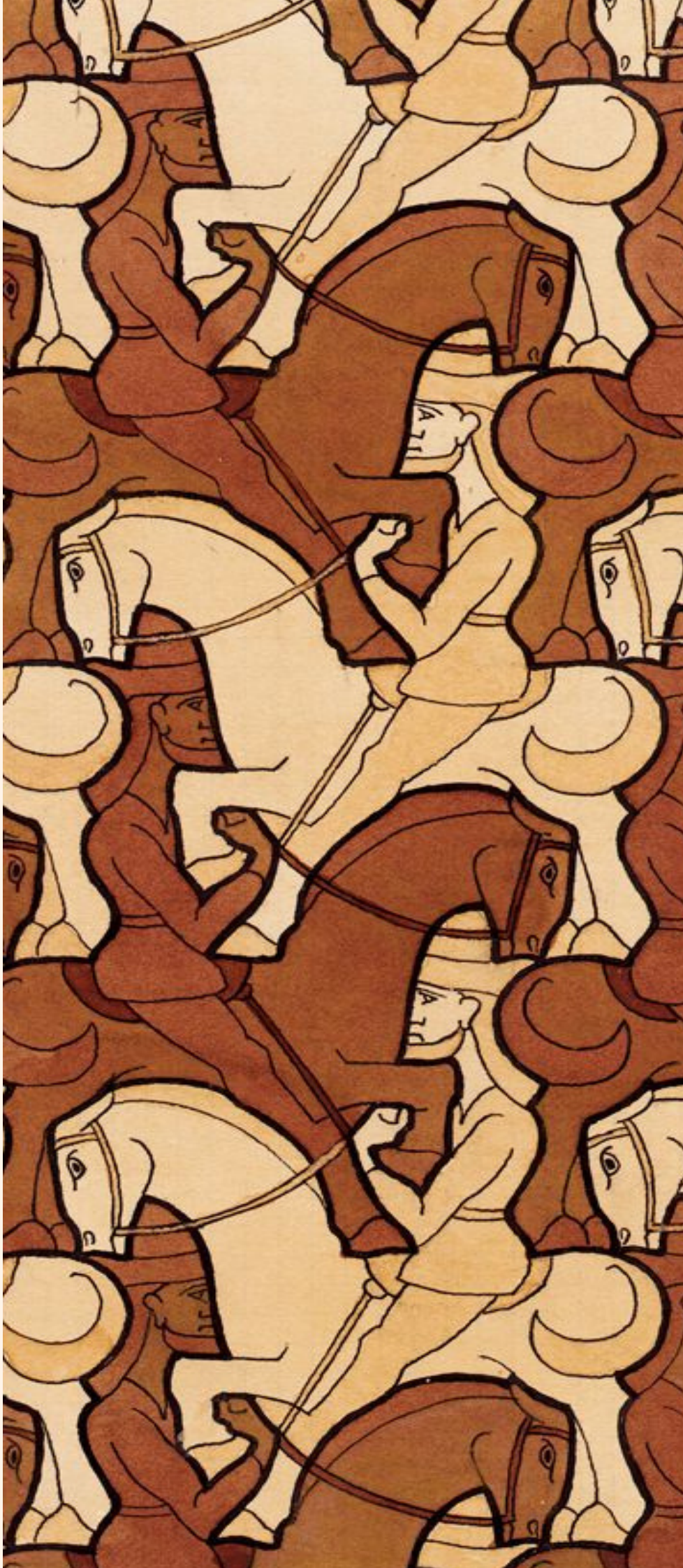
Podstawy Analizy Algorytmów

Jakub Bedra

Zbiór zadań z odpowiedziami



Zbiór dedykowany wszystkim studentom poległym przez brak materiałów do nauki tego przedmiotu. [*]

Spis treści

Część 1. Złożoności

1.1 Złożoności – Czas działania

1.1.1 Oblicz czas działania pętli:

a)

```
for i=1 to n do;
```

b)

```
for i=1 to √n do
        for j=1 to n do;
```

c)

```
for i=1 to √n do
        for j=i to n do k=k;
```

d)

```
 i = j = 1;
 while j<n do begin
        i = i + 2;
        j = j + i;
 end;
```

e)

```
k = 0;
for i=1 to n do
        for j=1 to n do k=k+1;
```

f)

```
k = 0;
for i=1 to n*n do
        for j=i to n do k = k + 1;
```

g)

```
while i<n do i = i+i;
```

h)

```
for i=1 to n^3 mod 9 do
    if i<n^n then for j=1 to n^2 do;
```

i)

```
for i=1 to n^2 do
    for j=1 to n do
    begin
        k=1;
        while k<n do k = k + k;
    end;
```

1.1 Złożoności – Czas działania

1.1.2 Zaprojektuj algorytm wykonujący sumę:

1 +

1 + 2 +

1 + 2 + 3 +

…

W czasie:

a) $O(n^2)$ b) O(n) c) O(1)

1.1.3 Podaj wzór który najściślej określa liczbę kroków następującego algorytmu:

```
for i=1 to n^2 do
    for j=i to n do
        begin
            k=1;
            while k<n do k = k + k;
        end;
```

1.1.4 Podaj liczbę kroków następującego algorytmu:

```
for i=1 to n^2 do
   if i^2 < n^3 then for j=1 to n^2 do
        for k=1 to n^2 % n do;
```

1.1 Złożoności – Czas działania

1.1.5 Oblicz czas działania następującej procedury:

```
procedurę zagadka(n)
begin
    read(i);
    while i<n do i=i*i;
end;
```

dla: a) i=0 b) i=1 c) i=2 d) i=3

1.1.6 Zaprojektuj algorytm wykonujący następującą sumę:

$$\sum_{i=1}^{n} \sum_{j=1}^{n} i*j$$

W czasie:

a) $O(n^3)$ b) $O(n^2)$ c) $O(n)$ d) $O(1)$

1.1.7 Co robi poniższa procedura?

```
if n=3 then return(6);
else  if n<= 4 then return(n);
else return(0);
```

1.1.8 Podaj liczbę kroków poniższego programu:

```
for i=1 to n^4(mod 9) do
    if i<n^2 then for j=1 to n^2 do;
```

1.2 Złożoności – Równania rekurencyjne

1.2.1 Oblicz czas działania funkcji:

a)

```
function A(n)
begin
  if n<2 then A = 1;
  else A = A(n/2) + A(n/2);
end;
```

b)

```
function A(n)
begin
 for i=1 to n do;
 if n<2 then A=1;
 else A = 2*A(n/2) + A(n/4) + 2*n*n;
end;
```

c)

```
procedure Hanoi(A, B, C, n)
begin
   if n=1 then przenieś(A, C);
   else
       Hanoi(A, C, B, n-1);
       Przenieś(A, C);
       Hanoi(B, A, C, n-1);
   end;
end;
```

1.2.2 Podaj czas działania poniższych funkcji:

a)

```
function A(n)
begin
   if n=1 then return 1;
   else return A(A(n-1));
end;
```

b)

```
function B(n)
begin
    if n=1 then return 1;
    else return B(B(n-1)) + 1;
end;
```

1.2 Złożoności – Równania rekurencyjne

1.2.3 Podaj pesymistyczny i optymistyczny czas działania poniższej procedury dla j=2:

```
procedure zagadka(n, j)
begin
    for i=1 to n*n % 4 do begin
        while i<=6 do begin
            i := i+i;
          zagadka(n-j, j);
        end;
    end;
end;
```

1.2.4 Oblicz czas działania funkcji:

```
function X(n)
begin
   for i=1 to n do;
   if n>2 then return 2*X(n/2) – X(n/4) + n;
end;
```

1.2.5 Oblicz czas działania funkcji:

```
function zagadka (n)
begin
   if n=1 then return 1;
   else return zagadka(zagadka(n-1)) + 1;
end;
```

1.2 Złożoności – Równania rekurencyjne

1.2.6 Podaj pesymistyczną i optymistyczną liczbę kroków poniższej procedury dla wywołania QS(1, n):

```
procedure QS(l, r)
begin
   if l<r then
   begin
      i := PodzielTablice(l, r);       //O(n)
      QS(l, i-1);
      QS(i, r);
   end;
end;
```

1.2.7 Co oblicza dana funkcja? Podaj jej czas działania:

```
function xyz(n)
begin
   for i=1 to 2n^2 do;
   if n=1 then return (2);
   else return (2xyz( xyz(n/2)/2 – xyz(n/2) + xyz(n/2) ));
end;
```

1.2.8 Rozwiąż podane równania rekurencyjne:

a) $T(k) = T(k-1) * T(k-1)$

b) $T(n) = 2T(\lfloor sqrt(n) \rfloor) + \ln n$

c) $T(n) = 2T(n/2) + log\{2\}(n)$

d) $T(n) = T(n-1) + T(n-2) + \ldots + T(1)$

1.2 Złożoności – Równania rekurencyjne

1.2.9 Co robi poniższa funkcja? Ile mnożeń wykonuje dla n=15? Podaj inny sposbób obliczenia zagadka(a, 15), wymagający tylko:
a) 5 mnożeń b) 4 mnożeń i 1 dzielenia

```
function zagadka(a, n)
begin
   if n=0 then return 1;
   else begin
      half = zagadka(a, n/2);
      half = half*half;
      if odd(n) then half = half * a;
      return half;
   end
end
```

1.2.10 Wyznacz czas działania pętli za pomocą równań rekurencyjnych.

```
procedure zagadka(n)
begin
   i := 2;
   while i<n do i = i * i;
end;
```

1.3 Złożoności - Tempo wzrostu i złożoność pamięciowa

1.3.1 Oblicz tempo wzrostu funkcji:

```
function X(n)
begin
   if n>2 then return 2*X(n/2) + X(n/4) + n*n;
end;
```

1.3.2 Na płaszczyźnie danych jest n okręgów. Niech T(n) stanowi maksymalną liczbę obszarów, na które dzielą one płaszczyznę. Na przykład T(1)=2, T(2)=4, T(3)=8. Wiadome jest, że T(n) = aT(n-1)+bn-b. Ustal wartości a i b. Oszacuj tempo wzrostu T(n).
Wskazówka: Oblicz najpierw T(4).

1.3.3 Na płaszczyznę rzucono n prostych. Niech R(n) oznacza największą możliwą liczbę regionów (ograniczonych lub nie) powstałych w ten sposób, np. R(0)=1, R(1)=2. Wiadomo, że R(n) = xR(n-1) + yn. Ustal wartość x i y. Oszacuj tempo wzrostu R(n). Podaj R(4).

1.3.4 Dana jest procedura jak niżej. Oszacuj jej złożoność pamięciową oraz tempo wzrostu.

```
procedure ab(n)
begin
   for i=1 to 2*n mod 9 do;
   if n<=2 then return(1);
   else return(8*ab(n/2)-2*ab(n/2 - 1)+ n*n);
end
```

1.3 Złożoności - Tempo wzrostu i złożoność pamięciowa

1.3.5 Dana jest procedura jak niżej. Oszacuj jej złożoność pamięciową oraz tempo wzrostu.

```
procedure X(n)
begin
   for i=1 to n^2 do;
   if n=1 then return 1;
   else return X(n-1) + X(n-2) + n^2;
end
```

1.3.6 Dana jest procedura sortująca jak niżej. Zakładając, że $\log_{1,5}3 = 2,7$, oszacuj jej złożoność czasową i pamięciową, jeśli została ona wywołana za pomocą: sortuj(tab, 1, n). Czy działa ona w miejscu?

```
procedure sortuj(tablica, lewy, prawy)
begin
   if tab[lewy] > tab[prawy] then zamien(tab[lewy], tab[prawy]);
   if prawy-lewy > 1 then k = (prawy-lewy+1)/3;
   sortuj(tab, lewy, prawy-k);
   sortuj(tab, lewy+k, prawy);
   sortuj(tab, lewy, prawy-k);
end
```

1.3.7 Oszacuj od dołu i od góry tempo wzrostu poniższej procedury:

```
procedure zagadka(n)
begin
   for i:=1 to 2n^2 do;
   if n<=2 then return 1;
   else return 8*zagadka(n/2)-zagadka(n/4);
end
```

1.4 Złożoności – Złożoność, a liczba kroków

1.4.1 Podaj złożoność czasową oraz czas działania procedury:

```
procedure Fib(n)
begin
   if n=1 or n=2 then return 1;
   else return Fib(n-1) + Fib(n-2);
end
```

oraz procedury hybrydowej:

```
procedure hybryda(n)
begin
   if n<9 then
   begin
      if n<=2 then return 1;
      else return hybryda(n-1) + hybryda(n-2);
   end
   else begin
      a = b = 1;
      for i=3 to n do begin
         b = b + a;
         a = b - a;
      end
      return b;
   end
end
```

1.4 Złożoności – Złożoność, a liczba kroków

1.4.2 Co oblicza podana funkcja? Podaj jej liczbę kroków oraz złożoność czasową.

```
function PR(x, n)
begin
   if n=0 then return 1;
   if n(mod2)=0 then return PR(x, n/2) * PR(x, n/2);
   else return PR(x, n/2) * PR(x, n/2) * x;
end
```

Jak zmodyfikować tą funkcję aby uzyskać lepszą złożoność?

1.4.3 Oszacuj liczbę kroków poniższej procedury. Rozważ 3 przypadki: i=0, i=1, i=2. Podaj oszacowanie złożoności obliczeniowej tej procedury. Złożoność obliczeniowa jest: subliniowa, liniowa, quasiliniowa, wielomianowa, superwielomianowa, wykładnicza, superwykładnicza (właściwe podkreśl).

```
procedure zagadka(n)
begin
   read(i);
   while i<n do i=i*i;
end
```

1.4.4 Oszacuj liczbę kroków i złożoność obliczeniową funkcji:

```
function A(n)
begin
   if n<=2 then return 1;
   else return A(⌈n/4⌉)+ A(⌊n/4⌋) + 2*n*n;
end
```

1.4 Złożoności – Złożoność, a liczba kroków

1.4.5 Oblicz oczekiwaną liczbę kroków oraz oczekiwaną złożoność obliczeniową poniższej procedury:

```
procedure random(n)
begin
   wylosuj l z przedziału (0, 1);
   if l * n^3 < 1 then for i=1 to n^9 do;
   else for i=1 to n do;
end
```

Co zrobić, aby złożoność obliczeniowa była taka sama jak liczba kroków?

1.4.6 Dany jest algorytm random jak następuje:

```
algorithm random(n, A[1..n])
begin
   wylosuj liczbę ze zbioru {1,2,..,n^2};
   if wylosowana liczba = 1 then for i=1 to n^2 do;
   else for i=1 to n do;
end
```

1. Jaka jest pesymistyczna złożoność obliczeniowa random?
2. Jaka jest oczekiwana złożoność obliczeniowa random?
3. Jaka jest optymistyczna złożoność obliczeniowa random?

1.5 Złożoności – Struktury danych

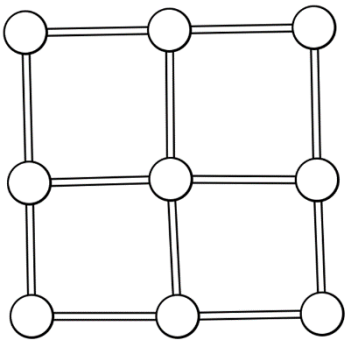
1.5.1 Mnożenie dwóch macierzy 5x5 według Makarova zajmuje 100 mnożeń.

a) Jaka jest złożoność obliczeniowa metody Makarova?
b) Czy jest ona wyższa od złożoności metody Strassena?
c) Która metoda jest lepsza do pomnożenia dwóch macierzy 25x25?
d) Załóżmy, że pomnożenie 2 macierzy 5x5 metodą Makarova zajmuje 0,1s. Ile minut trwa mnożenie dwóch macierzy 125x125?

1.5.2 Programista chce wykonać operację mnożenia trzech macierzy M_1 x M_2 x M_3 o rozmiarach odpowiednio: p x q, q x r, r x s. Dla jakiego przypadku oblicznie (M_1 x M_2) x M_3 będzie szybsze od M_1 x (M_2 x M_3)?

1.5.3 Dla pewnego problemu teoriografowego znanych jest 8 algorytmów o złożonościach obliczeniowych: O(sqrt(m)), O(m/n), $O(n^2 \Delta \log m)$, $O(n \Delta m^2)$, $O(m^4/\Delta)$, $O(n^3 \text{sqrt}(m))$, $O(n^m)$, $O(m^n)$

Uszereguj te złożoności rosnąco dla krat kwadratowych.



Przykład kraty kwadratowej.

1.5.4 Dla pewnego problemu teoriografowego znanych jest 8 algorytmów o złożonościach obliczeniowych: O(sqrt(m)logn), O(m/n), $O(n^2 \Delta \log m)$, $O(n \Delta m^2)$, $O(m^4/\Delta)$, $O(n^3 \text{sqrt}(m))$, $O(n^m)$, $O(m^n)$.

Uszereguj te złożoności rosnąco dla dopełnień grafów dwudzielnych pełnych $K_{r,s}$ w których r = s = p.

1.5 Złożoności – Struktury danych

1.5.5 Rozważ następujący problem decyzyjny:

„Dany jest graf n-wierzchołkowy G w postaci macierzy sąsiedztwa wierzchołków A; pytanie: czy suma stopni wierzchołków grafu G jest parzysta?”

Napisz efektywny algorytm, który zwraca true, gdy suma ta jest parzysta, a false gdy nie jest oraz podaj jego złożoność.

1.5.6 Dany jest graf G i wierzchołek v. Chcemy wiedzieć, czy w G istnieje cykl nieparzysty przechodzący przez v. Wykorzystując procedurę boolowską EvenPath(G, u, v), która działa w czsasie O(m) i sprawdza czy pomiędzy dwoma niesąsiednimi wierzchołkami u,v istnieje droga parzystej długości, napisz odpowiedni algorytm OddCycle(G, v) i oszacuj jego złożoność obliczeniową.

1.5.7 Dany jest problem decyzyjny: „Dany jest graf G, czy G zawiera dwa wierzchołki o tym samym stopniu?”. Napisz efektywny algorytm, który poprawnie odpowiada na to pytanie i oszacuj jego złożoność obliczeniową.

1.5.8 W problemie znajdowania k najkrótszych dróg pomiędzy ustaloną parą wierzchołków w sieci gęstej znane są algortymy o złożonościach: O(m+n+k), O(n+nlogm+k), O(m+nlogn+klogk), O(k(m+logn)), O(kn(m+loglogn)), O(kn(m+logn)). Uporządkuj te algorytmy od asymptotycznie najszybszego do asymptotycznie najwolniejszego dla: (1) ekstremalnie małych k; (2) ekstremalnie dużych k.

Wskazówka: Oszacuj wykładniczą liczbę dróg k dla przypadku (2).

1.5 Złożoności – Struktury danych

1.5.9 W pewnym drzewie n-wierzchołkowym, n>2 ponumerowano wierzchołki od minimalnego do maksymalnego stopnia, a następnie zapisano to drzewo w 3 różnych strukturach danych: macierzy sąsiedztwa, list sąsiadów, pęków wyjściowych. Interesuje nas, czy: a) to drzewo ma co najmniej 2 liście, b) co najwyżej 2 liście. Naszkicuj algorytmy i podaj ich złożoności obliczeniowe.

1.5.10 Liczba cyklomatyczna grafu jest zdefiniowana jako $\gamma(G) = m-n+1$. Naszkicuj 3 algorytmy dla obliczenia liczby $\gamma(G)$ i oszacuj ich złożoność dla:
a) macierzy sąsiedztwa
b) list sąsiedztwa
c) pęków wyjściowych

1.5.11 Następujący algorytm rozwiązuje problem k-kliki w G.

```
algorithm CliqueParametrized(G, k)
begin
   if k>Δ+1 then return false;
   for każdy wierzchołek v ∈ V(G)
      for każdy podzbiór S zbioru N(v), |N(v)|<= Δ
         if G(S ∪ {v}) jest k-kliką then return true;
   end_for
   return false;
end
```

1. Oszacuj jego złożoność w terminach n i Δ.
2. Podaj przykłady grafów dla których jest wielomianowy. Dla każdego przykładu podaj tę złożoność.
3. Podaj przykłady grafów, dla których jest niewielomianowy. Podaj tę złożoność.

1.5.12 Poniższa funkcja otrzymuje na wejściu macierz sąsiedztwa grafu prostego:

```
function X(G[1..n, 1..n], n)
begin
   B=G*G;
   s=0;
   for i=1 to n-1 do
      for j=i+1 to n do
         if G[i, j]=1 then s=s+B[i, j];
   return s/3;
end
```

1. Jaki parametr grafu G zwraca powyższa funkcja?
2. Jak należy ją zmodyfikować, by czas działania był asymptotycznie jak najmniejszy?

1.5.13 W historii minimaksowego pełnego skojarzenia grafu spójnego znane są dwa najlepsze algorytmy:
A1 o złożoności $O(m\sqrt{(n\log n)})$
A2 o złożoności $O(n\sqrt{(nm)})$

a) Podaj przedział zmienności m dla którego drugi algorytm jest asymptotycznie szybszy od pierwszego.

Programista wpadł na pomysł zaimplementowania ich obu:

```
procedure zagadka(G, n)
begin
   m=liczba krawędzi w G;
   if m<=n^2/log_2 n then A1 else A2;
end
```

b) Oszacuj czas działania zagadki dla przypadku grafu rzadkiego i grafu gęstego w funkcji n.

1.5 Złożoności – Struktury danych

1.5.14 Dany jest graf G. Interesuje nas, ile jest w nim gwiazd spinających. Napisz odpowiednie procedury i szacuj ich złożoności obliczeniowe.
a) MS(G), dla macierzy sąsiedztwa
b) LS(G), dla list sąsiedztwa
c) PW(G), dla pęków wyjściowych

1.5.15 W pewnym systemie informacyjnym chcemy mieć bazę danych, która w czasie O(1) będzie odpowiadała na pytania, jaka jest odległość pomiędzy dwoma wierzchołkami u, v ∈ V(G). Zaprojektuj taką strukturę danych i oszacuj funkcję n jej złożoność pamięciową oraz złożoność czasową jej zbudwania.
Wskazówka: zastosuj algorytm Dijkstry.

1.5.16 Graf n-wierzchołkowy G podany jest w postaci pęków wyjściowych. Napisz algorytm srawdzający, czy G ma podgraf K_4-e. Twój algorytm powinien mieć złożoność $O(n^3)$.

1.5.17 Dany jest n-wierzchołkowy graf G w postaci macierzy sąsiedztwa. Interesuje nas, czy graf ten zawiera trójkąt C_3. Naszkicuj odpowiedni algorytm wykonujący się w czasie $o(n^3)$.

1.5.18 Dany jest n-wierzchołkowy graf G w postaci macierzy sąsiedztwa. Interesuje nas, czy graf ten zawiera podgraw w postaci cyklu C_4. Napisz odpowiednią procedurę i oszacuj jej złożoność.

1.5.19 Napisz algorytm zwracający dla danego grafu G, liczbę jego cykli C_4. Oszacuj jego złożoność obliczeniową.

1.5 Złożoności – Struktury danych

1.5.20 Graf kubiczny G zapisano w macierzy sąsiedztwa wierzchołków. Napisz program, który zwraca liczbę diamentów (K_4-e) i uzasadnij jego poprawność oraz złożoność obliczeniową.

1.5.21 W historii problemu maksymalnego przepływu (MFP) w sieci znane są m. in. następujące algorytmy:
[1969] Edmondsa-Karpa
[1970] Dinica
[1974] Karzanova
[1977] Cherkaskyego
[1978] Galila
[1978] Shiloacha
[1980] Sleatora-Tarjana
[1986] Goldberga-Tarjana
[2013] Orlina

O złożonościach: $O(nm)$, $O(nm\log(n^2/m))$, $O(nm\log n)$, $O(nm\log^2 n)$, $O(n^{5/3}m^{2/3})$, $O(n^2\sqrt{m})$, $O(n^3)$, $O(n^2m)$, $O(nm^2)$.
Uszereguj te złożoności malejąco dla:
a) grafów planarnych
b) grafów samodopełniających

1.5.22 Zaprojektuj algorytm sprawdzający czy graf jest Eulerowski oraz oszacuj jego złożoność dla:
a) Macierzy Sąsiedztwa b) List Sąsiadów c) Pęków wyjściowych

Część 2. Problemy w informatyce

2.1 Problemy w informatyce – α redukcja

2.1.1 Sprowadź podany problem do 3-SAT:

$x_1 + \sim(x_2 + x_3) + \sim x_4$

2.1.2 Marek ma czarną skrzynkę, która dostaje na wejście wyrażenie 3^+-SAT, które odpowiada TAK jeśli jest spełnialne lub NIE jeśli nie jest. Andrzej ma wyrażenie 3-SAT i chciałby wiedzieć, czy jest spełnialne. W jaki sposób Andrzej może zmodyfikować wyrażenie 3-SAT, aby móc skorzystać z skrzynki Marka? Co by było, gdyby szkrzynka działała dla 3^{++}-SAT?

Przykładowe wyrażenie 3^+-SAT: (3-SAT)(a+~b+~c+d)

Przykładowe wyrażenie 3^{++}-SAT: (3-SAT) + a +~b +~c + d

2.1.3 Marek i Andrzej mają 2 kolokwia i czarną skrzynkę 3P, która rozwiązuje problem 3-Podziału. Zastanawiają się, czy mogą podzielić zadania na 2 równe części względem ich punktów. Co muszą zrobić, żeby użyć czarnej skrzynki do stwierdzenia, czy jest to możliwe?

2.1.4 Marek ma czarną skrzynkę odpowiadającą TAK lub NIE, która rozstrzyga problem 4-SAT. Andrzej ma wyrażenie 3-SAT. W jaki sposób zmodyfikuje on to wyrażenie, aby móc skorzystać ze skrzynki Marka?

2.1.5 Marek ma magiczną skrzynkę, która odpowiada na pytanie czy graf posiada klikę o rozmiarze <= k, ale tylko wtedy, kiedy graf ten posiada gwiazdę spinającą. Andrzej ma graf 100-wierzchołkowy G, który nie ma gwiazdy spinającej. Jak ma on zmodyfikować graf żeby móc skorzystać ze skrzynki Marka?

2.1 Problemy w informatyce – α redukcja

2.1.6 Załóżmy, że ja mam algorytm dający odpowiedź na problem: „Czy w danym grafie G znajduje się cykl Hamiltona”. Ty chcesz wiedzieć czy w twoim grafie znajduje się ścieżka Hamiltona. Jak zmodyfikujesz twój wejściowy graf aby móc skorzystać z mojego algorytmu?

2.1.7 Problem Ważonego Pokrycia Wierzchołkowego (WPW) zdefiniowany jest następująco: „Dany jest graf G_w z obciążonymi (wagami) wierzchołkami oraz próg p; czy w grafie G_w istnieje pokrycie wierzchołkowe o łącznej wadze <= p?”. Pokaż, że POKRYCIE WIERZCHOŁKOWE α WPW.

2.1.8 Marek ma czarną skrzynkę, która odpowiada TAK jeżeli ω(G) >= k, lub NIE jeżeli nierówność ta niezachodzi, ale tylko wtedy, kiedy podany na wejście graf jest Eulerowski. Andrzej ma graf G, który nie jest Eulerowski. Jak zmodyfikuje graf, aby mógł skorzystać z czarnej skrzynki?

2.1.9 Załóżmy, że ja mam algorytm rozwiązujący problem Pokrycia Wierzchołkowego przyjmujący na wejście graf G oraz liczbę k, a ty chcesz wiedzieć, czy w twoim grafie istnieje klika rozmiaru r. Jak zmodyfikujesz swój graf, aby skorzystać z mojego algorytmu?

2.1.10 Ty masz graf 100-wierzchołkowy G zapisany w postaci macierzy sąsiedztwa A(G). Zależy Ci na tym, aby stwierdzić, czy G zawiera klikę K_{10} lub większą. Ja mam program, który rozwiązuje Twój problem ale tylko wtedy, gdy na wejściu poda się mu macierz sąsiedztwa grafu niehamiltonowskiego. Jak przerobisz macierz sąsiedztwa aby skorzystać z mojego programu? Napisz procedurę do przekształcenia danych oraz oszacuj jej złożoność obliczeniową.

2.1.11 Dany jest problem optymalizacyjny PNP (PROBLEM NAJCIĘŻSZEGO PODGRAFU), który jest sformuowany następująco: ”Dany jest graf G obciążony krawędziowo oraz liczba k; Zwróć sumę wag najcięższego podgrafu k-wierzchołkowego.”

a) Sformułuj problem PNP w postaci decyzyjnej

b) Pokaż, że KLIKA α PNP_d

c) Pokaż, że transformacja jest wielomianowa oraz, że zachowuje problem

2.2 Problemy w informatyce – przynależność do klasy

2.2.1 Do jakiej klasy należą następujące podproblemy NP-zupełnego problemu 2P?

a) $2P_{int}$ przykład: {2,1,3,7,4,2,0}

b) $2P_{real}$ przykład: {2.0,3.7,8.8,sqrt(2)}

c) $2P_{1,2}$ przykład: {1,1,1,1,2,2,2,2}

d) $2P_{fib}$ przykład: 1,1,2,3,{5,8,13},21,…

2.2.2 Do jakiej klasy należą następujące problemy grafowe?

a) $\chi(G) <= 3$

b) $\chi(G_{PL}) <= 3$ (grafy planarne)

c) $\chi(G) >= k$

d) $\omega(G) >= k$

e) $\omega(G_{PL}) >= k$ (grafy planarne)

f) Czy G jest Eulerowski?

2.2.3 Do jakiej klasy należą następujące problemy kolorowania grafów planarnych?

a) $\chi(G) <=1$ b) $\chi(G) <=2$ c) $\chi(G) <= 3$ d) $\chi(G) <= 4$

e) $\chi(G) >=1$ f) $\chi(G) >=2$ g) $\chi(G) >= 3$ h) $\chi(G) >= 4$

2.2 Problemy w informatyce – przynależność do klasy

2.2.4 Załóżmy, że masz problem Π, który należy do klasy NP i prowadzisz nad nim badania. Co ogłosisz światu w przypadku gdy:

a) Złożoność każdego algorytmu rozwiązującego problem Π jest $\Omega(2^n)$

b) $\Pi \in P$

c) $\Pi \in NPC \wedge \Pi \in P$

d) $3SAT\ \alpha\ \Pi$

e) $\Pi\ \alpha\ 3SAT$

2.2.5 Marek robi pierniczki na święta i rozwałkowała ciasto do rozmiarów w x h. Zastanawia się, czy może wyciąć z niego pierniczki tak, aby nic nie zostało z ciasta. W oparciu o problem SUBSET SUM pokaż, że $PRC \in NPC$, gdzie PRC to Problem Rozkroju Ciasta.

2.2.6 Dany jest problem SP (SUBSET PRODUCT), którego dane wejściowe to zbiór liczb A oraz liczba p. Odpowiedź dla problemu wynosi TAK wtedy i tylko wtedy, gdy w zbiorze A istnieje iloczyn liczb równy p. W oparciu o problem SS (SUBSET SUM) pokaż, że $SP \in NPC$.

2.2.7 Marek robi zakupy w sklepie NieWyRe, który nie wydaje reszty oraz wszystkie ceny w nim zaokrąglone są do pełnych złotówek. Mając przy sobie banknot 100zł zastanawia się, czy może zrobić zakupy tak, aby nie zmarnować ani złotówki. Rozstrzygnij, do jakiej klasy należy ten problem.

2.2.8 Udowodnij, że decyzyjny problem kliki w grafie G należy do klasy NP. Przyjmij że graf jest w postaci macierzy sąsiedztwa.

2.2 Problemy w informatyce – przynależność do klasy

2.2.9 Udowodnij, że decyzyjny problem kliki w grafie G należy do klasy NPC.

2.2.10 Pokaż, że decyzyjny problem cyklu Hamiltona należy do NP.

2.2.11 Problem MEC(G, k) (Maximum Edge Coloring) zdefiniowany jest następująco: „Dane są: graf G i liczba naturalna k; znaleźć spójny podgraf grafu G zawierający największą możliwą liczbę krawędzi, który jest k-barwny krawędziowo”.

a) Podaj rozwiązanie problemu MEC(C_7, 2).

b) Pokaż, że MEC(G, 2) jest NP-trudny.

c) Rozpatrz problem mEC(G, k), który zamiast maksymalizować, minimalizuje liczbę krawędzi, i rozstrzygnij do jakiej klasy należy.

2.2.12 Dla problemu 2P (2-podziału) rozstrzygnij czy dane jego podproblemy również są NPC, czy też stają się wielomianowe:

a) n jest podzielne przez 3

b) liczby są 33 kolejnymi wyrazami ciągu Fibonacciego

c) wszystkie liczby a_i są podzielne przez 3

d) sumy elementów B i C nie muszą być równe, lecz mogą się róznić o conajwyżej 3

e) sumy te muszą się różnić o więcej niż 3
f) dwa zbiory B i C zastępujemy trzema B, C, D o równych sumach elementu

Udowodnij dla jednego podproblemu P i dla jednego podproblemu NPC jego przynależność do danej klasy.

2.2 Problemy w informatyce – przynależność do klasy

2.2.13 Marek i Andrzej wybrali się na zakupy. Przy kasie okazało się, że muszą zapłacić równo 100zł, a kasa nie wydaje reszty. W portfelu mają: a_1 monet 1zł, a_2 monet 2zł, a_3 monet 5zł, a_4 banknotów 10zł, a_5 banknotów 20zł, a_6 banknotów 50zł, gdzie ai>=0. Zastanawiają się, czy mogą uzbierać dokładnie 100zł.

a) Spróbuj zaprojektować algorytm wielomianowy rozwiązujący ten problem.

b) Jeśli nie potrafisz, spróbuj udowodnić jego NPC.

2.2.14 Problem zmotoryzowanego kimiwojażera zdefiniowany jest następująco: „Dany jest graf pełny obciążony wagami mającymi sens zużycia paliwa przy przejeździe z miasta a do miasta b oraz nagroda n_i >= 0 związana z odwiedzeniem każdego miasta i i mająca sens darowizny n_i litrów aliwa dla zmotoryzowanego kimiwojażera. Chcemy znaleźć taki cykl, odwiedzający każdy wierzchołek dokładnie raz, że wydatki kierowcy na paliwo są tak małe jak to możliwe.”

a) Sformułuj ten problem w postaci decyzyjnej określając zarówno dane jak i pytanie i nazwij ją PZK.
b) Udowodnij, że CYKL HAMILTONA α PZK.
c) Udowodnij, że ŚCIEŻKA HAMILTONA α PZK.

2.2.15 Graf planarny jest maksymalny, gdy dodanie jakiejkolwiek krawędzi łączącej niesąsiednie wierzchołki psuje jego planarność. Maksymalny Graf Planarny (MGP) jest 3-barwny wtedy i tylko wtedy, gdy jest Eulerowski. Czy problem wyznaczenia liczby chromatycznej maksymalnego grafu planarnego jest NPC?
a) Jeśli tak, przeprowadź odpowiednią α-redukcję.
b) Jeśli nie, to naszkicuj odpowiedni algorytm wielomianowy i oszacuj jego złożoność.
c) Podaj najmniejszy maksymalny graf planarny.

2.2 Problemy w informatyce – przynależność do klasy

2.2.16 Dany jest następujący problem decyzyjny: „Czy można rozbić graf G na k klik?”. Rozpatrz status problemu dla k=1, k=2, k=3. Jeżeli któryś z podproblemów jest wielomianowy, podaj jego złożoność dla najbardziej optymistycznej dla niego struktury danych.

2.2.17 OGRANICZONY PROBLEM KLIKI (OPK) zdefinowany jest następująco: „Dany jest graf G i liczby naturalne a, b, gdzie a<=b; czy w G istnieje klika o rozmiarze r, taka,że a<=r<=b?” Pokaż, że:

a) OPK ∈ NP
b) OPK ∈ NPC

2.2.18 Przedyskutuj złożoność obliczeniową następujących problemów decyzyjnych, zakładając, że graf G jest kubiczny i zapamiętany jest w postaci list sąsiedztwa. Ponadto, jeśli odpowiedź jest oczywista, napisz czy ona brzmi TAK czy NIE.

a) $\chi(G) <= 1$ b) $\chi(G) <= 2$ c) $\chi(G) <= 3$ d) $\chi(G) <= 4$

e) $\chi(G) >= 1$ f) $\chi(G) >= 2$ g) $\chi(G) >= 3$ h) $\chi(G) >= 4$

2.2.19 Wypełnij poniższą tabelę wpisując do niej:
TAK, NIE, NW (Nie Wiadomo).

Klasa problemów	Weryfikowalne w czasie wielomianowym	Rozwiązywalne w czasie wielomianowym
P		
NP		
NPC		
NPH		

2.2 Problemy w informatyce – przynależność do klasy

2.2.20 Dane są problemy decyzyjne A,…, F oraz X. Przypuśćmy, że A i B są problemami należacymi do klasy P, C i D są problemami w klasie NP-P, E jest NP-zupełny, zaś F nie należy do P. Zakładając że nie zachodzi P=NP, dla każdego z poniższych pytań wybierz najbardziej pasującą odpowiedź: TAK – prawda, NIE – nieprawda, BM – być może, czasami fałszywa albo nie wynika z założeń.

1.	Czy A α C?	11.	Czy 2SAT α B?
2.	Czy C α A?	12.	Czy 3SAT α C?
3.	Czy F α A?	13.	Czy C α 3SAT?
4.	Czy A α E?	14.	Czy 3SAT α E?
5.	Czy E α A?	15.	Czy E α 3SAT?
6.	Czy C α E?	16.	Czy A ∈ NPC?
7.	Czy C α D?	17.	Czy X ∈ NPC, gdy X α E?
8.	Czy E ∈ P?	18.	Czy X ∈ NPC, gdy E α X?
9.	Czy D ∈ P?	19.	Czy X ∈ NP, gdy X α C?
10.	Czy E ∈ NP?	20.	Czy X ∈ NP, gdy C α X?

2.2.21 Dany jest następujący problem decyzyjny: „Dany jest graf 100 wierzchołkowy G; Czy w grafie G znajduje się cykl Hamiltona?”. Rozstrzygnij do jakiej klasy należy ten problem.

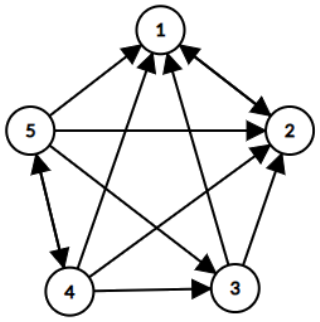
2.2.22 Dany jest następujący problem MLSP (Minimum Leaf Spanning Tree), którego treść jest następująca: „Dany jest graf G; Znajdź drzewo spinające w G o minimalnej liczbie liści!”. Pokaż, że problem MLSP jest NP-trudny.

2.2 Problemy w informatyce – przynależność do klasy

2.2.23 Uzupełnij poniższą tabelę przykładowymi problemami optymalizacyjnymi:

min/max	P/P	NPH/P	P/NPH	NPH/NPH
Nazwa problemu				
Złożoność wersji P				

2.2.24 Dany jest digraf relacji α między problemami. Rozstrzygnij, do jakiej klasy należa problemy 1 – 5, oraz uzupełnij tabelę przykładowymi problemami decyzyjnymi.



	Klasa	Problem	Dane	Pytanie
1.				
2.				
3.				
4.				
5.				

2.2.25 Kolorowanie kosztowe (KK) polega na tym, że kolory mają swoje koszty $c_1 <= c_2 <= \ldots <= c_n$ i za każdym razem, gdy kolorujemy kolejny wierzchołek, przydzielamy mu koszt tego koloru. Problem polega na tym, by tak pokolorować graf, aby sumaryczny koszt kolorowania wszystkich wierzchołków był jak najniższy. Pokaż, że problem KK jest NPH nawet wówczas, gdy $c_1 = c_2 = \ldots = c_k < c_{k+1}$...

2.3.1 Na płaszczyźnie znajduje się n punktów. Chcemy znaleźć ich średnice, czyli największą odległość pomiędzy dwoma z nich. Następujący algorytm rozwiązuje ten problem w sposób 2-przybliżony. Udowodnij ten fakt.

```
procedure polsrednica(n)
begin
   for i=2 to n do
      znajdz odleglosc i-tego punktu od punktu 1 i oznacz ją d_{i,1};
   max = max(d_{2,1}, d_{3,1}, …, d_{n,1});
   return max;
end
```

2.3.2 Które z poniższych stwierdzeń są prawdziwe w odniesieniu do problemu średnicy z zadania **2.3.1**?

a) Problem jest wielomianowy
b) Problem ten jest NPC
c) Problem ten jest NPH
d) Istnieje algorytm wielomianowy 3/2-aproksymacyjny
e) Istnieje algorytm wielomianowy 5/2-aproksymacyjny
f) Istnieje algorytm wielomianowy 2-absolutnie aproksymacyjny
g) Nie istnieje schemat PTAS

2.3.3 Problem Maksymalnej Liczby Zapamiętanych Programów (MLZP) określony jest następująco: „Dany jest zbiór n programów o długościach l_1, l_2, …, l_n i 2 dyskietki o pojemności L każda. Znaleźć upakowanie jak największej liczby programów na tych dyskietkach”. Jak wiadomo, algorytm Shortest-First (SF) dla problemu MLZP myli się o conajwyżej 1 program i polega na tym, że dyskietki zapełniamy w kolejności od najkrótszego do najdłuższego programu.

a) Pokaż, że SF jest 4/3-aproksymacyjny.
b) Pokaż, że dla MLZP nie istnieje schemat FPTAS, chyba że P=NP.

2.3.4 Jako kierownik zmiany masz pod opieką m=2 identyczne obrabiarki, które rozpoczynają pracę o godzinie 7^{00}. Do obrobienia jest n róznych elementów. Wiesz, że optymalny harmonogram przydziału tych elementów do maszyn ma długość 6 godzin, lecz nie wiesz, jak on wygląda. O godzinie 15^{00} nadają w telewizji program, który pragniesz po pracy obejrzeć. Twój komputer dysponuje schematem PTAS o złożoności $O(n\log^{1/\varepsilon}n+m^m)$. Który algorytm zastosujesz, by zdążyć na swój program, to znaczy jaka będzie jego złożoność obliczeniowa?

2.3.5 Zaprojektuj wielomianowy algorytm 4/3-aproksymacyjny, który zwraca liczbę chromatyczną grafu planarnego, oraz oszacuj jego złożoność obliczeniową, przyjmując iż jest on w postaci macierzy sąsiedztwa. Uzasadnij tą aproksymację.

2.3.6 Zaprojektuj wielomianowy algorytm aproksymacyjny, który zwraca indeks chromatyczny grafu. Podaj oraz uzasadnij jego aproksymacje oraz oszacuj jego złożoność obliczeniową.

2.3.7 Pokaż, że nie istnieje wielomianowy algorytm 1-absolutnie aproksymacyjny kolorujący wierzchołkowo grafy, chyba że P=NP.

2.3.8 Umówmy się, że ty masz 100-wierzchołkowy graf G, a ja posiadam wielomianowy schemat aproksymacyjny PTAS dla kolorowania grafów, który na twoim komputerze wykonuje się w czasie $n^{1/\varepsilon}$ mikrosekund, gdzie n jest liczbą wierzchołków. Chcesz wiedzieć, czy $\chi(G) = 4$. W jaki sposób skorzystasz z mojego schematu? Oszacuj ile sekund będą trwały twoje obliczenia.

2.3.9 Zaprojektuj algorytm przybliżony o jak najlepszej złożoności, który rozwiązuje problem maksymalnej kliki w grafie kubicznym. Podaj jego aproksymację oraz oszacuj jego złożoność.

2.3.10 Problem Minimal Bisection (MB) pyta, jak równo rozdzielić wierzchołki n-wierzchołkowego grafu G, tak by zminimalizować liczbę krawędzi łączących wierzchołki różnych podgrafów. Najszybszy znany algorytm dla problemu MB ma złożoność $O(2^{k*k*k}n^3\log^3 n)$, gdzie k oznacza wielkość minimalnego rozcięcia.
a) Podaj możliwie najszerszą rodzinę grafów, dla której algorytm ten staje się wielomianowy.
b) Jaką złożoność obliczeniową ma wówczas ten algorytm?
c) Podaj przykłady różnych klas grafów należacych do tej rodziny.
d) Udowodnij, że dla problemu MB nie istnieje schemat FPTAS, chyba że P=NP.

2.3.11 NP-trudny problem Maximum Acyclic Subgraph (MAS) pyta o maksymalny zbiór łuków digrafu D, który generuje digraf acykliczny. Problem ten ma prosty algorytm 2-aproksymacyjny, który polega na podzieleniu D na dwa podgrafy.
a) udowodnij 2-aproksymowalność algorytmu
b) pokaż, że dla MAS nie istnieje schemat FPTAS, chyba że P=NP

2.3.12 Rozważ problem wyznaczania indeksu chromatycznego n-wierzchołkowego grafu G, czyli $\chi'(G)$. Uzasadnij prawdziwość lub fałszywość następujących stwierdzeń dotyczących tego problemu.
a) Istnieje algorytm 1-absolutnie aproksymacyjny o złożoności O(n).
b) Istnieje wielomianowy algorytm 3/2-aproksymacyjny.
c) Istnieje wielomianowy algorytm 4/3-aproksymacyjny.
d) Istnieje wielomianowy algorytm 5/4-aproksymacyjny.
e) Istnieje niewielomianowy schemat aproksymacyjny.
f) Istnieje całkowicie wielomiaowy schemat aproksymacyjny.

2.3.13 Algorytm LF (Largest First) dla kolorowania wierzchołków grafu maluje je zachłannie poczynając od wierzchołka o najwyższym stopniu i kończąc na wierzchołku o najniższym stopniu.
a) Oszacuj złożoność obliczeniową algorytmu LF.
b) Udowodnij, że LF optymanie koloruje cykle C_4 i C_5 ale suboptymalnie C_6.
c) Udowodnij, że LF nie jest aproksymacyjny, tj nie istnieje stała c, taka, że LF(G) <= cχ(G).
d) Udowodnij, że LF jest k-bezwzględnie aproksymacyjny i l-względnie aproksymacyjny w odniesieniu do grafów kubicznych.

2.3.14 Zakładając, że nie zachodzi równość P=NP, pokaż że dla problemu wierzchołkowego kolorowania grafów nie istnieje:
a) Schemat FPTAS
b) Schemat PTAS

2.3.15 Dany jest następujący algorytm 2-aproksymacyjny rozwiązujący problem pokrycia wierzchołkowego:

```
procedure VertexCover(G, n, k+2)
begin
   if m = 0 then return k;
   wybierz dowolna krawedz uv;
   return VertexCover(G-u-v, n-2, k+2);
end;
```

Udowodnij jego 2-aproksymalność.

2.3.16 Dany jest następujący algorytm 2-aproksymacyjny rozwiązujący problem SP (SUMA PODZBIORU):

```
procedure SS*(A, n, p)
begin
   sort(A); //sortowanie nierosnąco
   b=0;
   for i=1 to n do
      if b+A[i] <= p then b = b+A[i];
   return b;
end;
```

a) Oszacuj złożoność tej procedury.
b) Podaj 3 liczby dla których $A_{opt} = 2A$
c) Niech SS** będzie modyfikacją powyższego algorytmu, z której wykreślono instrukcję sortowania. Pokaż, że nie istnieje takie ε, dla którego SS** jest ε-aproksymacyjny.

Odpowiedzi

Złożoności - Odpowiedzi

1.1.1:

a) $T(n) = \Theta(n)$ b) $T(n) = \Theta(n\sqrt{n})$ c) $T(n) = \Theta(n\sqrt{n})$
d) $T(n) = \Theta(\sqrt{n})$ e) $T(n) = \Theta(n^2)$ f) $T(n) = \Theta(n^2)$
g) $T(n) = \Theta(\log n)$ h) $T(n) = O(n^2)$ i) $T(n) = \Theta(n^3 \log n)$

1.1.2:

a)

```
s=0;
for i=1 to n do
    for j=1 to i do
        s = s + j;
```

b)

```
s=0
for i=1 to n do
s = s + ((i + 1)/2)*i;
```

c)

```
return (n*n*n + 3*n*n + 2*n)/6;
```

1.1.3:

$LK(n) = n^2 \log_2 \sqrt{n}$

1.1.4:

$LK(n) = \Theta(n^3 \sqrt{n})$

1.1.5:

a) brak własności stopu b) brak własności stopu dla n>1;

dla n <= 1: T(n) = O(1);

c) $T(n) = \Theta(\log\log n)$ d) $T(n) = \Theta(\log\log n)$

1.1.6:

a)

```
s=0;
for i=1 to n do
   for j=1 to n do
       for k=1 to i do
          s = s + j;
```

b)

```
s=0;
for i=1 to n do
    for j=1 to n do
        s = s + i*j;
```

c)

```
s=0;
for i=1 to n do
   s = s + i * n*(n+1)/2;
```

d)

```
return (n*(n+1) * n*(n+1))/4;
```

1.1.7

1! mod 10 = 1; 2! mod 10 = 2; 3! mod 10 = 6; 4! mod 10 = 4;
5! mod 10 = 0; 6! mod 10 = (6 * 5!) mod 10 = 0 . . .
Procedura zwraca: (n!)(mod 10) dla n >= 1 (ostatnią cyfrę silni)

Jeżeli przyjmiemy, że n może się równać zero, to w takim razie procedura zwraca: ((n!) mod 10) * min(n, 1) dla n>=0

1.1.8

$\Theta(n^2)$, pierwsza pętla wykonuje się O(1) razy, druga zawsze n^2 razy

1.2.1

a) T(n) = Θ(n) b) T(n) = O(nlogn) ∧ T(n) = Ω(n) c) $T(n) = \Theta(2^n)$

1.2.2

a) T(n) = O(n), wywołanie A(n-1) kończy się na tym, że funkcja ta zwraca 1, zatem kolejne wywołanie A(A(n-1)) jest równoważne A(1),
Czyli mamy: T(n) = T(n-1) + 1; T(n) = O(n)

b) $T(n) = O(2^n)$, wywołanie B(n-1) kończy się na tym, że funkcja ta zwraca dokładnie n, zatem kolejne wywołanie B(B(n-1)) jest równoważne B(n-1), czyli T(n) = 2*T(n-1) + 1, zatem $T(n) = O(2^n)$

1.2.3

Kwadrat liczby parzystej modulo 4 zawsze daje 0, natomiast kwadrat liczby nieparzystej 1

Optymistyczny (liczba parzysta): T(n) = O(1), pętla się nie wykona

Pesymistyczny (liczba nieparzysta): $T(n) = O(3^n)$, pierwsza pętla (for) wykona się dokładnie 1 raz, druga (while) dokładnie 3 razy, czyli T(n) = 3*T(n-2) + c = $O(3*T(n-1) + c) = O(3^n)$, zakładając, iż działamy na liczbach naturalnych, i n > 0, przy czym w przypadku gdy mamy n = 1, musimy zalozyc ze n-j = 0, w pzeciwnym wypadku, procedura nie ma wlasnosci stopu

1.2.4

T(n) = T(n/2) + T(n/4) + n = O(2*T(n/2) + n) = O(nlogn)

1.2.5

$T(n) = 2*T(n-1) + 1 = 2^{(n-1)} = O(2^n)$

1.2.6

$T_o(n) = 2T_o(n/2) + n = O(n\log n)$

$T_p(n) = T_p(n-1) + n = O(n^2)$

1.2.7

funkcja oblicza 2*n;

$T(n) = 4T(n/2) + n^2 = \Theta(n^2 \log n)$

1.2.8

a) $\log T(k) = 2\log T(k-1)$; $U(k) = 2*U(k-1)$; $U(k) = 2^k$; $T(k) = 2^{(2\wedge k)}$

b) $m = \ln n$; $T(e^m) = 2*T(\lfloor e^{(m/2)} \rfloor) + m <= 2*T(e^{(m/2)}) + m$;
$U(m) <= 2*U(m/2) + m$; $U(m) = O(m\log m)$; $T(n) = O(\log n \log\log n)$

c) $\log_2 n = m$; $T(2^m) = 2*T(2^{(m-1)}) + m$; $U(m) = O(2^m)$;
$U(m) = 2*T(m-1) + m$; $T(2^n) = O(2^n)$; $T(n) = O(n)$

d) $T(n-1) = T(n-2) + T(n-3) + \ldots + T(1)$; $T(n) = 2*T(n-1)$; $T(n) = \Theta(2^n)$

1.2.9

Funckja zwraca a^n. Dla n=15 wykonuje 8 mnożeń.

a) {x = a*a; y = x*a; z= x*y; return (z*z*z);}
b) {b = a; a = a*a; a = a*a; a = a*a; a = a*a; a = a /b; return a;}

1.2.10

$T(i) = T(i-1)*T(i-1)$; $T(i) = T\wedge 2(i-1)$; $\log\{2\}(T(i)) = 2\log(T(i-1))$; $U(k) = 2U(k-1)$;
$U(k) = 2^k$; $2^k = \log n$; $T(i) = 2^{(2\wedge k)}$; $k = O(\log\log n)$; //wersja Prof. Kubale

Liczymy tempo wzrostu i:

$T(i) = T^2(i-1)$; $\log\{2\}(T(i)) = 2\log\{2\}(T(i-1))$; $U(i) = 2U(i-1)$; $U(i) = \Theta(2^i)$;

$T(i) = \Theta(2^{(2\wedge i)})$; // w takim tempie rośnie i

$n > 2^{(2\wedge i)}$; $\log\{2\}(n) > 2^i$; $\log\{2\}(\log\{2\}(n)) > i$; $i = O(\log\log n)$;

$T(n) = O(\log\log n)$

1.3.1

$TW(n) = 2TW(n/2) + TW(n/4) + n^2$

$TW(n) <= 3TW(n/2) + n^2 \ \wedge\ TW(n) >= 3TW(n/4) + n^2$

$TW(n) = O(n^2) \wedge TW(n) = \Omega(n^2)$

$TW(n) = \Theta(n^2)$

1.3.2

$T(n) = aT(n-1) + b(n-1)$; $T(3) = aT(n-1) + 2b$; $T(3) = 4a + 2b$

$T(4) = 8a + 3b$; $T(4) = 2T(3) – b$;

Wiemy, że każdy dodany okrąg ma conajwyżej 2 punkty wspólne z każdym z innych okręgów, zatem dostajemy maksymalnie 3*2 nowych obszarów.

$T(4) = 8 + 3*2$; $T(4) = 14$; $14 = 2*T(3) – b$; $14 = 16 – b$; **b = 2**

$T(3) = 4a + 4$; **a = 1**; **T(n) = T(n-1) + 2(n-1)**

$TW(n) = T(n-1) + n$; $a = 1, d(n) = n$; $d(n) = \omega(a^n)$; $TW(n) = O(n^2)$

1.3.3

Można rozrysować problem na kartce. R(2)=4, R(3)=7.

$R(2) = 2x + 2y$; $R(3) = 4x + 3y$; $R(3) = 3/2 * R(2) + x$; $R(3) = 6 + x$; $x=1;\ y=1;$

$R(n) = R(n-1) + n$; $R(n) = O(n^2)$

1.3.4

$TW(n) = 8TW(n/2)-2TW(n/2 -1)+n^2$; $TW(n) >= 6TW(n/2)+n^2$; $TW(n) = \Omega(n^{\log\{2\}(6)})$

$ZP(n) = \max\{ZP(n/2), ZP(n/2-1)\} + 1$; $ZP(n) = \Theta(\log n)$;

1.3.5

$TW(n) = T(n-1) + T(n-2) + n^2$; $TW(n) <= 2T(n-1) + n^2$; $a^n = 2^n$; $d(n) = n^2$

$n^2 = O(2^n/n^\varepsilon), \varepsilon > 1$

$TW(n) = O(2^n)$

1.3.6

prawy = n, lewy = 1, możemy zatem zauważyć, ze wywołania rekurencyjne będą następujące:

sortuj(tab, 1, 2/3 * n);
sortuj(tab, 1/3 * n, n);
sortuj(tab, 1, 2/3 * n);
$T(n) = 3T(2/3*n) + 1$; $T(n) = 3T(n/(3/2)) + 1$; $a=3, b=3/2, d(n) = 1 = d(b) < a$;
$T(n) = \Theta(n^{\log\{1,5\}(3)})$; $T(n) = \Theta(n^{2,7})$

$ZP(n) = ZP(n/(3/2)) + 1$; $ZP(n) = O(\log n)$

1.3.7

$TW(n) = 8TW(n/2) – TW(n/4) + 1$

$TW(n) <= 8TW(n/2) + 1$; $TW(n) = O(n^3)$

$TW(n) >= 7TW(n/2) + 1$; $TW(n) = \Omega(n^{\log\{2\}(7)})$

1.4.1

Procedura Fib:

$T(n) = T(n-1) + T(n-2) + 1$; $T(n) <= 2T(n-1) + 1$; $T(n) = O(2^n)$;

Rozmiarem danych jest tutaj liczba bitów potrzebnych do zapisania liczby n, zatem: $\mathbf{n = 2^r}$, czyli: $ZC(r) = O(2^{(2\hat{}r)})$;

Procedura hybryda

Pierwsza część procedury wykonuje się w $\Theta(1)$, druga w czasie $\Theta(n)$,

zatem: $T(n) = O(n)$;

$n = 2^r$; $ZC(r) = O(2^r)$

1.4.2

Funkcja oblicza x^n. $T(n) = 2*T(n/2) + 1$, czyli $T(n) = O(n)$;
$ZC(r) = O(2^r)$; złożoność jest wykładnicza

Modyfikacja:

Traktujemy podnoszenie do kwadratu jako jedną operację, mamy wtedy:

$T(n) = T(n/2) + 1$; a = d(b) = 1; czyli $T(n) = O(n^{\log\{2\}(1)}\log n) = O(\log n)$;

$ZC(r) = O(\log(2^r)) = O(r)$; złożoność jest liniowa

1.4.3

Dla i=0: brak własności stopu
Dla i=1: brak własności stopu dla n>1, w przeciwnym wypadku $LK(n) = \Theta(1)$
Dla i=2: $LK(n) = \Theta(\log\log n)$, $ZO(r) = \Theta(\log n)$, złożoność jest **subliniowa**

1.4.4

$T(n) = 2T(n/4) + 1$; $T(n) = \Theta(\sqrt{n})$; $ZC(n) = \Theta(\sqrt{n})$

Rozmiarem danych wejściowych jest liczba, zatem $n = 2^r$

$ZO(r) = \Theta(2^{r/2})$

1.4.5

$LK_{best}(n) = O(n)$; $Lk_{worst}(n) = O(n^9)$; $LK_{av}(n) = 1/n^3\, n^9 + (1 - 1/n^3)*n = O(n^6)$;

$ZO_{av}(r) = O(2^{6r})$; aby złożoność była taka sama jak liczba kroków, należy na wejście podać **tablicę**, której rozmiarem jest liczba n.
Wtedy mamy: $ZO_{av}(n) = O(n^6)$

1.4.6

1. $ZO_{worst}(n) = \Theta(n^2)$

2. $(n^2-1)/n^2 * n + 1*n^2 * 1/n^2$; $ZO_{av}(n) = O(n)$

3. $ZO_{best}(n) = \Theta(n)$

Rozmiarem danych jest tutaj rozmiar tablicy, zatem LK = ZO

1.5.1

a) $ZO = O(n^{\log\{5\}(4)})$
b) Metoda Strassena ma lepsza złożoność, gdyż $\log_2 7 < \log_5 100$
c) Makarova: $n^{\log\{5\}(100)} = 100^{\log\{5\}(n)}$; $100^{\log\{5\}(25)} = 10000$
   Strassena: $n^{\log\{2\}(7)} = 7^{\log\{2\}(n)}$; (rozszerzamy macierz do rozmiarów 32x32);
   $7^{\log\{2\}(32)} = 7^5 = 16807$;
   W tym przypadku bardziej opłacalna jest metoda Makarova.
d) Mnożenie macierzy o rozmiarach 125x125 zajmuje 1000000 mnożeń, czyli trwa 1000s, czyli 16,6min.

1.5.2

M1 x M2 wymaga **pqr** obliczeń, w wyniku otrzymujemy macierz o rozmiarach p x r, kolejne mnożenie wymaga **prs** obliczeń. ostatecznie mamy: **pqr + prs**

M2 x M3 wymaga **qrs** obliczeń, w wyniku otrzymujemy macierz o rozmiarach q x s, kolejne mnożenie wymaga **pqs** mnożeń. Ostatecznie mamy: **qrs + pqs**

Aby obliczenie pierwszego przypadku było szybsze od drugiego musi być spełniona nierówność:

pqr + prs < qrs + pqs |:pqrs

1/s + 1/q < 1/p + 1/r

1.5.3

Możemy zauważyć, że **m < 2n**. (dokładny wzór w przypadku kraty rozmiaru sqrt(n) x sqrt(n) wynosi m=2n-2sqrt(n)). Ponadto dla krat kwadratowych **Δ <= 4**.

sqrt(n), 1, n^2logn, n^3, n^4, n^3sqrt(n), n^{2n}, $(2n)^n$

$n^{2n} = n^n * n^n$; $(2n)^n = 2^n * n^n$

O(m/n), O(sqrt(m)), O(n^2Δlogm), O(nΔm^2), O(n^3sqrt(m)), O(m^4/Δ), O(m^n), O(n^m)

1.5.4

Z racji tego, iż w tym zadaniu r = s = p (czyli każdy z obu zbiorów ma dokładnie p wierzchołków), możemy wyznaczyć:

Δ = n/2 = O(n) oraz **m = n/2 * n/2 = O(n^2).**

O(m/n), O(sqrt(m)logn), O(n^2Δlogm), O(n^3sqrt(m)), O(nΔm^2), O(m^4/Δ), O(m^n), O(n^m)

1.5.5

```
function parzystaSumaStopni(G, n)
begin
   return true;
end;
```

ZO(n) = Θ(1)

Uzasadnienie: dla dowolnego grafu nieskierowanego, suma stopni wierzchołków jest zawsze parzysta, gdyż każda dodana krawędź zwiększa stopień dwóch wierzchołków o 1, czyli suma stopni wierzchołków = 2*m, co dla każdego n należącego do liczb naturalnych z zerem będzie parzyste

1.5.6

```
function OddCycle(G, v)
begin
   for u ∈ N(v), gdzie N(v) jest sąsiedztwem wierzchołka v
   begin
      removeEdge(G, u, v);
      inOddCycle = EvenPath(G, u, v);
      addEdge(G, u, v);
      if inOddCycle then return true;
   end
   return false;
end
```

przejście po wierzchołkach sąsiednich do v ma czas O(Δ). Zakładamy, że removeEdge i addEdge wykonują się w czasie stałym. Zatem złożoność tej procedury wynosi: O(Δm). Przyjmując dodatkowo, że Δ <= n-1, możemy oszacować złożoność procedury na O(nm), aczkolwiek stracimy trochę na dokładności oszacowania.

1.5.7

```
function has2SameDegrees(G, n)
begin
   if n=1 then return false;
   else return true;
end
```

ZO(n) = Θ(1)

uzasadnienie: załóżmy, że w danym grafie wszystkie wierzchołki mają różne stopnie oraz n>1. Istnieje zatem wierzchołek v1, który ma stopień 0 oraz wierzchołek v2, który ma stopień n-1, co jest ze sobą sprzeczne, gdyż wierzchołek v2 musiałby być połączony ze wszystkimi innymi wierzchołkami, również z v1, który ma stopień 0.

1.5.8

Wiemy, że jest to sieć gęsta, zatem możemy oszacować, iż jest $O(n^2)$ krawędzi.
$m = O(n^2)$

W przypadku **(1)** k traktujemy jako stałą, gdyż jest ekstremalnie małe. **$k = \Theta(1)$**
Mamy zatem po podstawieniach:
n^2+n; $n+n\log n$; $n^2+n\log n$; $n^2+\log n$; $n(n^2+\log\log n)$; $n(n^2+\log n)$
Po uszeregowaniu otrzymujemy:
$O(n+n\log m+k)$, $O(k(m+\log n))$, $O(m+n+k)$, $O(m+n\log n+k\log k)$, $O(kn(m+\log\log n))$, $O(kn(m+\log n))$
//ten przypadek jest dziwny (dziwnie zapisany w orginale) i wymaga weryfikacji
W przypadku **(2)** szacujemy k, jako funkcję wykładniczą. (możemy wyobrazić sobie ciąg bitowy rozmiaru $O(n^2)$, gdzie 1 oznacza, że krawędź należy do drogi, a 0 że nie należy), czyli mamy liczbę możliwych dróg: $k = O(2^{n^2})$
n^2+n+2^n; $n+n\log n+2^n$; $n^2+n\log n+n2^n$; $2^n(n^2+\log n)$; $2^n(n^3+n\log\log n)$; $2^n(n^3+n\log n)$
Po uszeregowaniu otrzymujemy:
$O(n+n\log m+k)$, $O(m+n+k)$, $O(m+n\log n+k\log k)$, $O(k(m+\log n))$, $O(kn(m+\log\log n))$, $O(kn(m+\log n))$

1.5.9

a)

Dla wszystkich 3 struktur danych, jest ta sama procedura:

MS, LS, PW:

```
procedure conajmniej2(G, n)
begin
   return true;
end
```

ZO(n) = Θ(1)

uzasadnienie: każde drzewo o liczbie wierzchołków n>2, ma conajmniej 2 liście.

b)

Jako iż wiemy, że w drzewie wierzchołki ponumerowane są od minimalnego do maksymalnego stopnia, to liście muszą być na samym początku struktury. Zatem we wszystkich przypadkach wystarczy sprawdzić, czy trzeci wierzchołek to liść.

MS: ZO(n) = O(n)

```
procedure conajwyzej2(G, n)
begin
   stopien=0;
   for j=1 to n do
      if G[3][j]>0 then stopien=stopien+1;
   if stopien > 1 then return false;
end
return true;
end
```

LS: ZO(n) = O(1)

```
procedure conajwyzej2(G, n)
begin
   v = listy[3];
   stopien = 0;
   for u ∈ N(v) do
   begin
      stopien = stopien+1;
      if stopien > 1 then return false;
   end
   return true;
end
```

PW: ZO(n) = O(1)

```
procedure conajwyzej2(V, E, n)
begin
   //vecV – wektor wierzchołków
   if vecV[4] – vecV[3] > 1 then return false;
   return true;
end
```

1.5.10

a) $ZO(n) = \Theta(n^2)$

```
procedure gammaMS(G, n)
begin
   int m=0;
   for i=1 to n do
      for j=i do
         if G[i][j]>0 then m=m+1;
   return m-n+1;
end
```

b) $ZO(n) = \Theta(n+m)$

```
procedure gammaLS(G, n)
begin
    int m=0;
   for v ∈ V(G) do
       for N(v) do m=m+1;
    return m/2 – n + 1;
end
```

c) $ZO(n) = \Theta(1)$

```
procedure gammaPW(V, E, n)
begin
   return (V[n+1]-1)/2 – n + 1;
end
```

1.5.11

Jako iż nie mamy nigdzie napisanej funkcji, wyznaczającej Δ, zakładamy iż jest ona znana, w przeciwnym wypadku musielibyśmy założyć, że taka funkcja ma złożoność $O(n^2)$.

1. $ZO(n) = O(n\Delta^2 2^{\Delta})$.
2. hiperkostki Q_{logn}, $O(n^2 \log^2 n)$; drogi P_n, $O(n)$; cykle C_n, $O(n)$;
3. grafy pełne K_n, $O(n^3 2^n)$; gwiazdy S_n, $O(n^3 2^n)$; koła W_n, $O(n^3 2^n)$

grafy dwudzielne pełne $K_{r,s}$, $O(n^3 2^{n/2})$; grawy trójdzielne pełne $K_{r,s,t}$, $O(n^3 2^{2n/3})$

1.5.12

1. Zliczamy wszystkie drogi o długości 2, między wierzchołkami połączonymi krawędzią, na końcu dzielimy tą liczbę przez 3. Zatem procedura ta zwraca liczbę trójkątów (C_3) w grafie G. Dzielenie przez 3 wynika z tego, że każdy trójkąt ma 3 takie wierzchołki.
2. Aby czas działania był asymptotycznie jak najmniejszy, można zastosować szybkie mnożenie macierzy, np. wg Strasena, o złożoności $O(n^{\log\{2\}(7)})$

1.5.13

a) A2 jest asymptotycznie szybsze od A1, gdy: $m > n^2/\log_2 n$

b) Zakładamy, że graf rzadki ma O(n) krawędzi, a graf gęsty $O(n^2)$ krawędzi.

$ZO_{rzadkie}(n) = O(n\sqrt{}(n\log n))$; $ZO_{gęste}(n) = O(n^2\sqrt{}n)$;

1.5.14

a) ZO = $\Theta(n^2)$

```
procedure MS(G)
begin
   g=0;
   for i=1 to n do begin
      deg=0;
      for j=1 to n do
         if G[i,j]>0 then deg=deg+1;
      if deg=n-1 then g=g+1;
   end
   return g;
end
```

b) ZO = $\Theta(n+m)$

```
procedure LS(G)
begin
   g=0;
   for v=1 to n do begin
      deg=0;
      for u ∈ N(v) do
         deg=deg+1;
      if deg=n-1 then g=g+1;
   end
```

c) ZO = $\Theta(n)$

```
procedure PW(G)
begin
   g=0;
   for i=1 to n do    //przyjmujemy, że vecV to mniejszy wektor
      if vecV[i+1]-vecV[i]=n-1 then g=g+1;
   return g;
end
```

1.5.15

Struktura danych, którą musimy zbudować to macierz kwadratowa rozmiarów n x n. W konkretynych polach zawiera odległości między danymi wierzchołkami. Jej złożoność pamięciowa zatem, to $\Theta(n^2)$.
Algorytm Dijkstry ma złożoność $O(n^2)$.
Aby wypełnić naszą strukturę, musimy dla n wierzchołków wykonać algorytm Dijkstry, i wpisać wyniki w odpowiednie pola, to jest dla danego wiersza u, odległości od wszystkich wierzchołków v.
Wykonujemy n razy algorytm o złożoności $O(n^2)$, zatem złożoność czasowa zbudowania naszej struktury wynosi: $O(n^3)$.

1.5.16

```
procedure maDiament(G)
begin
   //przerobienie pęków wyjściowych na macierz sąsiedztwa
   //vecV – mniejszy wektor, vecE – wektor z krawędziami
   for i=1 to n do
      for j=0 to n do M[i,j]=0;
   for i =1 to n do
      for j=vecV[i] to vecV[i+1]-1 do
         M[i,vecE[j] = 1;
   M2 = M*M;  //mnożenie wg Strassena, O(n^log{2}(7))
   for i=1 to n do
      for j=1 to n do
         if M[i,j]=1 and M2[i,j]>1 then return true;
   return false;
end
```

ZO = $O(n^{\log\{2\}(7)})$

1.5.17

```
procedure hasTriangle(G)
begin
   //mnożenie wg Strassena O(n^log{2}(7))
   M2 = G*G;
   for i=1 to n do
      for j=1 to n do
         if G[i,j]>0 and M2[i,j]>0 then
            return true;
   return false;
end
```

1.5.18

```
procedure hasC4(G)
begin
    M2 = G*G; //wg Strassena
    for i=1 to n do
       for j=1 to n do
          if M2[i,j]>1 and i!=j then
             return true;
    return false;
end
```

ZO = $O(n^{\log\{2\}(7)})$

1.5.19

```
procedure numberOfC4(G)
begin
   c4 = 0;
   M2 = G*G; //wg Strassena
   for i=1 to n do
      for j=1 to n do
         if M2[i,j] > 1 and i!=j then c4 = c4 + M2[i,j]*(M2[i,j]-1)/2;
   return c4/4;
end
```

ZO = $O(n^{\log\{2\}(7)})$

1.5.20

```
procedure liczbaDiamentow(G, n)
begin
   M2 = pomnoz(G, G);  //mnozenie wg Strassena, O(n^log{2}(7))
   diamenty = 0;
   for i=1 to n do
      for j=1 to n do
         if M[i][j]=1 and M2[i][j]>1 then diamenty=diamenty+1;
   return diamenty/2;
end
```

uzasadnienie: z racji tego, iż graf jest kubiczny, nie może on mieć 2 sklejonych diamentów. Zliczamy wszystkie pary wierzchołków, takie że istnieje krawędź v1, v2 oraz dwie drogi długości 2 z v1 do v2. Ostateczną wartość dzielimy przez 2.

1.5.21

a)

m <= 3n – 6 = O(n)

n^2, $n^2\log n$, $n^2\log n$, $n^2\log^2 n$, $n^{7/3}$, $n^2\sqrt{n}$, n^3, n^3, n^3

$O(n^3)$, $O(n^3)$, $O(n^3)$, $O(n^2\sqrt{n})$, $O(n^2\log^2 n)$, $O(n^2\log n)$, $O(n^2\log n)$, $O(n^2)$

b)

graf samodopełniający ma m = polowie liczby krawędzi grafu pełnego

$m = n(n-1)/4 = O(n^2)$

n^3, n^3, $n^3\log n$, $n^3\log^2 n$, n^3, n^3, n^3, n^4, n^5

$O(n^5)$, $O(n^4)$, $O(n^3\log^2 n)$, $O(n^3\log n)$, $O(n^3)$, $O(n^3)$, $O(n^3)$, $O(n^3)$, $O(n^3)$

1.5.22

a) MS: ZO = $O(n^2)$

```
procedure isEulerian(G, n)
begin
   for i=1 to n do
   begin
      degv = 0;
      for j=1 to n do
         if G[i,j] >0 then
            degv = degv+1;
      if degv(mod 2) = 1 then
         return false;
   end
   return true;
end
```

b) LS: ZO = O(n+m)

```
procedure isEulerian(G, n)
begin
    for v=1 to n do
     begin
        degv = 0;
        for u ∈ N(v) do degv = degv+1;
        if degv (mod 2) = 1 then return false;
     end
     return true;
 end
```

c) PW: ZO = O(n)

```
procedure isEulerian(G)
begin
   for i=1 to n do
   begin
      if (vecv[i+1] – vecv[i]) (mod 2) = 1 then return false;
   end;
   return true;
end;
```

2.1.1

$x_1 + \sim(x_2 + x_3) + \sim x_4 = (x_1+\sim x_4+x_5+\sim x_2)(x_1+\sim x_4+x_5+\sim x_3)$

dodajemy nowe zmienne x6 oraz x7, które nie wpłyną na spełnialność orginalnego wyrażenia:

$(x_1+\sim x_4+x_6)(x_5+\sim x_2+\sim x_6)(x_1+\sim x_4+x_7)(x_5+\sim x_3+\sim x_7)$

2.1.2

Wystarczy, że Andrzej pomnoży swoje wyrażenie 3-SAT przez $(x_n + \sim x_n)$, co jest równoważne przemnożeniu przez 1, zatem nie wpływa na ostateczny wynik.

W przypadku, gdyby skrzynka działała dla 3^{++}-SAT, zwracała by zawsze TAK, zatem Andrzej nie mógłby sprawdzić, czy jego wyrażenie jest spełnialne.

2.1.3

Muszą dodać sztuczny element o punktacji połowy wszystkich zadań. Tym sposobem wiadomo, że jedną z 3 równych części na pewno będzie ten sztuczny element, zatem dla całego zbioru prawdziwych zadań zostanie rozstrzygnięty problem 2-Podziału. Ponadto jeżeli istnieje 2-podział w orginalnym zbiorze, to istnieje 3-podział w zbiorze z dodanym sztucznym elementem.

2.1.4

Aby wyrażenie było 4-SAT, Andrzej musi dodać kolejną zmienną do nawiasów w ten sposób, aby nie wpływała ona na końcowy wynik, np.:

$(x_1+\sim x_2+x_3) => (x_1+\sim x_2+x_3+x_4)(x_1+\sim x_2+\sim x_3+\sim x_4)$

2.1.5

1. Dodajemy dodatkowy wierzchołek tworząc graf G^+ i łączymy go z każdym innym, powstanie tak gwiazda spinająca.

2. Przyjmujemy k = 50

3. Metodą bisekcji (zwiększamy/zmniejszamy wynik o połowę w zależności od odpowiedzi) uzyskujemy rozmiar maksymalnej kliki w g^+.

4. Od rozmiaru kliki odejmujemy 1, otrzymując odpowiedź dla orginalnego grafu G.

2.1.6

Wystarczy stworzyć graf H poprzez dodanie wierzchołka uniwersalnego do grafu G, czyli wierzchołka połączonego ze wszystkimi innymi wierzchołkami. (H=G+v) Jeżeli istnieje w grafie G ścieżka Hamiltona, to w grafie H mamy pewność, że dwa końce ścieżki są ze sobą połączone, czyli graf jest Hamiltonowski. Jeżeli graf H jest Hamiltonowski i usuniemy z niego dodany wierzchołek, to będzie on posiadał ścieżkę Hamiltona.

2.1.7

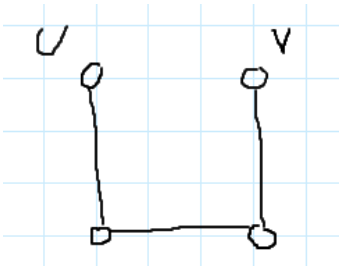
Aby sprowadzić problem POKRYCIE WIERZCHOŁKOWE (PW) do WPW należy z grafu wejściowego G stworzyć graf G_w, którego wszystkie wierzchołki mają wagę 1. Da się to zrealizować w czasie wielomianowym.

Jeżeli w grafie istnieje pokrycie wierzchołkowe o rozmiarze <= p, to PWP dla $w_i = 1$ zwróci nam TAK.

Jeżeli WPW odpowiedział nam TAK, to biorąc pod uwagę, że mamy wszystkie wagi ustawione na 1, wiemy że w grafie istnieje pokrycie wierzchołkowe o rozmiarze <= p.

2.1.8

Aby graf był Eulerowski, wszystkie wierzchołki muszą być parzystego stopnia. Wiedząc, że w każdym grafie jest parzysta ilość nieparzystych wierzchołków, możemy dla każdej pary wierzchołków u i v dodać ścieżkę P_2 i połączyć ją z nimi w następujący sposób, tworząc graf G', który będzie Eulerowski:



Dodawanie wierzchołków do grafu da się zrealizować w czasie wielomianowym.

Jeżeli w grafie G istnieje klika o rozmiarze k, to tworząc graf G', który jest spójny **nie naruszamy ω(G)**, ponieważ nie tworzymy żadnego nowego podgrafu pełnego. Jeżeli czarna skrzynka odpowie TAK dla G' i k, to odpowiedź dla G i k również będzie TAK po usunięciu dodatkowych krawędzi i wierzchołków.

2.1.9

Tworzymy G' będący dopełnieniem grafu G, podstawiamy k = n-r.

Da się to zrealizować w czasie wielomianowym, a dokładniej $O(n^2)$.

Niech K będzie kliką w G. Jeżeli w G' zbiór wierzchołków K jest niezależny, to zbiór pozostałych n-|K| wierzchołków, musi pokrywać wszystkie krawędzie w grafie G.

Jeżeli S jest pokryciem wierzchołkowym G', to wtedy V-S musi tworzyć podgraf pełny w G.

2.1.10

```
procedure nonHamiltonian(A[1..n,1..n])
begin
   A2 = array[1..n, 1..n];
   for i=1 to n+1 do
      for j=1 to n+1 do
         if i<n and j<n then A2[i,j] = A[i,j];
         else A2[i,j] = 0;
   return A2;
end
```

Wystarczy, że dodamy do grafu wierzchołek izolowany. Procedura ma złożoność $O(n^2)$.

2.1.11

a)
dane: graf obciążony krawędziowo G, liczba k, próg p
pytanie: „Czy graf G zawiera taki podgraf k-wierzchołkowy, którego wagi krawędzi sumują się do >= p?"

b)
KLIKA:
dane: graf G, liczba k
pytanie: „Czy w grafie G istnieje klika rozmiaru >= k?"
Wystarczy, że stworzymy graf G', będący obciążoną krawędziowo wersją grafu Gm w której każda krawędź ma wagę 1, oraz przyjmiemy p=k(k-1)/2.

c)

Utworzenie grafu G’ oraz ustawienie wag jego krawędzi da się zrobić w czasie $O(n^2)$ dla macierzy sąsiedztwa. Wyznaczenie p=k(k-1)/2 jest O(1). Zatem transformacja jest wielomianowa.

Jeżeli w grafie G istnieje klika rozmiaru k, to w grafie G’ istnieje taki podgraf k-wierzchołkowy, którego suma wag krawędzi jest równa k(k-1)/2, gdyż graf pełny ma n(n-1)/2 krawędzi.

Jeżeli w grafie G’ istnieje podgraf k-wierzchołkowy, którego suma wag krawędzi jest równa k(k-1)/2, to musi to być podgraf pełny k-wierzchołkowy.

2.2.1

a) NPC, zwykły problem 2-podziału

b) NPC, stanowi uogólnienie problemu 2-podziału

c) P, rozdzielamy dwójki na dwa możliwie równe zbiory i uzupełniamy je potem jedynkami, tak aby je wyrównać.

d) P, każdy wyraz ciągu Fibonacciego jest równy sumie dwóch poprzednich.

2.2.2

a) NPC, sprawdzenie czy $\chi(G) = 1$ jest wielomianowe, sprawdzenie czy $\chi(G) = 2$ również jest wielomianowe, gdyż musimy zbadać dwudzielność co da się zrobić w $O(n^2)$, w uogólnionym przypadku O(n+m). Sprawdzenie, czy $\chi(G) = 3$ jest natomiast problemem NPC.

b) NPC, z tych samych powodów co w podpunkcie a)

c) NPH, gdyż sprawdzamy problem k-kliki dla wszystkich wartości od 1 do k.

d) NPC, problem k-kliki należy do problemów NPC.

e) P, maksymalny możliwy podgraf pełny grafu planarnego to K_4. Wystarczy stworzyć nową macierz sąsiedztwa na podstawie orginalnej (również ją trzeba stworzyć w przypadku gdy graf jest w innej strukturze danych), problem sprowadza się do znalezienia 2 dróg długości 2 między 2 wierzchołkami oraz sprawdzenia czy wszystkie wierzchołki są połączone krawędzią w tym zbiorze.

f) P, wystarczy sprawdzić, czy wszystkie jego wierzchołki są parzystego stopnia

2.2.3

a) jeżeli znamy liczbę krawędzi, problem staje się trywialny O(1), i sprowadza się do sprawdzenia czy graf jest pusty. Dla macieczy sąsiedztwa jest on wielomianowy.
b) P, musimy sprawdzić dwudzielność, jeżeli graf nie jest pusty
c) NPC, gdyż problem 3-kolorowalności grafu jest NPC
d) Trywialny, $\chi(G) <= 4$ zachodzi dla każdego grafu planarnego
e) Trywialny, każdy graf koloruje się co najmniej 1 kolorem
f) Trywialny, zachodzi gdy graf nie jest pusty
g) P, zachodzi gdy graf nie jest ani pusty ani dwudzielny
h) co – NPC

2.2.4

a) P != NP b) $\Pi \in P$ c) P = NP d) $\Pi \in NPC$ e) NIC

2.2.5

Wiemy, że SS (SUBSET SUM) należy do NPC. Spróbujmy zredukować problem SS do PRC.

Dane wejściowe dla problemu SS: zbiór A, suma s

Dane wejściowe dla problemu PRC: zbiór Foremek, szerokośc w, wysokość h

Wystarczy, że zbiór liczb A, zamienimy w zbiór A' prostokątów o bokach a_i x 1, za szerokość przyjmiemy s, a za wysokość 1. Stworzenie zbioru prostokątów da się zrealizować w czasie O(n), gdzie n to liczba elementów w zbiorze A.

Jeżeli istnieje suma podzbioru równa s, to tworząc prostokąty o szerokości równej wartości tych liczb oraz wysokości równej 1 jesteśmy w stanie uzyskać prostokąt rozmiarów s x 1.

Jeżeli jesteśmy w stanie uzyskać prostokąt odługości s ze zbioru A', to będziemy w stanie również uzyskać sume s z liczb ze zbioru A.

2.2.6

Wiemy, że SS (SUBSET SUM) należy do NPC. Spróbujmy zredukować problem SS do SP.

Dane wejściowe dla SS: zbiór A, liczba p

Dane wejściowe dla SP: zbiór A, liczba p

Możemy zamienić zbiór A = $\{a_1, a_2, \dots, a_n\}$ w zbiór A' = $\{2^{a1}, \dots, 2^{an}\}$, oraz p w 2^p. Da się to zrealizować w czasie wielomianowym, zakładając z góry ograniczony rozmiar liczb a_i.

Jeżeli w zbiorze A istnieje suma liczb równa p, to w zbiorze A' istnieje taki iloczyn liczb równy 2^p, ponieważ mamy:

$2^{a1} * 2^{a2} \dots = 2^{a1 + a2 + \dots} = 2^p$

Nasz problem jest zachowany, zatem SS α SP, a co za tym idzie: SP ∈ NPC.

2.2.7

W najbardziej pesymistycznym przypadku mamy n produktów po 1 zł każdy. Wybieramy z nich 100. Mamy zatem n!/(100!(n-100)!) możliwości, czyli $O(n^{100})$, zatem problem jest wielomianowy.

2.2.8

Dane: macierz sąsiedztwa grafu G, liczba k

Pytanie: Czy w grafie G istnieje klika rozmiaru >= k?

Losujemy zbiór k wierzchołków grafu G, i sprawdzamy czy z każdy wylosowany wierzchołek jest połączony krawędzią z każdym innym. Da się to zrobić w czasie wielomianowym, a dokładniej $O(n^2)$.

2.2.9

Aby udowodnić, że KLIKA ∈ NPC, możemy zredukować do niego problem ZBIÓR NIEZALEŻNY.

W tym celu tworzymy graf G', będący dopełnieniem grafu G. Da się to zrobić w czasie wielomianowym, a dokładniej $O(n^2)$.

W grafie G istnieje klika o rozmiarze >= k wtedy i tylko wtedy, gdy w dopełnieniu tego grafu istnieje zbiór niezależny o rozmiarze >=k.

ZBIÓR NIEZALEŻNY α KLIKA, czyli KLIKA ∈ NPC

2.2.10

Dane: graf G

Pytanie: Czy w grafie G istnieje cykl Hamiltona?

Lusujemy permutacje zbioru wierzchołków grafu. W czasie wielomianowym O(n), możemy zweryfikować, czy wierzchołki w wylosowanej kolejności tworzą cykl.

Zatem, CYKL HAMILTONA $\in$ NP.

2.2.11

a) P_7

b) W przypadku gdy nasz podgraf będzie zawierał wszystkie wierzchołki grafu, będzie to cykl lub ścieżka Hamiltona, zatem rozwiązanie tego szczególnego przypadku sprowadza się do problemu cyklu Hamiltona, który jak wiemy jest NPC. Wiedząc, że problemy NPH zdefiniowane są jako problemy co najmniej tak trudne jak NPC, dochodzimy do wniosku że MEC(G, 2) jest NPH.

c) Wystarczy znaleźć największą gwiazdę k-barwna, co da się zrobić w czasie wielomianowym, sprawdzając stopień każdego wierzchołka

2.2.12

a) NPC b) P c) NPC d) NPC e) P f) NPC

Dowód dla e:
Problem staje się wielomianowy, gdyż wystarczy znaleźć jego najmniejszy element i obliczyć sumę pozostałych. Rozwiązaniem problemu będzie TAK wtedy i tylko wtedy, gdy pierwszy element będzie różnił się od sumy pozzostałych o więcej niż 3.
Dowód dla f:
Nazwijmy problem o którym mowa 3P (3-podział).
Dodajemy do orginalnego zbioru A element o wartości połowy sumy orginalnego zbioru, tworząc zbiór A', da się to zrealizować w czasie wielomianowym, O(n).
Jeżeli odpowiedź dla 3P będzie TAK, to dla 2P również, bo wiemy że jeden z 3 zbiorów będzie jednoelementowym zbiorem o wartości połowy sumy orginalnego zbioru.
Jeżeli odpowiedź dla 2P będzie TAK, to po dodaniu takiego dodatkowego elementu, odpowiedź dla 3P również będzie TAK.

2.2.13

(a) jest to szczególny przypadek NPC problemu sumy podzbioru, jednakże wiemy że liczba różnych elementów z których się składa zbiór jest ograniczona z góry przez 6. Mamy zatem O(1) kombinacji, czyli nasz algorytm polega na sprawdzeniu ich wszystkich i jest wielomianowy, a dokładniej O(1).

2.2.14

a)

Dane: graf pełny G, liczba k

Pytanie: czy w grafie G istnieje taki cykl Hamiltona, w którym wydatki na paliwo będą <= k?

b)

Ustawiamy wagi wszystkich wierzchołków i krawędzi w grafie G na 1, dodajemy nowe krawędzie, tak aby powstał graf pełny, i ustawiamy ich wagi na 2. Da się to zrealizować w czasie wielomianowym, a dokładnie $O(n^2)$, gdyż do orginalnego grafu G, dodajemy krawędzie dopełnienia tego grafu, tworząc graf G'. Za K przyjmujemy 0, zakładając, że przy powrocie do pierwszego wierzchołka cyklu nie dostaniemy w nim drugi raz nagrody.

W grafie G istnieje cykl Hamiltona wtedy i tylko wtedy gdy w grafie G' istnieje taka trasa, której koszty paliwa będą wynosiły 0, gdyż ilość wierzchołków cyklu jest równa jego ilości krawędzi, a wszystkie wierzchołki jak i krawędzie w grafie G', które należą do G mamy ustawione na 1, w przeciwnym wypadku koszty będą wynosiły n.

c)

Postępujemy podobnie jak w podpunkcie b), aczkolwiek ustawiamy k=1.
-Jeżeli w grafie G istnieje ścieżka Hamiltona, to w grafie G' istnieje cykl Hamiltona o kosztach k=1.
-Jeżeli w grafie G' istnieje cykl Hamiltona o kosztach 1, to w grafie G istnieje ścieżka Hamiltona.

2.2.15

a)

```
procedure LiczbaChromatycznaMPG(G)
begin
   if G is empty then return 1;            //O(n)
   else if G is bipartite then return 2;  //O(n+m)
   else if G is Eulerian then return 3;   //O(n+m)
   else return 4;
end
```

ZO(n) = O(n+m)

b) $K_5 – e$

2.2.16

Dla k=1: P, wystarczy sprawdzić, czy jest to graf pełny.
Złożoność obliczeniowa dla pęków wyjściowych: O(1)
Dla k=2: P, Tworzymy graf G', będący dopełnieniem grafu G. Ilość zbiorów niezależnych w grafie jest równoważna liczbie chromatycznej tego grafu. Sprawdzamy zatem, czy graf da się pokolorować 2 kolorami, co sprowadza się do sprawdzenia jego dwudzielności. Jeżeli TAK, to orginalny graf G da się rozbić na 2 kliki.
Złożoność obliczeniowa dla macierzy sąsiedztwa: $O(n^2)$
Dla k=3: NPC, postępujemy podobnie jak dla k=2, aczkolwiek musimy sprawdzić, czy graf G' daje się pokolorować3 kolorami, a to jest problem NPC.

2.2.17

a)

-wybieramy dowolna liczbę r, gdzie a<=r<=b
-losujemy zbiór wierzchołków grafu G o mocy r
Teraz wystarczy sprawdzić czy każdy z wierzchołków z wylosowanego zbioru jest połączony krawędzią z każdym innym wierzchołkiem w wylosowanym zbiorze, co da się zrealizować w czasie wielomianowym.

b)

problem K-KLIKA jest NPC problemem odpowiadającym na pytanie „Czy w grafie G istnieje klika o rozmiarze k?", jego dane to graf G oraz liczba całkowita dodatnia k<=n.
aby przekształcić dane problemu K-KLIKA na dane do problemu OPK, wystarczy przyjąć a=k oraz b=k, co da się zrobić w czasie wielomianowym, a nawet stałym O(1). OPK zwróci TAK wtedy i tylko wtedy gdy K-KLIKA zwróci TAK.
K-KLIKA α OPK, zatem OPK ∈ NPC

2.2.18

a) trywialne, NIE; graf kubiczny nie może być pusty

b) P, O(n+m); należy sprawdzić dwudzielność algorytmem DFS/BFS

c) P, O(1); należy sprawdzić, czy nie jest to graf K_4

d) trywialne, TAK; graf kubiczny ma liczbe chromatyczną conajwyżej 4

e) trywialne, TAK; każdy graf da się pokolorować co najmniej 1 kolorem

f) trywialne, TAK; graf kubiczny nie może być pusty

g) P, O(n+m); należy sprawdzić dwudzielność algorytmem DFS/BFS

h) P, O(1); należy sprawdzić czy jest to graf K_4

2.2.19

Klasa problemów	Weryfikowalne w czasie wielomianowym	Rozwiązywalne w czasie wielomianowym
P	TAK	TAK
NP	TAK	TAK, NW
NPC	TAK	NW
NPH	TAK, NIE, NW	NW, NIE

2.2.20

1.	Czy A α C? TAK	11.	Czy 2SAT α B? TAK
2.	Czy C α A? NIE	12.	Czy 3SAT α C? BM
3.	Czy F α A? NIE	13.	Czy C α 3SAT? TAK
4.	Czy A α E? TAK	14.	Czy 3SAT α E? TAK
5.	Czy E α A? NIE	15.	Czy E α 3SAT? TAK
6.	Czy C α E? TAK	16.	Czy A ∈ NPC? NIE
7.	Czy C α D? TAK	17.	Czy X ∈ NPC, gdy X α E? BM
8.	Czy E ∈ P? NIE	18.	Czy X ∈ NPC, gdy E α X? BM
9.	Czy D ∈ P? NIE	19.	Czy X ∈ NP, gdy X α C? TAK
10.	Czy E ∈ NP? TAK	20.	Czy X ∈ NP, gdy C α X? BM

2.2.21

Problem polega na ustawieniu wierzchołków w losowej kolejności i zweryfikowaniu w czasie wielomianowym czy tworzą one cykl, i jest to podproblem NPC problemu cyklu Hamiltona, aczkolwiek jest on ograniczony z góry, gdyż zawsze mamy 100! Możliwości, więc rozwiązanie problemu polega na przejrzeniu ich wszystkich, zatem problem jest rozwiązywalny w czasie stałym O(1), czyli należy do P.

2.2.22

Optymalizacyjny problem jest NPH, wówczas gdy jego wersja decyzyjna jest NPC. Przedstawmy problem MLSP w wersji decyzjnej.

Problem: $MLSP_d$

Dane: graf G, liczba k

Pytanie: „czy graf G ma drzewo spinające o <= k liściach?"

Jak wiemy, drzewo spinające to acykliczny podgraf grafu, zawierający wszystkie jego wierzchołki, oraz drzewo o 2 liściach jest ścieżką, zatem $MLSP_d(G, 2)$ będzie równoważne znalezieniu ścieżki Hamiltona w tym grafie, co jest problemem NPC, zatem HPP α $MLSP_d$, czyli $MLSP_d \in NPC$, czyli $MLSP \in NPH$

2.2.23

min/max	P/P	NPH/P	P/NPH	NPH/NPH
Nazwa problemu	SORT	COL	CLIQUE	TSP
Złożoność wersji P	O(nlogn)	O(n)	O(1)	-

SORT – sortujemy tablicę, zarówno dla sortowania niemalejącego jak i nierosnącego jesteśmy w stanie uzyskać złożoność O(nlogn)

COL – problem kolorowania wierzchołkowego, wersja minimalizacyjna jest NPH, wersja maksymalizacyjna polega na kolorowaniu wierzchołków kolejnymi kolorami, co da się zrobić w czasie O(n)

CLIQUE – problem kliki, w wersji maksymalizacyjnej jest on NPH, w wersji minimalizacyjnej, jest O(1), bo minimalna klika w grafie ma zawsze rozmiar 1 (podgraf pełny K_1, czyli pojedynczy wierzchołek)

TSP – Traveling Salesman Problem (problem komiwojażera), zarówno minimalizacyjna jak i maksymalizacyjna wersja tego problemu jest NPH

2.2.24

	Klasa	Problem	Dane	Pytanie
1.	NPC	3SAT	Wyraz Q	Czy Q jest spełnialne?
2.	NPC	2P	Zbiór liczb	Czy istnieje 2-podział?
3.	NPI	Izomorfizm	grafy G_1, G_2	Czy grafy G_1 i G_2 są izomorficzne?
4.	P	Cykl Eulera	graf G	Czy graf G ma cykl Eulera?
5.	P	SORT	Zbiór liczb	Czy liczby są w kolejności niemalejącej?

2.2.25

Ustawiając kolory odpowiednio: $c_1 = c_2 = \ldots = c_k = 1$ oraz przykładowo $c_{k+1} = c_{k+2} = \ldots = 2$ możemy do problemu KK sprowadzić problem Kolorowania Wierzchołkowego (KW), który jak wiemy jest NP-trudny. Z wyżej wymienionym ustawieniem, możemy sprawdzić czy $\chi(G) <= k$, gdyż dzięki temu, że ustawiliśmy kolory od c_1 do c_k to nasz graf będzie k-barwny wtedy i tylko wtedy gdy da się go pokolorować kolorami o łącznych kosztach k.

2.3.1

Najbardziej pesymistyczny przypadek to taki, gdy punkt 1 znajduje się w środku okręgu, utworzonego przez reszte punktów. W takim wypadku maksimum będzie równe ½ średnicy, czyli promieniu, zatem jest to algorytm 2-aproksymacyjny.

2.3.2

Problem polega na wyznaczeniu wszystkich możliwych odległości między punktami, zatem przy n punktach mamy $O(n^2)$ możliwości, czyli problem ten, da się rozwiązać w czasie wielomianowym. Ponadto, jeżeli znamy dokładny algorytm wielomianowy, to jesteśmy również w stanie stworzyć wielomianowy algorytm przybliżony z dowolną dokładnością.

a) TAK b) NIE, bo jest P c) NIE, bo jest P d) TAK

e) TAK f) TAK g) NIE

2.3.3

a) Jeżeli nasze n=3, to SF upakowuje programy optymalnie, dla wartości n>3 może dojść do sytuacji jak np.:

Zbiór programów = (1,1,2,2), L=3

Wynik działania SF: (1,1), (2), jest to nieoptymalne, gdyż widzimy, że zmieściłby się jeszcze jeden program: (1,2), (1,2).

Zatem dla przypadków n>3 algorytm jest 1-absolutnie aproksymacyjny, a co za tym idzie jest on 4/3-aproksymacyjny.

b) $1 <= A_{opt}/A <= 1 + \varepsilon$; $A <= A_{opt} <= A + \varepsilon A$

$\varepsilon A < 1$; $A < n+1$; $\varepsilon < 1/A$; $\varepsilon < 1/(n+1)$, będzie to zawsze ułamkiem. Z racji tego, że działamy na liczbach całkowytych to dostajemy $A = A_{opt}$, czyli znaleźliśmy algorytm wielomianowy rozwiązujący nasz problem odkładnie, co jest nieprawdą, chyba że P=NP.

2.3.4

$A = 8$, $A_{opt} = 6$; $1 <= A/Aopt <= 1 + \varepsilon$; $8/6 <= 1 + \varepsilon$; $1/3 <= \varepsilon$

ε może być większy lub równy 1/3, zatem musimy użyć algorytmu o ε równym co najmniej 1/3, co daje nam złożoność: $O(n\log^3 n)$, gdyż m^m jest stałe w naszym wypadku i wynosi 4.

2.3.5

```
procedure kolorujPlanarny(G)
begin
   if G is empty then return 1;
   else if G is bipartite then return 2;
   else return 4.
end
```

$ZO(n) = O(n^2)$. Sprawdzanie czy graf jest pusty dla macierzy sąsiedztwa ma złożoność $O(n^2)$. Sprawdzanie dwudzielności dla macierzy sąsiedztwa, również jest $O(n^2)$.
Na mocy twierdzenia o 4 barwach graf planarny da się pokolorować conajwyżej 4 kolorami. W najgorszym przypadku graf jest 3-kolorowalny, a algorytm zwróci nam 4. Czyli $A/A_{opt} = 4/3$.

2.3.6

```
procedure indeksChromatyczny(G)
begin
   if G is empty then return 0;                          //O(n+m)
   else if G is P2 then return 1;                        //O(n+m)
   else if G is even Cycle or Path then return 2; //O(n+m)
   else return Δ(G)+1;
end
```

$ZO(n) = O(n+m)$.
Podany algorytm optymalnie koloruje grafy o indeksach chromatycznych 0, 1 oraz 2. W najgorszym przypadku $A = 4$ oraz $A_{opt} = 3$, czyli algorytm jest 4/3-względnie aproksymacyjny. Ponadto, z racji tego, iż algorytm myli się conajwyżej o 1 kolor, jest on 1-absolutnie aproksymacyjny.

2.3.7

Weźmy graf G, oraz graf H = G+G. $A_{opt}(G) = k$, $A_{opt}(H)=2k$.
$|A-A_{opt}| <= 1$; $0 <= A-A_{opt} <= 1$; $A_{opt} <= A <= A_{opt} + 1$;
$A_{opt}(H) <= A(H) <= A_{opt}(H) + 1$; $2k <= A(H) <= 2k + 1$; dzielimy przez 2
$k <= A(H)/2 <= k + ½$;
Z racji tego, że działamy na liczbach całkowitych, otrzymujemy
$A(H)/2 = A(G) = k$, czyli dla przypadku grafu G dostalibyśmy odpowiedź dokładną w czasie wielomianowym, co jest nieprawdą, chyba że P=NP.

2.3.8

$1 <= A/A_{opt} <= 1 + \varepsilon$; $A_{opt} <= A <= A_{opt} + \varepsilon A_{opt}$;
$\varepsilon A_{opt} < 1$; $\varepsilon < 1/A_{opt}$; $\varepsilon < ¼$;
Musimy wziąć zatem ε mniejszego od ¼, np. 1/5.
Dla ε=1/5, obliczenia będą trwały $n^5 = 100^5 = 10\ 000\ 000\ 000$ µs = 10 000s.

2.3.9

```
procedure cubicClique(G, n)
begin
   if n=4 then return 4;    //K4
   else return 2;
end
```

Wiemy, że graf kubiczny ma klikę rozmiaru 4, wtedy i tylko wtedy, gdy jest to graf K_4. Jest to maksymalna klika jaką może mieć graf kubiczny.
Graf kubiczny posiada klikę rozmiaru 3, wtedy i tylko wtedy, jeżeli zawiera trójkąt. Sprawdzenie czy trójkąt jest podgrafem grafu kubicznego dało by się zrealizować w czasie wielomianowym, ale ze względu na porządaną możliwie najlepszą złożoność, ten przypadek pomijamy.
Graf kubiczny musi mieć klikę co najmniej rozmiaru 2, gdyż nie jest on pusty.
$ZO_{cubicClique} = O(1)$. Algorytm jest 1-absolutnie aproksymacyjny oraz 3/2-aproksymacyjny.

2.3.10

a) $2^{k\wedge 3} = n^c$; $k^3 = \log n^c$; $k^3 = c\log n$; $k = (\log n)^{1/3}$;
Są to zatem grafy, które mają $k = (\log n)^{1/3}$.
b) $ZO = O(n^c n^3 \log^3 n)$
c) cykle, ścieżki, drabiny, drzewa
d) $1 <= A/A_{opt} <= 1 + \varepsilon$; $A_{opt} <= A <= A_{opt} + \varepsilon A_{opt}$;
$\varepsilon A_{opt} < 1$; $A_{opt} < m+1$; $\varepsilon < 1/(m+1)$;
Schemat rozwiązywałby problem dokładnie, co jest nieprawdą, chyba że P=NP.

2.3.11

a)
Rozpatrzmy najbardziej pesymistyczny przypadek: dany na wejście digraf D jest już acykliczny, i algorytm podzieli go na 2 równe podgrafy acykliczne. Zwróci nam zatem A = m/2, podczas gdy A_{opt} = m. Mamy zatem $A_{opt}/A = 2$, czyli algorytm jest 2-aproksymacyjny.

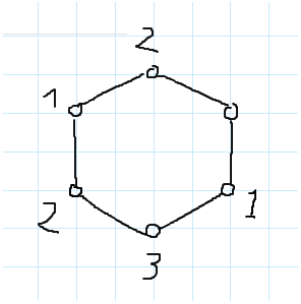
b)
$1 <= A_{opt}/A <= 1 + \varepsilon$; $A <= A_{opt} <= A + \varepsilon A$; $\varepsilon A < 1$; $\varepsilon < 1/A$; $A < m+1$;
$\varepsilon < 1/(m+1)$;
Zatem schemat FPTAS zwracałby wartość dokładną, co jest nieprawdą, chyba że P=NP.

2.3.12

a)
Prawda, jeśli graf zapisany jest w postaci pęków wyjściowych, to jesteśmy wstanie w czasie O(n) sprawdzić Δ(G). Nasz algorytm zwraca Δ(G)+1, gdyż wiemy z twierdzenia Vizinga, że $\Delta(G) <= \chi'(G) <= \Delta(G)+1$.
b)
Prawda, dla Δ(G) = 1 zwraca on 1, dla Δ(G) >= 2, zwraca on Δ(G)+1. Czyli mamy w najgorszym przypadku A=3, A_{opt}=2, $A/A_{opt} = 3/2$.
c)
Prawda, dla Δ(G) = 1 zwraca on 1, dla Δ(G) >= 3, zwraca on Δ(G)+1, a dla Δ(G) = 2 jesteśmy w stanie sprawdzić w wielomianowym czasie, czy graf nie jest cyklem parzystym lub scieżką, gdyż sa to jedyne grafy, których $\chi'(G) = 2$.
d)
Fałsz, chyba, że P=NP. Taki algorytm rozwiązywałby NPC problem 3-kolorowalności krawędziowej w czasie wielomianowym.
e)
Prawda, jeżeli mamy algorytm dokładnie rozwiązujący problem w czasie niewielomianowym, to jesteśmy również w stanie stworzyć niewielomianowy schemat aproksymacyjny. Możemy nawet sam algorytm zwracający dokładne rozwiązanie potraktować jako schemat aproksymacyjny, w którym ε = 0.
f)
Fałsz, chyba że P=NP. Taki schemat rozwiązywałby NPC problem 3-kolorowalności krawędziowej w czasie wielomianowym.

2.3.13

a)
ZO = $\Theta(n^2)$
b)
Nie ważne w jakiej kolejności kolorujemy C4 i C5, zawsze uzyskamy odpowiednio 2 i 3 kolory. W przypadku C6 może dojść do następującej sytuacji:



c)
Dla grafu dwudzielnego pełnego mamy w najgorszym przypadku LF(G) = n/2, podczas gdy $\chi(G) = 2$, zatem ilosć kolorów wyznaczona przez LF będzie rosła w tempie O(n), zatem nie będzie ona stała, czyli algorytm nie jest aproksymacyjny.
d)
Najgorszy przypadek, to gdy nasz graf jest dwudzielny i każdy zbiór niezależny ma 4 wierzchołki, zachodzi wówczas: A = 4, A_{opt} = 2, czyli k = 4/2; l = 2

2.3.14

a)
$1 <= A/A_{opt} <= \varepsilon + 1$; $A_{opt} <= A <= \varepsilon A_{opt} + A_{opt}$;
Musimy dobrać takie ε, żeby zachodziło: $\varepsilon A_{opt} < 1$.
$A_{opt} < n + 1$; $\varepsilon < 1/n+1$; czyli możemy przyjąć: $\varepsilon = 1/n$
Dla takiego ε zawsze otrzymamy $\varepsilon A_{opt} < 1$, czyli $A=A_{opt}$, co oznacza, że schemat zwracałby poprawną wartość w czasie wielomianowym, co jest nieprawdą, chyba że P=NP.
b)
$1 <= A/A_{opt} <= \varepsilon + 1$; $A_{opt} <= A <= \varepsilon A_{opt} + A_{opt}$;
Wiemy, że problem 3-kolorowalności wierzchołkowej jest NPC, możemy przyjąć: $A_{opt} = 3$
$\varepsilon A_{opt} < 1$; $\varepsilon < 1/3$; czyli na przykład: $\varepsilon = ¼$;
$3 <= A <= 3*1/4 + 3$; co daje nam $A = A_{opt}$;
Taki schemat rozwiązywałby problem 3-kolorowalności w czasie wielomianowym, co jest nieprawdą, chyba że P=NP

2.3.15

Z każdą usuniętą krawędzią {u,v} usuwamy 2 wierzchołki u, v, do czasu aż nie zostaną żadne krawędzie. Z racji tego, że każda z krawędzi musi być pokryta, to co najmniej połowa z usuniętych wierzchołków tworzy pokrycie wierzchołkowe. Mamy zatem $A/A_{opt} = 2$.

2.3.16

a)
ZO(n) = O(nlogn)
b)
Wiemy, że po posortowaniu zachodzi: $a_1 >= a_2 >= a_3$, zatem aby otrzymać $A_{opt} = 2A$, musiałoby zachodzić $a_1=p/2$ oraz $a_2 > a_1$, co jest nieprawdą zatem takie liczby dla n = 3 nie mogą istnieć, aczkolwiek może zachodzić:
$a_1 >= a_2 >= a_3$, gdzie $a_1 = p/2 + 1$, oraz $a_2 = p/2$, czyli $A=(p/2 +1)$, $A_{opt} = p$, co w nieskończoności daje nam: $\lim_{p\to\infty} \frac{Aopt}{A} = \lim_{p\to\infty} \frac{p}{\frac{p}{2}+1} = 2$, czyli algorytm jest 2-aproksymacyjny.
c)
Weźmy A={1, p}, gdzie p jest elementem o wartości równej progowi p.
Dla każdego dowolnie dużego p, zwracanym wynikiem będzie 1, zatem
$A = 1$, $A_{opt} = p$. $\lim_{p\to\infty} \frac{1}{p} = 0$
Zatem algorytm SS** nie jest aproksymacyjny.

BG