**Cuplaj maxim** = maximum (cardinality) matching <https://en.wikipedia.org/wiki/Matching_(graph_theory)>

-> **in bipartite** **graph** = undirected graph impartit in 2 multimi intre elementele carora trebuie sa trag muchii adica sa le unesc fara a crea loop-uri adica oricare 2 muchii nu trebuie sa aiba noduri comune

* un cuplaj e maximal daca sunt folosite toate muchiile grafului
* un cuplaj e maxim daca sunt folosite cat mai multe din muchiile grafului. Exista mai multe posibilitati de a face cuplaje maxime pentru un graf.Orice cuplaj maxim e si maximal, dar nu orice cuplaj maximal e maxim

Obs: gasirea cuplajului maxim se reduce la gasirea fluxului maxim (max flow)

=> Alg Ford-Fulkerson in O(max\_flow \* E), varianta Edmonds-Karp => O(EV3)

<https://www.geeksforgeeks.org/max-flow-problem-introduction/>

<https://www.geeksforgeeks.org/ford-fulkerson-algorithm-for-maximum-flow-problem/>

<https://www.programiz.com/dsa/ford-fulkerson-algorithm>

Alti algoritmi de aflare:

1. Pt graf bipartit neponderat => Hopcroft-Karp Algorithm in  O(rad(V)E) time <https://www.geeksforgeeks.org/hopcroft-karp-algorithm-for-maximum-matching-set-2-implementation/>
2. Pt graf bipartit ponderat (-> maximum-weight maching / the assignment problem) => Bellman-ford + Hungarian alg / Dijkstra
3. Pt graf nebipartit ponderat => Edmonds' blossom algorithm
4. Pt cuplaj maximal => Greedy

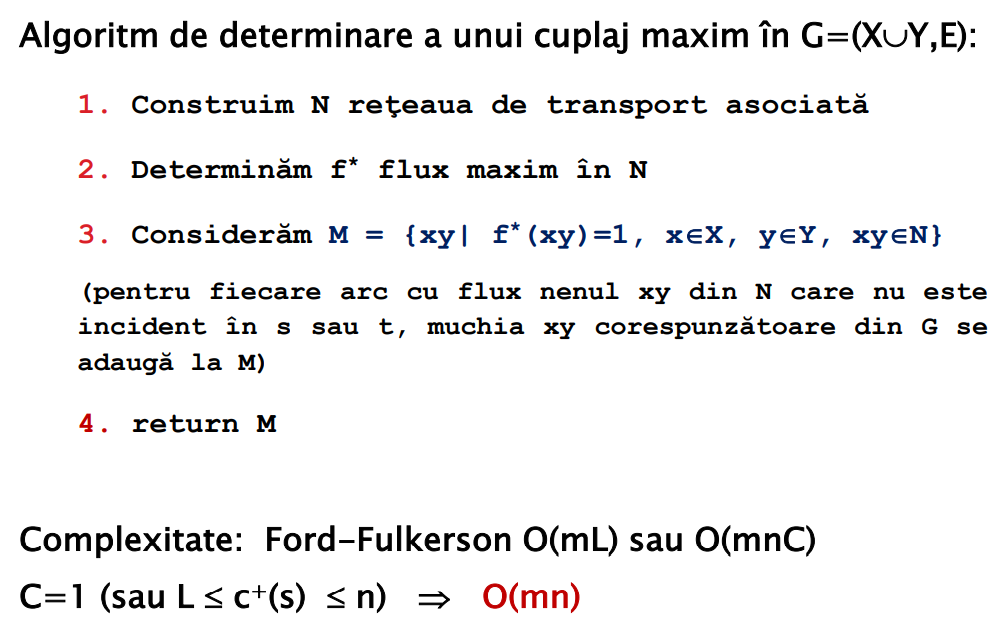
Inainte de cuplaj maxim -> flux maxim => **Ford-Fulkerson Algorithm in O(max\_flow \* E), varianta Edmonds-Karp => O(EV3)**

* <https://cp-algorithms.com/graph/edmonds_karp.html?fbclid=IwAR2yev5zuz7lSFDE0H9uXDxSsEIc5nRIAQ7b8glHiAVCWUorK4mlfKhiyds>

Pt flux maxim trb sa tinem un **graf rezidual** adica fluxul aditional posibil pentru fiecare muchie (daca mai poate fi adaugat flux pe muchia respectiva)

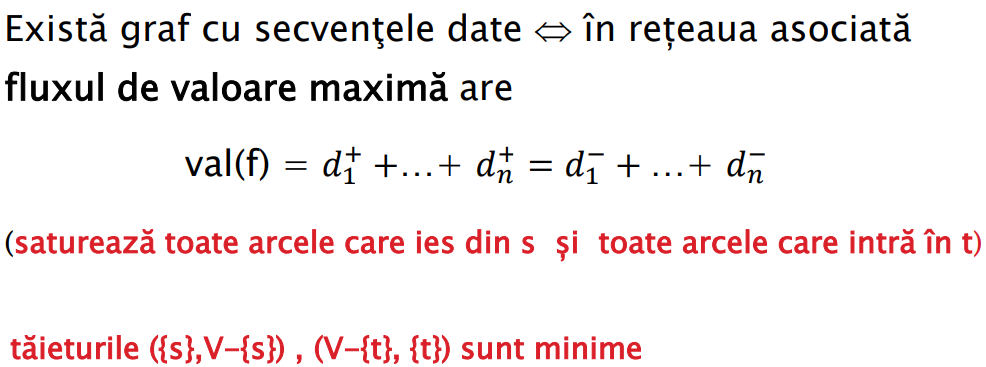
Daca nu exista muchia (x,y) atunci **capacitatea reziduala** pt ea (adica valoarea de pe muchia (x,y) in graful rezidual) e **0**.

Pentru a verifica daca exista drum de la sursa la destinatie se fol bfs sau dfs. Noi fol BFS (as BFS always picks a path with minimum number of edges).

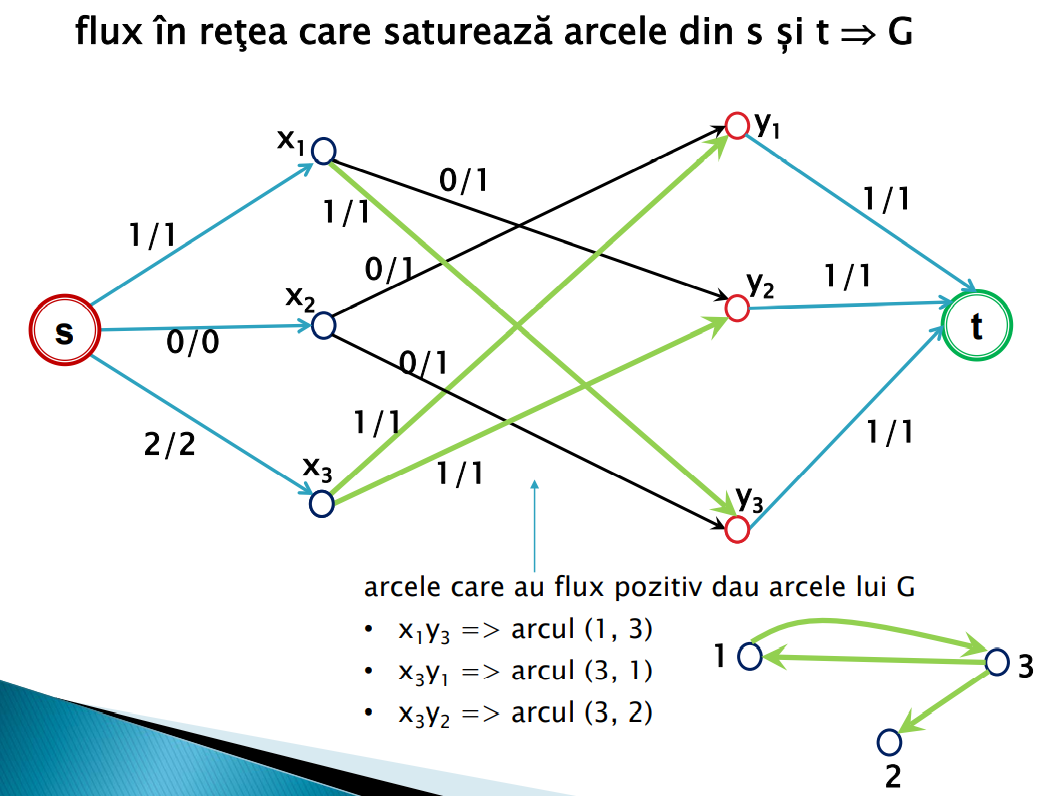


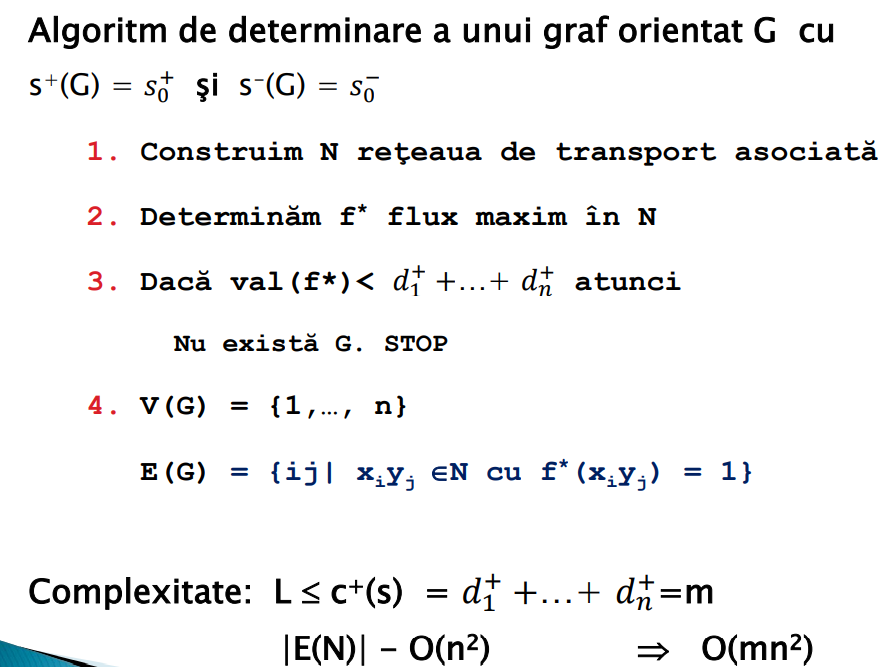
**Aplicatie: Constructie graf orientat din secvente de grade date**

Există graf cu secvenţele date ⇔ în rețeaua asociată fluxul de valoare maximă saturează toate arcele care ies din s și toate arcele care intră în t adica fluxul e



* tăieturile ({s},V-{s}) , (V-{t}, {t}) sunt minime

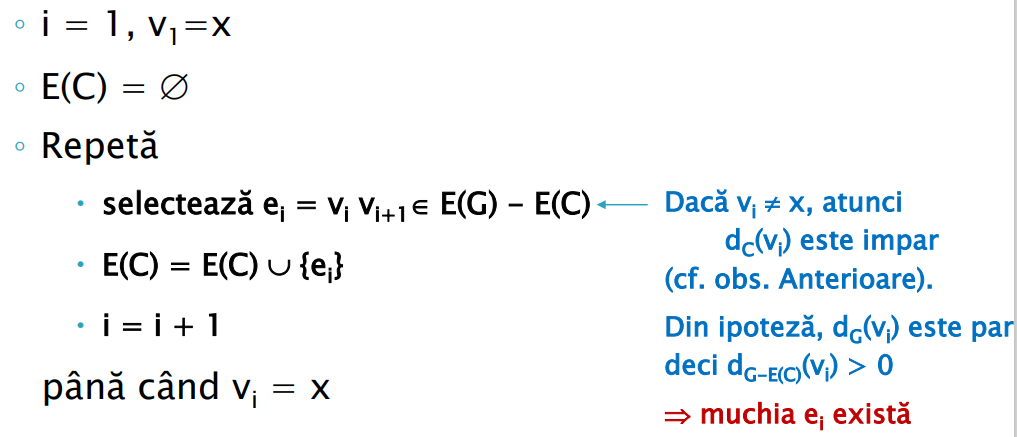




**Grafuri Euleriene**

Pt un graf neorientat, conex, cu toate vârfurile de grad par și E≠∅ => fiecare nod x face sigur parte din minim un ciclu

Algoritm de determinare a unui ciclu care conține x:

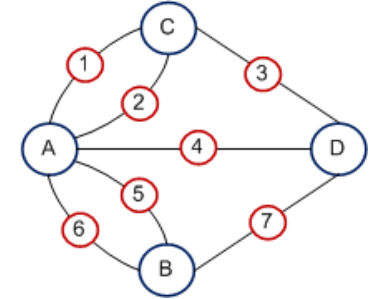


A connected multigraph has an **Euler Circuit** if and only if each of its vertices has even degree.  
(b) A connected multigraph has an Euler Path but not an Euler Circuit if and only if it has exactly two vertices of odd degree.  
(c) A complete graph (Kn) has a Hamilton Circuit whenever n ≥ 3.

Euler si-a pus pb daca poate parcurge o singura data fiecare pod din orasul sau

=> **Problema celor 7 poduri din Konigsberg**

=> se transf in graf pt care vreau sa parcurg toate muchiile si sa ma intorc in acelasi nod din care am plecat => grafurile cu aceasta propr s.n. euleriene (au un **ciclu euleria**n)

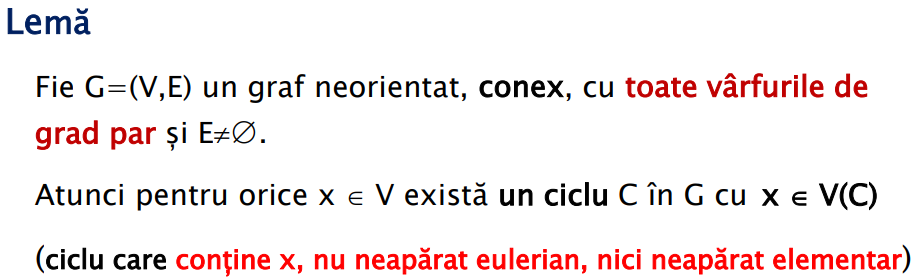
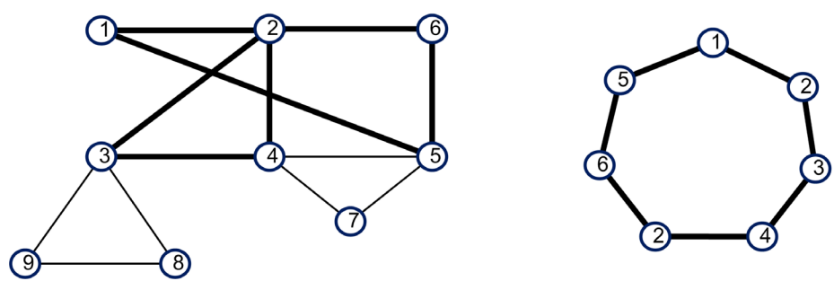


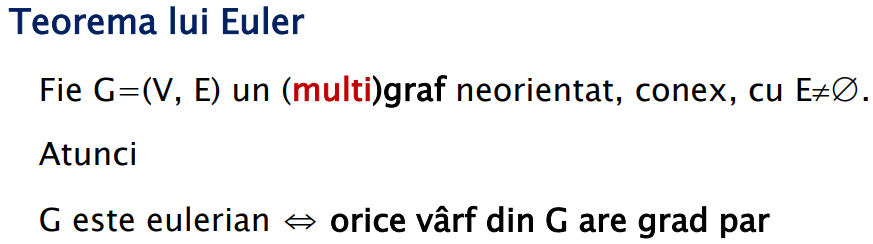
obs: graf eulerian = si daca nu ne intoarcem in acelasi nod si exista **lant eulerian** (nu ciclu) **simplu** (= nu se repeta muchiile)

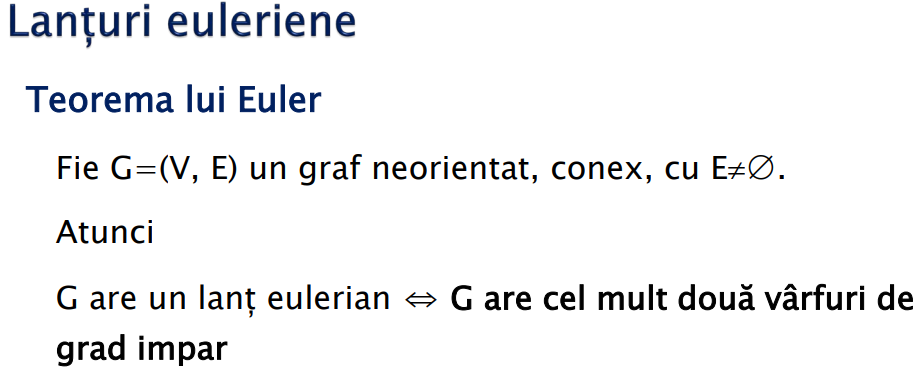
Interpretare:

Se poate desena diagrama printr-o curbă continuă închisă fără a ridica creionul de pe hârtie şi fără a desena o linie de două ori (în plus: să terminăm desenul în punctul în care l-am început) / a-l ridica o singura data? -> exemplu practic: taierea unui material fara a scoate foarfeca si fara a avea pierderi mari de material

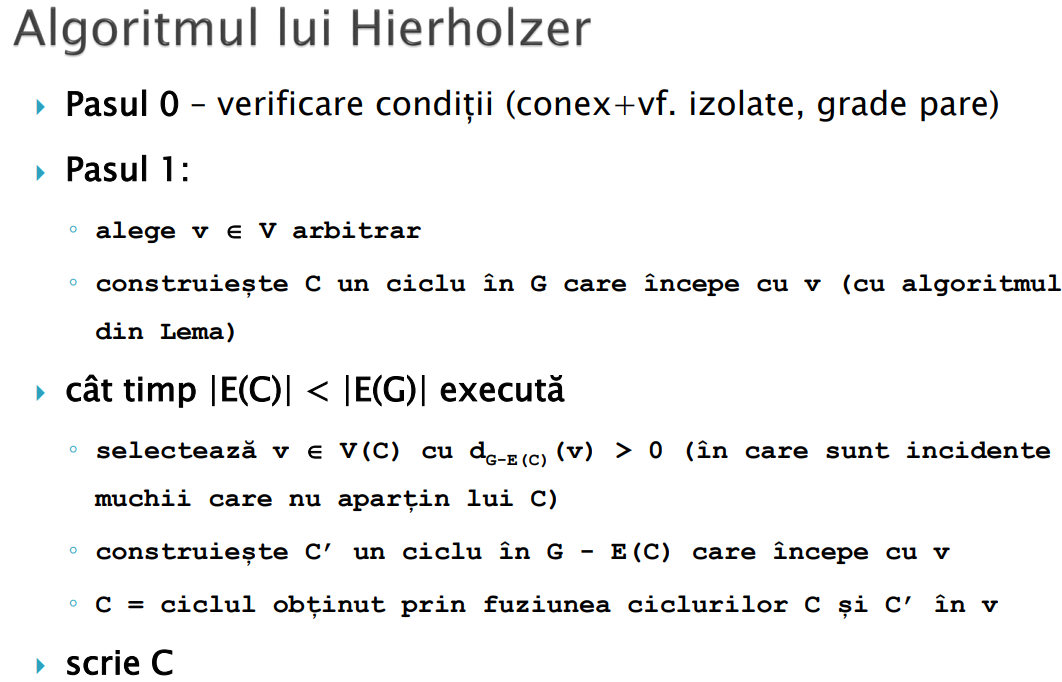
**Gradul varfurilor din ciclul eulerian** (/ lantul eulerian) **e par!**

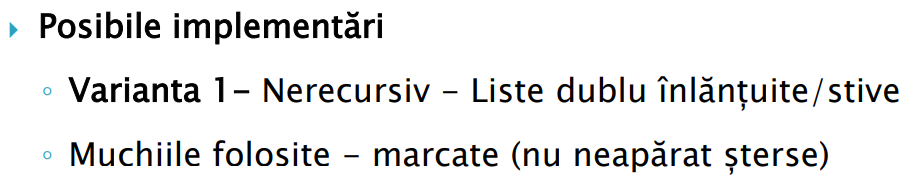
****

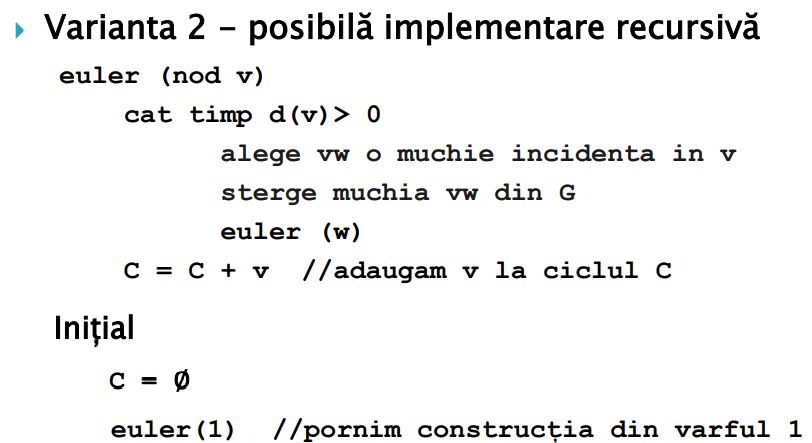
****

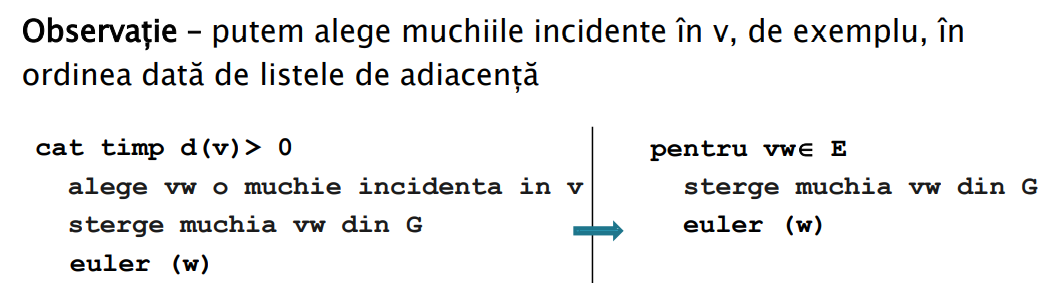


**Determinarea unui ciclu eulerian într-un graf conex (sau un graf conex+ vârfuri izolate) cu toate vârfurile de grad par -> O(m)**

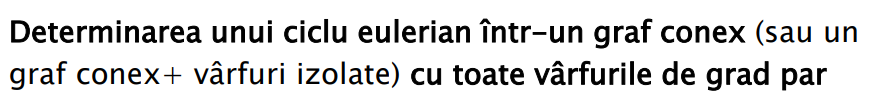


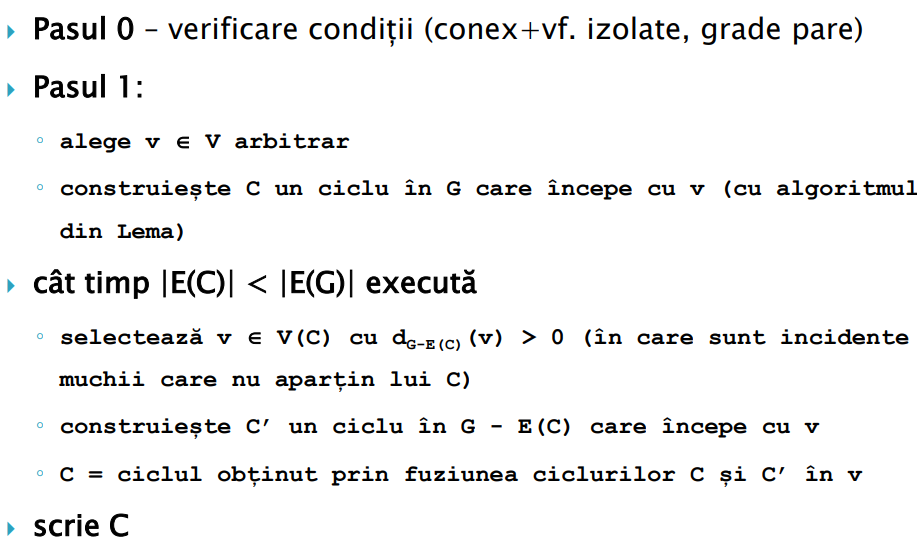






** => O(m)**

****



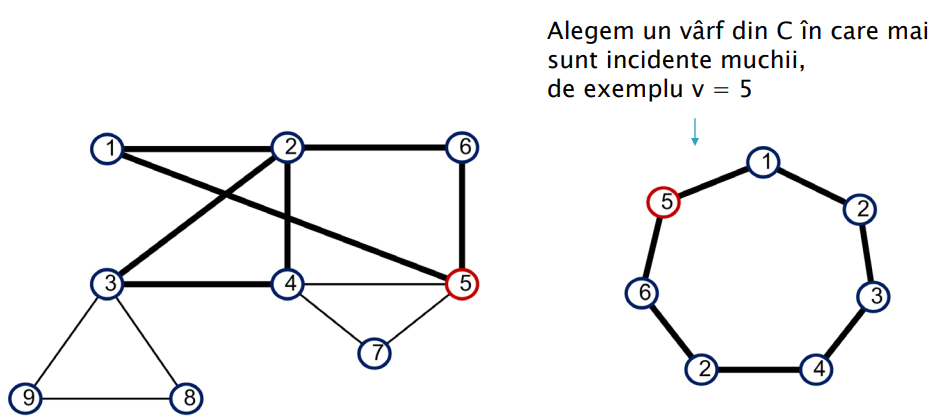
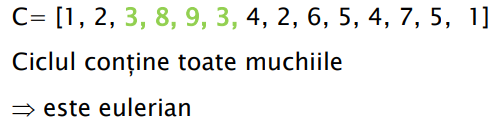
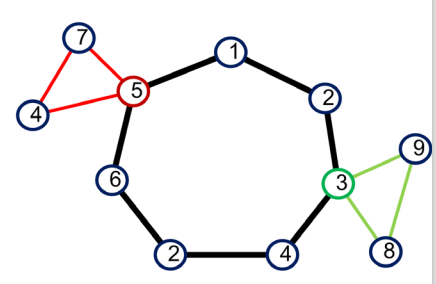
<=> cat timp mai exista muchii care nu fac parte din ciclul asta

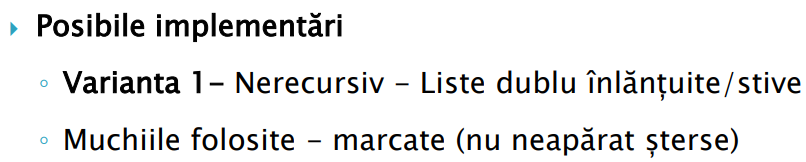
cu alg pe care tocmai l-am aratat

aleg un vf care face deja parte din ciclu si are gradul 0 dar dand la o parte muchiile care fac parte din ciclu deja

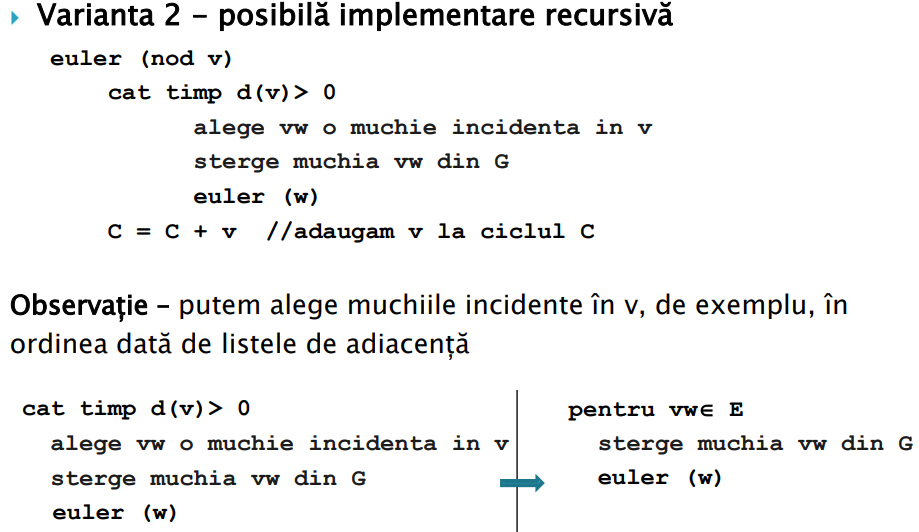
ma intreb daca graful mai are alte muchii incidente cu varful v din ciclu, dar care sa nu faca deja parte din ciclu

daca da, aplic iar pasul 1

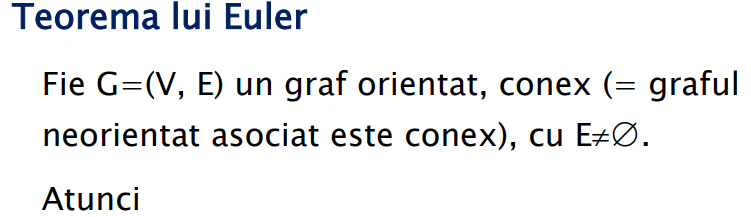
**=>**

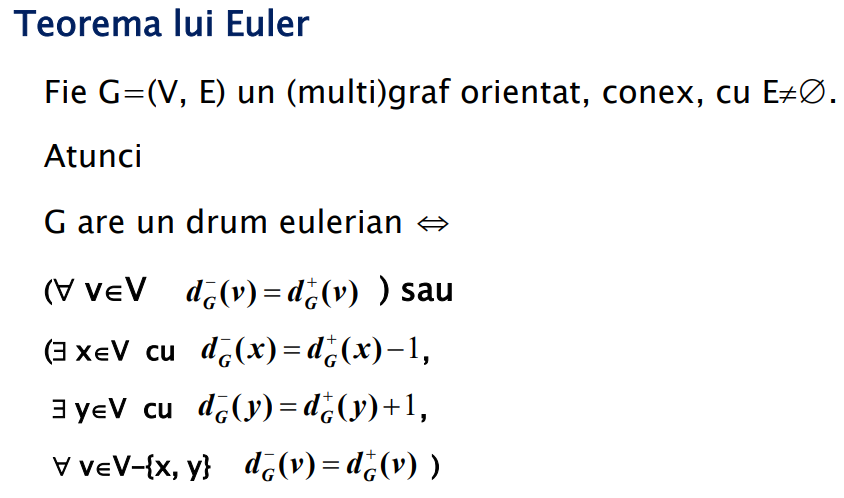
****

* cu stiva daca dim e f mare

****

****

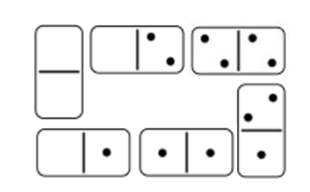
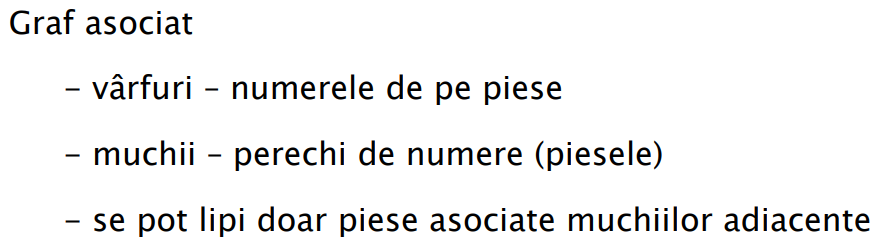
****

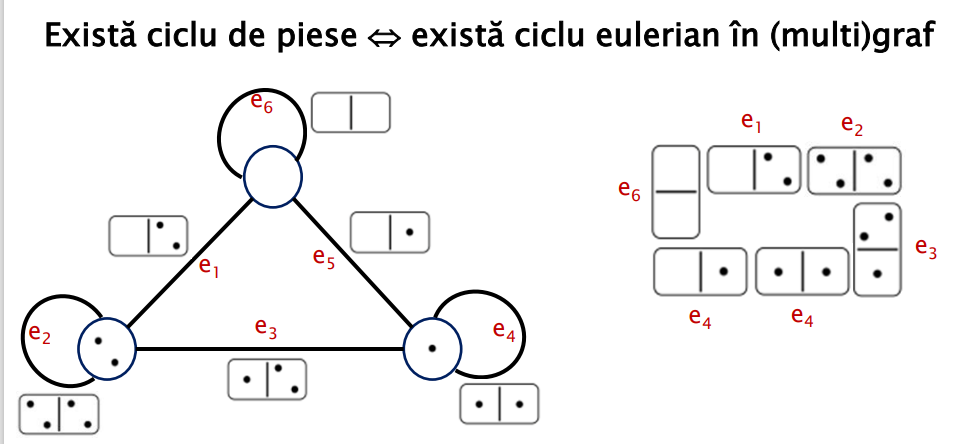
****

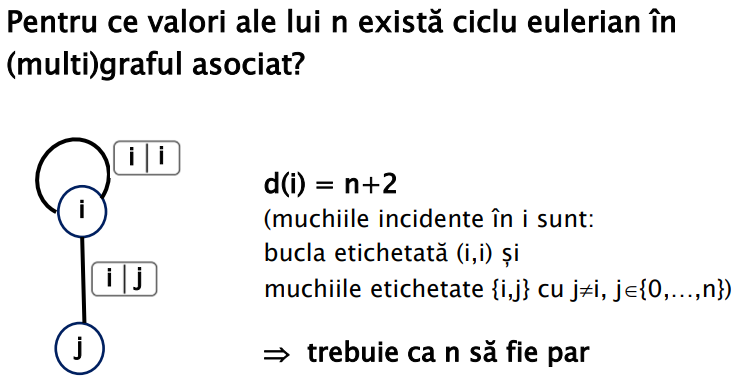
**Problemă – joc domino**

Am un sir de piese de domino. Respecta regula de construcție: primul număr de pe piesa adăugată la șir = al doilea număr de pe ultima piesă din șir

Se poate forma un șir de piese de domino care să conțină toate piesele + să se termine cu același număr cu care a început (un șir circular)?

 **=> **

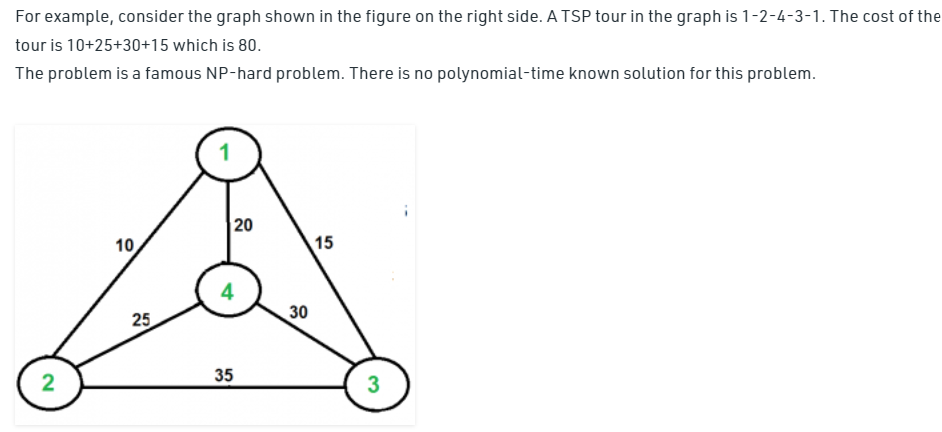
****

****

**Algoritmi Greedy de aproximare => cu optime**

* Problema comis-voiajorului (Travelling Salesman Problem-TSP)
* **ciclu hamiltonian** = trece o singura data prin fiecare nod

TSP ca problemă de decizie: Dat un graf G complet (neorientat) ponderat și un număr L, există un ciclu hamiltonian în G de cost cel mult L? Să se determine un ciclu hamiltonian de cost minim.



* naive sol: permutations
* progr dinamica: se considera nodul 1 sursa si destinatia. Pt fiecare nod i se calc costul drumului de la 1 la i, incercand sa luam pe rand toate seturile de noduri ramase pt a afla costul pt i, adica in functie de cate noduri am deja inaintea lui i <https://www.geeksforgeeks.org/travelling-salesman-problem-set-1/>
* greedy: construim APCM, consideram nodurile in ordinea parcurgerii DFS si in aceasta ordine vor forma un ciclu hamiltonian
* Problema rucsacului
* Problema planificarii echilibrate – NP-dificila
* Problema clicii -> **Acoperirea minima intr-un graf = Vertex cover** => NP-completa

acoperire minima = multimea minima de noduri care acopera maximul de muchii adica ce noduri se leaga prin cele mai multe muchii + restul nodurilor care mai sunt

Lightbox

pt ca nu exista muchie pt ca 3 acopera toate muchiile pt ca 4 are cele mai multe muchii

dar mai ramane muchia (0,2)

Alg de determinarea a acoperirii minime:

GFG:

pune toate muchiile intr-un set

cat timp mai exista muchii in set:

alege o muchie la intamplare => 2 noduri u si v adaugate in acoperirea min

sterge muchiile incidente cu nodurile u si v

* daca au mai ramas muchii, se rep proc si rez sunt nodurile din acop min

Problema clicii

clica = mulțime de vârfuri V' ⊆ V care formeaza un graf complet ( for every edge (u, v) of the graph, either ‘u’ or ‘v’ is in the vertex cover)

* daca se realiz graful complementar, clica sa e complementara clicii grafului normal