Programowanie dynamiczne

IIUWr. II rok informatyki. *Opracował: Krzysztof Loryś*

2.3 Przynależność do języka bezkontekstowego

2.3.1 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, α) zapisujemy jako $A \to \alpha$.

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 słowo* $\alpha A\beta$ wyprowadza 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=(()()):

$$S \Rightarrow LT \Rightarrow LSR \Rightarrow LSSR \Rightarrow LLRSR \Rightarrow LLRLRR \Rightarrow (LRLRR \Rightarrow$$
$$(LRL)R \Rightarrow (L)L)R \Rightarrow (L)()R \Rightarrow (()()R \Rightarrow (()())$$

□

Problem:
Dla ustalonej gramatyki bezkontekstowej $G = \langle V_N, V_T, P, S\rangle$ w normalnej postaci Chomsky'ego
Dane: słowo $w = a_1 \ldots a_n$ ($a_i \in V_T$ dla $i = 1, \ldots, n$)
Wynik: `"TAK"` - jeśli $w \in L(G)$
`''NIE''` - w przeciwnym przypadku.

2.3.2 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.

2.3.3 Algorytm dynamiczny

Idea:
Jeśli $w = a_1 \ldots a_n$ 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 z $A \stackrel{*}{\Rightarrow} a_1 \ldots a_i$ oraz $B \stackrel{*}{\Rightarrow} a_{i+1} \ldots a_n$.

Na postawie tej obserwacji możemy łatwo zbudować algorym 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 $a_1 \ldots a_i$ (dla $i = 1, \ldots, n-1$); wewnątrz każdego z tych wywołań będzie ona znów wywoływana m.in. dla A i podsłów $a_1 \ldots a_j$ $(j = 1, \ldots, i-1)$.

□

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 $m_{i,j}$ $(1 \leq i \leq j \leq n)$:

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

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

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

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

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

gdzie $m_{i,k} \otimes m_{k+1,j} = \{A \mid (A \rightarrow BC) \in P$ dla pewnych $B \in m_{i,k}$ 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 $B \in m_{i,k}$ **do**
  for each $C \in m_{k+1,j}$ **do**
    if BC jest prawą stroną produkcji z P
    then $m_{i,j} \leftarrow \{$ symbol z lewej strony tej produkcji $\}$

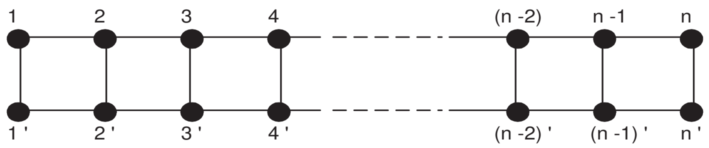
Przy odpowiednim zapamiętaniu informacji o produkcjach, koszt takiego obliczenia nie zależy od liczby produkcji i jest proporcjonalny do iloczynu długości list, co w rozważanym przypadku może być znacznie mniejsze od $|V_N|^2$. Jeśli liczba produkcji jest niewielka opłacalne może być zastosowanie innego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

for each $(A \to BC) \in P$ **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 $\Theta(|P|)$.

2.4 Drzewa rozpinające drabin

Definicja 5 Drabiną *n-elementową nazywamy graf D_n przedstawiony na rysunku 1*



Rysunek 1: *Drabina D_n*

PROBLEM:

Dane: liczby naturalne n, k;
ciąg par liczb naturalnych $\{u_i, v_i\}$ $(i = 1, \ldots, m)$
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 $\{1, \ldots, i, 1', \ldots, i'\}$.

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

- *drzewem rozpinającym drabiny 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 drabiny 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 drabiny D_i, a N_i zbiór lasów rozpinających, o których mowa w Fakcie 1, w drabinie D_i. Naszym celem jest policzenie wartości $|S_n|$.

Idea algorytmu: Kolejno dla $i = 1, \ldots, n$ liczymy wartości $|S_i|$ oraz $|N_i|$, korzystając z zależności przedstawionych w poniższym fakcie:

Fakt 2 *(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) Oczywiste.

(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 izolowaneo 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. □

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 ($j = 0, \ldots, k$) 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 2 pozostawiamy jako proste ćwiczenie.