Gagnagrindur

Listar, forgangsbiðraðir og sammengisleit

Bergur Snorrason

9. febrúar 2019

Efnisyfirlit

Listar

- Listi er einföld gagnagrind notuð til að geyma gögn á skilvirkann hátt.

Listar

- Listi er einföld gagnagrind notuð til að geyma gögn á skilvirkann hátt.
- Við viljum getað:

Listar

- Listi er einföld gagnagrind notuð til að geyma gögn á skilvirkann hátt.
- Við viljum getað:
    - Bætt staki í listann.

Listar

- Listi er einföld gagnagrind notuð til að geyma gögn á skilvirkann hátt.
- Við viljum getað:
    - Bætt staki í listann.
    - Leitað að staki í listanum.

Listar

- Listi er einföld gagnagrind notuð til að geyma gögn á skilvirkann hátt.
- Við viljum getað:
    - Bætt staki í listann.
    - Leitað að staki í listanum.
    - Eytt staki úr listanum.

Eintengdir listar

- Einfaldasta (og sennilega algengasta) leiðin til að útfæra lista er með eintengdum lista (singly linked list).

Eintengdir listar

- Einfaldasta (og sennilega algengasta) leiðin til að útfæra lista er með eintengdum lista (singly linked list).
- Eintengdur listi samanstendur af nóðum.

Eintengdir listar

- Einfaldasta (og sennilega algengasta) leiðin til að útfæra lista er með eintengdum lista (singly linked list).
- Eintengdur listi samanstendur af nóðum.
- Hver nóða geymir stak í listanum og upplýsingar um staðsetningu (bendi á) næstu nóðu.

Útfærsla

```
typedef struct snode
{
    int v;
    struct snode* n;
} node;

class llist
{
    public:
    node* h;
    node* t;
    llist()
    {
        h = NULL; t = NULL;
    }
    void add(int n)
    {
        ...
    }
    bool find(int n)
    {
        ...
    }
    bool del(int n)
    {
        ...
    }
};
```

Útfærsla

```
void add(int n)
{
    node* a = new node;
    a->v = n;
    a->n = NULL;
    if (h == NULL)
    {
        h = a; t = a;
    }
    else
    {
        t->n = a;
        t = t->n;
    }
}
```

Útfærsla

```
bool find(int n)
{
    node* g = h;
    while (g != NULL)
    {
        if (g->v == n)
        {
            break;
        }
        g = g->n;
    }
    return g != NULL;
}
```

Útfærsla

```
bool del(int n)
{
    if (h->v == n)
    {
        if (h == t)
        {
            delete h; h = NULL; t = NULL; return true;
        }
        node* g = h; h = h->n; delete g; return true;
    }
    node* gg = h; node* g = h->n;
    while (g != NULL)
    {
        if (g->v == n)
        {
            break;
        }
        gg = g; g = g->n;
    }
    if (g == NULL)
    {
        return false;
    }
    gg->n = g->n;
    if (g == t)
    {
        t = gg;
    }
    delete g;
    return true;
}
```

Kostir og gallar

- Gallar:

Kostir og gallar

- Gallar:
    - Mjög hæg leit.

Kostir og gallar

- Gallar:
  - Mjög hæg leit.
  - Býður ekki upp á handahófskennda vísun (e. random access).

Kostir og gallar

- Gallar:
  - Mjög hæg leit.
  - Býður ekki upp á handahófskennda vísun (e. random access).
- Kostir:

Kostir og gallar

- Gallar:
  - Mjög hæg leit.
  - Býður ekki upp á handahófskennda vísun (e. random access).
- Kostir:
  - Er hluti af `C++ STL`.

Kostir og gallar

- Gallar:
  - Mjög hæg leit.
  - Býður ekki upp á handahófskennda vísun (e. random access).
- Kostir:
  - Er hluti af `C++ STL`.
  - Unnt er að skeyta saman listum í $\mathcal{O}(1)$, eitthvað sem er til dæmis ekki unnt með `vector` í `C++`.

Tímaflækjur

- Innsetning: $\mathcal{O}(1)$.

Tímaflækjur

- Innsetning: $\mathcal{O}(1)$.
- Leita að staki: $\mathcal{O}(n)$.

Tímaflækjur

- Innsetning: $\mathcal{O}(1)$.
- Leita að staki: $\mathcal{O}(n)$.
- Eyða taki: $\mathcal{O}(n)$.

Efnisyfirlit

Forgangsbiðraðir

- Forgangsbiðraðir (e. priority queues) eru þægilegar gagnagrindur að hafa.

Forgangsbiðraðir

- Forgangsbiðraðir (e. priority queues) eru þægilegar gagnagrindur að hafa.
- Við notum þær til að geyma röðuð gögn.

Forgangsbiðraðir

- Forgangsbiðraðir (e. priority queues) eru þægilegar gagnagrindur að hafa.
- Við notum þær til að geyma röðuð gögn.
- Við viljum getað:

Forgangsbiðraðir

- Forgangsbiðraðir (e. priority queues) eru þægilegar gagnagrindur að hafa.
- Við notum þær til að geyma röðuð gögn.
- Við viljum getað:
  - Bætt staki í biðröðina.

Forgangsbiðraðir

- Forgangsbiðraðir (e. priority queues) eru þægilegar gagnagrindur að hafa.
- Við notum þær til að geyma röðuð gögn.
- Við viljum getað:
  - Bætt staki í biðröðina.
  - Fundið *besta* stakið í biðröðinni.

Forgangsbiðraðir

- Forgangsbiðraðir (e. priority queues) eru þægilegar gagnagrindur að hafa.
- Við notum þær til að geyma röðuð gögn.
- Við viljum getað:
    - Bætt staki í biðröðina.
    - Fundið *besta* stakið í biðröðinni.
    - Eytt besta stakinu úr biðröðinni.

Hrúgur

- Algengasta leiðin til að útfæra forgangsbiðraðir er að nota hrúgu (e. heap)

Hrúgur

- Algengasta leiðin til að útfæra forgangsbiðraðir er að nota hrúgu (e. heap)
- Hrúga er tvíundartré sem uppfyllir *hrúguskilyrðinu*.

Hrúgur

- Algengasta leiðin til að útfæra forgangsbiðraðir er að nota hrúgu (e. heap)
- Hrúga er tvíundartré sem uppfyllir *hrúguskilyrðinu*.
- *Hrúguskilyrðið*: Sérhver nóða er betri en, eða jafn góð, börnum sínum.

Framsetning hrúga í tölvum

- Hrúgur eru nokkuð einfaldar í útfærslu.

Framsetning hrúga í tölvum

- Hrúgur eru nokkuð einfaldar í útfærslu.
- Hægt er að tákna *tréð* sem *fylki*.

Framsetning hrúga í tölvum

- Hrúgur eru nokkuð einfaldar í útfærslu.
- Hægt er að tákna *tréð* sem *fylki*.
- Helsta atriðið verður að útfæra *hrúguskilyrðis lagara*.

Framsetning hrúga í tölvum

- Hrúgur eru nokkuð einfaldar í útfærslu.
- Hægt er að tákna *tréð* sem *fylki*.
- Helsta atriðið verður að útfæra *hrúguskilyrðis lagara*.
- Þegar tvíundartré eru útfærð með fylkjum er yfirleitt notuð önnur af tveimur aðferðum.

Fylki sem tré

- Sú fyrri:

Fylki sem tré

- Sú fyrri:
    - Rótin er í staki 1 í fylkinu.

Fylki sem tré

- Sú fyrri:
  - Rótin er í staki 1 í fylkinu.
  - Vinstra barn staksins i er stak $2 \times i$.

Fylki sem tré

- 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

- 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

- 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

- 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

- 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

- 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

- 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 $\left\lfloor \frac{i-1}{2} \right\rfloor$.

Hrúgur í C

```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)
{
    ...
}
```

Hrúgur 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);
}
```

Hrúgur í C

```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)
    {
        int s = h[i];
        h[i] = h[mx];
        h[mx] = s;
        fix_down(mx);
    }
}
```

Hrúgur í C

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

Kostir og gallar

- Gallar:

Kostir og gallar

- Gallar:
  - Styður ekki beint almenna leit.

Kostir og gallar

- Gallar:
    - Styður ekki beint almenna leit.
    - Styður ekki beint samruna.

Kostir og gallar

- Gallar:
  - Styður ekki beint almenna leit.
  - Styður ekki beint samruna.
- Kostir:

Kostir og gallar

- Gallar:
  - Styður ekki beint almenna leit.
  - Styður ekki beint samruna.
- Kostir:
  - Er hluti af `C++ STL`.

Kostir og gallar

- Gallar:
  - Styður ekki beint almenna leit.
  - Styður ekki beint samruna.
- Kostir:
  - Er hluti af `C++ STL`.
  - Heldur kvikt utan um röðun.

- Finna *besta* stakið: $\mathcal{O}(1)$.

Tímaflækjur

- Finna *besta* stakið: $\mathcal{O}(1)$.
- Innsetning: $\mathcal{O}(\log n)$.

Tímaflækjur

- Finna *besta* stakið: $\mathcal{O}(1)$.
- Innsetning: $\mathcal{O}(\log n)$.
- Eyða *besta* stakinu: $\mathcal{O}(\log n)$.

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.

Útfærsla á frumstæðri sammengisleit

- 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.
- Foreldrin myndi keðjur.

Útfærsla á frumstæðri sammengisleit

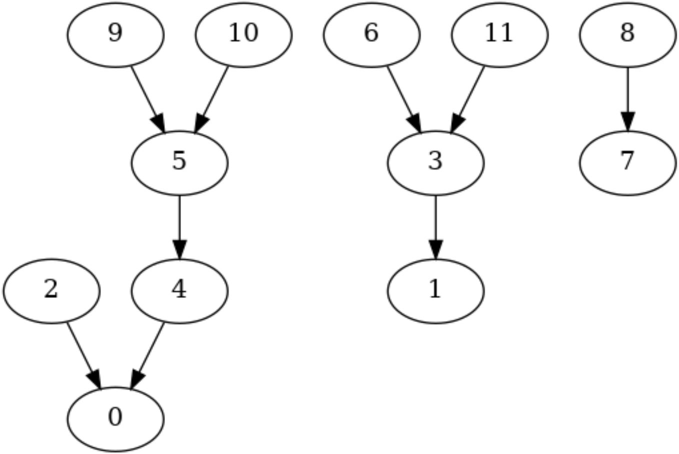
- Gerum ráð fyrir að tölurnar sem við munum vinna með séu jákvæðar og minni en n.
- Við munum þá gefa okkur n staka fylki p, þar sem i-ta stakið í fylkinu er upphafstillt sem i.
- Fylkið p mun nú geyma *foreldri* sérhvers stak.
- Foreldrin myndi keðjur.
- Sérhver keðja endar í einhverju staki, sem við 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;
    }
    else
    {
        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 glæruni gefur til kynna er hugmyndin að þjappa keðjunum saman í hvert skipti sem kallað er á `find`.

Keðjuþjöppuð sammengisleit

- Eins og nafnið á glæruni 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

```c
int p[MAX];

int find(int x)
{
    if (p[x] == x)
    {
        return x;
    }
    else
    {
        p[x] = find(p[x]);  //keðjuþjöppunin á sér stað í þessar línu
        return 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.

Efnisyfirlit

Forskot á sæluna

- Forgangsbiðraðir eru notaðar í reiknirit Dijkstra.

Forskot á sæluna

- Forgangsbiðraðir eru notaðar í reiknirit Dijkstra.
- Sammengisleit er notuð í reiknirit Kruskals.

Forskot á sæluna

- Forgangsbiðraðir eru notaðar í reiknirit Dijkstra.
- Sammengisleit er notuð í reiknirit Kruskals.
- Við munum nota hrúgur aftur í næstu viku.