2. Klasy złożoności obliczeniowej algorytmów i NP zupełność

2.1 Teoria złożoności obliczeniowej

Złożoność obliczeniową algorytmu (ang. *computational complexity*) zdefiniowaliśmy już w rozdziale 1. Podzieliliśmy złożoność obliczeniową algorytmu na czasową i przestrzenną. Z kolei złożoność czasową podzieliliśmy na pesymistyczną złożoność czasową i średnią (lub oczekiwaną złożoność czasową). Teraz zajmiemy się klasami czasowej złożoności obliczeniowej. Będziemy mieli do czynienia z klasą **P, NP, co-NP** oraz **NPC** oraz klasami probabilistycznej złożoności obliczeniowej **ZPP**, **RP** i **BPP**.

Główny cel teorii złożoności obliczeniowej to stworzenie metody, aparatu, mechanizmu umożliwiającego **klasyfikację problemów obliczeniowych (i w następnej kolejności algorytmów komputerowych)** stosownie do zasobów potrzebnych do ich rozwiązania.

Zasobami są przy tym:

1.czas obliczeń lub co na jedno wychodzi, ilość działań np.: mnożeń, które trzeba wykonać, by osiągnąć cel (na ogół koncentrujemy się na działaniach czasowo najdłuższych lub takich, których jest najwięcej – np.: w algorytmach mnożenia macierzy jest to mnożenie liczb, w algorytmach sortowania jest to porównywanie liczb).

2.ilość potrzebnej do osiągnięcia celu pamięci (ang. *storage space*)

3. Inne zasoby jakie bierze się czasem pod uwagę to: liczba dostępnych procesorów i dostępność bitów losowych (generator losowego bitu okazuje się w praktyce z uwagi na algorytmy probabilistyczne ważnym zasobem)

Głównie interesuje nas w teorii złożoności obliczeniowej czas obliczeń a dokładniej złożoność czasowa algorytmu. Złożoność przestrzenna jest równie ważna jest jednak stosunkowo prosto wymienna na złożoność czasową z uwagi na istnienie bardzo pojemnych pamięci zewnętrznych o pojemnościach często przekraczających 1 TB. W dalszym ciągu zajmujemy się wyłącznie złożonością czasową algorytmów.

Przeprowadzona klasyfikacja, ocena złożoności obliczeniowej nie powinna zależeć od konkretnego modelu obliczeń (np.: maszyna Turinga, maszyna RAM), lecz odzwierciedlać „wewnętrzną trudność problemu" i rzeczywiście tak jest złożoność jest niezależna od modelu obliczeń (czyli zastosowanego modelu maszynowego). Złożoność obliczeniowa jest w pewnym sensie parametrem algorytmu.

Najczęściej w teorii złożoności obliczeniowej wykorzystujemy dwa modele obliczeń:

1.maszynę Turinga (mówimy też o abstrakcyjnej maszynie Turinga). Istnieje kilka równoważnych odmian maszyn Turinga (np. maszyna jednotaśmowa, wielotaśmowa)

2.Sekwencyjną maszynę o dostępie swobodnym tzw. maszynę RAM, której zasadnicze cechy już opisaliśmy w rozdziale 1.

Teoria złożoności obliczeniowej (inaczej „teoria algorytmów i obliczeń”) jest ściśle związana z lingwistyką matematyczną czyli teorią automatów skończonych, gramatyk i języków.

2.2 Pojęcie problemu (problemu obliczeniowego)

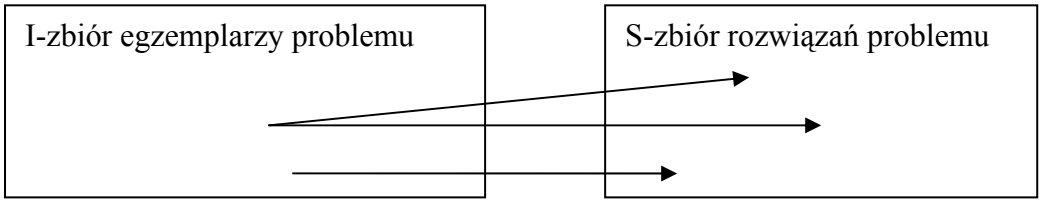
Definicja problemu obliczeniowego:
Problem obliczeniowy (lub abstrakcyjny problem obliczeniowy) Q to z definicji relacja dwuargumentowa na zbiorze I egzemplarzy problemu (I od ang. *instance,* czasami też egzemplarz problemu nazywamy w literaturze polskiej „instancją”) i zbiorze S rozwiązań problemu tzn. $Q \subseteq I \times S$, gdzie

I – zbiór egzemplarzy problemu
S – zbiór rozwiązań problemu

I – bo: instances
S – bo: solutions

Do rozwiązywania problemu obliczeniowego czyli definiowania relacji $Q \subseteq I \times S$ służy algorytm. Z punktu widzenia algorytmu zbiór egzemplarzy problemu I, to zbiór możliwych danych wejściowych a zbiór rozwiązań S jest zbiorem możliwych danych wyjściowych algorytmu.

Czasami problem obliczeniowy nazywamy problemem abstrakcyjnym lub krótko problemem.



Rys. 2.1 Ilustracja pojęcia problemu obliczeniowego

Przykład 1. Mamy dany graf nieskierowany $G = (V, E)$ bez wag na krawędziach. Rozważmy problem o nazwie SHORTEST-PATH. Problem polega na znalezieniu najkrótszej ścieżki między dwoma wierzchołkami. Egzemplarzem problemu SHORTEST-PATH jest trójka uporządkowana (graf nieskierowany, wierzchołek 1, wierzchołek 2) czyli symbolicznie (G, v_1, v_2) .

Rozwiązanie to ciąg wierzchołków grafu G, być może pusty (co oznacza, że ścieżka nie istnieje). Ponieważ najkrótsze ścieżki nie muszą być wyznaczone jednoznacznie, dany

egzemplarz problemu może mieć więcej niż jedno rozwiązanie. Ogólnie rzecz biorąc relacja $Q \subseteq I \times S$ z definicji problemu może ale nie musi być funkcją $Q : I \to S$.

Problem SHORTEST-PATH to **relacja** wiążąca każdy egzemplarz grafu i parę wierzchołków z najkrótszą ścieżką w tym grafie łączącą dwa wierzchołki.

Uwaga. Oczywiście można rozpatrywać problem SHORTEST-PATH dla grafu nieskierowanego $G = (V, E)$ z wagami na krawędziach wówczas zagadnienie staje się i ogólniejsze i bardziej praktyczne ■

Przykład 2. Analogicznie można rozpatrywać problem LONGEST-PATH, którego rozwiązaniem jest najdłuższa droga łącząca 2 wierzchołki grafu.

Przykład 3. Problem istnienia drogi (cyklu) Hamiltona w grafie skierowanym nieskierowanym lub nieskierowanym HAMILTON-CYCLE.

Cykl Hamiltona w grafie skierowanym $G = (V, E)$ to cykl prosty, który zawiera każdy wierzchołek ze zbioru wierzchołków V. Cykl Hamiltona w grafie nieskierowanym definiowany jest identycznie.

Przykład 4. Problem pokrycia wierzchołkowego grafu nieskierowanego VERTEX-COVER
Pokryciem wierzchołkowym grafu nieskierowanego $G = (V, E)$ jest taki podzbiór $V' \subseteq V$, że jeśli (u, v) jest krawędzią grafu G to $u \in V'$ lub $v \in V'$. Rozmiar pokrycia wierzchołkowego to liczba należących do niego wierzchołków. Problem pokrycia wierzchołkowego polega na znalezieniu w danym grafie nieskierowanycm pokrycia wierzchołkowego minimalnego rozmiaru.

2.3 Problemy decyzyjne

Dla uproszczenia w teorii złożoności obliczeniowej ograniczamy się do rozważania tzw. problemów decyzyjnych.

Definicja problemu decyzyjnego. Problemy decyzyjne są to takie problemy obliczeniowe, dla których rozwiązanie stanowi odpowiedź „TAK" albo „NIE" (1 albo 0).

Uwaga. Mówimy problem obliczeniowy decyzyjny lub krótko problem decyzyjny. W tym przypadku patrzymy na abstrakcyjny problem decyzyjny jak na funkcję odwzorowującą zbiór egzemplarzy problemu I w zbiór rozwiązań $S = \{0,1\}$.

Przykład. Problem decyzyjny PATH nawiązujący do problemu SHORTEST-PATH.

Dany jest graf nieskierowany $G = (V, E)$, dwa wierzchołki $u, v \in V$ i nieujemna liczba całkowita k. Czy istnieje ścieżka o długości co najwyżej k w grafie G łącząca wierzchołki u i v.

Zatem egzemplarz problemu decyzyjnego PATH wygląda tak: $i = (G, u, v, k)$

PATH $(i) = 1$ (czyli TAK) jeśli najkrótsza ścieżka od u do v ma długość co najwyżej k.

PATH $(i) = 0$ (czyli NIE) w przeciwnym przypadku.

■

Wiele problemów abstrakcyjnych to nie problemy decyzyjne, ale raczej optymalizacyjne (trzeba „coś" zminimalizować lub maksymalizować) ale łatwo problemy optymalizacyjne przeformułować i przerobić na decyzyjne, najczęściej przez nałożenia ograniczenia na optymalizowaną wielkość.

Przykładem jest zastąpienie problemu SHORTEST-PATH problemem PATH przez narzucanie ograniczenia k.

Jeśli zastąpimy problem optymalizacyjny odpowiadającym jej problemem decyzyjnym, to możemy wyniki teorii złożoności obliczeniowej koncentrującej się na problemach decyzyjnych wykorzystać do oceny złożoności problemu wyjściowego o charakterze optymalizacyjnym.

Oczywiście, jeśli potrafimy szybko rozwiązać problem optymalizacyjny, to potrafimy również szybko rozwiązać odpowiadający mu problem decyzyjny. Porównujemy bowiem wartość otrzymaną jako rozwiązanie problemu optymalizacyjnego z ograniczeniem stanowiącym jedną z danych wejściowych.

Jednocześnie, jeśli stwierdzimy, że dany problem decyzyjny jest trudny obliczeniowo, to stanowi to oczywisty argument, by uznać odpowiadający mu problem optymalizacyjny za trudny obliczeniowo.

Wniosek. Ograniczenie się w teorii złożoności do problemów decyzyjnych nie jest poważnym ograniczeniem.

Warto jeszcze zauważyć, że dane wejściowe (egzemplarz problemu decyzyjnego) można utożsamiać ze słowem nad ustalonym alfabetem (np. binarnym) a więc finalnie zbiór wszystkich takich słów skojarzonych z odpowiedzią TAK jest pewnym językiem. Algorytm rozwiązujący problem decyzyjny jest w gruncie rzeczy algorytmem rozpoznającym słowa pewnego języka. Sprowadzamy w ten sposób zagadnienie badania złożoności algorytmu na ścisły grunt teorii automatów, języków i obliczeń (por. [1],[2],[7]).

2.4 Algorytmy z czasem wielomianowym

Definicja (algorytmu z czasem wielomianowym)
Algorytm z czasem wielomianowym (a dokładniej algorytm z wielomianowym czasem wykonania lub algorytm o wielomianowej złożoności czasowej ang. „*a polynomial time complexity algorithm*") to taki algorytm, którego pesymistyczna złożoność czasowa (*worst case running time*) jest postaci $O(n^k)$, gdzie n oznacza rozmiar danych wejściowych a $k \in N$ jest stałą.

Każdy algorytm, którego czas obliczeń nie może być w ten sposób przedstawiony jest nazywany **algorytmem z czasem wykładniczym** (ang. *exponential time algorithm*) lub algorytmem z wykładniczą złożonością czasową.

Intuicja. Mówiąc niezbyt dokładnie, algorytm z czasem wielomianowym jest „dobrym", efektywnym algorytmem a algorytm z wykładniczą złożonością obliczeniową jest algorytmem nieefektywnym.

Widać jednak, że nie do końca tak jest, bowiem algorytm z wielomianowym czasem wykonania $O(n^{100})$ jest raczej mniej efektywny dla małych n niż algorytm z czasem wykładniczym $O(n^{\ln\ln n})$ (raczej, bo w definicji notacji „duże O" mamy dowolnie dobierane stałe i stała kryjąca się za notacją „duże O" może być duża).

Mówimy o „klasie problemów rozwiązywalnych w czasie wielomianowym". jest to tzw. klasa **P** (od ang *polynomial*)

Warto zauważyć, że często złożoność średnia (ang. *average case complexity*) jest dla nas znacznie ważniejsza od złożoności najgorszego przypadku (ang. *wort case complexity*).

W kryptografii zależy nam np. na tym na przykład, by algorytm był prawie zawsze trudny obliczeniowo a nie tylko dla izolowanych przypadków. Oczywiście jeśli mówimy o średniej złożoności obliczeniowej wymaga to wprowadzenia rozkładu prawdopodobieństwa na danych dla ustalonego rozmiaru n. Z reguły jest to rozkład równomierny.

Z dotychczasowych rozważań wynika, że zawsze poszukujemy algorytmów o jak najmniejszej złożoności obliczeniowej czyli jak najefektywniejszych. Na ogół cieszy nas jeśli znaleźliśmy algorytm efektywny. Jednak pojawienie się efektywnego algorytmu może być kłopotliwe. W kryptografii na przykład brak efektywnych algorytmów kryptoanalizy (np.

algorytmów do faktoryzacji liczb całkowitych lub algorytmów obliczania logarytmu dyskretnego) jest warunkiem bezpieczeństwa stosowanych algorytmów kryptograficznych.

Dla uproszczenia teoria złożoności obliczeniowej ogranicza się na ogół do tzw. **problemów decyzyjnych**, to jest problemów, dla których danymi wyjściowymi jest odpowiedź TAK lub NIE. Każdy typowy **problem obliczeniowy** daje się przeformułować na **problem decyzyjny** w prosty i efektywny sposób.

Krótko: Efektywny algorytm dla problemu decyzyjnego daje efektywny algorytm dla odpowiadającego mu problemu obliczeniowego i odwrotnie.

Definicja. Klasa złożoności **P** (tzw. wielomianowa klasa złożoności) jest zbiorem wszystkich problemów decyzyjnych, które są rozwiązywalne w czasie wielomianowym.

Definicja. Klasa złożoności **NP** (**NP** to skrót od *Nondeterministic Polynomial Time*) jest zbiorem wszystkich problemów decyzyjnych, których odpowiedź TAK może być zweryfikowana w czasie wielomianowym pod warunkiem, że mamy dodatkową informację tzw. certyfikat.

Definicja.
Klasa złożoności co-NP jest zbiorem wszystkich problemów decyzyjnych, dla których odpowiedź NIE może być zweryfikowana w czasie wielomianowym, pod warunkiem, że mamy dodatkową informację tzw. certyfikat.

Uwaga. Te certyfikaty z powyższych definicji z założenia istnieją, ale to nie znaczy, że łatwo je obliczyć.

Przykład. (Przykład problemu decyzyjnego klasy NP)
Rozważamy następujący problem decyzyjny o nazwie FAKTORYZACJA. Problem polega na odpowiedzi na pytanie czy liczba naturalna n jest złożona tzn. czy istnieją takie liczby naturalne $a,b>1$, że $n=a\cdot b$.

Egzemplarz problemu: Dane WEJŚCIOWE: liczba naturalna $n \in N$
Dane WYJŚCIOWE: TAK albo NIE

Problem decyzyjny FAKTORYZACJA należy do klasy **NP**, ponieważ fakt złożoności może być zweryfikowany w czasie wielomianowym jeśli mamy dany dzielnik a liczby n, gdzie $1<a<n$. Certyfikat (czyli inaczej „świadek") to w tej sytuacji dzielnik a liczby n.

Problem decyzyjny FAKTORYZACJA należy również do klasy **co-NP**.

Nie wiadomo, czy problem decyzyjny FAKTORYZACJA należy do klasy **P**.
■

Fakt Dla klas złożoności zachodzą następujące inkluzje $\mathbf{P} \subseteq \mathbf{NP}$ oraz $\mathbf{P} \subseteq \mathbf{co-NP}$
Dowód. Zawieranie jest oczywiste ponieważ certyfikatem może być po prostu rozwiązanie problemu decyzyjnego uzyskane w czasie wielomianowym. ■

Nie wiadomo jednak czy:

P = **NP** ?

NP = **co** – **NP** ?

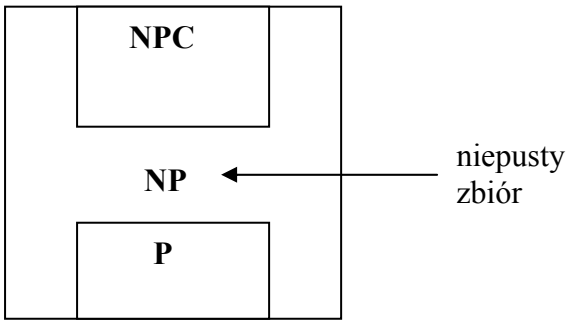
P = **NP** ∩ **co** – **NP** ?

Sądzimy, że odpowiedź na te pytania jest NIE ale nie istnieją dowody, że tak jest istotnie.

Podział na klasy złożoności obliczeniowej **P**, **NP**, **co-NP** nie wyczerpuje wszystkich rozważanych klas złożoności mamy np. klasę „ponad **NP**" i kilka ważnych probabilistycznych klas złożoności.

Fakt Jeśli **P** ≠ **NP** to istnieje problem (język), który nie należy ani do klasy **P** ani **NPC**.

Zatem jeśli **P** ≠ **NP** to na pewno sytuacja jest na pewno taka



Rys. 2.5 Wzajemne relacje pomiędzy klasami złożoności **P**, **NPC** i **NP**

2. 5 Redukowalność i problemy NP -zupełne

Bardzo użytecznym w teorii złożoności pojęciem, jest pojecie **redukowalności**. Redukowalność umożliwia porównywanie, ocenę względnej trudności problemów.

Definicja. Niech L_1 i L_2 będą dwoma problemami decyzyjnymi. Mówimy, że L_1 wielomianowo redukuje się do L_2 (co zapisujemy jako $L_1 \leq_p L_2$) jeśli istnieje algorytm, który rozwiązuje L_1 wykorzystujący jako procedurę algorytm rozwiązywania L_2 i który działa w czasie wielomianowym.

Intuicja: Jeśli $L_1 \leq_p L_2$, to L_2 jest co najmniej tak trudny jak L_1. Można też powiedzieć, że L_1 nie jest trudniejszy niż L_2.

Definicja:
Jeśli L_1, L_2 są dwoma problemami decyzyjnymi oraz $L_1 \leq_p L_2$ i $L_2 \leq_p L_1$, to L_1, L_2 nazywamy **obliczeniowo równoważnymi**.

Fakt
Niech L_1, L_2, L_3 będą trzema problemami decyzyjnymi, wówczas :

1) Jeśli $L_1 \leq_p L_2$ i $L_2 \leq_p L_3$ to $L_1 \leq_p L_3$ (przechodniość)
2) Jeśli $L_1 \leq_p L_2$ i $L_2 \in P$ to $L_1 \in P$

Definicja (problemu NP-zupełnego)
Problem decyzyjny L jest NP-zupełny, jeśli $L \in \mathbf{NP}$ i $L_1 \leq_P L$ dla każdego $L_1 \in \mathbf{NP}$.

Uwaga. Klasę wszystkich problemów NP-zupełnych oznaczamy symbolem **NPC** (od ang. *Nondeterministic Polynomial Complete*)

Problemy z klasy **NPC** czyli problemy NP-zupełne są tymi najtrudniejszymi problemami w klasie **NP** w tym sensie, że są one co najmniej tak trudne jak każdy inny problem w klasie **NP**. Istnieje bardzo dużo problemów z teorii liczb, analizy kombinatorycznej czy logiki, o których wiadomo, że są NP-zupełne.

Fakt Niech L_1 i L_2 będą dwoma problemami decyzyjnymi:

1) Jeśli L_1 jest NP-zupełny i $L_1 \in \mathbf{P}$ to $\mathbf{P} = \mathbf{NP}$
2) Jeśli $L_1 \in \mathbf{NP}$, L_2 jest NP-zupełny i $L_2 \leq_p L_1$ to L_1 jest takie jak NP-zupełny
3) Jeśli L_1 jest NP-zupełny i $L_1 \in \mathbf{co-NP}$ to $\mathbf{NP} = \mathbf{co-NP}$.

Uwaga. Z 1) wynika, że gdyby dało się znaleźć algorytm z czasem wielomianowym dla choćby jednego NP zupełnego problemu, to wówczas **P=NP** co wydaje się mało prawdopodobne.

Typowe postępowanie przy dowodzeniu NP zupełności problemu decyzyjnego jest takie:

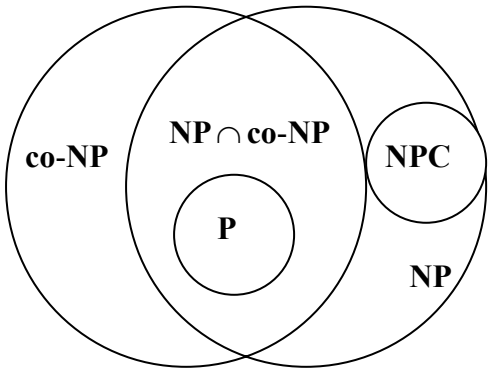
1. dowieść, że $L_1 \in NP$
2. wybrać problem L_2, o którym wiadomo, że jest NP zupełny.
3. dowieść, że $L_2 \leq_p L_1$

Definicja. Problem jest NP-trudny (ang. NP hard) jeśli istnieje pewien NP-zupełny problem decyzyjny, który w czasie wielomianowym redukuje się do niego.

($L_1 \leq_p L_2$ czytamy L_1 w czasie wielomianowym redukuje się do L_2).

Uwaga. Warto zauważyć, że klasyfikacja NP- hard nie ogranicza się tylko do problemów decyzyjnych. Oczywiście problem NP-zupełny jest również NP trudny.

Często używamy określenia „problem trudny obliczeniowo". Określenie to oznacza przede wszystkim brak efektywnych algorytmów rozwiązywania danego problemu ale często mamy też na myśli problem NP-trudny.



Rys. 2.1 Przypuszczalne zależności pomiędzy klasami złożoności **P, NP, co-NP i NPC.**

2.7 Przykłady problemów NP-zupełnych

Przykład 1 Problem plecakowy (problem sumy, problem sumy podzbioru, problem upakowania), (ang. *knapsack problem* lub *subset sum problem*).

Problem plecakowy jest następujący. Mamy zbiór n liczb naturalnych $\{a_1, a_2, ...a_n\}$ tzw. plecak i liczbe naturalną s. Stwierdzić, czy istnieje taki podzbiór tego zbioru, który ma tę własność, że liczby wchodzące do tego podzbioru dają w sumie s tzn.

$$\sum_{i \in K} a_i = s$$

gdzie $K \subseteq < 1, n >$.

Rozwiązanie problemu plecakowego może istnieć dokładnie jedno, możemy mieć kilka rozwiązań lub może nie być żadnego rozwiązania.

Jest to problem NP zupełny, czyli problem klasy **NPC** ale dla pewnej klasy plecaków tzw. plecaków superrosnących można znaleźć rozwiązanie tego problemu w czasie liniowym.

Można jednak nie pytać czy istnieje odpowiedni podzbiór, ale zapytać jaki to jest podzbiór. Problem przestaje być już wtedy problemem decyzyjnym. Jest to również problem NP-trudny.

Istnieje cały szereg algorytmów szyfrowania i deszyfrowania opartych na problemie plecakowym jest to m.in. algorytm Merklego-Hellmanna i algorytm Chora-Rivesta.. Bezpieczeństwo kryptosystemów tego typu wynika z faktu, że problem plecakowy jest NP-zupełny. ■

Przykład 2
Problem cyklu Hamiltona. Cykl Hamiltona w grafie skierowanym $G = (V, E)$ to cykl prosty, który zawiera każdy wierzchołek ze zbioru wierzchołków V. Cykl Hamiltona w grafie nieskierowanym definiowany jest identycznie.

Problem stwierdzenia czy graf skierowany ma cykl Hamiltona jest problemem NP zupełnym. Podobnie problem stwierdzenia czy graf nieskierowany ma cykl Hamiltona jest problemem NP - zupełnym.

Przykład 3
Problem pokrycia wierzchołkowego grafu nieskierowanego VERTEX-COVER jest problemem NP.-zupełnym.

Przykład 4
Problem spełnialności układów logicznych CIRCUIT-SAT jak również problem spełnialności funkcji boolowskiej SAT są problemami NP-zupełnymi.

Przykład 5
Znanym przykładem problemu NP.-zupełnego jest znany z badań operacyjnych problem komwojażera

2. 7 Klasy złożoności algorytmów probabilistycznych

Algorytmy rozważane do tej pory były głównie **deterministyczne**. Algorytm deterministyczny charakteryzuje się tym, ze sposób jego wykonania dla tych samych danych wejściowych jest zawsze taki sam.

Inaczej jest w **algorytmach probabilistycznych**, (*probabilistic algorithms* lub *randomized algorithms*), które podejmują w pewnych punktach algorytmu **losowe decyzje**. W konsekwencji „ścieżka wykonania" algorytmu może być dla tych samych danych inna. Decyzje losowe są podejmowane za pomocą generatora liczb losowych.

Istnieje wiele problemów obliczeniowych, dla których algorytmy probabilistyczne są znacznie bardziej efektywne niż algorytmy deterministyczne (i to zarówno jeśli chodzi o **złożoność czasową**, jak i złożoność przestrzenną).

Algorytmy probabilistyczne dla problemów decyzyjnych mogą być sklasyfikowane zależnie od tego z jakim prawdopodobieństwem zwracają poprawna odpowiedź.

Definicja. Niech A będzie algorytmem probabilistycznym dla problemu decyzyjnego L a i oznacza dowolny egzemplarz (instancję) problemu decyzyjnego L.

1.Algorytm probabilistyczny A ma 0 stronny błąd jeśli:

$P(A$ odpowiada TAK | prawdziwa odpowiedź dla i jest TAK$)=1$

oraz

$P(A$ odpowiada TAK| prawdziwa odpowiedź dla i jest NIE$)=0$

Praktycznie więc algorytm działa bezbłędnie poza być może zbiorem miary 0.

2. Algorytm probabilistyczny A ma 1 stronny błąd jeśli:

$P(A$ odpowiada TAK |prawdziwa odpowiedź dla i jest TAK$) \geq \varepsilon$
oraz

P(A odpowiada TAK | prawdziwa odpowiedź dla i jest NIE$)=0$

Uwaga. Liczba ε jest tutaj wybrana arbitralnie i może być dowolną liczbą z przedziału (0,1]

Algorytm A niezbyt chętnie (a dokładniej z prawdopodobieństwem $\geq \varepsilon$) udziela odpowiedzi twierdzącej TAK ale nie kłamie. Jeśli nie udzielił odpowiedzi TAK za pierwszym razem to ponawiamy, wykonanie algorytmu czyli używając terminologii probabilistycznej dokonujemy nowego doświadczenia lub całej serii k niezależnych doświadczeń jak w ciągu doświadczeń Bernoulliego.

Jeśli prawdziwa odpowiedź dla instancji *i* jest TAK a algorytm stale nie odpowiada TAK to przy $k \to +\infty$ prawdopodobieństwo tego zdarzenia dąży do 0.

Rozsądnie jest więc przyjąć w tej sytuacji, że prawdziwa odpowiedź dla instancji *i* jest NIE podając prawdopodobieństwo ewentualnej pomyłki. Z reguły *k* dobieramy tak duże by prawdopodobieństwo pomyłki było bardzo małe np. 10^{-100}.

3. Algorytm *A* ma 2 stronny błąd jeśli:

$P(A \text{ odpowiada TAK } | \text{prawdziwa odpowiedź dla } i \text{ jest TAK}) \geq \frac{1}{2} + \varepsilon$

oraz

$P(A \text{ odpowiada TAK } | \text{prawdziwa odpowiedź dla } i \text{ jest NIE}) \leq \frac{1}{2} - \varepsilon$

gdzie $\varepsilon \in (0, \frac{1}{2}]$

Algorytm *A* co prawda trochę kłamie ale dokonując całej serii *k* niezależnych doświadczeń (polegających na wykonaniu algorytmu *A*) jak w ciągu doświadczeń Bernoulliego i obliczając wartość średnią liczby odpowiedzi TAK jesteśmy w stanie stwierdzić: czy prawdziwa odpowiedź dla *i* jest TAK czy prawdziwa odpowiedź dla *i* jest NIE.

Możemy się pomylić ale prawdopodobieństwo pomyłki możemy dowolnie zminimalizować dobierając odpowiednią liczbę doświadczeń czyli odpowiednie *k*.

Definicja (oczekiwanego czasu wykonania algorytmu probabilistycznego) (ang. expected running time).

Oczekiwany czas wykonania algorytmu probabilistycznego jest górnym ograniczeniem na wartość oczekiwaną czasu wykonania dla każdych konkretnych danych wejściowych algorytmu. Uśrednianie przy tym bierzemy po wszystkich danych wyjściowych generatora liczb losowych. Oczekiwany czas wykonania algorytmu probabilistycznego wyrażamy jako funkcję rozmiaru danych wejściowych *n*.

Zdefiniujemy teraz 3 ważne klasy złożoności dla algorytmów probabilistycznych: klasę **ZPP, RP** i **BP.**

Definicja (probabilistycznych klas złożoności)

1.Klasa złożoności ZPP (ang. *zero-sided probabilistic polynomial time*) to zbiór wszystkich problemów decyzyjnych, dla których istnieje algorytm probabilistyczny z 0-stronnym błędem, który ma wielomianowy oczekiwany czas wykonania.

2.Klasa złożoności RP (ang. *randomized polynomial time*) to zbiór wszystkich problemów decyzyjnych, dla których istnieje algorytm probabilistyczny z 1-stronnym błędem wykonujący się w (najgorszym przypadku) w czasie wielomianowym.

3.Klasa złożoności BPP (ang. *bounded error probabilistic polynomial time*) to zbiór wszystkich problemów decyzyjnych, dla których istnieje algorytm probabilistyczny z 2-stronnym błędem, który wykonuje się w (najgorszym przypadku) w czasie wielomianowym.

Fakt Zachodzą następujące inkluzje pomiędzy probabilistycznymi klasami złożoności obliczeniowej:

$\mathbf{P} \subseteq \mathbf{ZPP} \subseteq \mathbf{RP} \subseteq \mathbf{BPP}$

$\mathbf{RP} \subseteq \mathbf{NP}$

Uwaga. Nie wiemy czy $\mathbf{BPP} \subseteq \mathbf{NP}$?.

Zadania

Zadanie1
Oszacować złożoność obliczeniową (pesymistyczną i średnią) najprostszego algorytmu Euklidesa (algorytmu bez dzielenia) i porównać go ze złożonością obliczeniową typowego algorytmu Euklidesa z dzieleniem. Oba algorytmy podane są poniżej. Sporządzić wykres czasu wykonania algorytmu w funkcji rosnącego rozmiaru danych (w bitach). Czy problem obliczania NWD jest NP-zupełny, czy należy do klasy NP, co-NP a może P.

a)

Algorytm Algorytm Euklidesa z odejmowaniem

Dane wejściowe $a,b \in N$

Dane wyjściowe $d = NWD(a,b)$

---

begin
początek: **if** $a = b$ **then begin** $d := a$; **goto** *stop* **end**
if $a > b$ **then** $a := a - b$ **else** $b := b - a$
goto *początek* ;
stop: *write*(‘*NWD* = ’ , *d*)

end

Fakt wykorzystywany w algorytmie: dla każdego $a,b \in Z$ $NWD(a,b) = NWD(a-b,b)$

b)

Dane wejściowe $a,b \in N$

Dane wyjściowe $d = NWD(a,b)$

begin

początek: oblicz resztę r z dzielenia *a* przez *b*

if $r = 0$ **then begin** $d := b$; **goto** *stop* **end**
else begin $a := b$; $b := r$ **end**;

goto *początek*;

stop: write(NWD=*d*)

end

Fakt wykorzystywany w algorytmie: dla każdego $a,b \in Z$ $NWD(a,b) = NWD(b,r)$

Zadanie 2
Oszacować złożoność obliczeniową (pesymistyczną i średnią) najprostszego algorytmu Euklidesa (algorytmu bez dzielenia) i porównać go ze złożonością obliczeniową algorytmu Steina (binarny algorytm Euklidesa). Oba algorytmy podane są poniżej. Sporządzić wykres czasu wykonania algorytmu w funkcji rosnącego rozmiaru danych (w bitach). Czy problem obliczania NWD jest NP-zupełny, czy należy do klasy NP, co-NP a może P.

a) Dane wejściowe $a,b \in N$
Dane wyjściowe $d = NWD(a,b)$

begin

początek: **if** $a = b$ **then begin** $d := a$; **goto** *stop* **end**;
if $a > b$ **then** $a := a - b$ **else** $b := b - a$;
goto *początek*
stop: *write*(NWD=*d*)

end

Zasada wykorzystywana: dla każdego $a,b \in Z$ $NWD(a,b) = NWD(a-b,b)$

b) algorytm Steina obliczania NWD

Dane wejściowe $a,b \in N$

Dane wyjściowe $d = NWD(a,b)$

begin

n: =0;

pętla1: **if** (*a* i *b* parzyste) **then**

begin „przesuń *a* i *b* o 1 pozycję w prawo”; $n := n+1$; **goto** *pętla1* **end**;

pętla2: **if** (*a* parzyste) **then begin** „przesuń *a* o 1 pozycję w prawo”; **goto** *pętla2* **end**;

pętla3: **if** (*b* parzyste) **then begin** „przesuń *b* o 1 pozycję w prawo”; **goto** *pętla3* **end**;

if $a = b$ **then begin** $d := 2^n a$; **goto** *stop* **end**;

if $a > b$ **then** $a := a - b$ **else** $b := b - a$

goto *pętla2*;

stop: *write*(‘NWD=’, *d*)

end

Fakty na, które wykorzystuje algorytm Steina są następujące:

Jeśli $a = 2^k c$, $b = 2^k d$ gdzie $a,b,c,d,k \in N$ to $NWD(a,b) = 2^k NWD(c,d)$

Jeśli $a \in N$ jest liczbą parzystą a $b \in N$ nieparzystą to $NWD(a,b) = NWD(a/2,b)$

Jeśli $a,b \in N$ to $NWD(a,b) = NWD(a,b-a) = NWD(b,a-b)$

Zadanie 3
Oszacować złożoność obliczeniową poniżej podanego tzw. schematu Hornera (jest to powszechnie używany algorytm do szybkiego obliczania wartości wielomianu w punkcie). Do jakiej klasy złożoności zaliczyć można schemat Hornera.

Wykorzystujemy w schemacie Hornera następujący pomysł prawdziwy dla dowolnego wielomianu o współczynnikach w dowolnym ustalonym ciele ***K***.

$$a_n x^n + a_{n-1} x^{n-1} + ... + a_1 x + a_0 = ((...((a_{n-1} x + a_{n-1}) x + a_{n-2}) x + ...) x + a_1) x + a_0$$

Uwaga. Znaki mnożenia i dodawania w powyższym wzorze trzeba interpretować zależnie od tego z jakiego ciała ***K*** pochodzą współczynniki wielomianu. Czy klasa złożoności algorytmu zależy od ciała ***K***.

Sporządzić wykres czasu wykonania algorytmu w funkcji rosnącego rozmiaru danych *n* dla przypadku, gdy współczynniki wielomianu są liczbami zespolonymi.

DANE WEJŚCIOWE : współczynniki wielomianu $a_n, a_{n-1}, ..., a_1, a_0 \in C$, liczba zespolona z
DANE WYJŚCIOWE: wartość wielomianu w punkcie *z* czyli $a_n z^n + a_{n-1} z^{n-1} + ... + a_1 z + a_0$

begin
$y := A[n]$;
for $i := n$ -1 **step** -1 **until** 0 **do** $y := y * x + A[i]$;
write('wartość wielomianu w punkcie $x = z$ jest równa', y)
end

Uwaga. Zrobić jeszcze raz to samo co wyżej dla ciała skończonego Z_p, gdzie *p* jest liczbą pierwszą (mnożenie i dodawanie będą teraz inne).

Zadanie 4
Oszacować złożoność obliczeniową algorytmu szybkiego podnoszenia do potęgi modulo *m* $x^n \pmod m$ .Co przyjąć co rozmiar danych wejściowych? Istotą algorytmu jest wykorzystanie równości

$$x^n = a^{a_k 2^k + a_{k-1} 2^{k-1} + ... + a_1 2 + a_0} = (a^{2^k})^{a_k} (a^{2^{k-1}})^{a_{k-1}} (a^{2^{k-2}})^{a_{k-2}} ... (a^{2^0})^{a_0}$$

Algorytm: Algorytm szybkiego podnoszenia do potęgi

Dane WEJŚCIOWE : $x \in Z_m$ liczba podnoszona do potęgi czyli podstawa potęgi,
$m \geq 2$ ilość elementów pierścienia Z_m
$n \in N \cup \{0\}$, wykładnik potęgi
Dane WYJŚCIOWE: $y = x^n \pmod m$

---

begin

$y := 1$; $kw := x$;

początek: **if** (n=0) **then goto** koniec;

if (bit LSB słowa n jest =1) **then** $y := y * kw \pmod m$;

$kw := kw * kw \pmod m$;

shift n o 1 bit w prawo;

goto początek;

koniec: *write*(wartość $x^n \pmod m$ jest równa y)

end

Literatura

[11] C.H. Popadimitriov; Złożoność obliczeniowa; WNT, Warszawa 2002.

[2] T. H. Cormen, C.E. Leiserson, R.L. Rivest, C.Stein ; Wprowadzenie do algorytmów; WNT, Warszawa 2004.

[3] A.Kościelski; Teoria obliczeń; Wydawnictwo Uniwersytetu Wrocławskiego; Wrocław 1997.

[4] J. Błażewicz; Złożoność obliczeniowa problemów kombinatorycznych; WNT, Warszawa 1988.

[5] A. Menezes, P. Oorschot, S. Vanstone; Handbook of Applled Cryptography; CRC Press Inc., 1997. (treść książki jest zamieszczona na stronie: http://cacr.math.uwaterloo.ca/hac)

[6] V.V.Vaziriani; Algorytmy aproksymacyjne; WNT, Warszawa 2004.

[7] J.E.Hopcroft, J.D.Ullman; Wprowdzenie do teorii automatów języków i obliczeń; PWN, Warszawa 1994.