

## PROGRAMOWANIE DYNAMICZNE

**2.3 Problem Plecakowy**

Problem plecakowy obejmuje szeroką klasę problemów optymalizacji kombinatorycznej. Dla danego zbioru przedmiotów o określonych wagach i wartościach należy wybrać podzbiór o jak największej sumarycznej wartości i sumarycznej wadze nie przekraczającej zadanego ograniczenia.

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

*Wynik:* wielozbiór zbiór  $\{i_1, \dots, i_k\}$  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**

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

□

Czas działania:  $O(nW)$ .

**2.3.2 Wersja bez powtórzeń**

PROBLEM:

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

*Wynik:* zbiór  $\{i_1, \dots, i_k\}$  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  $\{1, \dots, j\}$ .

**Fakt 2**

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

□

Czas działania:  $O(nW)$ .

## 2.4 Najkrótsze ścieżki między wszystkimi parami wierzchołków

Do uzupełnienia.

## 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.

Zwyczajowo produkcje  $(A, \alpha)$  zapisujemy jako  $A \rightarrow \alpha$ .

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

- $A \rightarrow BC$  lub
- $A \rightarrow a$ ,

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  $A \rightarrow \gamma$  jest produkcją z  $P$ .

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

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

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

PRZYKŁAD 1. Niech

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

Jak łatwo sprawdzić  $L(G)$  jest językiem zawierający wszystkie słowa zbudowane z poprawnie rozstawionych nawiasów.

Przykładowe wyprowadzenie słowa  $w = (()())$ :

$$\begin{aligned} S &\Rightarrow LT \Rightarrow LSR \Rightarrow LSSR \Rightarrow LLRSR \Rightarrow LLRLRR \Rightarrow (LRLRR \Rightarrow \\ &(LRL)R \Rightarrow (L)L)R \Rightarrow (L)()R \Rightarrow (()()R \Rightarrow (()()) \end{aligned}$$

□

PROBLEM:

Dla ustalonej gramatyki bezkontekstowej  $G = \langle V_N, V_T, P, S \rangle$  w normalnej postaci Chomsky'ego

*Dane:* słowo  $w = a_1 \dots a_n$  ( $a_i \in V_T$  dla  $i = 1, \dots, n$ )

*Wynik:* "TAK" - jeśli  $w \in L(G)$   
"NIE" - w przeciwnym przypadku.

### 2.5.2 Algorytm naiwny

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

```

 $F_0 \leftarrow \{S\}$ 
for  $i = 1$  to  $2|w| - 1$  do
     $F_{i+1} \leftarrow \bigcup_{w \in F_i} M(w)$ 
if  $w \in F_{2|w|-1}$  then return "TAK" else return "NIE"

```

POPRAWNOŚĆ: Każda produkcja gramatyki w normalnej postaci Chomsky'ego albo zwiększa o jeden długość wyprowadzanej frazy albo zamienia symbol nieterminalny na terminalny. Tak więc każde słowo z języka o długości  $n$  jest wyprowadzane z  $S$  po  $2n - 1$  krokach.

KOSZT: Czynnikiem determinującym koszt algorytmu jest koszt pętli wewnętrznej, a ten w głównym stopniu zależy od wielkości zbiorów  $F_i$ . Niestety, nawet dla tak prostych gramatyk jak ta z Przykładu 1, zbiory  $F_i$  mogą zawierać wykładniczo wiele słów.

### 2.5.3 Algorytm dynamiczny

IDEA:

Jeśli  $w = a_1 \dots a_n$  jest słowem z języka  $L(G)$ , to pierwsza produkcja zastosowana w jego wyprowadzeniu (o ile  $n > 1$ ) musi mieć postać  $S \rightarrow AB$ . Ponieważ dalsze wyprowadzenie z symbolu  $A$  jest niezależne od wyprowadzenia z symbolu  $B$ , więc musi istnieć  $i$  ( $1 \leq i \leq n-1$ ) takie, że  $A \xRightarrow{*} a_1 \dots a_i$  oraz  $B \xRightarrow{*} a_{i+1} \dots a_n$ .

Na podstawie tej obserwacji możemy łatwo zbudować algorytm rekurencyjny, jednak czas jego działania może być wykładniczy. W szczególności algorytm taki wielokrotnie może próbować wyprowadzać ten sam fragment słowa  $w$  z tego samego symbolu nieterminalnego.

PRZYKŁAD 2. Niech gramatyka zawiera (między innymi) produkcje  $S \rightarrow AB$  oraz  $A \rightarrow AA$ . Na drugim poziomie rekursji rekurencyjna procedura może być wywoływana dla  $A$  i podsłów  $a_1 \dots a_i$  (dla  $i = 1, \dots, n-1$ ); wewnątrz każdego z tych wywołań będzie ona znów wywoływana m.in. dla  $A$  i podsłów  $a_1 \dots a_j$  ( $j = 1, \dots, i-1$ ).

□

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

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

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

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

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

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

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

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 potrzebujemy  $\sim (1/2)n^2|V_N|$  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  $B \in m_{i,k}$  do
  for each  $C \in m_{k+1,j}$  do
    if  $BC$  jest prawą stroną produkcji z  $P$ 
      then  $m_{i,j} \leftarrow m_{i,j} \cup \{ \text{symbol z lewej strony tej produkcji} \}$ 

```

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:

```

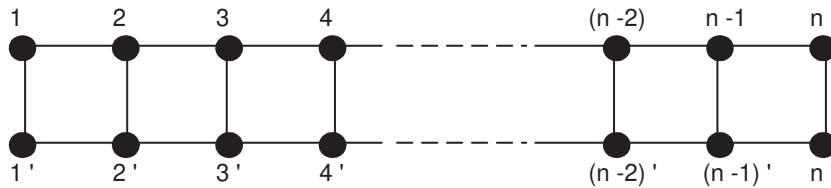
for each  $(A \rightarrow BC) \in P$  do
  if  $B \in m_{i,k}$  &  $C \in m_{k+1,j}$  then  $m_{i,j} \leftarrow m_{i,j} \cup \{A\}$ 

```

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  $\Theta(|P|)$ .

## 2.6 Drzewa rozpinające drabin

**Definicja 5** Drabiną  $n$ -elementową nazywamy graf  $D_n$  przedstawiony na rysunku ??



Rysunek 1: *Drabina  $n$  elementowa.*

PROBLEM:

*Dane:* liczby naturalne  $n, k$ ;

ciąg par liczb naturalnych  $\{u_i, v_i\}$  ( $i = 1, \dots, m$ )

INTERPRETACJA: pary  $\{u_i, v_i\}$  określają wyróżnione krawędzie w  $n$ -elementowej drabinie;

*Wynik:* Liczba drzew rozpinających o  $k$  krawędziach wyróżnionych.

Ideę algorytmu przedstawimy rozważając prostszy problem, a mianowicie problem wyznaczania liczby drzew rozpinających w  $D_n$  (bez uwzględniania krawędzi wyróżnionych). Co prawda, w takim przypadku można w prosty sposób wyprowadzić zwięzły wzór na tę liczbę, lecz nie to jest naszym celem.

W dalszym ciągu, mówiąc o drabinie  $D_i$ , będziemy mieć na myśli podgraf drabiny  $D_i$  indukowany przez wierzchołki  $\{1, \dots, i, i', \dots, i'\}$ .

**Fakt 3** Niech  $T$  będzie dowolnym drzewem rozpinającym drabinę  $D_{i+1}$ , dla dowolnego  $i \geq 1$ . Wówczas  $T \cap D_i$  jest albo

- drzewem rozpinającym drabinę  $D_i$  albo
- lasem rozpinającym grafu  $D_i$  złożonym z dwóch drzew; jedno z tych drzew zawiera wierzchołek  $i$  a drugie - wierzchołek  $i'$ .

Analogiczna własność zachodzi, gdy  $T$  jest lasem rozpinającym drabinę  $D_{i+1}$ , złożonym z dwóch drzew, przy czym jedno z tych drzew zawiera wierzchołek  $(i+1)$ , a drugie - wierzchołek  $i'+1$ .

Niech  $S_i$  oznacza zbiór drzew rozpinających drabinę  $D_i$ , a  $N_i$  zbiór lasów rozpinających, o których mowa w Fakcie ??, w drabinie  $D_i$ . Naszym celem jest policzenie wartości  $|S_n|$ .

IDEA ALGORYTMU: Kolejno dla  $i = 1, \dots, n$  liczymy wartości  $|S_i|$  oraz  $|N_i|$ , korzystając z zależności przedstawionych w poniższym fakcie:

**Fakt 4** (a)  $|S_1| = |N_1| = 1$

(b) Dla każdego  $i > 1$ :

$$|S_i| = 3|S_{i-1}| + |N_{i-1}|,$$

$$|N_i| = 2|S_{i-1}| + |N_{i-1}|.$$

DOWÓD:

(a) Oczywiście.

(b) Niech  $K_i = \{(i-1, i), ((i-1)', i'), (i, i')\}$  będzie zbiorem krawędzi, którymi  $D_i$  różni się od  $D_{i-1}$ .

Z dowolnego drzewa rozpinającego  $T \in S_{i-1}$  można utworzyć trzy różne drzewa rozpinające z  $S_i$  poprzez dodanie dowolnych dwóch krawędzi ze zbioru  $K_i$ . Ponadto, dodając wszystkie krawędzie z  $K_i$  do dowolnego lasu z  $N_{i-1}$  można utworzyć jedno drzewo z  $S_i$ . To uzasadnia pierwszy ze wzorów.

Dodając krawędź  $(i-1, i)$  do drzewa  $T \in S_{i-1}$  otrzymujemy las z  $N_i$ . Jedno z jego drzew zawiera wierzchołek  $i$ , a drugie z drzew składa się z izolowanego wierzchołka  $\{i'\}$ . Analogicznie otrzymujemy jeden las dodając do  $T$  krawędź  $((i-1)', i')$ . Ponadto, z każdego lasu z  $N_{i-1}$ , po dodaniu dwóch poziomych krawędzi  $(i-1, i)$  oraz  $((i-1)', i')$ , otrzymujemy jeden las z  $N_i$ . To uzasadnia drugi wzór.  $\square$

Teraz w prosty sposób możemy tę metodę uogólnić do rozwiązania problemu liczenia drzew rozpinających z wyróżnionymi krawędziami. W tym celu zamiast czterech zbiorów  $S_i$  i  $N_i$ , rozważamy  $2(k+1)$  zbiorów:  $S_i(j)$ ,  $N_i(j)$ , gdzie parametr  $j$  ( $j = 0, \dots, k$ ) oznacza liczbę krawędzi wyróżnionych. Przykładowo:  $S_i(j)$  będzie się równać liczbie drzew rozpinających w drabinie  $D_i$  zawierających dokładnie  $j$  krawędzi wyróżnionych. Wyprowadzenie wzorów analogicznych do tych z Faktu ?? pozostawiamy jako proste ćwiczenie.

### 3 Appendix. $\mathcal{NP}$ -zupełność

Do zrozumienia materiału wykładu w zasadzie wystarczy potoczna wiedza, że problemy  $\mathcal{NP}$ -trudne czy też  $\mathcal{NP}$ -zupełne oznaczają problemy, dla których najprawdopodobniej nie istnieją algorytmy działające w czasie wielomianowym. Nie oznacza to jednak, że podanych poniżej pojęć i faktów można nie znać:-)

### 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:

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

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

Aby się przekonać, że nie zawsze tak łatwo, jak dla *NonPrime*, można podać odpowiedni algorytm świadczący o przynależności problemu do klasy  $\mathcal{NP}$ , spróbuj rozwiązać poniższe ćwiczenie:

ĆWICZENIE.

Podaj algorytm świadczący o przynależności problemu *Prime*<sup>1</sup> do klasy  $\mathcal{NP}$ .

### 3.2 Więcej o klasie $\mathcal{NP}$

Nie wszystkie problemy 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.

---

<sup>1</sup>W rzeczywistości problem ten należy do klasy  $\mathcal{P}$ , o czym wiadomo od 2002 roku: M.Agrawal, N.Kayal, N.Saxena, *Primes in P*, tech. rep.; wersja czasopismowa w *Annals of Mathematics* 160(2): 781-793, 2004

**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  $f$ ,
- $\forall x \in A \Leftrightarrow f(x) \in B$ .

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  $\phi$  w postaci 3CNF.
  - Pytanie: Czy  $\phi$  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  $c : A \rightarrow \mathbb{Z}^+$ .
  - Pytanie: Czy istnieje podzbiór  $A' \subseteq A$ , taki, że

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

Problemy NP-trudne.