

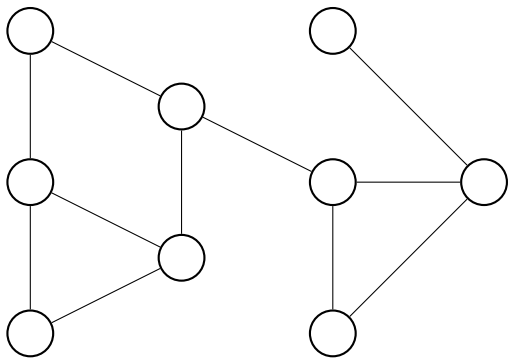
Dýptarleit og breiddarleit

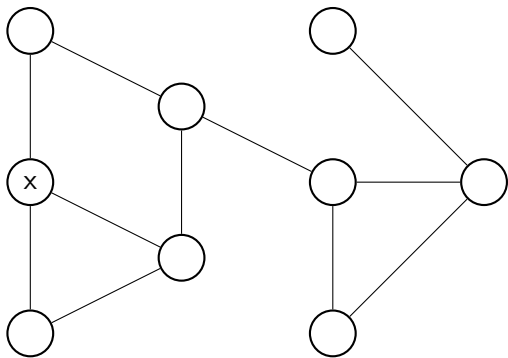
Bergur Snorrason

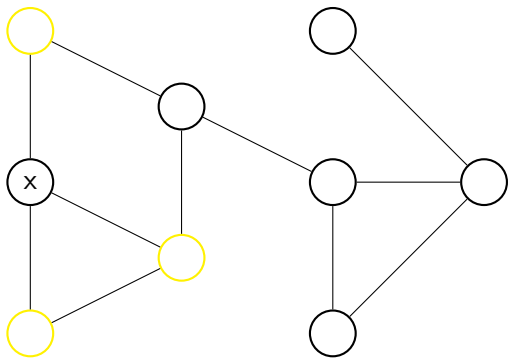
27. febrúar 2023

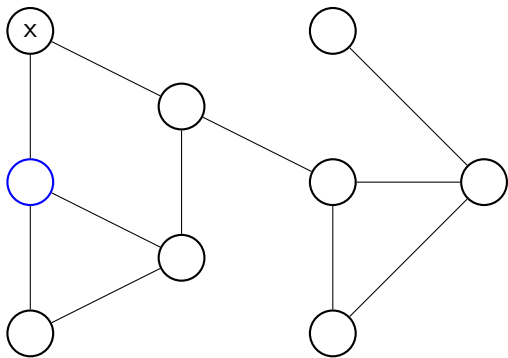
- ▶ Hvernig ítrum við í gegnum alla hnúta nets.
- ▶ Þetta má að sjálfsgöðu gera á marga vegu, en algengt er að notast við annað að tvennu:
 - ▶ *Dýptarleit* (e. *deapth-first search*).
 - ▶ *Breiddarleit* (e. *breadth-first search*).
- ▶ Báðar aðferðir byggja á því að byrja í einhverjum hnút og heimsækja svo nágranna hans.

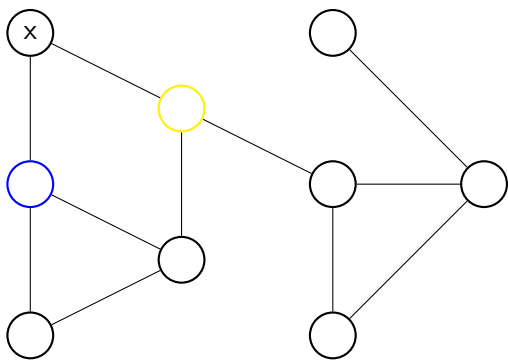
- ▶ Dýptarleit byrjar í einhverjum hnút.
- ▶ Sá hnútur er kallaður *upphafshnúturinn*.
- ▶ Í hverju skrefi heimsækir leitin einhvern nágranna hnútsins sem hefur ekki verið heimsóttur áður í leitinni.
- ▶ Ef allir nágrannar hafa verið heimsóttir þá er farið til baka og nágrannar síðasta hnúts eru skoðaðir.
- ▶ Tökum dæmi.

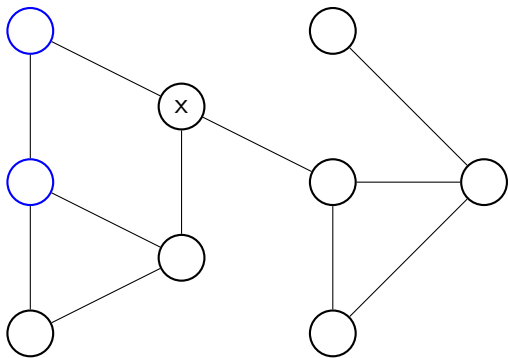


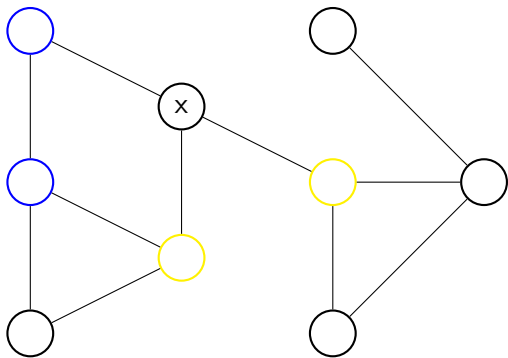


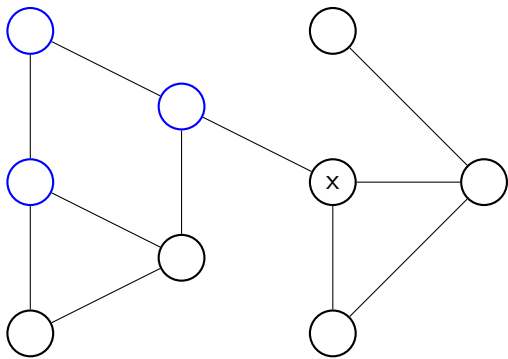


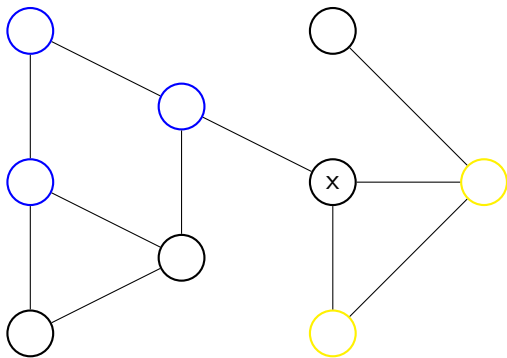


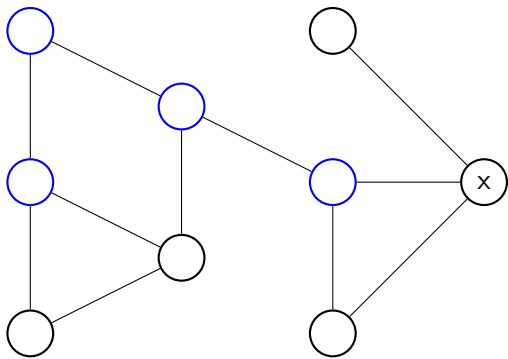


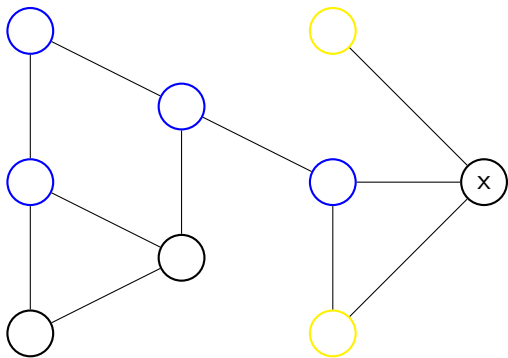


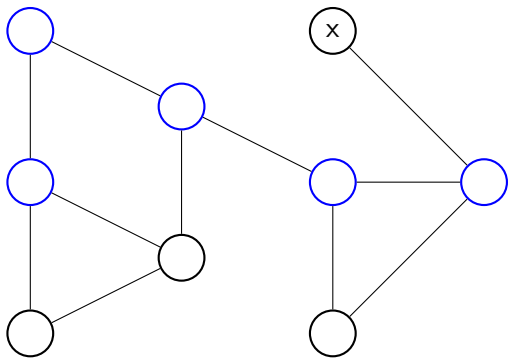


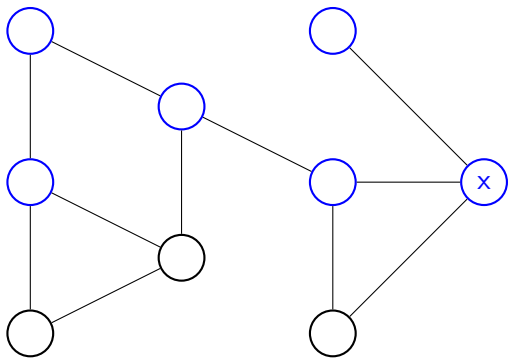


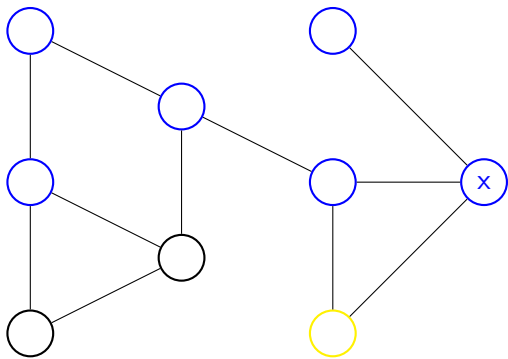


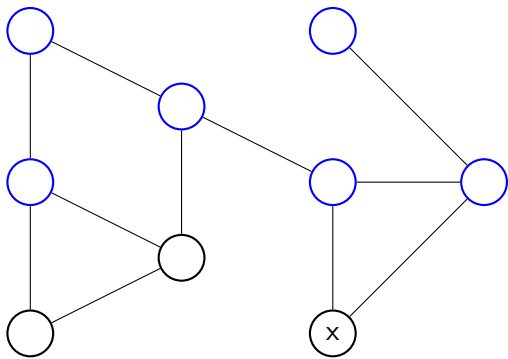


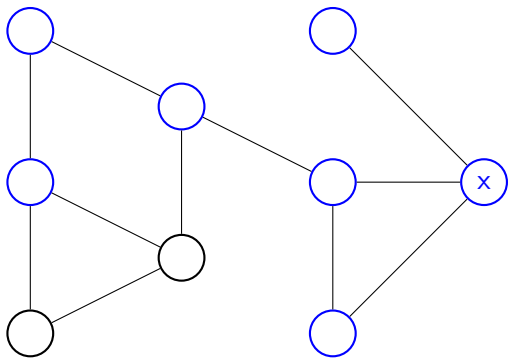


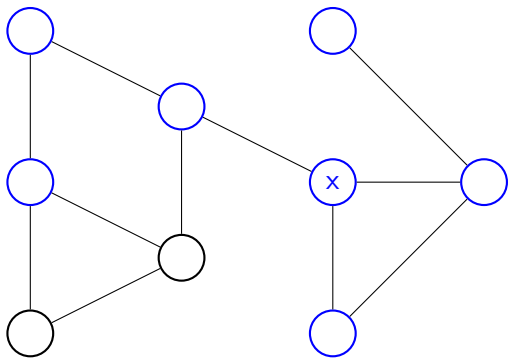


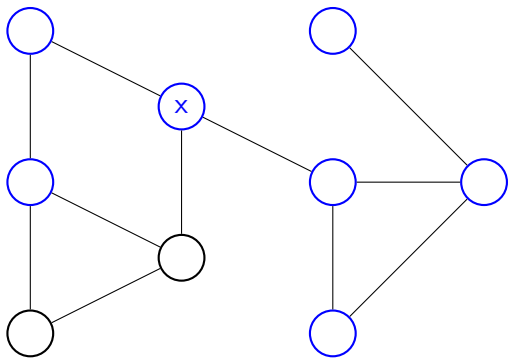


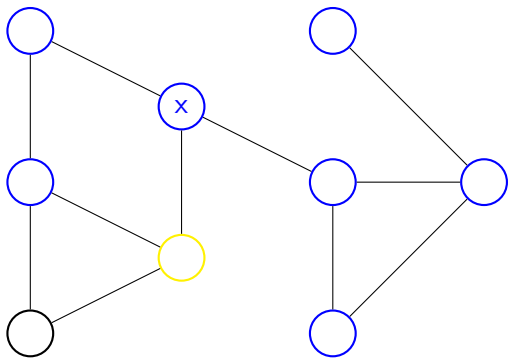


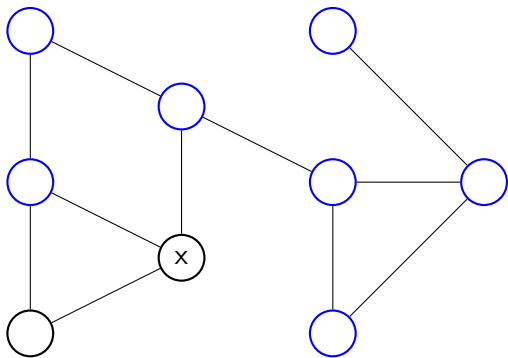


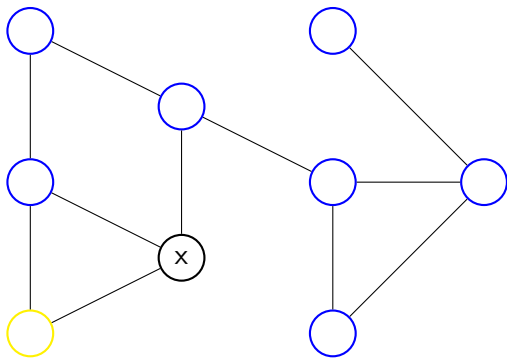


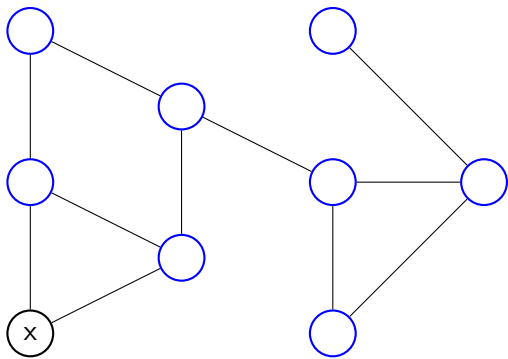


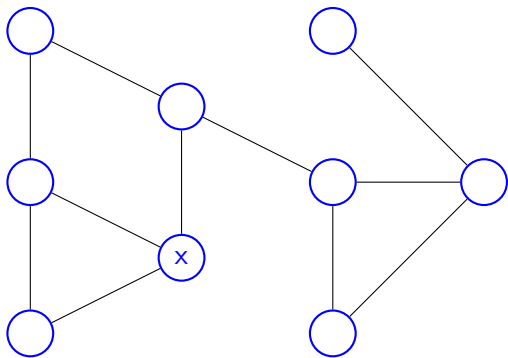


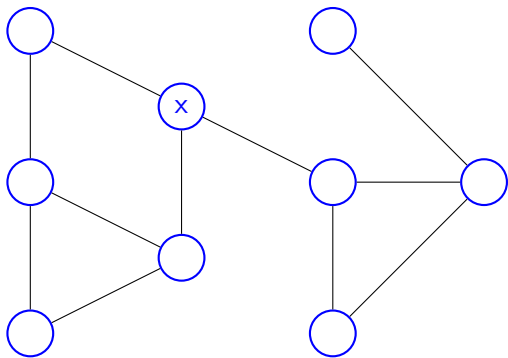


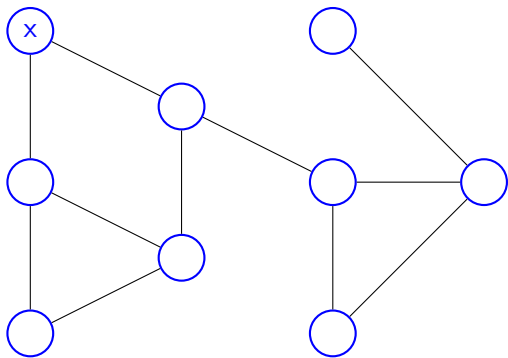


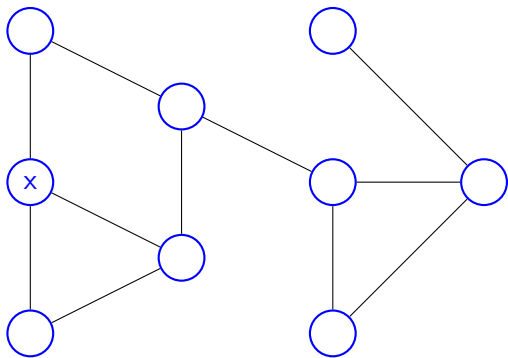


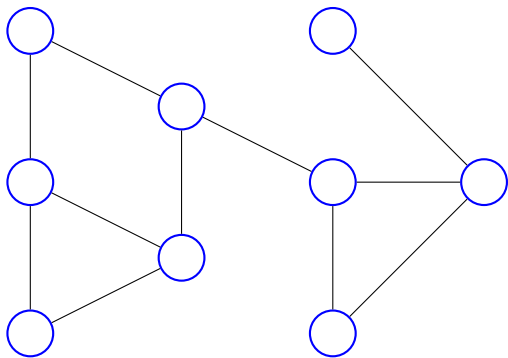












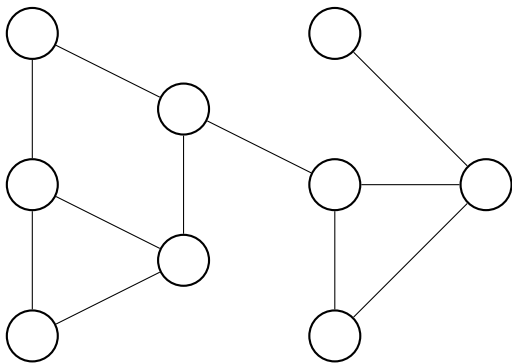
- ▶ Þegar kemur að því að útfæra dýptarleit er oftast notast við endurkvæmni.
- ▶ Endurkvæmnin sér sjálfkrafa um að „fara til baka”.

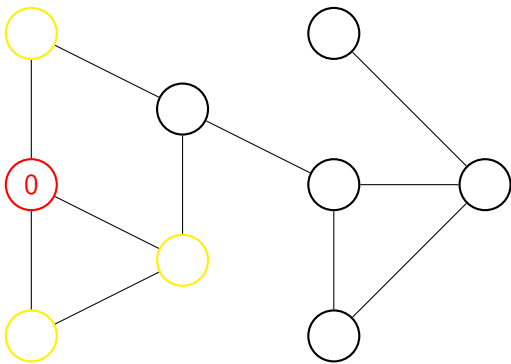
```
6 vi v;  
7 void dfs(vvi& g, int x)  
8 {  
9     int i;  
10    printf("Vid erum i nodu %d\n", x + 1);  
11    v[x] = 1;  
12    for (i = 0; i < g[x].size(); i++) if (v[g[x][i]] == 0)  
13        dfs(g, g[x][i]);  
14 }
```

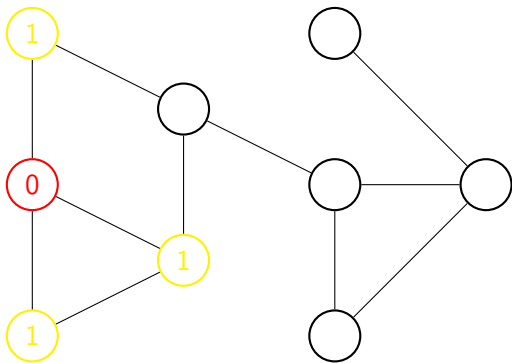
- ▶ Eftir kall á `dfs(g, 0)` segir `v[j]` okkur hvort til sé vegur frá hnúti 0 til hnúts *j*.

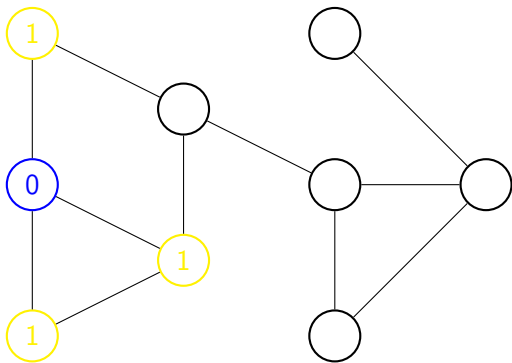
- ▶ Tökum eftir að leitin heimsækir hvern hnút í mesta lagi einu sinni og ferðast eftir hverjum legg í mesta lagi tvisvar (einu sinni í stefndu neti).
- ▶ Svo tímaflækjan er $\mathcal{O}(E + V)$.
- ▶ Við getum í rauninni ekki beðið um betri tímaflækju.
- ▶ Við munum alltaf þurfa að skoða alla hnúta og ef við skoðum ekki alla leggi þá erum við að hunsa uppbyggingu netsins.

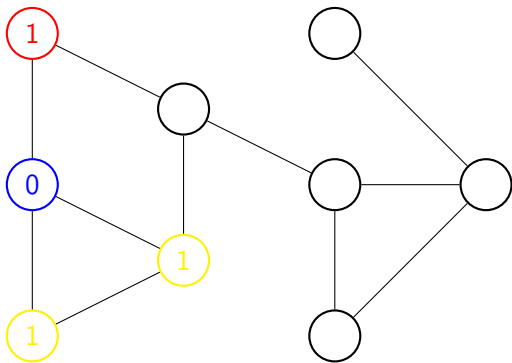
- ▶ Við byrjum á að merkja alla hnúta sem „ósnera“, nema við merkjum einn hnút sem „séðan“.
- ▶ Sá hnútur er kallaður *upphafshnúturinn*.
- ▶ Við endurtökum svo eftirfarandi þar til engir „séðir“ hnútar eru eftir:
 - ▶ Veljum þann „séða“ hnút sem við sáum fyrst.
 - ▶ Merkjum alla „ósnera“ nágranna hans sem „séða“.
 - ▶ Merkjum upprunalegu hnútinn „kláraðann“.
- ▶ Tökum dæmi.
- ▶ Við munum merkja „séða“ hnúta með hvenær við sáum þá.

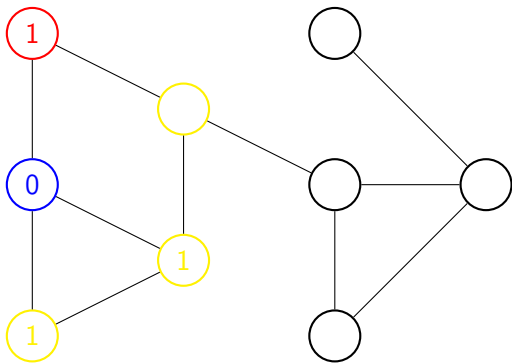


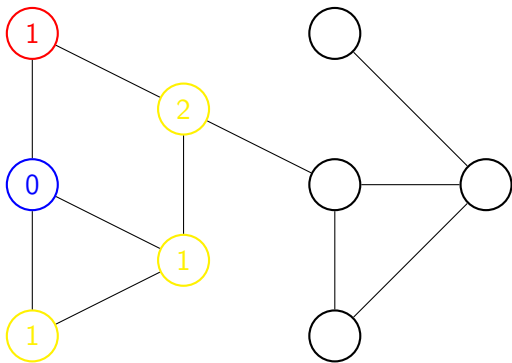


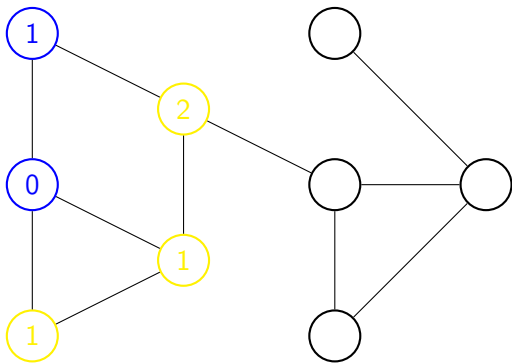


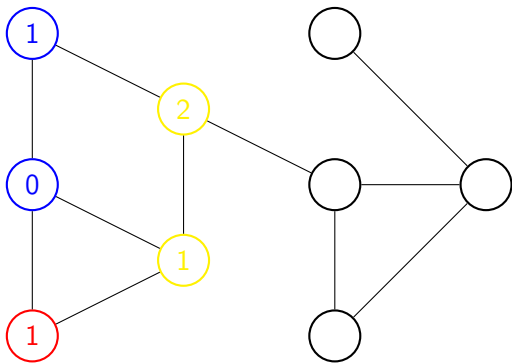


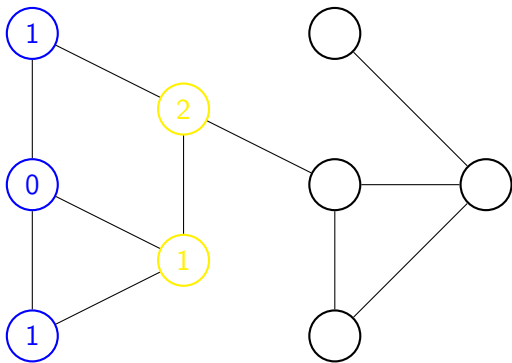


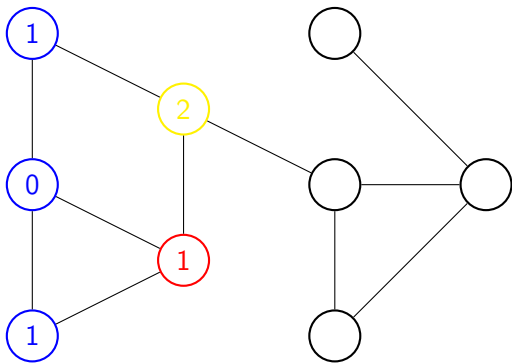


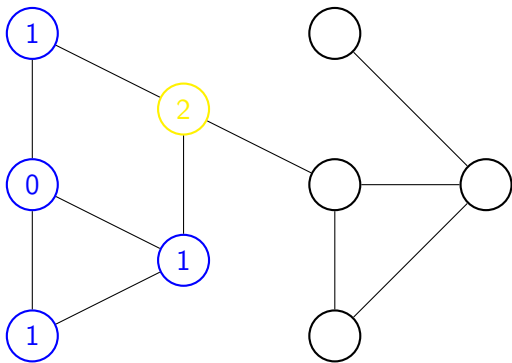


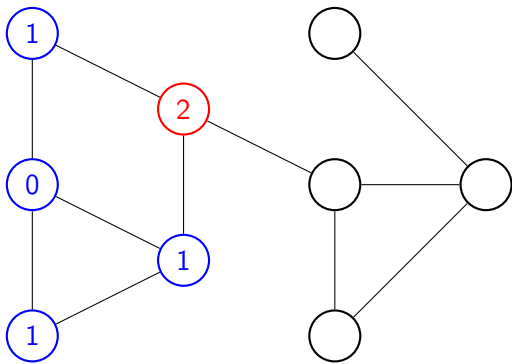


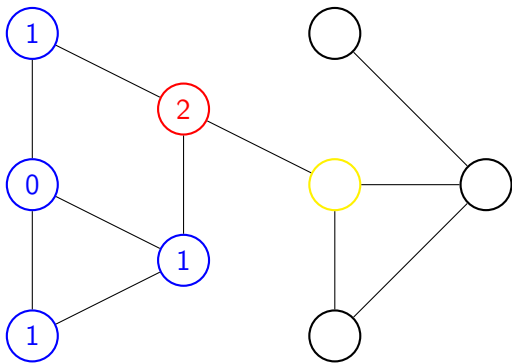


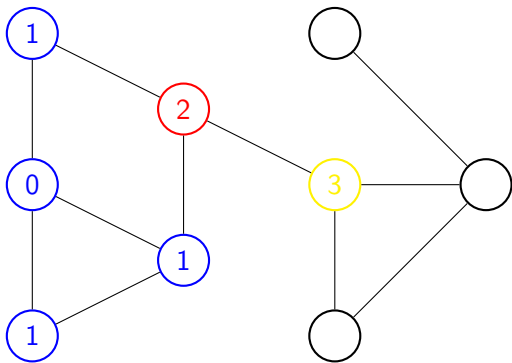


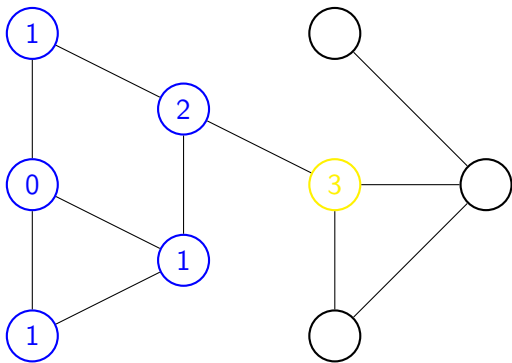


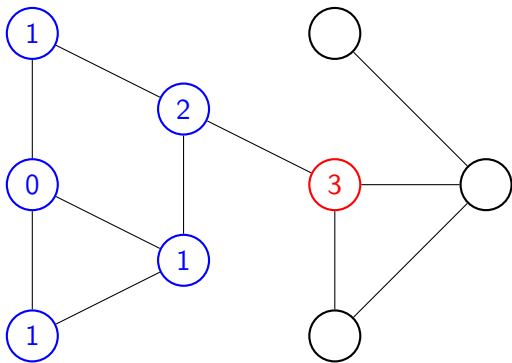


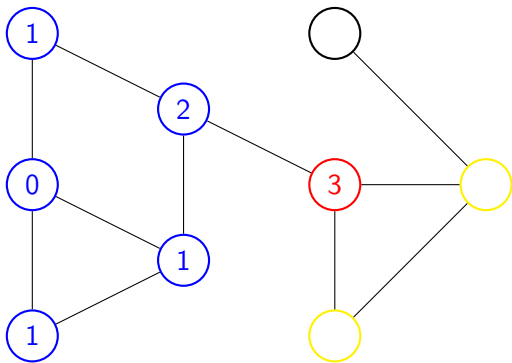


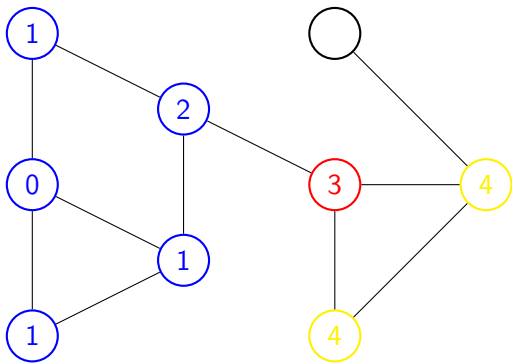


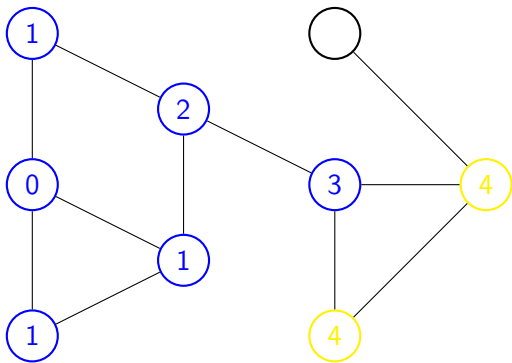


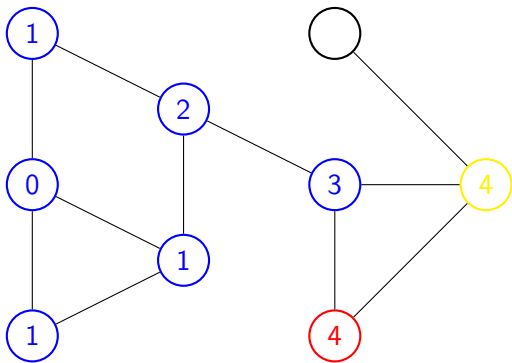


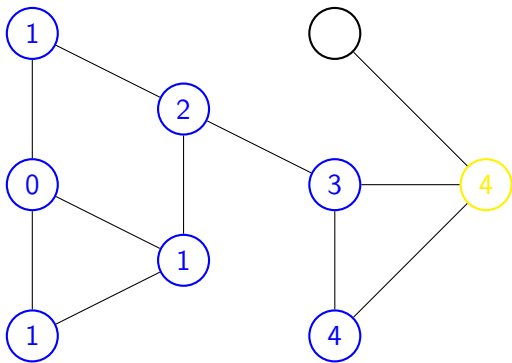


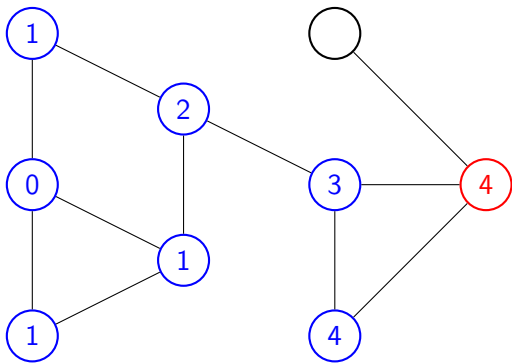


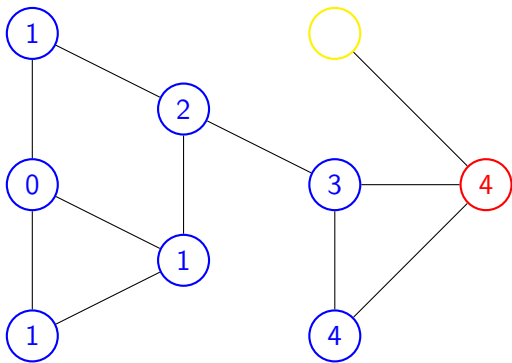


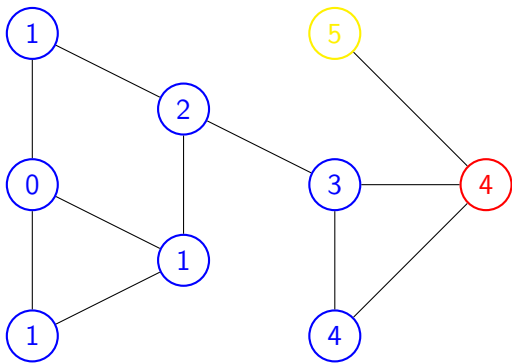


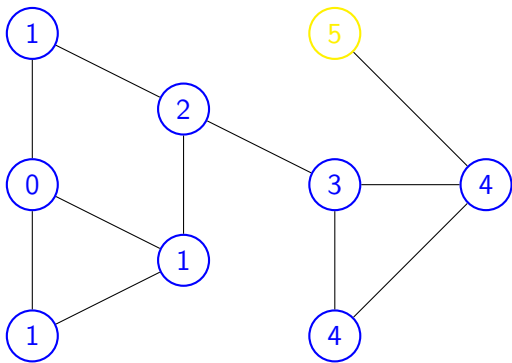


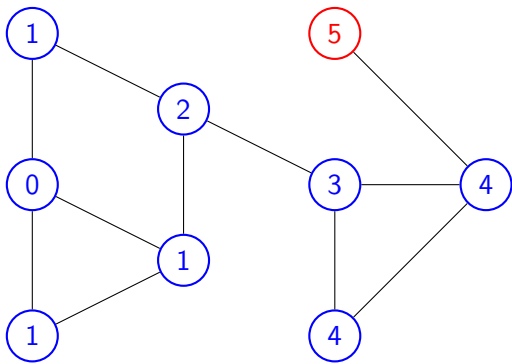


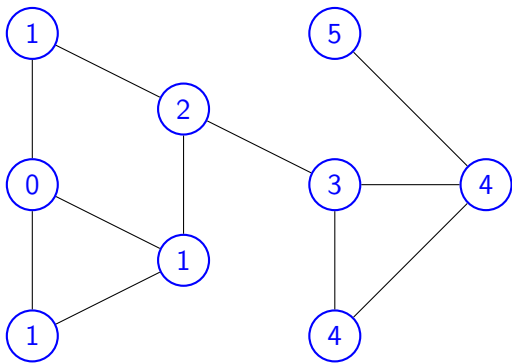












- ▶ Við munum halda utan um „séða” hnúta með biðröð.
- ▶ Við byrjum því á að setja upphafshnútin okkar í biðröðina.
- ▶ Við tökum svo hnút úr biðröðinni, setjum alla „óséða” nágranna hans í biðröðina og höldum áfram þangað til biðröðin er tóm.

```
18     vi d(n, -1);
19     queue<int> q;
20     q.push(0);
21     d[0] = 0;
22     while (q.size() > 0)
23     {
24         int x = q.front();
25         q.pop();
26         for (i = 0; i < g[x].size(); i++) if (d[g[x][i]] == -1)
27         {
28             q.push(g[x][i]);
29             d[g[x][i]] = d[x] + 1;
30         }
31     }
```

- ▶ Við segjum að hnútar u og v séu fjarlægð k frá hvorum öðrum ef stysti vegurinn frá u til v er af lengd k .
- ▶ Við segjum líka að það séu k skref á milli hnútanna.
- ▶ Ef enginn vegur er á milli hnútanna segjum við að lengdin á milli þeirra sé ∞ .
- ▶ Mikilvægur eiginleiki breiddarleitar er að hún heimsækir fyrst þá hnúta sem eru næst upphafshnútnum.
- ▶ Með öðrum orðum, ef u er k_1 skref frá upphafshnútnum og v er k_2 skref frá upphafshnútnum, $k_1 \neq k_2$, þá heimsækir breiddarleit u á undan v þá og því aðeins að $k_1 < k_2$.
- ▶ Við getum því notað breiddarleit til að finna fjarlægðina frá upphafshnútnum að öllum öðrum hnútum.

- ▶ Líkt og í dýptarleit þá heimsækjum við hvern hnút í mesta lagi einu sinni og ferðumst eftir hverjum legg í mesta lagi tvisvar (einu sinni í stefndu neti).
- ▶ Svo tímaflækjan er aftur $\mathcal{O}(E + V)$.

- ▶ Báðar leitirnar segja okkur til hvaða hnúta megi komast frá upphafshnútnum og gera það með sömu tímaflækju.
- ▶ Breiddarleit gefur okkur einnig fjarlægð allra hnúta frá upphafshnútnum.
- ▶ Í dýptarleit getum við unnið áfram með gögnin eftir endurkvæma kallið okkar, sem býður upp á mikla fjölbreyttni.
- ▶ Dýptarleit má því finna í reikniritum sem finna grannröð neta, tengipunkta og brýr (þetta verður allt skilgreint seinna).

- ▶ Tökum dæmi.
- ▶ Fyrsta lína inntaksins inniheldur tvær heiltölur, r og c .
- ▶ Inntakið inniheldur síðan r strengir, allir af lengd c .
- ▶ Strengirnar byrjar og endar allir á 'X' ásamt því að fyrsti og síðasti strengurinn inniheldur bara stafinn 'X'.
- ▶ Annars innihalda strengirnir bara stafina 'X', '.' og eitt stykki 'O'.

```

1  11 14
2  XXXXXXXXXXXXXXXX
3  X.X . . . . . X
4  X.X.XXXX.X.X.X
5  X.X . . . . XXXX.X
6  XOXXXXXX...X.X
7  X..X.X...X.X.X
8  X...XX.XXX.X.X
9  X...X...X.X.X
10 XXXX.XXX.X.X.X
11 X . . . . . X . . X
12 XXXXXXXXXXXXXXXX

```

- ▶ Við viljum svo prenta sama borð, nema í stað bókstafana á að koma hversu fá skref við þurfum að taka til frá 'O' til að komast þangað ef við megum ferðast upp, niður, til hægri og til vinstri, en ekki á reitunum með 'X'.
- ▶ Fyrir þá reiti sem við komumst ekki á prentum við -1.

► Sem dæmi hefur inntakið

```
1 11 14
2 XXXXXXXXXXXXXXXX
3 X.X . . . . . X
4 X.X.XXXX.X.X.X
5 X.X . . . . XXXX.X
6 XOXXXXXX...X.X
7 X..X.X...X.X.X
8 X...XX.XXX.X.X
9 X....X...X.X.X
10 XXXX.XXX.X.X.X
11 X . . . . . X . . X
12 XXXXXXXXXXXXXXXX
```

úttakið

```
1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
2 -1 3 -1 47 46 45 44 43 42 41 40 39 38 -1
3 -1 2 -1 48 -1 -1 -1 -1 43 -1 41 -1 37 -1
4 -1 1 -1 49 50 51 52 53 -1 -1 -1 -1 36 -1
5 -1 0 -1 -1 -1 -1 -1 -1 21 22 23 -1 35 -1
6 -1 1 2 -1 -1 -1 18 19 20 -1 24 -1 34 -1
7 -1 2 3 4 -1 -1 17 -1 -1 -1 25 -1 33 -1
8 -1 3 4 5 6 -1 16 15 14 -1 26 -1 32 -1
9 -1 -1 -1 -1 7 -1 -1 -1 13 -1 27 -1 31 -1
10 -1 11 10 9 8 9 10 11 12 -1 28 29 30 -1
11 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
```

- ▶ Hvernig tengist þetta dæmi efni vikunnar?
- ▶ Við getum túlkað þessa mynd sem net.
- ▶ Ímyndum okkur að hver auður reitur sé hnútur.
- ▶ Við tengjum svo aðliggjandi auða hnúta með leggjum.
- ▶ Þar sem við viljum finna fjalægðir frá tilteknum hnút til allra annara hnúta notum við breiddarleit.
- ▶ Fjöldi hnúta í netinu er alltaf minni en $r \cdot c$ og fjöldi leggja er alltaf minni en $2 \cdot r \cdot c$.
- ▶ Svo þetta reiknirit er $\mathcal{O}(E + V) = \mathcal{O}(r \cdot c)$.

```

5 int isin(int x, int y, int r, int c)
6 {
7     if (x >= r || x < 0 || y >= c || y < 0) return 0;
8     return 1;
9 }
10
11 int main()
12 {
13     int i, j, r, c, x, y, g[4][2] = {{-1, 0}, {1, 0}, {0, -1}, {0, 1}};
14     cin >> r >> c;
15     string a[r];
16     for (i = 0; i < r; i++) cin >> a[i];
17     for (i = 0; i < r; i++) for (j = 0; j < c; j++) if (a[i][j] == 'O')
18         x = i, y = j;
19     int d[r][c];
20     for (i = 0; i < r; i++) for (j = 0; j < c; j++) d[i][j] = -1;
21     queue<ii> q;
22     q.push(ii(x, y)), d[x][y] = 0;
23     while (q.size() > 0)
24     {
25         ii p = q.front(); q.pop();
26         x = p.first, y = p.second;
27         for (i = 0; i < 4; i++)
28         {
29             int xx = x + g[i][0], yy = y + g[i][1];
30             if (a[xx][yy] == 'X' || !isin(xx, yy, r, c) || d[xx][yy] != -1)
31                 continue;
32             q.push(ii(xx, yy)), d[xx][yy] = d[x][y] + 1;
33         }
34     }
35     for (i = 0; i < r; i++)
36     {
37         for (j = 0; j < c; j++) printf("%2d ", d[i][j]);
38         printf("\n");
39     }
40     return 0;
41 }

```

- ▶ Tökum annað dæmi.
- ▶ Ykkur er gefið net og tiltekinn hnút u .
- ▶ Prentið alla einfalda vegi í netinu sem byrja í u .
- ▶ Munið að vegur er einfaldur ef hann heimsækir aldrei sama hnútinn tvisvar.

- ▶ Við getum leyst þetta með því að breita dýptarleitar útfærslunni okkar lítillega.
- ▶ Til að koma í veg fyrir að heimsækja hnút oftari en einu sinni í dýptarleit merkjum við hann og heimsækjum ekki merktu hnúta.
- ▶ Við munum ennþá þurfa að merkja hnúta því við erum að leita að einföldum vegum (almennt er ekki takmarkaður fjöldi vega í neti).
- ▶ Munurinn er að við munum merkja hnút þegar við sjáum hann, halda áfram endurkvæmt til ómerktra nágranna hans og afmerkja hann svo.


```

7 void dfs(vvi& g, int x)
8 {
9     int i;
10    v[x] = 1;
11    p.push_back(x);
12    for (i = 0; i < p.size(); i++) printf("%d ", p[i] + 1);
13    printf("\n");
14    for (i = 0; i < g[x].size(); i++) if (v[g[x][i]] == 0)
15        dfs(g, g[x][i]);
16    p.pop_back();
17    v[x] = 0;
18 }

```

- ▶ Þetta forrit prentar alla einfalda vegi sem byrja í hnút u .
- ▶ Til að hámarka fjölda slíkra vega getum við búið til net þar sem öll pör hnúta eru nágrannar.
- ▶ Þá myndi þetta forrit prenta allar umraðanir sem byrja á x .
- ▶ Tímaflækjan er því $\mathcal{O}(V!)$.

