Graafidest programmeerijatele

Targo Tennisberg

Veebruar 2014

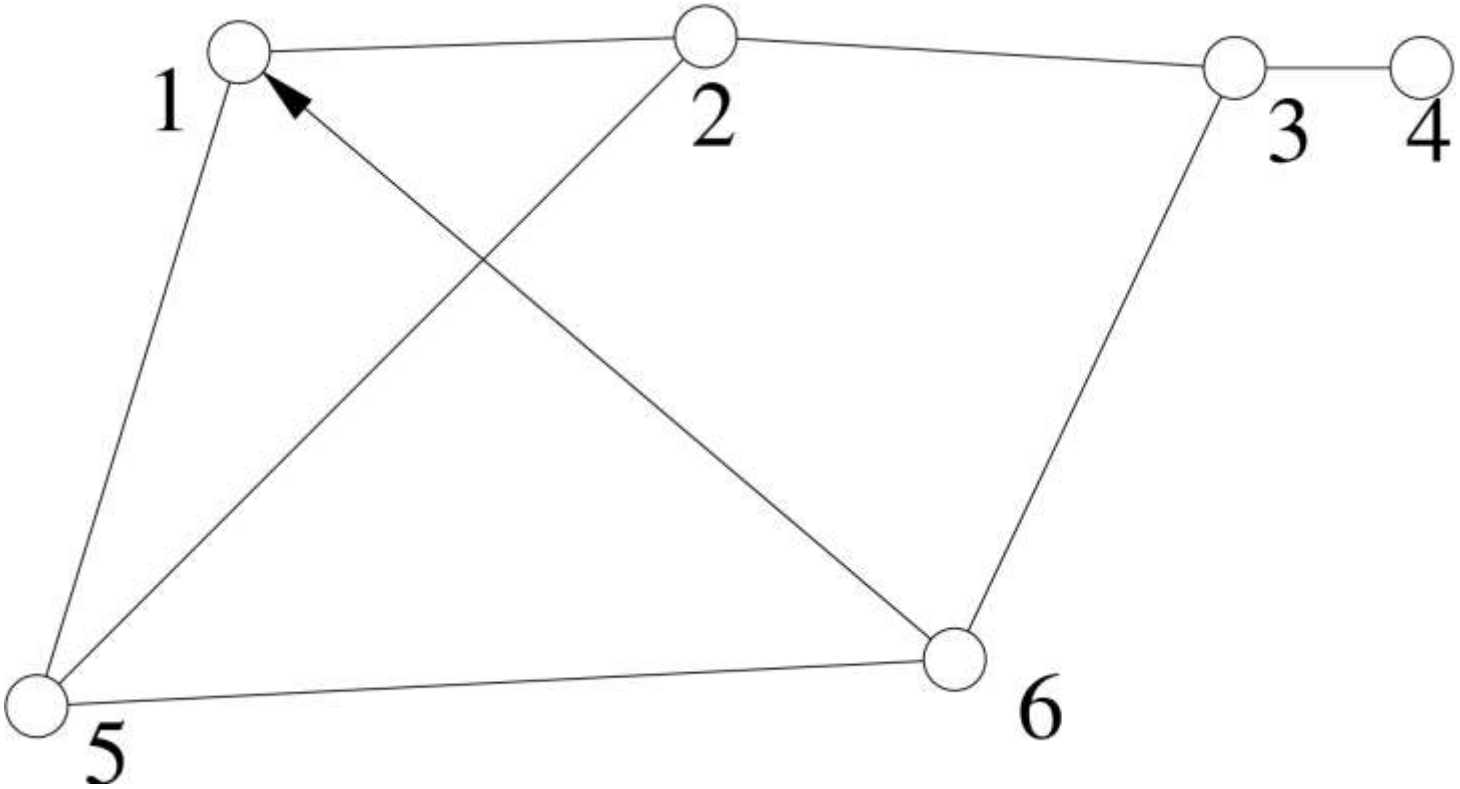
http://www.targotennisberg.com/tarkvara

Kava

- Alg
  - graafi mõiste ja põhialgoritmid
  - sügavuti ja laiuti läbimine
  - sidususkomponentide leidmine
  - topoloogiline sortimine
  - lühimad teed kaaludeta graafis
- Kesk
  - tõsisemad graafialgoritmid I
  - Floyd-Warshall
  - Dijkstra
- Taga
  - tõsisemad graafialgoritmid II
  - Euleri ahel
  - toesepuud

Põhimõisted

- Graaf — hulk **tippe** ja neid ühendavaid **servi**.



Esitused arvutis 1: 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.

Esitused arvutis 2: 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.

Esitused arvutis 3: 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 kaidud[tippude_arv] = {false, ...};
function otsi(tipp t)
{
        kaidud[t] = true;
        tootle(t);
        for n in naabrid(i)
                if (kaidud[n] == false)
                        otsi(n);
}
otsi(algus);
```

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

Sügavuti otsimine – ratsu teekond

- Ülesanne: leida maleratsu teekond malelaual, mis läbib kõik ruudud täpselt üks kord.

- Idee optimiseerimiseks: luua kõigepealt graaf, mis sisaldab kõiki võimalikke ruutudevahelisi seoseid

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

```
boolean lisatud[tippude_arv] = {false, ...};
tipp jarjekord[tippude_arv];
jarjekord[0] = algus;
int i = 0, k = 1;
while (i < k)
{
    tipp t = jarjekord[i];
    tootle(t);
    for n in naabrid(t)
    {
        if (lisatud[n] == false)
        {
            lisatud[n] = true;
            jarjekord[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.

Laiuti otsimine – tee punktist punkti

- Ülesanne: leida maleratsu lühim tee ühelt ruudult teisele

Sidususkomponentide leidmine

- Ülesanne: Normaalselt saab ratsu liikuda kõigi malelaua ruutude vahel. Aga praegu on seal ka hulk ettureid – nendele ruutudele ratsu käia ei saa. Leida, mitu ratsut saab malelauale paigutada, nii et nad ei saa üksteise ruutudele käia (ükskõik, mitme käiguga).

- Võib kasutada nii sügavuti kui laiuti otsimist

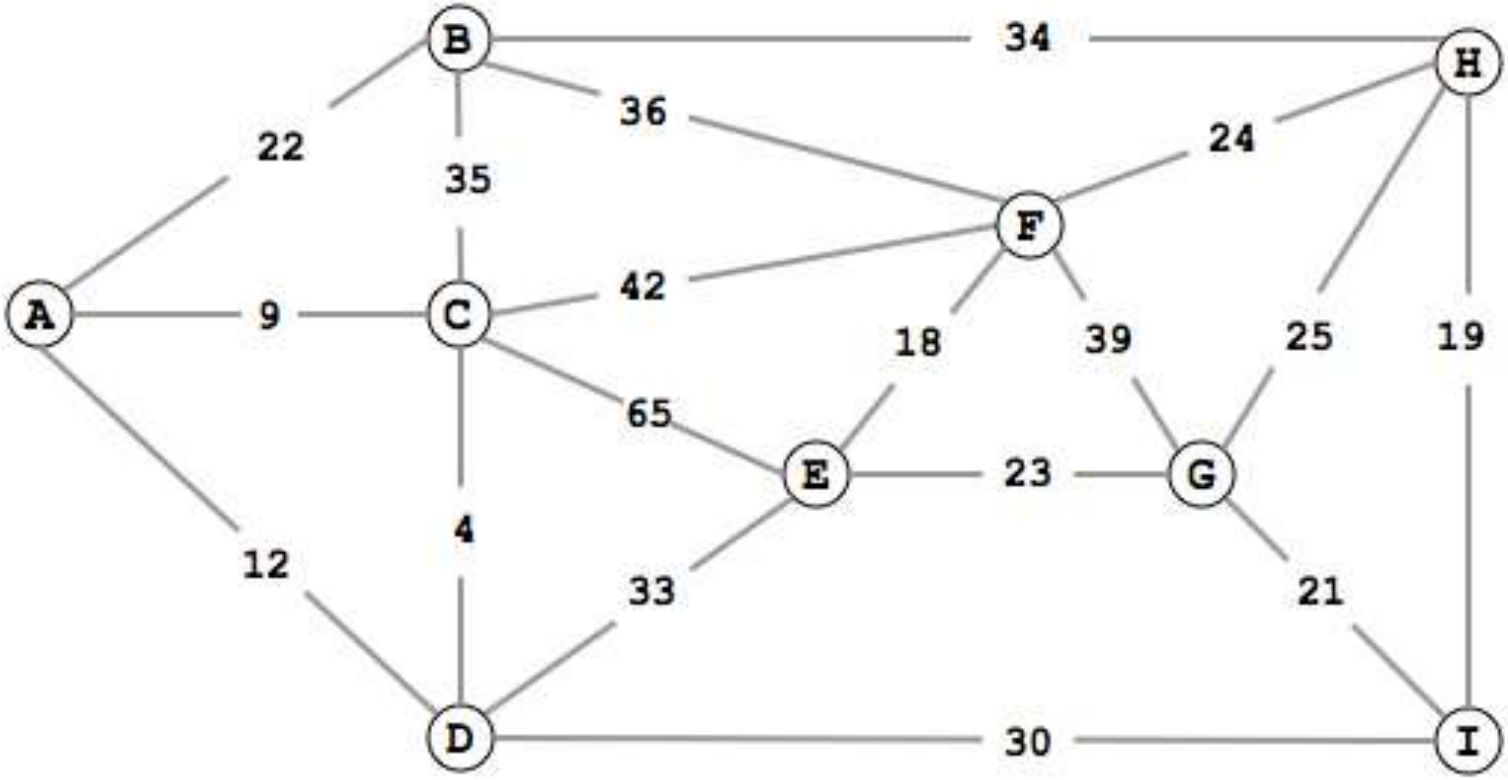
Topoloogiline sorteerimine

```
L ← Empty list that will contain the sorted elements
S ← Set of all nodes with no incoming edges
while S is non-empty do
    remove a node n from S
    add n to tail of L
    for each node m with an edge e from n to m do
        remove edge e from the graph
        if m has no other incoming edges then
            insert m into S
if graph has edges then
    return error (graph has at least one cycle)
else
    return L (a topologically sorted order)
```

Topoloogiline sorteerimine

- Ülesanne: antud on hulk loomade paare, kus esimene loom on teisest tugevam. Sorteerida kõik loomad tugevuse järjekorda.

Kaaludega graaf



Lühim tee kaaludega graafis – Dijkstra algoritm

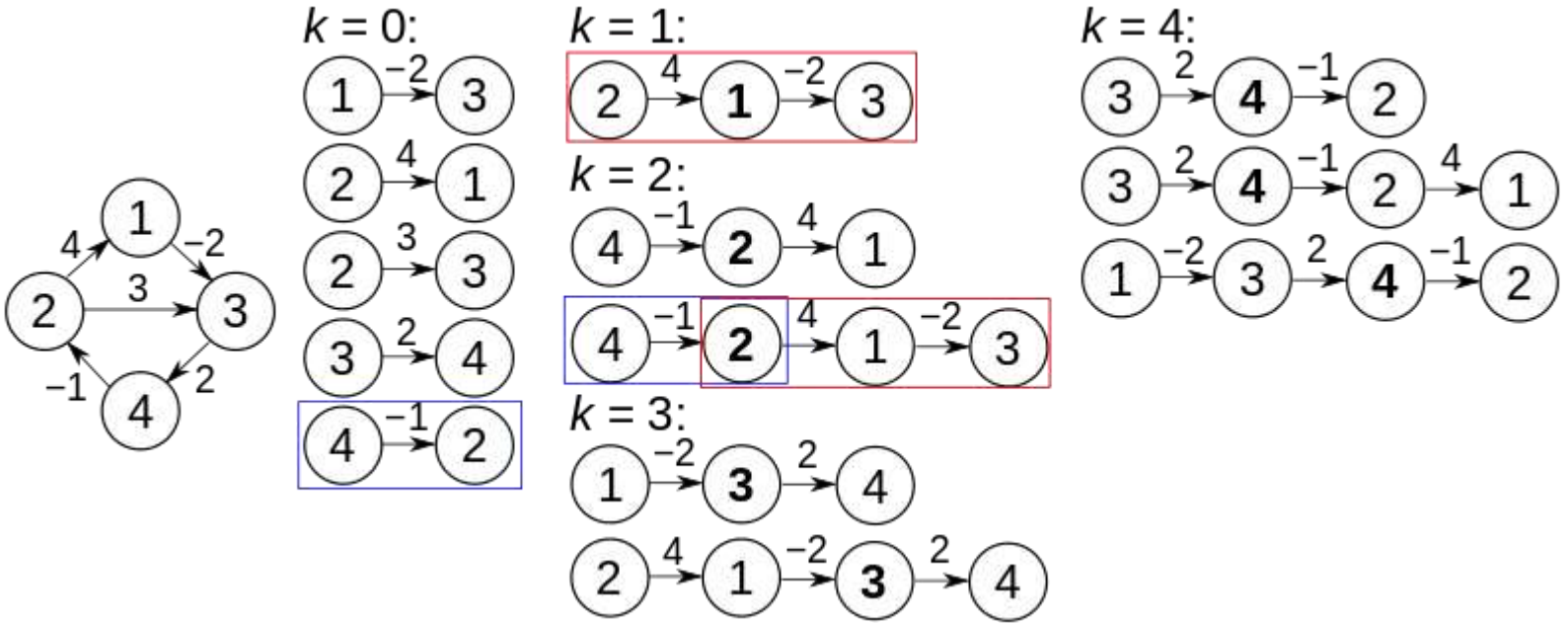
- Laiuti otsimise üldistus

```
käimata = tippude hulk;
s = kauguste maatriks;
real kaugus[tippude_arv] = {∞, ... };
kaugus[algus] = 0;
while(käimata not empty)
{
        for i in käimata  // leiame vähima kaugusega tipu
                if (kaugus[i]<kaugus[t])
                        t=i;
        if (t==otsitav) break;
        remove(käimata, t);
        for n in naabrid(t)
                if (kaugus[n] > kaugus[t] + s[t,n])
                        kaugus[n] = kaugus[t]+s[t,n];
}
```

Kõigi lühimate teede leidmine – Floyd-Warshalli algoritm

```
real D[tippude_arv][tippude_arv] =
{ Naabrusmaatriks servapikkustega,
kus puuduva serva kohal on ∞ }
for i = 1 to tippude_arv
    for j = 1 to tippude_arv
        for k = 1 to tippude_arv
            if (D[j][i]+D[i][k]<D[j][k])
                D[j][k]=D[j][i]+D[i][k];
```

Floyd-Warshalli algoritm - näide



Floyd-Warshalli algoritm - ülesanne

- http://community.topcoder.com/stat?c=problem_statement&pm=2356&rd=4740

Dijkstra - ülesanne

- http://community.topcoder.com/stat?c=problem_statement&pm=2449&rd=5073

Euleri ahel ja tsükkel

- **Marsruut** = tippude jada, mille iga kaks järjestikust tippu on ühendatud kaarega
- **Ahel** = marsruut, mille kõik kaared on erinevad
- **Euleri ahel** = ahel, mis läbib graafi kõik kaared
- **Tsükkel** = ahel, mille algtipp ja lõpptipp on samad
- **Euleri tsükkel** = Euleri ahel, mis on tsükkel ☺
- Sidusal graafil leidub Euleri ahel **parajasti siis**, kui ülimalt kaks tema tippudest on paaritud
- Sidusal graafil leidub Euleri tsükkel **parajasti siis**, kui kõik tema tipud on paaristipud

Euleri tsükkel – Hierholzeri algoritm

- Choose any starting vertex *v*, and follow a trail of edges from that vertex until returning to *v*. It is not possible to get stuck at any vertex other than *v*, because the even degree of all vertices ensures that, when the trail enters another vertex there must be an unused edge leaving *w*. The tour formed in this way is a closed tour, but may not cover all the vertices and edges of the initial graph.
- As long as there exists a vertex *v* that belongs to the current tour but that has adjacent edges not part of the tour, start another trail from *v*, following unused edges until returning to *v*, and join the tour formed in this way to the previous tour

Euleri tsükkel - ülesanne

- http://uva.onlinejudge.org/index.php?option=onlinejudge&page=show_problem&problem=995

Toesepuu (aluspuu)

- **Puu** = sidus graaf, milles puuduvad tsüklid
- **Graafi aluspuu** = alamgraaf, mis on puu ja mis sisaldab kõiki selle graafi tippe

Minimaalse kaaluga aluspuu – Kruskali algoritm

- Alusta tühjast puust.
- Kuni lisatud on vähem kui n-1 serva, lisa lühim serv nende hulgast, mis ei tekita eelnevatega koos tsüklit.

Minimaalse kaaluga aluspuu – Prim'i algoritm

- Vali suvaline tipp toesepuu alguseks.
- Kuni leidub veel tippe, mis puusse pole lisatud, leia lühim serv, mis seob mõnda puu tippu seal mitteolevaga,
- Lisa see serv puusse.

- Väga sarnane Dijkstra algoritmiga!

Minimaalne aluspuu - ülesanne

- Antud N linna oma täisarvuliste koordinaatidega
- Koostada minimaalse pikkusega elektriliinide võrk, mis kõiki linnu ühendab, kui linnu saab ühendada ainult otseteid pidi

Maksimaalne voog

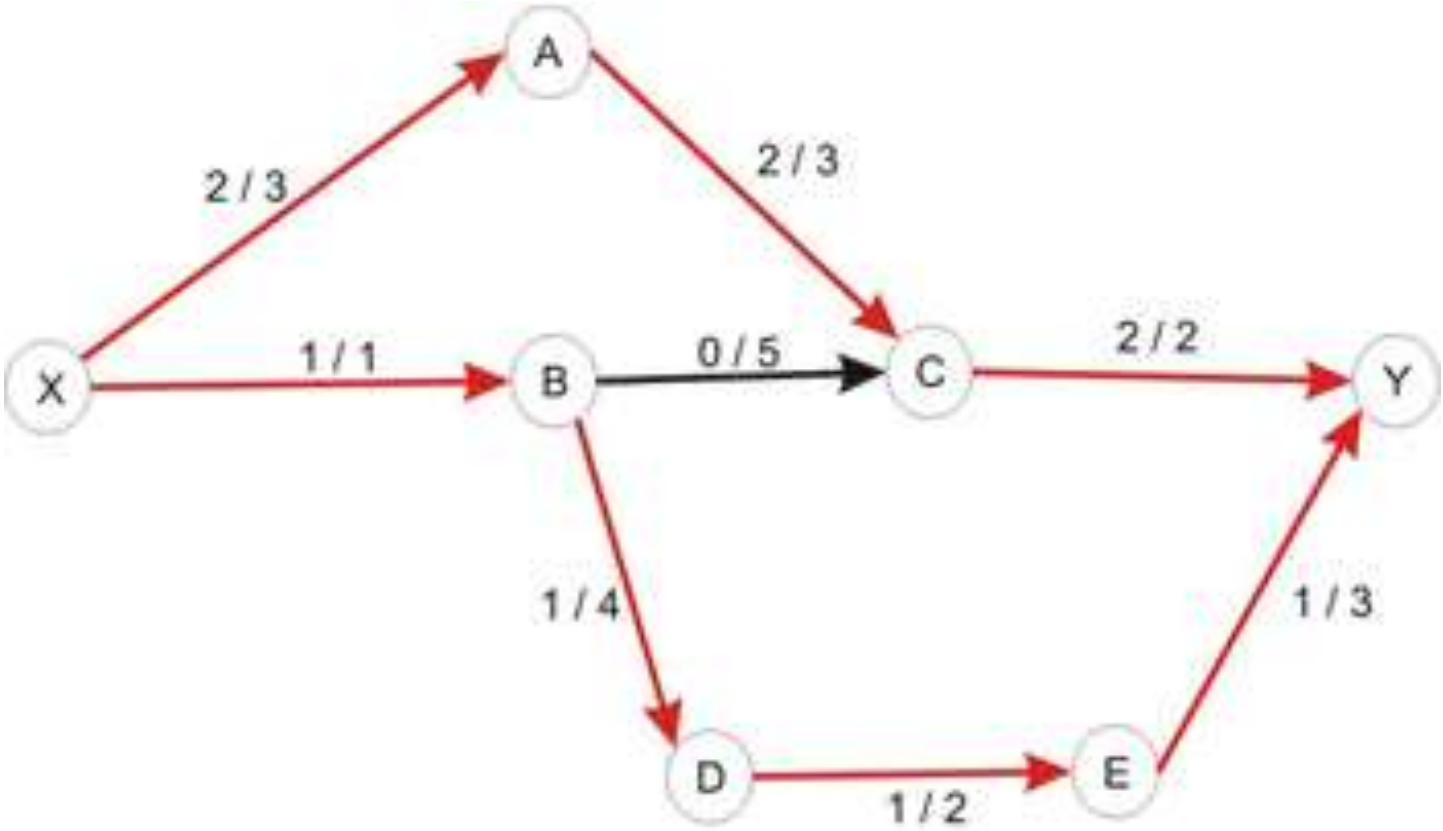


Figure 1a - Maximum Flow in a network

1. Vaatleme suvalist voogu

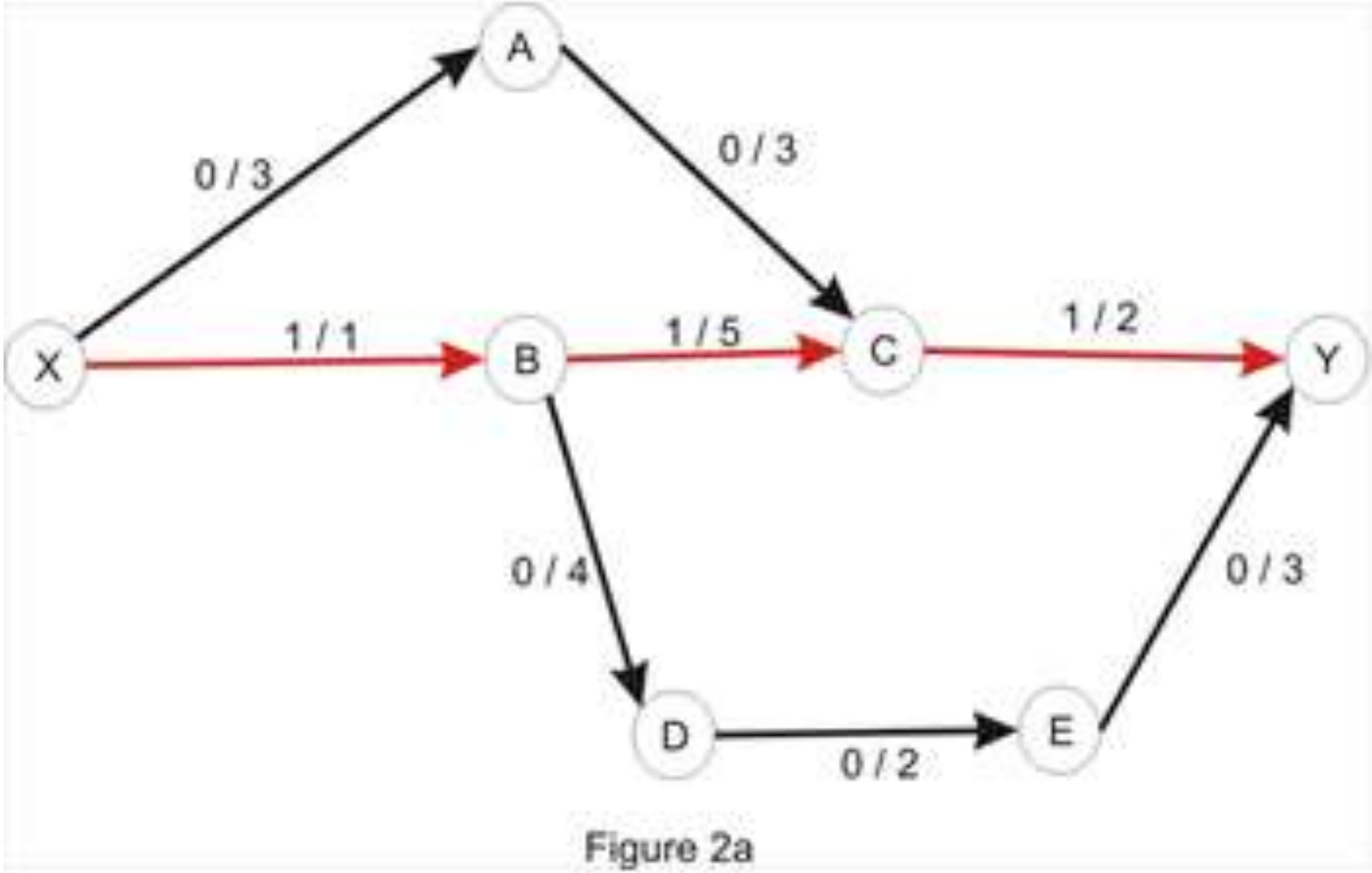


Figure 2a

2. Konstrueerime täiendavad servad

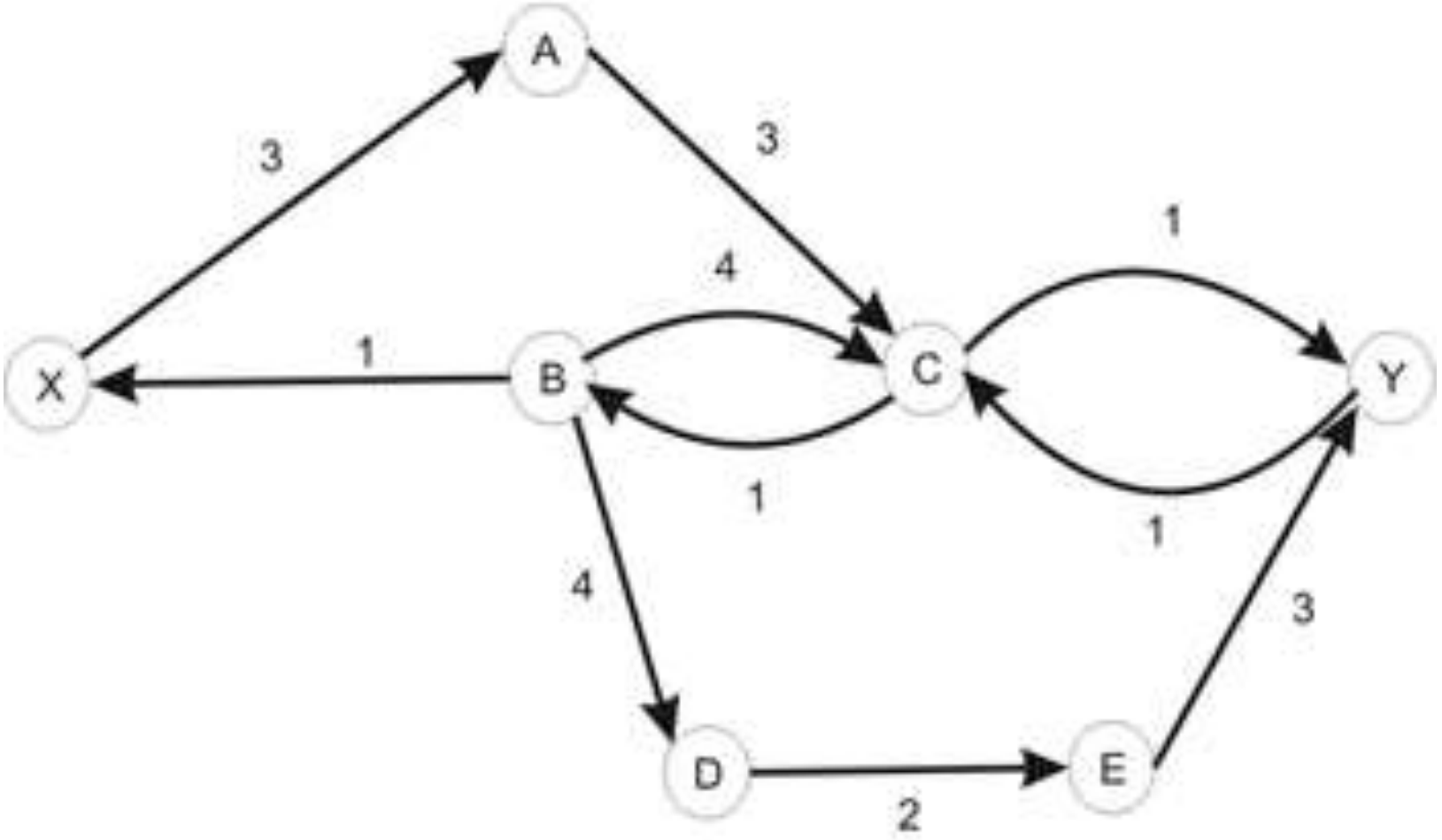


Figure 2b - The residual network of the network in 2a

Kordame mõne teise ahela jaoks

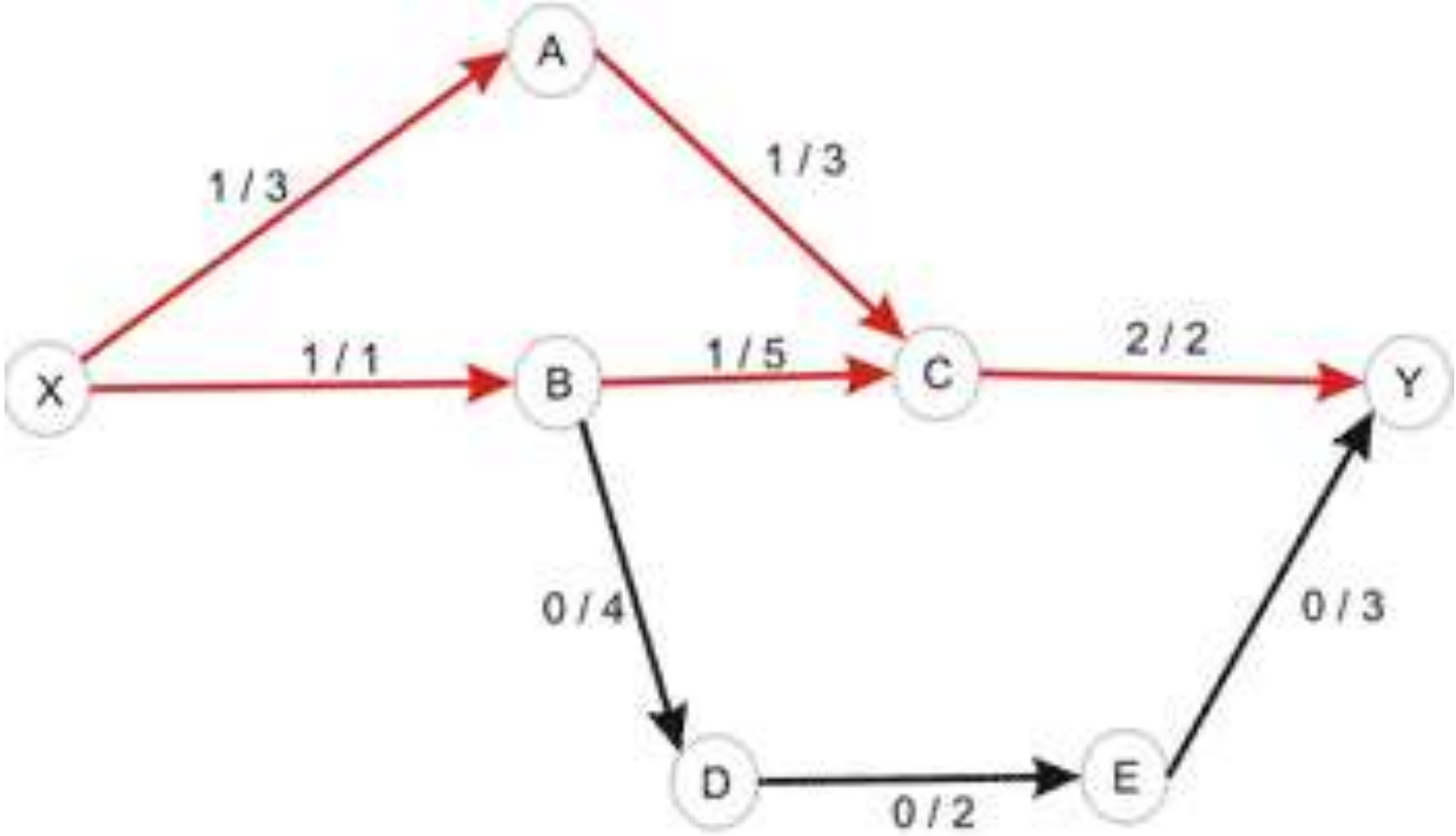


Figure 3a

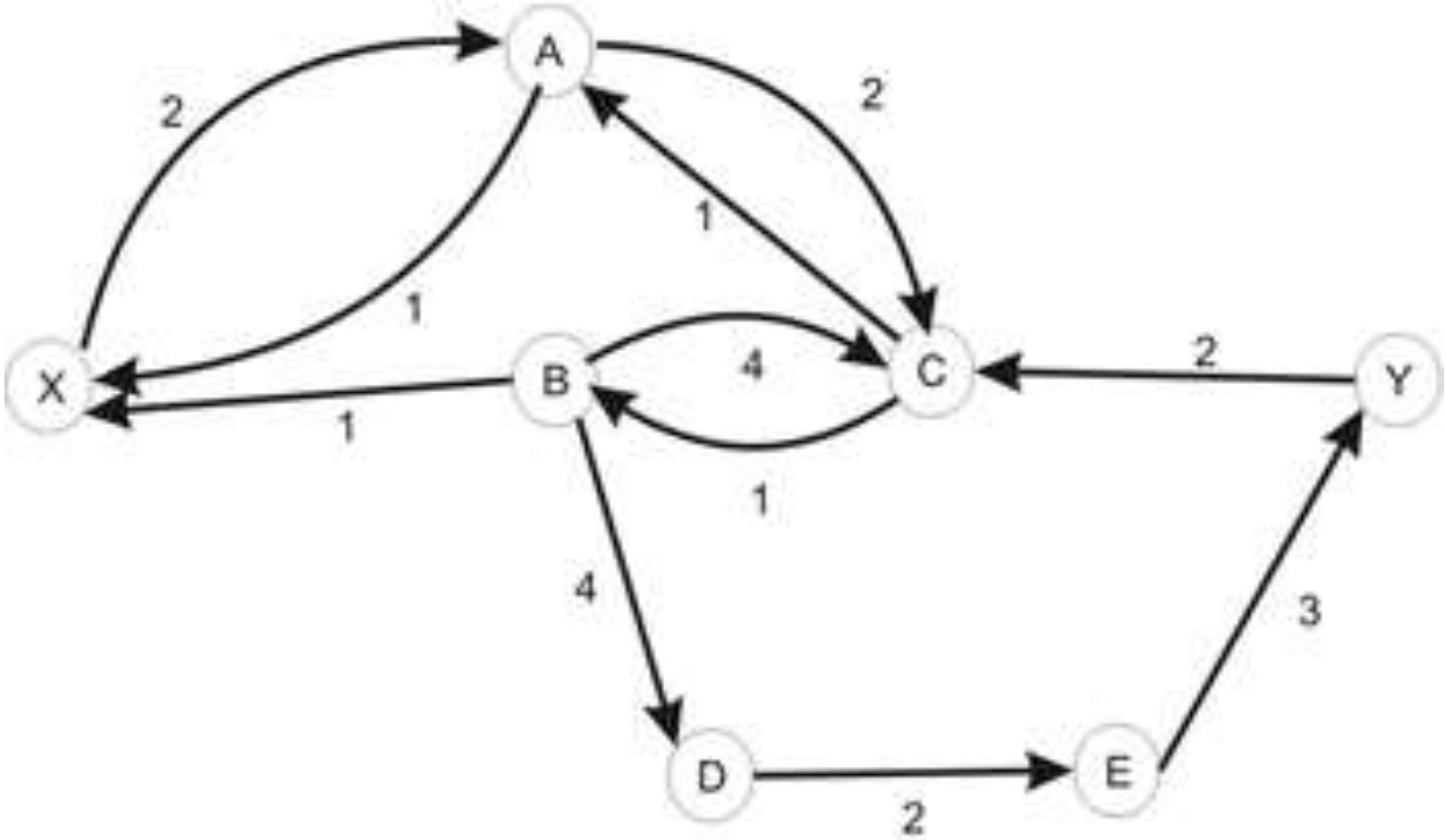


Figure 3b - The residual network of the network in 3a

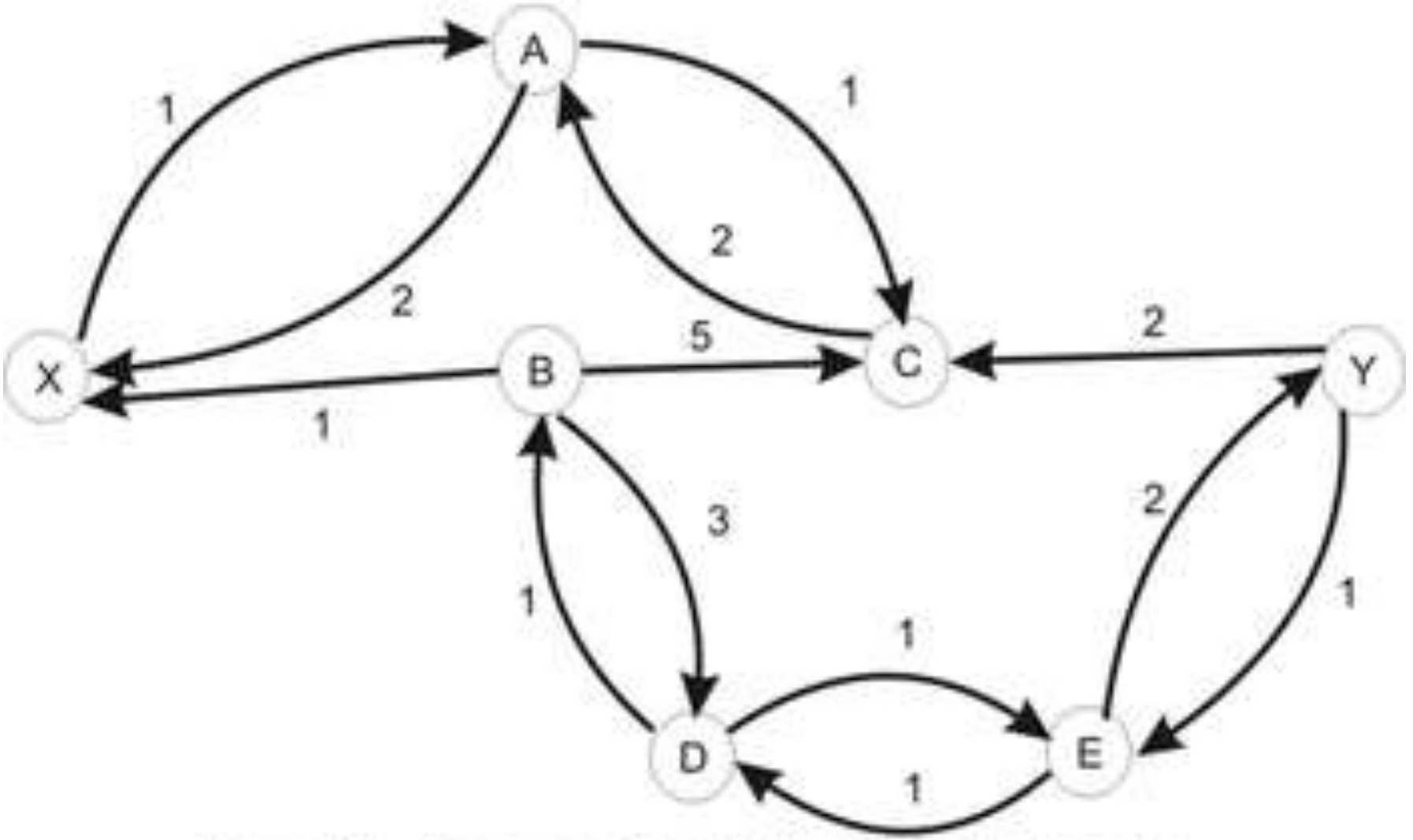


Figure 1b - The residual network of the network in 1a

Ford-Fulkersoni algoritm

1. Alustame null-vooga
2. Kuni leidub tee algusest lõppu
    1. Suurendada voogu ühe võrra
    2. Täiendada graafi

```
int max_flow()
result = 0
  while (true)
    // the function find_path returns the path capacity of the augmenting path found
    path_capacity = find_path()
    // no augmenting path found
    if (path_capacity == 0) break
    else result += path_capacity
  end while
  return result
```

Tee leidmine – laiuti otsing

```
int bfs()
  queue Q
  push source to Q
  mark source as visited
  // keep an array from: from[x] is the previous vertex visited in the shortest path from the source to x;
initialize from with -1 (or any other sentinel value)
  while Q is not empty
    where = pop from Q
    for each vertex next adjacent to where
      if next is not visited and capacity[where][next] > 0
        push next to Q
        mark next as visited
        from[next] = where
        if next = sink
          break
    end for
  end while
  // compute the path capacity
  where = sink, path_cap = infinity
  while from[where] > -1
    prev = from[where] // the previous vertex
    path_cap = min(path_cap, capacity[prev][where])
    where = prev
  end while
  // we update the residual network; if no path is found the while loop will not be entered
  where = sink
  while from[where] > -1
    prev = from[where]
    capacity[prev][where] -= path_capacity
       capacity[where][prev] += path_capacity
    where = prev
  end while
  // if no path is found, path_cap is infinity
  if path_cap = infinity
    return 0
  else return path_cap
```

Mitu algust ja lõppu

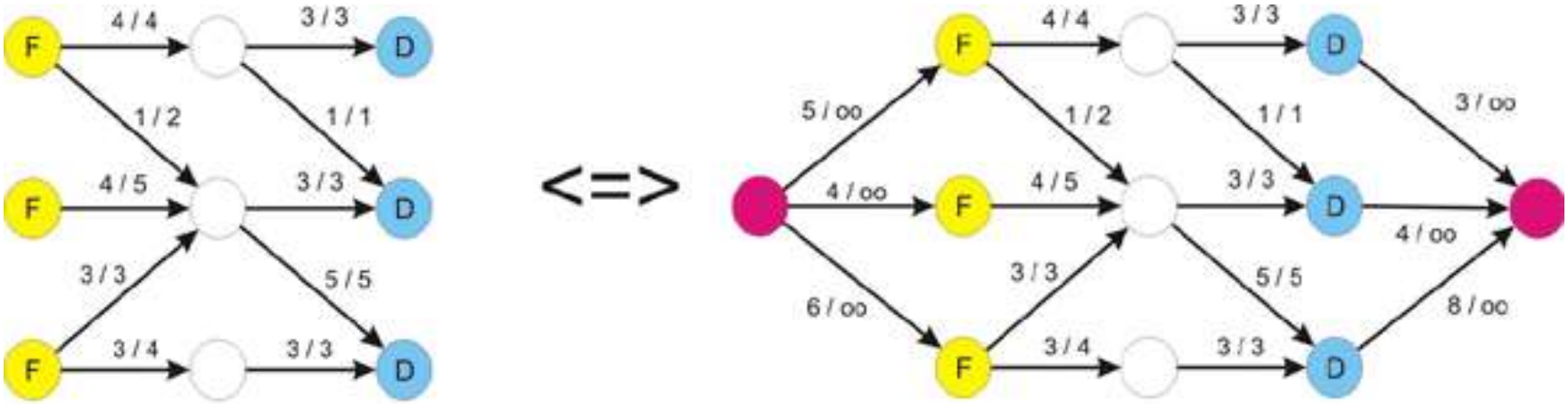


Figure 6 - Reduction of a multiple-source / multiple-sink max-flow problem

Kui tippudel on ka kaalud

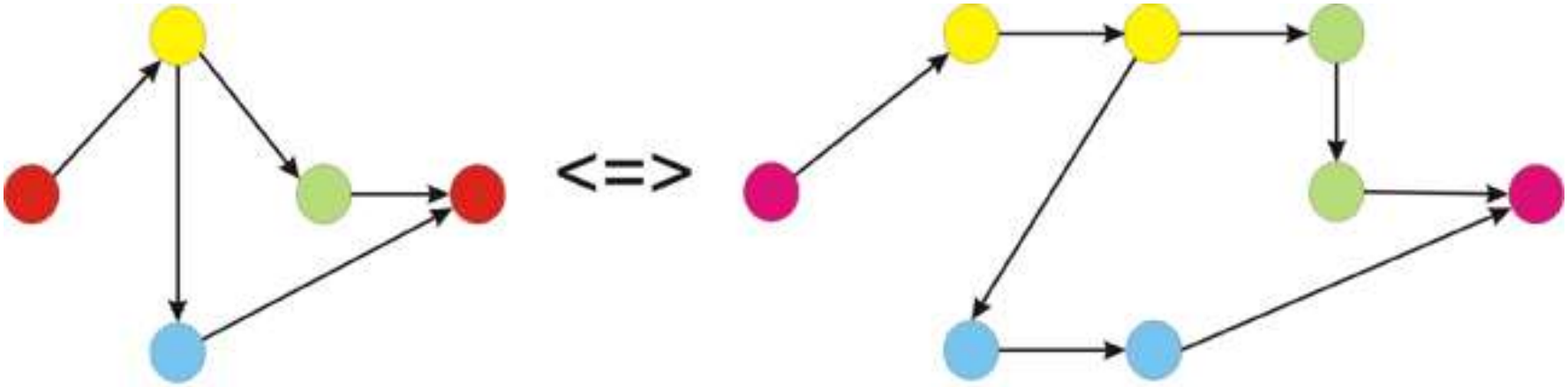


Figure 7 - Eliminating vertex-capacities

Max flow - ülesanne

- Suunatud graafi kohta on antud igasse tippu sisenevate servade arv ning igast tipust väljuvate servade arv. Iga tipupaari vahel tohib olla ainult üks serv. Konstrueerida täielik graaf.
- Näide: (2, 1, 1, 1) ja (1, 2, 1, 1).

Ülesanne 2

- Meil on hulk töötajaid ja hulk erinevaid töid, mis tuleb ära teha. Iga töötaja suudab teha mingit hulka töid.
- Kas meil on võimalik kõik tööd ära teha?

Ülesanne 3

- Meil on antud hulk kuubikujulisi täheklotse
- Kas me saame nendest kokku panna etteantud sõna?