowania liniowego w liczbach całkowitych, które z kolei mogą być przetransformowane w problem plecakowy.

W tej pracy interesować nas będą jedynie transformacje dające się zrealizować w czasie wielomianowym.

Niech dane będą dwa problemy decyzyjne $L, M \subseteq \{0, 1\}^*$. Mówimy wtedy, że problem $L$ jest *transformowalny* w $M$ (co oznaczać będziemy czasem $L \propto M$), jeżeli istnieje funkcja transformująca $f$ spełniająca następujące warunki:

(i) $f$: $\{0, 1\}^* \to \{0, 1\}^*$, tj. $f$ odwzorowuje łańcuchy w łańcuchy,

(ii) istnieje wielomianowy algorytm obliczania $f$, i

(iii) transformacja $f$ zachowuje problem, tj., $f(x) \in M$ wtedy i tylko wtedy, gdy $x \in L$.

Zatem, aby sprawdzić, czy $x \in L$ obliczamy najpierw $f(x)$, a następnie sprawdzamy, czy $f(x) \in M$. Zauważmy, że $\propto$ jest relacją przechodnią i zachowującą wielomianowość algorytmów obliczeń, tzn., jeżeli $L \propto M$ oraz $M$ ma wielomianowy algorytm akceptujący, to także $L$ ma algorytm wielomianowy. Przedstawimy dalej szereg problemów wielomianowo transformowalnych jeden w drugi, które stanowią w tym sensie klasę problemów równoważnych. Oznacza to, że albo każdy z nich ma, albo żaden z nich nie ma algorytmu wielomianowego. Podstawowym problemem tej klasy jest problem *spełnialności* (satisfiability) pojawiający się w elementarnej logice. Niech $x_1, x_2, \ldots, x_m$ będą prostymi zmiennymi boolowskimi przyjmującymi wartości *true* (1) lub *false* (0), a przez $X_i$ oznaczmy *literę*, czyli $x_i$ lub $\bar{x}_i$. *Normalną postacią koniunkcyjną* wyrażenia boolowskiego (w skrócie CNF) nazywamy koniunkcję alternatyw, które mogą być złożone jedynie z liter $X_i$. Wyrażenie boolowskie jest *spełnialne* (satisfiable), jeżeli istnieje takie przyporządkowanie wartości zmiennym prostym $x_1, \ldots, x_m$, dla którego wyrażenie boolowskie przyjmuje wartość 1. Problem *spełnialności* polega na określeniu, czy dane wyrażenie boolowskie jest spełnialne. Problem *CNF spełnialności* jest problemem spełnialności wyrażeń boolowskich w postaci CNF. Dla przykładu, wyrażenie $(x_1 \cup \bar{x}_2 \cup x_3) \cap (\bar{x}_1 \cup x_2) \cap (x_2 \cup \bar{x}_3)$ jest spełnialne, gdyż przyjmuje wartość 1 dla $x_1 = x_2 = 1$ oraz $x_3 = 0$.

Przedstawimy teraz trzy wielomianowe transformacje problemu CNF spełnialności odgrywające podstawową rolę w dalszych rozważaniach tego paragrafu.

*Problem CNF spełnialności $\propto$ problem k-kliki.* Załóżmy, że $C = C_1 \cap C_2 \cap \ldots \cap C_q$ jest wyrażeniem boolowskim w normalnej postaci koniunkcyjnej, gdzie $C_i$ jest alternatywą postaci $(X_{i1} \cup X_{i2} \cup \ldots X_{ip_i})$, a $X_{ij}$ oznacza zmienną prostą lub jej negację. Dla wyrażenia $C$ budujemy graf zwykły $G = (V, E)$, w którym wierzchołkami są pary $[i, j]$ $(1 \leqslant i \leqslant q,\ 1 \leqslant j \leqslant p_i)$ odpowiadające poszczególnym literom, tj. $[i, j]$ odpowiada $j$-tej literze w $i$-tej alternatywie. Zbiór krawędzi grafu $G$ tworzą pary $([i, j], [k, l])$, które spełniają $i \neq k$ oraz $X_{ij} \neq \bar{X}_{kl}$, tzn. sąsiednimi są te pary, które odpowiadają różnym alternatywom i możliwe jest takie nadanie wartości zmiennym prostym stojącym pod literami $X_{ij}$ i $X_{kl}$, że $C_i$ i $C_k$ przyjmują jednocześnie wartość 1.

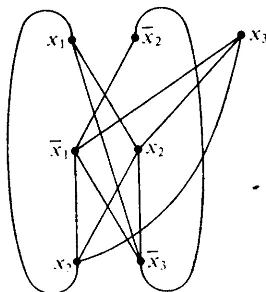
Rysunek 6 przedstawia graf $G$ zbudowany dla wyrażenia $(x_1 \cup \bar{x}_2 \cup x_3) \cap$ $\cap (\bar{x}_1 \cup x_2) \cap (x_2 \cup \bar{x}_3)$.

Pokażemy teraz, że graf $G$ zawiera $q$-klikę wtedy i tylko wtedy, gdy $C$ jest wyrażeniem spełnialnym.

Niech $C$ będzie wyrażeniem spełnialnym, tzn. istnieje takie przyporządkowanie wartości zmiennym prostym $x_1, x_2, \ldots, x_m$, że $C = 1$, czyli $C_i = 1$ $(i = 1, 2, \ldots$ $\ldots, q)$, a więc każda alternatywa $C_i$ zawiera co najmniej jedną literę $X_{ir_i}$ o wartości 1. Można łatwo sprawdzić, że zbiór wierzchołków $\{[i, r_i]\colon 1 \leqslant i \leqslant q\}$ tworzy $q$-klikę grafu $G$.

Podobnie łatwo można pokazać, że każdej $q$-klice grafu $G$ odpowiada zbiór wartości zmiennych prostych, dla których w rezultacie $C = 1$.

Wyrażenie, któremu odpowiada graf przedstawiony na rysunku 6, przyjmuje wartość 1 dla $x_1 = 0$ oraz $x_2 = x_3 = 1$. Takiemu przyporządkowaniu wartości zmiennym prostym odpowiada 3-klika ([1, 3], [2, 1], [3, 1]) grafu $G$. Graf zawiera jeszcze jedną klikę, a mianowicie ([1, 1], [2, 2], [3, 2]), której odpowiadają wartości zmiennych $x_1 = x_2 = 1$ oraz $x_3 = 0$, dla których także rozpatrywane wyrażenie przyjmuje wartość 1.



Rys. 6

Zauważmy, że liczba wierzchołków grafu $G$ jest ograniczona przez długość wyrażenia $C$, a liczba krawędzi jest co najwyżej kwadratem tej długości. Zatem utworzenie i zakodowanie danych dla powyższego problemu $q$-kliki może być wykonane w czasie ograniczonym przez wielomian długości danych dla problemu CNF spełnialności. Jest to więc wielomianowa transformacja. ■

Niech $k$-CNF oznacza problem spełnialności dla wyrażeń boolowskich sprowadzonych do normalnej postaci koniunkcyjnej, w której alternatywy ograniczone są do co najwyżej $k$ liter. Kolejna redukcja jest postaci:

*Problem CNF spełnialności* $\propto$ *Problem 3-CNF spełnialności.* Każdą alternatywę $C_i = X_{i1} \cup X_{i2} \cup \ldots \cup X_{ip_i}$ $(p_i \geqslant 4)$ zastępujemy koniunkcją

$$C'_i = (X_{i1} \cup X_{i2} \cup y_1) \cap (X_{i3} \cup \bar{y}_1 \cup y_2) \cap (X_{i4} \cup \bar{y}_2 \cup y_3) \cap \ldots \cap$$
$$\cap (X_{ip_i-2} \cup \bar{y}_{p_i-4} \cup y_{p_i-3}) \cap (X_{ip_i-1} \cup X_{p_i} \cup \bar{y}_{p_i-3}),$$

gdzie $y_1, \ldots, y_{p_i-3}$ są nowymi zmiennymi boolowskimi. Można pokazać, że tak utworzone wyrażenie $C'_i$ dla pewnego układu wartości nowych zmiennych ma wartość 1 wtedy i tylko wtedy, gdy któraś z liter $X_{ij}$ $(j = 1, 2, \ldots, p_i)$ ma wartość 1. Zatem wyrażenie $C$ jest spełnialne wtedy i tylko wtedy, gdy spełnialne jest wyrażenie $C' = C'_1 \cap C'_2 \cap \ldots \cap C'_q$. Jak łatwo zauważyć, złożoność tej transformacji jest ograniczona przez stałą razy długość wyrażenia $C$. ■

Problemy CNF i 3-CNF spełnialności nie są wielomianowo transformowalne do 2-CNF spełnialności, dają się natomiast przetransformować do pewnej modyfikacji tego ostatniego problemu. Niech $C_1, C_2, \ldots, C_q$ będą alternatywami

złożonymi z co najwyżej dwóch liter, a $k$ niech będzie liczbą naturalną. Problem *max-2-CNF spełnialności* polega na określeniu układu wartości zmiennych, dla których spełnionych jest co najmniej $k$ alternatyw.

*Problem 3-CNF spełnialności* $\propto$ *Problem max-2-CNF spełnialności.* Bez straty ogólności załóżmy, że każda alternatywa $C_i$ ($i = 1, 2, \ldots, s$) w boolowskim wyrażeniu $C$, które sprowadzone jest do postaci 3-CNF zawiera dokładnie trzy litery $a_i \cup b_i \cup c_i$. Wtedy rodzinę alternatyw $D$ oraz liczbę $k$ w odpowiednim problemie max-2-CNF określamy w następujący sposób

$$D = \bigcup_{i=1}^{s} \{a_i, b_i, c_i, d_i, \bar{a}_i \cup \bar{b}_i, \bar{a}_i \cup \bar{c}_i, \bar{b}_i \cup \bar{c}_i, a_i \cup \bar{d}_i, b_i \cup \bar{d}_i, c_i \cup \bar{d}_i\},$$

$$k = 7s.$$

Sprawdzić teraz można, że $7s$ alternatyw w zbiorze $D$ jest jednocześnie spełnialnych wtedy i tylko wtedy, gdy $C_1 \cap C_2 \cap \ldots \cap C_s$ jest wyrażeniem spełnialnym. ∎

**9.3. Problemy klasy *NP*.** Problemy decyzyjne należące do klasy ***NP*** są rozwiązywalne za pomocą *niedeterministycznego algorytmu o wielomianowym czasie działania* (nondeterministic algorithm operating in polynomial time). Formalna definicja klasy ***NP*** może być podana w terminach *niedeterministycznej maszyny Turinga*, tutaj jednak poprzestaniemy na przedstawieniu kilku nieformalnych kryteriów na to, aby język $L$ należał do ***NP***.

Zasadnicza różnica między algorytmem deterministycznym a niedeterministycznym polega na tym, że w tym drugim przypadku, w każdym kroku algorytmu mamy możliwość wykonania instrukcji polegającej na wyborze kierunku obliczeń. W ten sposób jakiś łańcuch jest akceptowalny przez niedeterministyczną maszynę Turinga, jeśli istnieje ciąg instrukcji, z których pewne mogą być instrukcjami wyboru, kończący się odpowiedzią „tak''. Algorytm niedeterministyczny działa w czasie wielomianowym, jeżeli istnieje wielomian $q(\cdot)$ o tej własności, że dla dowolnego łańcucha $x$, długość ciągu instrukcji jest ograniczona przez $q(|x|)$.

Inna, nieformalna definicja klasy ***NP***, ma następującą postać: język $L$ należy do ***NP***, jeżeli istnieje algorytm typu *podziału i ograniczeń* (branch-and-bound algorithm) o wielomianowej głębokości, który akceptuje $x$ wtedy i tylko wtedy, gdy $x \in L$.

Do klasy ***NP*** należą między innymi: problem spełnialności, problemy istnienia dróg i cykli Hamiltona w grafie i digrafie, oraz problem istnienia całkowitoliczbowego rozwiązania układu nierówności liniowych o współczynnikach całkowitych.

Podstawowe twierdzenie tego rozdziału zostało po raz pierwszy udowodnione przez Cooka [16].

TWIERDZENIE COOKA. *Każdy problem decyzyjny należący do klasy **NP** jest wielomianowo transformowalny do problemu spełnialności.*

Z twierdzenia tego wynika, że jeżeli problem spełnialności jest rozwiązywalny w czasie wielomianowym, to każdy problem z ***NP*** należy do ***P***. Stąd otrzymujemy

WNIOSEK. *$P = NP$ wtedy i tylko wtedy, gdy problem spełnialności należy do $P$.*

D o w ó d. Jeśli problem spełnialności należy do $P$, to każdy problem $L \in NP$ także należy do $P$, ponieważ $L$ jest wielomianowo transformowalny do problemu spełnialności. Z drugiej strony, jeśli problem spełnialności nie należy do $P$, to oczywiście $NP \neq P$. ■

Zatem problem spełnialności jest tak trudny, jak każdy inny problem z klasy $NP$. Kontynuując pracę Cooka, Karp w [59] wykazał, że w powyższych rozważaniach problem spełnialności może być zastąpiony przez wiele innych, trudnych problemów kombinatorycznych.

Problem decyzyjny $L$ nazywać będziemy *NP-zupełnym* (NP-*complete*) jeżeli (i) $L \in NP$ oraz (ii) problem spełnialności jest wielomianowo transformowalny w $L$.

Wykazaliśmy w paragrafie 9.2, że problem spełnialności jest wielomianowo transformowalny do problemu kliki, a więc problem kliki jest problemem NP-zupełnym.

Jeżeli język $L$ jest NP-zupełny, to na podstawie twierdzenia Cooka, $L$ jest wielomianowo transformowalny do problemu spełnialności, czyli $L \in P$ wtedy i tylko wtedy, gdy problem spełnialności należy do $P$. Zatem, albo wszystkie problemy NP-zupełne należą do $P$, albo żaden nie należy. Ta pierwsza możliwość zachodzi wtedy i tylko wtedy, gdy $P = NP$.

Widać zatem, że w definicji NP-zupełności, problem spełnialności może być zastąpiony przez dowolny inny problem NP-zupełny.

**9.4. Problemy NP-zupełne.** W dotychczasowych rozważaniach pojawiły się następujące problemy NP-zupełne: ogólny problem spełnialności i jego szczególne postaci CNF, 3-CNF i max-2-CNF spełnialności oraz problem istnienia $k$-kliki. Od momentu ukazania się prac Cooka i Karpa, klasa problemów NP-zupełnych rozrasta się niemalże z dnia na dzień. We wszystkich dowodach NP-zupełności można wydzielić dwie zasadnicze części, pierwsza, łatwiejsza część polega na pokazaniu, że rozpatrywany problem $L_0$ należy do klasy $NP$, natomiast w drugiej części dowodzi się, że jakiś inny problem $L$, o którym uprzednio udowodniono, że jest NP-zupełny, może być wielomianowo przetransformowany w $L_0$.

Wymienimy teraz szereg ważniejszych i bardziej znanych problemów NP-zupełnych, a następnie pokażemy kilka wielomianowych transformacji zachodzących między niektórymi z nich. Każdy problem opisany jest za pomocą *danych* i *własności*, która musi być spełniona, aby konkretne dane zostały zaakceptowane.

(1) Problemy spełnialności, CNF, 3-CNF i max-2-CNF spełnialności.

(2) $k$-klika.
*Dane*: Graf zwykły $G$ i liczba naturalna $k$.
*Własność*: $G$ zawiera $k$-klikę, tj. podgraf pełny o $k$ wierzchołkach.

(3) Pokrycie grafu klikami.
*Dane*: Graf zwykły $G = (V, X)$ i liczba naturalna $k$.
*Własność*: $V$ jest sumą co najwyżej $k$ klik grafu $G$.

(4) Izomorficzne włożenie.
*Dane*: Grafy zwykłe $G$ i $F$.
*Własność*: $F$ jest izomorficzny z podgrafem grafu $G$.

(5) Kolorowanie grafu.
*Dane*: Graf zwykły $G$ i liczba naturalna $k$.
*Własność*: $G$ można pomalować $k$ kolorami.

(5.1) Kolorowanie trzema kolorami grafu płaskiego o ograniczonych stopniach wierzchołków.
*Dane*: Graf płaski $G$, w którym każdy wierzchołek jest co najwyżej stopnia 4.
*Własność*: $G$ można pomalować trzema kolorami.

(6) Pokrycie grafu.
*Dane*: Graf zwykły $G = (V, X)$ i liczba naturalna $k$.
*Własność*: W $G$ istnieje podzbiór wierzchołków $U \subseteq V$ taki, że $|U| \leqslant k$ i każda krawędź grafu $G$ jest incydentna z co najmniej jednym wierzchołkiem w $U$.

(6.1) Pokrycie grafu o ograniczonych stopniach wierzchołków.
*Dane*: Graf zwykły $G$, w którym każdy wierzchołek jest co najwyżej stopnia 3 i liczba naturalna $k$.
*Własność*: Jak w (6).

(6.2) Pokrycie grafu płaskiego o ograniczonych stopniach wierzchołków.
*Dane*: Graf płaski $G$, w którym każdy wierzchołek jest co najwyżej stopnia 6 i liczba naturalna $k$.
*Własność*: Jak w (6).

(7) Zbiór wierzchołków rozrywających wszystkie kontury.
*Dane*: Digraf $D$ i liczby naturalna $k$.
*Własność*: W $D$ istnieje $k$ wierzchołków, których usunięcie powoduje rozerwanie wszystkich konturów (tj. cykli zorientowanych).

(8) Zbiór łuków rozrywających wszystkie kontury.
*Dane*: Digraf $D$ i liczba naturalna $k$.
*Własność*: W $D$ istnieje $k$ łuków, których usunięcie powoduje rozerwanie wszystkich konturów.

(9) Cykl Hamiltona.
*Dane*: Graf zwykły $G$.
*Własność*: $G$ zawiera cykl Hamiltona.

(9.1) Cykl Hamiltona w kubicznym i 3-spójnym grafie płaskim.
*Dane*: Kubiczny, 3-spójny graf płaski $G$.
*Własność*: $G$ zawiera cykl Hamiltona.

(10) Kontur Hamiltona.
*Dane*: Digraf $D$.
*Własność*: $D$ zawiera kontur Hamiltona.

(10.1) Droga Hamiltona w digrafie płaskim o ograniczonych stopniach wierzchołków.

*Dane*: Digraf płaski $D$, w którym dla każdego wierzchołka $v$ zachodzą nierówności out-degree$(v) \leqslant 4$ i in-degree$(v) \leqslant 3$.

*Własność*: $D$ zawiera drogę Hamiltona.

(11) Problem komiwojażera.

*Dane*: Graf $G = (V, X)$, $c\colon X \to Z$ i liczba naturalna $k$.

*Własność*: $G$ zawiera cykl Hamiltona o sumarycznej wadze nie przekraczającej $k$.

(12) Podział grafu.

*Dane*: Graf zwykły $G = (V, X)$ o $2p$ wierzchołkach i liczba naturalna $k$.

*Własność*: Zbiór wierzchołków $V$ może być rozłożony na dwa podzbiory $V = V_1 \cup V_2$ takie, że $|V_1| = |V_2| = p$ i liczba krawędzi z $V_1$ do $V_2$ jest mniejsza lub równa $k$.

(13) Pokrycie grafu trójkątami.

*Dane*: Graf zwykły $G$ o $3p$ wierzchołkach.

*Własność*: Graf $G$ może być pokryty $p$ trójkątami.

(14) Maksymalne cięcie w sieci.

*Dane*: Graf zwykły $G = (V, X)$, $w\colon V \to Z$ i liczba naturalna $k$.

*Własność*: W $G$ istnieje podzbiór wierzchołków $U \subseteq V$ taki, że

$$\sum_{\substack{[u,v] \in X \\ u \in U \\ v \notin U}} w(u, v) \geqslant k.$$

(14.1) Maksymalne cięcie grafu.

Problem poprzedni dla $w(x) = 1 (\forall x \in X)$.

(15) Jądro digrafu.

*Dane*: Digraf $D = (V, X)$.

*Własność*: W $D$ istnieje podzbiór wierzchołków $K \subseteq V$ taki, że żadne dwa wierzchołki z tego zbioru nie są połączone łukiem i dla każdego $u \notin K$ istnieje łuk z $u$ do wierzchołka $v \in K$.

(16) Optymalne uporządkowanie wierzchołków.

*Dane*: Graf zwykły $G = (V, X)$ i liczba naturalna $k$.

*Własność*: Istnieje wzajemnie jednoznaczne przyporządkowanie $\pi\colon V \leftrightarrow \{1, 2, \ldots, n = |V|\}$ takie, że

$$\sum_{[u,v] \in X} |\pi(u) - \pi(v)| \leqslant k.$$

(17) Drzewo Steinera.

*Dane*: Graf zwykły $G = (V, X)$, podzbiór $U \subseteq V$, liczba naturalna $k$ oraz funkcja $w\colon V \to Z$.

*Własność*: Graf $G$ zawiera drzewo $T = (W, Y)$ takie, że $U \subseteq W \subseteq V$ oraz

$$\sum_{[u,v]\in Y} w(u, v) \leqslant k.$$

(18) Zbiór reprezentantów rodziny podzbiorów.
*Dane*: Rodzina podzbiorów $\{U_i\}$ zbioru $\{s_j\colon j = 1, 2, \ldots, r\}$.
*Własność*: Istnieje podzbiór $W$ taki, że $|U_i \cap W| = 1$ dla każdego $i$.

(19) Rozkład liczb naturalnych.
*Dane*: Zbiór liczb naturalnych $\{A_1, A_2, \ldots, A_n\}$.
*Własność*: Istnieje podzbiór $I \subseteq \{1, 2, \ldots, n\}$ taki, że $\sum_{i\in I} A_i = \sum_{i\notin I} A_i$.

(20) Pokrycie zbiorów zbiorami rozłącznymi.
*Dane*: Rodzina zbiorów skończonych $\{S_i\}_{i\in J}$.
*Własność*: Istnieje podrodzina $\{S_i\}_{i\in I} (I \subset J)$ zbiorów rozłącznych taka, że

$$\bigcup_{i\in I} S_i = \bigcup_{i\in J} S_i.$$

(21) Pokrycie zbiorów.
*Dane*: Rodzina zbiorów skończonych $\{S_i\}_{i\in J}$ i liczba naturalna $k$.
*Własność*: Istnieje podrodzina $\{S_i\}_{i\in L}$ taka, że $|L| \leqslant k$ i $\bigcup_{i\in L} S_i = \bigcup_{i\in J} S_i$.

(22) 3-wymiarowe skojarzenie.
*Dane*: Trzywymiarowa tablica $p \times p \times p$, której pewne elementy uważamy za dopuszczalne.
*Własność*: Istnieje $p$ dopuszczalnych elementów tablicy, z których żadne dwa nie mają żadnej wspólnej współrzędnej.

(23) Programowanie w liczbach całkowitych.
*Dane*: Macierz $A$ i wektor $d$ o elementach całkowitych.
*Własność*: Istnieje wektor o elementach 0 i 1 taki, że $Ax = d$.

(24) Problem plecakowy.
*Dane*: Liczby całkowite $a_1, a_2, \ldots, a_n, b$.
*Własność*: Równanie $\sum_{i=1}^{n} a_i x_i = b$ ma rozwiązanie, w którym $x_i = 0$ lub 1 $(i = 1, 2, \ldots, n)$.

(25) Problem szeregowania zadań.
*Dane*: Danych jest $n$ zadań $T_1, T_2, \ldots, T_n$, z których każde musi być wykonane kolejno na $m$ maszynach $P_1, P_2, \ldots, P_m$. Niech $(t_{1i}, t_{2i}, \ldots, t_{mi})$ będzie wektorem czasów wykonywania $i$-tego zadania na kolejnych maszynach. Dana jest także liczba naturalna $k$.
*Własność*: Istnieje uszeregowanie wszystkich zadań, którego długość nie przekracza $k$.

(25.1) Problem szeregowania zadań na trzech maszynach.

Problem (25) dla $n = 3$.

Większość z wymienionych tutaj problemów NP-zupełnych znaleźć można w pracach Cooka [16], Karpa [59], [61], Gareya, Johnsona i Stockmeyera [40] oraz w książce [2]. Tam też podane są kompletne transformacje dowodzące, że przynależą one do klasy problemów NP-zupełnych.

W dalszej części tego paragrafu, przedstawione w paragrafie 9.2 transformacje uzupełnimy szkicem tylko kilku innych, których wyprowadzenie nie wymaga zbyt rozbudowanego aparatu teorii grafów i teorii mnogości.

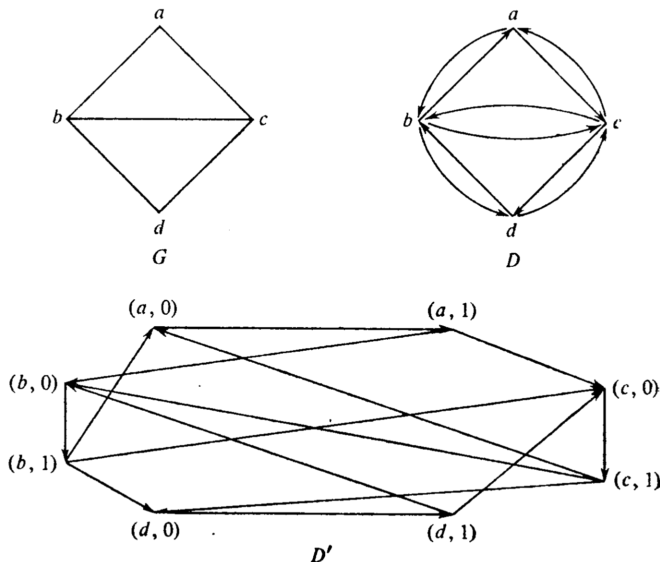
*k-klika* (2) $\propto$ *Pokrycie grafu* (6). Niech $G = (V, X)$ będzie grafem, który badamy, czy zawiera $k$-klikę. Transformacja zachowująca problem polega na przejściu do grafu dopełniającego $G' = (V, V\times V-X)$, który badamy, czy zawiera pokrycie mocy $|V|-k$. ■

*Pokrycie grafu* (6) $\propto$ *Zbiór wierzchołków rozrywających wszystkie kontury* (7).

*Pokrycie grafu* (6) $\propto$ *Zbiór łuków rozrywających wszystkie kontury* (8).

Niech $G = (V, X)$ będzie grafem, który badamy, czy zawiera $k$ wierzchołkowe pokrycie. Transformacja problemu pokrycia grafu do problemu (7) polega na zastąpieniu grafu $G$ digrafem $D$, w którym każda krawędź z $G$ zastąpiona jest przeciwnie skierowanymi łukami. Badamy teraz, czy digraf $D$ zawiera $k$ wierzchołków rozrywających wszystkie kontury. Dla wykazania, że transformacja te zachowuje problem wystarczy zauważyć, że każdej krawędzi z $G$ odpowiada w $D$ kontur długości 2.

Transformacja problemu (6) do problemu (8) polega na konstrukcji digrafu $D'$, w którym każdy wierzchołek $v$ grafu $G$ zastąpiony jest parą $(v, 0)$ i $(v, 1)$ połączoną




Rys. 7

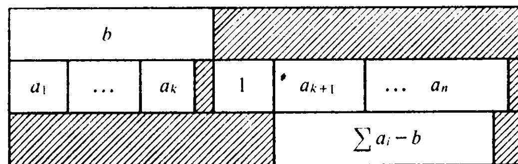
łukiem skierowanym do $(v, 1)$, a każdej krawędzi $(v, u)$ grafu $G$ odpowiada łuk $((u, 1), (v, 0))$. W tak utworzonym digrafie $D'$ sprawdzamy, czy istnieje $k$ łuków rozrywających wszystkie kontury. Rysunek 7 ilustruje obie te transformacje. ■

*Kolorowanie grafu* (5) $\propto$ *Pokrycie grafu klikami* (3). Dla zbadania, czy graf $G$ może być pokolorowany $k$ kolorami potrzeba i wystarcza sprawdzić, czy jego dopełnienie ma pokrycie zbioru wierzchołków $k$-klikami. ■

*Pokrycie zbiorów zbiorami rozłącznymi* (20) $\propto$ *Zbiór reprezentantów rodziny podzbiorów* (18). Aby zbadać, czy istnieje podrodzina $\{S_i\}_I$, $I \subseteq J$, zbiorów rozłącznych takich, że $\bigcup_{i \in I} S_i = \bigcup_{i \in J} S_i$, potrzeba i wystarcza sprawdzić, czy istnieje zbiór $W$, który ma dokładnie jeden element wspólny z każdym ze zbiorów $U_l$, gdzie $s_k \in U_l$ wtedy i tylko wtedy, gdy $u_l \in S_k$. ■

*Problem plecakowy* (24) $\propto$ *Problem szeregowania zadań na trzech maszynach* (25.1). Aby znaleźć rozwiązanie problemu plecakowego o danych $a_1, a_2, \ldots, a_n, b$ pytamy, czy możliwe jest wykonanie na trzech maszynach $n+1$ zadań o następujących czasach wykonywania $(0, a_1, 0)$, $(0, a_2, 0)$, ..., $(0, a_n, 0)$ i $\left(b, 1, \sum_{i=1}^{n} a_i - b\right)$ w czasie $\sum_{i=1}^{n} a_i + 1$.

Rysunek 8 może być pomocny przy dowodzie, że tak określony zbiór zadań może być wykonany na trzech maszynach w czasie $\sum_{i=1}^{n} a_i + 1$ wtedy i tylko wtedy, gdy istnieje podzbiór liczb $\{a_i\}$ dający w sumie $b$. ■




Rys. 8

Wśród wyszczególnionych wyżej problemów NP-zupełnych można znaleźć bardzo wiele problemów ogólnych, w wielu jednak przypadkach staraliśmy się podać także bardziej szczegółowe wersje problemów, utworzone z problemów ogólnych przez ograniczenie rozpatrywanej klasy grafów i wartości parametrów. Postąpiliśmy tak z dwóch powodów. Po pierwsze, szczegółowe wersje problemów ogólnych, np. (5.1), (6.1), (6.2), (9.1), (10.1) i (25.1) wyznaczają w pewnym sensie granicę między klasami grafów, przekroczenie której powoduje radykalną zmianę typu złożoności problemu. Oznacza to w tych przypadkach, że zmniejszenie wartości któregoś z parametrów tylko o 1 powoduje, iż każdy z tych NP-zupełnych problemów staje się problemem klasy $\boldsymbol{P}$. Na przykład, rozwiązanie problemu uszeregowania $n$ zadań

na *dwóch* maszynach może być znalezione w czasie $O(n \log n)$. Z drugiej strony, te szczegółowe wersje problemów ogólnych pozwalają na określenie złożoności obliczeniowej tych problemów dla innych, zwykle obszerniejszych klas grafów. I tak np., na podstawie problemu (9.1) wnioskujemy, że problemy istnienia cyklu Hamiltona w 3-spójnym grafie płaskim lub w kubicznym grafie płaskim są także NP-zupełne.

Musimy pamiętać jednak, że istnieją i takie klasy grafów, dla których wiele wymienionych wyżej problemów można rozwiązywać za pomocą wielomianowych algorytmów, np. maksymalna klika w $n$-wierzchołkowym grafie płaskim może być znaleziona w czasie ograniczonym przez $O(n^4)$, gdyż graf płaski może zawierać klikę o co najwyżej czterech wierzchołkach.

Przed końcowymi uwagami przypomnijmy jeszcze raz dwie najważniejsze własności charakteryzujące problemy NP-zupełne:

(a) Dla żadnego problemu NP-zupełnego nie jest znany obecnie żaden wielomianowy algorytm rozwiązywania.

(b) Jeśli znany będzie jakikolwiek algorytm wielomianowy dla chociaż jednego z tych problemów, to będzie można otrzymać algorytm wielomianowy dla każdego problemu NP-zupełnego.

W świetle tych dwóch własności przypuszcza się, że dla żadnego z problemów NP-zupełnych nie istnieje wielomianowy algorytm rozwiązywania. Głębsza analiza problemów trudnych doprowadziła w wielu przypadkach do wykazania że nie wszystkie problemy NP-zupełne są tak samo „trudne", por. pracę Gareya i Johnsona [44].

Dla ilustracji rozważmy problem rozkładu zbioru liczb naturalnych (19) $\{A_i\}_{i \in I}$ na dwa rozłączne podzbiory $I_1$, $I_2$ o tej własności, że $\sum_{i \in I_1} A_i = \sum_{i \in I_2} A_i$. Bez straty ogólności załóżmy, że $\sum_{i \in I} A_i = 2b$.

Problem ten może być rozwiązany za pomocą algorytmu programowania dynamicznego, który ma złożoność $O(nb)$. Zatem widzimy, że NP-zupełność tego problemu ma swoje źródło w modelu obliczeń i jest wynikiem dopuszczenia do rozważań dowolnie dużych liczb naturalnych. Gdyby wartości wszystkich liczb $\{A_i\}$ były ograniczone z góry, np. przez wielomian zmiennej $n$, to ten algorytm byłby o wielomianowej złożoności.

Istnieje wiele praktycznych zastosowań, w których występujące liczby są w naturalny sposób ograniczone z góry. Dla przykładu, po pierwsze, można wykazać, że powyższy problem rozkładu jest równoważny problemowi wyznaczenia uszeregowania $n$ zadań na dwóch maszynach, którego długość nie przekracza $b$. Z drugiej zaś strony, w tym konkretnym przykładzie zastosowań, praktyczne wartości parametru $b$ są ograniczone, a więc algorytm o złożoności $O(nb)$ może okazać się w praktyce bardzo efektywny. Takie algorytmy, których złożoność jest ograniczona przez wielomian, którego zmiennymi są rozmiar problemu i maksymalna różnica między

występującymi liczbami nazywamy *pseudo-wielomianowymi*. Problemy, które nie mają nawet pseudo-wielomianowych algorytmów (chyba że $P = NP$) uważać możemy za *silnie* NP-zupełne. W tym kontekście, problem komiwojażera (11) jest silnie NP-zupełny, gdyż nawet dla wszystkich wag równych 1 pozostaje on nadal problemem NP-zupełnym (problem 9).

## 10. Zakończenie. Problemy

Nasze rozważania zakończymy kilkoma uwagami o aktualnych problemach badawczych w dziedzinie złożoności obliczeniowej problemów kombinatorycznych.

Oczywiście, centralnym problemem badawczym jest stwierdzenie, czy $P = NP$. Następnie należałoby wyjaśnić status wielu ważnych problemów takich jak badanie izomorfizmu grafów i programowanie liniowe. O tym pierwszym problemie wiadomo tylko, że należy do klasy $NP$, natomiast o ogólnym zagadnieniu programowania liniowego udowodniono jedynie, że powszechnie stosowana metoda simpleks i wiele jej uogólnień i modyfikacji są w najgorszych przypadkach algorytmami o złożoności niewielomianowej [67], [122]. Ten ostatni fakt wydaje się mieć większe znaczenie teoretyczne niż praktyczne, bo metoda simpleks wraz ze swoimi modyfikacjami z powodzeniem stosowana jest do rozwiązywania rzeczywistych problemów z górą od 30 lat.

Niepowodzenia w badaniach nad udowodnieniem równości klas $P = NP$ skłaniają do przypuszczenia, że dla żadnego problemu NP-zupełnego nie istnieje wielomianowy algorytm rozwiązywania. W tym kontekście bardzo ważnym przedsięwzięciem staje się stwierdzenie, czy dany problem jest NP-zupełny, czy nie, bo jeżeli należy do tej klasy problemów, to w poszukiwaniu metod rozwiązywania musimy się raczej zdecydować na podejścia heurystyczne (przybliżone), które byłyby w stanie szybko dostarczać rozwiązań dobrze przybliżających rozwiązanie optymalne. Nie jest więc żadną niespodzianką, że heurystyczne metody rozwiązywania problemów NP-zupełnych stały się ostatnio tak bardzo popularne. W pracy Gareya i Johnsona [42] znaleźć można omówienie najnowszych prac poświęconych tym problemom. Dla przykładu wspomnijmy o problemie komiwojażera. Z jednej strony, opracowano szereg efektywnych metod dokładnych dla szczególnych układów danych oraz efektywnych algorytmów przybliżonych, o których wiadomo, z jaką dokładnością mogą dać rozwiązanie, patrz [99]. Z drugiej zaś wykazano, że niektóre ze stosowanych obecnie podejść nie mogą doprowadzić do szybkich i dobrych algorytmów przybliżonych, np. udowodniono ten fakt dla metody lokalnej optymalizacji, patrz [101].

Jak już wspomnieliśmy przy końcu poprzedniego paragrafu, w wielu przypadkach problemy trudne stają się wielomianowo rozwiązywalne dla niektórych szczególnych podklas.

Dążyć więc należy do wydzielania coraz to większych podklas danych, dla których problemy NP-zupełne stają się wielomianowo rozwiązywalne. Badania nad strukturą tych łatwych podprzypadków rzucają dodatkowe światło na strukturę

problemów trudnych. Z drugiej strony, dążyć należy do coraz większego uściślenia parametrów problemów trudnych, co pozwoli dokładniej określić granicę między łatwymi i trudnymi przypadkami problemów, o których wiadomo, że w ogólnym przypadku są NP-zupełne.

Jednym z najpowszechniej stosowanych podejść do rozwiązywania trudnych problemów kombinatorycznych jest metoda podziału i ograniczeń, która stosowana jest bardzo często bez należytego zbadania jej teoretycznych własności. Wielu użytkowników tego podejścia skłania się do przekonania, że metoda ta, tak jak i wiele innych metod, takich jak metody cięć czy metody programowania dynamicznego, są raczej bezużyteczne w rozwiązywaniu problemów o praktycznych rozmiarach.

Inny kierunek badań nad złożonością obliczeniową reprezentują prace poświęcone charakteryzacji zbiorów (zwykle wielościanów) rozwiązań dopuszczalnych problemów trudnych. Dotychczas nie udało się otrzymać żadnych zadowalających wyników w tym kierunku.

Na zakończenie już wspomnijmy o szerokim zainteresowaniu średnią złożonością obliczeniową problemów i algorytmami probabilistycznymi, patrz dla przykładu prace Karpa [62] i Rabina [87]. W pracy Karpa zilustrowano szereg probabilistycznych metod rozwiązywania problemów NP-zupełnych, które przy ustalonym rozkładzie prawdopodobieństwa danych, pozwalają w wielomianowym czasie i dla prawie wszystkich układów danych wyznaczać rozwiązania optymalne lub bliskie optymalnym. Rabin natomiast, zamiast posługiwać się średnią złożonością obliczeniową, która wymaga określenia rozkładu prawdopodobieństwa danych, wprowadził pojęcie *algorytmu probabilistycznego.* Innymi słowy, losowość przeniesiona została z danych na sam algorytm. Trochę nieoczekiwaną propozycją jest w ostatniej pracy algorytm probabilistyczny, który z pewnym prawdopodobieństwem (większym od zera) może dawać rozwiązanie niepoprawne.

Badania nad średnią złożonością obliczeniową są spokrewnione z badaniami przybliżonych metod rozwiązywania, patrz jeszcze raz [42].

## 11. Podziękowania

Autor bardzo dziękuje dr. W. Lipskiemu, Jr., z Instytutu Podstaw Informatyki PAN, dr. M. Kubale z Instytutu Informatyki Politechniki Gdańskiej oraz anonimowemu recenzentowi za wiele cennych uwag, które pozwoliły ulepszyć pierwotną wersję tej pracy rozpowszechnianą początkowo jako Raport Nr N-19 Instytutu Informatyki Uniwersytetu Wrocławskiego.

## Bibliografia

[1]* A. V. Aho (red.), *Currents in the theory of computing*, Prentice-Hall, 1973.

[2]* A. V. Aho, J. E. Hopcroft i J. D. Ullman, *The design and analysis of computer algorithms*, Addison-Wesley, 1974 (tłum. ros., 1979).

[3] E. A. Akkoyunlu, *The enumeration of maximal cliques of large graphs*, SIAM J. Comput. 2 (1973), str. 1–6.

[4] M. Bellmore i G. I. Nemhauser, *The travelling salesman problem: a survey*, Operations Res. 16 (1968), str. 538–558.

[5]* A. T. Berztiss, *Data structures: theory and practice*, Academic Press, 1971.

[6] —, *A backtrack procedure for isomorphism of directed graphs*, J.ACM 20(1973), str. 365–377.

[7] M. Blum, R. W. Floyd, V. R. Pratt, R. L. Rivest i R. E. Tarjan, *Time bounds for selection*, J. Computer and System Sciences 7 (1972), str. 448–461.

[8] A. B. Borodin, *Computational complexity: theory and practice*, w [1] str. 35–89.

[9]* A. B. Borodin i I. Munro, *The computational complexity of algebraic and numeric problems*, American Elsevier, 1975.

[10] D. Cheriton i R. E. Tarjan, *Finding minimum spanning trees*, SIAM J. Comput. 5 (1976), str. 724–742.

[11]* N. Christofides, *Graph theory: An algorithmic approach*, Academic Press, 1975 (tłum. ros., 1978)

[12]* E. G. Coffman, Jr. (red.), *Computer and job-shop scheduling theory*, J. Wiley and Sons, 1976. Teoria szeregowania zadań, WNT, 1980.

[13] J. Cohen i M. Roth, *On the implementation of Strassen's fast multiplication algorithm*, Acta Informatica 6 (1976), str. 341–355.

[14] S. A. Cook, *Linear time simulation of deterministic two-way pushdown automata*, Proc. IFIP Congress 71, TA-2, North-Holland, 1971, str. 172–179.

[15] —, *Characterizations of pushdown machines in terms of time-bounded computers*, J. ACM 18 (1971), str. 4–18.

[16] —, *The complexity of theorem-proving procedures*, Proc. 3rd ACM Symp. Theory of Computing, 1971, str. 151–158.

[17] S. A. Cook i R. A. Reckow, *Time-bounded random access machines*, J. Computer and System Sciences 7 (1973), str. 354–362.

[18] D. G. Corneil, *Graph isomorphism*, Ph. D. thesis, University of Toronto, 1968.

[19] —, *The analysis of graph theoretical algorithms*, Proc. 5th S-E Conf. on Combinatorics, Graph Theory and Computing, Utilitas Mathematica, 1974, str. 3–38.

[20]* N. Deo, *Graph theory with applications to engineering and computer science*, Prentice-Hall, 1974; *Teoria grafów i jej zastosowania w technice i informatyce*, PWN, 1980.

[21] —, *Design and analysis of algorithms*, J. Computer Soc. India 6 (1976), str. 3–8.

[22] N. Deo, J. M. Davis i R. E. Lord, *A new algorithm for digraph isomorphism*, BIT 17 (1977), str. 16–30.

[23] E. Dijkstra, *A note on two problems in connection with graphs*, Numer. Math. 1 (1959), str. 269–271.

[24] E. A. Dinic, *Algorithm for solution of a problem of maximum flow in a network with power estimation*, Sov. Math. Dokl. 11 (1970), str. 1277–1280.

[25] D. Dreyfus, *An appraisal of some shortest path algorithms*, Operations Res. 17 (1969), str. 395–412.

[26] J. Edmonds, *Paths, trees and flowers*, Canad. J. Math. 17 (1965), str. 449–467.

[27] J. Edmonds i E. L. Johnson, *Matching: A well-solved class of integer linear programs*, Combinatorial structures and their applications, Gordon and Breach, 1970, str. 89–92.

[28] J. Edmonds, *Matroids and the greedy algorithms*, Mathematical Programming 1 (1971), str. 127–136.

[29] J. Edmonds i R. M. Karp, *Theoretical improvements in algorithmic efficiency for network flow problems*, J. ACM 19 (1972), str. 248–264.

[30]* S. Even, *Algorithmic combinatorics*, Macmillan, 1973.

[31] S. Even i O. Kariv, *An $O(n^{5/2})$ algorithm for maximum matching in general graphs*, Proc. 16th Ann. Symposium on Foundations of Comp. Sci., IEEE, 1975, str. 100–112.

[32] S. Even i R. E. Tarjan, *Network flow and testing graph connectivity*, SIAM J. Comput. 4 (1975), str. 507–518.

[33] M. J. Fischer i A. R. Meyer, *Boolean matrix multiplications and transitive closure*, Conf. Record, IEEE 12th Ann. Symp. on Switching and Automata Theory, 1971, str. 129–131.

[34] R. W. Floyd, *Nondeterministic algorithm*, J. ACM 14 (1967), str. 636–644.

[35] M. Fontet, *A linear algorithm for testing isomorphism of planar graphs*, Automata, languages and programming, Edinburgh University Press, 1976, str. 411–424.

[36] L. R. Ford, Jr. i S. M. Johnson, *A tournament problem*, Amer. Math. Monthly 66 (1959), str. 387–389.

[37] W. D. Frazer, *Analysis of combinatory algorithm — A sample of current methodology*, Spring Joint Computer Conference, 1972, str. 483–491.

[38] M. L. Fredman, *New bounds on the complexity of the shortest path problem*, SIAM J. Comput. 5 (1976), str. 87–89.

[39] H. Gabow, *An efficient implementation of Edmonds' algorithm for maximum matching*, J. ACM 23 (1976), str. 221–234.

[40] M. R. Garey, D. S. Johnson i L. Stockmeyer, *Some simplified NP-complete graph problems*, Theoretical Computer Science 1 (1976), str. 237–267.

[41] M. R. Garey i D. S. Johnson, *The complexity of near-optimal graph coloring*, J. ACM 23 (1976), str. 43–49.

[42] —, —, *Approximation algorithms for combinatorial problems: an annotated bibliography*, w [115], str. 41–52.

[43] M. R. Garey, D. S. Johnson i R. E. Tarjan, *The planar Hamiltonian circuit problem is NP-complete*, SIAM J. Comput. 5 (1976), str. 704–714.

[44] M. R. Garey i D. S. Johnson, *Strong NP-completeness results: motivation, examples and implications*, J. ACM 25 (1978), str. 499–508.

[45]* F. Harary, *Graph theory*, Addison-Wesley, 1969 (tłum. ros., 1973).

[46] A. J. Hoffman i S. Winograd, *Finding all shortest distances in a directed network*, IBM J. Res. Develop. 16 (1972), str. 412–414.

[47] R. C. Holt i E. M. Reingold, *On the time required to detect cycles and connectivity in graphs*, Mathematical Systems Theory 6 (1972), str. 103–106.

[48] J. E. Hopcroft i L. Kerr, *On minimizing the number of multiplications necessary for matrix multiplication*, SIAM J. Appl. Math. 20 (1971), str. 30–36.

[49] J. E. Hopcroft i R. E. Tarjan, *Isomorphism of planar graphs*, w [81], str. 131–152.

[50] —, —, *Dividing a graph into triconnected components*, SIAM J. Comp. 2 (1973), str. 135–158.

[51] J. E. Hopcroft i R. E. Tarjan, *Algorithm 447. Efficient algorithms for graph manipulation*, Comm. ACM 16 (1973), str. 372–378.

[52] J. E. Hopcroft i R. M. Karp, *An $n^{5/2}$ algorithm for maximum matching in bipartite graphs*, SIAM J. Comp. 2 (1973), str. 225–231.

[53] J. E. Hopcroft i J. K. Wong, *Linear time algorithm for isomorphism of planar graphs (extended abstract)*, Proc. 6th Ann. ACM Symp. on Theory of Computing, 1974, str. 172–184.

[54] J. E. Hopcroft i R. E. Tarjan, *Efficient planarity testing*, J. ACM 21 (1974), str. 541–568.

[55] J. E. Hopcroft, *Complexity of computer computations*, Information Processing 74, North-Holland, str. 620–626.

[56] M. I. Irland i P. C. Fischer, *A bibliography on computational complexity*, Research Report CSRR 2028, Dept. of Appl. Analysis and Computer Sci., University of Waterloo, Oct. 1970.

[57] M. Jankowski i H. Woźniakowski, *O złożoności obliczeniowej w analizie numerycznej*, Matematyka Stosowana 5 (1975), str. 5–27.

[58] D. S. Johnson, *Approximation algorithms for combinatorial problems*, J. Comp. System Sci. 9 (1974), str. 256–278.

[59] R. M. Karp, *Reducibility among combinatorial problems*, w [81], str. 85–103.

[60]* R. M. Karp (red.), *Complexity of computation*, SIAM-AMS Proc., 1974.

[61] R. M. Karp, *On the computational complexity of combinatorial problems*, Networks 5 (1975), str. 45–68.

[62] —, *The probabilistic analysis of some combinatorial search algorithms*, w [115], str. 1–22.

[63] B. W. Kernighan i S. Lin, *An efficient heuristic procedure for partitioning graphs*, Bell Syst. Tech. J. 2 (1970), str. 291–307.

[64] A. Kerschenbaum i R. Van Slyke, *Computing minimum spanning trees efficiently*, Proc. 25th Am. Conf. of the ACM, 1972, str. 518–527.

[65] D. Kirkpatrick, *Determining graph properties from matrix representation*, Proc. 6th Ann. ACM Symp. on Theory of Computing, 1974, str. 84–90.

[66] S. S. Kislicyn, *On the selection of the k-th element of an ordered set by pairwise comparisons*, Sibirsk. Math. Zh. 5 (1964), str. 557–564.

[67] V. Klee i G. J. Minty, *How good is the simplex algorithm*?, Inequalities III (O. Shisha, red.), Academic Press, 1972, str. 159–175.

[68]* D. E. Knuth, *The art of computer programming*, Vol. 1, *Fundamental algorithms*, Addison-Wesley, 1968, Vol. 3, *Sorting and searching*, Addison-Wesley, 1973 (tłum. ros., 1976, 1979).

[69] —, *Optimum binary search trees*, Acta Informatica 1 (1971), str. 14–25.

[70]* Z. Kohavi i A. Paz (red.), *Theory of machines and computations*, Academic Press, 1971.

[71]* J. Kucharczyk i M. M. Sysło, *Algorytmy optymalizacji w języku ALGOL 60*, PWN, Warszawa 1975, Wyd. III rozszerzone, 1982.

[72] R. E. Ladner, *On the structure of polynomial time reducibility*, J. ACM 22 (1975), str. 155–171.

[73] E. L. Lawler, *The complexity of combinatorial computations: A survey*, Proc. of 1971 Polytechnic Inst. of Brooklyn Symp. on Computers and Automata, 1971, str. 305–312.

[74] —, *Algorithms, graphs, and complexity*, Networks 5 (1975), str. 89–92.

[75]* —, *Combinatorial optimization: networks and matroids*, Holt, Reinhart and Winston, 1976.

[76] D. H. Lehmer, *Teaching combinatorial tricks to a computer*, Combinatorial Analysis (R. Bellman i M. Hall, Jr., red.), Proc. Sympos. Appl. Math. 10 (1960), str. 179–193, American Mathematical Society, Providence, Rhode Island.

[77] —, *The machine tools of combinatorics*, Applied Combinatorial Mathematics (E. Beckenbach, red.), Wiley, 1964, str. 5–31.

[78] J. K. Lenstra, A. H. G. Rinnooy Kan i P. Brucker, *Complexity of machine scheduling problems*, Annals of Discrete Math. 1 (1977), str. 343–362.

[79]* C. L. Liu, *Introduction to combinatorial theory*, McGraw-Hill, 1968.

[80] R. B. Marimont, *A new method of checking the consistency of precedence matrices* J. ACM 6 (1959), str. 164–171.

[81]* R. E. Miller i J. W. Thatcher (red.), *Complexity of computer computations*, Plenum Press, 1972.

[82] G. J. Minty, *A simple algorithm for listing all the trees of a graph*, IEEE Trans. on Circuit Theory CT-12 (1965), str. 120.

[83] I. Munro, *Some results in the study of algorithms*, Technical Report Nr 32, Department of Computer Science, University of Toronto, Toronto, Ontario, Canada.

[84] M. Nakamori, *A note on the optimality of some all-shortest-path algorithms*, J. Operations Res. Soc. Japan 15 (1972), str. 201–204.

[85] P. E. O'Neil i E. J. O'Neil, *A fast expected time algorithm for Boolean matrix multiplication and transitive closure*, Information and Control 22 (1973), str. 132–138.

[86] I. Pohl, *A sorting problem and its complexity*, Comm. ACM 15 (1972), str. 462–464.

[87] M. O. Rabin, *Probabilistic algorithms*, w [115], str. 21–39.

[88] R. C. Read, *Teaching graph theory to a computer*, Recent Progress in Combinatorics (T. Tutte, red.), Academic Press, 1969, str. 161–163.

[89] —, *Graph theory algorithms*, Graph Theory and its Application (B. Harris, red.), Academic Press, 1970, str. 51–78.

[90]* R. C. Read (red.), *Graph theory and computing*, Academic Press, 1972.

[91] R. C. Read i R. E. Tarjan, *Bounds on bactrack algorithms for listing cycles, paths and spanning trees*, Networks 5 (1975), str. 237–252.

[92] E. M. Reingold, *On the optimality of some set algorithms*, J. ACM 19 (1972), str. 649–659.

[93] —, *Establishing lower bounds on algorithms — A survey*, Proc. AFIPS 1972 Spring Joint Computer Conf., str. 471–481.

[94] —, *A bibliography of the analysis of algorithms*, Dept. of Comp. Sc., University of Illinois, Urbana, 1972, str. 1–12.

[95]* E. M. Reingold, J. Nievergelt i N. Deo, *Combinatorial computing*, Prentice-Hall, 1977 (tłum. ros., 1980).

[96] R. L. Rivest i D. E. Knuth, *Bibliography 26. Computer Sorting*, Computing Reviews, 1972.

[97] R. L. Rivest i J. Vuillemin, *On recognizing graph properties from adjacency matrices*, Theoret. Comp. Sci. 3 (1976/77), str. 371–384.

[98] A. L. Rosenberg, *On the time required to recognize properties of graphs: a problem*, SIGACT News 5 (1973).

[99] D. J. Rosenkrantz, R. E. Stearns i P. M. Lewis II, *An analysis of several heuristics for the travelling salesman problem*, SIAM J. Comput. 6 (1977), str. 563–581.

[100]* A. Rustin (red.), *Computational complexity*, Courant Computer Science Symposium 7, 1971, Algorithmic Press, 1973.

[101] S. Savage, P. Weiner i A. Bagchi, *Neighborhood search algorithm for guaranteeing optimal travelling salesman tours must be inefficient*, J. Comp. Sys. Sci. 12 (1976), str. 25–35.

[102]* H. Steinhaus, *Kalejdoskop matematyczny*, PZWS, Warszawa, 1956.

[103] V. Strassen, *Gaussian elimination is not optimal*, Numer. Math. 13 (1969), str. 354–356.

[104] M. M. Sysło, *Stosowana teoria grafów I. Zastosowanie teorii grafów w metodach numerycznych*, Matematyka Stosowana 5 (1975), str. 69–87.

[105] —, *Stosowana teoria grafów II. Problem ekstremalnych dróg w grafach i sieciach*, Archiwum Automatyki i Telemechaniki 20 (1975), str. 287–300.

[106] M. M. Sysło i Z. Skupień, *Stosowana teoria grafów III. Grafy Eulera i Hamiltona. Zagadnienie komiwojażera*, Matematyka Stosowana 10 (1977), str. 1–54.

[107] M. M. Sysło i J. Dzikiewicz, *Computational experiences with some transitive closure algorithms*, Computing 15 (1975), str. 33–39.

[108] M. M. Sysło i M. Iri, *Efficient outerplanarity testing*, Fundamenta Informaticae 2 (1979), 261–275.

[109] M. M. Sysło, *Generalizations of the standard travelling salesman problem*, Zastosow. Matem. 16 (1980), str. 621–629.

[110] —, *Linear time algorithm for coding outerplanar graphs*, Beiträge zur Graphentheorie und deren Anwendungen, Ilmenau 1978, str. 259–269.

[111]* —, *Teoria grafów w ujęciu algorytmicznym*, PWN (w przygotowaniu).

[112] R. E. Tarjan, *Depth-first search and linear graph algorithms*, SIAM J. Comput. 1 (1972), str. 146–160.

[113] —, *Testing graph connectivity*, Proc. 6th Annual ACM Symposium on Theory of Computing, 1974, str. 185–193.

[114] —, *Complexity of combinatorial algorithms*, SIAM Review 20 (1978), str. 457–491.

[115]* J. F. Traub (red.), *Algorithms and complexity. New directions and recent results*, Academic Press, 1976.

[116]* J. F. Traub (red.), *Analytic computational complexity*, Academic Press, 1976.
[117] J. D. Ullman, *NP-complete scheduling problems*, J. Comp. Sys. Sci. 10 (1975), str. 384–393.
[118]* M. B. Wells, *Elements of combinatorial computing*, Pergamon Press, 1971.
[119]* N. Wirth, *Algorithms + data structures = programs*, Prentice-Hall, 1976.
[120] A. Ch.-Ch. Yao, *An $O(|E|\log|V|)$ algorithm for finding minimum spanning trees*, Information Processing Letters 4 (1975), str. 21–29.
[121]* J. Yen, *Shortest path network problems*, Verlag A. Hain, 1975.
[122] N. Zadeh, *A bad network problem for the simplex method and other minimum cost flow algorithm*, Mathematical Programming 5 (1973), str. 255–266.

**Dodane w korekcie.** Od momentu przyjęcia tej pracy do druku jesteśmy świadkami istnego potopu publikacji poświęconych konstrukcjom, analizie i złożoności obliczeniowej algorytmów kombinatorycznych. Nie sposób więc w kilku zdaniach uzupełnić i uaktualnić całą pracę, która z założenia i tak dotyka jedynie większości z omawianych problemów.

Przede wszystkim, teoria problemów NP-zupełnych doczekała się systematycznego opracowania w postaci książki Gareya i Johnsona [4'], która jest jednocześnie największą kolekcją tych problemów.

Niestety, problem, czy $P = NP$, nadal jest otwarty, natomiast nastąpił duży postęp w dziedzinie dwóch najpopularniejszych problemów, o których w chwili pisania pracy nie było wiadomo, czy są NP- zupełne czy też należą do klasy $P$. Problemy te, to problem izomorfizmu grafów i problem programowania liniowego. W obu przypadkach wprowadzone zostały klasy problemów im równoważnych, a więc odpowiednio I-zupełne i LP-zupełne (patrz [2'] i [3'], odpowiednio). Za największe jednak osiągnięcie w tej dziedzinie, i w całej złożoności obliczeniowej ostatnich dziesięciu lat uważa się wielomianowy algorytm rozwiązywania problemów programowania liniowego podany przez Khachiyana w 1979 roku. Zatem problem programowania liniowego należy do klasy $P$! Algorytm Khachiyana, zwany algorytmem elipsoidalnym, został najpierw zaanonsowany w pracy [5'], a następnie opublikowany z pełnymi dowodami w [6']. W międzyczasie ukazała się istna lawina prac objaśniających pracę [5'] oraz informacji o tym odkryciu; patrz np. [1'] i [7'], gdzie ta ostatnia pozycja zawiera bibliografię publikacji poświęconych algorytmowi Khachiyana, które ukazały się w ciągu roku od opublikowania [5'].

[1'] B. Aspvall, R. E. Stone, *Khachiyan's linear programming algorithm*, STAN-CS-79-776, Dept. of Computer Science, Stanford University, 1979.
[2'] K. S. Booth, C. J. Colbourn, *Problems polynomially equivalent to graph isomorphism*, CS-77/04, Department of Comp. Science, University of Waterloo, 1979.
[3'] D. P. Dobkin, S. P. Reiss, *The complexity of linear programming*, Theoretical Comp. Science 11 (1980), str. 1–18.
[4'] M. R. Garey, D. S. Johnson, *Computers and intractability. A guide to the theory of NP-completeness*, Freeman, 1979.
[5'] L. G. Khachiyan, *A polynomial algorithm in linear programming*, Doklady AN USSR, 244 (1979), str. 1093–1096.
[6'] —, *A polynomial algorithm in linear programming*, Zh. Vychislitelnoi Matematiki i Matematicheskoi Fizyki 20 (1980), str. 51–68.
[7'] P. Wolfe, *A bibliography for the ellipsoid algorithm*, RC 8237, IBM Thomas J. Watson, Research Center, 1980.