Diskretno modeliranje 2013/2014

8. & 9. vaje

Dinamično programiranje

1. Na predavanjih ste napisali algoritem za iskanje izplačila s čim manj kovanci. Program dopolni, tako da vrne še katere kovance uporabiti. Preveri delovanje na primeru $n = 7$ ter $v = [1, 3, 4, 5]$. Pravilen odgovor je $[4, 3]$.

Algoritem 1: minKovancev(n, v)

```
T = {0};
Za vsak i si zapomni, katere kovance smo izbrali! Pripravi seznam seznamov
ustrezne dolžine, kamor boš to shranil.
for i ← 1 to n do
    m = n;
    for c ∈ v do
        if c ≤ i and T_{i−c} < m then
            m = T_{i−c};
            Zapomni si, kateri kovanec smo izbrali, ko je bil dosežen minimum.
        end
        Seznamu za znesek (i − kovanec) dodaj izbrani kovanec.
    end
    T_i ← m + 1;
end
return T_n, Zadnji element seznama s kovanci;
```

Razlaga: V seznamu v podamo vrednosti kovancev, n pa je vrednost, ki jo želimo izplačati. Min. št. kovancev T_n dobimo kot

$$\min_{c \in v; c \leq n} \{T_{n-c}\} + 1,$$

saj smo že prej rešili problem za T_{n-c}, tako da sedaj dodamo le še en kovanec (z vrednostjo c).

2. Na predavanjih ste napisali algoritem za iskanje dolžine najdaljšega skupnega podzaporedja dveh seznamov. Dopolni ga tako, da hkrati vrne tudi skupno podzaporedje. Preveri delovanje na primeru $a = [1, 2, 3, 2, 4, 1, 2]$ ter $b = [2, 4, 3, 1, 2, 1]$.

Algoritem 2: maxSeznam(a, b)

$d = [[0, \ldots, 0], \ldots, [0, \ldots, 0]]$;
d je matrika dimenzije dolžina seznama a × dolžina seznama b;
Vsakemu elementu d_{ij} matrike dolžine d pripada tudi seznama indeksov (skupnega) podzaporedja $s_1(i, j)$ in $s_2(i, j)$.
for $i \leftarrow 1$ **to** *dolzina(a)* **do**
  for $j \leftarrow 1$ **to** *dolzina(b)* **do**
    if $a_{i-1} == b_{j-1}$ **then**
      $d_{i,j} \leftarrow d_{i-1,j-1} + 1$;
      Podzaporedju dodamo vse prejšnje elemente ujemanja ter trenutnega, indeksa dodanega elementa dodamo v seznama $s_1(i, j)$ in $s_2(i, j)$.
    else
      $d_{i,j} \leftarrow \max(d_{i-1,j}, d_{i,j-1})$;
      Podzaporedju dodamo obstoječi seznam, ki je daljši. Indekse elementov, ki so nespremenjeni, si zapomnimo v seznamih $s_1(i, j)$ in $s_2(i, j)$.
    end
  end
end
return $d(dolzina(a), dolzina(b))$, Zadnji element seznama podzaporedij;

Razlaga: Z a in b podamo seznama, $d(i, j)$ pa je dolžina trenutno najdaljšega skupnega podzaporedja zaporedij $a_0, \ldots, a_{i-1}$ ter $b_0, \ldots, b_{j-1}$. Če se na nekem mestu (i, j) ujemata, se ujemanje poveča za 1, sicer se vzame trenutno največje ujemanje, kjer je bodisi prvo bodisi drugo podzaporedje za 1 krajše. Pozor, skupno podzaporedje ni nujno enolično določeno.

3. Na predavanjih ste napisali algoritem za iskanje minimalnega števila operacij pri množenju matrik. Dopolni ga tako, da poveš še vrstni red množenja matrik. Preveri delovanje za $p = [6, 7, 3, 1, 2]$.

Algoritem 3: matrixMult(p)

$n = dolzina(p) - 1$;
$m = n \times n$ matrika ničel;
$s = n \times n$ matrika z vsemi elementi enakimi -1;
Vsakemu elementu matrike m pripada tudi element s, ki pove kje so postavljeni oklepaji.
for $\ell \leftarrow 1$ **to** $n - 1$ **do**
  for $i \leftarrow 1$ **to** $n - \ell$ **do**
    $j = i + \ell$;
    $q = \{\}$;
    for $k \leftarrow i$ **to** $j - 1$ **do**
      $q \leftarrow \{q, m_{i-1,k-1} + m_{k,j-1} + p_{i-1} \cdot p_k \cdot p_j\}$;
    end
    $m_{i-1,j-1} = \min\{q\}$;
    V istoležni element matrike s shranimo vrednost k, kjer je dosežen minimum;
    $s_{i-1,j-1} =$ (indeks minimalnega elementa v q) $+ i$;
  end
end
return $m_{0,n-1}, s$, s je matrika s položaji oklepajev;

Razlaga: Z n označimo število matrik, ki jih množimo. Poiskati moramo vse vrednosti $m[i, j]$, ki nam povedo najmanjše št. operacij pri množenju matrik $A_i \cdot A_{i+1} \cdots A_j$. Pri tem nam k pove, kje med matrikami postavimo oklepaj. Npr. pri $m[0, 4]$ (op: v pythonu sta indeksa za ena manjša) pomeni izbor $k = 2$, da množimo $(A_1A_2)(A_3A_4A_5)$.

4. Fibonaccijevo zaporedje je definirano z:

$$F_i := \begin{cases} 0, & i = 0; \\ 1, & i = 1; \\ F_{i-1} + F_{i-2}, & i > 1. \end{cases}$$

Zapiši program $fib(i)$ za računanje n-tega člena Fibonaccijevega zaporedja. Nato zapiši še program $fibM(i)$, ki bo to zaporedje izračunal tako, da si bo shranjeval vmesne rezultate v globalen slovar $memo = \{indeks : vrednost\}$. Primerjaj njuno časovno zahtevnost.