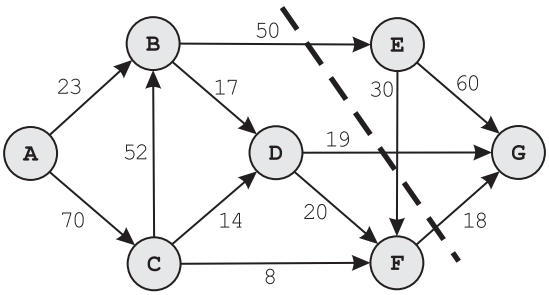
6.11.1. *Minimum cut*

Min-cut teorem tvrdi da je maksimalan tok kroz mrežu jednak minimalnom kapacitetu koji se mora ukloniti iz mreže kako ne bi bilo toka od izvora do odvoda. Drugim riječima: minimalna suma težine bridova, koje je potrebno ukloniti iz grafa da se ne bi moglo doći od izvora do odvoda, jednaka je problemu *Network flow*. Znači:



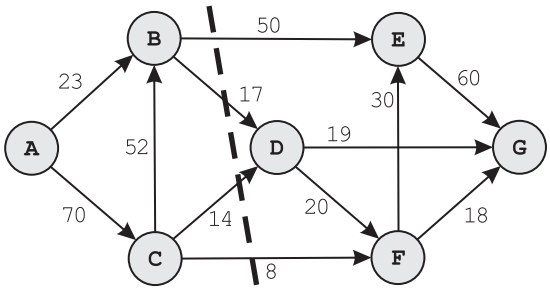
Slika 6.27: *Minimum cut*

```
Minimum cut == Network flow
```

Na slici 6.27 crtkana linija označava *Minimum cut*. U tom slučaju će *Minimum cut* presjeći bridove BE, DG i FG (bridovi idu s lijeve strane crtkane linije prema desnoj, zato ne brojimo brid EF). Tada se više ne može od čvora A doći do čvora G. Primijetite da je suma težina bridova BE, DG i FG jednaka `50+19+18 = 87`, a toliko iznosi i *Network flow* (od čvora A do čvora G) za prikazan graf.

Ako okrenemo brid EF, onda je *Network flow* 89, a *Minimum cut* također. Ta situacija prikazana je na grafu sa slike 6.28. *Minimum cut* postižemo presjecanjem bridova BE, BD, CD i CF. Suma težina tih bridova je `50+17+14+8 = 89`.

Vrijednost za *Minimum cut* može se lako izračunati najvećim tokom kroz graf. Utvrditi točno o kojim se bridovima radi može biti vrlo teško. Jedan način da se to napravi jest pokušati izbaciti sve kombinacije bridova koje Ford–Fulkersonov algoritam postavi na `0` i „poplaviti" (engl. *flood fill*) izvor kako bi se provjerilo je li graf i dalje povezan.



Slika 6.28: *Minimum cut*

Drugi način za utvrđivanja koji bridovi čine *Minimum cut* jest pokušati izbaciti svaki brid (od lakših prema težim) i ako se *Network flow* smanji za težinu izbačenog brida, znamo da je taj brid dio *Minimum cuta* i valja nastaviti bez njega.

Zadatak:

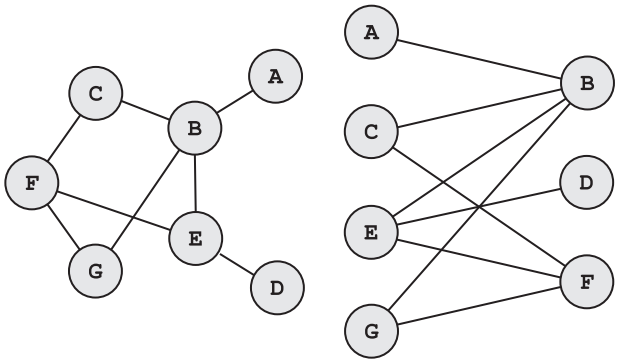
Poslovna računala su povezana u mrežu mrežnim kabelima. Između nekih parova računala nalazi se dvosmjerni mrežni kabel. Ako iskopčamo neki mrežni kabel stvaramo određen poslovni gubitak. Računalo X je jako bitno i do njega ne smije doći virus koji se nalazi u računalima Y1, Y2, ..., Yn. Koliki je minimalni poslovni gubitak koji je potrebno pretrpjeti kako bi se zaštitilo računalo X od virusa?

Rješenje:

Mreža računala predstavlja graf nad kojim moramo napraviti *Minimum cut* od računala Y1, Y2, ..., Yn do X. Računala Y1, Y2, ..., Yn su izvori, a X je odvod. Budući da ima više izvora, moramo dodati virtualan izvor s bridovima beskonačne težine do izvora Y1, Y2, ..., Yn. Graf je neusmjeren pa svaki brid od čvora A do čvora B težine C tumačimo kao dva usmjerena brida, od A do B i od B do A, oba težine C. Nad grafom provedemo Ford-Fulkersonov algoritam i ispišemo rješenje.

6.11.2. *Maximum bipartite matching*

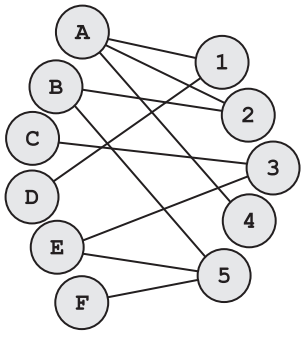
Bipartitivni graf ili bigraf je graf čiji se čvorovi mogu podijeliti u dvije skupine tako da svaki brid grafa ima jedan čvor u prvoj skupini, a drugi u drugoj. Na slici 6.29 je dva puta prikazan isti bipartitivan graf. Čvorove možemo podijeliti u dvije skupine V1 = {A, C, E, G} i V2 = {B, D, F}, tako da nema brida koji povezuje dva čvora iz iste skupine.



Slika 6.29: Bipartitivan graf

Najveće moguće uparivanje (engl. *maximum matching*) u bipartitivnom grafu je problem uparivanja što većeg broja čvorova iz V1 s čvorovima iz V2.

Pogledajmo graf sa slike 6.30. Čvorovi nazvani slovima predstavljaju dečke, a čvorovi nazvani brojkama cure. Brid između dvaju čvorova postoji ako su si te dvije osobe međusobno simpatične. Graf je bipartitivan jer su sve relacije heteroseksualne. Postavlja se pitanje: Koliko se najviše parova može upariti ako svaki dečko može biti uparen sa samo jednom curom, a svaka cura sa samo jednim dečkom?

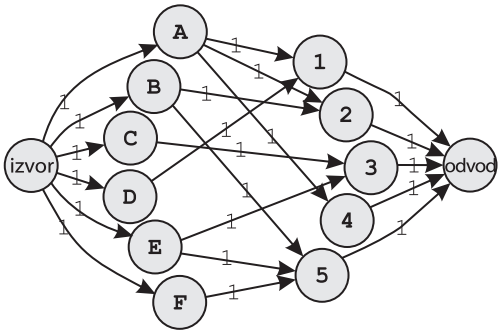


Slika 6.30: Primjer grafa

Ako uparimo A i 1, onda će čvor D ostati sam. Ako uparimo A i 2, onda će čvor B morati biti uparen s 5, pa će čvor F ostati sam. Dakle, uparimo A s 4, D s 1, B s 2, F s 5, i npr. E s 3. Time je C ostao bez para.

Najveće moguće uparivanje je 5, jer se najviše 5 parova može upariti.

Navedeni problem može se riješiti pomoću *Network flowa* (slika 6.31). Sve bridove usmjerimo od slova prema brojkama i damo im protočnost 1. Dodamo čvor koji predstavlja izvor i od njega do svakog čvora nazvanog slovom spojimo brid protočnosti 1. Time omogućujemo dečkima da koriste brid samo prema jednoj curi. Sve čvorove koji su označeni brojkom povežemo prema novododanom čvoru koji predstavlja odvod i damo im protočnost 1. Time omogućimo svakoj curi da se poveže samo s jednim dečkom. Nakon što napravimo *Network flow*, kao rezultat dobit ćemo najveći broj parova koji se mogu upariti, a gledajući koji bridovi su postavljeni na 0 možemo saznati tko je s kime uparen.



Slika 6.31: *Network flow*

Što zapravo *Network flow* radi? Recimo da prvo pronađe put `Izvor-A-1-Odvod`, pa zato postavi protočnost brida od A1 na 0, a brida 1A na 1. To je isto kao da smo jednostavno brid A1 okrenuli tako da imamo brid 1A. Sljedeći put se pronađe npr. put `Izvor-B-2-Odvod`, pa se brid B2 pretvori u 2B. Sljedeći put koji se pronađe je npr. `Izvor-C-3-Odvod`, pa je sada brid C3 postao 3C. Sljedeći put je npr. `Izvor-D-1-A-2-B-5-Odvod`. U tom putu koristimo brid 1A (nastao okretanjem A1) i brid 2B (nastao okretanjem B2). Brid D1 se pretvara u 1D (to znači da je D uparen s 1). Brid 1A se vraća u A1 (znači A i 1 više nisu upareni). Brid A2 postaje 2A (znači da je A uparen s 2). Brid 2B postaje B2 (znači da B više nije uparen s 2). Brid B5 postaje 5B (znači B je uparen s 5). Zapravo smo u ovoj iteraciji *Network flowa* preusmjerili A s 1 na 2, B s 2 na 5, a D povezali s 1. Sljedeći put koji Ford-Fulkersonov algoritam pronađe je npr. `Izvor-E-5-Odvod`, pa se brid E5 pretvori u 5E (znači da je E uparen s 5). Sada više nema putova i Ford-Fulkersonov algoritam završava. Ukupan *Network flow* je 5, što znači da se najviše 5 parova može upariti. Umjesto zadnjeg navedenog puta Ford-Fulkersonov algoritam je mogao pronaći i `Izvor-F-5-Odvod`. Različiti putovi bi različito povezali parove, ali bi broj parova i dalje bio 5.

Primijetite da se ponašanje Ford-Fulkersonovog algoritma za problem najvećeg mogućeg uparivanja na bipartitivnom grafu može simulirati i rekurzijom te brže pronaći.

Riješimo rekurzijom sljedeći zadatak:

Unosi se n1 (broj čvorova iz prvog skupa), n2 (broj čvorova iz drugog skupa) i m, zatim m bridova opisanih s po dva broja a i b. Čvor a (0 <= a < n1) iz prvog skupa je povezan bridom s čvorom b (0 <= b < n2) iz drugog skupa. Ispiši koliko najviše parova čvorova se može povezati (nekim od postojećih bridova) tako da je svaki čvor povezan s najviše jednim čvorom.

Primjer unosa:

```
6 5 10
0 0 0 1 0 3 1 1 1 4
2 2 3 0 4 2 4 4 5 4
```

Unos odgovara prije navedenom grafu.

Odgovarajući ispis:

```
5
  Slijedi dodani ispis:
A + 4
B + 2
C + 3
D + 1
E + 5
```

Rješenje:

```
01. #include <vector>
02. #include <iostream>
03. using namespace std;
04.
05.   vector<vector<int> > graf;
06.   vector<int> spojenSa;
07.   vector<int> bio;
08.   int n1,n2,m;
09.
10. bool DFS(int x) {
11.   bio[x]=1; // kako ne bi pozvali rekurziju za isti cvor
12.   int tko;
13.   for (int i=0;i<graf[x].size();i++) {
14.     tko=graf[x][i];
15.     if (spojenSa[tko]==-1 || (!bio[spojenSa[tko]]
16.                               && DFS(spojenSa[tko]))) {
17.       spojenSa[tko]=x;
18.       return 1;
19.     }
20.   }
21.   return 0;
22. }
23.
24. int main() {
25.   cin >> n1 >> n2 >> m;
26.   vector<int> vi;
27.   graf.insert(graf.begin(),n1,vi);
28.   spojenSa.insert(spojenSa.begin(),n2,-1); // -1 znaci slobodan
29.   int a,b;
30.   for (int i=0;i<m;i++) {
31.     cin >> a >> b; graf[a].push_back(b); }
32.
```

```
33.   int rjesenje=0;
34.   for (int i=0;i<n1;i++) {
35.     bio.clear();
36.     bio.insert(bio.begin(),n1,0);
37.     rjesenje+=DFS(i);
38.   }
39.   cout << rjesenje << endl;
40.   cout << "  Slijedi dodani ispis: " << endl;
41.   for (int i=0;i<n2;i++)
42.     cout << (char)(i+'A') << " + " << spojenSa[i]+1 << endl;
43.   return 0;
44. }
45.
```

Pojašnjenje:

Od 25. do 31. linije je unos. U vektor `graf` zapisujemo popis susjeda. Vektor `spojenSa` je dimenzije `n2` i bilježi `-1` ako nitko nije spojen s odgovarajućim čvorom, dok inače bilježi broj čvora. U 33. liniji inicijaliziramo rješenje na `0`. U 34. liniji prolazimo kroz sve čvorove iz skupa veličine `n1` i svaki uparujemo (37. linija). `DFS` vraća `1` ako se čvor uspije spojiti, a inače `0`. Prije poziva `DFS` prvo postavimo u 35. i 36. liniji sve elemente vektora `bio` (veličine `n1`) na `0`, kako bi rekurzija mogla pamtiti za koje čvorove je pozvana i ne pozivati se ponovno za isti čvor. `DFS` prima čvor iz skupa veličine `n1` za koji se poziva. U 11. liniji postavimo da smo bili u čvoru `x`. U 14. liniji varijabla `tko` je neki čvor iz skupa veličine `n2` dohvatljiv bridom. 15. i 16. linija koda su dosta komplicirane te odgovaraju na pitanje može li se `x` spojiti s `tko`. Ako se može, onda u 17. liniji bilježimo s kime je tko spojen i vraćamo `1`. U 15. liniji prvo provjeravamo je li `tko` slobodan: `spojenSa[tko]==-1`. Ako je to istinito, onda se ostatak `if` naredbe ne izvršava. Ako to nije istinito, treba provjeriti može li se onaj s kime je `tko` spojen prespojiti na neki drugi čvor. To radimo samo ako rekurzija nije već pozvana za čvor s kime je `tko` spojen. `DFS` će pokušati čvor `spojenSa[tko]` prespojiti na neki drugi čvor različit od `tko` jer je `spojenSa[tko]==-1` neistinit, kao i `!bio[spojenSa[tko]]`. Navedeni rekurzivni poziv je pomalo kompliciran pa pokušajte još malo sami razmisliti o tome što se točno događa.

Složenost navedene implementacije je u najgorem slučaju `O(V*E)`, točnije `O(n1*E)`. Rekurziju pozivamo `n1` puta, a rekurzija pamti u kojim čvorovima je bila, tako da u najgorem slučaju prođe preko svih `E` bridova.

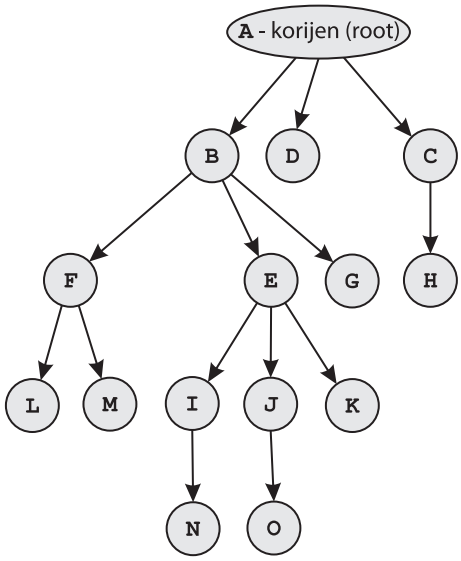
7. Algoritmi – stabla i strukture

7.1. O stablima

Naučili smo što su stabla kao grafovi. Za njih vrijedi:

- ne sadrže ciklus,
- imaju V-1 bridova,
- povezani su,
- svaki je brid most i
- svaka dva čvora su povezana jedinstvenim putem.

Stabla kao strukture podataka su iste strukture kao stabla-grafovi, međutim ne gledamo na njih kao na graf i crtamo ih na karakterističan način kako je prikazano na slici 7.1.



Slika 7.1: Stablo

Naučimo neke pojmove:
Na slici 7.1 prikazano je stablo s korijenom A. **Korijen** (engl. ***root***) je najviši čvor u stablu. U stablu svaki čvor ima jednog roditelja. Pojam **dijete** i **roditelj** se određuje za svaki čvor pojedinačno. Npr. za stablo desno čvor F ima dvoje djece (L i M) i roditelja B. Čvor D nema djece, a roditelj mu je A. Čvor J ima jedno dijete (O) i roditelja E. **Potomci** su djeca i djeca djece, a preci su roditelji i roditelji roditelja. **Brat** nekom čvoru je čvor s kojim dijeli roditelja. C ima braću B i D. **Podstablo** je dio stabla. Svaki čvor zajedno sa svim svojim potomcima čini podstablo u kojemu je on korijen. Npr. čvor E je korijen podstabla s čvorovima E, I, J, K, N i O. Čvorovi C i H čine podstablo s korijenom C. **Listovi** stabla su čvorovi koji nemaju djece. U navedenom stablu su listovi L, M, N, O, K, G, D i H. **Unutarnji čvorovi** su svi čvorovi koji nisu listovi. **Stupanj** nekog čvora je broj djece tog čvora (broj podstabala). U navedenom stablu čvorovi A, B i E imaju stupanj 3, a čvorovi I, J i C stupanj 1. Listovi imaju stupanj 0. Samo F ima stupanj 2 jer ima dvoje djece. **Stupanj stabla** je najveći stupanj od svih čvorova u tom stablu. Stupanj navedenog stabla je 3.

Razina čvora je udaljenost od korijena. U navedenom stablu je A na razini 1. B, D i C su na razini 2. F, E, G i H su na razini 3. L, M, I, J i K su na razini 4. N i O su na razini 5. Za razinu čvora se kaže i dubina čvora, iako neke literature razdvajaju ta dva pojma, pa dubinu smatraju najvećom razinom nekog čvora u stablu.

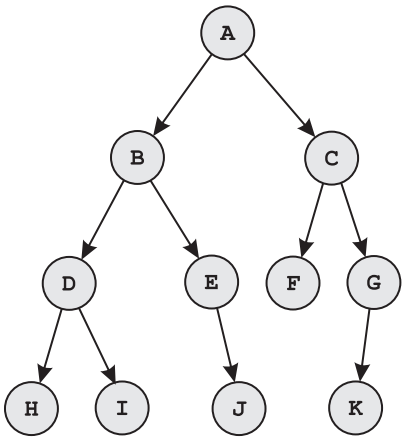
7.1.1. Obilazak stabla

Stablo u kojem svaki čvor ima stupanj 0, 1 ili 2 nazivamo **binarnim stablom**. Na slici 7.2 prikazano je binarno stablo. Binarna stabla su stabla koja nas najviše zanimaju. **Obilazak stabla** je redoslijed kojim obilazimo sve čvorove stabla. Binarna stabla se mogu obići na tri standardna načina, ovisno o položaju čvora u odnosu na podstabla. To su:

inorder: lijevo podstablo, pa korijen podstabla, pa desno podstablo,

preorder: korijen podstabla, pa lijevo podstablo, pa desno podstablo,

postorder: lijevo podstablo, pa desno podstablo, pa korijen podstabla.



Slika 7.2: Binarno stablo

Inorder obilazak stabla prikazanog desno jest: H, D, I, B, E, J, A, F, C, K, G. Takav obilazak postižemo pozivanjem sljedeće rekurzije za korijen stabla:

```
rek (čvor)
  ako postoji lijevo dijete: rek (lijevo dijete);
  ispiši čvor;
  ako postoji desno dijete: rek (desno dijete);
```

Preorder obilazak jest: A, B, D, H, I, E, J, C, F, G, K. Takav obilazak postižemo pozivanjem sljedeće rekurzije za korijen stabla:

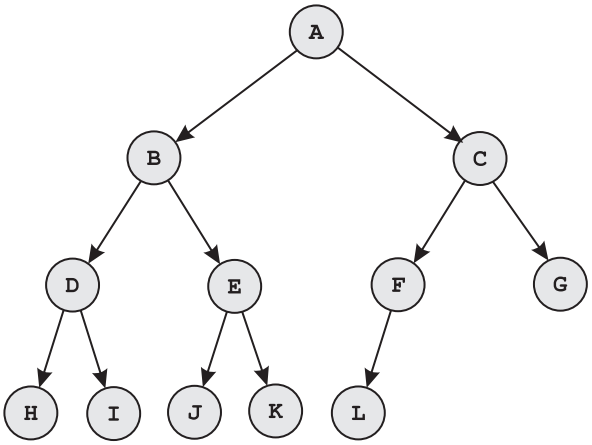
```
rek (čvor)
  ispiši čvor;
  ako postoji lijevo dijete: rek (lijevo dijete);
  ako postoji desno dijete: rek (desno dijete);
```

Postorder obilazak jest: H, I, D, J, E, B, F, K, G, C, A. Takav obilazak postižemo pozivanjem sljedeće rekurzije za korijen stabla:

```
rek (čvor)
  ako postoji lijevo dijete: rek (lijevo dijete);
  ako postoji desno dijete: rek (desno dijete);
  ispiši čvor;
```

7.1.2. Potpuno binarno stablo

Potpuno binarno stablo ima sve čvorove na svim razinama, osim na zadnjoj razini. Takvo stablo je izrazito zanimljivo za razne algoritme. Potpuno binarno stablo ima na svakoj sljedećoj razini (osim zadnje) dvostruko više čvorova. Na prvoj razini je samo korijen, na drugoj su 2 čvora, na trećoj 4, na četvrtoj 8, a na x-toj 2^{x-1} čvorova.



Slika 7.3: Potpuno binarno stablo

Budući da za potencije broja 2 vrijedi $2^0+2^1+2^2+23+...+2^{x-1} = 2^x-1$, znamo da svaka sljedeća razina ima za jedan više čvorova od svih razina prije. To ujedno znači da potpuno binarno stablo s x čvorova ima $\log_2(x+1)$ razina.

Potpuno binarno stablo može se na specifičan način zapisati u memoriju u obliku polja (ili vektora):

indeks u polju:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
zapisani čvor:	-	A	B	C	D	E	F	G	H	I	J	K	L	prazno	

Usporedite graf sa slike 7.3 i navedeno polje. U polju se na indeksu x nalazi roditelj čvorova s indeksima 2x i 2x+1. Na indeksu x nalazi se dijete čvora s indeksa x/2 (cjelobrojno podijeljeno). Dakle, budući da je E na indeksu 5, njegova djeca J i K su na indeksima 10 i 11. Isto tako znamo da je roditelj čvora L (koji je na 12. indeksu) na indeksu 12/2 = 6. Slično je roditelj od K (11. indeks) na indeksu 11/2 = 5. Primijetite da kod zapisa potpunog binarnog stabla u polje ne koristimo nulti indeks.

7.2. *Heap* (hr. gomila)

Heap je uređeno potpuno binarno stablo. Ta struktura podataka postoji u STL-u i zove se `priority_queue`. Proučimo kako funkcionira i čemu služi.

Heap je struktura podataka koja može u složenosti O(1) reći koji je najveći (ili najmanji, ovisno kako je implementiramo) elementu u *heapu*. Dok u složenosti O(log n) može izbaciti najveći ili ubaciti novi element u *heapu*.

U tom potpunom binarnom stablu roditelj uvijek mora biti veći od svoje djece (ili uvijek manji).