Fenwickovo drevo

Aleksander Kelenc

Opis problema

Neki spletni portal se je odločil, da bo za potrebe analize starosti svojih uporabnikov le-te uvrščal v 15 različnih starostnih kategorij glede na njihovo starost ob prvi prijavi v portal. V tabeli 1 so prikazane posamezne kategorije in ustrezno število uporabnikov, ki so v določenem trenutku uvrščeni v to kategorijo.

Omenjeni portal v različnih trenutkih zanima, koliko njihovih uporabnikov je uvrščenih med dvema izbranima kategorijama, npr. od kategorije 21–25 do kategorije 46–50 (vključno z mejama) je 13 uporabnikov. Seveda se z novimi uporabniki tabela kategorij nenehno spreminja. Ker spletni portal jemlje analizo podatkov zelo resno, želi takšna povpraševanja izvajati kolikor hitro je le mogoče. Dobili smo nalogo, da opišemo podatkovno strukturo, ki bo to omogočala.

Za lažje razmišljanje opišimo podan problem z računalniškim jezikom. Podan imamo končen seznam števil $a_1, a_2, \ldots, a_n$. Zanima nas vsota števil med indeksoma i in j (vključno z mejama), kjer velja $1 \leq i \leq j \leq n$. Če si seznam števil shranimo v polje a, potem bomo za eno takšno popraševanje morali pregledati vse elemente polja med indeksoma i in j. V splošnem za eno poizvedbo to pomeni linearno časovno zahtevnost glede na velikost seznama. Ali smo lahko bolj učinkoviti?

Hitro opazimo, da bi si pri tem lahko pomagali s kumulativnimi frekvencami. Naj bo k polje kumulativnih frekvenc za polje a. Za vsak indeks i, kjer je $1 \leq i \leq n$, je kumulativna frekvenca k_i enaka vsoti vseh elementov od a_1 do a_i, torej $k_i = a_1 + \ldots + a_i$. Za izračun polja k naj bo $k_1 = a_1$ in element $k_i = k_{i-1} + a_i$ za vsak $i \in \{2, \ldots, n\}$. Vidimo, da je vsota števil med indeksoma i in j sedaj enaka kar $k_j - k_{i-1}$. Če izračunamo polje kumulativnih frekvenc, potem dobimo konstantno časovno zahtevnost za eno poizvedbo vsote elementov med dvema indeksoma. Zelo učinkovito, ampak v portalu se število uporabnikov nenehno spreminja in zato moramo omogočiti še posodabljanje elementov v polju a. Ko posodobimo element a_i to pomeni, da moramo v polju kumulativnih frekvenc k posodobiti vse elemente od k_i do k_n. Vidimo, da v splošnem dobimo linearno časovno zahtevnost za posodobitev polja kumulativnih frekvenc. Z uporaba polja kumulativnih frekvenc smo dobili konstantno časovno zahtevnost za poizvedbo, vendar smo s posodabljanjem elementov spet prišli do linearne časovne zahtevnosti.

Drevesne podatkovne strukture se pri podobnih problemih izkažejo kot zelo učinkovite, saj velikokrat pohitrijo reševanje problema. Razvitih je precej različnih drevesnih podatkovnih struktur in ena izmed njih je tudi Fenwickovo drevo. Fenwickovo drevo je podatkovna struktura, ki z uporabo ideje o kumulativnih frekvencah omogoča poizvedbo za vsoto elementov med dvema indeksoma in posodabljanje elementov polja v logaritemski časovni zahtevnosti glede na velikost polja. Fenwickovo drevo je leta 1994 prvi predstavil Peter M. Fenwick.

Osnovna ideja

Osnovna ideja Fenwickovega drevesa je, da lahko vsako naravno število izrazimo kot vsoto ustreznih potenc števila 2. To pomeni, da bi lahko kumulativno frekvenco k_i predstavili kot vsoto ustreznih delnih kumulativnih frekvenc (predstavimo s poljem f) glede na razcep števila i na vsoto potenc števila 2. Če lahko indeks i predstavimo kot vsoto treh potenc števila 2, potem bo npr. k_i enak vsoti treh delnih kumulativnih frekvenc. Seveda morajo biti delne kumulativne frekvence podane za ustrezne intervale, da bomo s tem pristopom uspeli sestaviti celotno kumulativno frekvenco za indeks i. Za vsako potenco števila 2 iz razcepa indeksa i bo delna kumulativna frekvenca obsegala interval, ki bo širok toliko, kot,

starost	do 10	11–15	16–20	21–25	26–30	31–35	36–40	41–45
Ľtevilo	3	1	0	0	2	2	5	3

starost	46–50	51–55	56–60	61–65	66–70	71–75	nad 75	
Ľtevilo	1	5	4	0	0	2	3	

TABELA 1.
Tabela kategorij

je vrednost te potence. V zgornjem delu tabele 2 so prikazani intervali za delne kumulativne frekvence do števila 15. Oglejmo si osnovno idejo na konkretnem primeru za indeks 11. Število $11 = 2^3 + 2^1 + 2^0$, kar pomeni, da bo k_{11} sestavljen iz delnih kumulativnih frekvenc za intervale širine 8, 2 in 1 oziroma iz delnih kumulativnih frekvenc za intervale 1..8, 9..10 in 11.

V spodnjem delu tabele 2 lahko vidimo primer, kjer imamo v prvi vrstici zapisano polje a, v drugi vrstici polje za kumulativne frekvence k in v tretji vrstici polje za delne kumulativne frekvence f. Vrnimo se na primer indeksa 11. Kumulativna frekvenca za indeks 11 je sestavljena iz $k_{11} = f_{11} + f_{10} + f_8$, kar ustreza vsoti $= a_{11} + a_{9..10} + a_{1..8}$, kjer je $a_{1..8} = a_1 + \ldots + a_8$. Zaporedje števil $11, 10, 8$ dobimo, če v dvojiški predstavitvi števila $11 = 1011_{(2)}$ po vrsti nadomeščamo najbolj desno enico z ničlo. Izračun kumulativne frekvence k_i iz polja delnih kumulativnih frekvenc f torej dobimo tako, da seštejemo vse elemente polja f, ki jih dobimo, ko indeksu i po vrsti nadomeščamo najbolj desno enico z ničlo, dokler ne pridemo do števila 0.

Takšno indeksiranje nam iz polja delnih kumulativnih frekvenc generira drevo. Za koren drevesa postavimo indeks 0. Drevo ni uravnoteženo. Vsako vozlišče drevesa ima toliko otrok, kot je število ničel do prve enice v dvojiški predstavitvi indeksa, gledano iz desne strani. Globina vozlišča (razdalja do korena) pa je enaka številu enic v dvojiški predstavitvi indeksa. Polje, opremljeno s takšnim indeksiranjem imenujemo *Fenwickovo drevo*. Zaradi načina indeksiranja takšno strukturo imenujemo tudi *binarno indeksirano drevo*. Na sliki 1 je predstavljeno drevo za primer polja velikosti 15. V vozliščih drevesa so navedeni indeksi, v katerih se nahajajo ustrezni elementi v polju delnih kumulativnih frekvenc.

Operacije

S pomočjo opisane strukture želimo v logaritemskem času izvajati naslednji dve operaciji:

- poizvedbo za kumulativno frekvenco do nekega indeksa i,
- posodobitev polja delnih kumulativnih frekvenc f glede na posodobitev elementa a_i.

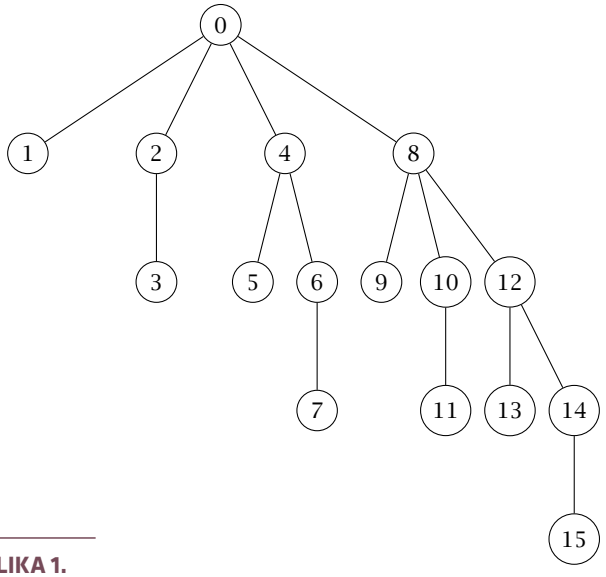
Pri obeh operacijah bomo indekse računali s pomočjo najbolj desne enice v dvojiški predstavitvi. Za izračun najbolj desne enice v dvojiški predstavitvi nekega števila si bomo pomagali z dvojiškim komplementom. Z dvojiškim komplementom so v računalniku predstavljena negativna števila. Iz pozitivnega števila m dobimo z dvojiškim komplementom $-m$ tako, da v dvojiški predstavitvi od m obrnemo vse bite in prištejemo ena. V tabeli 3 imamo primer za dvojiški komplement števila $18 = 00010010_{(2)}$, ki znaša 11101110. Opazimo, da je edino mesto, kjer se enica hkrati pojavi tako v dvojiški predstavitvi števila 18 kot tudi v dvojiški predstavitvi števila -18, ravno mesto najbolj desne enice v obeh dvojiških predstavitvah. S pomočjo logičnega operatorja $\wedge$ dobimo rezultat 00000010. Najbolj desno enico indeksa i dobimo z $(i \wedge -i)$, kar pomeni, da v drevesu za poizvedbe indeks starša vozlišča i izračunamo kot $i - (i \wedge -i)$.

Poizvedba za kumulativno frekvenco

Spodaj imamo prikazano funkcijo v programskem jeziku C++, ki vrne kumulativno frekvenco za indeks i. Za polje delnih kumulativnih frekvenc uporabimo `vector<int> f`. Število iteracij `while` zanke je enako številu enic indeksa i v bitni predstavitvi - v drevesu za poizvedbe to predstavlja dolžino poti od vozlišča i do korena. V najslabšem primeru to pomeni,

indeks	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
interval	1	1..2	3	1..4	5	5..6	7	1..8	9	9..10	11	9..12	13	13..14	15
polje a	3	1	0	0	2	2	5	3	1	5	4	0	0	2	3
polje k	3	4	4	4	6	8	13	16	17	22	26	26	26	28	31
polje f	3	4	0	4	2	4	5	16	1	6	4	10	0	2	3

TABELA 2.
Primer za polje velikosti 15



SLIKA 1.
Drevo za poizvedbe na polju velikosti 15

da je časovna zahtevnost funkcije enaka logaritmu velikosti polja a.

```
int vrniKumulativnoFrekv(int i) {
    int vsota = 0;
    while (i > 0) {
        vsota += f[i];
        i -= (int)(i & -i);
        // odštejemo najbolj desno enico
    }
    return vsota;
}
```

Ko imamo metodo za izračun kumulativne frekvence za poljubni indeks, potem lahko vsoto elementov med indeksoma i in j izračunamo kot `vrniKumulativnoFrekv(j)-vrniKumulativnoFrekv(i-1)`.

18	00010010
	11101101
+	00000001
-18	11101110

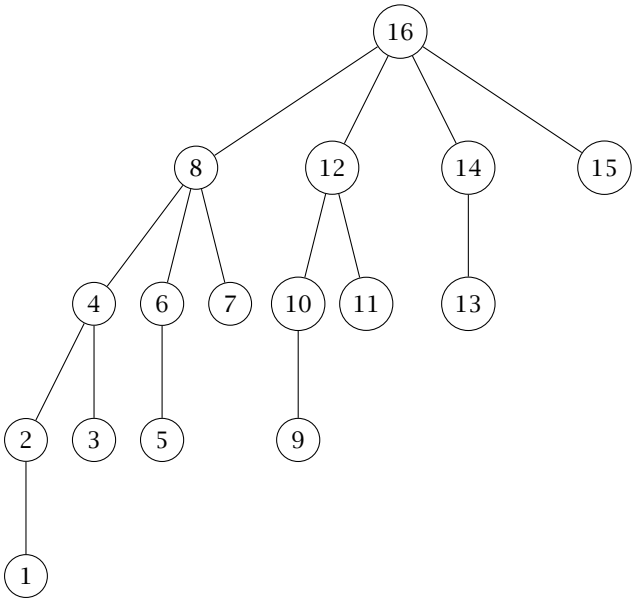
TABELA 3.
Dvojiški komplement števila 18

Posodabljanje elementa polja

Pri poizvedbi za kumulativno frekvenco smo indeksu vsakič enico na desni strani nadomestili z ničlo. S tem smo se premikali proti začetku polja delnih kumulativnih frekvenc oziroma proti korenu drevesa za poizvedbe. Za posodobitev elementa polja a pa moramo posodobiti vrednosti vsem delnim kumulativnim frekvencam, ki vsebujejo ta element. V tabeli 2 poglejmo, katere elemente polja delnih kumulativnih frekvenc f moramo posodobiti, če v polju a posodobimo elemnent a_3. Elementi polja f, katerih intervali vsebujejo element a_3, so f_3, f_4 in f_8. Iz indeksa 3 se moramo premakniti na indeks 4 (prištejemo 1) in potem na indeks 8 (prištejemo 4). Opazimo, da namesto odstranjevanja (odštevanja) enega bita na desni strani, moramo k trenutnemu indeksu dodati (prišteti) najbolj desni bit na vsakem koraku. Ta korak ponavljamo, dokler ne pridemo preko velikosti polja.

Z indeksiranjem za posodabljanje elementov dobimo drugačno drevo, kot smo ga dobili z indeksiranjem za poizvedbe. Drevo za polje velikosti 15 je prikazano na sliki 2.

Za koren drevesa dodamo vozlišče z indeksom 16, ki služi kot stražar, pri katerem se moramo zaustaviti pri posodabljanju. Opazimo, da je glede na



SLIKA 2.
Drevo za posodabljanje elementov polja velikosti 15

obliko drevo za posodabljanje zrcalna slika drevesa za poizvedbe.

Napišimo še funkcijo v programskem jeziku C++, ki posodobi polje delnih kumulativnih vsot, če k elementu polja a_i prištejemo vrednost v, kjer je vrednost v lahko tudi negativna.

```
void posodobi(int i, int v) {
    while (i < f.size()) {
        f[i] += v;
        i += (int) (i & -i);
        // prištejemo najbolj desno enico
    }
}
```

Časovno zahtevnost funkcije za posodabljanje lahko ocenimo s pomočjo časovne zahtevnosti funkcije poizvedbe. Število iteracij `while` zanke je enako dolžini poti od vozlišča i do korena v drevesu za posodabljanje. Ker je drevo za posodabljanje zrcalna slika drevesa za poizvedbe, sledi, da je najslabša časovna zahtevnost metode `void posodobi(int i, int v)` prav tako enaka logaritmu velikosti polja a.

S tem smo opisali podatkovno strukturo, ki omogoča, da za podano polje števil a izvajamo poizvedbe za kumulativne vsote in posodabljanje elementov v logaritemski časovni zahtevnosti glede na velikost polja a. V našem primeru starostnih kategorij spletnega portala je polje velikosti 15. V praksi sprememba časovne zahtevnosti iz linearne na logaritemsko pri polju velikosti 15 ne pride do izraza, vendar pa opisano podatkovno strukturo lahko uporabimo za poljubno velikost polja. S povečevanjem velikosti polja pa pridemo do bistvene razlike pri časovni učinkovitosti omenjenih operacij.

Literatura

[1] P. M. Fenwick, *A New Data Structure for Cumulative Frequency Tables*, Software-practice and experience **24** 3, (1994), 327–336.

[2] F. Halim in S. Halim, *Competitive Programming 3: The New Lower Bound of Programming Contests*, 2013.

[3] Wikipedia: Fenwick tree, ogled: 14. 1. 2017. `https://en.wikipedia.org/wiki/Fenwick_tree`.

Barvni sudoku

REŠITEV BARVNI SUDOKU

1	7	4	**6**	**5**	**3**	**8**	**2**
3	**5**	8	**2**	1	**6**	**7**	**4**
2	**3**	**6**	**8**	**7**	**1**	**4**	**5**
7	**4**	**5**	1	3	2	6	8
6	**2**	7	**4**	**8**	5	1	**3**
5	**8**	**1**	3	**4**	7	**2**	**6**
4	**6**	3	**7**	**2**	8	5	**1**
8	**1**	**2**	**5**	6	**4**	**3**	7