Rozwiązania zadań z PAA

Dominik Lau

18 stycznia 2023

Spis treści

1 Algorytmy

Karatsuba

a) oszacuj złożoność obliczeniową algorytmu Karatsuby
b) opracowałeś metodę mnożenia dwóch liczb 4-cyfrowych za pomocą 10 mnożeń elementarnych,. Czy twoja metoda jest lepsza od metody klasycznej, czy jest lepsza od algorytmu Karatsuby. Ile mnożeń wykona twoja metoda, metoda Karatsuby i metoda klasyczna dla 8 cyfrowych liczb.

rozwiązanie:
a) $M(n) = 3M(n/2) + n = \theta(n^{lg3})$
b)
metoda klasyczna(4)= 16 mnożeń (gorsza)
metoda karatsuby(4) = 9 mnożeń (lepsza)

dla 8 mnożeń:
moja metoda ok. 32 mnożenia
metoda karatsuby 27 mnożeń
metoda klasyczna 64 mnożenia

Wyszukiwanie sekwencyjne vs binarne

Jak wiele wyszukiwań binarnych trzeba wykonać w najgorszym przypadku danych w posortowanej tablicy, żeby opłacił się czas jej wstępnego posortowania? Przyjmij, że współczynniki proporcjonalności są równe 1.

rozwiązanie:
$T_s = n$ - czas pojedynczego wyszukiwania sekwencyjnego
$T_b = log(n)$ - czas pojedynczego wyszukiwania binarnego
$nlogn + x * logn \leq xn \rightarrow x \geq \frac{nlogn}{n-logn}$
czyli potrzeba $x = \frac{nlogn}{n-logn}$ wyszukań.

Euklides

```
def Euklides1(i,j):
   while i != j:
      if i > j:
         i = i - j
      else:
         j = j - i
   return i

def Euklides2(i,j):
   while i != 0 and j != 0:
      if i > j:
         i = i mod j
      else:
         j = j mod i
   return max{i,j}
```

dla algorytmów Euklides1, Euklides2:

1. Udowodnij poprawność algorytmu
2. Oszacuj pesymistyczna złożoność obliczeniową algorytmu, czy jest to złożoność wielomianowa czy niewielomianowa
3. Oszacuj złożoność pamięciową

rozwiązanie:

Euklides1:

1)
wartości i, j tworzą ciąg malejący, malejący ciąg liczb naturalnych musi być skończony, dlatego algorytm ma własnosc stopu, z własności NWD, NWD(i,j) = NWD(j,i), NWD(i,i) = i oraz NWD(i,j) = NWD(i-j, j), gdzie i > j zatem po każdej iteracji mamy NWD(i,j) = NWD(i-j, j) aż w końcu, gdy i = j to

NWD(i,j) = i = NWD(i_0, j_0)

2)
najwięcej operacji wykona się, gdy i = n a j = 1, bo wtedy będziemy mieli $T(n) = T(n-1) + 1 = O(n)$ kroków, natomiast złożoność obliczeniowa $T(r) = O(2^r)$ gdzie r- liczba cyfr danych wejściowych, jest to złożoność wykładnicza

3)
$M(n) = O(1)$ algorytm potrzebuje stałej dodatkowej pamięci

Euklides2:

1)
wartości i, j tworzą ciąg malejący, malejący ciąg liczb naturalnych musi być skończony, dlatego algorytm ma własnośc stopu„ z własności NWD, NWD(i,j) = NWD(j,i), NWD(i,0) = i oraz NWD(i, j) = NWD(i mod j, j)
przechodzimy przez ciąg przekształceń NWD(i,j) = NWD(i mod j, j) itd. aż dochodzimy do NWD(i,0) = i = NWD(i_0, j_0)

2)

3) $M(n) = O(1)$

Dodawanie wektorów

udowodnij poprawność algorytmu dodawania wektorów A i B

```
def add(A,B):
   C = arr[1..n]
   i = 1
   while i <= n:
      C[i] = A[i] + B[i]
      i += 1
   return C
```

rozwiązanie:

przyjmujemy za niezmiennik $P(k) \iff$ po k-tej iteracji C[1.. k] = A[1..k] + B[1..k]
dla P(1) oczywiste, bo C[1] = A[1] + b[1]
zakładamy P(k), w k+1 iteracji pętli dodajemy C[k+1] = A[k+1] + B[k+1], czyli dostajemy C[1..k+1] = A[1..k+1] + B[1..k+1] $\iff$ $P(k+1)$, udowodniliśmy zatem niezmiennik
po n iteracjach będzie zatem zachodniło P(n), czyli C[1..n] = A[1..n] + B[1..n]

cnd

Największa wartość w wektorze

udowodnij poprawnośc algorytmu znajdowania maksymalnej wartości w wektorze L

```
def max(L[1..n]):
   i = 2
   max = L[1]
   while i <= n:
      if L[i] > max:
         max = L[i]
      i+=1
   return max
```

rozwiązanie:

przyjmujemy za niezmiennik $P(k)$ $\iff$ po k-tej iteracji max = max(L[1..k+1])
zaczynamy od max = L[1] dla pierwszej iteracji mamy, że albo L[2] > max, wówczas max = L[2] i otrzymujemy max = max (L[1..2]) albo L[2] < max, wówczas wciąż max = max(L[1..2]) stąd wynika P(1)
załóżmy P(k), czyli max = max(L[1 ..k+1], w k+1 iteracji mamy, że albo L[k+1] > max albo L[k+1] < max, w drugim przypadku max = max(L[1..k+1]) = max(L[1..k+2]), w drugim przypadku nowy max = L[k+2] > stary max, czyli jest też większy niż wszystkie inne wartości L[1..k+1], zatem mamy max = max(L[1..k+2]), wykazaliśmy więc indukcyjnie P(k)
czyli po n-1 wykonaniach pętli mamy P(n-1) czyli max = max(L[1..n]) = max całego wektora, to jest wartość przez nas zwracana, cnd

Wartość wielomianu

algorytm oblicza wartość wielomianu $p(x) = a_n x^n + a_{n-1}x^{n-1} + ... + a_1 x + a_0$

```
def W(a: coefficients,x: argument)
   p = a[0]
   xpower = 1
   for i = 1 to n:
      xpower = x * xpower
      p = p + a[i] * xpower
   return p
```

1. ile mnożeń trzeba wykonać w najgorszym przypadku, ile dodawań, a ile w przypadku przeciętnym
2. podaj algorytm, który wykona n mnożeń i n dodawań

rozwiązanie:

1)
dla wielomianu W, gdzie st. W = n mamy $mno(n) = 2n$ oraz $dod(n) = n$ zarówno w przypadku pesymistycznym jak i optymistycznym

2)

```
def horner(a: coefficients, x: argument)
   p = 0
   for i = n to 0:
      p = p * x + a[i]
   return p
```

Złożoność obliczeniowa*

Udowodniono, że pewien algorytm ma złożoność $T(n) = \theta(n^{2,5})$. Określ prawdziwość zdań:

1. istnieją c_1, c_2 takie, że dla wszystkich n czas działania A jest krótszy niż $c_1 n^{2,5} + c_2$
2. dla każdego n istnieje zestaw danych rozmiaru n, dla którego czas działania A jest krótszy niż $n^{2,4}$ sekund
3. dla każdego n istnieje zestaw danych rozmiaru n, dla którego czas działania A jest krótszy niż $n^{2,6}$ sekund
4. dla każdego n istnieje zestaw danych rozmiaru n, dla którego czas działania A jest dłuższy niż $n^{2,4}$ sekund
5. dla każdego n istnieje zestaw danych rozmiaru n, dla którego czas działania A jest dłuższy niż $n^{2,6}$ sekund

rozwiązanie:
mamy $T(n) = \theta(n^{2,5}) \rightarrow c_1 n^{2,5} \leq T(n) \leq c_2 n^{2,5}$ dla prawie wszystkich n

1. prawda (np. $c_2 = 0$)
2. fałsz
3. prawda
4. prawda
5. fałsz

Funkcja malejąca zmieniająca znak

dla $f : R^+ \to R$ funkcji malejącej zmieniającej znak znajdź algorytm $o(n)$, który znajdzie największą liczbę naturalną n, dla której $f(n) \geq 0$

rozwiązanie:

```
def rozwiazanie(f):
   i = 1
   j = 1
   while f(j) >= 0:
      j = j * 2

   while j != i + 1:
      p = (i+j)/2
      if p >= 0:
         i = p
      else:
         j = p

   return i
```

złożoność: $O(logn)$

Przesunięcie cykliczne

napisz program, który przesuwa cyklicznie n-elementowy wektor A[1..n] o k pozycji w czasie liniowym, algorytm ma działać in-situ

rozwiązanie:
można zastosować przesuwanie z insertion sorta (przesuwamy k razy in-situ o jedną pozycję w lewo)

```
def shift(A[1..n], k):
   for i = 1 to k:
      current = A[n]
      for j = n to 1:
         swap(A[j], current)
      A[n] = current
```

Ciąg Fibonacciego

Napisz cztery wersje obliczenia n-tego wyrazu ciągu Fibonacciego o liczbie kroków: $O((\frac{1+\sqrt{5}}{2})^n)$, $O(n)$, $O(logn)$, $O(1)$

rozwiązanie:

$O((\frac{1+\sqrt{5}}{2})^n)$

```
def f(n):
   if n == 0 or n == 1:
      return 1
   return F(n-1) + F(n-2)
```

$O(n)$

```
def f(n):
   if n == 0 or n == 1:
      return 1
   x = 1
   y = 1
   for i = 2 to n:
      z = x + y
      y = z
      x = z
   return x
```

$O(logn)$

```
def f(n):
   A = (1 + sqrt(5)) / 2
   B = (1 - sqrt(5)) / 2
   return 1/sqrt(5) * (A^n - B^n)
```

$O(1)$??

Hybryda

Oszacuj liczbę kroków poniższego algorytmu liczącego wartość n-tego wyrazu ciągu fibonacciego. Określ, czy złożoność jest wielomianowa, superwielomianowa, wykładnicza, superwykładnicza

```
def hybryda(n):
   if n < 9:
      if n <= 2:
         return 1
      return hybryda(n-1) + hybryda(n-2)
   else:
      a = b = 1
      for i = 3 to n:
         b = b + a
         a = b - a
      return b
```

rozwiązanie:
nie obchodzi nas n<9

LK(n) = O(n)
ZO(r) = O(2^r) - złożoność jest wykładnicza

Hanoi

Pewien komputer wykonuje milion operacji przenieś z poniższego algorytmu w ciągu sekundy. Dla jakich wartości n będzie on pracował

1. minutę
2. godzinę
3. rok

```
def X(A,B,C,n):
   if n == 1:
      przenies(A,C)
   else:
      X(A,C,B,n-1)
      przenies(A,c)
      X(B,A,C,n-1)
```

---

rozwiązanie:
T(n) = 2T(n-1) + 1 = $\theta(2^n)$
1.
1 000 000 * 60 = 2^n
$n = lg(1000000 * 60)$

2.
$n = lg(1000000 * 60 * 60)$

3.
$n = lg(1000000 * 60 * 60 * 24 * 365)$

Złożoność czasowa

Pewien algorytm A ma złożonosc czasową $\theta(n^2)$. Określ prawdziwość zdań:

1. Istnieją stałe c_1, c_2 takie, że dla wszystkich n czas działania A jest krótszy niż $c_1n^2 + c_2$ sekund
2. Istnieją stałe c_1, c_2 takie, że dla wszystkich n czas działania A jest dłuższy niż $c_1n^2 + c_2$ sekund
3. Istnieją stałe c_1, c_2, c_3 takie, że dla wszystkich n czas działania A jest równy $c_1n^2 + c_2nlogn - c_3n$ sekund
4. dla każdego n istnieje zestaw danych rozmiaru n, dla którego czas działania A jest mniejszy niż $n^{1,9}$ sekund

5. dla każdego n istnieje zestaw danych rozmiaru n, dla którego czas działania A jest mniejszy niż $n^{2,1}$ sekund
6. dla każdego n istnieje zestaw danych rozmiaru n, dla którego czas działania A jest większy niż $n^{1,9}$ sekund
7. dla każdego n istnieje zestaw danych rozmiaru n, dla którego czas działania A jest większy niż $n^{2,1}$ sekund
8. dla pewnych n czas działania A jest równy 2^n sekund

rozwiązanie:

1. prawda
2. prawda
3. fałsz
4. fałsz
5. prawda
6. prawda
7. fałsz
8. prawda

Złożoność pesymistyczna

Algorytm A ma złożoność f(n), algorytm B ma złożoność g(n). Czy to prawda, że (tak,nie, nie wiadomo):

1. Czy w najgorszym przypadku B jest asymptotycznie szybszy od A, jeśli g(n) = $\Omega(f(n)logn)$
2. Czy w najgorszym przypadku B jest asymptotycznie szybszy od A, jeśli g(n) = $O(f(n)logn)$
3. Czy w najgorszym przypadku B jest asymptotycznie szybszy od A, jeśli g(n) = $\theta(f(n)logn)$
4. Czy w najgorszym przypadku B jest asymptotycznie szybszy od A, jeśli g(n) = $\tilde{\theta}(f(n))$
5. Czy w najgorszym przypadku A jest asymptotycznie szybszy od B, jeśli g(n) = $o(f(n)logn)$
6. Czy w najgorszym przypadku A jest asymptotycznie szybszy od B, jeśli g(n) = $\omega(f(n)logn)$

rozwiązanie:

1. nie
2. nie wiadomo
3. nie
4. nie
5. nie wiadomo
6. tak

dwie funkcje

Określ funkcję f jako liniową, wielomianową, superwielomianową, wykładniczą lub superwykładniczą oraz oszacuj jej złożoność.

```
def f(n):
   if n < 3:
      return n
   return f(n-2) + 2 * g(n)

def g(n):
   if n < 3:
      return n
   return 2f(n-2) + g(n / 3)
```

---

rozwiązanie:
$T_f(n) = T_f(n-2) + T_g(n) + 1 = 3T_f(n-2) + T_g(n/3)$
$3T_f(n-2) + 1 \leq T_f(n) \leq 4T_f(n-2) + 1$
$T_f(n) = \Omega((\sqrt{3})^n) \cap O(2^n)$ - liczba kroków
$T_f(r) = \Omega((\sqrt{3})^{2^n}) \cap O(2^{2^n})$ - złożoność obliczeniowa (jest to złożoność superwykładnicza)

Funkcja Padovana

Funkcja Padovana zdefiniowana jest następująco $P(0) = P(1) = P(2) = 1$, $P(n) = P(n-2) + P(n-3)$. Oszacuj tempo wzrostu funkcji P(n). Znajdź jak najlepsze oszacowanie.

rozwiązanie:
Proste oszacowanie $P(n) = \Omega((\sqrt[3]{2})^n) \cap O((\sqrt{2})^n)$
Korzystając z własności funkcji Padovana: $\frac{P(n-2)}{P(n-3)} \leq \frac{4}{3}$
otrzymujemy $P(n) = \Omega((\sqrt{\frac{7}{4}})^n) \cap O((\sqrt[3]{\frac{7}{3}})^n)$

Mediana

Dany jest ciąg n liczb naturalnych z przedziału [1 ... 10n]. Napisz algorytm znajdujący medianę, czyli (n+1)/2 największy element ciągu.

rozwiązanie:

```
def mediana(A[1..n]):
   posortuj_przez_zliczanie(A) // potrzeba nam tablicy
   return A[(n+1)/2]
```

złożoność obliczniowa O(n), pamięciowa O(n)

Własności NWD

Mamy dane własności NWD:

- NWD(a,b) = 2NWD(a/2, b/2) jeśli a,b parzyste
- NWD(a,b/2) jeśli a - nieparzyste, b -parzyste
- NWD((a-b)/2, b) jeśli a i b nieparzyste

napisz algorytm wykorzystujący te własności do obliczenia NWD i oszacuj jego złożoność obliczeniową.

rozwiązanie:

```
def NWD(a,b):
   if a == 0:
      return b
   if b == 0:
      return a

   if even(a) and even(b):
      return 2 * NWD(a/2, b/2)
   if odd(a) and even(b):
      return NWD(a,b/2)
   return NWD((a-b)/2, b)
```

even sprowadza się do sprawdzenia jednego bitu
złożoność obliczeniowa: T(r) = T(r-1) + 1 = O(r) gdzie r- liczba cyfr a i b

Zagadka 1

```
def z(A[1..n])
   x = 0
```

```
   for d = 1 to n:
      for g = d to n:
         suma = 0
         for i = d to g:
         suma = suma + A[i]
      x = max(x, suma)
   return x
```

---

odpowiedz na pytania:

1. jaki jest efekt działania powyższego kodu
2. jaka jest jego złożonośc obliczeniowa
3. napisz program wykonujący to samo zadanie w czasie O(n)

rozwiązanie:
1)
algorytm wyznacza największą sumę spójnego podciągu w tablicy A

2)
$T(n) = O(n^3)$

3)

```
def z2(A[1..n])
   l = A[1]
   s = A[1]

   for i = 2 to n:
      l = max(l + A[i], A[i])
      s = max(s, l)
   return s
```

Zagadka 2

oszacuj złożoność obliczeniową

```
def z(n):
   for i = 1 to n * n:
      j = 1
      while j < sqrt(n):
         j = j + j
```

---

rozwiązanie:

$T(n) = \theta(n^2 lg(\sqrt{n})) = O(n^2 lg(n))$ - liczba kroków
$T(r) = \theta(r2^{2r})$ - złożonośc obliczeniowa (r - rozmiar danych, czyli liczba cyfr)

Zagadka 3

oszacuj złożoność obliczeniową

```
def z(n):
   for i = 1 to n * n:
      k = 1
      l = 1
   while l < n:
      k = k + 2
      l = l + k
```

rozwiązanie:

$T(n) = \theta(n^{2,5})$ - liczba kroków
$T(r) = \theta(2^{2,5r})$ - złożoność obliczeniowa

Zagadka 4

oszacuj złożonośc obliczeniową

```
def z(n)
   for i = n - 1 to 1
      if odd(i) then
         for j = 1 to i:
            pass
         for k = i + 1 to n:
            x = x + 1
```

rozwiązanie:

$T(n) = \theta(n^2)$ - liczba kroków
$T(r) = \theta(2^{2r})$ - złożoność obliczeniowa

Zagadka 5

oszacuj złożoność obliczeniową

```
def z(n):
   for i = n -1 to 1:
      if odd(i)
         for j = 1 to i:
            for k = i+ 1 to n:
               x = x + 1
```

rozwiązanie:

$T(n) = \theta(n^3)$ - liczba kroków
$T(r) = \theta(2^{3r})$ - złożoność obliczeniowa

Zagadka 6

oszacuj złożoność obliczeniową

```
def z(n):
   for i = 1 to n-1:
      for j = i + 1 to n:
         for k = 1 to j:
            pass
```

rozwiązanie:

$T(n) = \theta(n^3)$ - liczba kroków
$T(r) = \theta(2^{3r})$ - złożoność obliczeniowa

Zagadka 7

Co wylicza poniższa funkcja, podaj jej liczbę kroków i złożoność obliczeniową.

```
def f(n):
   if n == 0 or n == 1:
      return 1
   return f(n-1) - f(n-2)
```

rozwiązanie:
wartości funkcji tworzą ciąg postaci (1,1,0,-1,-1,0,1,1,0,...)
liczba kroków: $T(n) = T(n-1) + T(n-2) = \theta(\phi^n)$
złożoność obliczeniowa: $T(r) = \theta(\phi^{2^r})$

Zagadka 8

Podaj liczbę kroków funkcji

```
def z(n):
   L = 0
   for i = 1 to n * n:
      for j = i to n:
         for k = 1 to n * n mod 100:
            L = L + 1
```

rozwiązanie:
wewnętrzna pętla wykonuje się $O(1)$
$T(n) = \theta(n^2)$

Zagadka 9***

Podaj liczbę kroków i złożonośc obliczeniową poniższej funkcji, znajdź jak najlepsze oszacowanie (!).

```
def Fibonacci(n):
   if i <= 2:
      return 1

   for i = 1 to 2^n / n^2:
      pass

   return Fibonacci(n-1) + Fibonacci(n-2)
```

rozwiązanie:
$T(n) = T(n-1) + T(n-2) + \frac{2^n}{n} \leq 2T(n-1) + \frac{2^n}{n} = O(n2^n)$
$T(r) = O(2^r * 2^{2^r})$
lepsze oszacowanie: $T(n) = T(n-1) + T(n-2) + \frac{2^n}{n} \leq ... + 2^n(\frac{1}{n} + \frac{1}{n-1} + \frac{1}{n-2} + ... + 1) \leq 2^n ln(n) \rightarrow T(n) = O(log(n)2^n)$

2 Struktury danych

LSAP*

W historii problemu przydziału dla ważonych grafów dwudzielnych znane są algorytmy o złożonościach: $O(\sqrt{n}Wlog(Cn^2/W)/logn))$, $O(\sqrt{n}mlog(nC))$, $O(n^{3/4}mlogC)$, ?,$O(nmlog(nC)$, $O(n^4)$, $O(n^3)$
przyjmij $C = O(1)$-największa waga krawędzi, $W(n)$ - suma wag
uporządkuj je malejąco

rozwiązanie:
przyjmujemy $m = O(n^2)$, $W(n) = O(n)$
kolejność: ?, $O(n^4)$, $O(nmlognC)$, $O(n^3)$, $O(n^{3/4}mlogC)$, $O(\sqrt(n)mlognC)$, $O(\sqrt{n}Wlog(Cn^2/W)/logn))$

Początkowe wyzerowanie macierzy

Początkowe wyzerowanie macierzy wymaga czasu $O(n^2)$. Podaj metodę, która uniknie początkowego wyzerowania macierzy.

rozwiązanie: tworzymy dwie niezainicjalizowane macierze A(mapa zainicjowanych elementów) i B(wartości zainicjowanych elementów) reprezentujące macierz M, przy odwołaniu do elementu $M[i,j]$ sprawdzamy, czy $A[i,j] = 0$ jeśli tak, to element jest zainicjowany i zwracamy jego wartość $B[i,j]$, jeśli $A[i,j] \neq 0$ to wstawiamy do $B[i,j]0$ i zwracamy 0.

składowe spójności bez DFS/BFS

Napisz algorytm znajdowania składowych spójności w grafie, w którym nie wykorzystuje się DFS ani BFS. Wskazówka: zastosuj mnożenie macierzy.

Rozwiązanie:

```
def skladowe(G[1..n,1..n]):
   S= I[1..n, 1..n] # macierz identycznosci
   W = 0[1..n, 1..n] # macierz zerowa
   for i = 1 to n: #O(n * n^lg7)
      S = S * G
      W = W + S

   # przyporzadkowanie skladowych spojnosci wierzcholkom O(n^2 lgn)
   tablica_skladowych = array of sets [1..n]
   for i = 1 to n:
      skladowe[i].add(i)
      for j = 1 to n:
         # W[i,j] - czy istnieje jakakolwiek sciezka miedzy i oraz j
         if W[i,j] != 0:
            skladowe[i].dodaj(j)

   # usuniecie powtarzajacych sie skladowych O(nlgn)
   skladowe = set of sets
   for i = 1 to n:
      skladowe.add(tablica_skladowych[i])

   return skladowe
```

---

złożoność $T(n) = O(n * n^{lg7})$
zakładam, że dodawanie do zbioru jest realizowane w czasie $O(lgn)$

macierz rozrzedzona

podaj reprezentację wiązaną macierzy, w której występować będą tylko elementy niezerowe

rozwiązanie:
chcąc reprezentować macierz $M[1..n, 1..m]$ tworzymy tablicę $L[1..n]$ list. Element $M[i, j]$ znajduje się w liście $L[i]$ w postaci pary (j, wartość).

merge

napisz algorytm scalania dwóch tablic posortowanych

rozwiązanie:

```
def merge(A,B):
   C = [1.. n+m]
   i = 1
   j = 1
   k = 1

   while i != n+1 or j != n+1:
      if A[i] < B[j]:
         C[k] = A[i]
         k+=1
         i+=1
      else:
         C[k] = B[i]
         k+=1
         j+=1
   if i > j:
      wstaw reszte B do C
   else if i < j:
      wstaw reszte A do C
   return C
```

odwracanie porządku listy

napisz algorytm odwracania porządku elementów listy liniowej i udowodnij jego poprawność

rozwiązanie

```
def reverse(L):
   R = list()
   while not L.empty():
      R.wstaw_na_koniec(L.ostatni)
      L.usun_ostatni()
   return R
```

dowód poprawności:
niezmiennik: $P(k)$ $\iff$ po k-tej iteracji pętli R[1..k] zawiera odwrócone elementy L[(n-k)..n]
$P(1)$ trywialne (R zawiera tylko ostatni element L
załóżmy $P(k)$, zatem R[1..k] zawiera odwrócone elementy L[(n-k)..n]
w następnej iteracji na pozycję R[k+1] wstawiamy ostatni element okrojonej listy L, czyli L[n-k-1]
mamy zatem R[1..k+1] = odwrócone elementy L[n-k-1..n] $\iff$ $P(k+1)$ czyli P(k) jest prawdziwy dla każdego k
po n-iteracjach (n-długość listy) mamy P(n) czyli R[1..n] zawiera odwrócone elementy L[1..n] cnd

Planarny graf dwudzielny

Podaj planamy graf dwudzielny, który nie może być umieszczony na płaszczyźnie w taki sposób, że każda ściana z wyjątkiem zewnętrznej jest wielokątem wypukłym.

rozwiązanie:
Odpadają wszystkie grafy $K_{p,q}$ w których $p \geq 3$ i $q \geq 3$ (bo $K_{3,3}$ jest nieplanarny). Graf, który spełnia treść zadania to na przykład $K_{2,4}$.

Macierzowa reprezentacja grafu z szybkim sprawdzeniem sąsiadów

Zaprojektuj macierzowy sposób reprezentacji grafu nieskierowanego, który: a) w czasie 0(1) umożliwia sprawdzenie, czy dana para wierzchołków u, v jest połączona krawędzią; b) w czasie O(degv) umożliwia przejrzenie wszystkich sąsiadów wierzchołka v. Naszkicuj procedurę boolowską B(u, v), która realizuje punkt (a).

rozwiązanie:
macierz będzie zawierała n+1 kolumn numerowanych od 0 i n wierszy numerowanych od 1. Kolumna 0 będzie zawierała informację, o ile ma przeskoczyć j, żeby trafić na następnego sąsiada. Wszystkie pozostałe komórki też będą zawierały tą informację. Jeśli dwa wierzchołki nie sąsiadują ze sobą, w komórce ma być 0. Jeśli w komórce znajduje się ostatni sąsiad to jej wartość powinna wynosić -1.

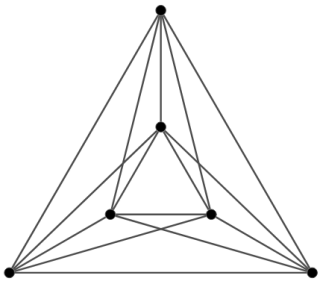
```
def hasEdge(M,i,j):
   if M[i,j] = 0:
      return false
   return true

def countDeg(M, i):
   deg = 0
   j = 0
   while true:
      if M[i,j] == -1 break
      j += M[i,j]
   return deg
```

Liczba przecięć K_6

Udowodnij, że $\xi(K_6) = 3$

rozwiązanie:

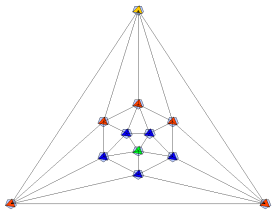


z rysunku wynika, że $\xi(K_6) \leq 3$
Załóżmy zatem, że $\xi(K_6) = 2$, oba przecięcia dotyczą czterech różnych wierzchołków. Usuńmy zatem jeden z wierzchołków, wówczas otrzymujemy graf K_5 bez przecięć co jest sprzecznością. Zatem $\xi(K_6) = 3$

Graf planarny 5-regularny

narysuj graf planarny 5-regularny

rozwiązanie:
jest to graf dwunastościan:



warto zauważyć, że wszystkie grafy platońskie są planarne

Umieszczanie grafów

Udowodnij, że graf może być umieszczony na płaszczyźnie $\iff$ może być umieszczony na powierzchni kuli.

rozwiązanie:

($\longrightarrow$)
skoro graf może być umieszczony na płaszczyźnie to $\xi(G) = 0$ natomiast genus grafu ograniczony jest od góry $g \leq \xi(G) \rightarrow g = 0$ zatem można go również umieścić na kuli, która ma genus = 0

($\longleftarrow$)
skoro graf możemy umieścić na kuli, która jest powierzchnią bez rączek to

znaczy, że nie ma przecięć więc można go również umieścić na płaszczyźnie

inne rozwiązanie:

każdą sferę można zrzutować na płaszczyznę z wyjątkiem jednego punktu (bieguna)

Nierówności trójkąta

Dany jest zbiór n liczb. Sprawdź czy w tym zbiorze są takie 3, które mogą być długościami boków trójkąta. Algorytm powinien mieć złożoność $o(n^2)$.

rozwiązanie:

```
def triangle_inequality(A[1..n]):
   A = sort(A)
   c1 = A[1]
   c2 = A[2]
   for i = 3 to n:
      if c1 + c2 > A[i]
         return True
      else:
         c1 = A[i]
         swap(c1,c2)
   return False
```

złożoność $O(nlogn)$

Najkrótszy cykl w grafie dwudzielnym

Znajdź algorytm, który w grafie $K_{p,q}$ $(p, q \geq 1)$ znajdzie najkrótszy cykl i oszacuj jego złożoność obliczeniową. (rozważ przypadek a) lista sąsiedztwa, b) macierz sąsiedztwa) Dlaczego jest ona mniejsza niż złożoność pamięciowa?

rozwiązanie:

```
def cykle(G):
   u1 = sasiad1(v)
   u2 = sasiad2(v)
   u3 = niesasiad(v)
   return (v, u1, u3, u2, v)
```

a)
złożoność obliczeniowa O(n) - przejrzenie sąsiedztwa v, mniejsze niż n + m

b)

złożoność obliczeniowa O(n) mniejsze niż n^2

złożoności obliczeniowe są mniejsze niż złożoność pamięciowa, bo zbiór danych nie składa się z samych danych istotnych

Harmoniczne kolorowanie

Harmoniczne kolorowanie - to takie pokolorowanie grafu, w którym:

- sąsiednie wierzchołki mają różne kolory
- dowolne dwie krawędzie mają różne pary kolorów

Minimalną ilość kolorów do pokolorowania harmonicznie grafu oznaczamy $h(G)$ albo $\chi_H(G)$.
Efektywną metodą na przechowywanie struktury grafu rzadkiego jest pokolorowanie go harmonicznie. Zapamiętujemy strukturę w postaci wektora kolorów W, gdzie $w_i \in W$ to kolor i-tego wierzchołka. Obok wektora zapamiętujemy macierz C o rozmiarze $h(G) \times h(G)$, w której $c_{i,j} = (u, v)$ gdy uv jest krawędzią o końcach pomalowanych kolorem i i kolorem j, w przeciwnym wypadku $c_{i,j} = 0$. Wiedząc, że $\sqrt{2m} < h(G) \leq n$ oszacuj:

1. złożoność czasową procedury B(u,v)
2. złożoność pamięciową macierzy W i C

rozwiązanie:

1)

```
def B(v,u, W, C):
   kolor1 = W[v]
   kolor2 = W[v]
   if C[kolor1, kolor2] == 0:
      return false
   return true
```

złożoność O(1)

2)
$M_W(n) = \theta(n)$
$M_C(n) = \Omega(m) \cap O(n^2)$

Znajdowanie drogi długości k

Posortuj złożoności malejąco dla algorytmu znajdowania drogi długości k w nieobciążonym grafie n-wierzchołkowym.
$O(4^k n^{O(1)})$, $O(k! n^{O(1)})$, $O(1,66^k n^{O(1)})$, $O((2e)^k n^{O(1)})$, $O(1,66^n n^{O(1)})$, $O(2^{3k/2} n^{O(1)})$, $O(2^k n^{O(1)})$

rozwiązanie:
ograniczenie górne na k: $k \leq n-1$
$O(k!n^{O(1)})$, $O((2e)^k n^{O(1)})$, $O(4^k n^{O(1)})$, $O(2^{3k/2} n^{O(1)})$, $O(2^k n^{O(1)})$, $O(1,66^n n^{O(1)})$, $O(1,66^k n^{O(1)})$

LGS

Naszkicuj program lgs zwracający liczbę gwiazd spinających zawartych w G. Oszacuj jego złożoność w zależności od m i n dla macierzy sąsiedztwa, listy sąsiedztwa, pęków wyjściowych.

rozwiązanie:

```
def lgs(G):
   l = 0
   for v in G:
      if |N(v)| = n-1:
         l += 1
   return l
```

dla macierzy sąsiedztwa $O(n^2)$, dla listy sąsiedztwa $O(n+m)$, dla pęków $O(n)$

Cykl

Napisz program, który stwierdza, czy graf G zapisany w macierzy sąsiedztwa wierzchołków jest cyklem i oszacuj jego złożoność obliczeniową.

rozwiązanie:

```
def is_cycle(G[1..n][1..n]):
   odwiedzone = tablica[1..n]
   licznik_odwiedzone = 0
   wyzeruj(odwiedzone)
   obecny_wierzcholek = 1
   while licznik_odwiedzone != n:
      odwiedzone[obecny_wierzcholek] = 1
      licznik_odwiedzone += 1
      licznik_odwiedzone =
      licznik_sasiadow = 0
      do_odwiedzenia = -1
      for i = 1 to n:
         if B(obecny_wierzcholek, i):
            licznik_sasiadow += 1
            if not odwiedzone[i]:
               do_odwiedzenia = i
```

```
    if B(obecny_wierzcholek, 0) and licznik_odwiedzone = n:
      return true
    if do_odwiedzenia = -1:
      return false
    if licznik_sasiadow != 2:
      return false

  return false
```

Mnożenie hybrydowe macierzy

Załóżmy, że mamy dwa algorytmy mnożenia macierzy, pierwszy wykonuje 22 mnożenia na macierzach 3x3, drugi 99 mnożeń na macierzach 5x5, którym algorytmem najlepiej jest pomnożyć macierze 15x15.

Rozwiązanie:
Najlepiej użyć metody hybrydowej, najpierw rozbijamy macierz na 3x3 macierzy 5x5, mnożymy pierwszym algorytmem a potem drugim, czyli mamy $22 * 99$ mnożeń.

3 Problemy

klika o rozmiarze $\leq$ k

Mamy algorytm, który odpowiada na pytanie, czy graf G zawiera klikę $\leq$ k jeśli G ma gwiazdę spinającą. Jak wykorzystać ten algorytm dla grafu, który nie ma gwiazdy spinającej?

rozwiązanie:
Chcemy się dowiedzieć, czy graf ma klikę $\leq$ k.
Dokładamy do G gwiazdę spinającą, następnie pytamy o to, czy powstały graf ma klikę $\leq$ k-1.

konwersja do 3CNF

Sprowadź podane wyrażenia do 3CNF

1. $x_1 + \overline{x}_2$
2. x_1
3. $x_1 + x_2 + \overline{x}_3 + x_4$
4. $x_1 + x_2 + x_3 + x_4 + x_5$

rozwiązanie:
1) $x_1 + \overline{x}_2 = (x_1 + \overline{x}_2 + y)(x_1 + \overline{x}_2 + \overline{y})$
2) $x_1 = (x_1 + y)(x_1 + \overline{y}) = (x_1 + y + z)(x_1 + y + \overline{z})(x_1 + \overline{y} + z)(x_1 + \overline{y} + \overline{z})$
3) $x_1 + x_2 + \overline{x}_3 + x_4 = (x_1 + x_2 + y)(\overline{x}_3 + x_4 + \overline{y})$
4) $x_1 + x_2 + x_3 + x_4 + x_5 = (x_1 + x_2 + y)(x_3 + x_4 + x_5 + \overline{y}) = (x_1 + x_2 + y)(x_3 + x_4 + \overline{z})(x_5 + \overline{y} + z)$

UWAGA!
w podpunkcie 3 i 4 zastosowaliśmy trik, niech ϕ będzie dowolną formułą logiczną DNF o literałach a, b, c, d to jest $\phi = a + b + c + d$
wówczas $\psi = (a + b + y)(c + d + \overline{y})$ jest równoważne w kontekście spełnialności do ϕ. Innymi słowy zawsze zachodzi SAT(ψ) = SAT(ϕ)

KLIKA $\in$ NPC

Udowodnij, że KLIKA $\in$ NPC. Użyj problemu CNF-SAT.

rozwiązanie:
1)
KLIKA $\in$ NP - mając dowolny podgraf n_1 wierzchołkowy w czasie n_1^2 jesteśmy w stanie sprawdzić, czy jest kliką

2)
T: CNF-SAT α KLIKA
D:
budujemy graf G

- jego wierzchołki to zmienne należące do poszczególnych klauzul formuły
- wierzchołki w obrębie jednej klauzuli są niezależne
- zmienne zanegowane występujące w różnych formułach (czyli np. $x, \neg x$) są niezależne między sobą
- pozostałe wierzchołki połączone są krawędziami

CNF-SAT(formuła) = KLIKA(G,Nklauzul)

(a) jeśli formuła nie jest spełnialna, czyli CNF-SAT(formuła) = NIE, to znaczy, że w grafie G nie ma kliki łączącej wszystkie klauzule-podzbiory, bo gdyby była, to dla pewnego wartościowania 0,1 wierzchołków do niej należących wartościowanie formuły wynosiłoby 1 co jest sprzeczne z założeniem, a zatem KLIKA(formuła, Nklauzul) = NIE
(b) jeśli formuła jest spełnialna, czyli CNF-SAT(formuła) = TAK, to znaczy, że w grafie G wszystkie wierzchołki należące do odpowiedniego wartościowania są ze sobą połączone i stanowią klikę, a zatem KLIKA(formuła, Nklauzul) = TAK

PW $\in$ NPC

Udowodnij, że PW (Pokrycie wierzchołkowe) $\in$ NPC. Użyj problemu KLIKA.

rozwiązanie:
1)
PW $\in$ NP - mając dowolny $S \subseteq V$ w czasie wielomianowym jesteśmy w stanie sprawdzić, czy jest pokryciem wierzchołkowym - lecimy po krawędziach i sprawdzdamy czy dla $\{v, w\}$ v lub $w \in S$, jeśli tak dla każdej krawędzi, to S jest pokryciem wierzchołkowym, w przeciwnym wypadku nim nie jest

2)
T: KLIKA α PW
D:
KLIKA(G,k) = PW(G', n-k)
(a) jeśli G ma klikę k-elementową (KLIKA(G, k) = TAK), to jego dopełnienie ma k-elementowy zbiór wierzchołków niezależnych, zatem żeby pokryć wszystkie krawędzie w najgorszym wypadku będziemy musieli w pokryciu umieścić wszystkie pozostałe wierzchołki (poza tymi niezależnymi) czyli PW(G', n-k) = TAK
(b) jeśli G nie ma kliki k-elementowej (KLIKA(G, k) = NIE) to w jego dopełnieniu każde k wierzchołków jest połączonych przynajmniej jedną krawędzią a co za tym idzie nie może być pokrycia n-k elementowego (bo jedna krawędź by została bez kolorowego wierzchołka), czyli PW(G', n - k) = NIE

Ważone pokrycie wierzchołkowe

Udowodnij, że problem WPW (Ważonego pokrycia wierzchołkowego) $\in$ NPC. WPW definiujemy tak:

- Dane wejściowe: graf G z obciążonymi wierzchołkami, liczba p $\in \mathbb{N}$
- Pytanie: Czy G zawiera pokrycie wierzchołkowe o wadze $\leq$ p

rozwiązanie:

1)
WPW $\in$ NP - mając dowolne S $\subseteq V$ jesteśmy w stanie zweryfikować w czasie wielomianowym czy waga sumuje się do p, a także, zweryfikować, czy S jest pokryciem

2)
T: PW α WPW
D:
PW(G,k) = WPW(G_w ,czyli G z wagami = 1 na każdym wierzchołku, k)
(a) jeśli graf ma pokrycie wierzchołkowe rozmiaru $\leq$ k, to G_w ma pokrycie wierzchołkowe o łącznej wadze $\leq k$, czyli oba problemy odpowiadają TAK

(b) jeśli graf nie posiada pokrycia wierzchołkowego rozmiaru $\leq$ k, to G_w również nie ma pokrycia o łącznej wadze $\leq k$, czyli dla obu problemów mamy NIE.

SORT α MM

Udowodnij, że SORT α MM.
SORT definujemy tak:

- Ciąg A n liczb
- Pytanie: Czy A jest rosnący?

MM definiujemy tak:

- Dane wejściowe: macierze A,B,C
- Pytanie: Czy $A \times B = C$

rozwiązanie:
zauważmy, że SORT da się rozwiązać wielomianowo, zatem nasza funkcja przekształcająca dane wejściowe będzie miała postać

```
SORT(A,n):
   if posortowany(A):
      MM(matrix(0), matrix(0), matrix(0))
   else:
      MM(matrix(0), matrix(0), matrix(1))
```

(a) jeśli SORT = TAK, to MM = TAK, ponieważ matrix(0) $\times$ matrix(0) = matrix(0)
(b) jeśli SORT = NIE, to MM = NIE, ponieważ matrix(0) $\times$ matrix(0) $\neq$ matrix(1)

Gwiazda spinająca i klika

Marek ma magiczną skrzynkę rozwiązującą problem k-kliki ale tylko, gdy w grafie jest gwiazda spinająca. Jak Andrzej ma zmienić swój graf niezawierający gwiazdy spinającej, żeby móc skorzystać ze skrzynki Marka?

rozwiązanie:

```
def Andrzej(G,k):
   G2 = G + gwiazda_spinajaca
   return Marek(G2, k+1)
```

(*) dodając gwiazdę spinającą zwiększamy rozmiar każdej kliki o 1

(a) jeśli Marek(G2, k+1) daje TAK, to oznacza, że w G2 mamy k+1-klikę zatem w G mamy k-klikę zgodnie z (*), czyli Andrzej(G,k) daje TAK
(b) jeśli Marek(G2, K+1) daje NIE, to oznacza, że w G2 nie mamy k+1-kliki, a zatem z faktu (*) w grafie G nie ma k-kliki, czyli Andrzej(G,k) daje NIE

czyli problem zostaje zachowany

Klika dec

Masz pudełko Klika dec, które dla wejściowego grafu G odpowiada TAK,NIE na problem KLIKA (dane wejściowe: G, próg p) w czasie O(n). Jak użyjesz go do znalezienia wierzchołków maksymalnej kliki. Złożoność algorytmu.

Rozwiązanie:

```
def w_kliki(G):
   omega = metoda_bisekcji_wyznacz_k_najwiekszej_kliki(G) #O(nlogn)

   for v in G: # O(n^2)
      G = G - v
      if Klika_dec(G) != omega:
         G = G + v
   return G
```

$T = O(n^2)$

4 Algorytmy aproksymacyjne

Kolorowanie wierzchołków

Dla poniższych algorytmów wymień, które grafy koloruje optymalnie, a dla których się myli.

1. LF
2. SL
3. SLF

rozwiązanie:
1)
optymalnie: K_n, $K_{p,q}$
myli się: P_6, koperta, J_n - graf Johnsona

2)
optymalnie: W_n, C_n, J_n, drzewa, grafy planarne, grafy Mycielskiego
myli się: grafy dwudzielne, grafy Colemena-Moore'a, pryzma, pryzmatoid

3)
optymalnie: dwudzielne, w tym J_n, drzewa, C_n, W_n, kaktusy
myli się: $K_{p,q,r}$

Znajdowanie klik w grafie kubicznym

Zaprojektuj algorytm 1-absolutnie aproksymacyjny znajdujący największą klikę w n-wierzchołkowym grafie kubicznym. Algorytm powinien mieć złożoność $O(n)$

rozwiązanie:
Algorytm k-absolutnie aproksymacyjny -> algorytm taki, że dla danych I, gdzie OPT(I) - optymalny wynik, mamy $|A(I) - OPT(I)| \leq 1$, czyli musimy znaleźć algorytm, który będzie mógł się pomylić o 1 w zwracaniu rozmiaru kliki. Oto on:

```
def clique(G):
   if n == 4:
      return {v1, v2, v3, v4}
   else:
      u = dowolnysasiad(v1)
      return {v1, u}
```

Czyli algorytm zwraca klikę K_4 lub K_2, możliwe jest, że w grafie występuje K_3 ale chcemy stworzyć algorytm aproksymacyjny, więc możemy się pomylić o 1. Algorytm ma złożoność $O(n)$, bo dowolnysasiad(v1) działa w czasie $O(n)$. Algorytm ma złożoność mniejszą niż złożoność pamięciowa, bo niewszystkie dane w macierzy sąsiedztwa reprezentującej graf są danymi istotnymi dla wyniku.

Problem komiwojażera

Udowodnij, że jeśli P $\neq$ NP to problem komiwojażera nie ma wielomianowego algorytmu względnie aproksymacyjnego.

rozwiązanie:
Załóżmy, że isnieje taki algorytm k-aproksymacyjny - użyjemy go do rozwiązania problemu Ścieżki Hamiltona. Dla pewnego grafu G dodajemy wagę 1 do jego krawędzi, następnie tworzymy $\bar{G}$ i dodajemy do jego krawędzi wagę kn, scalamy te grafy i otrzymujemy graf pełny G^*. Dla G^* uruchamiamy nasz algorytm. Wiemy, że jeśli G ma ścieżkę Hamiltona, to $OPT(G^*) = n$, w przeciwnym wypadku $OPT(G^*) > kn$ natomiast nasz algorytm zwraca $A(G^*) \leq kn$, więc sprawdzając, czy nasz algorytm zwrócił $\leq kn$ możemy stwierdzić w czasie P, że graf posiada cykl Hamiltona lub nie $\rightarrow$ P = NP, co jest sprzeczne z zał.

Pokrycie wierzchołkowe

Dla Pokrycia wierzchołkowego, gdzie k oznacza maksymalny rozmiar pokrycia:

1. udowodnij, że problem jest wielomianowy dla dowolnego ustalonego k
2. zaprojektuj algorytm wielomianowy dla k = 1
3. Udowodnij, że optymalizacyjna wersja PW nie ma algorytmu wielomianowego 1-absolutnie aproksymacyjnego (chyba, że P = NP)
4. Udowodnij, że optymalizacyjna wersja PW nie ma schematu FPTAS

rozwiązanie:
(1.)
rozwiązanie takiego problemu można przeprowadzić poprzez sprawdzenie wszystkich kombinacji tego, które wierzchołki są w pokryciu, a które nie, co można zrobić w czasie $O(n^k)$ i sprawdzenie dla każdego, czy pokrywa cały zbiór krawędzi, co mozna zrobić w czasie $O(m)$, zatem przy ustalonym k mamy problem wielomianowy

(2.)

```
def coverWithOne(G):
   for i = 1 to n:
      if vi covers entire E(G):
         return True
   return False
```

złożoność to O(nm)

(3.)
załóżmy, że istnieje taki algorytm A, powiedzmy, że mamy G taki, że pw(G) = k, skonstruujemy na jego podstawie graf $G^* = G \cup G$, dla którego pw(G^*) = 2k, ponieważ musimy pokryć obie składowe spójności. Poniższy algorytm umożliwia nam dokładne rozwiązanie problemu PW.

```
def ExactPolynomialVertexCover(G):
   Gstar = G U G
   a = A(Gstar) # 2k v 2k+1
   return floor(a/2)
```

dokładne rozwiązanie NP-trudnego problemu PW w czasie wielomianowym $\rightarrow$ P = NP, co jest sprzeczne z założeniem.

(4.)
załóżmy, że taki schemat FPTAS A(ε) istnieje, przyjmijmy $\varepsilon = \frac{1}{n+1}$, czyli nasz algorytm będzie wielomianowy, przyjąwszy to ε, otrzymujemy $\frac{A}{OPT} \leq 1 + \frac{1}{n+1}$ natomiast OPT $\leq$ n (z faktu, że jest to pokrycie wierzchołkowe), czyli

$A \leq OPT + \frac{OPT}{n+1} < OPT + 1$ więc A daje nam wynik dokładny, czyli w czasie wielomianowym rozwiązujemy NPC problem PW $\rightarrow$ P = NP sprzeczność

MΔST

Problem optymalizacyjny MΔST szuka drzewa spinającego o minimalnej Δ-cie. Pokaż, że problem jest NPH.

rozwiązanie:
dla $\Delta = 2$ problem jest odpowiednikiem ścieżki Hamiltona więc jest przynajmniej tak samo trudny jak SH, czyli jest NPH.

Dokładna liczba chromatyczna

Masz 100-wierzchołkowy graf G, a ja posiadam schemat PTAS, który działa w czasie $O(n^{\frac{1}{\varepsilon}})\mu$s. Chcesz stwierdzić, czy $\chi(G) = 3$, jak skorzystasz z mojego schematu i jak długo sekund będą trwały Twoje obliczenia?

rozwiązanie:
skorzystać można poprzez odpowiednie ustalenie ε: $a - \chi < 1 \rightarrow a < 1 + \chi$, stąd mamy $\frac{a}{\chi} \leq 1 + \varepsilon < \frac{1+\chi}{\chi} \rightarrow \varepsilon < \frac{1}{\chi}$ wówczas dla ustalnonego χ otrzymujemy dokładny wynik, zatem przyjmujemy $\chi = 3 \rightarrow \varepsilon < \frac{1}{3}$ czyli np. $\varepsilon = \frac{1}{4}$. Wówczas T = 100^4 μs = 100 s.

ODS

Problem OGraniczone drzewo spinające ODS(G,k) pyta: czy w grafie G można znaleźć drzewo spinające o maksymalnym stopniu k. Udowodnij, że

1. Graf półhamiltonowski α ODS(G,k)
2. jeśli P $\neq$ NP to dla każdego $\varepsilon < 1.5$ nie istnieje wielomianowy algorytm ε-przybliżony dla znajdowania minimalnego ODS

rozwiązanie:
(1.)

```
def SH(G):
   return ODS(G,2)
```

dla k = 2 ODS pyta, czy graf ma ścieżkę hamiltona

(2.)
załóżmy, że istnieje taki algorytm aproksymacyjny A, za pomocą poniższego algorytmu można rozwiązać w wielomianowym czasie NPC problem ścieżki hamiltona.

```
def SH(G):
   k = A(G)
   if k == 2:
      return 'TAK'
   return 'NIE'
```

W powyższym programie wychodzimy z faktu $\frac{A(G)}{OPT(G)} < \frac{3}{2}$, w szczególności dla $OPT(G) = 2$ mamy $\frac{A(G)}{2} < \frac{3}{2} \rightarrow A(G) < 3$ czyli gdy A daje wynik 2, to znaczy, że znajdujemy rozwiązanie optymalne. Zatem mamy rozwiązanie NPC problemu ścieżki hamiltona w czasie P, czyli P = NP, co jest sprzeczne z założeniami cnd..

PSK

Problem selektywnego komiwojażera to taka odmiana problemu komiwojażera, w której nie musi odwiedzić wszystkich miast. Mamy obciążony krawędziowo (odległości) graf pełny G i nieujemny zysk za odwiedziny każdego wierzchołka, chcemy znaleźć cykl odwiedzający każdy wierzchołek co najwyżej raz, który maksymalizuje zysk (po odliczeniu kosztów).

1. pokaż, że dla decyzyjnej wersji problem selektywnego komiwojażera (PSK) PSK $\in$ NPC
2. podaj przypadek szczególny problemu selektywnego komiwojażera, który można rozwiązać liniowo. Przedstaw odpowiedni algorytm.

rozwiązanie:
(1.)
PSK w wersji decyzyjnej
Dane: ważony G, nagroda za odwiedzenie wierzchołka, próg p
Pytanie: czy w G jest taki cykl selektywnego komiwojażera, że jego zysk $\geq$ p?

zredukujemy cykl hamiltona do PSK

```
def CH(G):
   for {u,v} in V(G):
      if {u,v} in E(G):
         waga({u,v}) = 0
      else:
         E(G) = E(G) U {u,v}
         waga({u,v}) = inf
   zysk = 1
   prog = n
   return PZK(G,zysk,prog)
```

jest to transformacja wielomianowa, bo dodawanie wag itp. to jest góra $O(n^2)$, ponadto zachowuje problem, bo jeśli G ma cykl hamiltona to ważony G będzie miał taki cykl, który idzie po wszystkich wierzchołkach (nie wchodzi na żadną

"nielegalną"krawędź), czyli otrzymuje zysk w wysokości = n. Jeśli G nie ma cyklu hamiltona, to żeby przejść po wszystkich wierzchołkach w celu uzyskania zysku wysokości n komiwojażer będzie musiał skorzystać z niedozwolonej krawędzi tym samym redukując swoje zyski do -inf.

(2.)
przypadek szczególny to gdy wagi są dużo większe od zysku. Wówczas znajdujemy tylko wierzchołek o maksymalnym zysku i się z niego nie ruszamy.

Problem podgrafu

Wiadomo, że sprawdzenie czy dla danej pary grafów (H,G) pierwszy z nich jest podgrafem drugiego stanowi problem NP-zupełny. Określ status (wielomianowy lub NP-zupełny) podproglemów tego zagadnienia.

1. H jest grafem pełnym
2. G jest grafem pełnym
3. H jest gwiazdą
4. H jest ścieżką
5. H jest K_3
6. H jest 1-regularny
7. G jest 1-regularny
8. H jest cyklem
9. G jest $K_{3,3}$

rozwiązanie:

1. $\in$ NPC (problem kliki)
2. $\in$ P (jeśli G ma więcej wierzchołków niż H to znaczy, że H jest jego podgrafem)
3. $\in$ P (czy jest wierzchołek stopnia $n(H) - 1$)
4. $\in$ NPC (dla n(H) = n(G) jest to problem ścieżki Hamiltona)
5. $\in$ P (szukamy trójkątów w grafie, mnożenie macierzy tego typu)
6. $\in$ P, patrzymy, czy G ma $\frac{n(H)}{2}$ niezależnych krawędzi (?, być może jednak NPC)
7. $\in$ P
8. $\in$ NPC (dla n(H) = n(G) jest to problem cyklu Hamiltona)
9. $\in$ P (ograniczona liczba możliwych podgrafów)

Mapa

Pewne państwo składa się z n prowincji (rysunek powyżej). Przywódca państwa rezydując w stolicy s, ma zamiar odwiedzić wszystkie stolice minimalizując koszty. Czy jest to problem NP-trudny czy wielomianowy? Uzasadnij.

Rozwiązanie:
Jest to problem wielomianowy - podproblem TSP na grafie W_n, który można rozwiązać w czasie $O(n^2)$. Znajdujemy parę najmniejszych krawędzi "sąsiednich"(o minimalnej sumie wag) z s to pewnych dwóch wierzchołków i okrążamy wszystkie miasta.

Minimalne rozcięcie

Problem Minimalne Rozcięcie pyta jak równo podzielić wierzchołki grafu o n parzystym, tak aby zminimalizować liczbę krawędzi łączących wierzchołki z różnych podzbiorów. Najszybszy znany algorytm dla problemu MR ma złożoność $O(2^{kkk}n^3log^3n)$ gdzie k jest rozmiarem minimalnego cięcia. Pokaż, że problem jest NP-trudny dla nieparzystej liczby wierzchołków. Dla jakich klas grafów jest on wielomianowy a dla jakich wykładniczy.

Rozwiązanie:
(I)
Do grafu o parzystej liczbie wierzchołków dokładamy wierzchołek izolowany i otrzymujemy tak samo trudny problem.

(II)
drzewa, ścieżki, cykle

(III)
grafy pełne, grafy dwudzielne pełne, koła, gwiazdy

Ponownie klika

Znajdź algorytm k-absolutnie aproksymacyjnym znajdowania w grafie planarnym spójnym G (zapisanym w macierzy sąsiedztwa) wierzchołków maksymalnej kliki. Algorytm ma mieć

1. minimalną złożoność

2. minimalne k

rozwiązanie:
(1.)

```
def Alg1(G):
   return [1]
```

złożoność O(1), aproksymacja k = 3
(2.)

```
def Alg2(G):
   G2 = G * G

   for i = 1 to n:
      for j = 1 to n:
         if G[i,j] == 1 and G2[i,j] >= 1:
            return [i, j, any_common_neighbor(i,j)]
   if G empty: return [1]
   else return [v,u] , gdzie v, u - dowolna krawedz
```

złożoność $O(n^3)$ (złożoność mnożenia macierzy), aproksymacja k = 1

3 PO 3

Alicja i Bogdan mają czarą skrzynkę o azwie 3 PO 3, która rozwiązuje problem 3P o pytaniu: 3 da się podzielić zbiór na 3 równe części? Zbiór jest postaci C = $c_1, c_2, c_3, ..., c_n$. $c_i \in \mathbb{N}$. Jak powinni skorzystać ze skrzynki, żeby rozwiązać następujące podproblemy problemu 2P:

1. problem 2P
2. podproblem, w którym $c_i \leq 3$
3. podproblem, w którym $c_i = 3^i$
4. 3 PO 3 akceptuje tylko liczby podzielne przez 3
5. 3 PO 3 akceptuje tylko ciągi długości n = 3k

Rozwiązanie:
(1.)
do zbioru dodajemy element równy $c_{n+1} = \frac{\Sigma C}{2}$

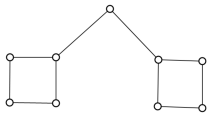
(2.)
można to w czasie wielomianowym obliczyć $O(n)$ na zmianę dodajemy

(3.)
odpowiedź to zawsze NIE, bo $3^1 + 3^2 + ... + 3^{n-1} < 3^n$ więc nie da się podzielić na dwie równe połówki.

(4.)
mnożymy wszystkie liczby w zbiorze * 3

(5.)
jeśli $n = 3k + 2$ to postępujemy tak jak w (1.), jeśli $n = 3k + 1$ to dodajemy do zbioru element $\frac{\Sigma C}{2} - 1$ i jedynkę, jeśli $n = 3k$ to dodajemy do zbioru element $\frac{\Sigma C}{2} + \varepsilon$ i dwie liczby ε, gdzie ε to wielka liczba

SL

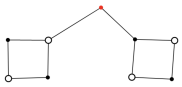


1. oszacuj złożoność obliczeniową SL
2. pokaż, że SL koloruje optymalnie wszystkie cykle
3. pokaż, że SL nie gwarantuje optymalnego kolorowania powyższego grafu
4. pokaż, że SL jest 4-absolutnie i 3-względnie aproksymacyjny dla grafów planarnych

rozwiązanie:
(1.)
O(n+m)

(2.)
Algorytm SL po kolei od tyłu wybiera najpierw dowolny wierzchołek cyklu a potem te odsłonięte stopnia 1, potem kolorując je zachłannie zawsze będziemy kolorowali wierzchołki przyległe więc nie użyjemy nadmiarowego koloru przez zły wybór.

(3.)

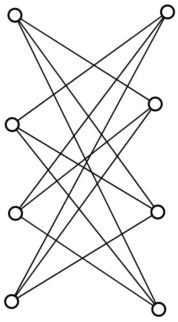


Przykładowe pokolorowanie do jakiego może doprowadzić SL (a jest to graf dwudzielny)

(4.)
W najgorszym przypadku usuwamy właśnie ten wierzchołek, potem znowu

znajdujemy wierzchołek minimalnego stopnia, który spełnia zależność $\delta \leq 5$ i tak dalej aż usuniemy wszystkie wierzchołki z grafu. Grafy, które będziemy po kolei zachłannie kolorować będą widziały co najwyżej 5 już pokolorowanych wierzchołków $\rightarrow$ SL da maksymalnie 6 kolorów.
Biorąc pod uwagę powyższe rozważania a także fakt, że SL koloruje optymalnie grafy o $\chi = 1$ (puste), widzimy, że SL może się pomylić począwszy od grafów dwudzielnych. Wówczas może dać maksymalnie 6, zatem jest 4-aproksymacyjny oraz 3-absolutnie aproksymacyjny.

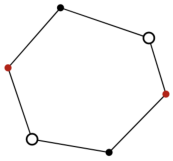
LF



1. oszacuj złożoność LF
2. udowodnij, że LF optymalnie koloruje cykle C_4, C_5 ale suboptymalnie C_6
3. wymień 4 klasy algorytmów, które LF koloruje optymalnie
4. wykonaj algorytm dla powyższego grafu
5. udowodnij, że LF nie jest aproksymacyjny, tj. nie istnieje stała c taka, że $LF(G) \leq c\chi(G)$
6. udowodnij, że LF jest k-bezwzględnie i l-względnie aproksymacyjny dla grafów kubicznych, ustal k i l.

rozwiązanie:
(1.)
O(n+m)

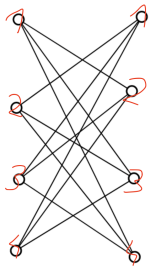
(2.)
dla C_4 i C_5 niezależnie od tego jak pokolorujemy to zawsze będzie dobrze (można to jakoś rozrysować) dla C_6 jest możliwe:



mimo, że optimum to 2

(3.)
gwiazdy, dwugwiazdy, cykle nieparzyste, ścieżki nieparzyste, grafy dwudzielne pełne symetryczne

(4.)



można uzyskać takie (nieoptymalne) pokolorowanie

(5.)
jak widać z powyższego pokolorowania, dla grafów tego typu używamy n/2 kolorów, mimo, że potrzebujemy tylko 2, czyli $LF(G) = O(n)$

(6.) dla grafów kubicznych na etapie kolorowania zachłannego algorytm "widzi"3 sąsiadów, którzy mają góra 3 różne kolory, więc dla grafów kubicznych $LF \leq 4$, LF optymalnie koloruje grafy puste, więc pierwszy raz może pomylić się przy $\chi(G) = 2$ stąd wynika, że jest 2-absolutnie aproksymacyjny oraz 2-aproksymacyjny

WZN

Problem ważonego zbioru niezależnego jest zdefiniowany tak: Dany jest graf G z obciążonymi wierzchołkami; zadaniem jest znalezienie zbioru niezależnego o maksymalnej wadze sumarycznej. Udowodnij, że WZN $\in$ NPH oraz zaprojektuj algorytm wielomianowy rozwiązujący WZN dla ścieżek.

rozwiązanie:

(I)
przyjmując w wersji decyzyjnej WZN_d próg p = k oraz wagi wierzchołków = 1 możemy rozwiązać problem k-zbiór niezależny, zatem WZN $\in$ NPH

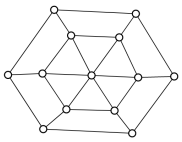
(II)
zaczynamy od początku ścieżki i wyznaczamy sumę co drugiego wierzchołka, zaczynamy od drugiego wierzchołka ścieżki i wyznaczamy sumę co drugiego, sprawdzamy, która suma jest większa i zwracamy

Mały Paryż *

Algorytm rozwiązuje problem k-pokrycia wierzchołkowego grafu G

```
def vertexcover(G,k)
   if E(G) empty:
      return true
   if k == 0:
      return false
   wybierz dowolne e = uv
   return vertexcover(G - u, k - 1) or vertexcover(G - v, k - 1)
```

1. jaka jest złożoność algorytmu w terminach m i k
2. podaj klasy grafów i wartości k, dla których vertexcover zwraca true w czasie wielomianowym
3. podaj klasy grafów i wartości k, dla których vertexcover zwraca true w czasie wykładniczym
4. rozwiąż problem VC dla poniższego grafu
5. dla k = 1 procedura wykonała się w czasie 15,6 milisekundy, ile zajmie dla poniższego grafu

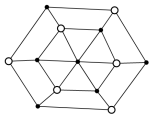


rozwiązanie:
(1.)
$T(k) = 2T(k-1) + m = 2^k + m + 2(m-1) + 4(m-2) + ... + 2^{k-1}(m-k) < 2^k + m + 2m + 4m + 8m + ... + 2^{k-1}m = O(m2^k)$

(2.)
gwiazdy (k = 1), dwugwiazdy (k=2)

(3.)
ścieżki, cykle, koła, grafy pełne, dwudzielne pełne

(4.)



k=7

(5.) $t = 2^6 * 15,6 = 998,1$ ms

5 Dowody grafowe

Tw. Eulera

Udowodnij, że jeśli G jest spójnym grafem płaskim to $s = m - n + 2$

rozwiązanie:
indukcja względem m:
Jeśli m = 1, to n = 1 i mamy jedną ścianę (przypadek trywialny P(1))
załóżmy P(m-1)
Rozważmy m-krawędziowy graf G. Jeśli G jest drzewem to m = n - 1 oraz f = 1 (jest acykliczny) czyli mamy P(m). Jeśli G zawiera cykl, usuńmy pewną krawędź należącą do cyklu, G-e ma m-1 krawędzi i s-1 ścian. Korzystamy z P(m-1) i mamy $s - 1 = m - 1 - n + 2 \rightarrow s = m - n + 2 \iff P(m)$. cnd

Lemat o pocałunkach

Udowodnij, że dla każdego spójnego grafu płaskiego G o $n \geq 3$ zachodzi $2m \geq 3s$

rozwiązanie:
Dwa przypadki:
1)G jest drzewem (s=1)
zatem $m = n - 1 \leq 2 \rightarrow 2m \leq 4 = 4s \leq 3s$
2) G zawiera cykl
usuwamy wszystkie wierzchołki stopnia 1 otrzymując w ten sposób $R(G)$ (rdzeń G). Każda jego ściana jest otoczona przez co najmniej 3 krawędzie oraz s(R(G)) = s(G) czyli mamy $m(R(G)) \geq 3s \rightarrow 2m(R(G)) \geq 3s$. Tym bardziej $m(G) \geq 3s$. cnd

Przydatne ograniczenie górne na m

Udowodnij, że dla grafu planarnego G o $n \geq 3$ mamy $m \leq 3n - 6$

rozwiązanie:
załózmy, że G jest płaski (jest izomorficzny do grafu płaskiego więc możemy tak zrobić)
mamy $s = m - n + 2$ oraz $2m \geq 3s$
wstawiamy do lematu o pocałunkach wzór na s i otrzymujemy wzór z twierdzenia. cnd

Własności drzewa

Udowodnij, że następujące własności są równoważne:

1. T jest drzewem
2. T jest acykliczny i ma n-1 krawędzi
3. T jest grafem spójnym i ma n-1 krawędzi
4. T jest grafem spójnym i każda krawędź jest mostem
5. każde dwa wierzchołki T są połączone dokładnie jedną drogą
6. dodanie do T jednej krawędzi stworzy dokładnie jeden cykl

rozwiązanie:
wszystkie własności są trywialne dla n =1, załóżmy prawdziwość wszystkich własności dla P(k), gdzie $k < n$

(1. $\rightarrow$ 2.)
T jest z definicji acykliczne, po usunięciu dowolnej krawędzi otrzymujemy $T - e = T_1 \cup T_2$ (rozspajamy). Z założenia indukcyjnego $m(T - e) = m(T_1) + m(T_2) = n(T_1) + n(T_2) - 2 \rightarrow m(T) = n(T_1) + n(T_2) - 1 = n(T) - 1$ zatem $m = n - 1$

(2. $\rightarrow$ 3.)
zakładamy, że T nie jest grafem spójnym zatem $T = T_1 \cup T_2$ z założenia indukcyjnego $m(T) = m(T_1) + m(T_2) = n(T_1) + n(T_2) - 2 \rightarrow m = n - 2$ co jest sprzeczne z 2. zatem T musi być spójny.

(3. $\rightarrow$ 4.)
k = 1 - ilość składowych spójności, czyli minimalna ilość krawędzi, która czyni go spójnym to n-1 więc każda krawędź musi być mostem

(4. $\rightarrow$ 5.)
załóżmy, że między pewną parą wierzchołków mamy dwie drogi, zatem jeśli usuniemy którąś z krawędzi należących do jednej z dróg to nie rozspoimy grafu

co jest sprzeczne z 4.

(5. $\rightarrow$ 6.)
załóżmy, że T zawiera cykl, wtedy dwa wierzchołki należące do tego cyklu połączone są dwiema różnymi drogami co jest sprzeczne z 5. po dodaniu krawędzi między tymi wierzchołkami tworzymy cykl, bo mamy jedną drogę, która była wcześniej i nową drogę przez tą krawędź, zatem $\gamma = 1$

(6. $\rightarrow$ 1.)
załóżmy, że graf nie jest spójny, jest to sprzeczne z 5. bo dodanie jednej krawędzi nie gwarantuje stworzenia cyklu, zatem T musi być spójny, z 6. jest również acykliczny zatem jest drzewem

Pąki w grafie planarnym

Udowodnij, że każdy graf planarny zawiera co najmniej 3 pąki (pąk - wierzchołek v taki, że $deg(v) \leq 5$

rozwiązanie:
załóżmy, że mamy w grafie planarnym tylko dwa pąki v i u, zatem $deg(v) + deg(u) + 6(n-2) \leq 2m \leq 6n - 12$ co jest sprzeczne ($deg(v) \neq 0$ i $deg(u) \neq 0$) (ostatnia nierówność z przydatnego oszacowania górnego m)

Liczba cyklomatyczna

Udowodnij, że dla dowolnego grafu spójnego liczba cyklomatyczna $\gamma(G) = m - n + 1$

rozwiązanie:
usuwamy tyle krawędzi, żeby graf stał się acykliczny (czyli żeby stał się drzewem), w drzewie m = n - 1, czyli musimy usunąć m - (n-1) krawędzi cnd.

Cykl w grafie dwudzielnym

Udowodnij, że graf jest dwudzielny $\iff$ nie ma nieparzystych cykli

rozwiązanie:
($\rightarrow$)
załóżmy, że graf dwudzielny ma nieparzysty cykl, zatem nie jest dwukolorowalny, bo nieparzyste cykle wymagają trzech kolorów, sprzeczność

($\leftarrow$)
załóżmy, że graf nie ma nieparzystych cykli, zatem dla pewnego wierzchołka v można przyporządkować każdemu innemu wierzchołkowi dwa kolory - jeden

dla tych, które są w odległości nieparzystej od v, drugi - parzystej, w ten sposób dowodzimy, że graf jest dwudzielny

Ograniczenie górne na χ

Udowodnij, że dla dowolnego grafu zachodzi $\chi(G) \leq \Delta + 1$

rozwiązanie:
dla n = 1 powyższa zależność zachodzi
załóżmy, że tw. zachodzi dla pewnego n, rozważmy graf n+1 wierzchołkowy G. Z założenia $\chi(G - v) \leq \Delta(G) + 1$, natomiast v ma co najwyżej Δ sąsiadów, czyli w najgorszym przypadku mamy jeden dostępny kolor, którym możemy go pokolorować, stąd $\chi(G) \leq \Delta(G) + 1$. cnd

stąd mamy tw. Brooksa
Dla dwóch klas grafów mamy sytuację $\chi = \Delta + 1$: dla grafów pełnych oraz cykli nieparzystej długości. Dla reszty $\omega \leq \chi \leq \Delta$

Liczba chromatyczna grafu planarnego

Dla grafu planarnego G udowodnij:

1. $\chi(G) \leq 6$
2. $\chi(G) \leq 5$
3. czy jest lepsze oszacowanie na liczbę chromatyczną dla grafów planarnych?

rozwiązanie:
(1.)
(2.)
(3.)
tak, $\chi(G) \leq 4$

Tw. Vizinga

Udowodnij tw. Vizinga: $\Delta \leq \chi' \leq \Delta + 1$

6 Trudne zadania

MFP

W historii problemu maksymalnego przepływu 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 $O(nm)$, $O(nmlog(n^2/m))$, $O(nmlogn)$, $O(nmlog^2n)$, $O(n^{5/3}m^{2/3})$, $O(n^2\sqrt{m})$, $O(n^3)$, $O(n^2m)$, $O(nm^2)$
przyporządkuj złożoności do odpowiednich algorytmów

rozwiązanie:
trzeba rozpatrzeć dwa przypadki: 1. grafy rzadkie, 2. grafy gęste i średnio gęste

Listowa reprezentacja drzew n-wierzchołkowych

Zaproponuj listowy sposób reprezentacji drzew n-wierzchołkowych w pamięci O(n) umożliwiający sprawdzanie O(1), czy para wierzchołków jest połączona krawędzią.