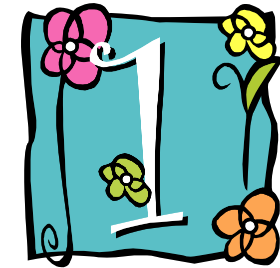
Backtracking

Ruxandra Stoean
http://inf.ucv.ro/~rstoean
ruxandra.stoean@inf.ucv.ro

Backtracking - exemplu



- Sa consideram programul:

```
natural(0).
natural(N) :- natural(N1), N is N1 + 1.
```

- Ce va genera Prolog-ul in urma unui apel de forma:

```
? – natural(N), write(N), nl, sleep(1), fail.
```

Backtracking - exemplu

- Se va genera un ciclu infinit:
  - numerele succesive sunt generate prin backtracking.
- Primul număr natural este generat şi afişat.
- Apoi *fail* forţeaza backtracking-ul să acţioneze.
- A doua clauză este apelată, generând numere naturale succesive.

Generare numere naturale

```
1 ?- natural(N), write(N), nl, sleep(1), fail.
0
1
2
3
4
5
6
7
8
9
10

Action (h for help) ? abort
% Execution Aborted
```

Un alt exemplu de backtracking

```
prefixN(N, [N]).
prefixN(N, [N|L]) :- N1 is N + 1, prefixN(N1, L).
```

- Pentru a folosi backtrackingul, avem nevoie de un apel de forma:

```
? - prefixN(1, L), write(L), nl, sleep(1), fail.
```

- Acesta generează o *infinitate* de liste care încep cu valoarea N.

Generare de liste incrementale care incep cu un element dat

```
1 ?- prefixN(1, L), write(L), nl, sleep(1), fail.
[1]
[1, 2]
[1, 2, 3]
[1, 2, 3, 4]
[1, 2, 3, 4, 5]
[1, 2, 3, 4, 5, 6]
[1, 2, 3, 4, 5, 6, 7]
[1, 2, 3, 4, 5, 6, 7, 8]
[1, 2, 3, 4, 5, 6, 7, 8, 9]
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

Action (h for help) ? abort
% Execution Aborted
```

Backtracking

- Backtracking-ul în Prolog este asadar foarte uşor de utilizat.
- Predicatul *fail/0* este cel care duce la forţarea backtrackingului:
  - el întoarce întotdeauna un rezultat negativ şi duce la reapelarea predicatelor care se află înaintea sa, ori de câte ori este posibil.

Permutari

- Sa generam permutările muţimii {a, b, c, d} folosind backtracking-ul in acest scop.
- Mai intai definim cele 4 litere:

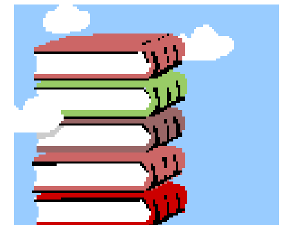
```
litera(a).
litera(b).
litera(c).
litera(d).
```

Permutari

```
permutare(X, Y, Z, T) :- litera(X), litera(Y), litera(Z),
   litera(T), X \= Y, X \=Z, Y \= Z, X \=T, Y\=T,Z \=T.

permutari :- permutare(X, Y, Z, T), write(X), tab(1),
   write(Y), tab(1), write(Z), tab(1), write(T), nl, fail.
permutari.
```

- Apelăm simplu, predicatul *permutari*:

```
? – permutari.
```

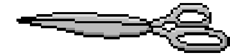


Permutari

- Iata permutarile rezultate:

```
3 ?- permutari.
a b c d
a b d c
a c b d
a c d b
a d b c
a d c b
b a c d
b a d c
b c a d
b c d a
b d a c
b d c a
c a b d
c a d b
c b a d
c b d a
c d a b
c d b a
d a b c
d a c b
d b a c
d b c a
d c a b
d c b a

Yes
```

Taietura (Cut)



- Uneori însă, din cauza backtracking-ului, sunt întoarse rezultate nedorite.
- Pentru evitarea acestora, Prologul ne vine în ajutor, prin ceea ce se numeşte taietura.
- Cu ajutorul acestui procedeu, Prologul oferă posibilitatea opririi backtrackingului.
- *Procedeul* se mai numeşte *cut* şi se notează cu semnul exclamării (**!**).

Taietura (Cut)

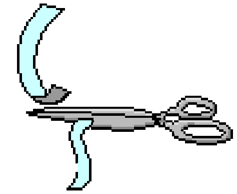
- Tăietura face ca Prologul să blocheze toate deciziile făcute până la momentul apariţiei sale (adică a semnului exclamării).

- Asta înseamnă că, dacă backtracking-ul era în desfăşurare, el va fi oprit şi nu se vor mai căuta alte alternative pentru ce se găseşte înainte de tăietură (!).

Exemplu

```
locul(1, simion).
locul(2, maria).
locul(3, sorin).
locul(3, cristian).

? – locul(3, Nume), write(Nume), nl, fail.
sorin
cristian

? – locul(3, Nume), !, write(Nume), nl, fail.
sorin
```

Mai adaugam:



```
despre(1, ' este castigatorul
   concursului.').
despre(1, ' primeste 100$.').
despre(2, ' a câştigat locul II.').
despre(2, ' primeste 50$.').
despre(3, ' a castigat locul III').
despre(3, ' primeste 25$.').
spune(Loc) :- locul(Loc, Cine),
   write(Cine), nl, despre(Loc,
   Despre), tab(3),
   write(Despre), nl, fail.
```

```
? – spune(3).
sorin
   a castigat locul III
   primeste 25$.
cristian
   a castigat locul III
   primeste 25$.
```

Cut

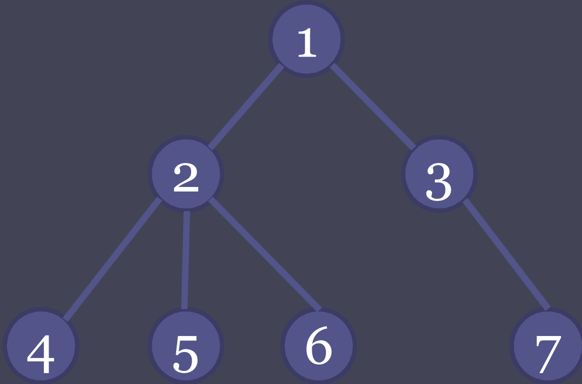
- Dacă vrem însă să ştim care este primul dintre cei care au luat locul trei şi câteva lucruri despre el, avem nevoie de tăietură - adăugăm la predicatul *spune/1* semnul exclamării astfel:

```
spune(Loc) :- locul(Loc, Cine), write(Cine), nl, !,
  despre(Loc, Despre), tab(3), write(Despre), nl,
  fail.
```

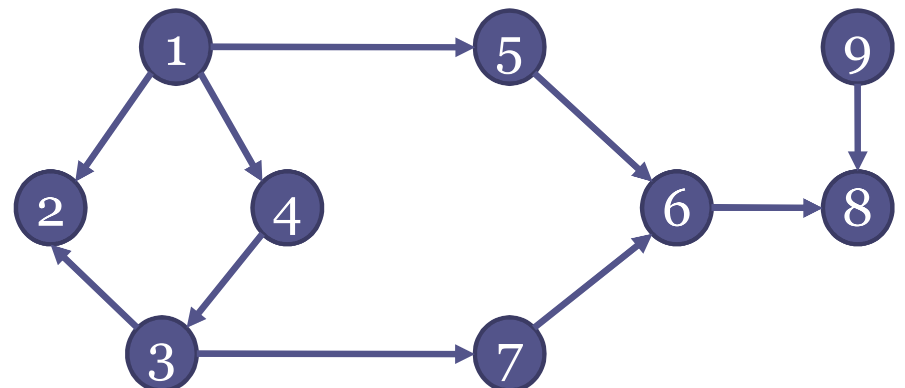
- Acum, după adăugarea făcută, backtrackingul se va aplica numai asupra predicatelor aflate dupa semnul exclamării.

Taietura

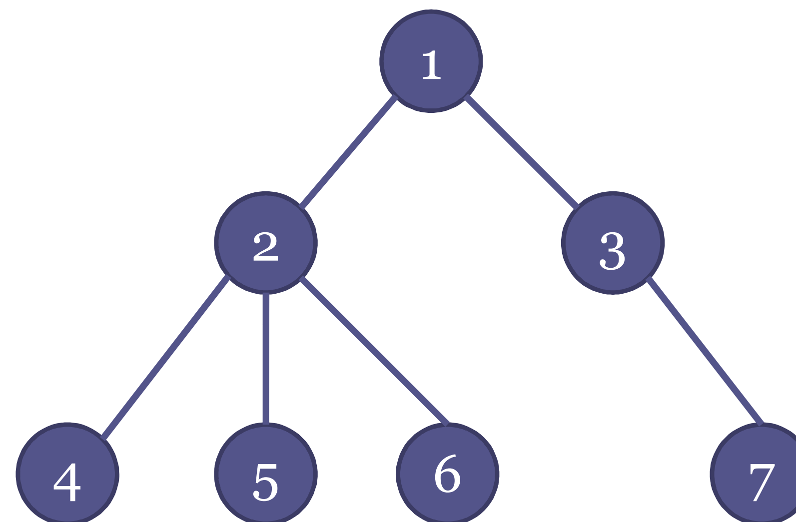
- Tăietura se foloseşte de obicei în interiorul unui predicat unde Prolog-ul a găsit primul răspuns posibil despre care se doresc mai multe detalii.
- Sau pentru a forţa un predicat să întoarcă un răspuns negativ (adică fail) într-o anumită situaţie şi nu vrem să caute alte soluţii.
- Tăietura însă, în general, măreşte complexitatea codului decât să o simplifice, iar utilizarea ei este bine să fie evitată pe cât posibil:
  - se spune chiar că ea reprezintă *goto*-ul programării logice.



Arbori si grafuri

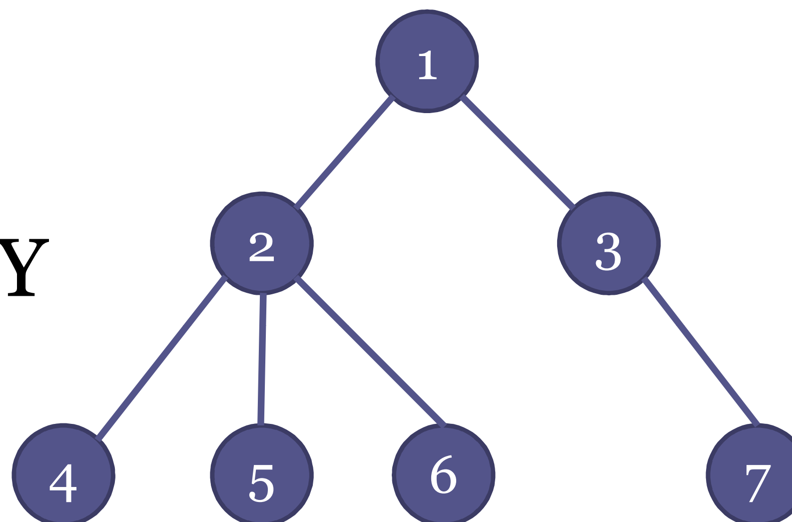


Arbori



- Cum reprezentam acest arbore?

Arbori

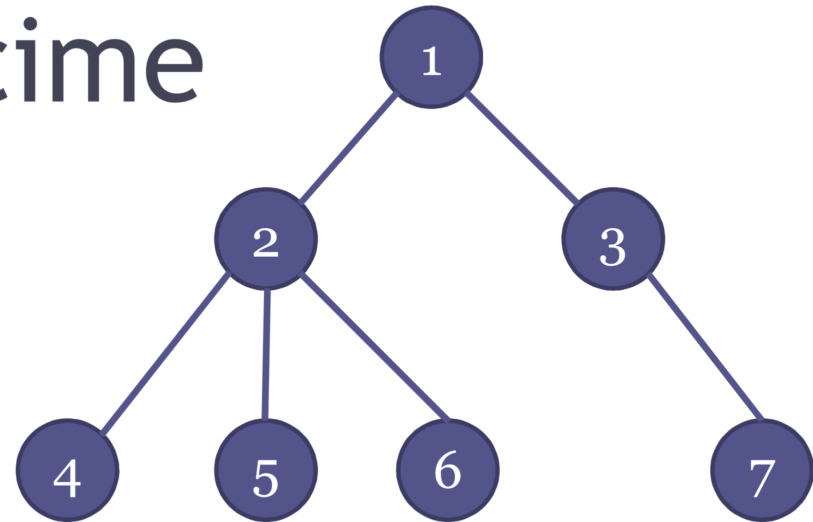


% tata(X,Y) - X este tatal lui Y

```
tata(0,1).
tata(1,2).
tata(1,3).
tata(2,4).
tata(2,5).
tata(2,6).
tata(3,7).
```

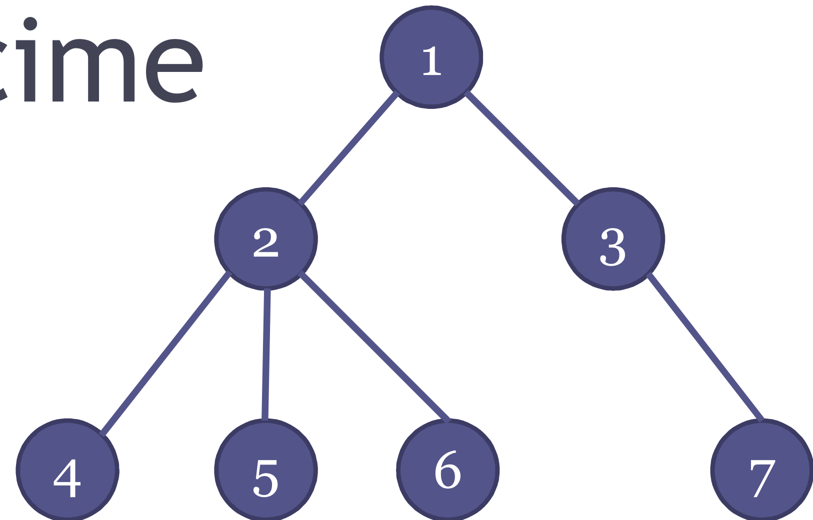
Aceasta confirma faptul ca 1 este radacina arborelui.

Parcurgere in adancime



- Care ar fi parcurgerea in adancime a acestui arbore?
- 1, 2, 4, 5, 6, 3, 7

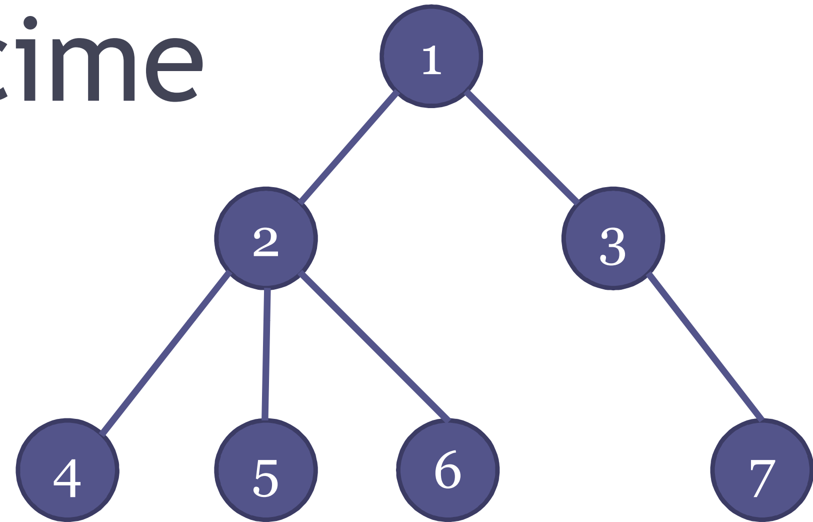
Parcurgere in adancime



Gaseste toti X care au o anumita relatie cu un Y dat; acestia se pun in lista L

```
findall(X, relatie(X,Y), L)

parcurgere:-tata(0, Rad), p([Rad]).

p([]).
p([Nod|Rest]) :- write(Nod), tab(2), findall(D,
   tata(Nod, D), LC), append(LC, Rest, Stiva),
   p(Stiva).
```

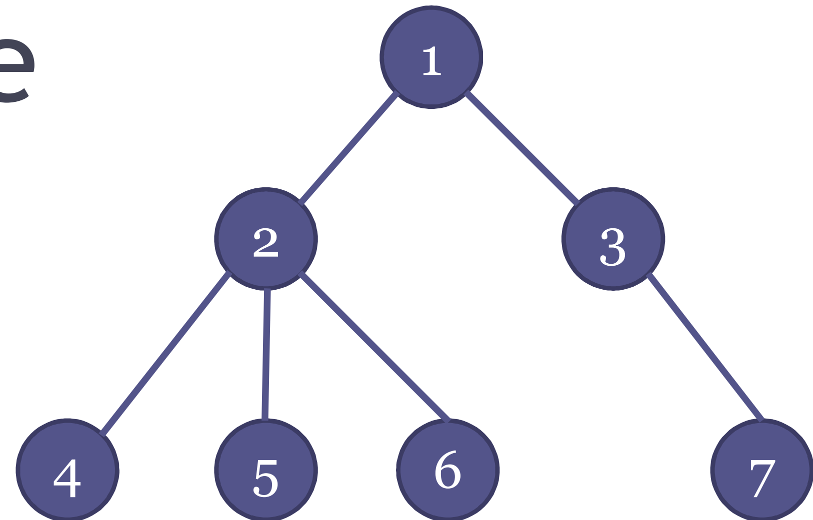
Parcurgere in adancime



?-parcurgere.

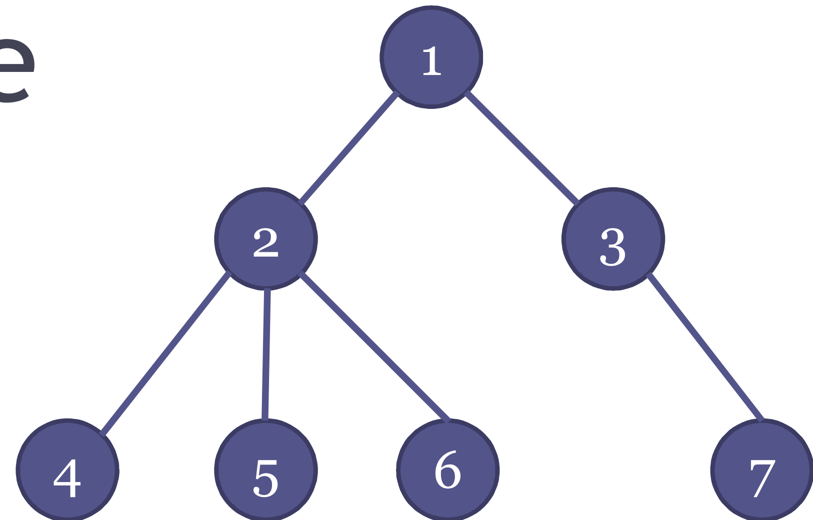
```
3 ?- parcurgere.
1 2 4 5 6 3 7

Yes
```

Parcurgere in latime



- Care ar fi parcurgerea in latime a acestui arbore?
- 1, 2, 3, 4, 5, 6, 7

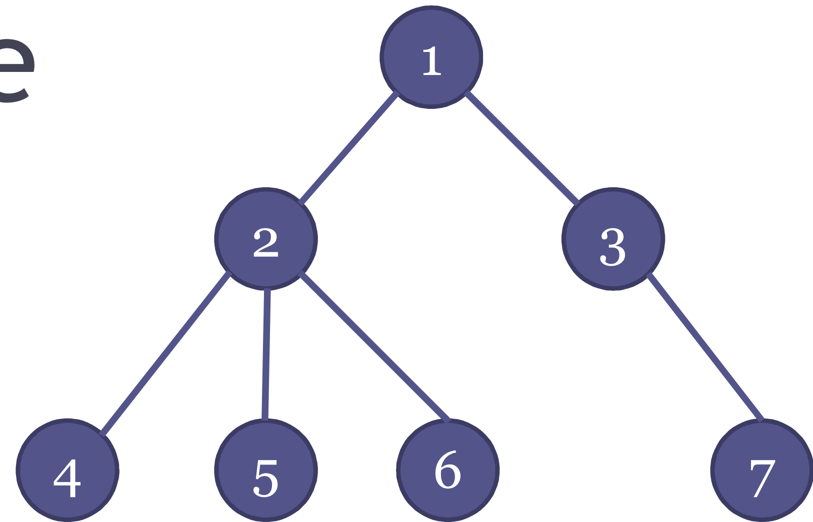
Parcurgere in latime




```
parcurgere_latime:-tata(0, Rad), c([Rad]).

c([]).
c([P|Rest]) :- write(P), tab(2), findall(D, tata(P,
  D), LC), append(Rest, LC, Coada), c(Coada).
```

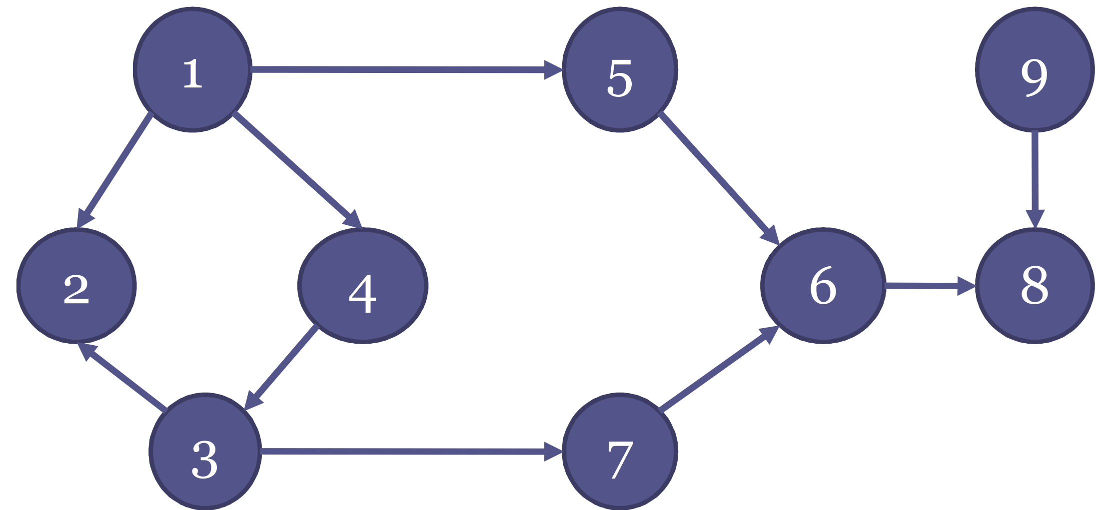
Parcurgere in latime

?-parcurgere_latime.



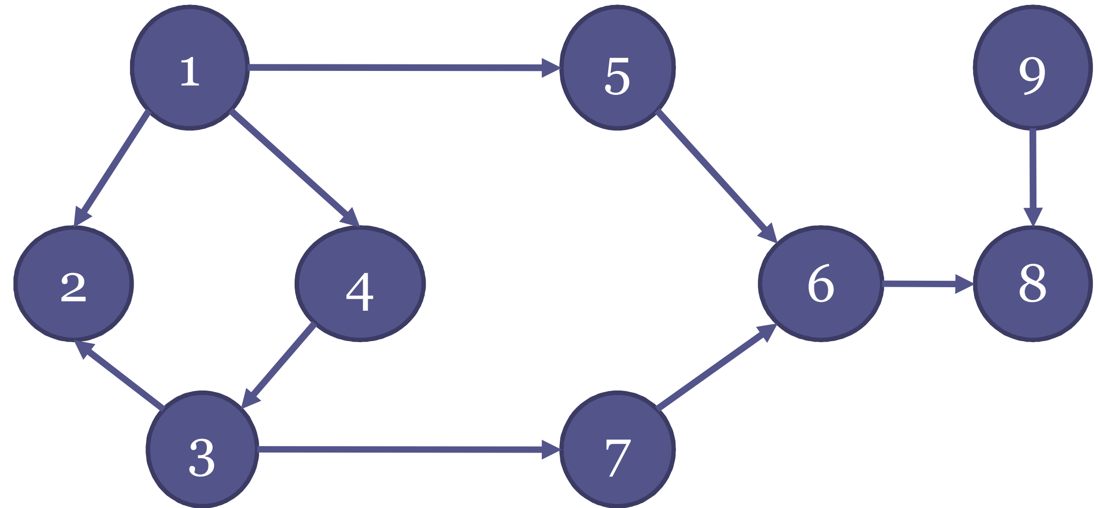

```
7 ?- parcurgere_latime.
1  2  3  4  5  6  7

Yes
```

Grafuri orientate



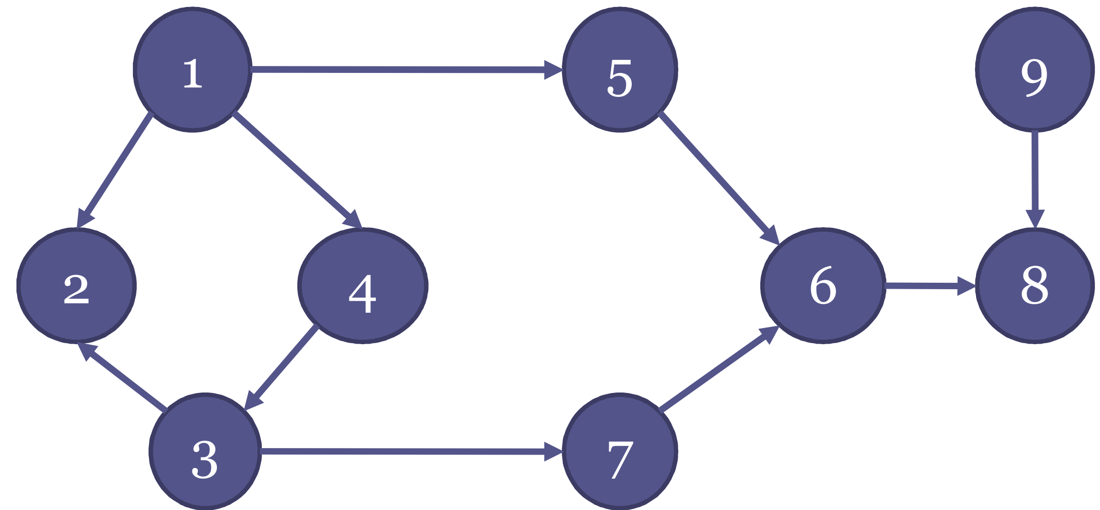
- Cum reprezentam acest graf?

Grafuri orientate



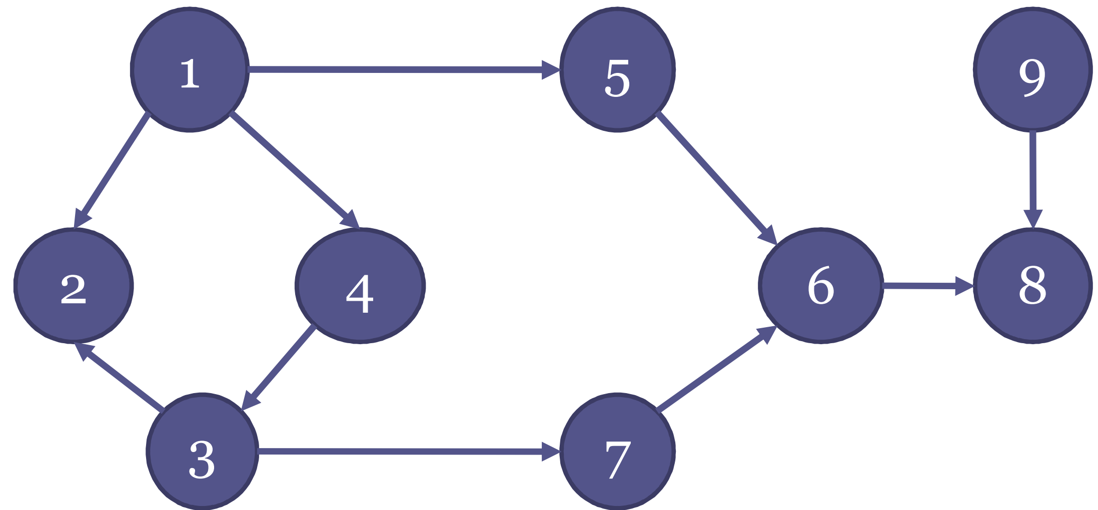
marc(1, 2).
marc(1, 4).
marc(1, 5).
marc(3, 2).
marc(3, 7).
marc(4, 3).
marc(5, 6).
marc(6, 8).
marc(7, 6).
marc(9, 8).

Grafuri orientate



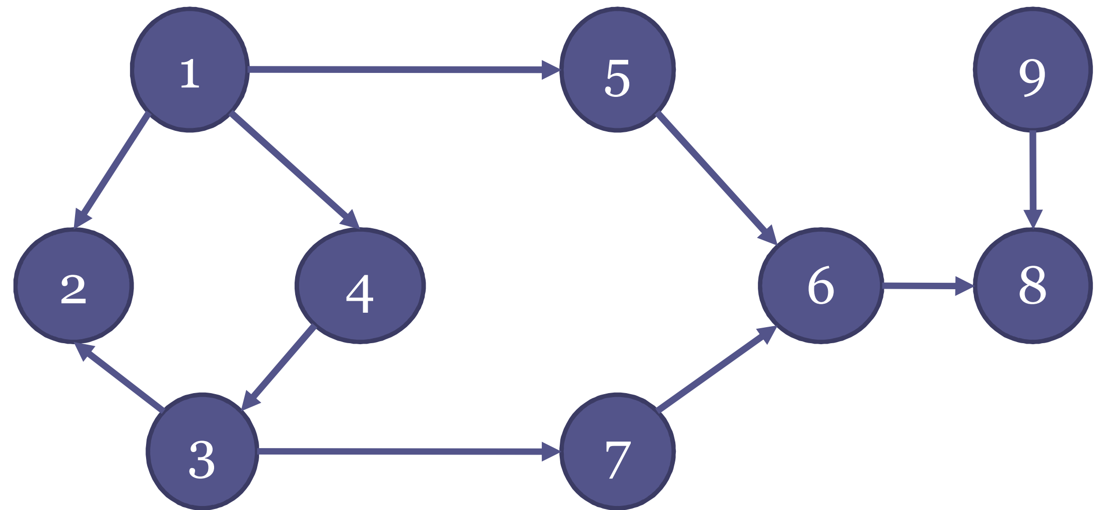
- Sa determinam toate drumurile intre doua noduri ale acestui graf.

Grafuri orientate




```
drum :- write('Introduceti nodul de start: '),
  read(X), write('Introduceti nodul destinatie: '),
  read(Y), drum(X, Y, L), write_ln(L), fail.
drum.
```

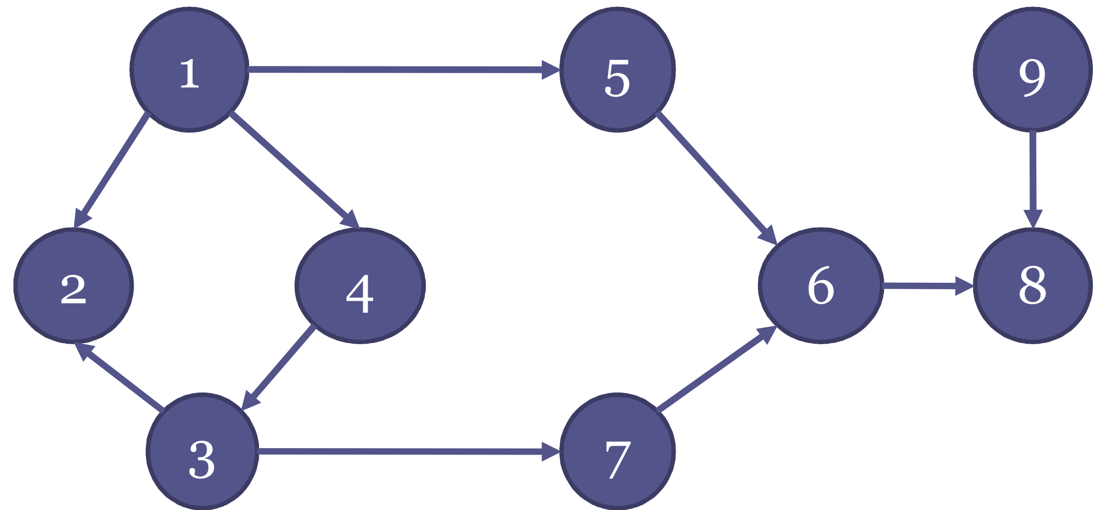
Grafuri orientate




```
drum(X, Y, L) :- drum(X, Y, [X], L).

drum(X, X, L, L).
drum(X, Y, L, Lista) :- marc(X, Z),
  not(member(Z, L)), append(L, [Z], L1), drum(Z,
  Y, L1, Lista).
```

Grafuri orientate

?- drum.



22 ?- drum.
Introduceti nodul de start: 1.
Introduceti nodul destinatie: 8.
[1, 4, 3, 7, 6, 8]
[1, 5, 6, 8]

Yes

Pe saptamana viitoare!