Reiknirit Bellmans og Fords (1958 og 1956)

Bergur Snorrason

March 4, 2024

- Hvað gerum við ef við viljum nota reinkirit Dijkstras en það mega vera neikvæðar vigtir á leggjunum.
- Við getum þá notað reiknirit sem er kennt við Bellman og Ford.
- Við þurfum þó að fórna keyrslutíma.

- Þetta reiknirit er að vissu leiti einfaldara en reiknirit Dijkstras.
- Við notum kvika bestun og svörum spurningunni „Hver er stysta leiðin frá u til v sem fer að mestu í k hnúta?".
- Hér táknar u upphafshnútinn á meðan v og k eru frjálsar breytur.
- Látum þá $f(v,k)$ tákna systa veg frá hnútnum u til hnútsins v sem fer ekki í fleiri en k hnúta.
- Til að einfalda skriftir þá skilgreinum við

$$E_u = \{v \in V : (u,v) \in E\}$$

og

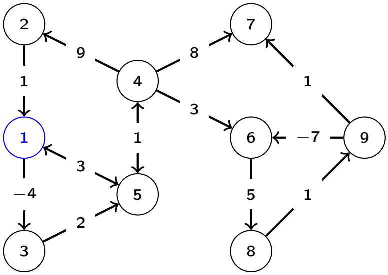
$$E^v = \{u \in V : (u,v) \in E\}.$$

- Við fáum að

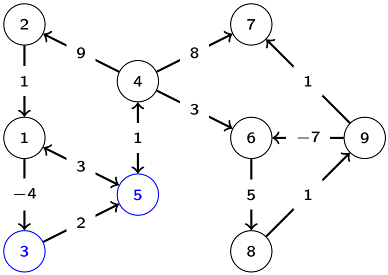
$$f(v,k) = \begin{cases} 0, & \text{ef } u = v \text{ og } k = 0 \\ \infty, & \text{ef } u \neq v \text{ og } k = 0 \\ \min(f(v,k-1), & \\ \ \min_{u \in E^v} w((u,v)) + f(u,k-1)), & \text{annars.} \end{cases}$$

- Við munum leysa þetta með neðansækinni kvikri bestun.
- Gerum ráð fyrir að taflan sem við notum fyrir minnun hafi dálk sem svari til k breytunnar.
- Þá er hver staða aðeins háð stöðum í röðinni fyrir ofan sig.
- Við notum því aðeins síðustu línu fylkisins þegar við fyllum inn í töfluna.
- Því má geyma tvívíða fylkið sem einvítt fylki.

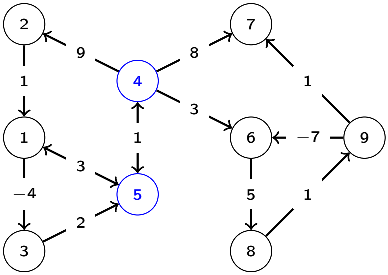
- Við erum ekki búin þegar við höfum reiknað öll gildin á $f(v, k)$.
- Hvað með neikvæðar rásir?
- Takið fyrst eftir að ef það er ekki neikvæð rás í netinu þá heimsækir systi vegur milli hnúta engan hnúta tvisvar.
- Einnig er ekki nóg að það sé neikvæð rás í netinu heldur þarf að vera hægt að komast í hana frá upphafshnútnum og svo má vera að það sé ekki hægt að komast frá rásinni í alla aðra hnúta.
- Við getum einfaldlega prófað að lengja vegina um $V - 1$ hnúta í viðbót.
- Ef vegalengdin styttist einhverntíman þá er betra að heimsækja einhvern hnút oftar en einu sinni, sem þýðir að það sé neikvæð rás á leiðinni.



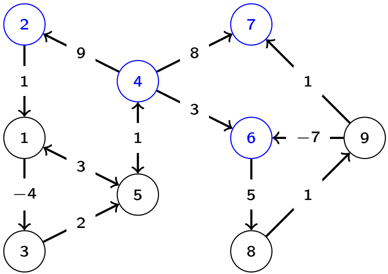
k	1	2	3	4	5	6	7	8	9
0	0	∞	∞	∞	∞	∞	∞	∞	∞



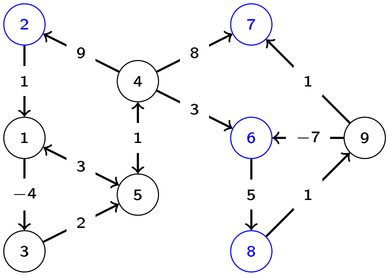
k	1	2	3	4	5	6	7	8	9
0	0	∞	∞	∞	∞	∞	∞	∞	∞
1	0	∞	-4	∞	3	∞	∞	∞	∞



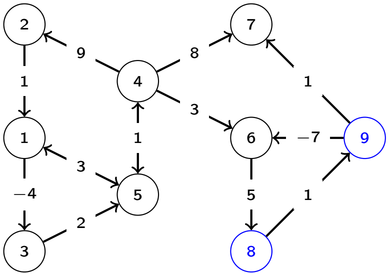
k	1	2	3	4	5	6	7	8	9
0	0	∞	∞	∞	∞	∞	∞	∞	∞
1	0	∞	-4	∞	3	∞	∞	∞	∞
2	0	∞	-4	4	-2	∞	∞	∞	∞



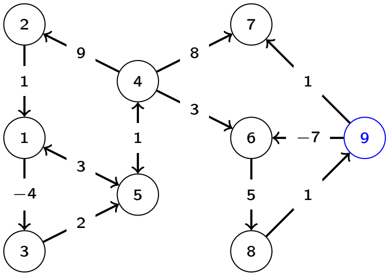
k	1	2	3	4	5	6	7	8	9
0	0	∞	∞	∞	∞	∞	∞	∞	∞
1	0	∞	-4	∞	3	∞	∞	∞	∞
2	0	∞	-4	4	-2	∞	∞	∞	∞
3	0	13	-4	-1	-2	7	12	∞	∞



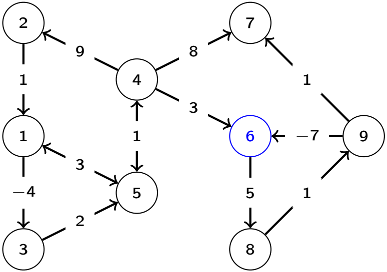
k	1	2	3	4	5	6	7	8	9
0	0	∞	∞	∞	∞	∞	∞	∞	∞
1	0	∞	-4	∞	3	∞	∞	∞	∞
2	0	∞	-4	4	-2	∞	∞	∞	∞
3	0	13	-4	-1	-2	7	12	∞	∞
4	0	8	-4	-1	-2	2	7	12	∞



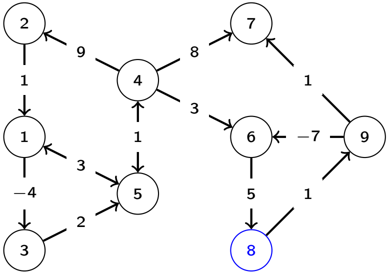
k	1	2	3	4	5	6	7	8	9
0	0	∞	∞	∞	∞	∞	∞	∞	∞
1	0	∞	-4	∞	3	∞	∞	∞	∞
2	0	∞	-4	4	-2	∞	∞	∞	∞
3	0	13	-4	-1	-2	7	12	∞	∞
4	0	8	-4	-1	-2	2	7	12	∞
5	0	8	-4	-1	-2	2	7	7	13



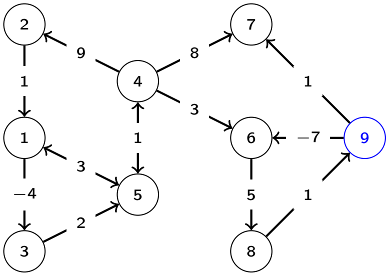
k	1	2	3	4	5	6	7	8	9
0	0	∞	∞	∞	∞	∞	∞	∞	∞
1	0	∞	-4	∞	3	∞	∞	∞	∞
2	0	∞	-4	4	-2	∞	∞	∞	∞
3	0	13	-4	-1	-2	7	12	∞	∞
4	0	8	-4	-1	-2	2	7	12	∞
5	0	8	-4	-1	-2	2	7	7	13
6	0	8	-4	-1	-2	2	7	7	8



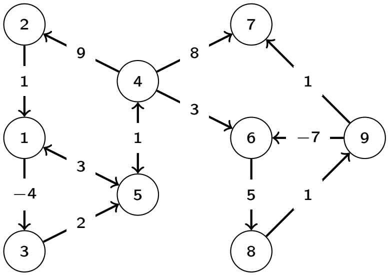
k	1	2	3	4	5	6	7	8	9
0	0	∞	∞	∞	∞	∞	∞	∞	∞
1	0	∞	−4	∞	3	∞	∞	∞	∞
2	0	∞	−4	4	−2	∞	∞	∞	∞
3	0	13	−4	−1	−2	7	12	∞	∞
4	0	8	−4	−1	−2	2	7	12	∞
5	0	8	−4	−1	−2	2	7	7	13
6	0	8	−4	−1	−2	2	7	7	8
7	0	8	−4	−1	−2	1	7	7	8



k	1	2	3	4	5	6	7	8	9
0	0	∞	∞	∞	∞	∞	∞	∞	∞
1	0	∞	-4	∞	3	∞	∞	∞	∞
2	0	∞	-4	4	-2	∞	∞	∞	∞
3	0	13	-4	-1	-2	7	12	∞	∞
4	0	8	-4	-1	-2	2	7	12	∞
5	0	8	-4	-1	-2	2	7	7	13
6	0	8	-4	-1	-2	2	7	7	8
7	0	8	-4	-1	-2	1	7	7	8
8	0	8	-4	-1	-2	1	7	6	8



k	1	2	3	4	5	6	7	8	9
0	0	∞	∞	∞	∞	∞	∞	∞	∞
1	0	∞	-4	∞	3	∞	∞	∞	∞
2	0	∞	-4	4	-2	∞	∞	∞	∞
3	0	13	-4	-1	-2	7	12	∞	∞
4	0	8	-4	-1	-2	2	7	12	∞
5	0	8	-4	-1	-2	2	7	7	13
6	0	8	-4	-1	-2	2	7	7	8
7	0	8	-4	-1	-2	1	7	7	8
8	0	8	-4	-1	-2	1	7	6	8
9	0	8	-4	-1	-2	1	7	6	7



k	1	2	3	4	5	6	7	8	9
0	0	∞	∞	∞	∞	∞	∞	∞	∞
1	0	∞	-4	∞	3	∞	∞	∞	∞
2	0	∞	-4	4	-2	∞	∞	∞	∞
3	0	13	-4	-1	-2	7	12	∞	∞
4	0	8	-4	-1	-2	2	7	12	∞
5	0	8	-4	-1	-2	2	7	7	13
6	0	8	-4	-1	-2	2	7	7	8
7	0	8	-4	-1	-2	1	7	7	8
8	0	8	-4	-1	-2	1	7	6	8
9	0	8	-4	-1	-2	1	7	6	7

```cpp
vi bellman_ford(vvii& g, int s)
{
    int i, j, k, n = g.size(), x, w;
    vi d(g.size(), INF);
    d[s] = 0;
    for (i = 0; i < n - 1; i++) for (j = 0; j < n; j++) if (d[j] != INF)
        for (k = 0; k < g[j].size(); k++)
            d[g[j][k].first] = min(d[g[j][k].first], d[j] + g[j][k].second);
    for (i = 0; i < n - 1; i++) for (j = 0; j < n; j++) if (d[j] != INF)
        for (k = 0; k < g[j].size(); k++)
        {
            x = g[j][k].first, w = g[j][k].second;
            if (d[x] != -INF && d[j] + w < d[x]) d[x] = -INF;
        }
    return d;
}
```

- Sjáum að í fyrri hluta reikniritsins ýtrum við í gegnum alla leggi og allar hnúta $(V-1)$-sinnum.
- Tímaflækjan á þeim hluta er því $\mathcal{O}(E \cdot V)$.
- Seinni hlutinn er svo að ítra yfir nákvæmlega það sama, svo tímaflækja þar er eins.
- Því fæst að reikniritið er í heildina $\mathcal{O}(E \cdot V)$.
- Þetta er töluvert verra en reiknirit Dijkstras (svipað og að fara úr $\mathcal{O}(n \log n)$ í $\mathcal{O}(n^2)$).

- Tökum eftir að það er ekki uppfært gildi í hnút v eftir legg (u, v) nema að u hafi verið uppfært í umferðinni áður.
- Í sýnidæminu svara þetta til bláu gildanna.
- Við getum notað þetta til að bæta keyrlsuhraðann, án þess þó að bæta tímaflækjuna.

```cpp
vi bellman_ford(vvii& g, int s)
{
    int i, j, k, f = 1, n = g.size(), x, w, q[n];
    vi d(n);
    for (i = 0; i < n; i++) d[i] = i == s ? 0 : INF, q[i] = i == s ? 0 : -1;
    for (i = 0; f ; i++) for (j = f = 0; j < n; j++) if (q[j] == i)
        for (k = 0; k < g[j].size(); k++)
    {
        x = g[j][k].first, w = g[j][k].second;
        if (d[x] != -INF && d[j] + w < d[x])
            d[x] = i < n ? d[j] + w : -INF, q[x] = i + 1, f = 1;
    }
    return d;
}
```

DATE	JUDGEMENT	RUNTIME	LANGUAGE
12:01:53	Accepted	0.12 s	C++
11:50:38	Accepted	0.04 s	C++

- Það ber að nefna að ef við notum forgangsbiðröð í stað þess að nota biðröð fáum við reiknirit Dijkstras.
- Ein lítil bæting í viðbót er að geyma hnútanna sem við eigum eftir að uppfæra á hlaða.
- Þetta getur sparað vinnu, þá aðallega þegar hlaðinn er nærri tómur.
- Það þarf þó að passa að setja aldrei hnút í biðröðina tvisvar.
- Í versta falli er þessi „bæting" jafn hröð og fyrri bætingin, og mögulega aðeins hægari.
- Hún virkar þó sérstaklega vel á slembin net.

```
vi bellman_ford(vvii& g, int s)
{
    int i, j = 0, k, n = g.size(), x, w, q[2*n*n], p[n], a[n], qs = 0, qe = 0;
    vi d(n);
    q[qe] = s, p[qe++] = 0;
    for (i = 0; i < n; i++) d[i] = i == s ? 0 : INF, a[i] = i == s ? 1 : 0;
    while (qe != qs)
    {
        i = q[qs], j = p[qs++], a[i] = 0;
        for (k = 0; k < g[i].size(); k++)
        {
            x = g[i][k].first, w = g[i][k].second;
            if (d[x] != -INF && d[i] + w < d[x])
            {
                d[x] = j < n ? d[i] + w : -INF;
                if (!a[x]) a[x] = 1, q[qe] = x, p[qe++] = j + 1;
            }
        }
    }
    return d;
}
```