Druge vaje APS2: algoritem quickselect z mediano median

1 Problem

Naj bo *rang* elementa v tabeli dolžine n indeks, na katerem bi se ta element nahajal, če bi tabelo naraščajoče uredili. Najmanjši element ima tako rang 0, drugi najmanjši rang 1, ..., največji pa rang $n-1$.[1]

Želimo rešiti problem iskanja elementa z rangom k v podani tabeli dolžine n. Zaradi enostavnosti bomo predpostavili, da so vsi elementi v tabeli med seboj različni, seveda pa algoritma, ki ju bomo predstavili, delujeta tudi takrat, ko to ne drži (v resnici moramo le besedi »manjši« in »večji« zamenjati z »manjši ali enak« in »večji ali enak«).

2 Osnovni algoritem quickselect

Algoritem quickselect se uporablja za iskanje elementa z rangom k. Deluje podobno kot quicksort:

- Izbere pivot. V osnovni različici bomo za pivot proglasili enostavno zadnji element tabele.
- Izvede porazdelitev tabele glede na pivot. To pomeni, da pivot prestavi na indeks p, vse elemente, ki so manjši od pivota, prestavi na indekse $0, \ldots, p-1$, vse elemente, ki so večji od pivota, pa prestavi na indekse $p+1, \ldots, n-1$. Po izvedbi porazdelitve je potemtakem element na indeksu p (torej pivot) ravno element z rangom p.
- Če je $k < p$, vemo, da se element z rangom k nahaja levo od indeksa p, zato ponovimo iskanje na podtabeli, določeni z intervalom indeksov $[0, p-1]$.
- Če je $k > p$, vemo, da se element z rangom k nahaja desno od indeksa p, zato ponovimo iskanje na podtabeli, določeni z intervalom indeksov $[p+1, n-1]$.

2.1 Implementacija

Algoritem quickselect lahko sprogramiramo takole:

```
// Vrne element z rangom k v podanem vektorju (rang 0 = najmanjši element).
int quickselect(vector<int>& v, int k) {
    return quickselect(v, 0, v.size() - 1, k);
}

// Vrne element z rangom k v podanem vektorju (rang 0 = najmanjši element),
// pri čemer obravnava samo elemente z indeksi od lm do vključno dm.
int quickselect(vector<int>& v, int lm, int dm, int k) {
    if (lm == dm) {
```

[1] Lepše je šteti od 1 naprej, a če že indekse začenjamo z 0, dajmo še range ...

```
        if (k != lm) throw runtime_error("Nekaj smrdi!");
        return v[lm];
    }

    // Izberemo pivot in ga prestavimo na zadnje mesto (funkcija porazdeli
    // pričakuje, da je pivot na koncu vektorja).
    int ixPivot = izberiPivot(v, lm, dm);
    swap(v[ixPivot], v[dm]);

    // Izvedemo porazdelitev glede na pivot.
    int p = porazdeli(v, lm, dm);

    // V tem trenutku je v[p] element z rangom p, saj so vsi elementi levo od
    // indeksa p manjši od v[p], vsi elementi desno od indeksa p pa večji od v[p].

    if (p == k) {
        // juhuhu!
        return v[p];
    }
    if (p > k) {
        // iskani element se nahaja levo od indeksa p
        return quickselect(v, lm, p - 1, k);
    }
    // iskani element se nahaja desno od indeksa p
    return quickselect(v, p + 1, dm, k);
}

// Vrne indeks pivota v vektorju v, pri čemer se upoštevajo zgolj elementi z
// indeksi od lm do vključno dm. V tem primeru je pivot preprosto zadnji element.
int izberiPivot(const vector<int>& v, int lm, int dm) {
    return dm;
}
```

Funkcija `porazdeli` preuredi podano tabelo glede na pivot, pri čemer pričakuje, da se pivot nahaja na indeksu $n-1$. Funkcija vrne indeks p, na katerem ob koncu porazdeljevanja pristane pivot.

```
int porazdeli(vector<int>& v, int lm, int dm) {
    int pivot = v[dm];
    int i = lm - 1;
    for (int j = lm; j <= dm - 1; j++) {
        if (v[j] <= pivot) {
            i++;
            swap(v[i], v[j]);
        }
    }
    swap(v[i + 1], v[dm]);
    return i + 1;
}
```

Na začetku vsakega obhoda zanke velja naslednja invarianta:

- elementi na indeksih $[l,\ i]$ so manjši od pivota;[2]
- elementi na indeksih $[i+1,\ j-1]$ so večji od pivota;
- elementov na indeksih $[j,\ d-1]$ še nismo obiskali;
- na indeksu d se nahaja pivot.

Ko se zanka zaključi, bodo torej elementi na indeksih $[l,\ i]$ manjši od pivota, tisti na indeksih $[i+1,\ d-1]$ pa večji od pivota. Pivot na indeksu d le še zamenjamo z elementom na indeksu $i+1$ in algoritem se zaključi. Delilna točka je $i+1$.

2.2 Primer izvedbe

Recimo, da iščemo element z rangom 5 v tabeli [7, 3, 15, 5, 12, 14, 11, 8, 4, 6].

- Zadnji element (torej 6) proglasimo za pivot.
- Tabelo porazdelimo glede na pivot (element na indeksu i je podčrtan):
  - $i = -1,\ j = 0.$

    $$[7, 3, 15, 5, 12, 14, 11, 8, 4, 6]$$

  - $i = -1,\ j = 1$: $v[1] < pivot$, $v[0] \leftrightarrow v[1]$.

    $$[\underline{3}, 7, 15, 5, 12, 14, 11, 8, 4, 6]$$

  - $i = 0,\ j = 3$: $v[3] < pivot$, $v[1] \leftrightarrow v[3]$.

    $$[3, \underline{5}, 15, 7, 12, 14, 11, 8, 4, 6]$$

  - $i = 1,\ j = 8$: $v[8] < pivot$, $v[2] \leftrightarrow v[8]$.

    $$[3, 5, \underline{4}, 7, 12, 14, 11, 8, 15, 6]$$

  - $v[3] \leftrightarrow v[9]$.

    $$[3, 5, 4, \underline{6}, 12, 14, 11, 8, 15, 7]$$

- Pivot je pristal na indeksu $p = 3$. Ker je iskani rang večji od p, omejimo iskanje na podtabelo desno od indeksa p:

  $$[\langle 4 \text{ elementi} \rangle, 12, 14, 11, 8, 15, 7]$$

- Za pivot proglasimo element 7 in znova izvedemo porazdelitev:

  $$[\langle 4 \text{ elementi} \rangle, \underline{7}, 14, 11, 8, 15, 12]$$

- Tokrat je $p = 4$, kar je še vedno manjše od iskanega ranga, zato iskanje znova omejimo na desni del tabele:

  $$[\langle 5 \text{ elementov} \rangle, 14, 11, 8, 15, 12]$$

- Pivot je tokrat element 12:

  $$[\langle 5 \text{ elementov} \rangle, 11, 8, \underline{12}, 15, 14]$$

[2] $l =$ `lm`, $d =$ `dm`

- Tokrat je $p = 7$, kar je večje od iskanega ranga, zato iskanje omejimo na levi del podtabele:

$$[\langle 5 \text{ elementov} \rangle, 11, 8, \langle 3 \text{ elementi} \rangle]$$

- Za pivot vzamemo element 8. Po porazdelitvi ta pristane na indeksu 5 ...

$$[\langle 5 \text{ elementov} \rangle, \underline{8}, 11, \langle 3 \text{ elementi} \rangle]$$

  ... zato je element na rangu 5 enak 8.

2.3 Časovna zahtevnost

Algoritem quickselect ima v povprečju časovno zahtevnost $\Theta(n)$. To velja tudi v precej neugodnih primerih. Na primer, če dolžino tabele vsakokrat zmanjšamo za 1 odstotek, dobimo

$$T(n) = T(99n/100) + cn,$$

pri čemer je člen $cn = \Theta(n)$ ocena časovne zahtevnosti porazdeljevanja. S popolno indukcijo (substitucijska metoda, Cormen 4.1) lahko dokažemo, da velja $T(n) \leqslant 100cn$. Predpostavimo, da trditev velja za 1, 2, ..., $n-1$, in pokažimo, da velja tudi za n:

$$T(n) = T(99n/100) + cn \leqslant 99(100cn)/100 + cn = 100cn.$$

Kot bomo spoznali na kasnejših vajah, lahko dejstvo, da je $T(n) = \Theta(n)$, pokažemo tudi z uporabo krovnega izreka.

Zalomi se nam le v najslabših možnih primerih. Če dolžino tabele vsakokrat zmanjšamo za kvečjemu M, kjer je M neka fiksna konstanta, dobimo $\Theta(n^2)$. To zlahka preverimo: $n + (n - M) + (n - 2M) + \ldots + M = \frac{1}{2}n(\frac{n}{M} + 1) = \Theta(n^2)$.

3 Algoritem quickselect z mediano median

Časovna zahtevnost algoritma quickselect je odvisna od izbire pivota. Če za pivot vsakokrat izberemo zadnji (ali pa prvi) element, nas bo $\Theta(n^2)$ doletelo vsakokrat, ko bo tabela urejena. Temu scenariju se lahko izognemo tako, da, denimo, pivot določimo kot mediano treh naključno izbranih elementov. Čeprav ta postopek v praksi dobro deluje, je *teoretično* še vedno možno, da dobimo $\Theta(n^2)$. Sedaj stopi v igro mediana median, pristop, ki *v vseh možnih primerih* zagotavlja zahtevnost $\Theta(n)$.

Algoritem quickselect z mediano median poteka takole:

1. Vhodno tabelo razdelimo na odseke po 5 elementov. Če n ni deljiv s 5, bo zadnji odsek krajši.
2. Za vsak odsek izračunamo mediano.[3] Dobljene mediane shranimo v novo tabelo.
3. Poiščemo mediano tabele median. To lahko storimo z rekurzivnim klicem funkcije za quickselect: poiščemo element z rangom $\lfloor (m-1)\,/\,2 \rfloor$, kjer je m dolžina tabele.
4. Mediana služi kot pivot. Nadaljujemo enako kot pri osnovnem quickselectu.

[3] Menda lahko mediano petih elementov izračunamo s pičlimi šestimi primerjavami. Že mogoče, a v resnici ni tako važno. Ker je število elementov konstantno, ima vsak algoritem za ta problem zahtevnost $\Theta(1)$. Cormen priporoča kar urejanje z navadnim vstavljanjem (insertion sort).

3.1 Implementacija

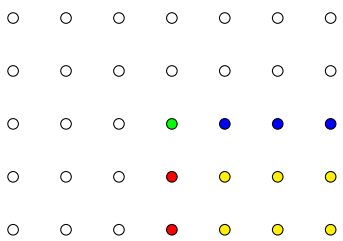
Obe funkciji `quickselect` sta *popolnoma* enaki kot pri osnovnem quickselectu, spremeni se le funkcija `izberiPivot`:

```
// Vrne indeks pivota v vektorju v, pri čemer se upoštevajo zgolj elementi z
// indeksi od lm do vključno dm. Tokrat pivot izberemo z algoritmom mediane median.
int izberiPivot(const vector<int>& v, int lm, int dm) {
    vector<int>::const_iterator it = v.begin() + lm;
    const vector<int>::const_iterator ite = v.begin() + dm + 1;
    vector<int> mediane;

    // Razdeli vektor v skupine po 5 elementov (zadnja skupina lahko ima manj
    // kot 5 elementov), izračunaj mediano vsake skupine in dobljene mediane
    // shrani v vektor mediane.
    while (it < ite) {
        vector<int> skupina;
        for (int j = 0; j < 5 && it < ite; j++) {
            skupina.push_back(*it++);
        }
        mediane.push_back(medianaKratkegaVektorja(skupina));
    }

    // Poišči mediano vektorja median.
    int medianaMedian = quickselect(mediane, (mediane.size() - 1) / 2);

    // Potrebujemo indeks mediane (tj. izbranega pivota), saj jo bomo morali
    // za potrebe funkcije porazdeli prestaviti na zadnje mesto vektorja.
    return find(v.begin() + lm, ite, medianaMedian) - v.begin();
}

// Vrne mediano podanega (kratkega) vektorja.
int medianaKratkegaVektorja(vector<int>& v) {
    sort(v.begin(), v.end());
    return v[(v.size() - 1) / 2];
}
```

3.2 Časovna zahtevnost

Rigorozen dokaz je v Cormenu, tukaj pa bomo ohlapnejši (beri: ignorirajmo umazane podrobnosti, pa tudi če pri tem ustrelimo kakšnega kozla). Naj me kdo, ki je matematike bolj vešč, popravi, če se mu bodo kje naježili lasje.

Elemente tabele si lahko predstavljamo takole:



Vsak stolpec predstavlja peterico zaporednih elementov. Recimo, da je vsak posamezen stolpec naraščajoče urejen, tako da je mediana ravno na sredini, in recimo, da so tudi

mediane posameznih stolpcev naraščajoče urejene. (V realnosti seveda ni nujno tako, a lahko mi verjamete, da lahko elemente brez škode premečemo tako, da takšno urejenost dosežemo.) Mediana median je potemtakem ravno zeleni element. Ta element (naš pivot) je *zanesljivo* manjši od vseh modrih (stolpci so naraščajoče urejeni po medianah), vseh rdečih (vsak stolpec je naraščajoče urejen) in seveda tudi vseh rumenih elementov. To pomeni, da je pivot gotovo manjši od polovice od treh petin elementov, torej od treh desetin elementov. V najbolj neugodnem primeru bomo torej pivot izbrali tako, da bomo v naslednjem koraku morali preiskati sedem desetin prvotne tabele.[4] Poraba časa na tabeli dolžine n je v najslabšem primeru torej

$$T(n) = T(n/5) + T(7n/10) + cn.$$

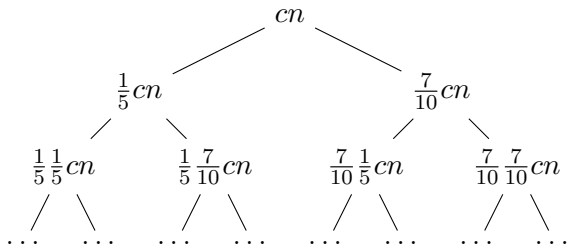
Od kod izvira prvi člen? Da poiščemo pivot, moramo z istim algoritmom (quickselect z mediano median) izračunati mediano tabele, sestavljene iz petine elementov celotne tabele.[5]

Dokažimo, da je $T(n) \leqslant 10cn$ in s tem $T(n) = \Theta(n)$. Predpostavimo, da lastnost velja za $1, 2, \ldots, n-1$, in preverimo, ali velja tudi za n. Res je:

$$\begin{aligned} T(n) &= T(n/5) + T(7n/10) + cn \\ &\leqslant (10cn)/5 + 7(10cn)/10 + cn \\ &= 10cn. \end{aligned}$$

3.3 Še bolj ohlapen dokaz (ali raje »dokaz«?)

Na podlagi formule $T(n) = T(n/5) + T(7n/10) + cn$ narišimo drevo:



Na prvem nivoju imamo cn dela, na drugem $(9/10)cn$, na tretjem $(9/10)^2cn$, na četrtem $(9/10)^3cn$ itd. Skupna količina dela je potemtakem kvečjemu

$$(1 + (9/10) + (9/10)^2 + \ldots)cn = \frac{1}{1 - 9/10}cn = 10cn = \Theta(n).$$

3.4 Zakaj ravno 5?

V resnici bi lahko računali tudi mediano sedmih, devetih, enajstih ... elementov. Pri mediani treh elementov pa bi dobili

$$T(n) = T(n/3) + T(2n/3) + cn,$$

to pa ni več $\Theta(n)$, ampak $\Theta(n \log n)$.

3.5 Komentar

Algoritem je teoretično sicer zanimiv, praktično pa ... nekoliko manj. Moja implementacija je pri naključnih tabelah približno 15-krat počasnejša od osnovnega quickselecta. Res pa je, da »izboljšani« quickselect preverjeno deluje v času $\Theta(n)$ tudi pri naraščajočih tabelah.

[4] Plus morda kakšna konstanta, ki pa jo tukaj nonšalantno zanemarimo — gl. prvi odstavek tega razdelka.
[5] Tudi v tem primeru smo (morda nedopustno) ohlapni, saj bi morali pisati $T(\lceil n/5 \rceil)$.