Programowanie dynamiczne

Przykład 1.

Problem:

Dane: Liczby naturalne n, k.

Wynik: $\binom{n}{k}$.

Naturalna metoda redukcji problemu obliczenia $\binom{n}{k}$ korzysta z zależności $\binom{n}{k}=\binom{n-1}{k-1}+\binom{n-1}{k}$

```
for i=1 to n do
    for j = 0 to k do tab_{i,j} ←"?"
        .....
function nPOk(n, k)
    if k = n or k = 0 then tab_{k,n} ← 1; return 1;
    if tab_{n-1,k-1} ="?" then tab_{n-1,k-1} ←nPOk(n − 1, k − 1)
    if tab_{n-1,k} ="?" then tab_{n-1,k} ←nPOk(n − 1, k)
    tab_{n,k} = tab_{n-1,k-1} + tab_{n-1,k}
    return tab_{n,k}
```

Rekurencyjne obliczanie $\binom{n}{k}$ z użyciem spamiętywania

```
for i = 1 to n do tab_{i,0} ← 1
    .....
function nPOk(n, k)
    for j = 1 to k do
        tab_{j,j} ← 1
        for i = j + 1 to n do tab_{i,j} ← tab_{i-1,j-1} + tab_{i-1,j}
    return tab_{n,k}
```

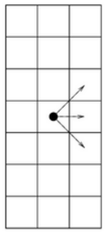
Obliczanie $\binom{n}{k}$ metodą programowania dynamicznego

Przykład 2.

Problem:

Dane: Tablica $\{a_{i,j}\}$ liczb nieujemnych $(i = 1, \ldots, n;\ j = 1, \ldots, m)$

Wynik: Ciąg indeksów $i_1, \ldots, i_m$ taki, że $\forall_{j=1,\ldots,m-1} |i_j - i_{j+1}| \leq 1$, minimalizujący sumę $\sum_{j=1}^{m} a_{i_j,j}$



Interpretacja: Ciąg $i_1, \ldots, i_m$ wyznacza trasę wiodącą od pierwszej do ostatniej kolumny tablicy a. Startujemy z dowolnego pola pierwszej kolumny i kończymy na dowolnym polu ostatniej kolumny. W każdym ruchu przesuwamy się o jedno pole: albo w prawo na wprost albo w prawo na ukos (jak pokazano na rysunku 1). Chcemy znaleźć trasę o minimalnej długości rozumianej jako suma liczb z pól znajdujących się na trasie.

```
for j = 1 to m do d_{0,j} ← d_{n+1,j} ← ∞
for i = 1 to n do d_{i,1} ← a_{i,1}
for j = 2 to m do
    for i = 1 to n do d_{i,j} ← a_{i,j} + min{d_{i-1,j-1}, d_{i,j-1}, d_{i+1,j-1}}
return min{d_{i,m} | i = 1, ..., n}
```

ZNALEZIENIE WARTOŚCI NAJKRÓTSZEJ ŚCIEŻKI

```
procedure trasa(i, j)
{   if j = 1 then return i
    if d_{i-1,j-1} < d_{i,j-1} then k ← i − 1 else k ← i
    if d_{i+1,j-1} < d_{k,j-1} then k ← i + 1
    return concat(trasa(k, j − 1), i)
}
    .....
write(trasa(i_0, m))
```

ODTWORZENIE ŚCIEŻKI

Programowanie dynamiczne jest częstą metodą rozwiązywania problemów optymalizacyjnych. Przykład 2 stanowi ilustrację klasycznego sposobu rozwiązania takiego problemu: najpierw znajdujemy wartość optymalnego rozwiązania a dopiero potem, na podstawie wyliczeń tej wartości, konstruujemy optymalne rozwiązanie.

2.1 Najdłuższy wspólny podciąg.

2.1.1 Definicja problemu

Definicja 1 *Ciąg $Z = \langle z_1, z_2, \ldots, z_k \rangle$ jest* podciągiem *ciągu rosnący ciąg indeksów $\langle i_1, i_2, \ldots, i_k \rangle$ $(1 \leq i_j \leq n)$ taki, że $X = \langle x_1, x_2, \ldots, x_n \rangle$, jeśli istnieje ściśle*

$$\forall_{i=1,2,\ldots,k}\ x_{i_i} = z_j.$$

Oznaczenia:

- $LCS(X, Y) = \{Z \mid Z$ jest wspólnym podciągiem X i Y o maksymalnej długości$\}$
- przez X_i oznaczamy i-literowy prefiks ciągu $X = \langle x_1, x_2, \ldots, x_n \rangle$, tj. podciąg $\langle x_1, x_2, \ldots, x_i \rangle$; w szczególności przez X_0 oznaczamy ciąg pusty.

Fakt 1 *Niech $d_{i,j}$ oznacza długość elementów z $LCS(X_i, Y_j)$. Wówczas:*
Tablicę d możemy obliczać kolejno wierszami (lub kolumnami), a wynik odczytamy z $d_{m,n}$

$$d_{i,j} = \begin{cases} 0 & \text{jeśli } i = 0 \text{ lub } j = 0, \\ 1 + d_{i-1,j-1} & \text{jeśli } i, j > 0 \text{ i } x_i = y_j, \\ \max(d_{i,j-1}, d_{i-1,j}) & \text{jeśli } i, j > 0 \text{ i } x_i \neq y_j \end{cases}$$

2.1.4 Koszt algorytmu

Obliczenie każdego elementu tablicy d odbywa się w czasie stałym. Tak więc całkowity koszt wypełnienia tablicy d jest równy $\Theta(n \cdot m)$. Koszt skonstruowania najdłuższego podciągu na podstawie tablicy d jest liniowy.

```
Procedure LCS(X_m, Y_n)
    for i ← 1 to m do d_{i,0} ← 0
    for j ← 0 to n do d_{0,j} ← 0
    for i ← 1 to m do
        for j ← 1 to n do
            if x_i = y_j then d_{i,j} ← 1 + d_{i-1,j-1}
            else d_{i,j} ← max{d_{i-1,j}, d_{i,j-1}}
```

2.3 Problem Plecakowy

Większość problemów plecakowych (w tym obie wersje, które przedstawimy) należy do klasy problemów $\mathcal{NP}$-trudnych, co oznacza, że raczej nie możemy spodziewać się rozwiązań działających w czasie wielomianowym od rozmiaru danych. Algorytmy, które pokażemy są pseudowielomianowe.

2.3.1 Wersja z powtórzeniami

PROBLEM:

Dane: ciąg $w_1, \ldots, w_n \in \mathcal{N}$
ciąg $v_1, \ldots, v_n \in \mathcal{R}$
liczba $W \in \mathcal{N}$

Wynik: wielozbiór zbiór $\{i_1, \ldots, i_k\}$ taki, że $\sum_{j=1}^{k} w_{i_j} \leq W$ oraz $\sum_{j=1}^{k} v_{i_j}$ jest maksymalna

Zakładamy, że waga każdego przedmiotu nie przekracza W.
Podproblemy: mniejszy plecak.

$K(w) =$ maksymalna wartość plecaka osiągalna dla plecaka o pojemności w.

Fakt 1

$$K(w) = \begin{cases} 0 & \textit{jeśli } w = 0, \\ \max_{i:w_i<w}\{K(w-w_i)+v_i\} & \textit{jeśli } w > 0 \end{cases}$$

Czas działania: $O(nW)$.

2.3.2 Wersja bez powtórzeń

PROBLEM:

Dane: ciąg $w_1, \ldots, w_n \in \mathcal{N}$
ciąg $v_1, \ldots, v_n \in \mathcal{R}$
liczba $W \in \mathcal{N}$

Wynik: zbiór $\{i_1, \ldots, i_k\}$ taki, że $\sum_{j=1}^{k} w_{i_j} \leq W$ oraz $\sum_{j=1}^{k} v_{i_j}$ jest maksymalna.

Podproblemy: mniejszy plecak pakowany podzbiorem przedmiotów.

$K(w, j) =$ maksymalna wartość plecaka osiągalna dla plecaka o pojemności w oraz przedmiotów $\{1, \ldots, j\}$

Fakt 2

$$K(w,j) = \begin{cases} 0 & \textit{jeśli } w = 0 \textit{ lub } j = 0 \\ \max\{K(w-w_j, j-1)+v_j, K(w,j-1)\} & \textit{wpp} \end{cases}$$

Czas działania: $O(nW)$.

2.5 Przynależność do języka bezkontekstowego

2.5.1 Definicja problemu

Rozpoczynamy od przypomnienia podstawowych pojęć związanych z gramatykami bezkontekstowymi (pojęcia te powinny być znane z wykładu Wstęp do Informatyki).

Definicja 1 Gramatyką bezkontekstową *nazywamy system* $G = \langle V_N, V_T, P, S\rangle$, *gdzie*

- V_N *i* V_T *są skończonymi rozłącznymi zbiorami (nazywamy je odpowiednio* alfabetem symboli nieterminalnych *i* alfabetem symboli terminalnych*);*
- P *jest skończonym podzbiorem zbioru* $V_N \times (V_N \cup V_T)^*$ *(elementy* P *nazywamy* produkcjami*);*
- $S \in V_N$ *i jest nazywany* symbolem początkowym gramatyki.

Definicja 2 *Jeśli każda produkcja gramatyki bezkontekstowej* G *jest postaci:*

- $A \to BC$ *lub*
- $A \to a$,

gdzie $A, B, C \in V_N$ *i* $a \in V_T$, *to mówimy, że* G *jest w* normalnej postaci Chomsky'ego.

Definicja 3 *Niech* $G = \langle V_N, V_T, P, S\rangle$; $\alpha, \beta, \gamma \in (V_N \cup V_T)^*$ *oraz* $A \in V_N$. *Mówimy, że ze słowa* $\alpha A \beta$ można wyprowadzić w G *słowo* $\alpha\gamma\beta$, *co zapisujemy* $\alpha A\beta \Rightarrow \alpha\gamma\beta$, *jeśli* $A \to \gamma$ *jest produkcją z* P.

Definicja 4 *Język* $L(G)$ *generowany przez gramatykę* $G = \langle V_N, V_T, P, S\rangle$ *definiujemy jako*

$$L(G) = \{w \mid w \in V_T^* \text{ oraz } S \stackrel{*}{\Rightarrow} w\},$$

gdzie $\stackrel{*}{\Rightarrow}$ *oznacza tranzytywne domknięcie relacji* $\Rightarrow$.

Przykład 1. Niech

- $V_N = \{S, T, L, R\}$;
- $V_T = \{(,)\}$;
- $P = \{S \to SS\ ;\ S \to LT\ ;\ S \to LR\ ;\ T \to SR\ ;\ L \to (\ ;\ R \to)\ \}$

Jak łatwo sprawdzić $L(G)$ jest językiem zawierający wszystkie słowa zbudowanie z poprawnie rozstawionych nawiasów.
Przykładowe wyprowadzenie słowa w=(()()):

$$S \Rightarrow LT \Rightarrow LSR \Rightarrow LSSR \Rightarrow LLRSR \Rightarrow LLRLRR \Rightarrow (LRLRR \Rightarrow$$

$$(LRL)R \Rightarrow (L)L)R \Rightarrow (L)()R \Rightarrow (()()R \Rightarrow (()())$$

2.5.2 Algorytm naiwny

Niech $M(w)$ oznacza zbiór słów wyprowadzalnych z w w jednym kroku.

```
F_0 ← {S}
for i = 1 to 2|w| − 1 do
    F_{i+1} ← ⋃_{w∈F_i} M(w)
if w ∈ F_{2|w|−1} then return "TAK" else return "NIE"
```

ALGORYTM DYNAMICZNY

Podejście dynamiczne polega na obliczeniu dla każdego podsłowa słowa w (począwszy od podsłów jednoliterowych a skończywszy na całym w) zbioru nieterminali, z których da się to podsłowo wyprowadzić. Innymi słowy, celem jest wyznaczenie zbiorów $m_{i,j}$ $(1 \le i \le j \le n)$:

$$m_{i,j} = \{A \mid A \in V_N \ \&\ A \stackrel{*}{\Rightarrow} a_i \dots a_j\}$$

Odpowiedzią algorytmu będzie wartość wyrażenia $S \in m_{1,n}$.

Zbiory $m_{i,j}$ wyznaczyć można na podstawie następujących zależności:

$$m_{i,i} = \{A \mid (A \to a_i) \in P\} \text{ dla } i = 1, \dots, n$$

$$m_{i,j} = \bigcup_{k=i}^{j-1} m_{i,k} \otimes m_{k+1,j} \text{ dla } 1 \le i < j \le n$$

gdzie $m_{i,k} \otimes m_{k+1,j} = \{A \mid (A \to BC) \in P \text{ dla pewnych } B \in m_{i,k} \text{ oraz } C \in m_{k+1,n}\}$

Koszt: łatwo sprawdzić, że algorytm wykonuje $\Theta(n^3)$ operacji $\otimes$. Ponieważ koszt jednej operacji $\otimes$ jest stały (patrz Uwagi implementacyjne), $\Theta(n^3)$ opisuje koszt całego algorytmu.

Uwagi implementacyjne. Elementy obliczanej tablicy są zbiorami. To stanowi istotną różnicę w stosunku do poprzednich przykładów, gdzie elementy tablicy były prostego typu. Przyjęcie odpowiedniej struktury danych do pamiętania zbiorów $m_{i,j}$ oraz wybór metody obliczania wyniku operacji $\otimes$ może mieć istotny wpływ na koszt algorytmu.

Przykładowo: zbiory $m_{i,j}$ możemy pamiętać jako wektory charakterystyczne lub jako listy. W pierwszym przypadku potrzebujeny $\sim (1/2)n^2|V_N|$ bitów na zapamiętanie tablicy. W drugim przypadku ponosimy spore koszty pamięciowe związane z używaniem wskaźników - jednak mogą one być opłacalne, gdy w średnim przypadku rozmiar zbiorów $m_{i,j}$ jest nieduży. W tym przypadku rozsądną metodą obliczania $m_{i,k} \otimes m_{k+1,j}$ może okazać się zwykłe przeglądanie list:

```
for each B ∈ m_{i,k} do
    for each C ∈ m_{k+1,j} do
        if BC jest prawą stroną produkcji z P
        then m_{i,j} ← m_{i,j} ∪ { symbol z lewej strony tej produkcji }
```

Przy odpowiednim zapamiętaniu informacji o produkcjach, koszt takiego obliczenia nie zależy od liczby produkcji i jest proporcjonalny do iloczynu długości list, co w rozważanym przypadku może być znacznie mniejsze od $|V_N|^2$. Jeśli liczba produkcji jest niewielka opłacalne może być zastosowanie innego sposobu:

```
for each (A → BC) ∈ P do
    if B ∈ m_{i,k} & C ∈ m_{k+1,j} then m_{i,j} ← m_{i,j} ∪ {A}
```

Sposób ten jest szczególnie atrakcyjny przy wektorowej reprezentacji zbiorów, ponieważ wówczas czas odpowiedzi na pytanie o przynależność elementu do zbioru jest stały i koszt powyższej pętli wynosi $\Theta(|P|)$.

3.1 Definicja klasy $\mathcal{NP}$

Klasa $\mathcal{NP}$ bywa definiowana na wiele sposobów. Najczęściej poprzez niedeterministyczne maszyny Turinga, jako klasa problemów decyzyjnych, które są przez te maszyny rozwiązywane w czasie wielomianowym. My przyjmiemy następującą:

Definicja 6 *Problem decyzyjny X należy do klasy problemów $\mathcal{NP}$, jeśli istnieje wielomianowy algorytm A oraz wielomian p, takie że:*

$$x \in X \Leftrightarrow \exists_{y_x} \ |y_x| \leq p(|x|) \wedge A(x, y_x) = "\ accept"$$

Przyjmujemy tu powszechną konwencję utożsamiania problemu decyzyjnego ze zbiorem danych, dla których jest odpowiedź "tak" (Przykładowo problem *Prime*, polegający na stwierdzeniu czy dana liczba jest pierwsza, utożsamiamy ze zbiorem liczb pierwszych). Tak więc powyższą definicję można odczytać w ten sposób, że problem X jest w klasie $\mathcal{NP}$, jeśli istnieje wielomianowy algorytm A, który dla każdego $x \in X$ zaakceptuje go, gdy x będzie podany na wejściu wraz z pewnym, nie za długim, argumentem y_x (o długości ograniczonej przez wielomian od długości x-a). Taki y_x nazywamy *świadkiem* przynależności x do X. Natomiast żaden x spoza X nie zostanie zaakceptowany przez A, niezależnie od tego z jakimi kandydatami na świadków nie dostarczalibyśmy go na wejściu dla A.

Przykład.
Rozważmy problem *NonPrime*, polegający na sprawdzeniu, czy dana liczba jest złożona. Prostym algorytmem świadczącym o przynależności *NonPrime* do klasy $\mathcal{NP}$ jest poniższy prościutki algorytm:

```
Algorytm D(x,y)
    if (1 < y < x && x mod y == 0) return ("accept")
    else return ("reject")
```

Świadkiem dla $x \in NonPrime$ jest dowolny jego dzielnik, natomiast $x \notin NonPrime$ nie zostanie zaakceptowany przez D z żadnym kandydatem na świadka. Oczywiście D działa w czasie wielomianowym. □

Nie wszystkie problemu z klasy $\mathcal{NP}$ są jednakowo trudne. W szczególności należą do niej wszystkie problemy z klasy $\mathcal{P}$, a więc takie, które można rozwiązać algorytmami wielomianowymi nie korzystającymi ze świadków.
W klasie

Definicja 7 *Problem decyzyjny A jest $\mathcal{NP}$-zupełny, jeśli:*

- $A \in \mathcal{NP}$,
- *dowolny problem B z klasy $\mathcal{NP}$ jest wielomianowo redukowalny do problemu A.*

Wyjaśnienia wymaga jeszcze drugi warunek w powyższej definicji.

Definicja 8 *Problem decyzyjny B jest* wielomianowo redukowalny *do problemu A, jeśli istnieje funkcja f oraz wielomian q, takie, że:*

- *istnieje wielomianowy algorytm obliczający wartości funkcji f,*
- $\forall_x x \in A \Leftrightarrow f(x) \in B$.

Mam nadzieję, że nie irytuje Was brak precyzji w powyższych definicjach. Spowodowany jest on przeświadczeniem, że rozbudowany formalizm konieczny dla zachowania precyzji zniechęciłby wielu z Was do przeczytania notatki. Jestem też przekonany, że łatwo "dopowiecie" sobie szczegóły.

Podstawowe problemy NP-zupełne:

- 3-SAT
  - Dane: Formuła ϕ w postaci 3CNF.
  - Pytanie: Czy ϕ jest spełnialna?
- Trójwymiarowe skojarzenie
  - Dane: Zbiór trójek $M \subseteq W \times X \times Y$, gdzie W, X, Y - rozłączne zbiory, każdy o mocy q
  - Pytanie: Czy istnieje $M' \subseteq M$, o mocy q, taki, że żadne elementy z M' nie mają takiej samej współrzędnej?
- Pokrycie wierzchołkowe
  - Dane: Graf $G = (V, E)$ i liczba $K \leq |V|$.
  - Pytanie: Czy istnieje $V' \subseteq V$ o mocy nie większej niż K, taki, że każda krawędź z E jest incydentna z co najmniej jednym wierzchołkiem z V'?
- Klika
  - Dane: Graf $G = (V, E)$ i liczba $K \leq |V|$.
  - Pytanie: Czy istnieje pełny podgraf w G o co najmniej K wierzchołkach?
- Cykl Hamiltona
  - Dane: Graf $G = (V, E)$.
  - Pytanie: Czy w G istnieje cykl przechodzący przez każdy wierzchołek z V dokładnie jeden raz?
- Rozbicie
  - Dane: Zbiór A i funkcja wagowa $c : A \to Z+$.
  - Pytanie: Czy istnieje podzbiór $A' \subseteq A$, taki, że

$$\sum_{a \in A'} c(a) = \sum_{a \in A \setminus A'} c(a)?$$

ZADANIA EGZAMINACYJNE

Treść

Podaj pseudowielomianowy algorytm, znajdujący rozkład liczby naturalnej na czynniki pierwsze. Uzasadnij stwierdzenie, że jest pseudowielomianowy.

▼ Rozwiązanie

```
int n, m = sqrt(n);

for(int i=2; (n != 1) && (i <= m); i++) {
    while(n%i == 0) {
        printf("%d ", i);
        n = n/i;
    }
}
// Gdyby n była lizbą pierwszą lub miała dzielnik pierwszy większy od sqrt(n)
if (n != 1)
    printf("%d\n", n);
```

Ilość wykonywanych operacji zależy od wielkość liczby na wejściu w postaci unarnej, więc algorytm jest pseudowielomianowy.

Treść

Zapisz w pseudokodzie algorytm wielomianowy, znajdujący minimalny koszt obliczenia iloczynu ciągu macierzy.

Treść

Zapisz w pseudokodzie algorytm znajdujący długość najdłuższego wspólnego podciągu dwóch ciągów.
+ TEN CIĄG

▼ Rozwiązanie

```
# postać ciągów
S1 = x1...xn
S2 = y1...ym

# Zerujemy pierwszy wiersz i kolumnę dp
dp[0][0...m] = 0
dp[0...n][0] = 0

def lcs(n, m):
  for i in range(1, n):
    for j in range(1, m):
      if S1[i] == S2[j]:
        dp[i][j] = 1 + dp[n-1][m-1]
      else:
        dp[i][j] = max(dp[i-1][j], dp[i][j-1])
  return dp[n, m]
```

Treść

Podaj definicję problemu plecakowego z powtórzeniami i przedstaw pseudowielomianowy algorytm rozwiązujący ten problem. Uzasadnij, że jest on pseudowielomianowy.

(a) Podaj definicję problemu plecakowego (wersja bez powtórzeń) a następnie podaj dla niego algorytm zachłanny działający w czasie pseudowielomianowym.

(b) Uzasadnij pseudowielomianość podanego algorytmu.

▼ Rozwiązanie

(a)

PROBLEM:

Dane: ciąg $w_1, \ldots, w_n \in \mathcal{N}$
ciąg $v_1, \ldots, v_n \in \mathcal{R}$
liczba $W \in \mathcal{N}$

Wynik: zbiór $\{i_1, \ldots, i_k\}$ taki, że $\sum_{j=1}^{k} w_{i_j} \leq W$ oraz $\sum_{j=1}^{k} v_{i_j}$ jest maksymalna.

Podproblemy: mniejszy plecak pakowany podzbiorem przedmiotów.

$K(w, j) =$ maksymalna wartość plecaka osiągalna dla plecaka o pojemności w oraz przedmiotów $\{1, \ldots, j\}$.

Fakt 2

$$K(w,j) = \begin{cases} 0 & \textit{jeśli } w = 0 \textit{ lub } j = 0 \\ \max\{K(w - w_j, j-1) + v_j, K(w, j-1)\} & \textit{wpp} \end{cases}$$

□

Czas działania: $O(nW)$.

(b) D-d pseudowielomianowości:

Jest pseudowielomianowy, ponieważ złożoność problemu nie zależy od rozmiaru wejścia, tylko od wartości liczb na wejściu. Dokładniej im większe W - ile możemy spakować, tym dłużej działa nasz algorytm. (Normalnie powinno być tak, że im więcej liczb na wejściu tym dłużej działa

Zadanie 1. Podaj definicję i sposób wyliczania $m_{i,j}$ w algorytmie sprawdzającym przynależność słowa do gramatyki.

ZADANIA EGZAMINACYJNE CD

+ ZADANIA Z ĆWICZEŃ

Treść

Podany na wykładzie algorytm obliczający minimalny koszt policzenia iloczynu n macierzy, o wymiarach odpowiednio d_0 × d_1, d_1 × d_2, . . . , d_n−1 × d_n, wylicza pewne wartości m_ij .

Podaj ich definicje i sposób wyliczenia.

Jeśli nie pamiętasz tego algorytmu, podaj inny algorytm o nie gorszej złożoności, który rozwiązuje ten problem.

▼ Rozwiązanie

Niech

$$m_{i,j} = \text{"minimalny koszt obliczenia } M_i \times M_{i+1} \times \cdots \times M_j\text{"}$$

Dla wygody przyjmujemy, że $m_{i,j} = 0$ (dla $i \geq j$). Wówczas

$$m_{i,j} = \min_{i \leq k < j} (m_{i,k} + m_{k+1,j} + d_{i-1} d_k d_j).$$

Składnik $m_{i,k}$ jest kosztem obliczenia $M_i \times M_{i+1} \times \cdots \times M_k$, składnik $m_{k+1,j}$ - kosztem obliczenia $M_{k+1} \times M_{k+2} \times \cdots \times M_j$, natomiast $d_{i-1} d_k d_j$ to koszt obliczenia iloczynu dwóch powstałych macierzy.

Opisz podany na wykładzie algorytm dla problemu wyznaczania optymalnej kolejności mnożenia ciągu macierzy.

Jeśli go nie pamiętasz podaj inny o nie gorszej złożoności

▼ Rozwiązanie

Niech

$$m_{i,j} = \text{"minimalny koszt obliczenia } M_i \times M_{i+1} \times \cdots \times M_j\text{"}$$

Dla wygody przyjmujemy, że $m_{i,j} = 0$ (dla $i \geq j$). Wówczas

$$m_{i,j} = \min_{i \leq k < j} (m_{i,k} + m_{k+1,j} + d_{i-1} d_k d_j).$$

Składnik $m_{i,k}$ jest kosztem obliczenia $M_i \times M_{i+1} \times \cdots \times M_k$, składnik $m_{k+1,j}$ - kosztem obliczenia $M_{k+1} \times M_{k+2} \times \cdots \times M_j$, natomiast $d_{i-1} d_k d_j$ to koszt obliczenia iloczynu dwóch powstałych macierzy.

```
procedure dyn − matmult(d[0..n]);
    int m[1..n, 1..n], p[1..n, 1..n]
    for i ← 1 to n do m_ii ← 0;
    for s ← 1 to n − 1 do
        for i ← 1 to n − s do
            j ← i + s
            m_ij ← min_{i<k<j}(m_{i,k} + m_{k+1,j} + d_{i−1} d_k d_j)
            p_ij ← "to k, przy którym osiągane było minimum dla m_ij"
    return p[1..n, 1..n]
```

Algorytm oblicza wartości $m_{i,i+s}$ (na podstawie powyższego wzoru) oraz wartości $p_{i,i+s}$, które umożliwiają pó"xniejsze skonstruowanie rozwiązania.

1. (1pkt) Zmień podany na wykładzie algorytm znajdujący najtańszą drogę przejścia przez tablicę tak, by znajdował drogę o drugim co do wielkości koszcie.

2. (1pkt) Uzupełnij podany na wykładzie algorytm sprawdzający przynależność słowa do języka generowanego przez bezkontekstową gramatykę w normalnej postaci Chomsky'ego tak, by w przypadku pozytywnej odpowiedzi wypisywał jego wyprowadzenie.

3. (1,5pkt) *Supersciągiem* ciągów X i Y nazywamy każdy taki ciąg Z, że zarówno X jak i Y są podciągami ciągu Z.

   Ułóż algorytm, który dla danych ciągów X i Y znajduje ich najkrótszy supersciąg.

4. (2pkt) W n-elementowej tablicy A pamiętany jest rosnący ciąg liczb naturalnych. Nie znamy wartości jej elementów, ale możemy się o nie pytać. Pytanie o wartość $A[i]$ kosztuje nas c_i.

   Ułóż algorytm, który dla danych liczb naturalnych $c_1, c_2, \ldots, c_n$ oraz liczby k obliczy najmniejszym kosztem (liczonym jako suma kosztów zadanych pytań), ile liczb w tablicy A ma wartość większą niż k.

5. (2pkt) Zmodyfikuj algorytm znajdujący najdłuższy wspólny podciąg dwóch ciągów n elementowych, tak by działał w czasie $O(n^2)$ i używał $O(n)$ pamięci.

6. (1,5pkt) Ułóż algorytm, który dla danego drzewa ważonego $T = (V, E; c)$, gdzie $c : V \to N$ jest funkcją wagową, znajduje niezależny podzbiór $V' \subseteq V$, którego suma wag wierzchołków jest możliwie największa.

7. (2pkt) Ułóż algorytmy, które dla danych podciągów x i y rozwiązują następujące wersje problemu znajdowania najdłuższego wspólnego podciągu:

   - znajdowanie najdłuższego wspólnego podciągu zawierającego podciąg "aaabb",
   - znajdowanie najdłuższego wspólnego podciągu nie zawierającego podciągu "aaabb",
   - znajdowanie najdłuższego wspólnego podciągu zawierającego podsłowo "aaabb",
   - znajdowanie najdłuższego wspólnego podciągu nie zawierającego podsłowa "aaabb".

8. (1pkt) Rozważmy następujący problem *3-podziału*. Dla danych liczb całkowitych $a_1, \ldots, a_n \in \langle -C, C \rangle$ chcemy stwierdzić, czy można podzielić zbiór $\{1, 2, \ldots, n\}$ na trzy rozłączne podzbiory I, J, K, takie, że

   $$\sum_{i \in I} a_i = \sum_{j \in J} a_j = \sum_{k \in K} a_k.$$

9. (2pkt) Ułóż algorytm, który dla dwóch ciągów liczb naturalnych X i Y, znajduje ich najdłuższy wspólny podciąg, który jest ciągiem rosnącym. Czy założenie, że liczby w ciągach są ograniczone przez pewną stałą k, upraszcza problem?

10. (2pkt) Dwie proste równoległe l' i l'' przecięto n prostymi $p_1, \ldots, p_n$. Punkty przecięcia prostej p_i z prostymi l' i l'' wyznaczają na niej odcinek. Niech Odc będzie zbiorem tych odcinków.

    (a) Ułóż algorytm, wyznaczający w Odc podzbiór nieprzecinających się odcinków, o największej mocy.

    (b) Ułóż algorytm, wyznaczający liczbę podzbiorów, o których mowa w poprzednim punkcie.