Reiknirit Tarjans

Bergur Snorrason

3. mars 2021

- Skilgreinum vensl $\sim$ á milli hnúta í óstefndu neti með því að $u \sim v$ ef og aðeins ef til er vegur milli u og v.
- Auðvelt er að sýna að þetta eru jafngildisvensl (sjálfhverf, samhverf og gegnvirk).
- Við megum því skilgreina *samhengisþátt* í netinu sem jafngildisflokka þessara vensla.
- Samhengisþáttur í neti er því óstækkanlegt mengi þannig að komast má hverjum hnút í menginu til hvers annars með vegi.
- Segja má að net sé samanhangandi þá og því aðeins að það innihaldi einn samhengisþátt.

- Við getum beitt dýptarleit eða breiddarleit til að finna alla hnúta sem eru í sama samhengisþætti og tiltekinn hnútur.
- Ef við pössum að hefja bara leit einu sinni fyrir hvern samhengisþátt þá getum við fundið alla samhengisþætti í $\mathcal{O}(E + V)$ tíma.

```
vi v;
void dfs(vvi& g, int x, int c)
{
    int i;
    v[x] = c;
    rep(i, g[x].size()) if (v[g[x][i]] == -1) dfs(g, g[x][i], c);
}
```

```
    v = vi(n, -1);
    rep(i, n) if (v[i] == -1) dfs(g, i, c++);
```

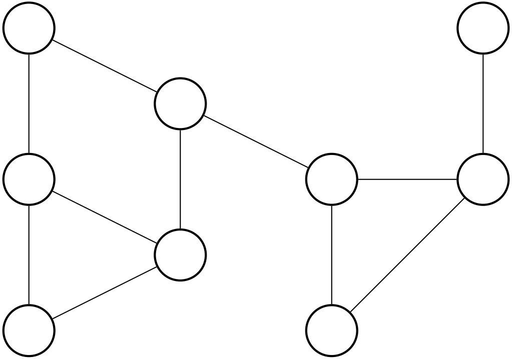
```cpp
#include <bits/stdc++.h>
#define rep(E, F) for (E = 0; E < (F); E++)
using namespace std;
typedef vector<int> vi;
typedef vector<vi> vvi;

vi v;
void dfs(vvi& g, int x, int c)
{
    int i;
    v[x] = c;
    rep(i, g[x].size()) if (v[g[x][i]] == -1) dfs(g, g[x][i], c);
}

// Fyrsta lína inntaksins eru tvær heiltölur, fjöldi hnúta og fjöldi leggja.
// Síðan koma m línur sem svara til leggjalistans.
int main()
{
    int i, j, n, m, c = 0, x , y;
    cin >> n >> m;
    vvi g(n);
    rep(i, m)
    {
        cin >> x >> y;
        x--, y--;
        g[x].push_back(y);
        g[y].push_back(x);
    }
    v = vi(n, -1);
    rep(i, n) if (v[i] == -1) dfs(g, i, c++);
    printf("Fjoldi samhengisthatta er %d.\n", c);
    rep(i, n) printf("Hnutur %d er i samhengisthaetti %d.\n", i + 1, v[i] + 1);
    return 0;
}
```

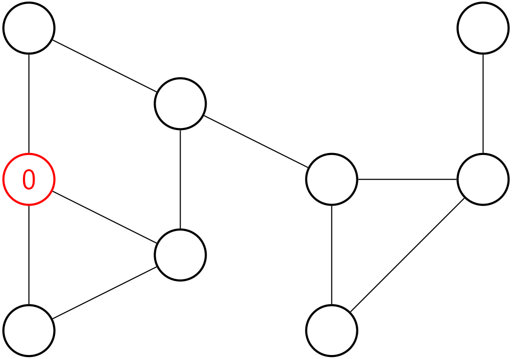
- Ef við tölum um að fjarlægja hnút úr neti þá er átt við að hnúturinn ásamt öllum leggjum til og frá honum eru fjarlægðir.
- Gerum ráð fyrir að við höfum net G og látum G_u tákna netið þar sem hnútur u hefur verið fjarlægður.
- Við segjum að hnútur u sé *liðhnútur* (e. *articulation point*) ef G hefur færri samhengisþætti en G_u.
- Til að fjarlægja legg úr neti nægir að fjarlægja legginn.
- Táknum þá netið G án leggsins e með G_e.
- Leggur e eru sagður vera *brú* (e. *bridge*) ef G hefur færri samhengisþætti en G_e.
- Með öðrum orðum er hnútur u liðhnútur (leggur e brú) ef til eru hnútar v_1 og v_2 í sama samhengisþætti þannig að allir vegir frá v_1 til v_2 fari í gegnum hnútinn u (legginn e).

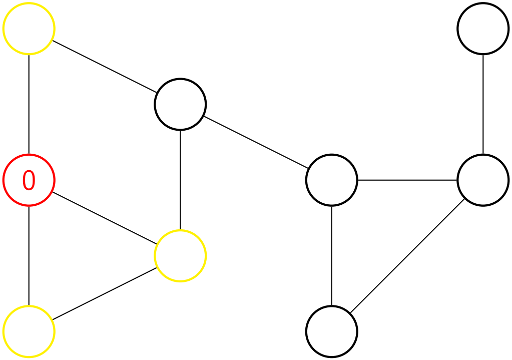
- Ein leið til að finna alla liðhnúta er að telja fyrst samhengisþætti netsins, fjarlægja hnút, telja samhengisþætti og endurtaka fyrir alla hnúta.
- Þar sem við þurfum að finna alla samhengisþætti $V + 1$ neta er þessi aðferð með tímaflækju $\mathcal{O}(V^2 + VE)$.
- Samskonar aðferð til að finna brýr væri með tímaflækju $\mathcal{O}(E^2 + VE)$.
- Þetta er þó ekki æskilegt, því getum við getum fundið bæði alla liðhnúta og allar brýr með einni dýptarleit.

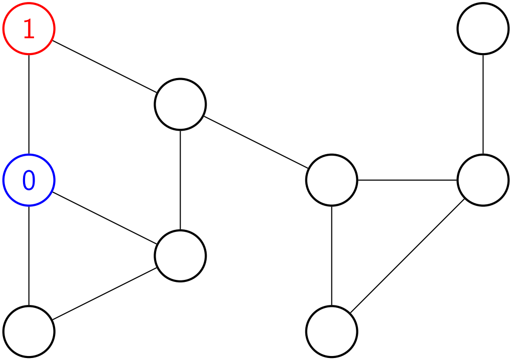
- Gerum ráð fyrir að netið okkar sé samanhangandi.
- Ef svo er ekki getum við beitt þessari aðferð á hvern samhengisþátt.
- Veljum einhvern hnút og framkvæmum dýptarleit frá honum.
- Skilgreinum svo tvær breytur fyrir hvern hnút u út frá þessari dýptarleit, u_{low} og u_{num}.
- Talan u_{num} segir hversu mörg skref í leitin við tókum til að finna hnútinn u.
- Talan u_{low} er minnsta gildið v_{num} þar sem v er hnútur sem við við getum ferðast til án þess að nota leggi sem hafa verið notaðir í leitinni.

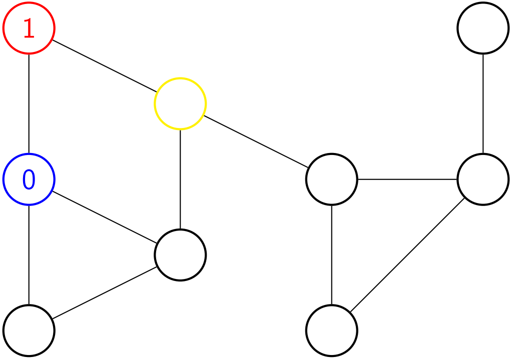
- Gerum ráð fyrir að leitin okkar fari úr hnút u í hnút v með legg e.
- Ef $v_{low} > u_{num}$ þá er e brú.
- Þetta þýðir að eina leiðin frá v til u er í gegnum legginn e.
- Ef $v_{low} \geq u_{num}$ þá er u liðhnútur.
- Þetta þýðir að eina leiðin frá v í fyrri hnúta leitarinnar er í gegnum hnútinn u.

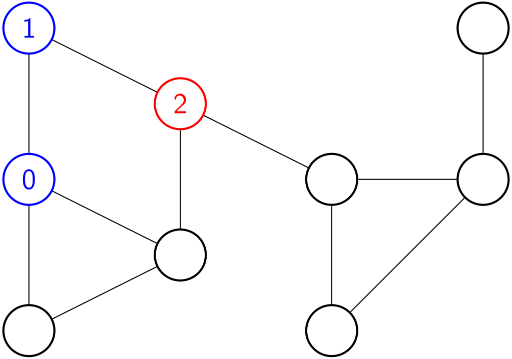



0

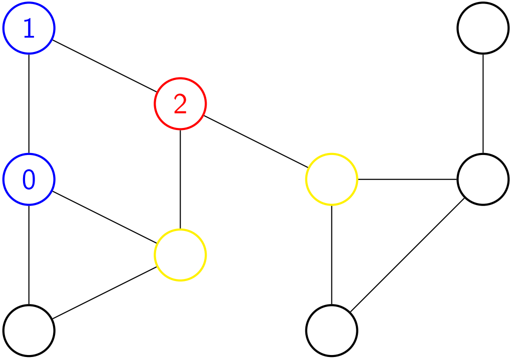

0


1
0

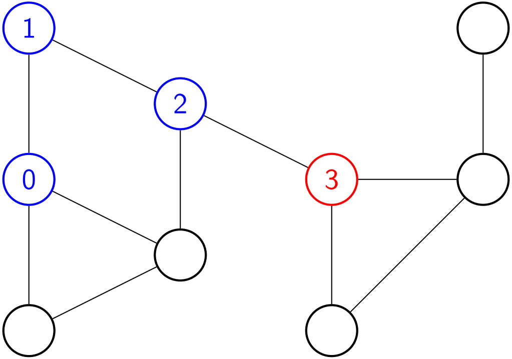

1
0

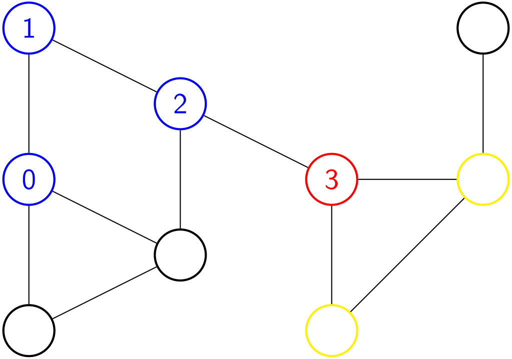


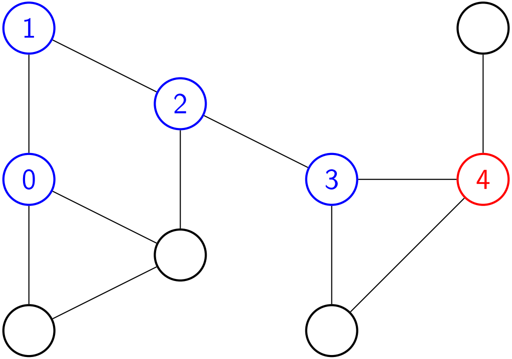
1
2
0

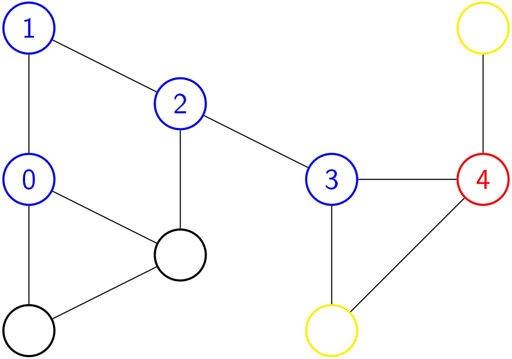


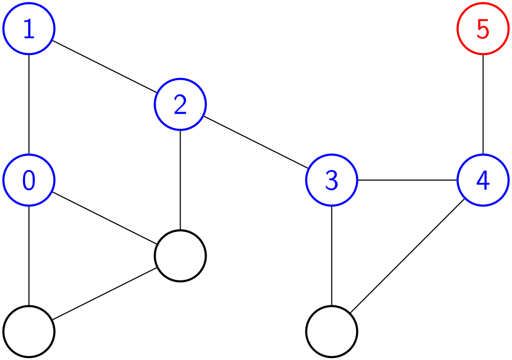
1
2
0

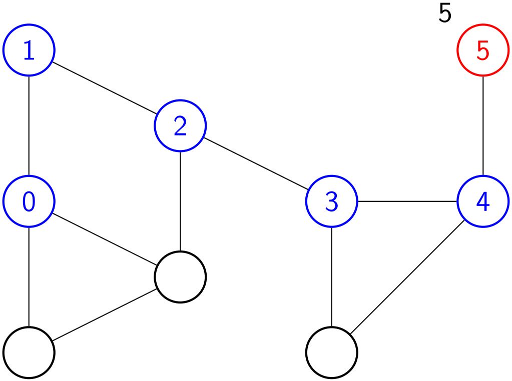

1
2
0
3


1
2
0
3

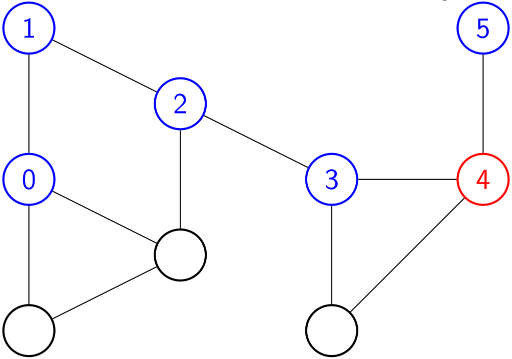

1
2
0
3
4


1
2
0
3
4

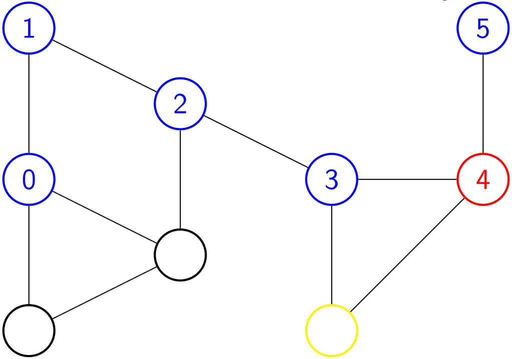

1
5
2
0
3
4

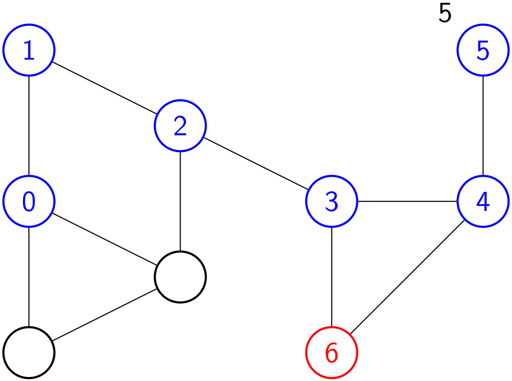

5
1
5
2
0
3
4

5

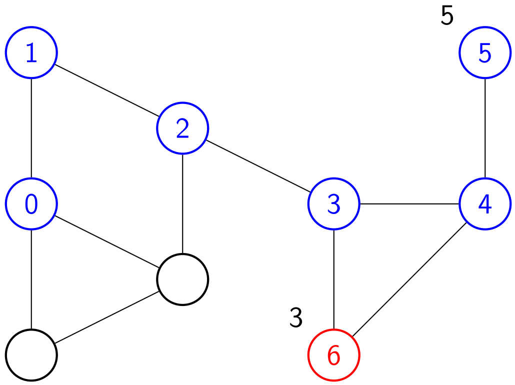


5

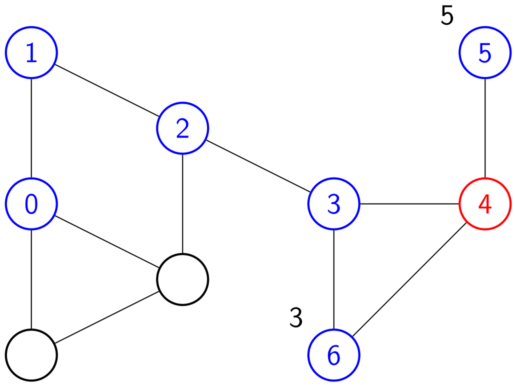




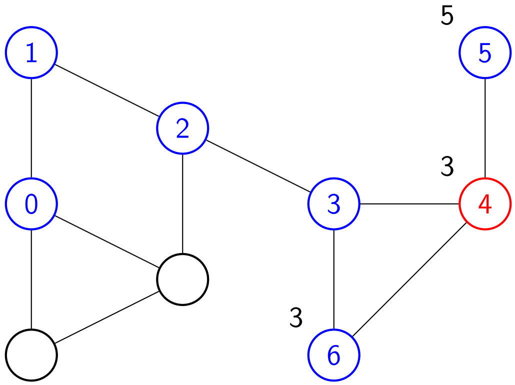
5
1
5
2
0
3
4
6

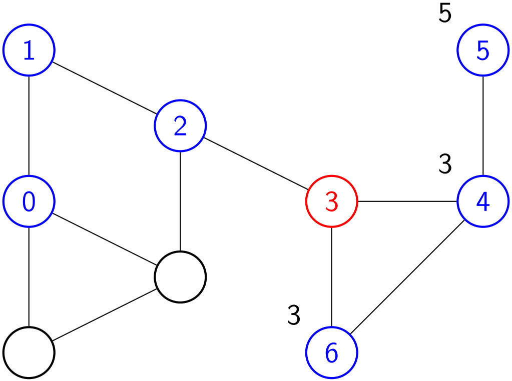


5
1
5
2
0
3
4
3
6

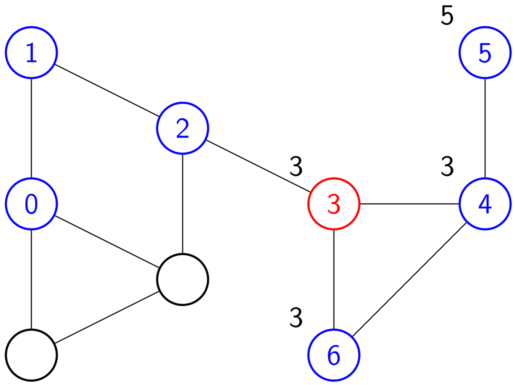


5
1
5
2
0
3
4
3
6

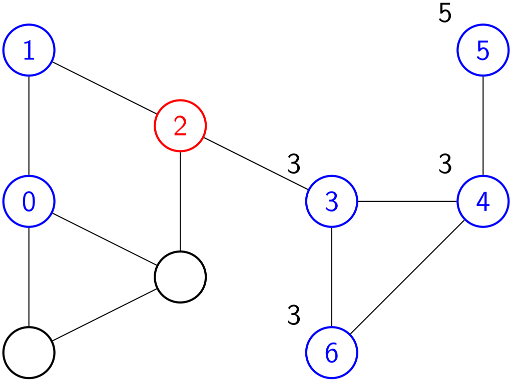

5
1
5
2
3
0
3
4
3
6



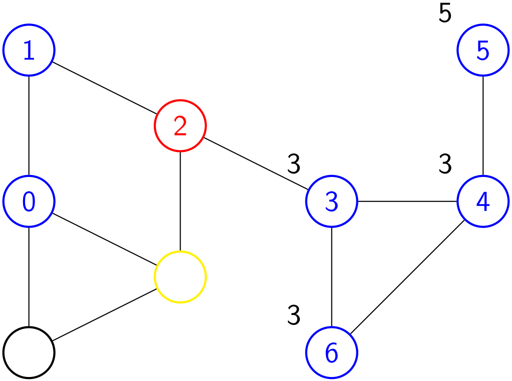
5
1
5
2
3
0
3
4
3
6

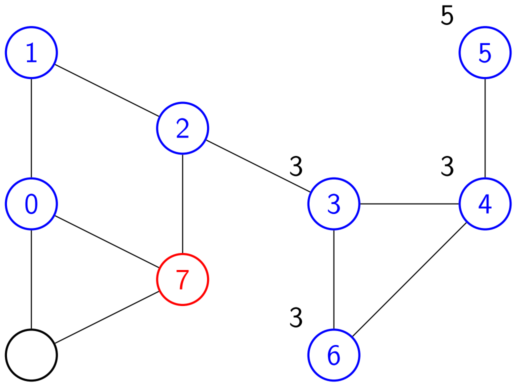


5
1
5
2
3
3
3
4
0
3
6

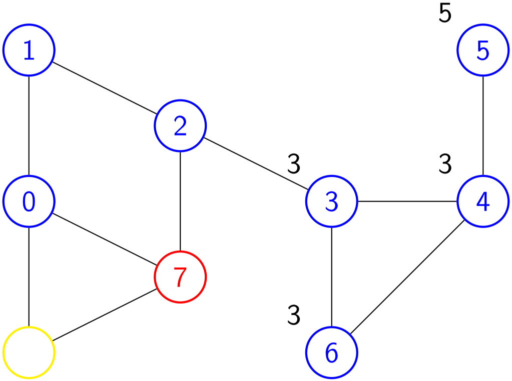


1
2
0
3
4
5
6
5
3
3
3

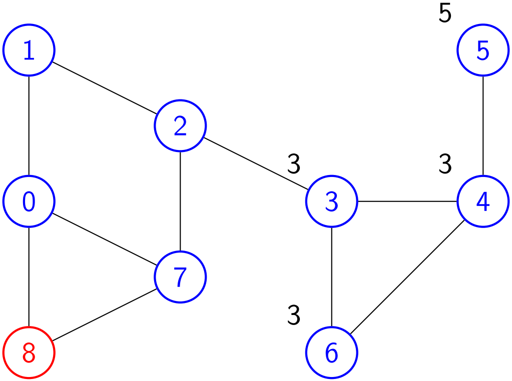

5
1
5
2
3
3
0
3
4
3
6



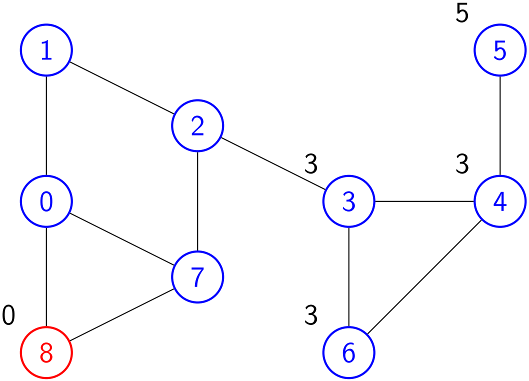
5
1
5
2
3
3
0
3
4
7
3
6



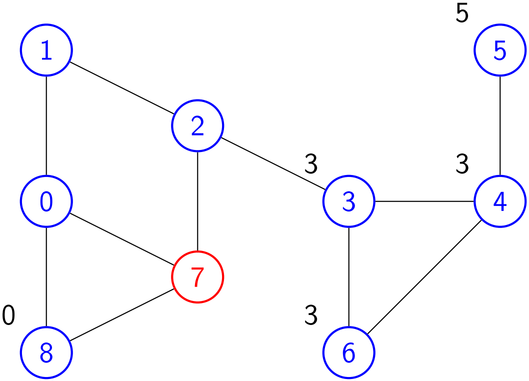
5
1
5
2
3
3
0
3
4
7
3
6



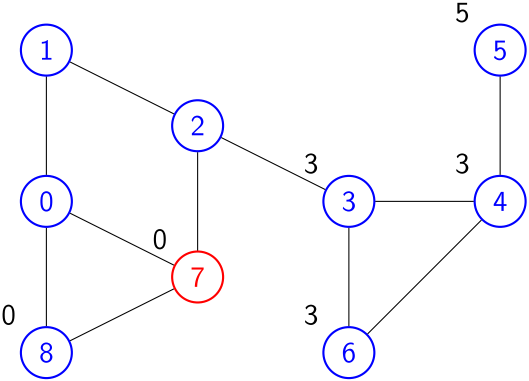
5
1
5
2
3
3
0
3
4
7
3
8
6



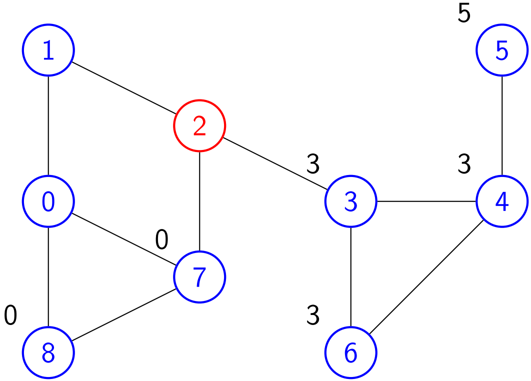
1
2
3
4
5
0
7
8
6
5
3
3
0
3

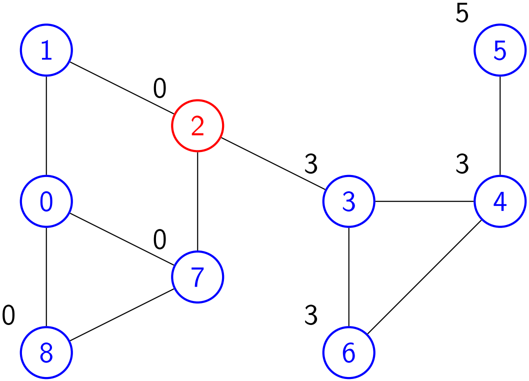


5
1
5
2
3
3
0
3
4
7
0
3
8
6

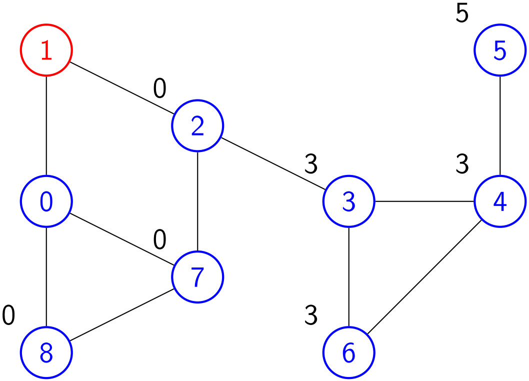


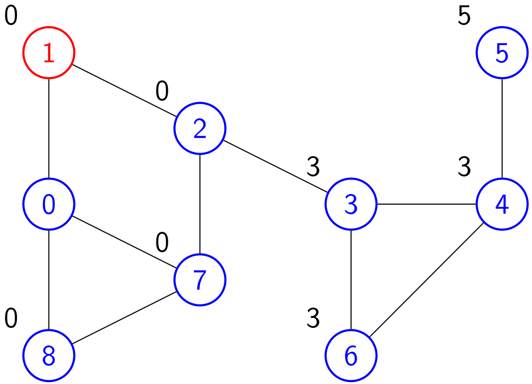
5
1
5
2
3
3
0
3
4
0
7
0
3
8
6

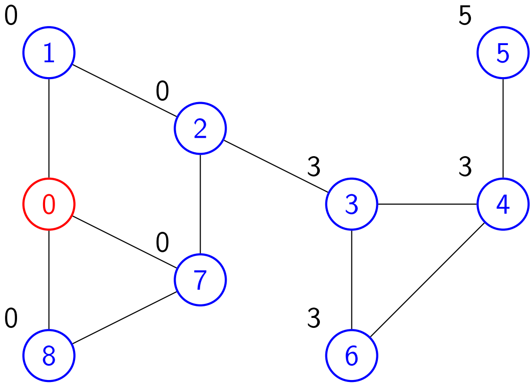

5
1
5
2
3
3
0
3
4
0
7
0
3
8
6

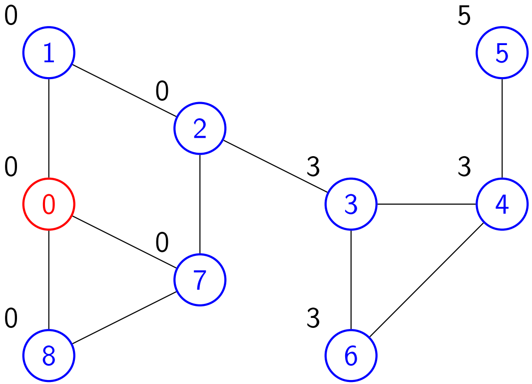


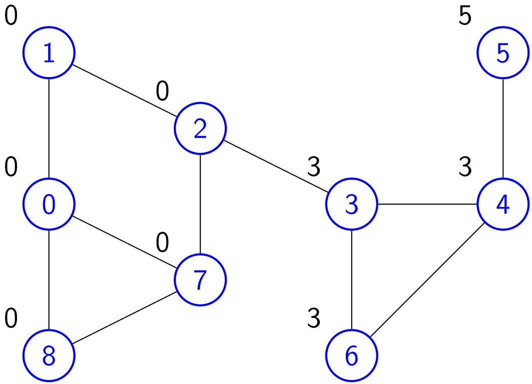
5
1
0
2
3
3
5
4
0
0
7
0
3
8
6


1
2
0
3
4
5
7
8
6
0
3
3
5
0
0
3

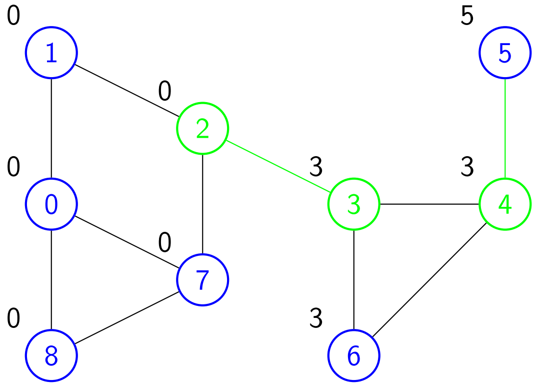

0
1
0
2
3
3
5
5
3
4
0
0
7
0
8
3
6


0
1
5
5
0
2
3
3
3
4
0
0
7
3
0
8
3
6


0
1
0
2
0
0
3
3
0
5
5
4
7
3
8
6



0
1
5
5
0
2
0
0
3
3
3
4
0
7
0
8
3
6

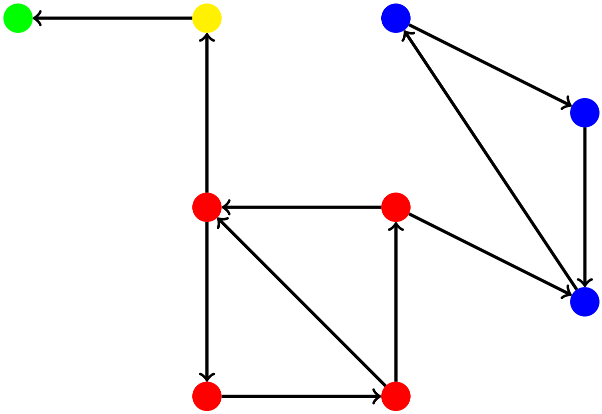

0
1
0
2
3
3
5
5
0
0
3
4
0
7
0
8
3
6

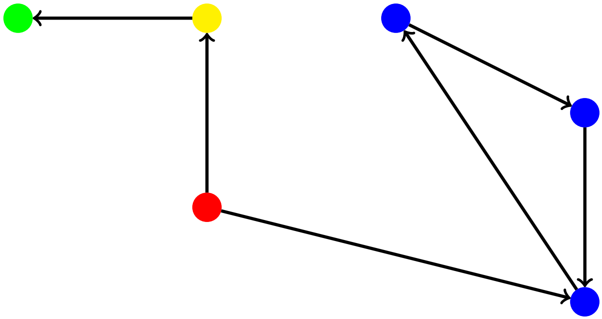
```
int low[MAXN], num[MAXN], curnum;
vi cp; vii bri;
void dfs(const vvi &g, int u, int p)
{
    low[u] = num[u] = curnum++;
    int i, cnt = 0, f = 0;
    rep(i, g[u].size())
    {
        int v = g[u][i];
        if (num[v] == -1)
        {
            dfs(g, v, u);
            low[u] = min(low[u], low[v]);
            cnt++;
            f = f || low[v] >= num[u];
            if (low[v] > num[u]) bri.push_back(ii(u, v));
        }
        else if (p != v) low[u] = min(low[u], num[v]);
    }
    if (f && (p != -1 || cnt > 1)) cp.push_back(u);
}

void cpb(const vvi &g)
{
    int i, n = g.size();
    memset(num, -1, n << 2);
    curnum = 0;
    rep(i, n) if (num[i] == -1) dfs(g, i, -1);
}
```

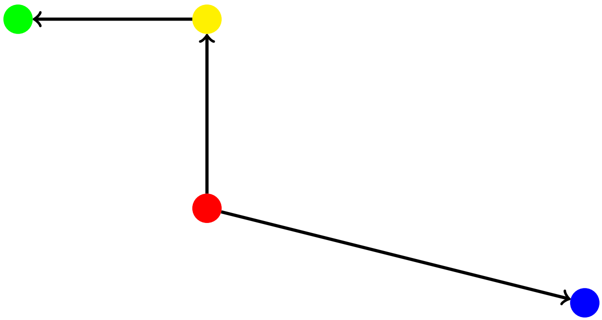
- Tímaflækjan er $\mathcal{O}(E+V)$ því það er tímaflækja dýptarleitar.

- Gerum ráð fyrir að við séum með stefnt net.
- Þá eru venslin sem við skilgreindum áðan ekki lengur jafngildisvensl því þau eru ekki samhverf.
- Við getum þó gert þau samhverf með því að krefjast að það sé til vegur í báðar áttir.
- Með öðrum orðum er $x \sim y$ ef og aðeins ef til er vegur frá u til v og vegur frá v til u.
- Jafngildisflokkar þessara vensla eru kallaðir *strangir samhengisþættir* (e. *strong connected components*).
- Ég mun þó leyfa mér að kalla þetta *samhengisþætti* þegar ljóst er að við séum að ræða um stenft net.

- Takið eftir að ef netið inniheldur rás þá eru allir hnútar í rásinni í sama samhengisþætti.
- Einnig gildir að ef netið inniheldur enga rás er hver hnútur sinn eigin samhengisþáttur.
- Slík net kallast *stefnd órásuð net* (e. *directed acycle graphs (DAG)*).
- Þau hafa ýmsa þæginlega eiginleik, til dæmis má beyta kvikri bestun á þau.
- Við getum breytt stefndu neti í órásað stefnt net með því að deila út jafngildisvenslunum.
- Nánar, þá lítum við svo á að hnútar í sama samhengisþætti séu í raun sami hnúturinn og verður leggur milli samhengisþátta ef vegur liggur milli einhverja hnúta í samhengisþáttunum sem fer ekki í annan samhengisþátt.
- Við köllum þetta net *herpingu* (e. *contraction*) upprunalega netsins.







- Til að finna herpinguna þurfum við fyrst að finna samhengisþættina.
- Við getum breytt lítilega forritinu sem við vorum með áðan til að finna samhengisþætti stefnds nets.
- Við getum skoðað hvort $u_{low} = u_{num}$ á leiðinni upp úr endurkvæmninni.
- Ef svo er þá er u fyrsti hnúturinn sem við sáum í samhengisþættinum sem u tilheyrir.
- Við geymum því hnútana sem við heimsækjum á hlaða.
- Þegar við finnum umrætt u (á leiðinni upp úr endurkvæmninni) tínum við af hlaðanum þangað til við sjáum u og setjum alla þá hnúta saman í samhengisþátt.

```
int low[MAXN], num[MAXN], curnum, cnt;
vi cp, st, vis, ans;
void dfs(vvi &g, int u, int p)
{
    low[u] = num[u] = curnum++;
    st.push_back(u);
    vis[u] = 1;
    int i;
    rep(i, g[u].size())
    {
        int v = g[u][i];
        if (num[v] == -1) dfs(g, v, u);
        if (vis[v]) low[u] = min(low[u], low[v]);
    }
    if (low[u] == num[u])
    {
        while (1)
        {
            int v = st.back(); st.pop_back();
            vis[v] = 0; ans[v] = cnt;
            if (u == v) break;
        }
        cnt++;
    }
}

void scc(vvi &g)
{
    int i, n = g.size();
    vis = vi(n);
    ans = vi(n, -1);
    memset(num, -1, n << 2);
    curnum = cnt = 0;
    rep(i, n) if (num[i] == -1) dfs(g, i, -1);
}
```

- Þar sem við leitum bara einu sinni í netinu með dýptarleit fæst að þetta reiknirit er $\mathcal{O}(E + V)$.
- Við köllum þetta reiknrit, ásamt því sem finnur liðhnúta og brýr, reiknrit Tarjans.