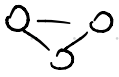
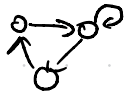
Introducere

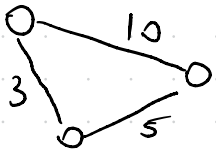
- utilizate în computer networks
- tipuri de graf → neorientat $(u,v) \equiv (v,u)$
  → orientat (digraph) $(u,v) \not\equiv (v,u)$

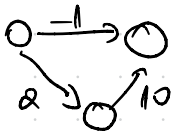




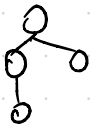
Grafuri ponderate

- au un cost per muchie (u, v, w)



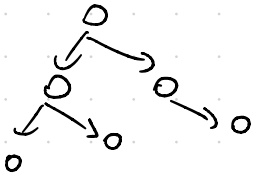


Arbori → graf neorientat fără cicluri



Arbori înrădăcinați → a rooted tree (Rooted Trees)

In-tree

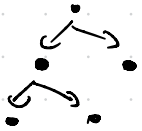


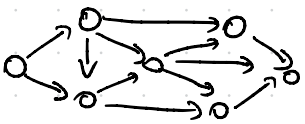
Out-tree

DAGs (Directected Acyclic Graphs)

- graf orientat
- nu are cicluri
- All out-trees are DAGs

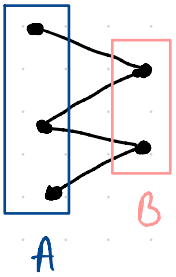
  Not all DAGs are out-trees





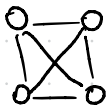
Grafuri bipartite

- vf. pot fi împărțite în 2 grupuri independente
- graful este two colourable
- nu există cicluri de lungime impară



Graf complet

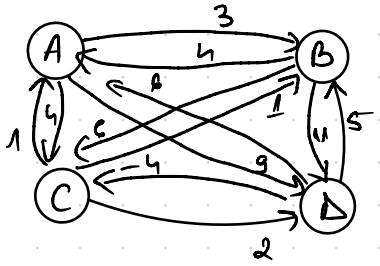
- grad nod : n-1



noduri conectate direct cu celelalte noduri

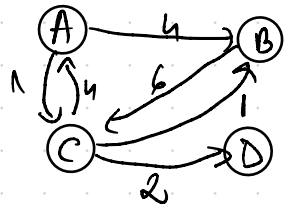
Moduri de a reprezenta grafurile

- matrice de adiacență



	A	B	C	D
A	0	4	1	9
B	3	0	6	11
C	4	1	0	2
D	6	5	-4	0

- listă de adiacență



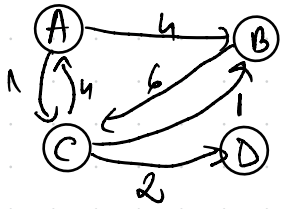
A: (B,4), (C,1)

B: (C,6)

C: (A,4), (B,1), (D,2)

D: []

- edge list

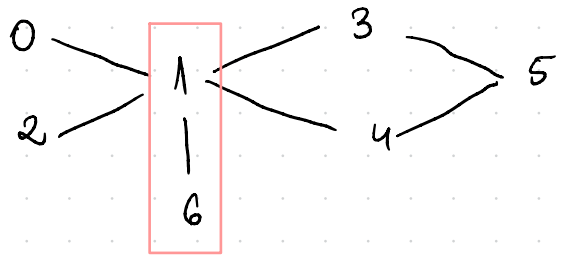


[(C,A,4), (A,C,1), (B,C,6), (A,B,4), (C,B,1), (C,D,2)]

Algoritmi

- the shortest path problem
  - BFS (unweighted graph)
  - Dijkstra
  - Bellman-Ford
  - Floyd Warshall
- Strongly connected components
  - Tarjan
  - Kosaraju
- flux maxim
  - Fulkerson, Edmonds Karp
- Connectivity
  - DFS
- Negative Cycles
  - Bellman-Ford
  - Floyd-Warshall
- prob. vânzătorului
  - Held-Karp
- Bridges, Articulation points
- Arbore minim de acoperire
  - Kruskal
  - Prim

Bridge in a graph

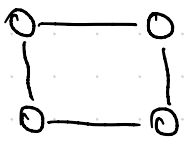


(1,6) is bridge

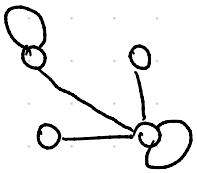
- o muchie care, dacă se elim., crește nr. componentelor conexe

Graf vs multigraf

graf simplu



multigraf

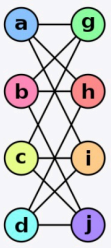
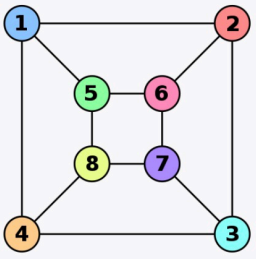


dif: multigraf: - muchii multiple (paralele, bucle)

Grafuri izomorfe

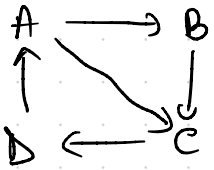
→ o corespondență între cele 2 grafuri

- același nr. de noduri
- același nr. de muchii
- gradul fiecărui nod este păstrat
- aceeași structură de adiacență

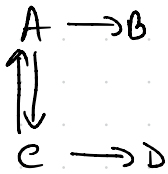
Graph G	Graph H	An isomorphism between G and H
		f(a) = 1 f(b) = 6 f(c) = 8 f(d) = 3 f(g) = 5 f(h) = 2 f(i) = 4 f(j) = 7

Conexitate

- graf conex = ∀ 2 vf. ∃ lanț
- graf tare conex = drum între ∀ 2 perechi de vf.
- graf slab conex = ∀ (u,v) ⇒ drum u-v dar ∄ drum v-u



graf tare conex



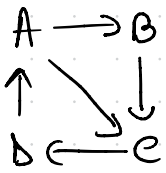
∄ drum de la B la D

NU este graf tare conex

Parcurgeri și drumuri în graf

- drum → simplu = nu folosește de 2 ori același arc
  - → compus, altfel
  - elementar = nu trece de 2 ori prin același vârf
- circuit = vf inițial = vf final
- drum → eulerian = trece prin toate arcele grafului
  - → hamiltonian = trece prin toate vârfurile

Matricea de incidență



	1	2	3	4	5	6
A	-1	0	0	1	-1	1
B	1	-1	0	0	0	0
C	0	1	-1	0	1	-1
D	0	0	1	-1	0	0

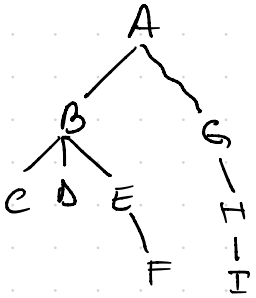
-1 iese din nod

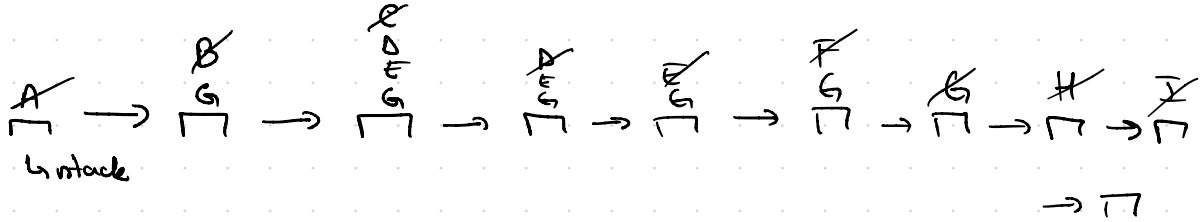
1 intră în nod

aferent A → B

DFS
(Depth First Search)

- O(V+E) complexitate
- utilizat în găsirea Bridge, Art. conectivitate
- parcurgere în adâncime (vertical, folosește stack)

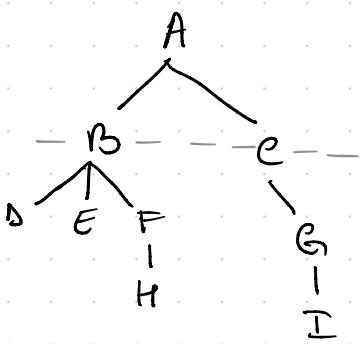




visited = A, B, C, D, E, F, G, H, I

BFS
(Breadth First Search)
↳ broad/wide

- căutare pe nivele (orizontal first)
- folosim coadă (FIFO)
- O(V+E) complexity
- finding the shortest path

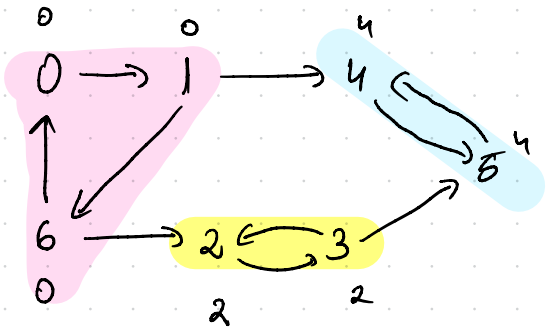


[A] → [C, B] → [F, E, D, B] → [G, F, E, B]
→ [G, F, E] → [G, F] → [H, G] → [I, H] →
→ [I] → []

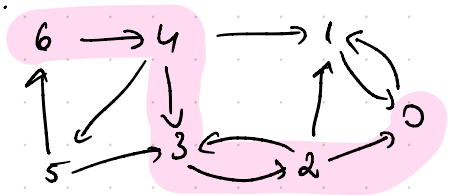
visited: A, B, C, D, E, F, G, H

Algoritmul Kosaraju

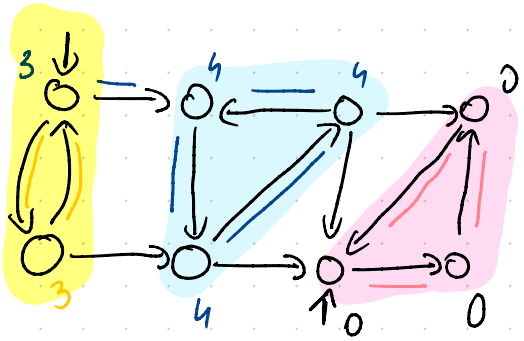
- det componentelor tare conexe, folosește DFS
- low link value of a node = the smallest node id reachable from that node



Probs: 0 ← fals!



! toate val. sunt 0 chiar dacă sunt mai multe componente conexe în graf

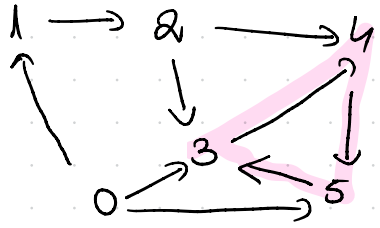


Rezolvare ⇒ Tarjan Algoritm

- aplic DFS

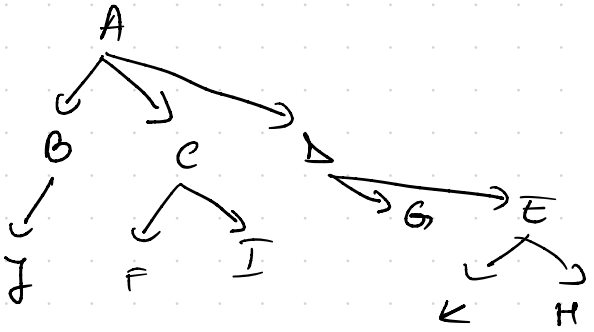
Sortare topologică

- ne oferă o ordine topologică a unui graf orientat
  ↳ aranjare liniară a nodurilor unui graf orientat aciclic
- fiecare muchie (u, v) → u apare înaintea lui v
- alg. bazat pe DFS
- soluția nu este unică
- nu fiecare graf are o ordine topologică

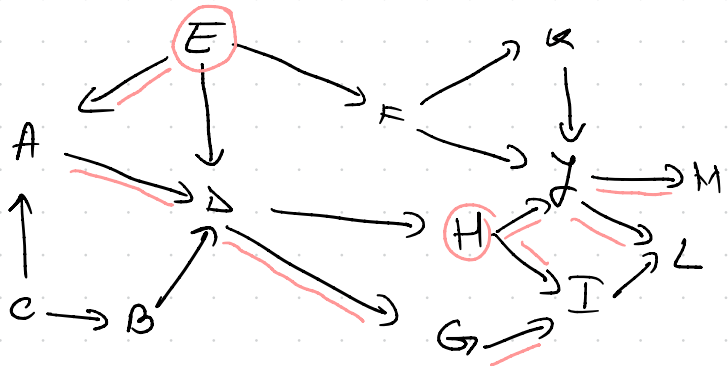


ciclu!
↳ nu ai de unde să începi
ca să ai cu găsi ne

- toți arborii au o ordine topologică (porniți de la frunze, nu cont. ord)



Ordine topologică:
A B C D E G H K J F I



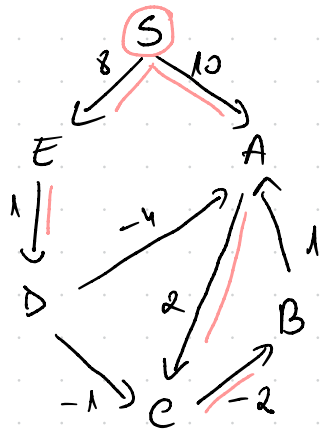
- se selectează un nod nevizitat (H)

DFS call Stack:
H, J, H ⇒ Backtrack → M în ord.
H, J, L ⇒ Backtrack → L în ord.
H, I ⇒ Backtrack

Topological order:
C B E F K A D G H I J L M

Algoritmul lui Bellman-Ford

- găsește pathul cel mai scurt de la nodul sursă la nodul destinație
- nu este cel mai eficient → Dijkstra mai bun
- nr. iterații = nr vf. - 1



Iterația 1

	S	A	B	C	D	E
	0	~~10~~	∞	∞	∞	~~8~~

→

	S	A	B	C	D	E
	0	10	∞	~~∞~~	∞	8

12 (10+2)

→

	S	A	B	C	D	E
	0	10	~~10~~	12	~~9~~	8

→

	S	A	B	C	D	E
	0	10	10	12	9	8

idk how to reach

Iterația 2

	S	A	B	C	D	E
	0	~~10~~ 5	10	~~12~~ 8	9	8

S → A → B → C → [D] → E

Iterația 3

S → [A] → B → [C] → D → E

	S	A	B	C	D	E
	0	5	~~10~~ 7	~~8~~	9	8

Iterația 4

nu se mai modif nimic

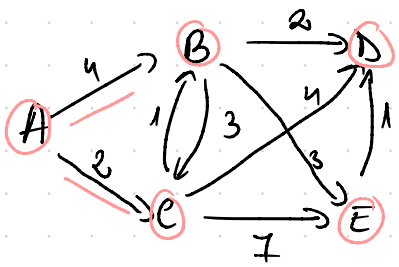
Rez:

	S	A	B	C	D	E
	0	5	5	7	9	8
Π:	nil	D	C	A	E	S

S → E → D → A → C → B.

Algoritmul lui Dijkstra

- non-negative weights, shortest path

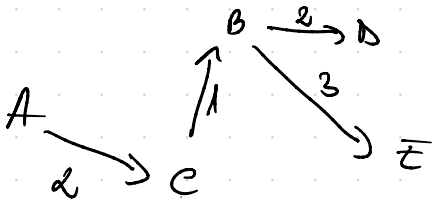


- mereu se alege pathul cel mai mic

Unvisited nodes: ~~A~~, ~~B~~, ~~C~~, ~~D~~, ~~E~~

src = A		3		5	6
d:	0	~~4~~	2	~~6~~	~~7~~
		~~∞~~	~~∞~~	~~∞~~	~~∞~~
	A	B	C	D	E
Π:	nil	~~A~~	A	~~C~~	~~C~~
		C		B	B

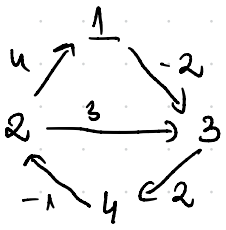
A → C → B → D → E



d:	0	3	2	5	6
	A	B	C	D	E
Π:	nil	C	A	B	B

Alg. Floyd - Warshall

- cele mai scurte drumuri între toate perechile de noduri
- se bazează pe distance array



	1	2	3	4
1	0	∞	-2	∞
2	4	0	~~3~~ 2	∞
3	∞	∞	0	2
4	∞	-1		0

1. $0 > 0 + \infty$
2. $\infty > 0 + \infty$
3. $-2 > 0 - 2$

⋮

$k = 1, i = 2, j = 3$

$3 > 4 + (-2)$

⇒ actualizez în mat. cu val. mai mică

$k = 1\ 2\ 3\ 4$

$i = 1\ 2\ 3\ 4$

$j = 1\ 2\ 3\ 4$

	1	2	3	4
1	0	-1	-2	0
2	4	0	2	4
3	5	1	0	2
4	3	-1	1	0

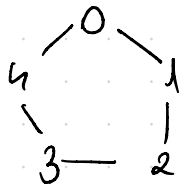
cond: $dist[i][j] > dist[i][k] + dist[k][j]$

Algoritmi pt shortest path

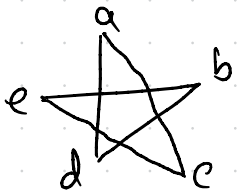
Dijkstra	Bellman - Ford	Floyd - Warshall
nod sre → nod dest	nod sre → nod dest + ponderi neg.	toate perechile de noduri + ponderi negative

Identificarea grafurilor izomorfe

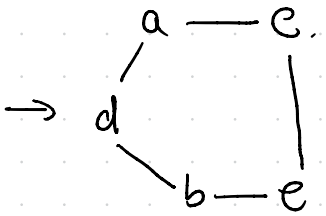
- structural la fel



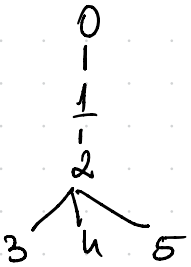
G_1



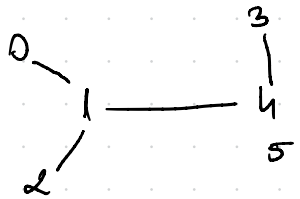
G_2



✓



G_1

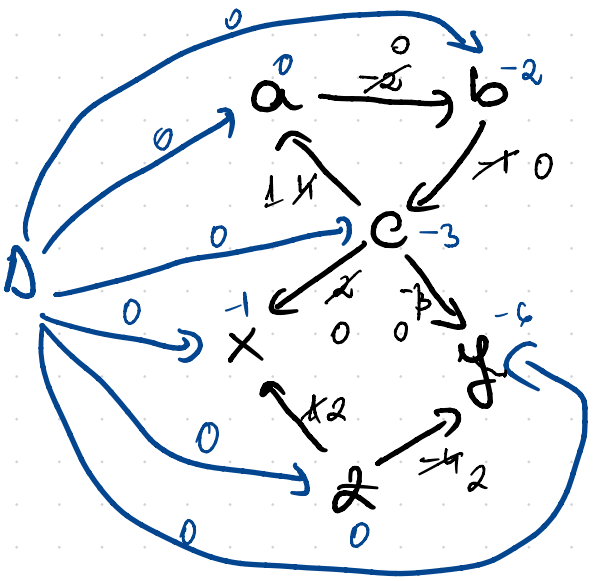


G_2

NU sunt izomorfe

Algoritmul lui Johnson

- alg eficient pt găsirea celor mai scurte drumuri între toate perechile nodurilor
- se adaugă un nod sursă fictiv → Bellman - Ford → reponderare: $w(u,v) + h(u) - h(v)$
  - ↓ ciclu negativ ⇒ se oprește
  - → pt fiecare nod: Dijkstra



d: 0 0 -2 -3 -1 -6 0

	Δ	a	b	c	x	y	z
π:	nil	Δ	a	b	c	c	Δ

reponderări muchie.

A-B: $-2+0+2=0$

c-A: $-3+4-0=1$

B-C: $-1-2+3=0$

c-x: $2-3+1=0$

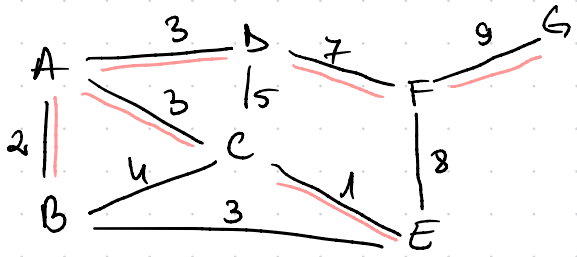
c-y: $-3-3+6=0$

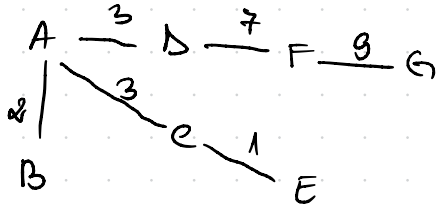
z-x: $1+0+1=2$

z-y: $-4+0+6=2$.

Algoritmul lui Kruskal

- Def. arborele minim de acoperire (with minimal cost)

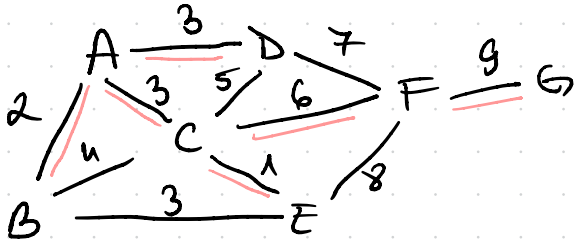


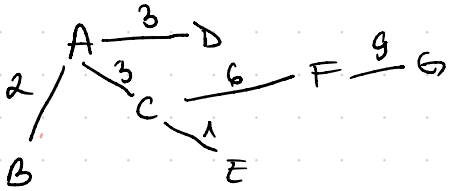


- se alege the smallest edge (C-E)
- sum. muchie cea mai mică care nu creează ciclu
- dacă sunt egale, se poate alege oricare
- dacă conectează 2 noduri ce sunt deja în calcul, nu se ia muchia.

Algoritmul lui Prim

- arborele de acoperire minim pe graf ponderat neorientat



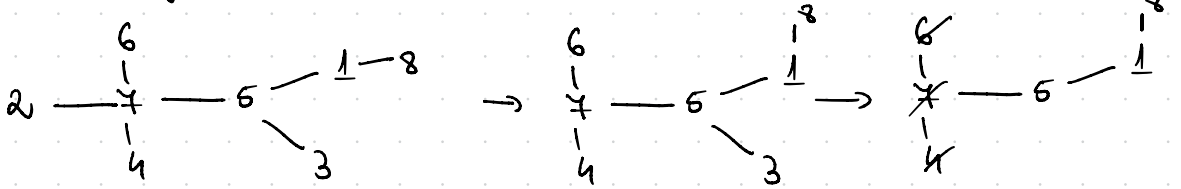


visited = A, B, C, E, D, F, G

- aleg un nod de start (A)
- aleg muchia cu costul cel mai mic
- dacă costurile sunt egale, se alege una random
- când ajung la E, A-D costul cel mai mic

Codare Prüfer

- se aleg frunzele (mereu cea mai mică val) și se trece părintele în listă



Codare: 7, 5, 7, 7, 5, 1

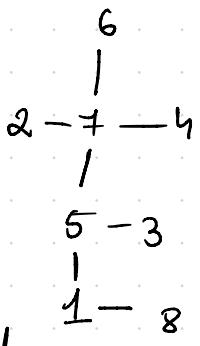
! când mai rămân doar 2 noduri, nu se mai trece nimic în listă

Decodare Prüfer

(~~7~~, ~~8~~, ~~7~~, ~~7~~, 5, 1) → n = 6 + 2 = 8 noduri

1, 2, 3, 4, 5, 6, 7, 8

nu apar în listă



- după ce terminăm cu 7 din listă, acesta devine disponibil pt a face pereche

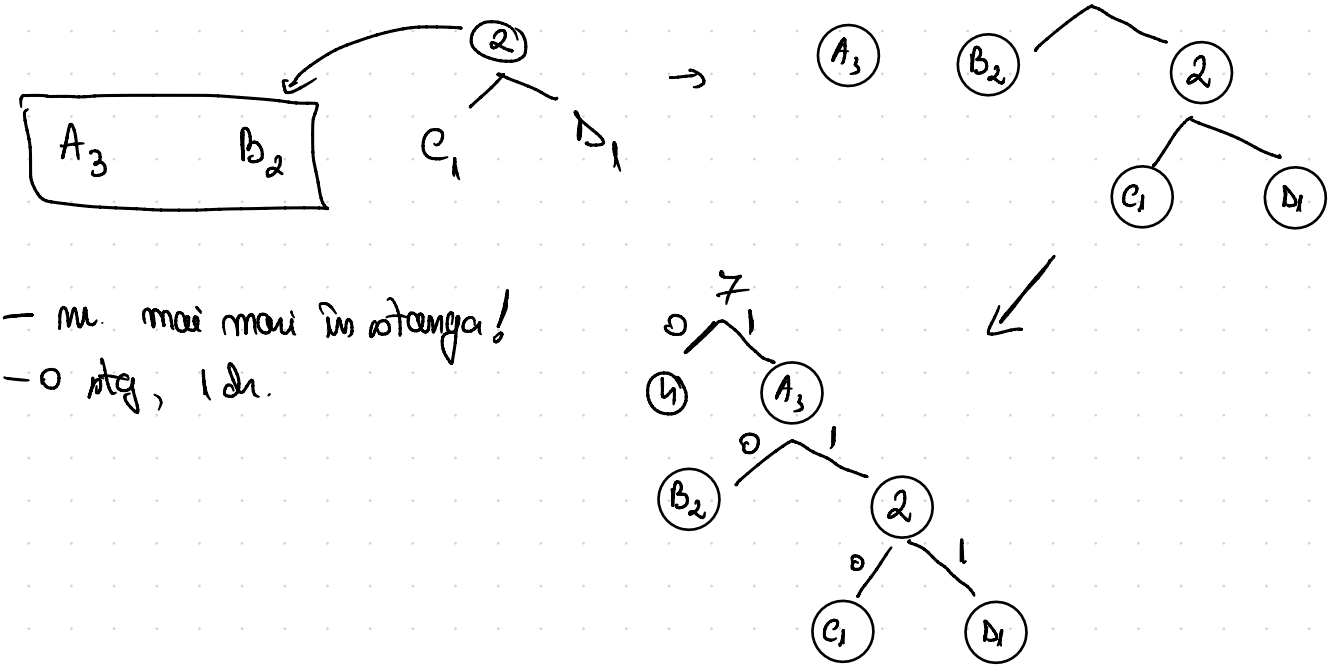
Huffman Coding

- codează cuvinte

AA B C BAD → 7 caractere

codare: 11 00 010 00 1011

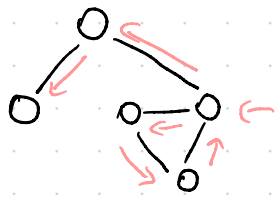
	freq.	code
A	3	1
B	2	00
C	1	010
D	1	011



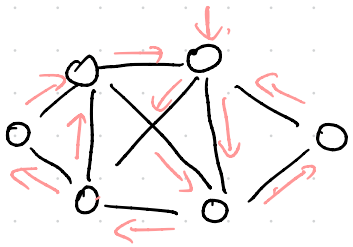
- nu mai mai în stânga!
- 0 stg, 1 dr.

Grafuri Euleriene

- vizitează toate muchiile o sg. dată



- circuit eulerian = începe și se termină la același vf.
- alg. Fleury

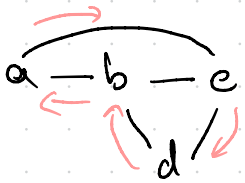


Condiții

	Circuit Eulerian	Drum Eulerian
Graf neorientat	grad fiecare nod – nr. par	fiecare vf grad par SAU doar 2 vf grad impar
Graf orientat	fiecare nod: grad intern = grad extern	max 1: → ext − int = 1, int − ext = 1

Grafuri Hamiltoniene

- trece prin toate nodurile o sg. dată
- ciclu hamiltonian: src = dest



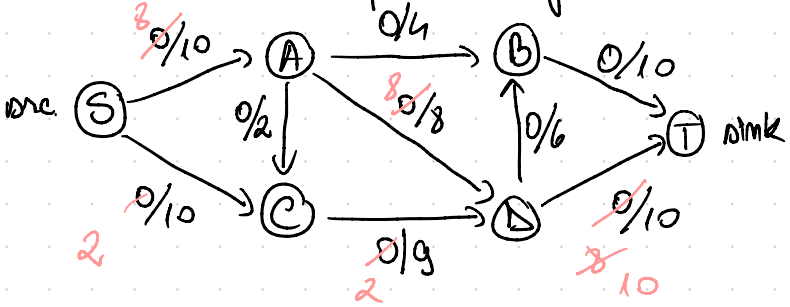
Cuplaj

= set de muchii fără noduri comune

- cuplaj maximal = nu se mai pot adăuga muchii ce să repete condiția
- cuplaj perfect = cuplaj în care toate nodurile sunt incluse
- cuplaj complet: nr. par de noduri
- alg. lui Karp-Hopcroft

Algoritmul Ford - Fulkerson

- maximum flow in a flow network

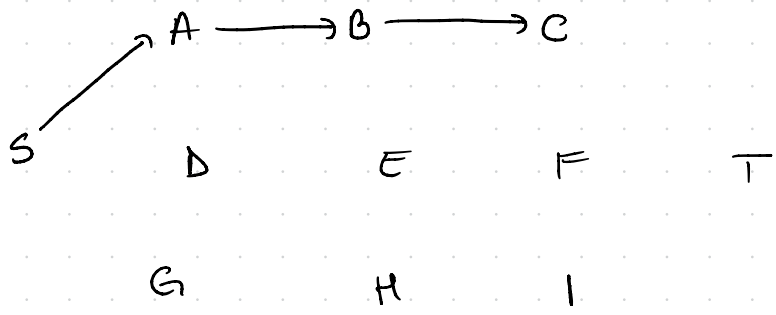


- utilizează DFS
- set the flow 0
- încerc să alg muchiile cu capacitate cât mai mare

1. S → A → D → T
2. S → C → D → T

Algoritmul Edmonds - Karp

- utilizează BFS



Drum critic

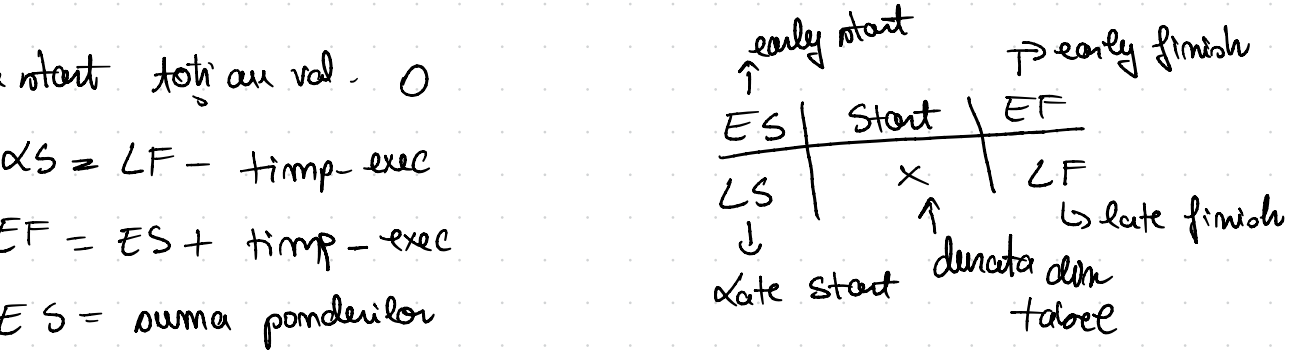
- la start toți au val. 0

$LS = LF - timp\_exec$

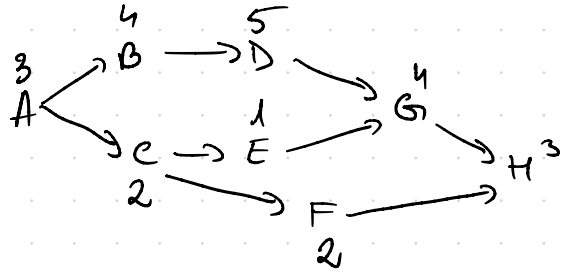
$EF = ES + timp - exec$

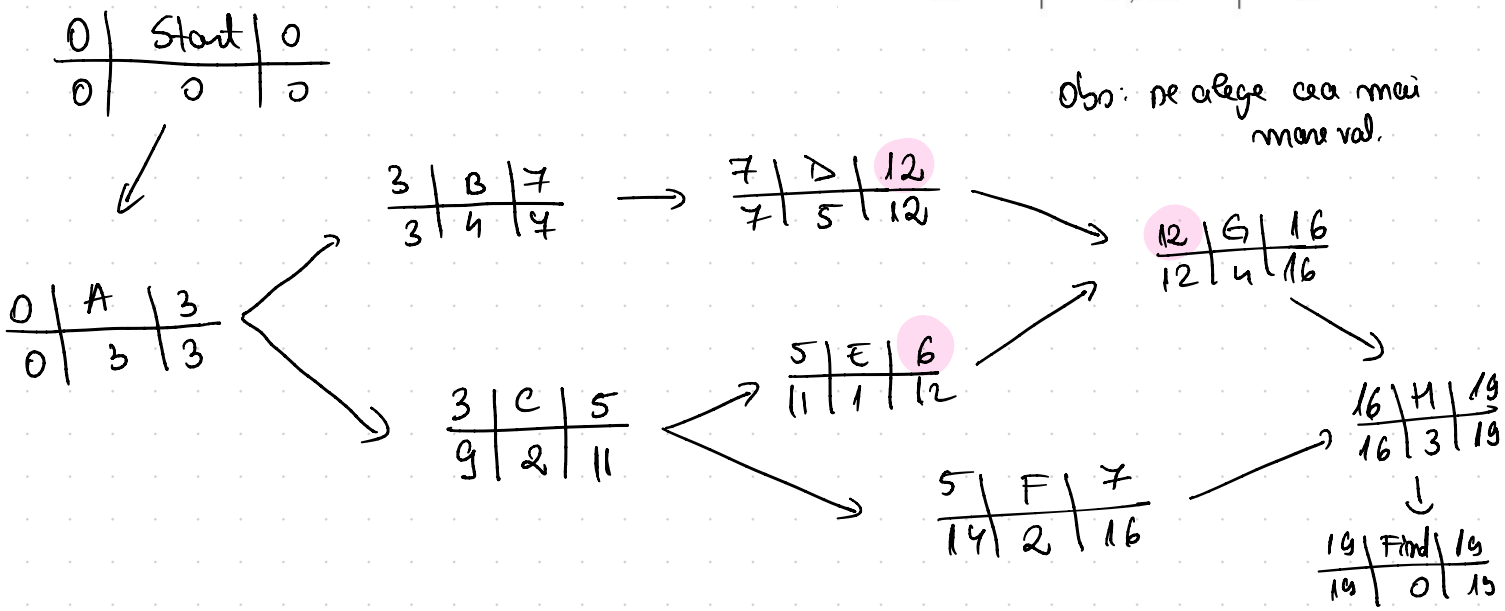
ES = suma ponderilor

LF = val minimă a drumului de la vf. final la activitatea i



activitate	predecesor	durata
A	-	3
B	A	4
C	A	2
D	B	5
E	C	1
F	C	2
G	D, E	4
H	F, G	3





Tips ~ Tricks + alte utile

- Prufer: să se det x din secvența Prufer: cu excepția rād, trb să apară de m. pe desen fiecare elem.

- $K_{n,n}$ are ciclu hamiltonian și cuplaj perfect

- K_n → graf complet, $K_{n,m}$ → graf bipartit