Pete vaje APS2: dinamično programiranje

Naloga 1: optimalno množenje matrik

Naloga

Množenje matrik je asociativno, kljub temu pa z vidika števila izvedenih operacij ni vseeno, v kakšnem vrstnem redu jih množimo, če matrike niso kvadratne. Na primer, če matrike A velikosti 7×3, B velikosti 3×5 in C velikosti 5×4 zmnožimo kot $(AB)C$, izvršimo $7 \cdot 3 \cdot 5 + 7 \cdot 5 \cdot 4 = 245$ skalarnih množenj, pri vrstnem redu $A(BC)$ pa imamo $3 \cdot 5 \cdot 4 + 7 \cdot 3 \cdot 4 = 144$ skalarnih množenj.

Podano je zaporedje n matrik $(A_0, A_1, \ldots, A_{n-1})$ velikosti $v_0 \times v_1$, $v_1 \times v_2$, $\ldots$, $v_{n-1} \times v_n$ (v tem vrstnem redu). Zapišite algoritem, ki določi vrstni red množenj matrik, ki minimizira število skalarnih množenj.

Razmislek

Na začetku se ne bomo ukvarjali z dejanskim vrstnim redom skalarnih množenj, ampak samo z minimalnim številom skalarnih množenj (ki jih seveda proizvede optimalni vrstni red).

Lahko razmišljamo podobno kot pri 0/1-nahrbtniku? Ena možnost je, da zmnožek matrik zapišemo kot $A_0(A_1 \ldots A_{n-1})$ in rekurzivno izračunamo minimalno število skalarnih množenj za zmnožek $A_1 \ldots A_{n-1}$. Druga možnost pa je $\ldots$ hm, težka bo, saj lahko prvo matriko na različne načine vključimo v prvi podproblem.

Sledeči razmislek bo obetavnejši. Zmnožek $A_0 A_1 \ldots A_{n-1}$ lahko razbijemo na $(A_0 A_1 \ldots A_t)$ $(A_{t+1} A_{t+2} \ldots A_{n-1})$ za nek $t \in [0, n-2]$. Sedaj rekurzivno izračunamo minimalno število skalarnih množenj za zmnožek $A_0 A_1 \ldots A_t$ (dobljeno število označimo s p) in minimalno število skalarnih množenj za zmnožek $A_{t+1} A_{t+2} \ldots A_{n-1}$ (dobljeno število označimo s q). Rezultat prvega zmnožka je matrika B velikosti $v_0 \times v_{t+1}$, rezultat drugega zmnožka pa je matrika C velikosti $v_{t+1} \times v_n$. Matriki B in C moramo na koncu še zmnožiti. Skupno število skalarnih množenj za izbrano razcepišče t je potemtakem $p + q + v_0 v_{t+1} v_n$.

Smo na dobri poti, vendar pa moramo upoštevati, da lahko razcepišče t postavimo kamorkoli med 0 in vključno $n-2$. Med rezultati, ki jih dobimo za različne postavitve razcepišča, izberemo seveda najmanjšega. Druga stvar, ki jo moramo upoštevati, pa je ta, da se z zmnožkom $A_0 A_1 \ldots A_{n-1}$ ukvarjamo samo na začetku. V procesu rekurzivnega razcepa bodo nastajali zmnožki $A_i A_{i+1} \ldots A_j$ za različne i in j $(0 \leqslant i \leqslant j \leqslant n-1)$.

Rekurenčna enačba

Sedaj lahko zapišemo rekurenčno enačbo za minimalno število skalarnih množenj pri računanju zmnožka $A_i A_{i+1} \ldots A_j$:

$$M(i,j) = \begin{cases} 0, & \text{če } i = j; \\ \min_{t=i}^{j-1} (M(i,t) + M(t+1,j) + v_i v_{t+1} v_{j+1}) & \text{če } i < j. \end{cases}$$

Primer

Oglejmo si vrednosti $M(i,j)$ za matrike A_0, $\ldots$, A_4 z velikostmi 7, 3, 8, 2, 6, 5 (velikost A_0 je torej 7×3, velikost A_1 je 3×8 itd.):

			j		
i	0	1	2	3	4
0	0	168	90	174	220
1	–	0	48	84	138
2	–	–	0	96	140
3	–	–	–	0	60
4	–	–	–	–	0

Na primer, končni rezultat ($M(0,4)$) izračunamo kot

$$\begin{aligned} M(0,4) &= \min\{M(0,0) + M(1,4) + v_0v_1v_5,\ M(0,1) + M(2,4) + v_0v_2v_5,\\ &\qquad M(0,2) + M(3,4) + v_0v_3v_5,\ M(0,3) + M(4,4) + v_0v_4v_5\}\\ &= \min\{0 + 138 + 7\cdot 3\cdot 5,\ 168 + 140 + 7\cdot 8\cdot 5,\ 90 + 60 + 7\cdot 2\cdot 5,\ 174 + 0 + 7\cdot 6\cdot 5\}\\ &= 220 \end{aligned}$$

Rešitev »od zgoraj navzdol«

Od rekurenčne enačbe je samo še korak do rekurzivnega algoritma, ki ga bomo kar takoj opremili z memoizacijo:

```
function OPT-MNOŽENJE(i, j)
    if i = j then
        return 0
    if D[i, j] ⩾ 0 then
        return D[i, j]
    D[i, j] ← ∞
    for t ← i to j − 1 do
        m ← OPT-MNOŽENJE(i, t) + OPT-MNOŽENJE(t + 1, j) + v_i v_{t+1} v_{j+1}
        D[i, j] ← min(D[i, j], m)
    return D[i, j]
```

Memoizacijsko tabelo D ustvarimo kot tabelo velikosti $n \times n$ in jo inicializiramo z vrednostmi -1. Funkcijo pokličemo kot OPT-MNOŽENJE$(0, n-1)$.

Rešitev »od spodaj navzgor«

Tabelo D lahko polnimo tudi iterativno; paziti moramo le na vrstni red polnjenja. Ker je vrednost $M(i,j)$ odvisna le od vrednosti $M(p,q)$, kjer je $q - p < j - i$, lahko tabelo D polnimo po naraščajoči razliki med indeksoma, torej po diagonalah. Najprej bomo celice $D[i,i]$ za vse $i \in \{0,\ldots,n-1\}$ nastavili na 0, nato bomo po rekurenčni enačbi polnili celice $D[i,i+1]$, nato $D[i,i+2]$ itd.

```
function OPT-MNOŽENJE-ITER(v_0, ..., v_n)
    ustvari tabelo D velikosti n × n
    for i ← 0 to n − 1 do
        D[i, i] ← 0
    for r ← 1 to n − 1 do
        for i ← 0 to n − r − 1 do
            j ← i + r
            D[i, j] ← ∞
            for t ← i to j − 1 do
```

$m \leftarrow D[i,t] + D[t+1,j] + v_i v_{t+1} v_{j+1}$
$D[i,j] \leftarrow \min(D[i,j], m)$
return $D[0, n-1]$

Računska zahtevnost

Pri obeh rešitvah je prostorska zahtevnost enaka $O(n^2)$, časovna pa $O(n^3)$. Pri iterativni rešitvi je to precej očitno, pri rekurzivni pa velja sledeči razmislek: vsako celico tabele D bomo napolnili kvečjemu enkrat, vsako polnjenje pa traja $O(n)$ časa (zanka po t). Ker je tabela velika $O(n^2)$, je skupna časovna zahtevnost enaka $O(n^3)$.

Optimalni vrstni red množenj

Doslej smo se ukvarjali zgolj z minimalnim *številom* skalarnih množenj, za dejanski optimalni *vrstni red* množenja matrik pa bomo morali vložiti še nekaj truda. Pri obeh rešitvah moramo za vsak par i in j shraniti optimalno razcepišče t, nato pa dobljena razcepišča uporabiti za določitev vrstnega reda množenj.

Kako pridobimo optimalna razcepišča? Poleg tabele D ustvarimo še tabelo Z velikosti $n \times n$, nato pa stavek

$D[i,j] \leftarrow \min(D[i,j], m)$

nadomestimo s kodo

if $m < D[i,j]$ **then**
$D[i,j] \leftarrow m$
$Z[i,j] \leftarrow t$

V obravnavanem primeru (7, 3, 8, 2, 6, 5) dobimo takšno tabelo Z:

	j				
i	0	1	2	3	4
0	–	0	0	2	2
1	–	–	1	2	2
2	–	–	–	2	2
3	–	–	–	–	3
4	–	–	–	–	–

Ker je $Z[0,4] = 2$, vemo, da moramo matrike zmnožiti kot $(A_0A_1A_2)(A_3A_4)$. Optimalni razcepišči podproblemov sta torej shranjeni v celicah $Z[0,2]$ in $Z[3,4]$. Vidimo, da je $Z[0,2] = 0$, zato zmnožek $A_0A_1A_2$ izračunamo kot $A_0(A_1A_2)$. Optimalni vrstni red množenja je torej

$$(A_0(A_1A_2))(A_3A_4).$$

Sledeča funkcija, ki jo pokličemo kot IZPIŠI-OKLEPAJE$(0, n-1)$, izpiše optimalno postavitev oklepajev ...

function IZPIŠI-OKLEPAJE(i, j)
if $i = j$ **then**
PRINT$(i,$ '␣')
else
$t \leftarrow Z[i,j]$
PRINT('(')
IZPIŠI-OKLEPAJE(i, t)

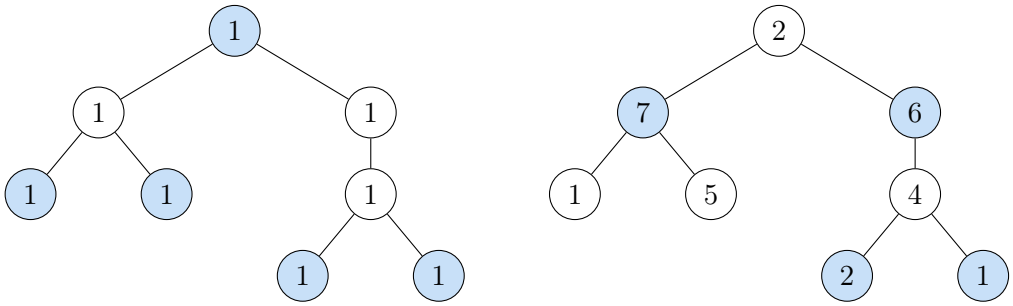
```
PRINT(')(')
IZPIŠI-OKLEPAJE(t + 1, j)
PRINT(')')
```

... bralec in bralka pa jo bosta zlahka predelala tako, da bo izpisala samo tiste oklepaje, ki so res potrebni (gornja funkcija jih izpiše bistveno preveč!).

Naloga 2: najtežja neodvisna množica na drevesu

Podan je neusmerjen graf (V, E), pri katerem ima vsako vozlišče v utež $W(v)$. Pri problemu najtežje neodvisne množice iščemo množico $U \subseteq V$ z največjo vsoto uteži, tako da noben par vozlišč v množici U ni med seboj povezan. Pri splošnih grafih je problem NP-težak, pri drevesih pa ga je, kot bomo videli, mogoče reševati v polinomskem času.

Oglejmo si dva primera. V prvem primeru so vse uteži enake, zato bo rešitev kar *največja* neodvisna množica, v drugem primeru pa dobimo manjšo, a težjo množico:



Naše drevo naj ima vozlišča $\{0, \ldots, n-1\}$. Zaradi enostavnosti bomo predpostavili, da je drevo »ukoreninjeno« (če ni, lahko poljubno vozlišče proglasimo za koren) in da številke vozlišč naraščajo od korena proti listom. Koren ima potemtakem številko 0, za vsako povezavo (u, v) pa velja $u < v$. Takšen vrstni red lahko v času $O(n)$ vzpostavimo s preprostim algoritmom na osnovi iskanja v globino (DFS). Poleg naštetega bomo s $C(u)$ označili množico otrok vozlišča u.

Pričnimo s korenom, torej z vozliščem 0. Dve možnosti imamo:

- Koren lahko dodamo v neodvisno množico M. Če se za to odločimo, potem v M ne smemo dodati nobenega od njegovih otrok.
- Lahko se odločimo, da korena ne bomo dodali v množico M. V tem primeru enak postopek odločanja rekurzivno izvedemo za vse otroke vozlišča 0. (Pozor: otroke bomo v tem primeru *lahko* dodali, ni pa nujno, da je to tudi optimalna odločitev!)

V postopku rekurzivnega razcepa bomo isti razmislek opravili za poljubno vozlišče u. Označimo z $S_{\text{lahko}}(u)$ težo najtežje neodvisne množice, ki jo dobimo, če imamo proste roke glede izbire vozlišča u, z $S_{\text{ne}}(u)$ pa težo najtežje neodvisne množice, ki jo dobimo, če vozlišča u ne dodamo oz. ne smemo dodati vanjo. Iščemo torej vrednost $S_{\text{lahko}}(0)$.

Zapišimo najprej enačbo za $S_{\text{ne}}(u)$:

$$S_{\text{ne}}(u) = \sum_{v \in C(u)} S_{\text{lahko}}(v).$$

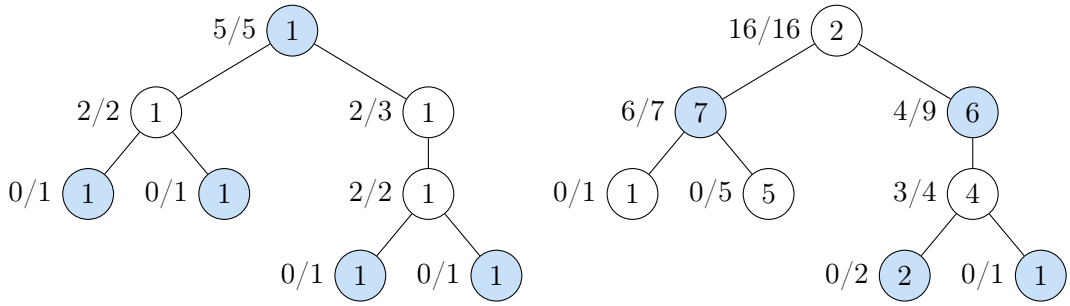
Vsakega od otrok lahko dodamo v neodvisno množico, skupna teža neodvisne množice pa je vsota tež najtežjih neodvisnih množic, ki nastanejo iz posameznih otrok.

Enačbo za $S_{\text{lahko}}(u)$ pa lahko zapišemo takole:

$$S_{\text{lahko}}(u) = \max(W(u) + \sum_{v \in C(u)} S_{\text{ne}}(v),\ S_{\text{ne}}(u))$$

Prva možnost je, da vozlišče u dodamo v neodvisno množico (v tem primeru otrok vanjo ne dodamo), druga pa je, da vozlišča u ne dodamo v množico. Izberemo seveda tisto možnost, ki nam dá težjo neodvisno množico.

Na sledeči sliki sta za vsako vozlišče u prikazani vrednosti $S_{\text{ne}}(u)$ in $S_{\text{lahko}}(u)$ (v obliki $S_{\text{ne}}/S_{\text{lahko}}$):



Na primer, če vozlišča s težo 6 na desni sliki ne vzamemo, lahko za njegovo poddrevo pridelamo največ težo 4, če ga vzamemo, pa lahko dobimo težo 9 $(6 + 2 + 1)$.

Bralec in bralka bosta rekurzivno različico z memoizacijo zapisala za vajo, tukaj pa bomo zapisali le iterativno različico. Naj bo D naša tabela in naj element $D[u, 0]$ hrani vrednost $S_{\text{ne}}(u)$, element $D[u, 1]$ pa vrednost $S_{\text{lahko}}(u)$. Iz rekurenčnih enačb lahko razberemo, da moramo tabelo D polniti od listov proti korenu (torej po padajočih številkah vozlišč), pri istem vozlišču u pa moramo najprej izračunati $D[u, 0]$ in šele potem $D[u, 1]$, saj v enačbi za $S_{\text{lahko}}(u)$ nastopa $S_{\text{ne}}(u)$.

```
function NAJTEŽJA-NEODVISNA-MNOŽICA(C, W)
    for u ← n − 1 downto 0 do
        t ← [0, 0]
        for all v ∈ C(u) do
            t[0] ← t[0] + D[v, 0]
            t[1] ← t[1] + D[v, 1]
        D[u, 0] ← t[1]
        D[u, 1] ← max(W(u) + t[0], D[u, 0])
    return D[0, 1]
```

Algoritem lahko na običajen način prilagodimo tako, da bomo dobili tudi najtežjo neodvisno množico (ne samo njeno težo), povejmo pa še nekaj o časovni zahtevnosti. Zaradi dvojne zanke bi časovno zahtevnost na prvi pogled postavili na $O(n^2)$, kar je sicer pravilna zgornja meja, ni pa tesna. Skupno število izvedb telesa notranje zanke je namreč $\sum_{u=0}^{n-1} |C(u)|$, torej število otrok vseh vozlišč skupaj, to pa je preprosto $n-1$. Časovna zahtevnost algoritma je potemtakem $\Theta(n)$. To je tudi teoretična spodnja meja za ta problem, saj moramo (če ne drugega) drevo vsaj prebrati.