

Inngangur að netafræði

Bergur Snorrason

3. mars 2021

Net

- ▶ Tvennd (V, E) , þar sem V er endanlegt mengi og $E \subset V \times V$, kallast *net* (e. *graph*).
- ▶ Stökin í V köllum við *hnúta* (e. *node*) og stökin í E köllum við *leggi* (e. *edge*).
- ▶ Ef venslin E eru samhverf, það er að segja ef

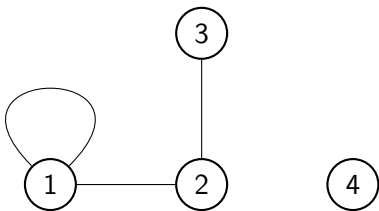
$$(u, v) \in E \Rightarrow (v, u) \in E,$$

þá segjum við að netið sé *óstefnt* (e. *undirected*).

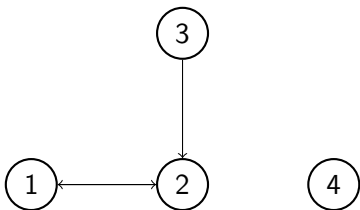
- ▶ Net sem er ekki óstefnt kallast *stefnt* (e. *directed*).
- ▶ Við segjum að hnúturinn v sé *nágranni* (e. *neighbour*) hnútsins u ef (u, v) er í E .
- ▶ Við segjum að hnútar u og v í óstefndu neti séu *nágrannar* ef (u, v) er í E .
- ▶ Við segjum einnig að það liggi *leggur* á milli u og v .

- ▶ Takið eftir að netið $G = (V, E)$ hefur $|V|$ fjölda hnúta og $|E|$ fjölda leggja (ögn flóknara ef netið er óstefnt).
- ▶ Tímaflækjur reinkirita í netafræði eru því iðulega gefnar sem föll af $|V|$ og $|E|$ (hingað til höfum við aðalega notað n).
- ▶ Þegar ég forrita netafræði dæmi læt ég yfirleitt n tákna fjölda hnúta og m tákna fjölda leggja.
- ▶ Því kemur fyrir að ég lýsa tímaflækjum reikniritana með þessum breytum.
- ▶ Ég mun einnig leyfa mér að nota V í stað $|V|$ og E í stað $|E|$, því þetta ætti aldrei valda ruglningi.
- ▶ Sem dæmi skrifa ég $\mathcal{O}(E + V)$ í stað $\mathcal{O}(|E| + |V|)$.

- ▶ Oft hjálpar að skilja tiltekið net með því að teikna það.
- ▶ Við byrjum á að teikna punkta fyrir hnútana.
- ▶ Ef netið er óstefnt teiknum við svo línu á milli nágranna (svo hver lína svarar til leggs).
- ▶ Ef netið er stefnt þá teiknum við ör í stað línu.
- ▶ Leggur (u, v) er þá táknaður með ör frá hnút u til hnúts v .



- Hér má sjá teikningu sem svarar til $V = \{1, 2, 3, 4\}$ og $E = \{(1, 1), (1, 2), (2, 1), (2, 3), (3, 2)\}$.



- Hér má sjá teikningu sem svarar til $V = \{1, 2, 3, 4\}$ og $E = \{(1, 2), (2, 1), (3, 2)\}$.

- ▶ Leggir af gerðinni (u, u) kallast *lykkjur* (e. *loop*) (ástæða nafngiftarinnar sést af fyrri myndinni).
- ▶ Net án leggja kallast *einfalt* (e. *simple*).
- ▶ Í umfjöllun okkar gerum við ráð fyrir að öll net séu einföld nema annað sé tekið fram.
- ▶ Yfirleitt eru net skilgreind á veg sem leyfa fleiri en einn legg milli hnúta.
- ▶ Einföld net þurfa þá líka að hafa í mesta lagi einn legg milli hnúta.

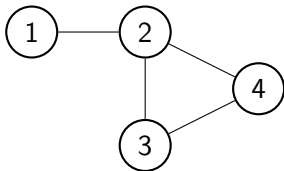
- ▶ Runa hnúta v_1, v_2, \dots, v_n kallast *vegur* (e. *path*) ef $(v_j, v_{j+1}) \in E$, fyrir $j = 1, 2, \dots, n - 1$.
- ▶ Vegur kallast *rás* (e. *cycle*) ef $v_1 = v_n$.
- ▶ Vegur kallast *einfaldur* (e. *simple*) ef engir tveir hnútar í v_1, v_2, \dots, v_n eru eins.
- ▶ Rás kallast *einföld* (e. *simple*) ef engir tveir hnútar í v_1, v_2, \dots, v_{n-1} eru eins.
- ▶ Við segjum að vegurinn v_1, v_2, \dots, v_n *liggi á milli hnútanna* v_1 og v_n .
- ▶ Óstefnt net er sagt vera *samanhangandi* (e. *connected*) ef til er vegur milli sérhverja tveggja hnúta.
- ▶ Óstefnt net er sagt vera *tré* (e. *tree*) ef það er samanhgandi og inniheldur enga rás.

Framsetning neta í tölvum

- ▶ Það er engin stöðluð leið til að geyma net í minni.
- ▶ Yfirleitt er notað eina af þremur gagnagrindum:
 - ▶ Leggjalista.
 - ▶ Nágrannafylki.
 - ▶ Nágrannalista (algengust).

Leggjalisti

- ▶ Látum $G = (V, E)$ tákna netið okkar.
- ▶ Þar sem V er endanlegt megum við gera ráð fyrir að $V = \{1, 2, \dots, n\}$, þar sem n er fjöldi hnúta í G .
- ▶ Látum m vera fjölda leggja í G .
- ▶ Listi af tvenndum sem inniheldur nákvæmlega sömu stök og E kallast *leggjalisti* netsins G .
- ▶ Við notum leggjalist sjaldan, en það kemur fyrir (til dæmis í reikniriti Kruskals).
- ▶ Net í dæmum í keppnisforritun eru þó oftast gefin með leggja lista.
- ▶ Í óstefndum netum er hver leggur tvítekinn í E og við leyfum okkur að sleppa annari endurtekningunni í listanum.

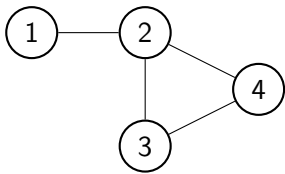


$L = [$
 $(1, 2),$
 $(2, 3),$
 $(2, 4),$
 $(3, 4)$
 $]$

- ▶ Helsti galli leggjalistans er að það tekur $\mathcal{O}(m)$ tíma að ákvarða hvort hnútar séu nágrannar eða finna nágranna tiltekins hnúts.

Nágrannafylki

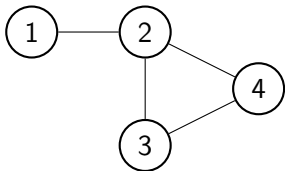
- ▶ Láttum A vera $n \times n$ fylki þannig að $A_{uv} = 1$ ef (u, v) er í E , en $A_{uv} = 0$ annars.
- ▶ Við köllum A *nágrannafylki* netsins G .
- ▶ Takið eftir að það tekur $\mathcal{O}(n^2)$ tíma að upphafsstillast A .
- ▶ Svo þessi aðferð er alferið of hæg ef, til dæmis, $n = 10^5$ (sem er oft raunin).
- ▶ Þegar n er nógu lítið eru nágrannafylki nytsamleg því við getum ákvarðað hvort tveir hnútar séu nágrannar í $\mathcal{O}(1)$ tíma.
- ▶ Einnig hefur A^p (fylkjamargföldun) hefur einnig áhugaverða talningarfræðilega merkingu sem við skoðum síðar.



$$A = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{pmatrix}$$

Nágrannalistar

- ▶ Látum nú V tákna lista af n listum.
- ▶ Táknum j -ta lista V með V_j .
- ▶ Látum nú V_u innihalda alla nágranna hnútsins u í netinu G , án endurtekningar.
- ▶ Við köllum V *nágrannalista* (fleirtölu) netsins G og V_u *nágrannalista* (eintölu) hnútsins u í netinu G .
- ▶ Helsti kostur nágrannalistanna er að við getum skoðað alla nágranna tiltekins hnúts án þess að skoða neitt annað.
- ▶ Við getum því ítrað í gegnum alla nágranna allra hnúta í $\mathcal{O}(E + V)$ tíma, óháð röð hnútanna.
- ▶ Þetta kemur að góðum notum þegar við erum að ferðast í gegnum netið.



$L = [$
 $[2]$
 $[1, 3, 4]$
 $[2, 4]$
 $[2, 3]$
 $]$

- ▶ Eins og minnst var á áðan eru net oftast gefin með leggjalista, en við vinnum yfirleitt með nágrannalista.
- ▶ Til að breyta á milli látum nágrannalistann okkar vera af tagi `vector<vector<int>>`.
- ▶ Við upphafsstillum hann með n tómum listum.
- ▶ Við lesum svo í gegnum alla leggina og bætum viðeigandi hnútum í tilheyrandi nágrannalista.
- ▶ Ef leggur (u, v) er í leggjalista stefnds nets þá bætum við v við V_u .
- ▶ Ef leggur (u, v) er í leggjalista óstefnds nets þá bætum við v við V_u og u við V_v .

```

1 #include <bits/stdc++.h>
2 #define rep(E, F) for (E = 0; E < (F); E++)
3 using namespace std;
4 typedef vector<int> vi;
5 typedef vector<vi> vvi;
6
7 // Fyrsta lína inntaksins eru tvær heiltölur, fjöldi nóða og fjöldi leggja.
8 // Síðan koma m línur sem svara til leggjalistans.
9 int main()
10 {
11     int i, j, n, m, x, y;
12     cin >> n >> m;
13     vvi g(n);
14     rep(i, m)
15     {
16         cin >> x >> y;
17         x--, y--;
18         g[x].push_back(y);
19         g[y].push_back(x); // Sleppa þesari línu ef netið er stefnt.
20     }
21     rep(i, n)
22     {
23         printf("%d: ", i + 1);
24         rep(j, g[i].size()) printf("%d ", g[i][j] + 1);
25         printf("\n");
26     }
27     return 0;
28 }

```

Vegin net

- ▶ Við segjum að þrenndin (V, E, w) , þar sem (V, E) er net og $w: E \rightarrow \mathbb{R} \cup \{-\infty, \infty\}$, sé *vegið net* (e. *weighted graph*).
- ▶ Það sem við erum í rauninni að gera er að gefa hverjum legg í netinu vigt.
- ▶ Ef við erum, til dæmis, með net sem lýsir gönguleiðum í Mosfellsbæ þá gætu vigtirnar verið lengd hvers leggs gönguleiðanna eða tíminn sem það tekur að labba leggin.
- ▶ Þegar við teiknum vegin net þá teiknum við netið líkt og áður og bætum vigtunum við hliðina á tilheyrandi leggjum.

Vegin net

- ▶ Þegar við geymum vegin net í einhverri gagnagrind þá bætum við yfirleitt bara vigtinni við.
- ▶ Leggjalisti vegins nets er því listi af þrenndum.
- ▶ Þrennd (u, v, w) þýðir þá að það liggi leggur frá hnút u til hnúts v með vigt w .
- ▶ Nágrannafylki vegins nets er gefið með $A_{uv} = \infty$ ef enginn leggur liggur á frá hnút u til hnúts v og $A_{uv} = w(e)$ er leggurinn e liggur frá hnút u til hnúts v .
- ▶ Nágrannalistar vegins nets eru listar af tvenndum, fyrra stak tvenndarinnar er nágranninn og seinna stakið er vigtin á leggnum til nágrannans.
- ▶ Nágrannalistarnir eru því geymdir með `vector<vector<pair<int, int>>>`.
- ▶ Við notum `typedef` til að stytta þetta niður í `vvi`.

```

1 #include <bits/stdc++.h>
2 #define rep(E, F) for (E = 0; E < (F); E++)
3 using namespace std;
4 typedef pair<int, int> ii;
5 typedef vector<ii> vii;
6 typedef vector<vii> vvii;
7
8 // Fyrsta lína inntaksins eru tvær heiltölur, fjöldi hnút og fjöldi leggja.
9 // Síðan koma m línur sem svara til leggjalistans.
10 int main()
11 {
12     int i, j, n, m, x, y, w;
13     cin >> n >> m;
14     vvii g(n);
15     rep(i, m)
16     {
17         cin >> x >> y >> w;
18         x--, y--;
19         g[x].push_back(ii(y, w));
20         g[y].push_back(ii(x, w)); // Sleppa þesari línu ef netið er stefnt.
21     }
22     printf("Nagrannar:\n");
23     rep(i, n)
24     {
25         printf("%d: ", i + 1);
26         rep(j, g[i].size()) printf("%d ", g[i][j].first + 1);
27         printf("\n");
28     }
29     printf("Vigtir:\n");
30     rep(i, n)
31     {
32         printf("%d: ", i + 1);
33         rep(j, g[i].size()) printf("%d ", g[i][j].second);
34         printf("\n");
35     }
36     return 0;
37 }

```

