Gagnagrindur

STL, biltré, sammengisleit

Bergur Snorrason

9. febrúar 2020

Efnisyfirlit

STL

- Í flestum forritunarmálum eru ýmsar gagnagrindur útfærðar.

STL

- Í flestum forritunarmálum eru ýmsar gagnagrindur útfærðar.
- Við köllum þessar gagnagrindur vanalega `STL`.

STL

- Í flestum forritunarmálum eru ýmsar gagnagrindur útfærðar.
- Við köllum þessar gagnagrindur vanalega `STL`.
- Skoðum helstu `STL` gagnagrindur í `C++`.

`vector`

- Gagnagrindin `vector` virkar svipað og fylki.
- Hún leyfir $\mathcal{O}(1)$ uppflettingu á i-ta stakinu.
- Hún leyfir $\mathcal{O}(1)$ breytingar á i-ta stakinu.
- Hún leyfir að skeyta staki aftast í $\mathcal{O}(1)$.
- Ef við getum gert eitthvað með fylki, þá getum við það líka með `vector`.

Listi (`list`)

- Gagnagrindin `list` er einteingdur listi.
- Við getum fundið k-ta stakið í lista í $\mathcal{O}(k)$.
- Við getum bætt staki fyrir aftan gefið stak í $\mathcal{O}(1)$ (en þurfum fyrst að finna stakið).
- Við getum skeytt saman listum í $\mathcal{O}(1)$.

Hlaði (`stack`)

- Gagnagrindin er mjög einföld.
- Við getum bætt við staki í $\mathcal{O}(1)$.
- Við getum fjarlægt/lesið stakið sem hefur verið styst í hlaðanum í $\mathcal{O}(1)$.

Biðröð (queue)

- Gagnagrindin er mjög einföld.
- Við getum bætt við staki í $\mathcal{O}(1)$.
- Við getum fjarlægt/lesið stakið sem hefur verið lengst í biðröðinni í $\mathcal{O}(1)$.

dequeue

- Nafnið er stytting á „double-ended queue".
- Við getum skeytt staki fremst eða aftast í $\mathcal{O}(1)$.
- Við getum fjarlægt/lesið stakið fremst eða aftast í $\mathcal{O}(1)$.

Forgangsbiðröð (`prioirit_queue`)

- Gagnagrindin geymir samanberanleg stök.
- Gerum ráð fyrir sambanburður stakan sé taki k tíma og það séu n stök í biðröðinni.
- Við getum bætt við stökum í $\mathcal{O}(k \log n)$.
- Við getum fjarlægt stakið með hæstan forgang í $\mathcal{O}(k \log n)$.
- Við getum lesið stakið með hæstan forgang í $\mathcal{O}(1)$.

Mengi (set)

- Gagnagrindin geymir samanberanleg stök.
- Gerum ráð fyrir sambanburður stakan taki k tíma og það séu n stök í menginu.
- Við getum bætt við staki í mengið í $\mathcal{O}(k \log n)$.
- Við getum fjarlægt stak úr menginu í $\mathcal{O}(k \log n)$.
- Við getum athugað hvort stak sé í menginu í $\mathcal{O}(k \log n)$.
- Einnig er til `multiset`, sem leyfir endurtekningar.

Efnisyfirlit

Hrúgur

- Rótartvíundatré sem uppfyllir að sérhver nóða er stærri en börnin sín er sagt uppfylla *hrúguskilyrðið*.

Hrúgur

- Rótartvíundatré sem uppfyllir að sérhver nóða er stærri en börnin sín er sagt uppfylla *hrúguskilyrðið*.
- Við köllum slík tré *hrúgur* (e. heap).

Hrúgur

- Rótartvíundatré sem uppfyllir að sérhver nóða er stærri en börnin sín er sagt uppfylla *hrúguskilyrðið*.
- Við köllum slík tré *hrúgur* (e. heap).
- Hrúgur eru heppilega auðveldar í útfærslu.

Hrúgur

- Rótartvíundatré sem uppfyllir að sérhver nóða er stærri en börnin sín er sagt uppfylla *hrúguskilyrðið*.
- Við köllum slík tré *hrúgur* (e. heap).
- Hrúgur eru heppilega auðveldar í útfærslu.
- Við geymum tréð sem fylki og eina erfiðið er að viðhalda hrúguskilyrðinu.

Fylki sem tré

- Þegar við geymum tréð sem fylki notum við eina af tveimur aðferðum.

Fylki sem tré

- Þegar við geymum tréð sem fylki notum við eina af tveimur aðferðum.
- Sú fyrri:

Fylki sem tré

- Þegar við geymum tréð sem fylki notum við eina af tveimur aðferðum.
- Sú fyrri:
  - Rótin er í staki 1 í fylkinu.

Fylki sem tré

- Þegar við geymum tréð sem fylki notum við eina af tveimur aðferðum.
- Sú fyrri:
  - Rótin er í staki 1 í fylkinu.
  - Vinstra barn staksins i er stak $2 \times i$.

Fylki sem tré

- Þegar við geymum tréð sem fylki notum við eina af tveimur aðferðum.
- Sú fyrri:
    - Rótin er í staki 1 í fylkinu.
    - Vinstra barn staksins i er stak $2 \times i$.
    - Hægra barn staksins i er stak $2 \times i + 1$.

Fylki sem tré

- Þegar við geymum tréð sem fylki notum við eina af tveimur aðferðum.
- Sú fyrri:
  - Rótin er í staki 1 í fylkinu.
  - Vinstra barn staksins i er stak $2 \times i$.
  - Hægra barn staksins i er stak $2 \times i + 1$.
  - Foreldri staks i er stakið $\left\lfloor \frac{i}{2} \right\rfloor$.

Fylki sem tré

- Þegar við geymum tréð sem fylki notum við eina af tveimur aðferðum.
- Sú fyrri:
  - Rótin er í staki 1 í fylkinu.
  - Vinstra barn staksins i er stak $2 \times i$.
  - Hægra barn staksins i er stak $2 \times i + 1$.
  - Foreldri staks i er stakið $\left\lfloor \frac{i}{2} \right\rfloor$.
- Sú seinni:

Fylki sem tré

- Þegar við geymum tréð sem fylki notum við eina af tveimur aðferðum.
- Sú fyrri:
  - Rótin er í staki 1 í fylkinu.
  - Vinstra barn staksins i er stak $2 \times i$.
  - Hægra barn staksins i er stak $2 \times i + 1$.
  - Foreldri staks i er stakið $\left\lfloor \frac{i}{2} \right\rfloor$.
- Sú seinni:
  - Rótin er í staki 0 í fylkinu.

Fylki sem tré

- Þegar við geymum tréð sem fylki notum við eina af tveimur aðferðum.
- Sú fyrri:
  - Rótin er í staki 1 í fylkinu.
  - Vinstra barn staksins i er stak $2 \times i$.
  - Hægra barn staksins i er stak $2 \times i + 1$.
  - Foreldri staks i er stakið $\left\lfloor \frac{i}{2} \right\rfloor$.
- Sú seinni:
  - Rótin er í staki 0 í fylkinu.
  - Vinstra barn staksins i er stak $2 \times i + 1$.

Fylki sem tré

- Þegar við geymum tréð sem fylki notum við eina af tveimur aðferðum.
- Sú fyrri:
    - Rótin er í staki 1 í fylkinu.
    - Vinstra barn staksins i er stak $2 \times i$.
    - Hægra barn staksins i er stak $2 \times i + 1$.
    - Foreldri staks i er stakið $\left\lfloor \frac{i}{2} \right\rfloor$.
- Sú seinni:
    - Rótin er í staki 0 í fylkinu.
    - Vinstra barn staksins i er stak $2 \times i + 1$.
    - Hægra barn staksins i er stak $2 \times i + 2$.

Fylki sem tré

- Þegar við geymum tréð sem fylki notum við eina af tveimur aðferðum.
- Sú fyrri:
  - Rótin er í staki 1 í fylkinu.
  - Vinstra barn staksins i er stak $2 \times i$.
  - Hægra barn staksins i er stak $2 \times i + 1$.
  - Foreldri staks i er stakið $\left\lfloor \frac{i}{2} \right\rfloor$.
- Sú seinni:
  - Rótin er í staki 0 í fylkinu.
  - Vinstra barn staksins i er stak $2 \times i + 1$.
  - Hægra barn staksins i er stak $2 \times i + 2$.
  - Foreldri staks i er stakið $\left\lfloor \frac{i-1}{2} \right\rfloor$.

Hrúga í C

```
#define PARENT(i) ((i - 1)/2)
#define LEFT(i)   ((i)*2 + 1)
#define RIGHT(i)  ((i)*2 + 2)
int h[1000000];
int hn = 0;

void fix_down(int i)
{
    ...
}

void fix_up(int i)
{
    ...
}

void pop()
{
    ...
}

int peek()
{
    ...
}

void push(int x)
{
    ...
}
```

Hrúga í C

```c
void pop()
{
    hn--;
    h[0] = h[hn];
    fix_down(0);
}

int peek()
{
    return h[0];
}

void push(int x)
{
    h[hn++] = x;
    fix_up(hn - 1);
}
```

Hrúga í C

```
void fix_down(int i)
{
    int mx = i;
    if (RIGHT(i) < hn && h[mx] < h[RIGHT(i)]) mx = RIGHT(i);
    if (LEFT(i) < hn && h[mx] < h[LEFT(i)]) mx = LEFT(i);
    if (mx != i)
    {
        swap(h[i], h[mx]);
        fix_down(mx);
    }
}
```

Hrúga í C

```c
void fix_up(int i)
{
    if (i == 0) return;
    else if (h[i] > h[PARENT(i)])
    {
        swap(h[i], h[PARENT(i)]);
        fix_up(PARENT(i));
    }
}
```

Efnisyfirlit

Dæmi

- Gefinn er listi með n tölum.

Dæmi

- Gefinn er listi með n tölum.
- Næst koma q fyrir spurnir, þar sem hver er af einni af tveimur gerðum:

Dæmi

- Gefinn er listi með n tölum.
- Næst koma q fyrir spurnir, þar sem hver er af einni af tveimur gerðum:
  - Breyttu i-tu tölunni í listanum í k.

Dæmi

- Gefinn er listi með n tölum.
- Næst koma q fyrir spurnir, þar sem hver er af einni af tveimur gerðum:
  - Breyttu i-tu tölunni í listanum í k.
  - Reiknaðu summu allra talna á bilinu $[i, j]$.

Dæmi

- Gefinn er listi með n tölum.
- Næst koma q fyrir spurnir, þar sem hver er af einni af tveimur gerðum:
  - Breyttu i-tu tölunni í listanum í k.
  - Reiknaðu summu allra talna á bilinu $[i, j]$.
- Það er auðséð að einföld útfærlsa á þessum fyrir spurnum gefur okkur $\mathcal{O}(1)$ fyrir þá fyrri og $\mathcal{O}(n)$ fyrir þá seinni.

Dæmi

- Gefinn er listi með n tölum.
- Næst koma q fyrir spurnir, þar sem hver er af einni af tveimur gerðum:
  - Breyttu i-tu tölunni í listanum í k.
  - Reiknaðu summu allra talna á bilinu $[i, j]$.
- Það er auðséð að einföld útfærlsa á þessum fyrir spurnum gefur okkur $\mathcal{O}(1)$ fyrir þá fyrri og $\mathcal{O}(n)$ fyrir þá seinni.
- Þar sem allar (eða langflestar) fyrirspurnir gætu verið af seinni gerðin yrði lausnin í heildin $\mathcal{O}(qn)$.

Dæmi

- Gefinn er listi með n tölum.
- Næst koma q fyrir spurnir, þar sem hver er af einni af tveimur gerðum:
    - Breyttu i-tu tölunni í listanum í k.
    - Reiknaðu summu allra talna á bilinu $[i, j]$.
- Það er auðséð að einföld útfærlsa á þessum fyrir spurnum gefur okkur $\mathcal{O}(1)$ fyrir þá fyrri og $\mathcal{O}(n)$ fyrir þá seinni.
- Þar sem allar (eða langflestar) fyrirspurnir gætu verið af seinni gerðin yrði lausnin í heildin $\mathcal{O}(qn)$.
- Það er þó hægt að leysa þetta dæmi hraðar.

Dæmi

- Gefinn er listi með n tölum.
- Næst koma q fyrir spurnir, þar sem hver er af einni af tveimur gerðum:
  - Breyttu i-tu tölunni í listanum í k.
  - Reiknaðu summu allra talna á bilinu $[i, j]$.
- Það er auðséð að einföld útfærlsa á þessum fyrir spurnum gefur okkur $\mathcal{O}(1)$ fyrir þá fyrri og $\mathcal{O}(n)$ fyrir þá seinni.
- Þar sem allar (eða langflestar) fyrirspurnir gætu verið af seinni gerðin yrði lausnin í heildin $\mathcal{O}(qn)$.
- Það er þó hægt að leysa þetta dæmi hraðar.
- Algengt er að nota til þess *biltré*.

Biltré

- Biltré (e. segment tree) er tvíundartré sem geymir svör við vissum fyrirspurnum af seinni gerðinni.

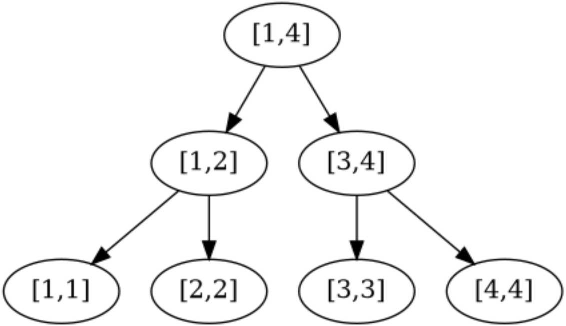
Biltré

- Biltré (e. segment tree) er tvíundartré sem geymir svör við vissum fyrirspurnum af seinni gerðinni.
- Rótin geymir svar við fyrirspurninni `1 n` og ef nóða geymir svarið við `i j` þá geyma börn hennar svör við `i m` og `m + 1 j`, þar sem m er miðja heiltölubilsins $[i, j]$.

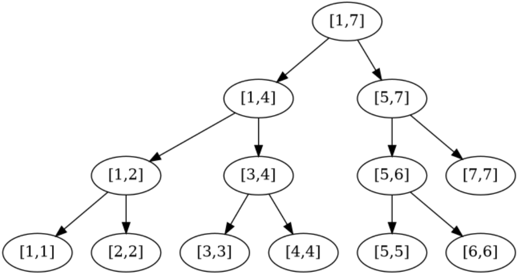
Biltré

- Biltré (e. segment tree) er tvíundartré sem geymir svör við vissum fyrirspurnum af seinni gerðinni.
- Rótin geymir svar við fyrirspurninni `1 n` og ef nóða geymir svarið við `i j` þá geyma börn hennar svör við `i m` og `m + 1 j`, þar sem m er miðja heiltölubilsins $[i, j]$.
- Þær nóður sem geyma svar við fyrirspurnum af gerðinni `i i` eru lauf trésins.

Mynd af biltré, $n = 4$



Mynd af biltré, $n = 7$



Notkun biltrjáa

- Gerum ráð fyrir að við höfum biltré eins og lýst er að ofan og látum H tákna hæð trésins.

Notkun biltrjáa

- Gerum ráð fyrir að við höfum biltré eins og lýst er að ofan og látum H tákna hæð trésins.
- Hvernig getum við leyst fyrirspurnirnar á glærunni á undan, og hver er tímaflækjan?

Notkun biltrjáa

- Gerum ráð fyrir að við höfum biltré eins og lýst er að ofan og látum H tákna hæð trésins.
- Hvernig getum við leyst fyrirspurnirnar á glærunni á undan, og hver er tímaflækjan?
- Fyrri fyrirspurnin er einföld.

Notkun biltrjáa

- Gerum ráð fyrir að við höfum biltré eins og lýst er að ofan og látum H tákna hæð trésins.
- Hvernig getum við leyst fyrirspurnirnar á glærunni á undan, og hver er tímaflækjan?
- Fyrri fyrirspurnin er einföld.
- Ef við eigum að breyta i-ta stakinu í k finnum við fyrst laufið sem svarar til fyrirspurnar `i i`, setjum svarið þar sem k og förum svo upp í rót í gegnum foreldrana og uppfærum á leiðinni gildin í þeim nóðu sem við lendum í.

Notkun biltrjáa

- Gerum ráð fyrir að við höfum biltré eins og lýst er að ofan og látum H tákna hæð trésins.
- Hvernig getum við leyst fyrirspurnirnar á glærunni á undan, og hver er tímaflækjan?
- Fyrri fyrirspurnin er einföld.
- Ef við eigum að breyta i-ta stakinu í k finnum við fyrst laufið sem svarar til fyrirspurnar `i i`, setjum svarið þar sem k og förum svo upp í rót í gegnum foreldrana og uppfærum á leiðinni gildin í þeim nóðu sem við lendum í.
- Þar sem við heimsækjum bara þær nóður sem eru á veginum frá rót til laufs (mest H nóður) er tímaflækjan á fyrri fyrirspurninn $\mathcal{O}(H)$.

Biltré í C

```c
// ath: p er af staerd 4*n + 1
#define LEFT(x) ((x)*2)
#define RIGHT(x) ((x)*2 + 1)
void update(int* p, int i, int j, int x, int y, int e)
{
    if (i == j) p[e] = y;
    else
    {
        int m = (i + j)/2;
        if (x <= m) update(p, i, m, x, y, LEFT(e));
        else update(p, m + 1, j, x, y, RIGHT(e));
        p[e] = p[LEFT(e)] + p[RIGHT(e)];
    }
}
```

Notkun biltrjáa

- Seinni fyrirspurnin er ögn flóknari.

Notkun biltrjáa

- Seinni fyrirspurnin er ögn flóknari.
- Auðveldast er að ímynda sér að við förum niður tréð og leitum að hvorum endapunktinum fyrir sig.

Notkun biltrjáa

- Seinni fyrirspurnin er ögn flóknari.
- Auðveldast er að ímynda sér að við förum niður tréð og leitum að hvorum endapunktinum fyrir sig.
- Á leiðinni upp getum við svo pússlað saman svarinu, eftir því hvort við erum að skoða hægri eða vinstri endapunktinn.

Notkun biltrjáa

- Seinni fyrirspurnin er ögn flóknari.
- Auðveldast er að ímynda sér að við förum niður tréð og leitum að hvorum endapunktinum fyrir sig.
- Á leiðinni upp getum við svo pússlað saman svarinu, eftir því hvort við erum að skoða hægri eða vinstri endapunktinn.
- Til dæmis, ef við erum að leita að vinstri endapunkti x og komum upp í bil $[i, j]$ þá bætum við gildinu í nóðu $[i, m]$ við það sem við höfum reiknað hingað til ef $x \in [m + 1, j]$, en annars bætum við engu við (því x er vinstri endapunkturinn).

Notkun biltrjáa

- Seinni fyrirspurnin er ögn flóknari.
- Auðveldast er að ímynda sér að við förum niður tréð og leitum að hvorum endapunktinum fyrir sig.
- Á leiðinni upp getum við svo pússlað saman svarinu, eftir því hvort við erum að skoða hægri eða vinstri endapunktinn.
- Til dæmis, ef við erum að leita að vinstri endapunkti x og komum upp í bil $[i, j]$ þá bætum við gildinu í nóðu $[i, m]$ við það sem við höfum reiknað hingað til ef $x \in [m + 1, j]$, en annars bætum við engu við (því x er vinstri endapunkturinn).
- Við göngum svona upp þar til við lendum í bili sem inniheldur hinn endapunktinn.

Notkun biltrjáa

- Seinni fyrirspurnin er ögn flóknari.
- Auðveldast er að ímynda sér að við förum niður tréð og leitum að hvorum endapunktinum fyrir sig.
- Á leiðinni upp getum við svo pússlað saman svarinu, eftir því hvort við erum að skoða hægri eða vinstri endapunktinn.
- Til dæmis, ef við erum að leita að vinstri endapunkti x og komum upp í bil $[i, j]$ þá bætum við gildinu í nóðu $[i, m]$ við það sem við höfum reiknað hingað til ef $x \in [m + 1, j]$, en annars bætum við engu við (því x er vinstri endapunkturinn).
- Við göngum svona upp þar til við lendum í bili sem inniheldur hinn endapunktinn.
- Með sömu rökum og áðan er tímaflækjan $\mathcal{O}(H)$.

Biltré í C

```c
int queryl(int* p, int i, int j, int x, int e)
{
    if (i == j) return p[e];
    int m = (i + j)/2;
    return (x <= m) ? (queryl(p, i, m, x, LEFT(e)) + p[RIGHT(e)])
                    : (queryl(p, m + 1, j, x, RIGHT(e)));
}

int queryr(int* p, int i, int j, int x, int e)
{
    if (i == j) return p[e];
    int m = (i + j)/2;
    return (x <= m) ? (queryr(p, i, m, x, LEFT(e)))
                    : (p[LEFT(e)] + queryr(p, m + 1, j, x, RIGHT(e)));
}

int query(int* p, int i, int j, int x, int y, int e)
{
    if (i == j) return p[e];
    int m = (i + j)/2;
    if (x <= m && y <= m) return query(p, i, m, x, y, LEFT(e));
    if (x > m && y > m) return query(p, m + 1, j, x, y, RIGHT(e));
    return queryl(p, i, m, x, LEFT(e)) + queryr(p, m + 1, j, y, RIGHT(e));
}
```

Tímaflækja biltrjáa

- Þar sem lengd hvers bils sem nóða svara til helmingast þegar farið er niður tréð er $\mathcal{O}(H) = \mathcal{O}(\log n)$.

Tímaflækja biltrjáa

- Þar sem lengd hvers bils sem nóða svara til helmingast þegar farið er niður tréð er $\mathcal{O}(H) = \mathcal{O}(\log n)$.
- Við erum því komin með lausn á upprunalega dæminu sem er $\mathcal{O}(q \log n)$.

Tímaflækja biltrjáa

- Þar sem lengd hvers bils sem nóða svara til helmingast þegar farið er niður tréð er $\mathcal{O}(H) = \mathcal{O}(\log n)$.
- Við erum því komin með lausn á upprunalega dæminu sem er $\mathcal{O}(q \log n)$.
- Þetta væri nógu hratt ef, til dæmis, $n = q = 10^5$.

Annað dæmi

- Fyrsta lína inntaksins inniheldur tvær tölur, n og m, báðar jákvæðar heiltölur minni en 10^5.

Annað dæmi

- Fyrsta lína inntaksins inniheldur tvær tölur, n og m, báðar jákvæðar heiltölur minni en 10^5.
- Næsta lína inniheldur n heiltölur, á milli -10^9 og 10^9.

Annað dæmi

- Fyrsta lína inntaksins inniheldur tvær tölur, n og m, báðar jákvæðar heiltölur minni en 10^5.
- Næsta lína inniheldur n heiltölur, á milli -10^9 og 10^9.
- Næstu m línur innihalda fyrirspurnir, af tveimur gerðum.

Annað dæmi

- Fyrsta lína inntaksins inniheldur tvær tölur, n og m, báðar jákvæðar heiltölur minni en 10^5.
- Næsta lína inniheldur n heiltölur, á milli -10^9 og 10^9.
- Næstu m línur innihalda fyrirspurnir, af tveimur gerðum.
- Fyrri gerðin hefst á 1 og inniheldur svo tvær tölur, x og y. Hér á að setja x-tu töluna sem y.

Annað dæmi

- Fyrsta lína inntaksins inniheldur tvær tölur, n og m, báðar jákvæðar heiltölur minni en 10^5.
- Næsta lína inniheldur n heiltölur, á milli -10^9 og 10^9.
- Næstu m línur innihalda fyrirspurnir, af tveimur gerðum.
- Fyrri gerðin hefst á 1 og inniheldur svo tvær tölur, x og y. Hér á að setja x-tu töluna sem y.
- Seinni gerðin hefst á 2 og inniheldur svo tvær tölu, x og y. Hér á að prenta út stærstu töluna á hlutbilinu $[x, y]$ í talnalistanum.

Annað dæmi

- Fyrsta lína inntaksins inniheldur tvær tölur, n og m, báðar jákvæðar heiltölur minni en 10^5.
- Næsta lína inniheldur n heiltölur, á milli -10^9 og 10^9.
- Næstu m línur innihalda fyrirspurnir, af tveimur gerðum.
- Fyrri gerðin hefst á 1 og inniheldur svo tvær tölur, x og y. Hér á að setja x-tu töluna sem y.
- Seinni gerðin hefst á 2 og inniheldur svo tvær tölu, x og y. Hér á að prenta út stærstu töluna á hlutbilinu $[x, y]$ í talnalistanum.
- Hvernig leysum við þetta?

Lausn

```
int queryl(int* p, int i, int j, int x, int e)
{
    if (i == j) return p[e];
    int m = (i + j)/2;
    return (x <= m) ? (max(queryl(p, i, m, x, LEFT(e)), p[RIGHT(e)]))
                    : (queryl(p, m + 1, j, x, RIGHT(e)));
}
int queryr(int* p, int i, int j, int x, int e)
{
    if (i == j) return p[e];
    int m = (i + j)/2;
    return (x <= m) ? (queryr(p, i, m, x, LEFT(e)))
                    : (max(p[LEFT(e)], queryr(p, m + 1, j, x, RIGHT(e))));
}
int query(int* p, int i, int j, int x, int y, int e)
{
    if (i == j) return p[e];
    int m = (i + j)/2;
    if (x <= m && y <= m) return query(p, i, m, x, y, LEFT(e));
    if (x > m && y > m) return query(p, m + 1, j, x, y, RIGHT(e));
    return max(queryl(p, i, m, x, LEFT(e)), queryr(p, m + 1, j, y, RIGHT(e)));
}
void update(int* p, int i, int j, int x, int y, int e)
{
    if (i == j) p[e] = y;
    else
    {
        int m = (i + j)/2;
        if (x <= m) update(p, i, m, x, y, LEFT(e));
        else update(p, m + 1, j, x, y, RIGHT(e));
        p[e] = max(p[LEFT(e)], p[RIGHT(e)]);
    }
}
```

Lausn

```c
int main()
{
    int n, m, i, x, y, z;
    scanf("%d%d", &n, &m);
    int a[n], p[4*n + 1];
    for (i = 0; i < n; i++) scanf("%d", &(a[i]));
    for (i = 0; i < 4*n + 1; i++) p[i] = 0;
    for (i = 0; i < n; i++) update(p, 0, n - 1, i, a[i], 1);
    while (m-- != 0)
    {
        scanf("%d%d%d", &x, &y, &z);
        if (x == 1) update(p, 0, n - 1, y - 1, z, 1);
        if (x == 2) printf("%d\n", query(p, 0, n - 1, y - 1, z - 1, 1));
    }

    return 0;
}
```

Efnisyfirlit

Sammengisleit

- Sammengisleit (e. union-find) er öflug leið til að halda utan um jafngildisflokka tiltekna vensla, eða m.ö.o. halda utan um *sundurlæg* mengi.

Sammengisleit

- Sammengisleit (e. union-find) er öflug leið til að halda utan um jafngildisflokka tiltekna vensla, eða m.ö.o. halda utan um *sundurlæg* mengi.
- Við viljum getað:

Sammengisleit

- Sammengisleit (e. union-find) er öflug leið til að halda utan um jafngildisflokka tiltekna vensla, eða m.ö.o. halda utan um *sundurlæg* mengi.
- Við viljum getað:
  - Borið saman samhengisþætti mismunandi staka.

Sammengisleit

- Sammengisleit (e. union-find) er öflug leið til að halda utan um jafngildisflokka tiltekna vensla, eða m.ö.o. halda utan um *sundurlæg* mengi.
- Við viljum getað:
  - Borið saman samhengisþætti mismunandi staka.
  - Sameinað samhengisflokka.

Sammengisleit

- Sammengisleit (e. union-find) er öflug leið til að halda utan um jafngildisflokka tiltekna vensla, eða m.ö.o. halda utan um *sundurlæg* mengi.
- Við viljum getað:
  - Borið saman samhengisþætti mismunandi staka.
  - Sameinað samhengisflokka.
- Við tölum um aðgerðirnar `find(x)` og `join(x, y)`.

Dæmi

- Tökum sem dæmi einstökungasafnið $\{\{1\},\{2\},\{3\},\{4\},\{5\}\}$.

Dæmi

- Tökum sem dæmi einstökungasafnið $\{\{1\},\{2\},\{3\},\{4\},\{5\}\}$.
- `join(1, 3)` gefur okkur $\{\{1,3\},\{2\},\{4\},\{5\}\}$.

Dæmi

- Tökum sem dæmi einstökungasafnið $\{\{1\}, \{2\}, \{3\}, \{4\}, \{5\}\}$.
- `join(1, 3)` gefur okkur $\{\{1, 3\}, \{2\}, \{4\}, \{5\}\}$.
- `join(2, 5)` gefur okkur $\{\{1, 3\}, \{2, 5\}, \{4\}\}$.

Dæmi

- Tökum sem dæmi einstökungasafnið $\{\{1\}, \{2\}, \{3\}, \{4\}, \{5\}\}$.
- `join(1, 3)` gefur okkur $\{\{1, 3\}, \{2\}, \{4\}, \{5\}\}$.
- `join(2, 5)` gefur okkur $\{\{1, 3\}, \{2, 5\}, \{4\}\}$.
- `join(2, 4)` gefur okkur $\{\{1, 3\}, \{2, 4, 5\}\}$.

Dæmi

- Tökum sem dæmi einstökungasafnið $\{\{1\}, \{2\}, \{3\}, \{4\}, \{5\}\}$.
- `join(1, 3)` gefur okkur $\{\{1, 3\}, \{2\}, \{4\}, \{5\}\}$.
- `join(2, 5)` gefur okkur $\{\{1, 3\}, \{2, 5\}, \{4\}\}$.
- `join(2, 4)` gefur okkur $\{\{1, 3\}, \{2, 4, 5\}\}$.
- `join(1, 4)` gefur okkur $\{\{1, 2, 3, 4, 5\}\}$.

Dæmi

- Tökum sem dæmi einstökungasafnið $\{\{1\}, \{2\}, \{3\}, \{4\}, \{5\}\}$.
- `join(1, 3)` gefur okkur $\{\{1, 3\}, \{2\}, \{4\}, \{5\}\}$.
- `join(2, 5)` gefur okkur $\{\{1, 3\}, \{2, 5\}, \{4\}\}$.
- `join(2, 4)` gefur okkur $\{\{1, 3\}, \{2, 4, 5\}\}$.
- `join(1, 4)` gefur okkur $\{\{1, 2, 3, 4, 5\}\}$.
- Á sérhverjum tímapunkti myndi `find(x)` skila einhverju staki sem er í sama mengi og `x`.

Dæmi

- Tökum sem dæmi einstökungasafnið $\{\{1\}, \{2\}, \{3\}, \{4\}, \{5\}\}$.
- `join(1, 3)` gefur okkur $\{\{1, 3\}, \{2\}, \{4\}, \{5\}\}$.
- `join(2, 5)` gefur okkur $\{\{1, 3\}, \{2, 5\}, \{4\}\}$.
- `join(2, 4)` gefur okkur $\{\{1, 3\}, \{2, 4, 5\}\}$.
- `join(1, 4)` gefur okkur $\{\{1, 2, 3, 4, 5\}\}$.
- Á sérhverjum tímapunkti myndi `find(x)` skila einhverju staki sem er í sama mengi og `x`.
- Aðalatriðið er að `find(x)` skilar sama stakinu fyrir sérhvert stak í sérhverjum samhengisflokki.

Dæmi

- Tökum sem dæmi einstökungasafnið $\{\{1\},\{2\},\{3\},\{4\},\{5\}\}$.
- `join(1, 3)` gefur okkur $\{\{1,3\},\{2\},\{4\},\{5\}\}$.
- `join(2, 5)` gefur okkur $\{\{1,3\},\{2,5\},\{4\}\}$.
- `join(2, 4)` gefur okkur $\{\{1,3\},\{2,4,5\}\}$.
- `join(1, 4)` gefur okkur $\{\{1,2,3,4,5\}\}$.
- Á sérhverjum tímapunkti myndi `find(x)` skila einhverju staki sem er í sama mengi og `x`.
- Aðalatriðið er að `find(x)` skilar sama stakinu fyrir sérhvert stak í sérhverjum samhengisflokki.
- Til dæmis, í þriðja punktinum myndi `find(1)` og `find(3)` alltaf skila sama stakinu.

- Gerum ráð fyrir að tölurnar sem við munum vinna með séu jákvæðar og minni en n.

Útfærsla á frumstæðri sammengisleit

- Gerum ráð fyrir að tölurnar sem við munum vinna með séu jákvæðar og minni en n.
- Við munum þá gefa okkur n staka fylki p, þar sem i-ta stakið í fylkinu er upphafstillt sem i.

Útfærsla á frumstæðri sammengisleit

- Gerum ráð fyrir að tölurnar sem við munum vinna með séu jákvæðar og minni en n.
- Við munum þá gefa okkur n staka fylki p, þar sem i-ta stakið í fylkinu er upphafstillt sem i.
- Fylkið p mun nú geyma *foreldri* sérhvers stak.

Útfærsla á frumstæðri sammengisleit

- Gerum ráð fyrir að tölurnar sem við munum vinna með séu jákvæðar og minni en n.
- Við munum þá gefa okkur n staka fylki p, þar sem i-ta stakið í fylkinu er upphafstillt sem i.
- Fylkið p mun nú geyma *foreldri* sérhvers stak.
- Foreldrin myndi keðjur.

Útfærsla á frumstæðri sammengisleit

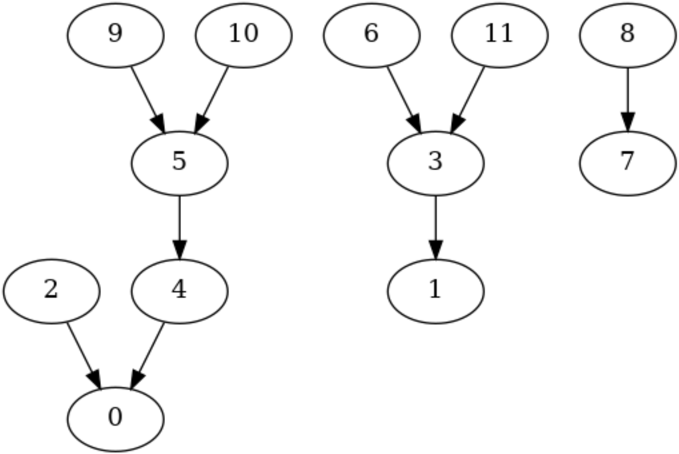
- Gerum ráð fyrir að tölurnar sem við munum vinna með séu jákvæðar og minni en n.
- Við munum þá gefa okkur n staka fylki p, þar sem i-ta stakið í fylkinu er upphafstillt sem i.
- Fylkið p mun nú geyma *foreldri* sérhvers stak.
- Foreldrin myndi keðjur.
- Sérhver keðja endar í einhverju staki, sem við munum kalla *ráðherra* jafngildisflokksins.

Mynd af keðjum

Keðjurnar sem fást með $\{\{0, 2, 4, 5, 9, 10\}, \{1, 3, 6, 11\}, \{7, 8\}\}$ gætu til dæmis verið gefnar með $p = [0, 1, 0, 1, 0, 4, 3, 7, 7, 5, 5, 3]$.

Mynd af keðjum

Keðjurnar sem fást með $\{\{0, 2, 4, 5, 9, 10\}, \{1, 3, 6, 11\}, \{7, 8\}\}$ gætu til dæmis verið gefnar með $p = [0, 1, 0, 1, 0, 4, 3, 7, 7, 5, 5, 3]$.



Útfærsla á frumstæðri sammengisleit

- Til að fá ráðherra flokks tiltekins staks er hægt að fara endurkvæmt upp keðjuna.

Útfærsla á frumstæðri sammengisleit

- Til að fá ráðherra flokks tiltekins staks er hægt að fara endurkvæmt upp keðjuna.
- Til að sameina flokka nægir að breyta foreldri ráðherra annars flokksins yfir í eitthvert stak hins flokksins (sér í lagi ráðherra þess).

Útfærsla á frumstæðri sammengisleit

- Til að fá ráðherra flokks tiltekins staks er hægt að fara endurkvæmt upp keðjuna.
- Til að sameina flokka nægir að breyta foreldri ráðherra annars flokksins yfir í eitthvert stak hins flokksins (sér í lagi ráðherra þess).
- Báðar þessar aðgerðir er auðvelt að útfæra.

Frumstæð sammengisleit

```c
int p[MAX];

int find(int x)
{
    if (p[x] == x) return x;
    return find(p[x]);
}

void join(int x, int y)
{
    p[find(x)] = find(y);
}

int main()
{
    int i;
    int n = MAX;
    for (i = 0; i < n; i++) p[i] = i;
    ...
}
```

Tímaflækjur frumstæðri sammengisleitar

- Við sjáum nú að tímaflækja `find` er línuleg í lengd keðjunnar, svo þar sem lengd keðjunnar getur verið í versta falli n þá er `find` $\mathcal{O}(n)$.

Tímaflækjur frumstæðri sammengisleitar

- Við sjáum nú að tímaflækja `find` er línuleg í lengd keðjunnar, svo þar sem lengd keðjunnar getur verið í versta falli n þá er `find` $\mathcal{O}(n)$.
- Fallið `join` gerir lítið annað en að kalla tvisvar á `find` svo það er líka $\mathcal{O}(n)$.

Tímaflækjur frumstæðri sammengisleitar

- Við sjáum nú að tímaflækja `find` er línuleg í lengd keðjunnar, svo þar sem lengd keðjunnar getur verið í versta falli n þá er `find` $\mathcal{O}(n)$.
- Fallið `join` gerir lítið annað en að kalla tvisvar á `find` svo það er líka $\mathcal{O}(n)$.
- Er samt ekki hægt að bæta þetta eitthvað?

Tímaflækjur frumstæðri sammengisleitar

- Við sjáum nú að tímaflækja `find` er línuleg í lengd keðjunnar, svo þar sem lengd keðjunnar getur verið í versta falli n þá er `find` $\mathcal{O}(n)$.
- Fallið `join` gerir lítið annað en að kalla tvisvar á `find` svo það er líka $\mathcal{O}(n)$.
- Er samt ekki hægt að bæta þetta eitthvað?
- Það er vissulega hægt!

Keðjuþjöppuð sammengisleit

- Eins og nafnið á glærunni gefur til kynna er hugmyndin að þjappa keðjunum saman í hvert skipti sem kallað er á `find`.

Keðjuþjöppuð sammengisleit

- Eins og nafnið á glærunni gefur til kynna er hugmyndin að þjappa keðjunum saman í hvert skipti sem kallað er á `find`.
- Þetta er gert með því að setja `p[x]` sem ráðherra flokks `x`, í hverju skrefi endurkvæmninnar.

Dæmi um keðjuþjöppun

- Gefum okkur $p = [0, 0, 1, 2, 3, 4, 5, 6, 7]$.

Dæmi um keðjuþjöppun

- Gefum okkur $p = [0, 0, 1, 2, 3, 4, 5, 6, 7]$.
- Ljóst er að `find(5)` skilar 0.

Dæmi um keðjuþjöppun

- Gefum okkur $p = [0, 0, 1, 2, 3, 4, 5, 6, 7]$.
- Ljóst er að `find(5)` skilar 0.
- Ef við notum frumstæða sammengisleit breytist p ekki neitt þegar kallað er á `find` en með keðjuþjappaðri sammengisleit þjappast keðjan frá og með 5 og því fæst $p = [0, 0, 0, 0, 0, 0, 5, 6, 7]$.

Keðjuþjöppað sammengisleit

```
int p[MAX];

int find(int x)
{
    if (p[x] == x) return x;
    return p[x] = find(p[x]);
}

void join(int x, int y)
{
    p[find(x)] = find(y);
}

int main()
{
    int i;
    int n = MAX;
    for (i = 0; i < n; i++) p[i] = i;
    ...
}
```

Tímaflækjur keðjuþjappaðar sammengisleitar

- Það er flóknara að lýsa tímaflækju keðjuþjappaðrar sammengisleitar.

Tímaflækjur keðjuþjappaðar sammengisleitar

- Það er flóknara að lýsa tímaflækju keðjuþjappaðrar sammengisleitar.
- *Á heildina litið* (e. amortized) er tímaflækjan er $\mathcal{O}(\alpha(n))$, þar sem α er andhverfa *Ackermann* fallsins.

Tímaflækjur keðjuþjappaðar sammengisleitar

- Það er flóknara að lýsa tímaflækju keðjuþjappaðrar sammengisleitar.
- *Á heildina litið* (e. amortized) er tímaflækjan er $\mathcal{O}(\alpha(n))$, þar sem α er andhverfa *Ackermann* fallsins.
- Fyrir þau n sem við fáumst við er $\alpha(n)$ nánast fast.