Računalništvo (Fizika): 2. izpit

28. junij 2012

Čas reševanja: 120 minut. Doseženih 100 točk šteje za maksimalno oceno.

Naloga 1 [25 + 10 točk]

Ortogonalna matrika dimenzije 2×2 je kvadratna matrika

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix}$$

z realnimi koeficienti a, b, c, d, v kateri sta stolpca normirana, se pravi $a^2 + c^2 = 1$ in $b^2 + d^2 = 1$ in ortogonalna, se pravi $ab + cd = 0$. Taka matrika je obrnljiva, njen inverz pa je kar transponirana matrika.

Kadar računamo z ortogonalnimi matrikami, moramo zaradi zaokrožitvenih napak dovoliti določeno odstopanje (toleranco) glede pogojev, da sta stolpca normirana in ortogonalna. Če je na primer t mera tolerance, ki si jo predstavljamo kot majhno pozitivno število, pravimo, da je vektor (x, y) normiran s toleranco t, če je njegova norma $\sqrt{x^2 + y^2} = 1 \pm t$. Podobno dopustimo pri ortogonalnsti vektorjev odstopanje $\pm t$ od ničle (skalarni produkt).

Sestavi razred `Ortogonalna`, ki predstavlja ortogonalno matriko dimenzije 2×2, ki ima spodaj naštete metode. Vse metode naj vedno preverijo podano toleranco in sprožijo izjemo, če ta ni dosežena.

(a) Konstruktor `__init__(self, u, v, t=10e-10)`, kjer sta $u = (a, c)$ in $v = (b, d)$ stolpca matrike in t toleranca.

(b) Metodo `__call__(self, w)`, ki za dani vektor $w = (x, y)$ vrne rezultat množenja matrike `self` in vektorja w. (Tu toleranca ne pride v poštev.)

(c) Metodo `__mul__(self, m)`, ki vrne produkt matrik `self` in `m`. Toleranca produkta je vsota toleranc množenih matrik.

(d) **(dodatna, +10 točk)** Metodo `__pow__(self, n)`, ki za dano matriko `self` in celo število `n` vrne `n`-to potenco matrike. Če je `n` negativno število, je treba izračunati ustrezno potenco inverzne matrike.

Potenco računaj učinkovito s pomočjo formul $M^{2k} = (M^k)^2$ in $M^{2k+1} = M \cdot (M^k)^2$, ki za n-to potenco potrebuje kvečjemu $O(\log n)$ množenj. Ustrezno premisli tudi, kakšna naj bo toleranca potence.

Seveda lahko definirate še kakšne druge metode, ki olajšajo delo, na primer `__repr__(self)`.

Naloga 2 [25 točk]

Iskalno drevo je podano z razredom `IskalnoDrevo`. Vozlišča drevesa vsebujejo cela števila. Dodatno vsako vozlišče vsebuje še spremenljivko n, ki pove, koliko elementov je v (pod)drevesu, katerega koren je.

(a) Dodaj metodo `inicializiraj(self)`, ki v vseh vozliščih iskalnega drevesa ustrezno izpolni spremenljivko n.

(b) Dodaj metodo `kTi(self, k)`, ki ob učinkoviti izrabi podatkov n v vozliščih vrne k-ti element glede na urejenost po velikosti od najmanjšega do največjega. Če k-tega elementa ni, vrne `None`.

(c) Dodaj metodo `inteval(self, a, b)`, ki ob učinkoviti izrabi podatkov n v vozliščih vrne število elementov v drevesu, za katere velja $a \leq x \leq b$. Pozor, ni nujno, da se a in b pojavita v drevesu.

(obrni list)

Naloga 3 [25 + 10 točk]

Za dano zaporedje celih števil $a = [a_0, a_1, \ldots, a_{n-1}]$ želimo najti najdražje strnjeno podzaporedje (NSP). Npr. v zaporedju $[-3, -16, -23, 18, 20, -7, 12, -5, -22]$ je eno od možnih takih NSP $[18, 20, -7, 12]$ s ceno 43 (t.j. vsoto podzaporedja).

(a) Naj za dano zaporedje a, vednost $s(i, j)$ predstavlja ceno podzaporedja $[a_i, a_{i+1}, \ldots, a_{j-1}]$. Vemo:

$$s(i, j) = s(1, j) - s(1, i).$$

Z upoštevanjem tega sestavi funkcijo `nspKvad(a)`, ki v času $O(n^2)$ (ali bolje) določi in vrne ceno ter indekse začetka in konca enega izmed NSP-jev za zaporedje a. Funkcija torej vrne trojico (c, i, j), kjer je c cena NSP-ja $[a_i, a_{i+1}, \ldots, a_{j-1}]$.

(b) Za dan indeks i, $0 \leq i < n - 1$ napiši funkcijo `nspVseb(a, i)`, ki v času $O(n)$ (ali manj) poišče NSP, ki vsebuje elementa a_i in a_{i+1}. Pazi, to ni nujno NSP celotnega zaporedja ampak le najdražja izbira strnjenega podzaporedja pri navedenem pogoju! Funkcija vrne trojico kot v točki (a).

(c) S pomočjo strategije *deli in vladaj* ter prejšnje točke izpelji hitrejši algoritem in napiši funkcijo `nspDV(a)`, ki vrne trojico podobno kot v prejšnjih dveh točkah. Namig: če tabelo razdeliš na polovico, upoštevaj, da se NSP lahko nahaja v celoti v levi ali desni polovici ali pa vsebuje dele iz obeh polovic. Utemelji časovno zahtevnost!

Naloga 4 [25 točk]

Kljub trudu pri prejšnji nalogi pa je mogoče s pomočjo dinamičnega programiranja NSP za zaporedje določiti na še bolj učinkovit način. Za dano zaporedje a kot v prejšnji nalogi naj $M(a, j)$ predstavlja ceno NSP za $[a_0, a_1, \ldots, a_{j-1}]$, medtem ko $m(a, j)$ predstavlja ceno najdražjega strnjenega podzaporedja istega zaporedja, le da podzaporedje vsebuje kot zadnji člen element a_{j-1}. Potem velja:

$$\begin{aligned} M(a, 0) &= m(a, 0) = 0, \\ M(a, j) &= \max\{M(a, j-1), m(a, j)\} \\ m(a, j) &= \max\{a_{j-1}, m(a, j-1) + a_{j-1}\} \end{aligned}$$

(a) Sestavi funkcijo `cene(a)`, ki vrne tabelo parov

$$[(M(a, 0), m(a, 0)), (M(a, 1), m(a, 1)), \ldots, (M(a, n-1), m(a, n-1))].$$

Funkcija naj bo učinkovita tudi za večje tabele. Oceni časovno zahtevnost svoje funkcije (odgovor zapiši kot komentar v kodi).

(b) Sestavi funkcijo `nspDyn(a)`, ki vrne enega od NSP-jev za seznam a. Upoštevaj dejstvo, da če je $M(a, j) = m(a, j)$ potem obstaja NSP za zaporedje $[a_0, a_1, \ldots, a_{j-1}]$, ki vsebuje zadnji element a_{j-1}.