Universitatea Politehnica din București
Facultatea de Automatică și Calculatoare
Departamentul de Calculatoare

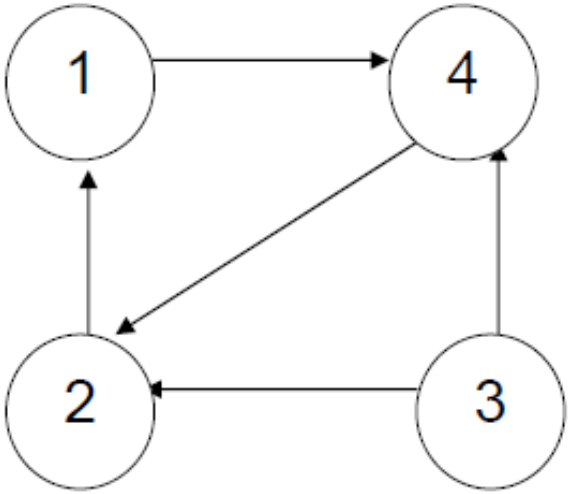
EXPLORAREA GRAFURILOR

Introducere

- **Explorarea** unui graf înseamnă **vizitarea tuturor nodurilor din graf**, folosind arcele existente, astfel încât să se treacă **o singură dată** prin fiecare nod
- Rezultatul explorării unui graf este o colecţie de arbori de explorare, numită şi "pădure" de acoperire

Introducere

- Rezultatul explorării unui graf orientat depinde de nodul de plecare
- Pentru graful orientat cu arcele (1,4),(2,1),(3,2),(3,4),(4,2) numai vizitarea din nodul 3 poate atinge toate celelalte noduri



Funcţia de explorare a grafului

- Primeşte ca parametru un nod de start şi încearcă să atingă cât mai multe noduri din graf
- Această funcţie poate fi apelată în mod repetat, pentru fiecare nod din graf, considerat ca nod de start

Observaţii

- Astfel, se asigură vizitarea tuturor nodurilor pentru orice graf
- Fiecare apel generează un arbore de acoperire a unei submulţimi de noduri
- Explorarea unui graf poate fi văzută ca o metodă de enumerare a tuturor nodurilor unui graf sau ca o metodă de căutare a unui drum către un nod dat din graf

Algoritmi de explorare

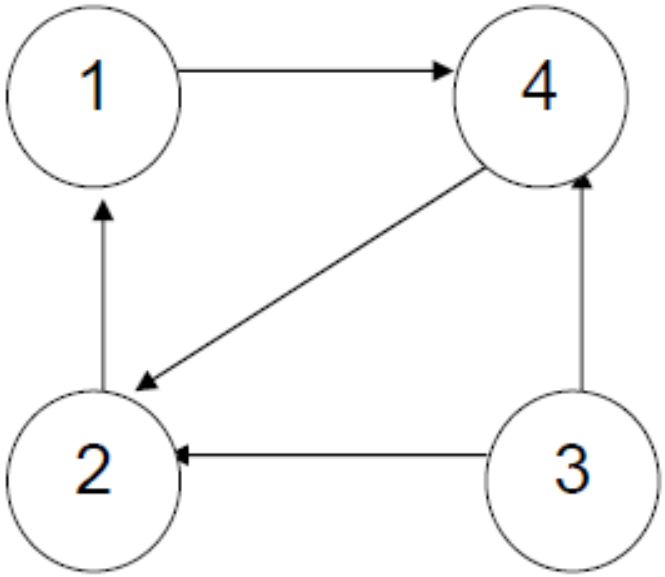
- Algoritmii de explorare dintr-un nod dat pot folosi două metode:
- 1) Explorarea în adâncime (**DFS** = Depth First Search)
- 2) Explorarea în lăţime (**BFS** = Breadth First Search)

Explorarea în adâncime

- Explorarea în adâncime foloseşte, la fiecare nod, un singur arc (către nodul cu număr minim) şi astfel se pătrunde cât mai repede în adâncimea grafului
- Dacă rămân noduri nevizitate, se revine treptat la nodurile deja vizitate, pentru a lua în considerare şi alte arce, ignorate în prima fază

Explorarea în adâncime

- Explorarea DFS din nodul 3 a grafului produce secvenţa de noduri 3, 2, 1, 4, iar arborele de acoperire este format din arcele 3-2, 2-1 şi 1-4

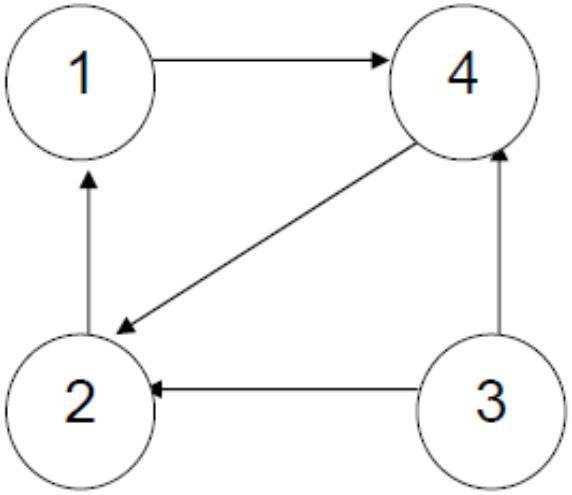


Explorarea în lăţime

- Explorarea în lăţime foloseşte, la fiecare nod, toate arcele care pleacă din nodul respectiv şi după aceea trece la alte noduri (la succesorii nodurilor vizitate)
- În felul acesta, se explorează mai întâi nodurile adiacente din "lăţimea" grafului şi apoi se coboară mai adânc în graf

Explorarea în lăţime

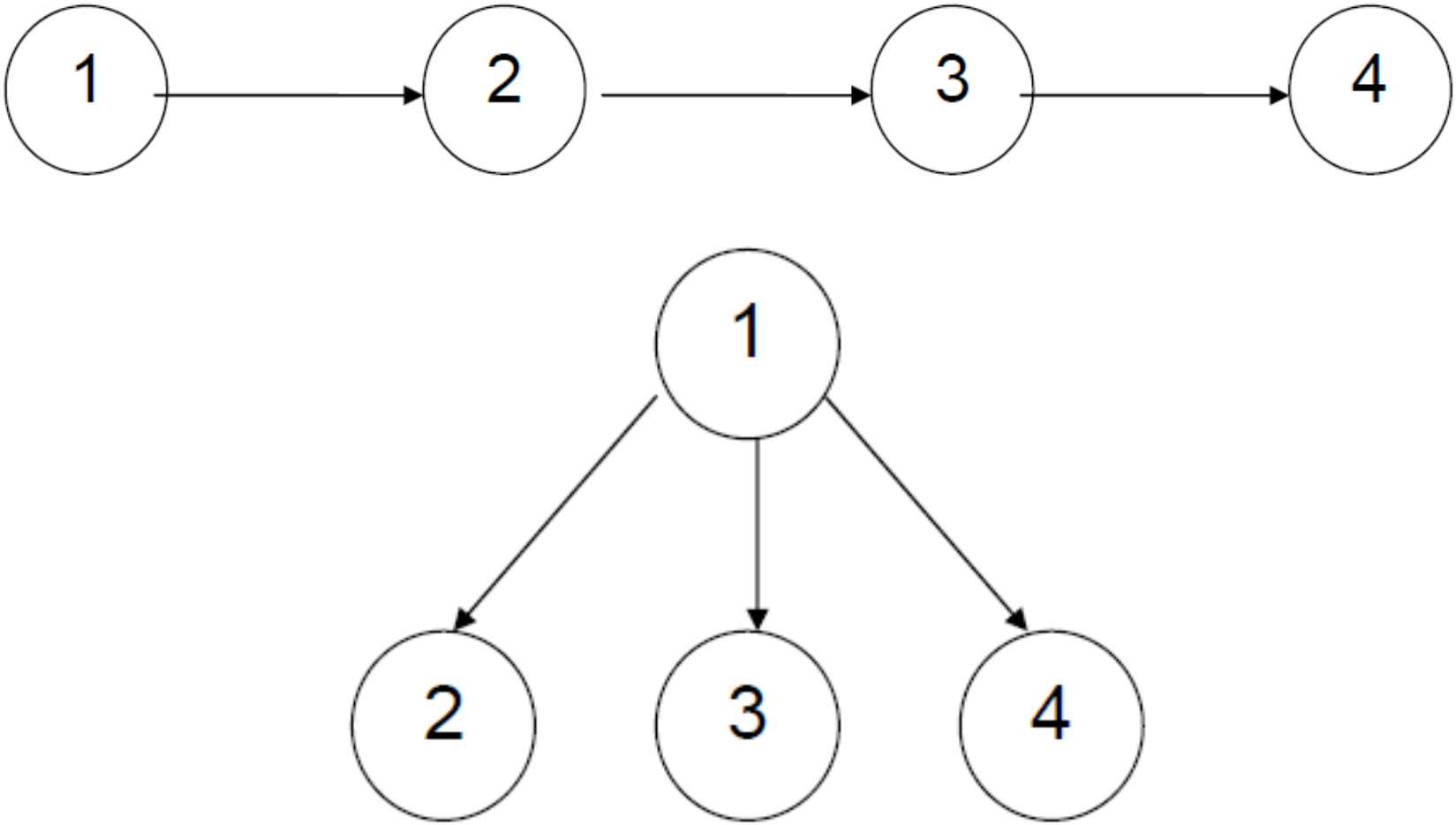
- Explorarea BFS din nodul 3 a grafului conduce la secvenţa de noduri 3, 2, 4, 1 şi la arborele de acoperire 3-2, 3-4, 2-1, dacă se folosesc succesorii în ordinea crescătoare a numerelor lor



Observaţii

- Este posibil ca pentru grafuri diferite să rezulte aceeaşi secvenţă de noduri
- Este posibil ca, pentru anumite grafuri, să rezulte acelaşi arbore de acoperire, atât la explorarea DFS, cât şi la explorarea BFS

Același arbore de acoperire la explorarea DFS și BFS



Algoritmul de explorare DFS

- Se poate exprima recursiv sau iterativ, folosind o stivă de noduri
- Ambele variante trebuie să ţină evidenţa nodurilor vizitate până la un moment dat, pentru a evita vizitarea repetată a unor noduri
- Cea mai simplă implementare a mulţimii de noduri vizitate este un tablou "văzut", iniţializat cu zerouri şi actualizat după vizitarea fiecărui nod x (*văzut[x] = 1*)

Funcţia recursivă de explorare DFS

```
// explorare DFS dintr-un nod dat v
void dfs (Graf g, int v, int vazut[]) {
 int  w, n = g.n;    // n = număr noduri din graful g
 vazut[v] = 1;       // marcare v ca vizitat
 printf ("%d ", v);          // afişare (sau memorare)
 for (w = 1; w <= n; w++)
 // repetă pentru fiecare vecin posibil w
   if (arc(g, v, w) && vazut[w] == 0)
         // dacă w este un vecin nevizitat al lui v
         dfs(g, w, vazut);  // continuă explorarea din w
}
```

Funcţia pentru vizitarea tuturor nodurilor

```
// explorare graf prin vizitarea tuturor nodurilor
// se pornește din primul nod
void df (Graf g) {
  int vazut[M] = {0};       // mulţime noduri vizitate
  int v;
  for (v = 1; v <= g.n; v++)
    if (!vazut[v]) {
      printf("\n Explorare din nodul %d \n", v);
      dfs(g, v, vazut);
   }
}
```

Observaţie

- Un algoritm DFS nerecursiv trebuie să folosească o stivă, pentru a memora succesorii (vecinii) neprelucraţi ai fiecărui nod vizitat, astfel încât să se poată reveni ulterior la aceştia

Pseudocod DFS iterativ

```
pune nodul de start în stivă
repetă cât timp stiva nu e goală
        scoate din stivă nodul x
        afişare şi marcare nodul x
        pune în stivă orice succesor nevizitat y al lui x
```

Observaţie

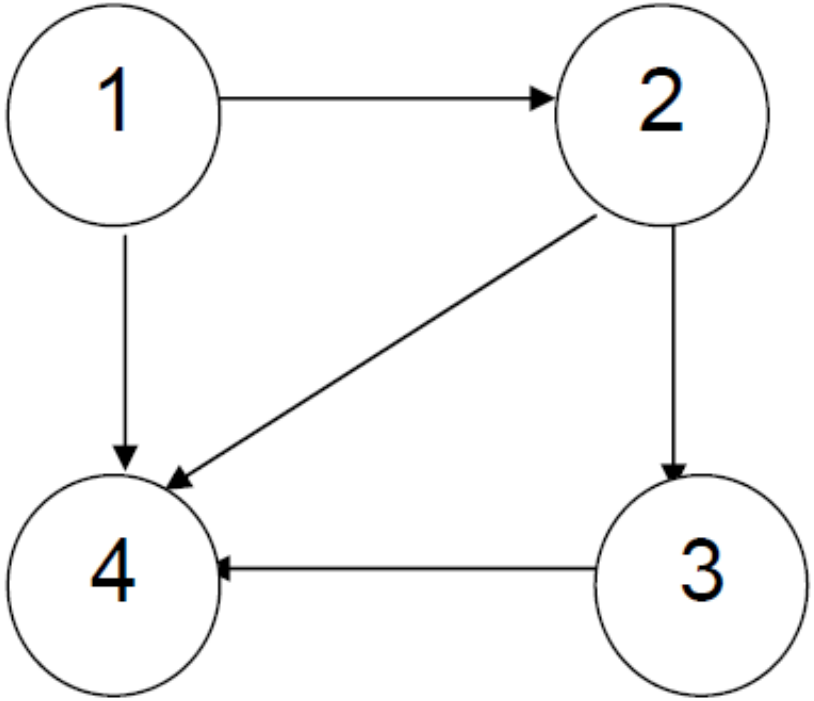
- Pentru ca funcţia DFS nerecursivă să producă aceleaşi rezultate ca şi funcţia DFS recursivă, succesorii unui nod sunt puşi în stivă în ordinea descrescătoare a numerelor lor, iar extragerea lor din stivă şi afişarea lor se va face în ordine inversă

Funcţia iterativă de explorare DFS

```
void dfs (Graf g, int v, int vazut[]) {
  int  x, y;
  Stack s;              // s este o stivă de numere naturale
  initSt(s);            // iniţializare stivă
  push(s, v);           // pune nodul v pe stivă
  while ( ! emptySt(s)) {
     x = pop(s);      // scoate din stivă nodul x
     vazut[x] = 1;  // marchează x ca vizitat
     printf("%d ", x);   // afişează nodul x
     for (y = g.n; y >= 1; y--)
         // caută un vecin al lui x, care este nevizitat
         if (arc(g, x, y) && ! vazut[y]) {
            vazut[y] = 1;
             push(s, y);       // pune nodul y pe stivă    } } }
```

Exemplu

- Se consideră graful cu arcele:
- (1,2), (1,4), (2,3), (2,4), (3,4)
- Evoluţia stivei şi a nodurilor x şi y este:



Stiva s	x	y	Afişare
1			
-	1		1
-	1	4	
4	1	2	
2, 4	1		
4	2		2
4	2	4	

Stiva s	x	y	Afişare
4, 4	2	3	
3, 4, 4	2		
4, 4	3		3
4, 4	3	4	
4, 4, 4	3		
4, 4	4		4
4	4		
-	4		

Algoritmul de explorare BFS

- Se afişează şi se memorează pe rând succesorii fiecărui nod
- Ordinea de prelucrare a nodurilor memorate este aceeaşi cu ordinea de introducere în coadă
- Algoritmul BFS este asemănător algoritmului DFS nerecursiv
- Diferenţa apare numai la tipul listei folosite pentru memorarea temporară a succesorilor fiecărui nod: stivă la DFS şi coadă la BFS

Funcţia de explorare în lăţime

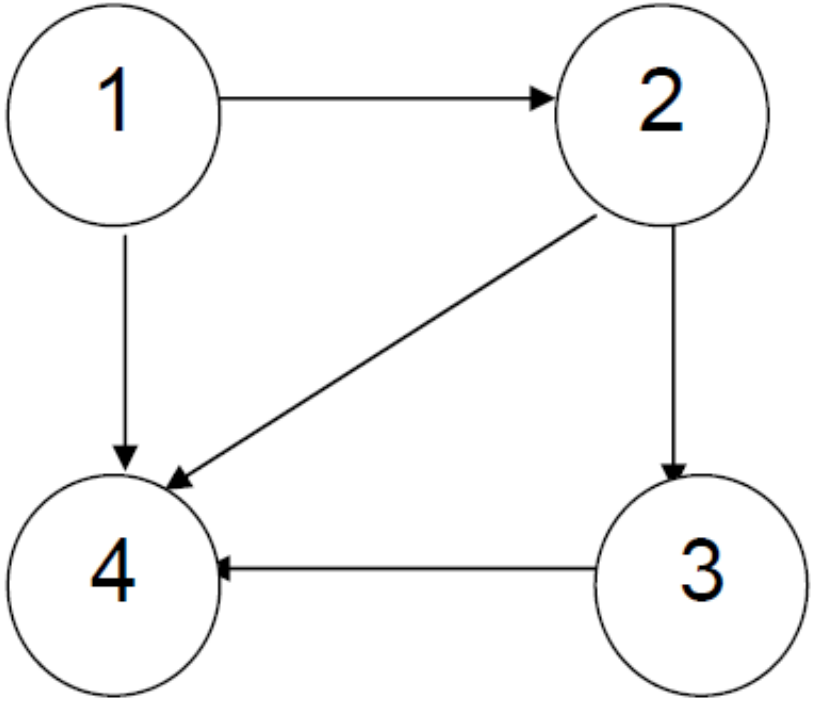
```
// explorare în lăţime dintr-un nod dat v
void bfs ( Graf g, int v, int vazut[]) {
 int  x, y;
 Queue q;   // q este o coadă de numere naturale
 initQ(q);
 vazut[v] = 1;  // marcare v ca vizitat
 enqueue(q, v);   // pune pe v în coadă
```

Funcţia de explorare în lăţime

```
while ( ! emptyQ(q)) {
   x = dequeue(q);                    // scoate pe x din coadă
   for (y = 1; y <= g.n; y++)
   // repetă pentru fiecare potenţial vecin cu x
     if (arc(g, x, y) && vazut[y] == 0) {
         // dacă y este vecin cu x şi nevizitat
       printf("%d - %d \n", x, y);  // scrie muchia x-y
       vazut[y] = 1;                 // y vizitat
       enqueue(q, y);                // pune pe y în coadă
     }
 }
}
```

Exemplu

- Se consideră graful cu arcele:
- (1,2), (1,4), (2,3), (2,4), (3,4)
- Evoluţia cozii şi a nodurilor x şi y este:



Coada q	x	y	Afişare
1			
-	1		
-	1	2	1 - 2
2	1	4	1 - 4
2, 4	1		
4	2		
4	2	3	2 - 3
4, 3	2		
4	3		
-	4		

Drum minim

- Un **drum minim** între două vârfuri este drumul care foloseşte cel mai mic număr de muchii
- Drumurile minime de la un vârf v la toate celelalte noduri pot fi găsite prin **explorare în lăţime** din nodul v, cu actualizarea distanţei faţă de v, la fiecare coborâre cu un nivel în graf
- Se foloseşte un tablou **d**, unde **d[y]** = distanţa vârfului y faţă de "rădăcina" v, precum şi un tablou **p**, cu **p[y]** = număr vârf predecesor pe calea de la v la y

Calcul distanţa minimă

```
//distanţa minimă de la v la toate celelalte noduri din graf
void dist_min (Graf g, int v, int vazut[], int d[], int p[]) {
 int  x, y;
 Queue q;
 initQ(q);
 vazut[v] = 1;
 d[v] = 0;
 p[v] = 0;
 enqueue(q, v);              // pune pe v în coadă
```

Calcul distanţa minimă

```
while ( ! emptyQ(q)) {
  x = dequeue(q);          // scoate pe x din coadă
  for (y = 1; y <= g.n; y++)
    if (arc(g, x, y) && vazut[y] == 0) {
      // dacă există arc între x şi y şi y nu a fost vizitat
      vazut[y] = 1;
      d[y] = d[x] + 1;       // y este un nivel mai jos decât x
      p[y] = x; // x este predecesorul lui y pe drumul minim
      enqueue(q, y);
    }
  }
}
```

Observaţii

- Pentru afişarea vârfurilor de pe un drum minim de la v la x, trebuie parcurs în sens invers tabloul p (de la ultimul element la primul):
- x → p[x] → p[p[x]] →… → v