Reţele de transport

prof. *Constantin Gălăţan*

În teoria grafurilor, o **reţea de transport** este un graf orientat, în care fiecare arc:

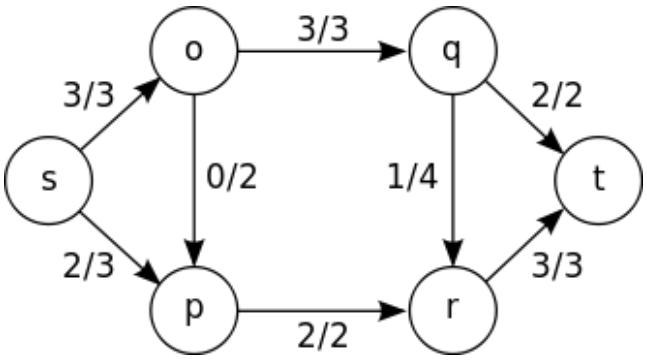
- are o capacitate (limită superioară)
- primeşte un flux care nu poate depăşi capacitatea sa.

Fluxul este rata (debitul) cu care un material poate să curgă prin reţea.

Exemple:

Reţea	Noduri	Arce	Flux
Transport urban	intersecţii de străzi	străzi	vehicule
Circuite electrice	nod reţea electrică	conductori de curent	curent electric
Circuite hidraulice, sau hidrografice	Rezervoare, pompe, staţii, lacuri	Conducte, canale, râuri	apă, alte fluide
Comunicaţii	Sateliţi, antene, staţii emisie	Cabluri, fibre optice, unde electromagnetice	voce, video, pachete de unde.
Financiară	Companii	Tranzacţii	bani

Într-o reţea de transport există un nod de intrare în reţea: **`s`** şi unul de ieşire din reţea: **`t`**, astfel încât cantitatea de flux care intră în reţea prin **`s`** este egală cu cantitatea de flux care iese din reţea din **`t`**.



Fluxul nu se poate acumula în noduri, adică suma cantităţilor de flux care intră într-un nod este egală cu suma cantităţilor de flux care ies din nod.

A. Flux maxim în reţele de transport

Definiţii:

- **Capacitate reziduală**

  Fie **`c[x][y]`** capacitatea unui arc de la nodul **`x`** la nodul **`y`** şi fie **`f[x][y]`** fluxul net pe arcul **`x->y`**

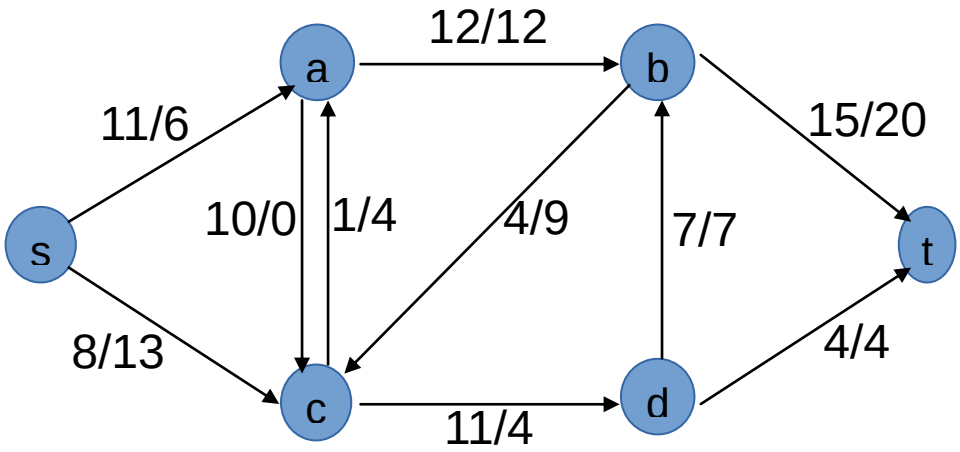
Atunci **`cr[x][y] = c[x][y] – f[x][y]`** se numeşte capacitatea reziduală a arcului **`x`**→**`y`**.

- **Reţea reziduală**
  Fie **`c[x][y]`** capacitatea unui arc de la nodul **`x`** la nodul **`y`** şi fie **`f[x][y]`** fluxul net pe arcul **`x->y.`** Atunci **`cr[x][y] = c[x][y] – f[x][y]`** şi
  `cr[y][x] = f[x][y]`.
- **Drum de ameliorare** (*drum de creştere*)
  Este un drum simplu de la **`s`** la **`t`** în reţeaua reziduală. Pe un drum de ameliorare fiecare arc (**`x`**, **`y`**) admite o o creştere pozitivă a fluxului, în limita constrângerii de capacitate.

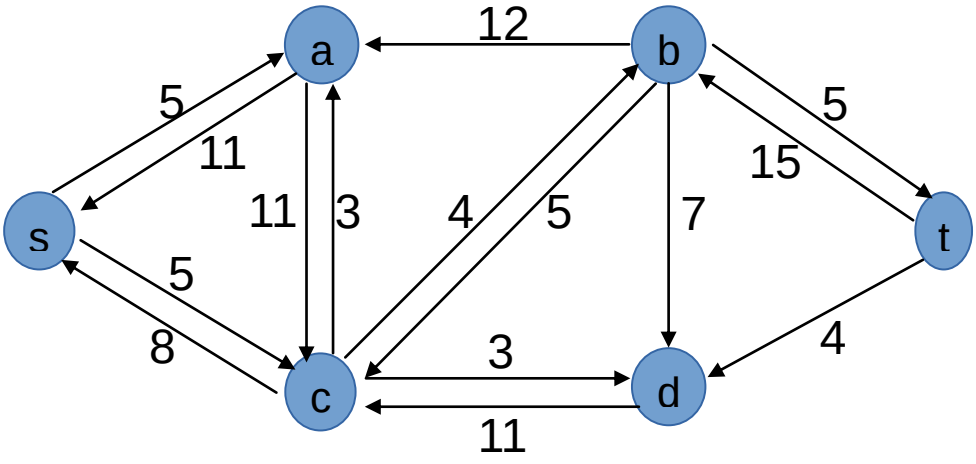
- **Tăietură (cut)**
  O tăietură (**`s`** ,**`t`**) a unei reţea de transport este o partiţie a mulţimii **`V`** a vârfurilor reţelei grafului în două submulţimi **`S`** şi **`T = V – S`**, asfel încât **`s`** aparţine **`S`** şi **`t`** aparţine **`T`**.

- **Tăietură minimă (***minimum cut***)**
  este o tăietură a cărei capacitate este minimă în raport cu toate tăieturile reţelei.

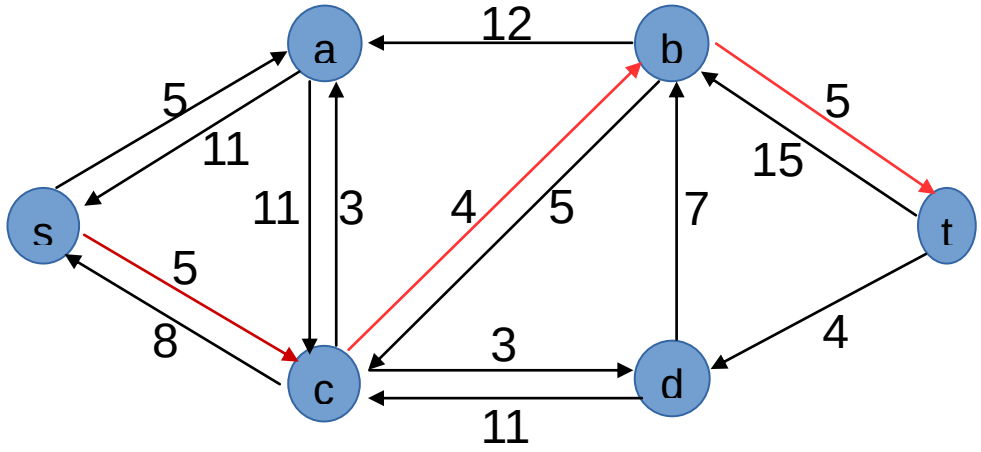
  Exemplu: o reţea de transport **`G`**, cu capacitate şi flux pe arce.



Aceeaşi reţea de transport, cu reprezentarea reţelei reziduale **`Gr`**:



Evidenţierea unei căi de ameliorare **`s → c → b → t`**. De-a lungul acestei căi, fluxul poate fi mărit cu min(**`5`**, **`4`**, **`5`**) = **`4`**.



Metodele de determinare a fluxului maxim se bazează pe câteva idei, comune şi altor algoritmi de flux:

- **Reţea rezidulă**
- **Drum de ameliorare** (creştere) – "***augmenting path***"
- **Tăietură** - "***cut***"

1. Metoda **Ford-Fulkerson**

Ford-Fulkerson este o metodă iterativă. Se porneşte de la fluxul **`0`**, pe toate arcele reţelei reziduale. La fiecare iteraţie, se caută un ***drum de ameliorare*** **`p`** de la **`s`** la **`t`** şi se măreşte fluxul total **`f`** cu valoarea **`f0`**, unde **`f0`** este numărul maxim unităţi de flux cu care poate fi mărit fluxul de-a lungul acestui drum.

Pseudocod:
Intrare: *Reţeaua* **`G`**, *sursa* **`s`**, *destinaţia* **`t`**
Initializare: *Fluxul total* **`f = 0`**
cat timp *există un drum de ameliorare* **`p`**
creşte fluxul **`f`** *de-a lungul drumului* **`p`** *cu valoarea* **`f0`**

Complexitate: **`O(m * fmax)`** unde **`fmax`** e fluxul maxim găsit de algoritm.

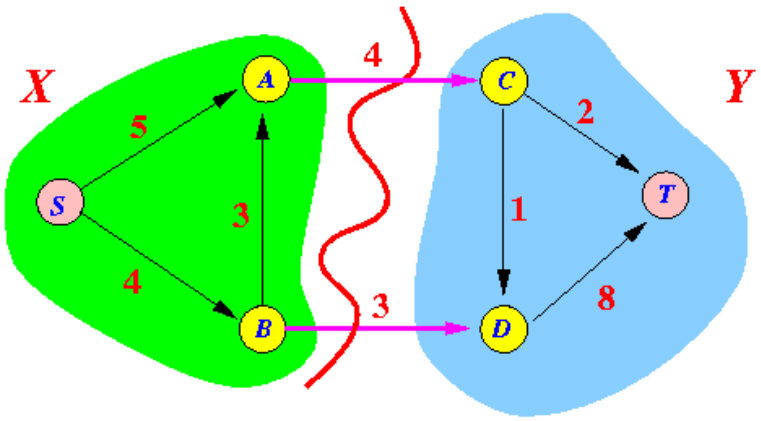
2. Algoritmul **Edmonds-Karp**

Acest algoritm utilizează metoda *Ford-Fulkerson*, doar că drumurile de ameliorare se caută cu parcurgeri în lăţime (*breadh-first search*), adică se caută mereu cele mai scurte drumuri de ameliorare. Timpul de execuţie se îmbunătăţeşte. Complexitate: **`O(n * m*m)`**

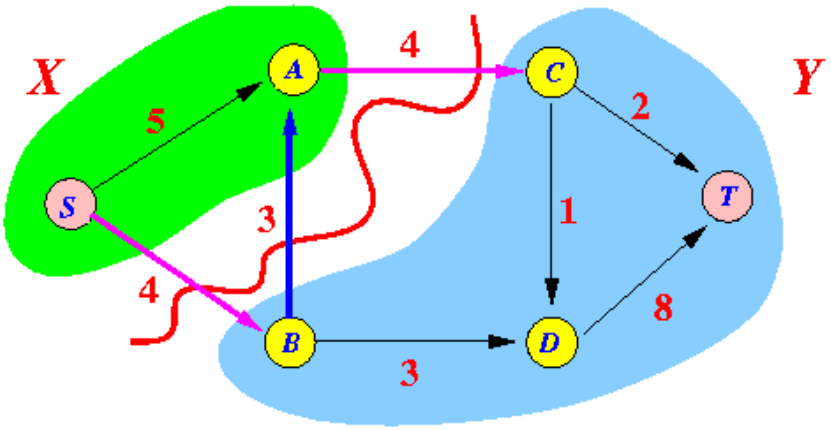
Flux-Maxim Tăietura-minimă (Max-Flow Min-Cut)

Teoremă:

Fluxul maxim care trece de la sursa **`s`** la destinaţia **`t`** este egal cu capacitatea totală a arcelor unei tăieturi minime (**Max-Flow = Min-Cut**). Altfel spus, fluxul maxim este egal cu capacitatea totală a arcelor, care tăiate, vor deconecta sursa de destinaţie (din punct de vedere al fluxului).



Capacitatea tăieturii: **4 + 3 = 7**



Capacitatea tăieturii: **4 + 4 = 8.**

Observaţi că se iau în considerare numai arcele `x->y` pentru care `x ∈ X` şi `y ∈ Y`, adică numai arcele care contribuie la creşterea fluxului net de la `s` la `t`.

Problemă
Network (*Bursele Agora*)

O companie deţine o reţea de calculatoare. Oricare două calculatoare pot comunica între ele în mod direct sau indirect. Datorită faptului că reţeaua este veche şi nu a fost întreţinută corespunzător, conducerea doreşte să ştie cât de mare este riscul ca două calculatoare din reţea să nu mai poată comunica. Compania v-a angajat să realizaţi un program care, pe baza legăturilor dintre calculatoare, să determine numărul minim de legături care ar trebui distruse astfel încât să existe două calculatoare care nu mai pot comunica.

Date de intrare
Fişierul de intrare **network.in** conţine pe prima linie două numere n şi m, separate între ele printr-un singur spaţiu, care reprezintă numărul de calculatoare din reţea, respectiv numărul de legături directe dintre ele.
Fiecare dintre următoarele **m** linii conţine câte două numere, separate printr-un singur spaţiu, care reprezintă numerele de ordine a două calculatoare între care există legătură directă. Calculatoarele din reţea sunt numerotate de la 1 la n şi între două calculatoare există cel mult o singură conexiune.

Date de ieşire
Fişierul de ieşire **network.out** trebuie să conţină o singură linie pe care se va afla un singur număr care reprezintă numărul minim de legături care ar trebui distruse astfel încât să existe două calculatoare care nu mai pot comunica în mod direct sau indirect.

Restricţii
- **1 ≤** n **≤ 100, 1 ≤** m **≤ 5000**

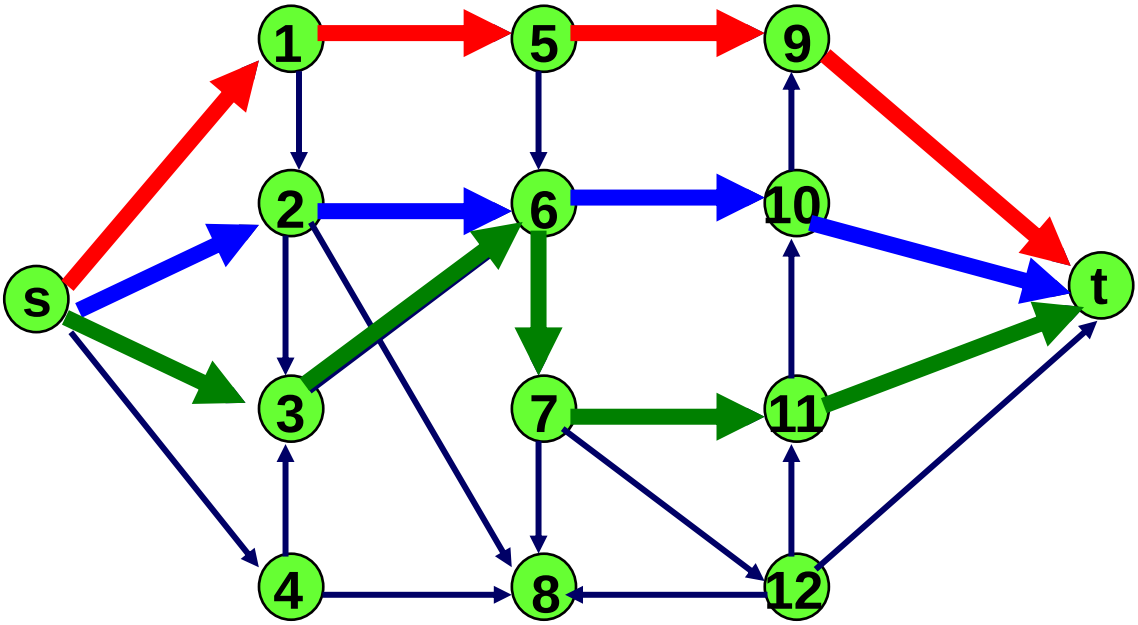
Exemplu

network.in	network.out
4 4 1 2 1 4 2 3 3 4	2

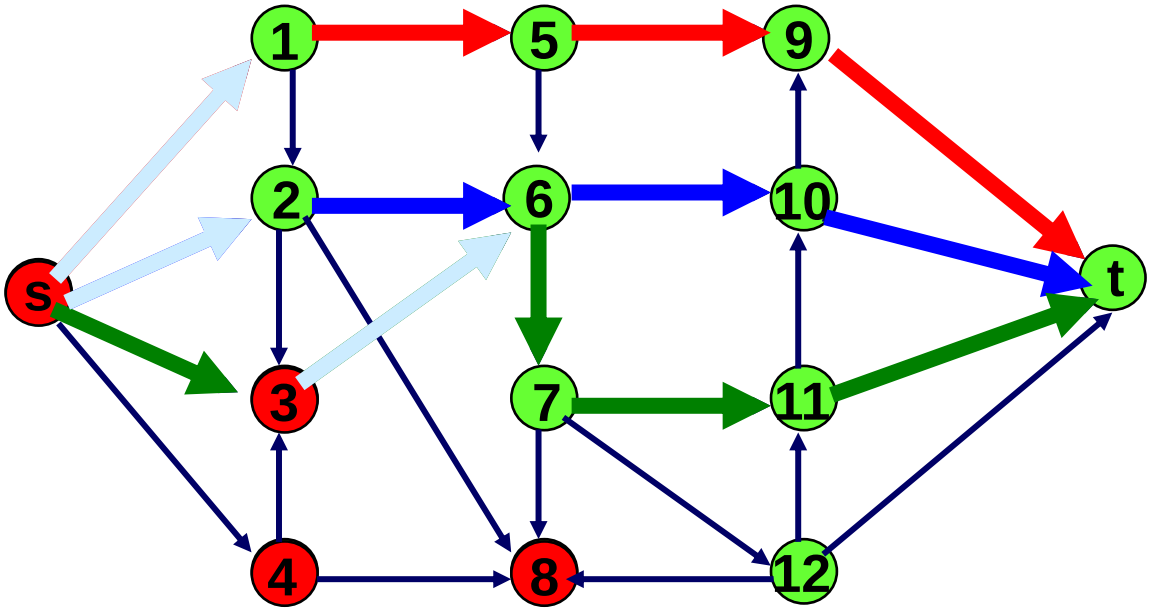
Soluţie:
Se construieşte o reţea de transport astfel: pentru fiecare muchie **x-y** se adaugă arcele **x→y** şi **y→x** de capacităţi **1**. Se fixează nodul sursă **s** ca fiind nodul **1** şi pentru fiecare nod destinaţie **t** : **1**, **2**, **...**, **n** se se determină fluxul maxim, aplicând algoritmul *Edmonds Karp*. Dimensiunea *tăieturii minime* (numărul minim de muchii care trebuie tăiate) se obţine ca fiind minimul fluxurilor în reţea pentru fiecare dintre aceste cazuri.

Teorema lui Menger

Teoremă:
Fie **G** un graf orientat. Atunci numărul maxim de **drumuri disjuncte din punct de vedere al arcelor** între două noduri **s** şi **t** din **G** este egal cu *tăietura minimă* (numărul minim de arce a căror tăiere ar deconecta **s** de **t**).
Consecinţă:
Coform teoremei *Max-Flow/Min-Cut*, numărul de **drumuri disjuncte din punct de vedere al arcelor** între două noduri **s** şi **t** este egal cu fluxul maxim în reţea, având sursa în **s** şi destinaţia în **t**.



În figură, numărul de drumuri disjuncte **s-t** este 3.



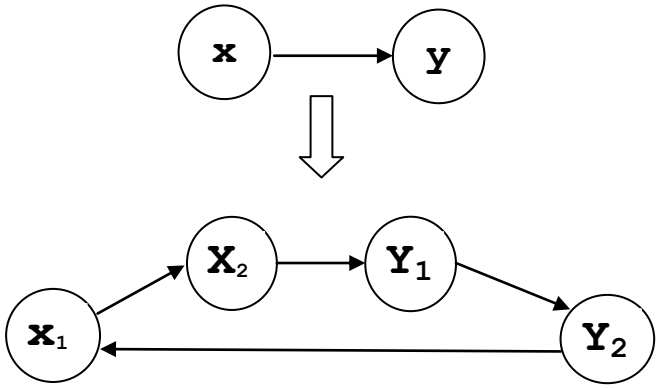
Fie mulţimea de noduri **X = {s, 3, 4, 8}**. Accesul de la această mulţime spre mulţimea **V – X** se face numai prin cele trei arce tăiate: **s→1, s→ 2, 3→ 6**. Observăm: numărul de drumuri disjuncte **s-t** coincide cu cardinalul tăieturii minime.

Node Splitting

Fie un graf orientat **G**. Atunci numărul maxim de **drumuri disjuncte din punct de vedere al vârfurilor** între două noduri **s** şi **t** din **G** se poate determina dacă plasăm capacităţi pe noduri (în cazul de faţă **capacitate(x) = 1**) modelând graful ca o reţea de transport astfel:

- fiecare nod **x** se înlocuieşte cu două noduri x_1 şi x_2 (x_1 va fi conectat cu toate arcele care intră în **x** iar x_2 e conectat cu toate arcele care ies din **x)**.
- Pentru fiecare arc **x->y** din **G**, se adaugă în reţeaua de transport arcele:
  x_1**->**x_2, x_2**->**y_1, y_1**->**y_2, şi y_2**->**x_1 având toate capacitatea **1**
  s_1**->**s_2 şi t_1**->**t_2 de capacitate infinită.

Apoi se rulează un algoritm de flux maxim între **s = s1** şi **t = t2**.



Prin dublarea nodurilor şi impunerea unui flux intern de capacitate 1 pentru fiecare nod, ne asigurăm că niciun nod nu va face parte din mai multe drumuri de ameliorare.
Este comod ca în noul graf x_1 să fie substituit cu **2*x**, iar x_2 cu **2*x + 1**.

Probleme

1. **Ghizi** (Baraj ONI 2003)

Se caută ghizi pentru Olimpiada Naţională de Informatică. Deoarece la Olimpiadă participă **K** echipe, trebuie ca în fiecare moment de timp din intervalul **[0, 100)** să existe exact **K** ghizi.

Pentru posturile de ghid s-au înscris **N** voluntari, care au fost numerotaţi distinct de la **1** la **N**. Fiecare dintre cei **N** voluntari a specificat un interval de timp în care poate asigura servicii de ghid. Voluntarul **i** poate fi ghid în intervalul **[**$T1_i$**,** $T2_i$**)** (intervalul este închis în $T1_i$ şi deschis în $T2_i$).

Cerinţă
Dându-se intervalele de timp asociate celor **N** voluntari, determinaţi o variantă de angajare astfel încât în fiecare moment de timp să fie prezenţi **exact K** ghizi. Numărul total de voluntari angajaţi este irelevant.

Date de intrare
Prima linie a fişierului de intrare **ghizi.in** conţine două numere întregi **N** şi **K**, separate printr-un spaţiu, cu semnificaţiile de mai sus. Voluntarii sunt numerotaţi distinct, de la **1** la **N**. Fiecare dintre următoarele **N** linii conţine descrierea unui voluntar; mai exact linia **i+1** conţine valorile întregi $T1_i$ şi $T2_i$ pentru voluntarul **i**.

Date de ieşire
În fişierul **ghizi.out** veţi afişa pe prima linie numărul total de ghizi angajaţi (**M**). Pe a doua linie veţi scrie **M** numere distincte între **1** şi **N**, ordonate crescător, reprezentând numerele de ordine ale ghizilor angajati.

Restricţii şi precizări
- **1 ≤ N ≤ 5000**
- **0 ≤ T1 < T2 ≤ 100** pentru fiecare dintre cei **N** voluntari
- **1 ≤ K ≤ N**
- Pot exista 2 voluntari cu acelaşi interval asociat.
- Toate testele date vor avea soluţie.
- Dacă există mai multe soluţii, afişaţi una oarecare.
- La preselecţie participă numai fete.

Exemplu

ghizi.in	ghizi.out
6 2 0 100 0 15 15 99 0 10 10 20 20 100	4 1 4 5 6

Soluţie:
Problema devine o problemă de flux, prin următoarea transformare:
- momentele întregi de timp se transformă în nodurile unui graf;
- un interval **[T_1,T_2)** asociat unui ghid introduce un arc de la **T_1** la **T_2**, cu capacitatea **1**. Două noduri pot fi conectate prin mai multe arce.

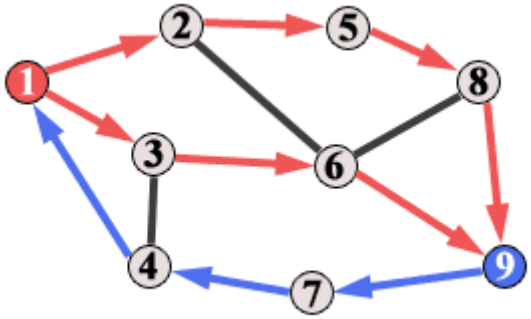
Un drum de la sursă (nodul **0**) la destinaţie (nodul **100**) acoperă fiecare moment de timp din **[0,100)** exact o dată. Astfel, problema poate fi reformulată:

"Dându-se graful construit conform transformării de mai sus, să se determine K drumuri disjuncte din punct de vedere al muchiilor, de la sursă la destinatie".

Aplicând teorema lui Menger, se introduce un arc de capacitate K de la o sursă fictivă la nodul 0 şi se execută un algoritm de flux maximal, sau se aplică de K ori *Edmonds Karp*.

2. **Joc** (Algoritmus)

Două persoane participă la un joc. Pe tabla de joc este reprezentat un graf conex, neorientat, cu **N** varfuri. Fiecare jucator are asociat un anumit vârf. La fiecare rundă unul dintre jucatori trebuie să găsească un drum de la nodul său la nodul adversarului. În momentul în care un jucător găseşte un astfel de drum este rândul adversarului să mute. Nodurile prin care s-a trecut nu mai pot fi folosite în construcţia unui alt drum (cu exceptia nodurilor asociate jucătorilor). O rundă este completă dacă jucătorul reuşeşte să determine un drum de la nodul său la nodul adversarului. Orice rundă incompletă determină incheierea jocului. Dat fiind graful pe care se desfăşoară jocul, să se determine numărul maxim de runde complete care se pot încheia.



Date de intrare
Pe prima linie a fișierului **JOC.IN** se află patru valori întregi **N**, **M**, **A** și **B**. **N** (**1≤N≤250**) reprezintă numărul de noduri ale grafului, **M** (**1<=M<=5000**) numărul de muchii iar **A** și **B** vârfurile asociate celor doi jucatori. Pe următoarele **M** linii se află câte două valori **x** și **y**, indicând că în graf există muchie între vârfurile **x** și **y**.

Date de ieșire
În fișierul **JOC.OUT** se va afișa numărul de runde complete ce se pot desfășura.

Exemplu

JOC.IN	JOC.OUT
9 13 1 9 1 2 1 3 1 4 2 5 2 6 3 4 3 6 4 7 5 8 6 8 6 9 7 9 8 9	3

Soluție

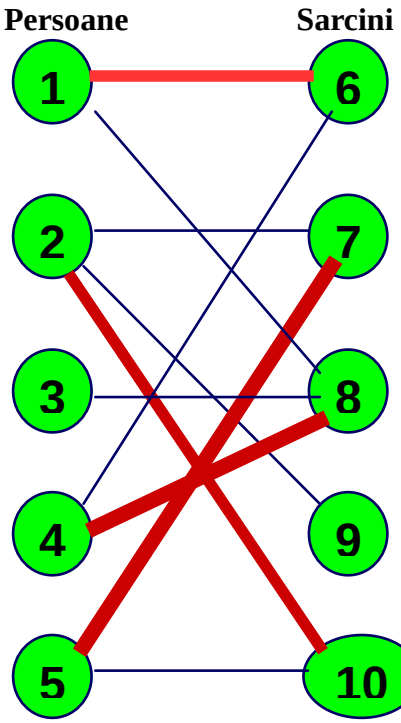
Suntem în cazul determinării drumurilor disjuncte din punct de vedere al nodurilor. Ținem seama că graful este neorientat. Prin urmare, pentru fiecare muchie **x->y** din **G**, se adaugă în rețeaua de transport arcele:

$x_1 \to x_2$, $x_2 \to x_1$, $x_2 \to y_1$, $y_1 \to y_2$, $y_2 \to y_1$ și $y_2 \to x_1$ având toate capacitatea **1**

Pe această rețea se aplică un algoritm de flux maxim. Valoarea acestuia este egală cu numărul de drumuri disjuncte din punct de vedere al nodurilor.

B. Cuplaj maxim în graf bipartit
(*maximum matching*)

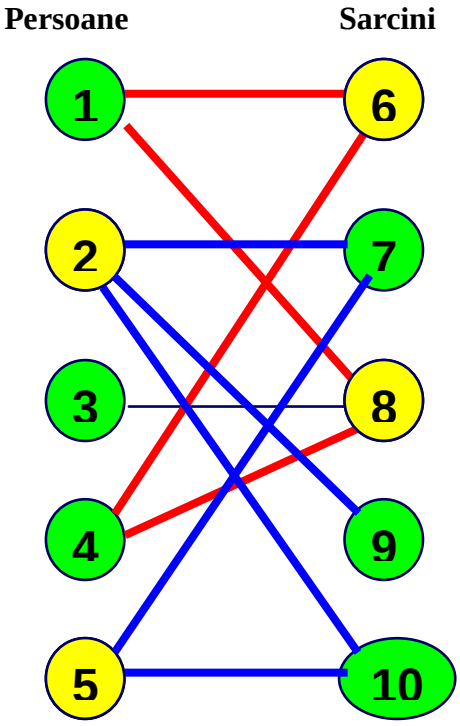
Graf bipartit
Un graf `G = (V, E)`, unde `V` e mulţimea nodurilor iar `E` este mulţimea arcelor, este bipartit dacă `V` poate fi partiţionat două submulţimi, asfel încât pentru oricare arc `x→y` dacă `x` aparţine unei submulţimi, atunci `y` aparţine celeilalte submulţimi.



Cuplaj în graful bipartit, este o submulţime de arce, astfel încât nu există două arce din submulţime incidente aceluiaşi nod.

Problema cuplajului – găsirea unui cuplaj de cardinalitate maximă.

Acoperirea cu noduri (*Node Cover*)



Fie un graf `G = (V, E)`. Acoperirea cu noduri este o submulţime `C` din `V`, astfel încât fiecare muchie din graf este incidentă cel puţin unui nod din mulţimea `C`.

Acoperirea minimă (*Minimum Vertex Cover*)

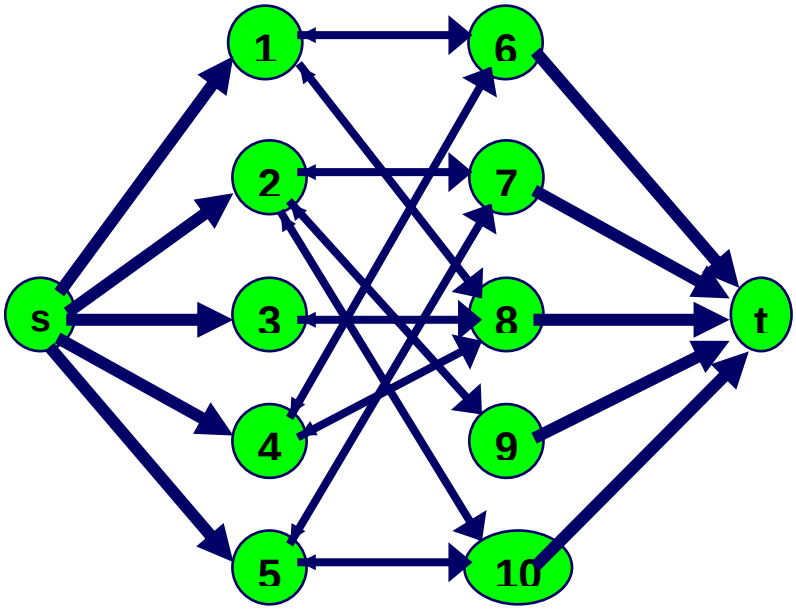
Acoperirea minimă `C` este o acoperire cu un număr minim de noduri în mulţimea `C`. Această problemă este NP completă pentru un graf general.
În exemplu, cardinal(`C`) `= 4. C = {2, 8, 5, 6}`

Mulţimea stabilă (suportul) (*Independent Set)*
este o submulţime I de vârfuri din graf, astfel încât pentru oricare două vârfuri din `I`, nu există în graf un arc să le conecteze.

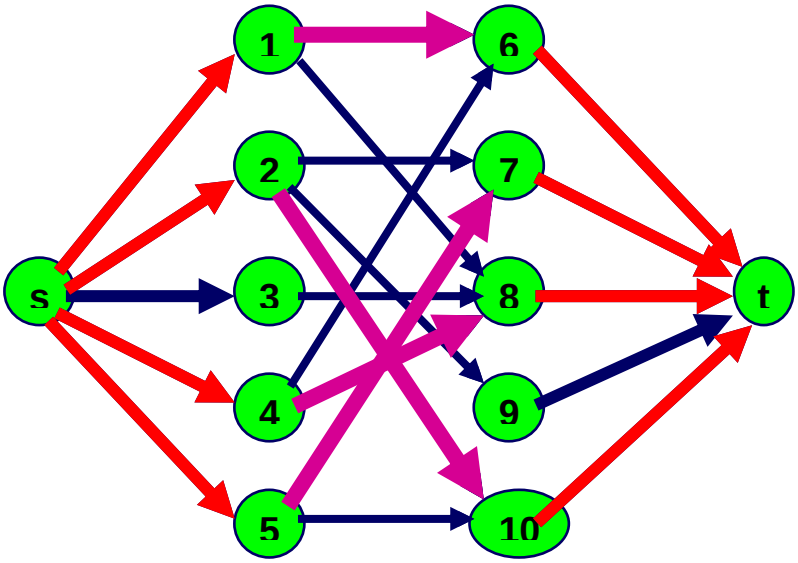
Maximum Independent Set
este un *Independent Set* de dimensiune maximă.

Determinarea cuplajului maxim într-un graf bipartit

1. Reducerea problemei la o problemă de flux.



Reţeaua de transport se construieşte astfel:
- arcele originale se înlocuiesc cu arce de capacitate infinită
- se adaugă două noduri **`s`** şi **`t`**.
- fiecare arc **`s`**→**`i`** primeşte capacitatea **`1`**.
- fiecare arc **`j`**→**`t`** primeşte capacitatea **`1`**.



În exemplul anterior, fluxul maxim **s-t** este 4. Cuplajul maxim are cardinalitatea 4.

2. **Teorema ungară (Kőnig-Egervary)**

În cazul grafurilor bipartite există o legătură între *Maximum Independent Set* (I) şi *Minimum Vertex Cover* (C). Pentru un graf **G (V, E)**, cu **n** noduri, **n = |V|**. Atunci **n = |C| + |I|**.
Altfel spus, mulţimea **I** este complementară mulţimii **C** în raport cu mulţimea **V**.

Teoremă
În oricare graf bipartit, numărul de muchii **M** din ***Cuplajul Maxim*** este egal cu numărul de vârfuri **|C|** din ***Acoperirea Minimă***. Deci **M = |C|.**

Lanţuri alternante
Pentru determinarea cuplajului maxim, există algoritmi care utilizează conceptul de ***lanţ alternant***.
Să considerăm un cuplaj non-maximal **M** într-un graf bipartit **G =** (**X**, **Y**, **E**), unde **X**, **Y** sunt mulţimile de noduri în cele două partiţii. Dacă un vârf nu este cuplat, îl numim ***liber***. Un lanţ alternant este un drum aciclic în **G**, care porneşte din partiţia **X**, de la un nod liber, şi se termină în partiţia **X** tot cu un nod liber. Astfel fiecare muchie cu număr de ordine par aparţine cuplajului. Numărul de muchii cuplate este cu unu mai mic decât numărul de muchii necuplate din lanţul alternant. Dacă interschimbăm muchiile cuplate cu cele necuplate, obţinem o mărire a cuplajului cu o unitate.

Algoritmul Hopkroft-Karp
Algoritmul determină valoarea cuplajului maxim într-un graf bipartit şi are un timp de rulare foarte bun: **O(sqrt(n)(n + m))**
Algoritmul se bazează pe căutarea repetată a lanţurilor alternante atâta timp cât este posibil.

Probleme:
1. **Pluto** (Bursele Agora)
Pe o tablă sunt desenate **N** cercuri. *Pluto* trebuie să deseneze un număr total de **M** săgeţi, astfel încât fiecare săgeată să pornească de la un cerc şi să ajungă la alt cerc. Pentru fiecare cerc se cunoaşte numărul săgeţilor care trebuie să plece din cercul respectiv şi numărul săgeţilor care trebuie să ajungă la cercul respectiv. Va trebui să verificaţi dacă *Pluto* poate desena săgeţile şi, dacă acest lucru este posibil, să descrieţi modul în care acestea vor fi desenate.

Date de intrare
Prima linie a fişierului de intrare **pluto.in** conţine numărul **N** al cercurilor şi numărul **M** al săgeþilor. Cea de-a doua linie conţine **N** numere întregi, separate prin câte un spaţiu, care reprezintă numărul săgeţilor care trebuie să pornească de la fiecare cerc. Cea de-a treia linie conţine **N** numere întregi, separate prin câte un spaţiu, care reprezintă numărul săgeţilor care trebuie să ajungă la fiecare cerc. Primul număr de pe fiecare dintre aceste linii corespunde primului cerc (identificat prin 1), al doilea număr corespunde celui de-al doilea cerc (identificat prin 2) etc. Suma numerelor de pe fiecare dintre aceste două linii este întotdeauna **M**.

Date de ieşire
Prima linie a fişierului de ieşire **pluto.out** va conţine mesajul **YES** în cazul în care există cel puţin o posibilitate de desenare a săgeţilor şi mesajul **NO** în caz contrar. În cazul în care săgeţile pot fi desenate, fişierul va mai conţine **M** linii pe care se vor afla câte două numere întregi **x** şi **y** cu semnificaţia: *va fi desenată o săgeată care porneşte de la cercul identificat prin* **x** *şi ajunge la cercul identificat prin* **y**.

Restricţii şi precizări

- **1 ≤ N ≤ 50**
- **1 ≤ M ≤ 500**
- cercurile sunt identificate prin numere naturale cuprinse între **1** şi **N**;
- poate fi desenată cel mult o săgeată care poneşte de la un cerc **x** şi ajunge la un cerc **y**;
- dacă s-a desenat o săgeată care poneşte de la un cerc **x** şi ajunge la un cerc **y**, atunci poate sau nu fi desenată şi o săgeată care poneşte de la cercul **y** şi ajunge la cercul **x**;
- dacă există mai multe soluţii atunci poate fi aleasă oricare dintre ele.

Exemplu

pluto.in	**pluto.out**
4 5 **1 2 2 0** **1 1 1 2**	**YES** **1 2** **2 3** **3 1** **2 4** **3 4**

Soluţie

Problema poate fi modelată ca o problemă de flux maxim în graf bipartit. Se modelează reţeaua de transport astfel:
- se adaugă nodul sursă **s = 0**, respectiv nodul destinaţie **t = 2*n + 1**
- se adaugă nodurile **2n + 1**, ..., **2n**, ca "oglinzi" (ale nodurilor **1**, **2**, ..., **n**)
- se adaugă arcele **s→i** (**i = 1**, **2**, **..n**) de capacităţi **out[i]** – gradul exterior al nodului **i**
- se adauga arcele **j→t** (**j = n + 1**, **n + 2**, ..., **2n**) de capacităţi **in[j]** – gradul interior al nodului **j**
- se conectează nodurile **i** din prima partiţie: **1**, **2**, ..., **n** cu nodurile **j**: **n + 1**, **n + 2**, **...**, **2n** din a doua partiţie cu arce de capacitate: **c[i][j] = 1** dacă **i != j - n** respectiv **c[i][j] = 0** dacă **i = j – n**.

Se rulează un algoritm de flux maxim pe această reţea. Dacă fluxul maxim este egal cu suma gradelor exterioare, deci dacă toate arcele incidente sursei vor fi saturate, atunci graful este admisibil, prin urmare este posibilă desenarea săgeţilor.

2. **Arranging Flowers -** (IPSC 2003) - enunţ redus.

Se doreşte aranjarea a mai multor tipuri de flori într-o matrice dreptunghiulară, astfel încât:
- fiecare rând al matricei să conţină fiecare tip de floare exact odată
- fiecare coloană a matricei să conţină fiecare tip de floare cel mult odată.

Date de intrare

Setul de date de intrare descrie câteva aranjamente florale. Pe prima linie se dă numărul seturilor de date. Fiecare descriere a unui aranjament floral începe cu două numere naturale **N** şi **M**, reprezentând numărul de coloane şi de rânduri ale matricei. Diferitele tipuri de flori sunt reprezentate de numerele **1**, **2**, ..., **N**. Următoarele **M** linii conţin câte **N** numere naturale reprezentând tipurile de flori pe un rând al aranjamentului.

Date de ieşire
Pentru fiecare aranjament floral afişaţi **K** – numărul maxim de linii care mai pot fi adăugate aranjamentului. Apoi afişaţi **K** linii, cu câte **N** numere fiecare, reprezentând liniile adăugate aranjamentului.

Restricţii
- **1 ≤ N, M ≤ 300**

Exemplu

2 3 2 3 2 1 1 3 2 4 2 1 4 3 2 2 1 4 3	1 2 1 3 2 3 2 1 4 4 3 2 1

Soluţie
a. Problema poate fi modelată ca o problemă de flux maxim în graf bipartit. O prima parţiţie e formată din nodurile **i**: **1**, **2**, ..., **N** care reprezintă tipurile de flori. A doua partiţie e formată din nodurile **j**: **n + 1**, **n + 2**, ..., **2n** reprezentând coloanele matricei.

Se modelează reţeaua de transport astfel:
- se adaugă nodul sursă **s = 0**, respectiv nodul destinaţie **t = 2*n + 1**
- se adaugă nodurile **2n + 1**, ..., **2n**, reprezentând coloanele matricei
- se adaugă arcele **s→ i** (**i = 1**, **2**, **. . .** ,**n**), fiecare de capacitate **1**
- se adauga arcele **j→ t** (**j = n + 1**, **n + 2**, ..., **2n**), fiecare de capacitate **1**
- se conectează nodurile din prima partiţie: **1**, **2**, ..., **n** cu nodurile **n + 1**, **n + 2**, **. . .**, **2n** din a doua partiţie cu arce de capacitate: **c[i][j] = 1**
- se elimină arcele **i→ j** care unesc cele două partiţii şi corespund florilor deja plasate în matrice (**c[i][j] = 0**).

Se rulează pentru fiecare caz de test un algoritm de flux maxim pe această reţea între **s** şi **t**. Pentru fiecare drum de ameliorare găsit, acesta corespunde unei noi linii în matrice.

b. Se crează un graf bipartit astfel: nodurile **1**, **2**, **. . .**, **n** din prima partiție reprezintă tipurile de flori, iar nodurile **1**, **2**, ..., **n** din a doua partiție reprezintă coloanele matricei. Toate nodurile din prima partiție sunt conectate prin arce cu toate nodurile din a doua partiție. Arcele corespunzătoare rândurilor deja completate din matrice se elimină. Pe acest graf se rulează *Hopcroft-Karp* atâta timp cât cuplajul este maxim, adică **n**. Un rând nou în matrice reprezintă un nou cuplaj maxim obţinut în graful bipartit.

3. **Knights -** (Baltic Olympiad 2001)
Se dă o tablă de şah de dimensiune **n x n** din care unele câmpuri au fost înlăturate. Se cere să se determine numărul maxim de cai care pot fi plasați în câmpurile rămase libere ale tablei, astfel încât aceştia să nu se atace.

Cerință
Scrieți un program care:
- Citește descrierea tablei de șah cu unele cîmpuri interzise din fișierul **kni.in**

- Determină numărul maxim de cai care pot fi plasați pe tabla de șah astfel încât să nu se atace
- Scrie rezultatul în fisierul **kni.out**

Date de intrare

Prima linie a fișierului **kni.in** conține două numere naturale **n** și **m** (*1≤ n ≤200, 0≤ m <n²*)
N este dimensiunea tablei de șah, iar m este numărul de câmpuri înlăturate. Fiecare dinte următoarele **m** linii conține două numere natural **x** și **y** *1 ≤ x, y ≤ n* reprezentând coordonatele câmpurilor înlăturate. Coordonatele colțului stânga sus sunt (**1**, **1**) , iar dreapta jos (**n**, **n**). Câmpurile înlaturate nu se repetă în fișier.

Date de ieșire

Fișierul de ieșire **kni.out** trebuie să conțină un singur număr natural **x**, reprezentând numărul maxim de cai care pot fi plasați pe tabla de șah astfel încât aceștia să nu se atace.

Exemplu

kni.in	**kni.out**
3 2 **1 1** **3 3**	**5**

Soluție

Construim un graf ale cărui vârfuri sunt câmpurile libere de pe tabla de șah, iar muchiile sunt mutările posibile ale cailor. La o mutare, calul sare întotdeauna de pe un câmp alb pe un câmp negru sau invers. Prin urmare, graful este bipartit. Câmpurile albe formează o partiție, iar cele negre cealaltă partiție.

Pentru a evita etichetarea nodurilor garfului, se poate identifica fiecare nod al grafului prin perechea (**i**, **j**) – (linie, coloană), iar culoarea câmpului se determină ad-hoc observând că expresia **(i+j)%2** are valoarea **0** pentru câmpuri albe, respectiv **1** pentru câmuri negre.

În acest graf bipartit, căutăm numărul maxim de vârfuri care nu se văd între ele, adică nu există două care să fie conectate printr-o muchie. Cu alte cuvinte, căutăm ***Mulțimea Stabilă Maximală*** / ***Maximum Independent Set*** (**I**). Utilizăm teorema ungară, conform căreia în orice graf bipartit dimensiunea ***Acoperirii Minime cu vârfuri*** / ***Minimum Vertex Cover*** (**C**) este egală cu dimensiunea cuplajului maxim **M**.

În același timp, mulțimea **I** este complementară mulțimii **C** în raport cu mulțimea tuturor vârfurilor (**V**), adică **|V| = |I| + |C|**, sau **|V| = |I| + |M|**. Deci **|I| = n*n – |M|**.

Gradul maxim al fiecărui nod este 8. Deci numărul maxim de muchii **m** a grafului este **8*|V| = O(V)**. Pentru determinarea cuplajului maxim în graful bipartit, se aplică algoritmul *Hopcroft-Karp*. Complexitatea algoritmului este $O((|E|+|V|)\sqrt{|V|}) = O(|V|^{2/3})$.