Algorytmy i struktury danych

Lista zadań 9

Zadanie 1

Znajdź (bez użycia komputera) optymalne nawiasowanie iloczynu macierzy, których wymiary tworzą ciąg [5, 10, 2, 12, 5, 50, 6]. Spamiętywanie (jeśli go użyjesz) wykonuj na kartce.

Macierz	A_1	A_2	A_3	A_4	A_5	A_6
Rozmiar	5×10	10×2	2×12	12×5	5×50	50×6

1	2	3	4	5	6	m
0	100	220	270	1220	1380	1
	0	240	220	1620	1340	2
		0	120	620	1220	3
			0	3000	1860	4
				0	1500	5
					0	6

2	3	4	5	6	s
1	2	2	2	2	1
	2	2	2	2	2
		3	4	5	3
			4	4	4
				5	5

```
PRINT–OPTIMAL–PARENS(s, i, j)
    if i == j
        print "A" + i
    else print "("
        PRINT–OPTIMAL–PARENS(s, i, s[i, j])
        PRINT–OPTIMAL–PARENS(s, s[i, j] + 1, j)
        print ")"
```

$$((A_1 \cdot A_2) \cdot (((A_3 \cdot A_4) \cdot A_5) \cdot A_6))$$

Zadanie 2

Podane w załączniku (oraz na panoramixie) programy `progdyn1.cc` oraz `progdyn2.cc`, rozwiązujące problem optymalnej triangulacji oraz optymalnego nawiasowania z wykorzystaniem spamiętywania, przepisz w taki sposób, by procedury znajdowania minimum nie korzystały z rekurencji, tylko wypełniały tablicę wyników w pętli w takiej kolejności, by w momencie gdy wyliczana jest wartość `F[i][j]` wszystkie potrzebne do jej wyliczenia elementy tablicy były już wypełnione. Wskazówka: należy wyliczać najpierw te pary `F[i][j]` dla których różnica $j - i$ jest najmniejsza, czyli najpierw wyliczyć wszystkie koszty mnożenia dwóch sąsiednich macierzy, potem trzech, itd.

Zadanie 3

Jaka jest złożoność procedury wyznaczającej koszt optymalnego mnożenia ciągu n macierzy, zaimplementowana metodą z poprzedniego zadania, czyli wstępującą (bez rekurencji). Ile mnożeń wykona ta procedura? (a) w notacji O. (b) dokładnie.

(a) w notacji O

$$\frac{n^3 - n}{3} = O(n^3)$$

(b) dokładna liczba mnożeń

$$2\sum_{i=1}^{n-1} i(n-i) = 2\sum_{i=1}^{n-1} in - i^2 = 2\sum_{i=1}^{n-1} in - 2\sum_{i=1}^{n-1} i^2 =$$
$$= \frac{2(n-1)(n+n(n-1))}{2} - \frac{2(n-1)(n-1+1)(2(n-1)+1)}{6} =$$
$$= (n^3 - n^2) - \frac{2n^3 - 3n^2 + n}{3} = \frac{n^3 - n}{3}$$

Zadanie 4

(0.5pkt) Udowodnij, że wszystkich sposobów pocięcia pręta długości n, na kawałki całkowitej długości, jest 2^{n-1}.

Dla pręta o całkowitej długości n mamy $n-1$ miejsc w których można dokonać cięcia. W każdym z tych miejsc możemy przeciąć lub nie przeciąć pręt, co oznacza że mamy 2^{n-1} możliwości pocięcia pręta na kawałki całkowitej długości.

Zadanie 5

(2pkt) Dany jest długi pręt stalowy długości n będącej całkowitą liczbą centymetrów, oraz tablica `double cena[n+1];` taka, że `cena[i]` określa, za ile można sprzedać kawałek długości i cm dla każdego $1 \leq i \leq n$ (kawałki długości zero są darmowe). Napisz regułę rekurencyjną pozwalającą obliczyć jak pociąć na kawałki całkowitej długości pręt, by najwięcej na tym zarobić:

(a) zakładając, że cięcie jest darmowe

```
tnijDrut(n)
    if n == 0
        return 0
    zysk = -inf
    dlugosci = []
    for i = 1 to n
        zysk_i, dlugosci_i = tnijDrut(n-i)
        if zysk_i + cena[i] > zysk
            zysk = zysk_i + cena[i]
            dlugosci = dlugosci_i + [i]
    return zysk, dlugosci
```

(b) zakładając, że każde przecięcie pręta kosztuje c zł.

```
tnijDrut(n)
    if n == 0
        return 0
    zysk = -inf
    dlugosci = []
    for i = 1 to n
        zysk_i, dlugosci_i = tnijDrut(n-i)
        if zysk_i + cena[i] - c > zysk
            zysk = zysk_i + cena[i] - c
            dlugosci = dlugosci_i + [i]
    return zysk, dlugosci
```

Zadanie 6

Zastosuj programowanie dynamiczne, by napisać program, który rozwiązuje problem z poprzedniego zadania: drukuje rosnącą listę długości kawałków, oraz całkowity zysk (dochód ze sprzedaży, pomniejszony w punkcie (b) o koszt cięcia). Wskazówka: procedura wypełnia tablice `MaxGain` oraz `LongestBit` (albo `ShortestBit`).

Zadanie 7

(2pkt) Optymalne drzewa poszukiwań binarnych. Istnieje wiele drzew BST zawierających dokładnie ten sam (uporządkowany rosnąco) zestaw kluczy. Zastosuj programowanie dynamiczne do znalezienie optymalnego drzewa poszukiwań binarnych (patrz Cormen). Załóż, że koszt wyszukania klucza to liczba węzłów, jakie trzeba odwiedzić, by go znaleźć (dla klucza w korzeniu koszt 1, piętro niżej - koszt 2, itd). Dane do zadania stanowi tablica ile, taka, że `ile[i]` mówi ile razy będzie wyszukiwany i-ty co do wielkości klucz.