Prioriteetidega järjekord e eelistusjärjekord

Tuletame meelde:

> **Järjekord** *(queue)* – kõik järjekorras olijad on võrdsed, neid teenindatakse saabumise järgi.
>
> **Eelistusjärjekord** *(priority queue)* – mõni on järjekorras võrdsem vastavalt temaga seotud prioriteeti väljendavale suurusele ja peab saama teenindatud varem, kuid sama prioriteediga tegelaste vahel kehitivad ikka järjekorra reeglid.

Järjekorras võib olla määratud mingi arv prioriteediklasse, kuid on ka võimalus, et igaühel on nö individuaalne prioriteet

Eelistusjärjekorra loomisel määratakse parameeter objekti omaduste hulgast, mille alusel eelistusi teha või lisatakse spetsiaalne omadus prioriteedi väljendamiseks, nn **võti**. Igal juhul tuleb selle parameetri järgi järjekord üles ehitada. Prioriteeti määrab tavaliselt arvuline suurus ja vastavalt olukorrale võib olla eelistatuim kõige suurema või kõige väiksema võtmeväärtudega objekt.

Põhilised **tegevused** eelitsusjärjekorras:

`Insert(Q,x)` – lisatakse element võtmega x

`Find-Minimum(Q)` või `Find-Maximum(Q)` – tagasta vähima või suurima võtmega elemendi väärtus järjekorrast.

`Delete-Minimum(Q)` või `Delete-Maximum(Q)` – võta järjekorrast vähima või suurima võtmega element.

Üldisemal juhl lisanduvad veel järgmised tegevused:

`DecreaseKey(Q,x)` – elemendi x võtmeväärtust vähendatakse ja seoses sellega tuleb talle leida järjekorras uus kõrgem koht. Ette antakse tavaliselt elemendi koht (aadress või indeks), sest kuhi ei võimalda edukat otsimist.

`Union(Q, Q1)` – luuakse uus eelistusjärjekord kahe järjekorra ühendamisele.

Prioriteetidega järjekorra **realiseerimiseks** on mitmeid erinevaid võimalusi ja valik sõltub **järjekorra iseloomust**.

1. Kui peale ühekordset elementide lisamist ja saba ehitamist on vaja neid ainult võtta ja töödelda, siis on lihtsaimaks kasutatavaks struktuuriks sorteeritud massiiv või dünaamiline nimistu (seda võib kutsuda ka järjekorraks, kuid ei pruugi).
2. Kui on vaja lisada, kustutada ja päringuid teha, on juba tarvis tõelist eelistusjärjekorda. Vastus tuleb anda järgmistele küsimustele:
   a) Mida on vaja veel teha lisaks vähima (suurima) võtmega elemendi eemaldamisele? Kas on veel lisaks vaja otsida ja/või kustutada teatud elemente?
   b) Kas järjekorda sattuvate elementide arv on eelnevalt enam-vähem teada või on see täiesti lahtine (kas kasutada staatilist või dünaamilist realisatsiooni)?
   c) Kas elementide prioriteedid võivad muutuda järjekorras olemise ajal või säilib neil see, millega nad sappa pandi (kas elemente on sabas olemise ajal vaja ringi korraldada)?

Realisatsioon

Võimalusi eelistusjärjekorra **realiseerimiseks** on kümneid. Mõned näited

Sorteeritud massiiv või dünaamiline nimistu
Väikseima elemendi leidmine ja eelmadamine on lihtne ja kiire (O(1)), probleemid tekivad, kui on vaja töö

käigus (palju) elemente lisada ja veel hullem on proriteetide muutmine.

Kahendkuhi
Nii lisamine kui eemaldamine toimub ajaga O(log n). (vt kahendkuhja kirjeldust sorteerimise materjalis). Elemendi eemaldamine toimub kahendpuu juurest ja elemendi lisamine vaba lehe kohale, peale mida see element sobivasse paika liigutatakse.
Järjekorrast **võetakse** alati **tipp** (massiivi 1. element), tema kohale tõstetakse kuhja viimane element ja rakendatakse talle protseduuri Heapify, et teda õigele kohale viia. Kuhja suurust vähendatakse 1 võrra.(vt kuhjaga sorteerimine).
Elemendi lisamiseks suurendatakse kõigepealt kuhja suurust ja uus element pannakse selleks elemendiks. Seejärel paigutatakse element oma õigele kohale (seda sorteerimise all ei ole):

```
Heap-Insert(A,key)
heap-size[A]<-heap-size[A]+1
i<-heap-size[A]
while i>1 and A[Parent(i)]<key do
   A[i]<-A[Parent(i)]
   i<-Parent(i)
endwhile
A[i]<-key
```

Kuna kahendkuhi realiseeritakse massiiviga, on see variant kasutatav siis, kui sabasse pandavate elementide arv on enam-vähem teada. Kahendkuhja kui struktuuri kutsutigi algselt prioriteetidega järjekorraks.

Piiratud kõrgusega *(bounded-height)* **eelistusjärjekord**
See struktuur kujutab endast massiivi külge ühendatud dünaamilisi nimistuid. Sobib kasutamiseks siis, kui prioriteediklasside arv on teada – st igale klassile seatakse vastavusse dünaamiliselt hallatav järjekord, kust eemaldamine ja kuhu lisamine saavad toimuda konstantse ajaga O(1). Kogu järjekorraga seotud viita hoitakse kõige kõrgema prioriteediga saba esimese elemendi peal, kus toimub eemaldamine. Lisatakse element aga vastavalt tema prioriteedile – vastava nimistu lõppu. Väikese arvu võtmete diskreetsete väärtuste korral on see parim lahendus.
Lisaks nimistu elemendi **eemaldamisele** tuleb järjekorra viit viia sama klassis oleva järgmise elemendi peale või kui see sabake tühjaks sai, siis järgmise elemente sisaldava sabakese esimesele elemendile.
Lisamisel toimub kogu järjekorra viida ümberpaigutamine juhul, kui lisatud element tuli juure viida asukohast kõrgemasse prioriteediklassi.

Otsimiskahendpuu
Täpsemalt selle puu kohta vt otsimiskahendpuu materjalist. sellest struktuurist on suhteliselt kerge kätte saada nii minimaalset kui ka maksimaalset elementi (aeg O(log n)). Ka lisamine vajab sama aega. Prioriteetide muutumise korral pole struktuur enam nii mugav. Seega on teda hea kasutada siis, kui kahendkuhi ei sobi, sest elementide arv pole teada.

Fibonacci kuhi
Fibonacci kuhja kirjeldasid 1984.a. M. Fredman ja R. Tarjan. Võrreldes kahendkuhjaga on F-kuhi kiire elemendi lisamisel ja võtme vähendamisel. F-kuhi osutub sobivaks siis, kui sabas olevate elementide prioriteedid võivad muutuda väiksemaks (graafis lühima tee leidmise juures – Dijktra algoritm, sealjuures prioriteedid võivad teoreetiliselt muutuda igal sammul mitme tipu jaoks, tippe on aga vaja järjekorrast eemaldada igaühte ainult üks kord).
Halvem pool F-kuhja juures on see, et ta kasutab rohkem mälu.
Kasutatakse kahe viidaga nimistuid, milledest ehitatakse puu. Iga tipuga seotakse seega viidad vasakule ja paremale ning lisaks eellasele ja järglasele. Veel seotakse iga tipuga väljad `Mark`, `Degree` ja `Negative_Infinity`, mis kõik on ühel või teisel viisil realisatsiooniks vajalikud. Realisatsiooni täpsemad detailid koos algoritmidega jäävad huvilistele iseseisvaks uurimiseks (vt näiteks artiklit *http://algorithm.myrice.com/resources/technical_artile/fibonacci_heap/fibonacci.htm – mis hetkel ei tundu töötavat*) või siis wikipeedia artiklit:
http://en.wikipedia.org/wiki/Fibonacci_heap

F-kuhi koosneb puudest, kus ühe tipu järglaste arv ei ole piiratud. Tipu järglased on omavahel ühendatud kahe

viidaga ringnimistusse. Igast tipust läheb viit vanemale ja lisaks on viidaväli, et sinna külge ühendada üks selle tipu lastest. Väli `Degree` ütleb, mitu last on antud tipul (milline on tipu järk). Ühe tipu järglastel peab igaühel olema erinev järk.
Ka kuhja kõige kõrgemal tasemel on kahe viidaga ringnimistu ja kogu kuhja viit näitab kõige väiksemale elemendile selles nimistus (ehk sellele elemendile, mis kõrgeimat prioriteeti omab).

F-kuhja reeglid
- Iga alampuu jaoks on puu juure võti väiksem tema kõigi järglaste võtmetest.
- Ühe tipu järglastel peab olema kõigil erinev järk.

Mis toimub erinevate operatsioonide puhul:

Lisamine – uus element lisatakse praeguse vähima elemendi kõrvale ja kui vaja, siis saab ta uueks väikseimaks elemendiks.

Ühendamine – kaks juure taseme nimistut ühendatakse kokku (mis dünaamilise nimistu puhul on kiire), seda operatsiooni kasutavad teised.

Väikseima eemaldamine – ajakulukas operatsioon. Tegelikult ei võta eemaldamine ise eriti aega, kuid eemaldamisega seotakse ka F-kuhja korrastamine:

- eemaldatud elemendi lapsed ühendatakse juurnimistusse
- vaadatakse läbi kõigi tippude järgud juurnimistus ja omavahel ühendatakse need, mille järgud on võrdsed; ühendamine toimub selliselt, et suurema väärtusega tipp pannakse väiksema väärtusega tipule lapseks ning viimase järk suureneb ühe võrra. Kui nüüd jälle on olemas sama järku tipp, toimub jälle ühendamine.
- peale tippude ühendamist leitakse uus vähima väärtusega tipp.

Võtme vähendamine – elemendi võti muudetakse ja võrreldakse teda vanema võtmega; kui uus võti on väiksem, peab see tipp kuhjas ülespoole liikuma. Ta eemaldatakse oma kohast ja viiakse juurnimistusse (kus tuleb teda ka väikseima tipuga võrrelda). Kuna selline keskelt võtmine ajab kuhja segamini, näeb algoritm ette, et kui tipul on juba teine järglane ära võetud, tõstetakse ta ise juurnimistusse. Laste kustutamise üle järje pidamiseks kasutatakse `Mark`-välja, mille abil kajastatakse olukorda, kas üks laps on juba läinud või ei ole.