Wstęp do programowania

Algorytmy zachłanne, programowanie dynamiczne

Paweł Daniluk

Wydział Fizyki

Jesień 2014



Algorytmy zachłanne

Strategia polegająca na budowaniu rozwiązania problemu przez dokonywanie lokalnie optymalnych wyborów.

Algorytmy zachłanne

Strategia polegająca na budowaniu rozwiązania problemu przez dokonywanie lokalnie optymalnych wyborów.

W większości przypadków ta strategia nie prowadzi do optymalnego rozwiązania

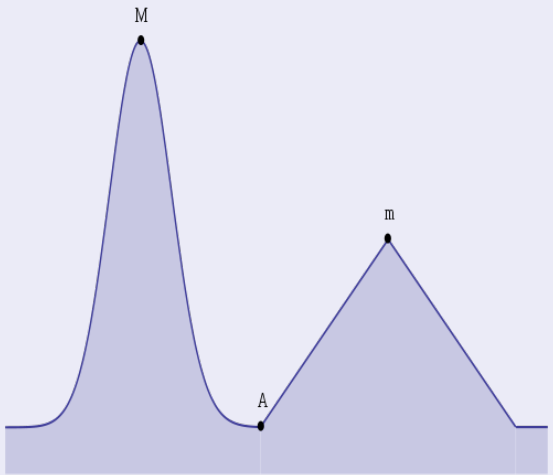
Algorytmy zachłanne

Strategia polegająca na budowaniu rozwiązania problemu przez dokonywanie lokalnie optymalnych wyborów.

W większości przypadków ta strategia nie prowadzi do optymalnego rozwiązania

Analogia

Podchodzenie pod górę:



Algorytmy zachłanne c.d.

Problem wydawania reszty

W jaki sposób dysponując monetami o danych nominałach (np. 5/2/1) wydać resztę używając do tego minimalnej liczby monet.

Rozwiązanie zachłanne polega na wydawaniu monety o największym możliwym nominale. Np.:

$$13 = 5 + 5 + 2 + 1$$

Algorytmy zachłanne c.d.

Problem wydawania reszty

W jaki sposób dysponując monetami o danych nominałach (np. 5/2/1) wydać resztę używając do tego minimalnej liczby monet.

Rozwiązanie zachłanne polega na wydawaniu monety o największym możliwym nominale. Np.:

$$13 = 5 + 5 + 2 + 1$$

Metoda zachłanna nie działa dla każdego zestawu monet. Np. dla monet 5/2:

$$6 = 5 + ?$$

Zastosowania

Własność optymalnej podstruktury

Optymalne rozwiązanie problemu jest funkcją optymalnych rozwiązań podproblemów.

Rozwiązania dokładne (optymalne)

- ortogonalizacja układu wektorów
- kodowanie Huffmana
- problem wyboru czynności
- algorytm Dijkstry

Zastosowania

Własność optymalnej podstruktury

Optymalne rozwiązanie problemu jest funkcją optymalnych rozwiązań podproblemów.

Rozwiązania dokładne (optymalne)

- ortogonalizacja układu wektorów
- kodowanie Huffmana
- problem wyboru czynności
- algorytm Dijkstry

Heurystyki

Algorytmy zachłanne są wydajne. Często się ich używa do znajdowania rozwiązań przybliżonych.

Algorytm Dijkstry

Problem

Dany jest graf nieskierowany z nieujemnymi wagami krawędzi. Wyznaczyć ścieżkę pomiędzy danymi wierzchołkami o najmniejszej wadze.

Przykładowe zastosowanie

Dane są połączenia drogowe pomiędzy miastami. Wyznaczyć najkrótszą trasę.

Algorytm Dijkstry

Algorytm

Przez s oznaczamy wierzchołek źródłowy, $w(i,j)$ to waga krawędzi (i,j).

1. Stwórz tablicę d odległości od źródła dla wszystkich wierzchołków grafu. Na początku $d[s] = 0$, zaś $d[v] = \infty$ dla wszystkich pozostałych wierzchołków.
2. Utwórz kolejkę priorytetową Q wszystkich wierzchołków grafu. Priorytetem kolejki jest aktualnie wyliczona odległość od wierzchołka źródłowego s.
3. Dopóki kolejka nie jest pusta:
   1. Usuń z kolejki wierzchołek u o najniższym priorytecie (wierzchołek najbliższy źródła, który nie został jeszcze rozważony)
   2. Dla każdego sąsiada v wierzchołka u dokonaj relaksacji poprzez u: jeśli $d[u] + w(u,v) < d[v]$ (poprzez u da się dojść do v szybciej niż dotychczasową ścieżką), to $d[v] := d[u] + w(u,v)$.

Na końcu tablica d zawiera najkrótsze odległości do wszystkich wierzchołków.

Dziel i zwyciężaj – przypomnienie

1. Podział problemu na 2 lub więcej części
2. Rozwiązanie każdej z części niezależnie
3. Scalenie rozwiązań części w rozwiązanie problemu pierwotnego

Każdy z podproblemów rozwiązywany jest tą samą metodą.

Naturalna implementacja przy pomocy rekurencji.

Hanoi

1. Podział: Hanoi(n, f, t, h) $\longrightarrow$ Hanoi(n-1, f, h, t), Hanoi(n-1, h, t, f)
2. Scalenie: Hanoi(n-1, f, h, t), f $\rightarrow$ t, Hanoi(n-1, h, t, f)

Dziel i zwyciężaj – przypomnienie

Nie każdy podział jest dobry.

$$F_{n+1} = F_n + F_{n-1}$$

F_n zawiera F_{n-1}.

Programowanie dynamiczne

Rozwiązania podproblemów są zapamiętywane. Gwarantuje to, że nic nie jest liczone dwa razy.

Top-down

Strategia analogiczna do rekurencji z tym, że wyniki wywołań funkcji są zapamiętywane.

Przykład

```
d={0:0, 1:1}

def fib(n, d):
    if n in d:
        return d[n]
    else:
        res=fib(n-1,d)+fib(n-2,d)
        d[n]=res
        return res

fib(5,d)
```

Programowanie dynamiczne

Rozwiązania podproblemów są zapamiętywane. Gwarantuje to, że nic nie jest liczone dwa razy.

Bottom-up

Wiadomo, że trzeba rozważyć wszystkie podproblemy, więc zaczyna się od najmniejszego.

Przykład

```
f=[0,1]

for i in range(2, 6):
    f.append(f[i-1]+f[i-2])
```

Podciąg ściśle rosnący

Dany jest ciąg liczb całkowitych. Podać maksymalną długość ściśle rosnącego podciągu.

Przykład

5 2 8 6 3 6 9 7

Podciąg ściśle rosnący

Dany jest ciąg liczb całkowitych. Podać maksymalną długość ściśle rosnącego podciągu.

Przykład

5 2 8 6 3 6 9 7

s_i – i-ty element ciągu
$L(i)$ – długość maksymalnego rosnącego podciągu kończącego się s_i

$$L(0) = 1$$
$$L(1) = 1$$
$$L(2) = 2$$
$$\ldots$$

Podciąg ściśle rosnący

Dany jest ciąg liczb całkowitych. Podać maksymalną długość ściśle rosnącego podciągu.

Przykład

5 2 8 6 3 6 9 7

s_i – i-ty element ciągu
$L(i)$ – długość maksymalnego rosnącego podciągu kończącego się s_i

$$L(0) = 1$$
$$L(1) = 1$$
$$L(2) = 2$$
$$\ldots$$

$$L(k) = 1 + \max_{i<k, s_i<s_k} L(i)$$

Podciąg ściśle rosnący

Dany jest ciąg liczb całkowitych. Podać maksymalną długość ściśle rosnącego podciągu.

Przykład

5 2 8 6 3 6 9 7

Podciąg ściśle rosnący

Dany jest ciąg liczb całkowitych. Podać maksymalną długość ściśle rosnącego podciągu.

Przykład

5 2 8 6 3 6 9 7

s_i – i-ty element ciągu
$L(i)$ – długość maksymalnego rosnącego podciągu kończącego się s_i

$$L(0) = 1$$
$$L(1) = 1$$
$$L(2) = 2$$
$$\ldots$$

Podciąg ściśle rosnący

Dany jest ciąg liczb całkowitych. Podać maksymalną długość ściśle rosnącego podciągu.

Przykład

5 2 8 6 3 6 9 7

s_i – i-ty element ciągu
$L(i)$ – długość maksymalnego rosnącego podciągu kończącego się s_i

$$L(0) = 1$$
$$L(1) = 1$$
$$L(2) = 2$$
$$\ldots$$

$$L(k) = 1 + \max_{i<k, s_i<s_k} L(i)$$

Podciąg ściśle rosnący c.d.

Algorytm

1. Dla $k = 0, \ldots, n-1$:
   1. oblicz $L(k)$ (używając $L(i)$ dla $i < k$)
2. Oblicz $\max_k L(k)$

Problem plecakowy

Złodziej dysponuje workiem, który może pomieścić przedmioty o łącznej wadze w. W jaki sposób może zmaksymalizować wartość łupu, jeżeli ma do wyboru n rodzajów przedmiotów o wagach i wartościach zadanych ciągami w_i, v_i. Rozważ dwa przypadki:

- Apple store – liczba dostępnych przedmiotów każdego rodzaju jest nieograniczona
- Galeria sztuki – każdy przedmiot występuje w dokładnie jednym egzemplarzu

Przykład

i	w_i	v_i
1	6	\$30
2	3	\$14
3	4	\$16
4	2	\$9

Problem plecakowy

Przykład

i	w_i	v_i
1	6	\$30
2	3	\$14
3	4	\$16
4	2	\$9

Apple store

1, 4, 4 $\longrightarrow$ \$48

Galeria

1, 3 $\longrightarrow$ \$46

Problem plecakowy – z powtórzeniami (Apple Store)

Jaka dekompozycja problemu?

- po rozmiarze plecaka
- po zbiorze przedmiotów

Problem plecakowy – z powtórzeniami (Apple Store)

Jaka dekompozycja problemu?

- po rozmiarze plecaka
- po zbiorze przedmiotów

$K(w)$ – maksymalny łup osiągalny przy pomocy plecaka o pojemności w

Problem plecakowy – z powtórzeniami (Apple Store)

Jaka dekompozycja problemu?

- po rozmiarze plecaka
- po zbiorze przedmiotów

$K(w)$ – maksymalny łup osiągalny przy pomocy plecaka o pojemności w

$$K(w) = \max_{i:w_i \leq w} K(w - w_i) + v_i$$

Problem plecakowy – z powtórzeniami (Apple Store)

Jaka dekompozycja problemu?

- po rozmiarze plecaka
- po zbiorze przedmiotów

$K(w)$ – maksymalny łup osiągalny przy pomocy plecaka o pojemności w

$$K(w) = \max_{i:w_i \leq w} K(w - w_i) + v_i$$

Algorytm wypełnia jednowymiarową tablicę.

Problem plecakowy – bez powtórzeń (galeria)

Jaka dekompozycja problemu?

- po rozmiarze plecaka
- po zbiorze przedmiotów

Problem plecakowy – bez powtórzeń (galeria)

Jaka dekompozycja problemu?

- po rozmiarze plecaka
- po zbiorze przedmiotów

$K(w, i)$ – maksymalny łup osiągalny przy pomocy plecaka o pojemności w i przedmiotów $0, \ldots, i$

Problem plecakowy – bez powtórzeń (galeria)

Jaka dekompozycja problemu?

- po rozmiarze plecaka
- po zbiorze przedmiotów

$K(w, i)$ – maksymalny łup osiągalny przy pomocy plecaka o pojemności w i przedmiotów $0, \ldots, i$

$$K(w, i) = \max\{K(w, i-1), K(w - w_i, i-1) + v_i\}$$

Druga wartość jest brana pod uwagę wtw. gdy $w_i \leq w$.

Problem plecakowy – bez powtórzeń (galeria)

Jaka dekompozycja problemu?

- po rozmiarze plecaka
- po zbiorze przedmiotów

$K(w, i)$ – maksymalny łup osiągalny przy pomocy plecaka o pojemności w i przedmiotów $0, \ldots, i$

$$K(w, i) = \max\{K(w, i-1), K(w - w_i, i-1) + v_i\}$$

Druga wartość jest brana pod uwagę wtw. gdy $w_i \leq w$.

Algorytm wypełnia dwuwymiarową tablicę.

Uliniowienie sekwencji

Para sekwencji DNA

ACACACTA
AGCACACA

Uliniowienie sekwencji

Wstawienie odstępów, tak aby zmaksymalizować jakość uliniowienia mierzoną podobieństwem aminokwasów i ilością odstępów.

A-CACACTA
AGCACAC-A

Uliniowienie sekwencji c.d.

Miara jakości uliniowienia

$$w : \Sigma \cup \{-\} \times \Sigma \cup \{-\} \to \mathbb{R}$$

Na przykład:

$$w(a,b) = \begin{cases} 2 & a = b \\ 0 & a = - \text{ lub } b = - \\ -1 & \text{w p.p.} \end{cases}$$

Algorytm Smitha-Watermana

$S_1 = a_1 a_2 \ldots a_n$
$S_2 = b_1 b_2 \ldots b_m$
$H(i,j)$ – miara najlepszego uliniowienia sekwencji $a_1 a_2 \ldots a_i$, $b_1 b_2 \ldots b_j$

Definicja rekurencyjna

$$H(i,0) = \sum_{k=1}^{i} w(a_k, -),\ 0 \leq i \leq m$$
$$H(0,j) = \sum_{k=1}^{j} w(-, b_k),\ 0 \leq j \leq n$$
$$H(i,j) = \max \left\{ \begin{array}{ll} H(i-1,j-1) + w(a_i, b_j) & \text{dopasowanie} \\ H(i-1,j) + w(a_i, -) & \text{delecja} \\ H(i,j-1) + w(-, b_j) & \text{insercja} \end{array} \right\},\ 1 \leq i \leq m, 1 \leq j \leq n$$

Algorytm Smitha-Watermana c.d.

Przykład

$$H = \begin{pmatrix} & - & A & C & A & C & A & C & T & A \\ - & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ A & 0 & 2 & 1 & 2 & 1 & 2 & 1 & 0 & 2 \\ G & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 \\ C & 0 & 0 & 3 & 2 & 3 & 2 & 3 & 2 & 1 \\ A & 0 & 2 & 2 & 5 & 4 & 5 & 4 & 3 & 4 \\ C & 0 & 1 & 4 & 4 & 7 & 6 & 7 & 6 & 5 \\ A & 0 & 2 & 3 & 6 & 6 & 9 & 8 & 7 & 8 \\ C & 0 & 1 & 4 & 5 & 8 & 8 & 11 & 10 & 9 \\ A & 0 & 2 & 3 & 6 & 7 & 10 & 10 & 10 & 12 \end{pmatrix}$$

Jak wybierać podproblemy?

Kilka standardowych podejść

1. Wejście postaci $x_0, x_1, \ldots, x_n$ – podproblemy postaci $x_0, x_1, \ldots, x_i$
2. Wejście postaci $x_0, x_1, \ldots, x_n$ i $y_0, y_1, \ldots, y_m$ – podproblemy postaci $x_0, x_1, \ldots, x_i$ i $y_0, y_1, \ldots, y_j$
3. Wejście postaci $x_0, x_1, \ldots, x_n$ – podproblemy postaci $x_i, x_{i+1}, \ldots, x_j$
4. Wejście postaci drzewa (ukorzenionego) – podproblemy postaci ukorzenionych poddrzew

Zadanie 1

Zaimplemetuj znajdowanie maksymalnego rosnącego podciągu.

Zadanie 2, 3 – problem plecakowy

Zaimplementuj algorytm rozwiązujący problem plecakowy w wariancje z powtórzeniami i bez.

Zadanie 3 – Smith-Waterman

Napisz program obliczający optymalne globalne uliniowienie podanych sekwencji.

Strona wykładu

http://bioexploratorium.pl/wiki/Wstęp_do_programowania_2014z