Luku 8

Aluekyselyt

Aluekysely on tiettyä taulukon väliä koskeva kysely. Tyypillisiä aluekyselyitä ovat, mikä on taulukon välin lukujen summa tai pienin luku välillä. Esimerkiksi seuraavassa taulukossa tummennetun välin lukujen summa on $3+1+7+2=13$ ja pienin luku on 1.

5	2	8	3	3	1	7	2	5	8

Aluekyselyyn vastaaminen on helppoa ajassa $O(n)$ käymällä läpi kaikki taulukon arvot annetulla välillä. Tämä on kuitenkin hidasta, jos aluekyselyitä täytyy tehdä usein. Tässä luvussa tavoitteena onkin kehittää tehokkaampia tietorakenteita, jotka soveltuvat erityisesti aluekyselyiden tekemiseen.

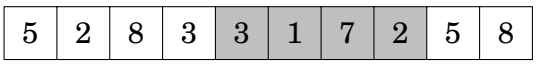
8.1 Summataulukko

Summataulukko mahdollistaa minkä tahansa taulukon välin summan laskemisen ajassa $O(1)$. Summataulukon jokaisessa kohdassa on taulukon lukujen summa kyseiseen kohtaan asti. Seuraavassa on esimerkki taulukosta ja sitä vastaavasta summataulukosta:

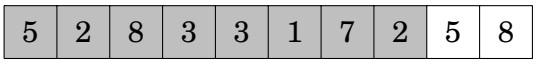
5	2	8	3	3	1	7	2	5	8

5	7	15	18	21	22	29	31	36	44

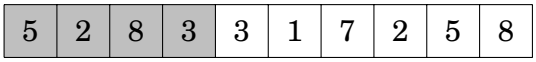
Summataulukon hyötynä on, että minkä tahansa taulukon välin summan saa laskettua kahdesta summataulukon arvosta. Esimerkiksi summa



saadaan vähentämällä summasta



summa

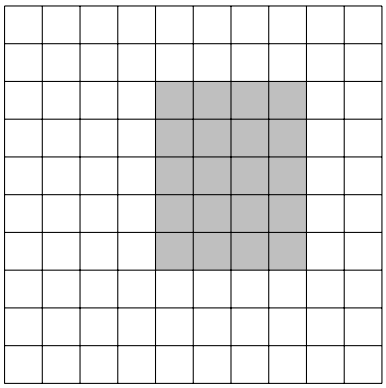


Summataulukon perusteella summa on $31 - 18 = 13$.

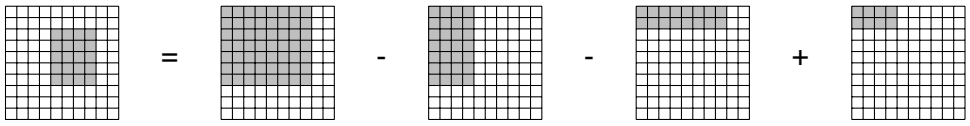
Yleisesti jos T on alkuperäinen taulukko ja S on summataulukko, niin kaava $S[b] - S[a-1]$ kertoo T:n summan välillä $a \ldots b$. Erikoistapauksena jos $a = 0$, niin summa on suoraan $S[b]$. Summataulukon muodostaminen vie aikaa $O(n)$, ja tämän jälkeen minkä tahansa välin summan pystyy laskemaan ajassa $O(1)$.

Myös kaksiulotteisesta taulukosta voi tehdä summataulukon, jolloin minkä tahansa suorakulmaisen alueen summan voi laskea ajassa $O(1)$. Nyt jokaisessa summataulukon kohdassa on sellaisen suorakulmion summa, joka alkaa taulukon vasemmasta ylänurkasta ja päättyy kyseiseen kohtaan.

Tarkastellaan esimerkiksi seuraavan alueen summan laskemista:



Ideana on ottaa lähtökohdaksi summa taulukon vasemmasta ylänurkasta halutun alueen oikeaan alanurkkaan, poistaa siitä vasemman reunan ja yläreunan summat sekä lisätä vasemman ylänurkan summa, joka tuli poistettua kahdesti. Tässä tapauksessa laskutoimitus on seuraava:



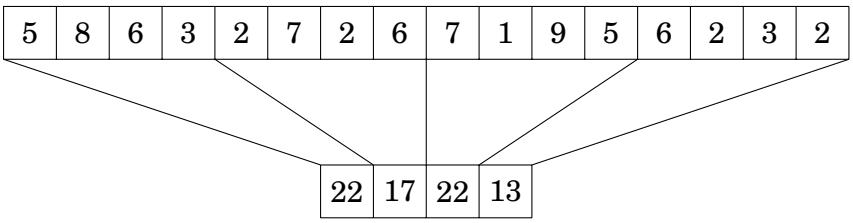
Yleisesti suorakulmion summa kohdasta $T[y_1][x_1]$ kohtaan $T[y_2][x_2]$ selviää kaavalla $S[y_2][x_2] - S[y_1-1][x_2] - S[y_2][x_1-1] + S[y_1-1][x_1-1]$. Jälleen jos $x_1 = 0$ tai $y_1 = 0$, niin vastaavat summat jäävät pois kaavasta. Nyt summataulukon muodostus vie aikaa $O(n^2)$ ja summan laskeminen $O(1)$. Samaa ideaa voi käyttää myös korkeammissa ulottuvuuksissa.

Summataulukko on tehokas ja helposti toteutettava ratkaisu, mutta siinä on kaksi heikkoutta. Ensinnäkin tietorakenne soveltuu vain summien laskemiseen eikä esimerkiksi pienimmän luvun etsimiseen. Lisäksi jos taulukon sisältö muuttuu, niin summataulukko täytyy laskea uudestaan. Seuraavaksi esiteltävät tietorakenteet korjaavat nämä molemmat ongelmat.

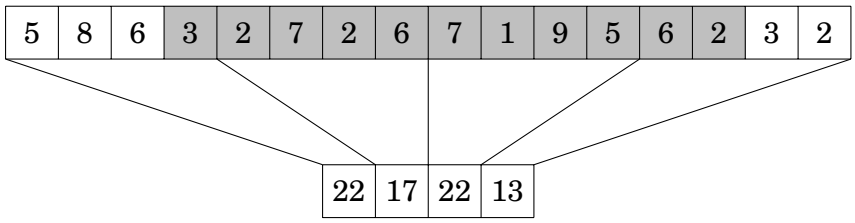
8.2 $\sqrt{n}$-rakenne

Seuraava tietorakenne on tavallisen taulukon ja summataulukon välimuoto, jossa taulukko jaetaan $\sqrt{n}$-kokoisiin väleihin. Taulukon lisäksi jokaisesta välistä tallennetaan aputaulukkoon lukujen summa, pienin luku tai muu haluttu tieto. Tämän ansiosta sekä aluekyselyn tekeminen että taulukon muuttaminen onnistuvat ajassa $O(\sqrt{n})$.

Tarkastellaan summien laskemista seuraavassa taulukossa, jossa $n = 16$. Nyt välin pituus on $\sqrt{n} = 4$, joten aputaulukkoon tulee 4 summaa.



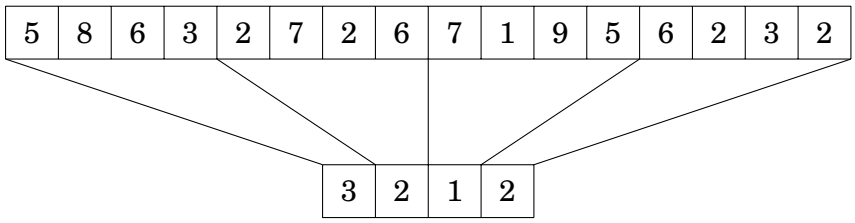
Lasketaan sitten seuraavan alueen summa:



Kaksi keskimmäistä $\sqrt{n}$-väliä ovat kokonaan mukana alueessa, minkä ansiosta niiden summat saa aputaulukosta. Alussa ja lopussa olevat luvut lasketaan mukaan summaan erikseen. Tämän perusteella alueen summaksi tulee $3+17+22+6+2=50$. Minkä tahansa alueen summan saa laskettua ajassa $O(\sqrt{n})$, koska alussa ja lopussa on molemmissa alle $\sqrt{n}$ lukua, joiden summa täytyy laskea käyttämättä aputaulukkoa.

Jos taulukko muuttuu, niin vastaava kohta aputaulukossa täytyy laskea uudestaan. Tähän kuluu aikaa $O(\sqrt{n})$, koska muutettavalla välillä on $\sqrt{n}$ lukua. Summan laskennassa muutos on mahdollista toteuttaa myös ajassa $O(1)$, koska riittää vähentää vanha arvo aputaulukosta ja lisätä uusi.

$\sqrt{n}$-rakenne soveltuu summan lisäksi minkä tahansa *liitännäisen* funktion aluekyselyihin. Liitännäisyys tarkoittaa, että peräkkäiset operaatiot voi laskea missä tahansa järjestyksessä. Esimerkiksi summa on liitännäinen, koska $a+(b+c)=(a+b)+c$. Vastaavasti pienin luku on liitännäinen, koska $\min(a,\min(b,c))=\min(\min(a,b),c)$. Pienimmän luvun kyselyissä tietorakenne näyttäisi seuraavalta:

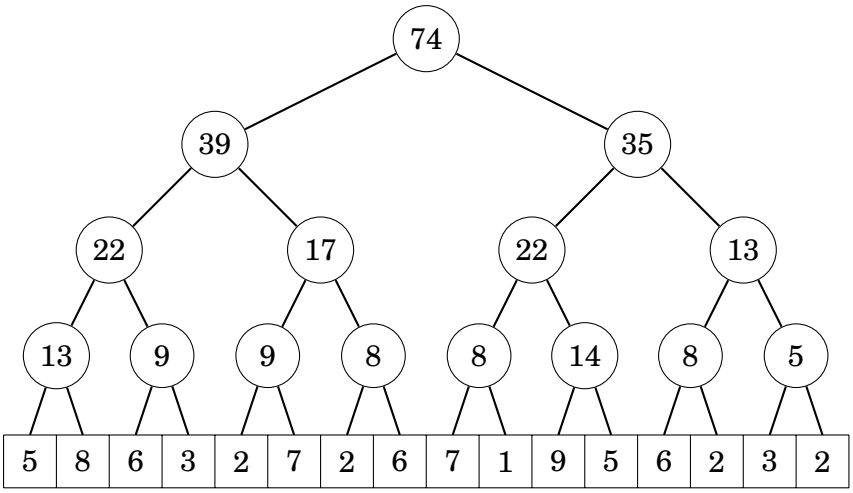


$\sqrt{n}$-rakenne on lähes yhtä helppo toteuttaa kuin summataulukko, ja se mahdollistaa taulukon muuttamisen tehokkaasti sekä muidenkin kuin summien laskemisen. Kuitenkin joskus myös $O(\sqrt{n})$ on liian hidas aikavaativuus. Seuraavaksi esiteltävä puurakenne mahdollistaa kaiken saman kuin $\sqrt{n}$-rakenne, mutta aikavaativuus on vain $O(\log n)$.

8.3 Segmenttipuu

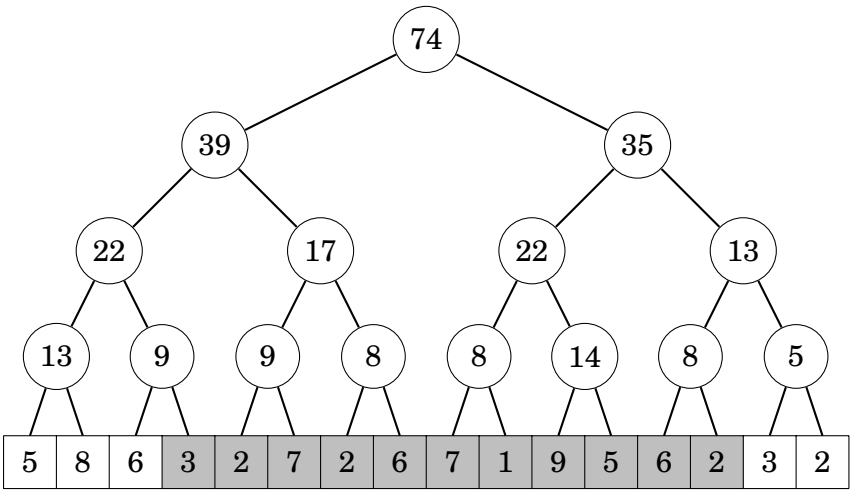
Segmenttipuu on puurakenne, jonka juuressa on aluekyselyn tulos koko taulukon alueelta, seuraavalla tasolla tulokset vasemmasta ja oikeasta puoliskosta, seuraavalla tasolla tulokset kustakin neljänneksestä jne. Sekä kyselyn että taulukon muuttamisen aikavaativuus on $O(\log n)$.

Seuraava segmenttipuu sisältää taulukon välien summat:

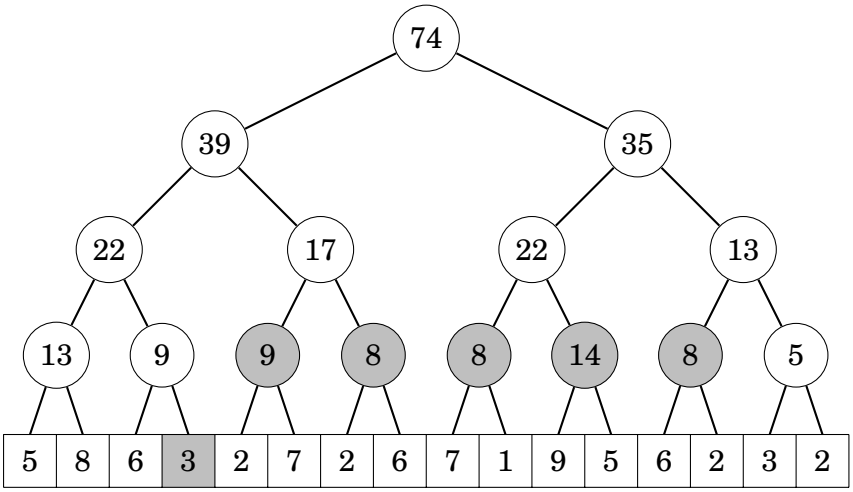


Kysely

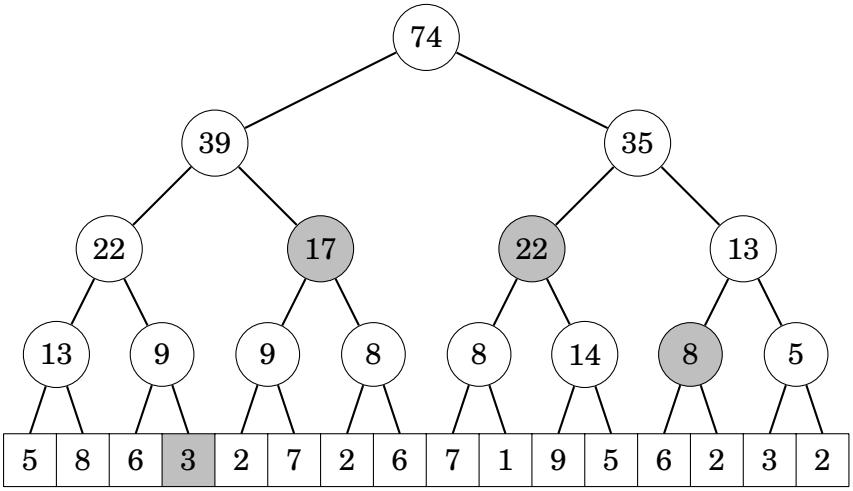
Tarkastellaan seuraavan summan laskemista segmenttipuusta:



Ideana on muodostaa välin summa mahdollisimman ylhäällä puussa olevista summista. Alkutilanteessa koko summan lukuun ottamatta ensimmäistä lukua 3 saa laskettua ylemmältä tasolta:



Tällä hetkellä summa on jaettu väleihin niin, että ensimmäisessä välissä on yksi luku ja viidessä seuraavassa välissä on kaksi lukua. Kahden luvun välejä voi vielä yhdistää neljän luvun väleiksi:

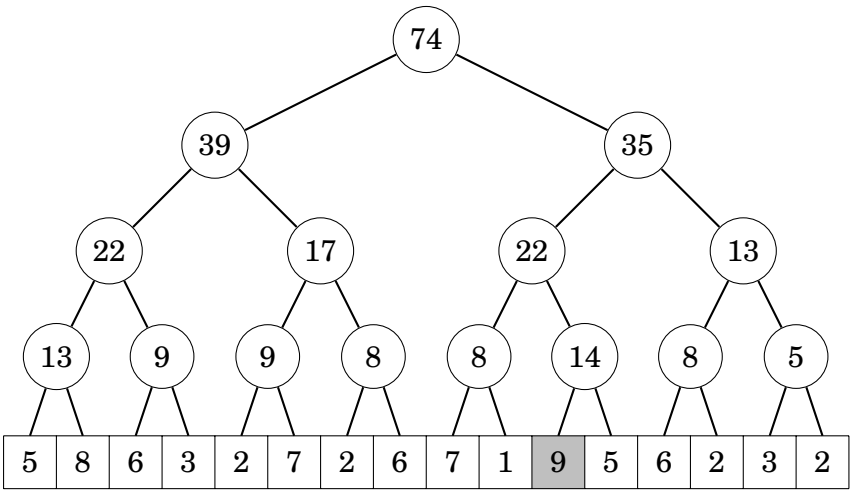


Tämän perusteella välin summa on $3 + 17 + 22 + 8 = 50$.

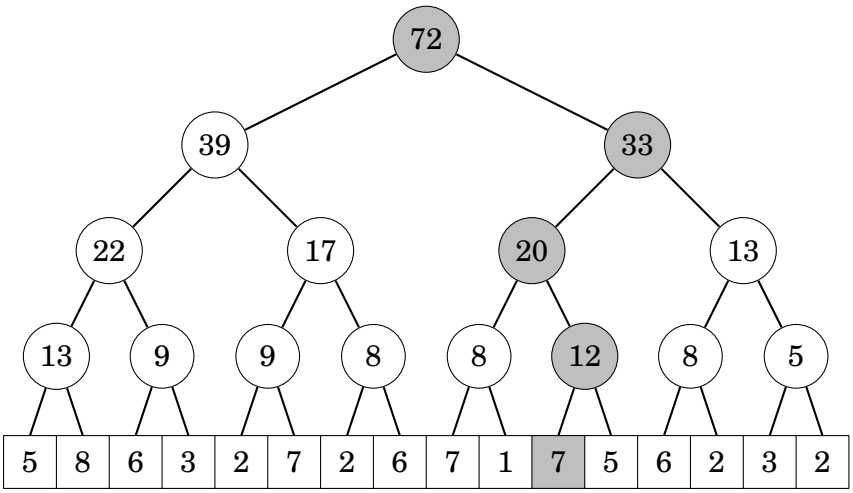
Jokaisessa kyselyssä summattavien osien määrä on luokkaa $O(\log n)$, minkä ansiosta kysely on mahdollista toteuttaa ajassa $O(\log n)$.

Päivitys

Segmenttipuun päivityksessä on ideana kulkea puuta ylöspäin juureen asti ja päivittää uusi tulos jokaiseen solmuun. Tarkastellaan esimerkiksi taulukon arvon 9 muuttamista:



Seuraavassa tilanteessa uudeksi arvoksi tulee 7. Muutos vaikuttaa seuraaviin segmenttipuun solmuihin:



Polun pituus taulukosta puun juureen on $O(\log n)$, minkä ansiosta muuttaminen onnistuu ajassa $O(\log n)$.

Toteutus

Segmenttipuun lyhyt ja tehokas toteutus perustuu taulukkoon, johon tallennetaan sekä puun solmujen sisältö että alkuperäisen taulukon sisältö. Ideana on tallentaa ensin puun solmut taso kerrallaan ja lopuksi taulukon sisältö. Puun solmujen määrä on $n-1$, kun taulukossa on n arvoa.

Seuraava funktio `alusta` muodostaa segmenttipuun taulukkoon `p` taulukon `t` sisällön perusteella. Taulukossa `p` indeksit $1 \ldots n-1$ sisältävät puun solmujen arvot ja indeksit $n \ldots 2n-1$ sisältävät taulukon `t` arvot.

```
const int n = 16;
int t[] = {5,8,6,3,2,7,2,6,7,1,9,5,6,2,3,2};
int p[2*n];

void alusta() {
    for (int i = 0; i < n; i++)
        p[n+i] = t[i];
    for (int i = n-1; i >= 1; i--)
        p[i] = p[2*i]+p[2*i+1];
}
```

Funktio `summa` laskee taulukon lukujen summan indeksistä a indeksiin b. Aluksi funktio lisää indekseihin arvon n, koska haku alkaa segmenttipuun taulukko-osasta. Tämän jälkeen funktio siirtyy joka askeleella tason ylemässä puussa jakamalla indeksit kahdella. Summa s kasvaa aina, kun hakualueen reunalla olevalle summalle ei ole paria.

```
int summa(int a, int b) {
    a += n; b += n;
    int s = 0;
    while (a < b) {
        if (a%2 == 1) s += p[a++];
        if (b%2 == 0) s += p[b--];
        a /= 2; b /= 2;
    }
    if (a == b) s += p[a];
    return s;
}
```

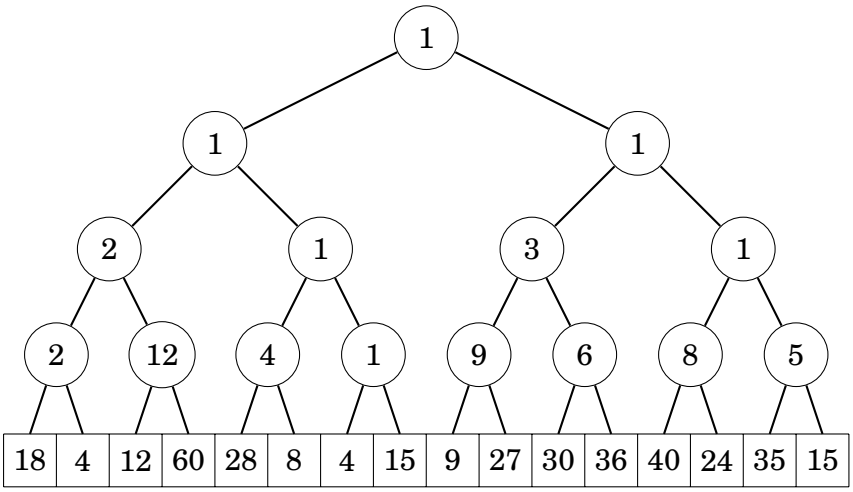
Seuraava funktio muuttaa taulukon indeksin k arvoksi x. Funktio kulkee puun ylös juureen saakka ja päivittää kaikkiin kohtiin uuden summan.

```
void paivita(int k, int x) {
    p[n+k] = x;
    k = (n+k)/2;
    while (k >= 1) {
        p[k] = p[2*k]+p[2*k+1];
        k /= 2;
    }
}
```

Huomaa, että tässä esitetty toteutus olettaa lukujen määrän olevan muotoa 2^k. Jos lukujen määrä on jokin muu, helppo ratkaisu on lisätä loppuun nollia tarvittava määrä.

Sovellukset

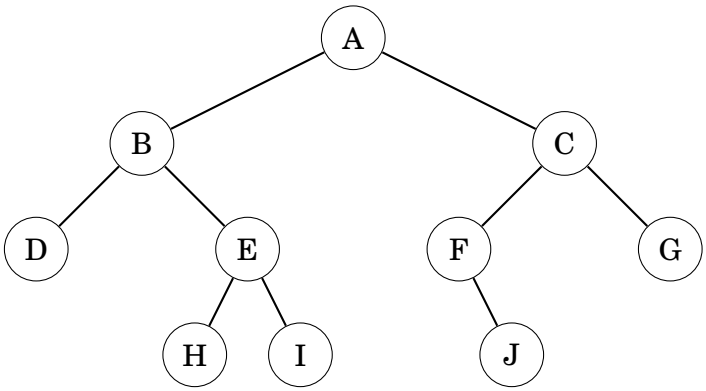
$\sqrt{n}$-rakennetta vastaavasti segmenttipuun avulla voi toteuttaa minkä tahansa liitännäisen laskutoimituksen aluekyselyn. Seuraava erikoinen segmenttipuu sisältää jokaisen välin lukujen suurimman yhteisen tekijän.



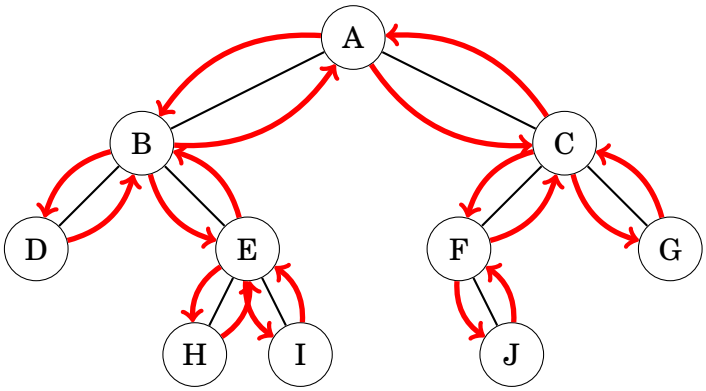
Segmenttipuuta muistuttava puurakenne on *binääri-indeksipuu*, jonka operaatiot ovat myös $O(\log n)$-aikaisia. Se on hieman segmenttipuuta kevyempi ja vie vähemmän muistia, mutta rakenne mahdollistaa vain summien käsittelyn ja sen toiminnan ymmärtäminen on vaikeampaa.

8.4 Alin yhteinen vanhempi

Puussa kahden solmun *alin yhteinen vanhempi* on sellainen solmu, josta pääsee molempiin solmuihin ja joka on mahdollisimman alhaalla puussa. Esimerkiksi seuraavassa puussa D:n ja H:n alin yhteinen vanhempi on B ja B:n ja J:n alin yhteinen vanhempi on A.



Katsotaan seuraavaksi, miten ongelman voi ratkaista tehokkaasti pienimmän luvun aluekyselyn avulla. Ensimmäinen vaihe on käydä puun solmut läpi syvyyssuuntaisesti:



Tästä muodostuu seuraava taulukko, joka sisältää solmut ja niiden syvyydet puussa läpikäynnin järjestyksessä:

A	B	D	B	E	H	E	I	E	B	A	C	F	J	F	C	G	C	A
0	1	2	1	2	3	2	3	2	1	0	1	2	3	2	1	2	1	0

Ideana on, että kahden solmun alimman yhteisen vanhemman löytää etsimällä niiden välisellä alueella olevan solmun, jonka syvyys on pienin. Esimerkiksi D:n ja H:n alin yhteinen vanhempi on B, jonka syvyys on 1:

A	B	D	B	E	H	E	I	E	B	A	C	F	J	F	C	G	C	A
0	1	2	1	2	3	2	3	2	1	0	1	2	3	2	1	2	1	0

Vastaavasti B:n ja J:n alin yhteinen vanhempi on A, jonka syvyys on 0:

A	B	D	B	E	H	E	I	E	B	A	C	F	J	F	C	G	C	A
0	1	2	1	2	3	2	3	2	1	0	1	2	3	2	1	2	1	0

Ratkaisu perustuu siihen, että solmujen välinen alue taulukossa kuvaa polkua niiden välillä. Polussa ylimpänä oleva solmu on ainoa polun solmu, josta pääsee laskeutumaan kumpaankin solmuun. Jos solmu esiintyy taulukossa monta kertaa eli se ei ole lehtisolmu, niin mikä tahansa solmun esiintymä kelpaa alueen päätekohdaksi.

Segmenttipuuta käyttäen taulukon välin pienin luku löytyy ajassa $O(\log n)$, joten minkä tahansa kahden solmun alin yhteinen vanhempi löytyy ajassa $O(\log n)$. Jokainen kaari tulee vastaan kahdesti käpikäynnissä, joten taulukko vie tilaa $O(n)$.