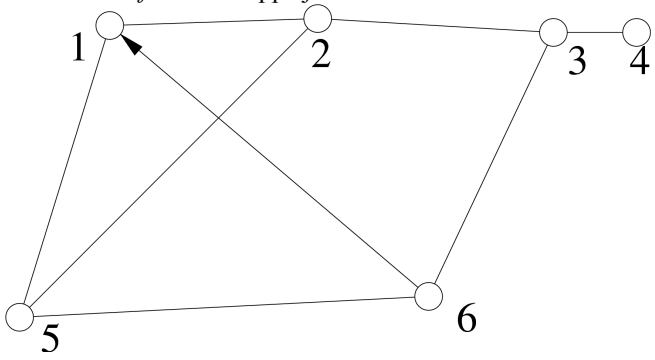
Graafid ja nendega seotud põhialgoritmid

Margus Niitsoo, Siim Ainsaar 2006

Graaf — hulk tippe ja neid ühendavaid servi.



Esitused arvutis:

- Servade nimekiri *(edge list):*

```
(1 2) (1 5) (2 1) (2 3) (2 5)
(3 2) (3 4) (3 6) (4 3) (5 1)
(5 2) (5 6) (6 1) (6 3) (6 5)
```

  - Mugav, kui servade kohta on palju lisainfot.
  - Enamikus algoritmides pole eriti praktiline.
  - Kui on ka suunaga servi, siis tuleb suunata servad topelt salvestada.

- Naabrusnimistu (*adjacency list*):

```
1: 2 5      2: 1 3 5
3: 2 4 6    4: 3
5: 1 2 6    6: 1 3 5
```

  - Võimaldab tippude kohta lisainfot säilitada.
  - Lubab kiiresti läbi vaadata tipu naabrid.
  - Eriti praktiline keerukamates algoritmides.

- Naabrusmaatriks (*adjacency matrix*):

	1	**2**	**3**	**4**	**5**	**6**
1	0	1	0	0	1	0
2	1	0	1	0	1	0
3	0	1	0	1	0	1
4	0	0	1	0	0	0
5	1	1	0	0	0	1
6	1	0	1	0	1	0

  - 0-de ja 1-de asemel võib sisaldada nt. servade pikkusi.
  - Kõige universaalsem.
  - Lihtne andmestruktuur.

Sügavuti otsimine (*depth-first search, DFS*)**:**

```
boolean käidud[tippude_arv] =
                    {false, ...};

function otsi(tipp t)
{
      käidud[t] = true;
      töötle(t);

      for n in naabrid(i)
            if (käidud[n] == false)
                  otsi(n);
}

otsi(algus);
```

- Lihtne kirjutada.
- Nõuab pisut vähem mälu kui laiuti otsing, küll aga kulutab *stack*'i.
- Otseselt üldistatav läbivaatusülesannetele.
- Hea tsüklite otsimiseks.
- Liigub ainult naabrilt naabrile.

Laiuti otsimine (*breadth-first search, BFS*)**:**

```
boolean lisatud[tippude_arv] =
                          {false, ...};

tipp järjekord[tippude_arv];
järjekord[0] = algus;

int i = 0, k = 1;

while (i < k)
{
      tipp t = järjekord[i];
      töötle(t);

      for n in naabrid(t)
      {
            if (lisatud[n] == false)
            {
                  lisatud[n] = true;
                  järjekord[k] = t;
                  k = k + 1;
            }
      }

      i = i + 1;
}
```

- Mugav lühimate teede leidmiseks (servade arvu mõttes).
- Vajab lisamälu järjekorra jaoks.
- Üldiselt laiema kasutusalaga kui sügavuti otsing.

Ühest tipust teistesse lühimate teede otsimine
(*Dijikstra algoritm*)**:**

```
tipp käimata[tippude_arv] = tipud;
real kaugus[tippude_arv] =
                          {∞, ... };

kaugus[algus] = 0;

repeat
{
  t=-1;
  for i in käimata
    if (kaugus[i]<kaugus[t])
      t=i;

  if (t==-1) exit;

  remove(t, käimata);

  for n in naabrid(t)
  if (kaugus[n] > kaugus[t] + s[t,n])
      kaugus[n] = kaugus[t]+s[t,n];
}
```

- Võimaldab lühimad teed leida ka siis, kui servadel on antud pikkused
- Servade pikkused peavad olema positiivsed
- Võimalik kirjutada keerukusega O(n log n) kui käimata ja kaugused hoida kahendkuhjas ja sealt miinimume leida.
- Tavaline lahendus on $O(n^2)$

Kõikjalt kõikjale lühimate teede otsimine
(*Floyd-Warshall'i algoritm*)**:**

```
real D[tippude_arv][tippude_arv] =
{ Naabrusmaatiks servapikkustega,
kus puuduva serva kohal on ∞ }

for i = 1 to tippude_arv
for k = 1 to tippude_arv
for l = 1 to tippude_arv
      if (D[k][i]+D[i][l]<D[k][l])
            D[k][l]=D[k][i]+D[i][l];
```

- Väga lihtne (5. rida koodi) viis kõikide lühimate teede leidmiseks naabrusmaatriksi põhjal.
- Keerukus $O(n^3)$, seega aeglasem kui Dijikstra algoritmi kiirema variandi kasutamine samal otstarbel (mis oleks $O(n^2 \log n)$).