PA - Grafuri

Daniel Chiș - 2022, UPB, ACS, An I, Seria AC

Grafuri

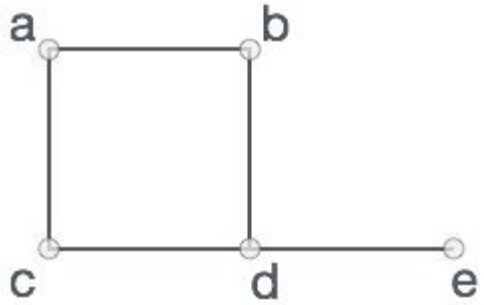
Grafuri Neorientate

Un graf neorientat este o reprezentare a unui set de obiecte conectate prin link-uri, formând o pereche de mulțimi (N,M). Obiectele sunt reprezentate de noduri (vertex) iar conexiunile dintre ele de muchii (edges).

Un graf are următoarele elemente:

- Mulțimea nodurilor N
- Mulțimea muchiilor M -
- Gradul nodului - numărul de muchii formate cu ajutorul nodului respectiv
- Nod izolat - Un nod ce nu formează nici o muchie
- Noduri terminale - Un nod ce formează o singură muchie
- Noduri adiacente - Noduri între care există o muchie
- Nod și muchie incidente - Nodul face parte dintr-o muchie
-

Grafuri



N = {a, b, c, d, e}

M = {ab, ac, bd, cd, de}

Lista de adiacență:

- A: B->C
- B: A->D
- C: A->D
- D: B->C->E
- E: D

Matrice de adiacență	A	B	C	D	E
A	0	1	1	0	0
B	1	0	0	1	0
C	1	0	0	1	0
D	0	1	1	0	1
E	0	0	0	1	0

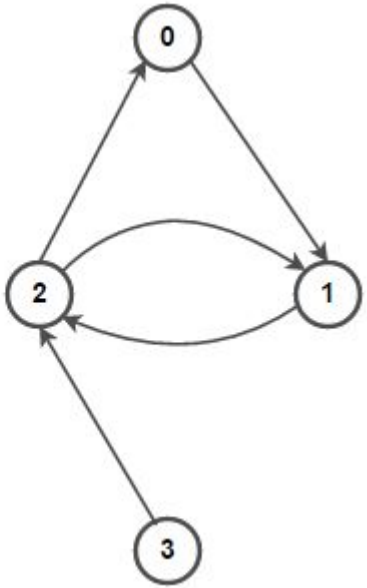
Grafuri Orientate

Un graf orientat este o pereche de mulțumi G={N,M} unde:

- N reprezintă mulțimea finită și nevidă a nodurilor
- M reprezintă mulțimea de perechi ordonate de elemente ce fac parte din N, numită mulțimea arcelor

Un graf orientat are:

- Gradul interior al unui nod: numărul de arce care intră în nod
- Gradul exterior al unui nod, numărul de arce care ies din nod



```
// Data structure to store a graph object
struct Graph
{
    // An array of pointers to Node to represent an adjacency list
    struct Node* head[N];
};

// Data structure to store adjacency list nodes of the graph
struct Node
{
    int dest;
    struct Node* next;
};

// Data structure to store a graph edge
struct Edge {
    int src, dest;
};
```

```c
// Function to create an adjacency list from specified edges
struct Graph* createGraph(struct Edge edges[], int n)
{
    unsigned i;

    // allocate storage for the graph data structure
    struct Graph* graph = (struct Graph*)malloc(sizeof(struct Graph));

    // initialize head pointer for all vertices
    for (i = 0; i < N; i++) {
        graph->head[i] = NULL;
    }

    // add edges to the directed graph one by one
    for (i = 0; i < n; i++)
    {
        // get the source and destination vertex
        int src = edges[i].src;
        int dest = edges[i].dest;

        // allocate a new node of adjacency list from `src` to `dest`
        struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
        newNode->dest = dest;

        // point new node to the current head
        newNode->next = graph->head[src];

        // point head pointer to the new node
        graph->head[src] = newNode;
    }

    return graph;
}
```

```c
// Function to print adjacency list representation of a graph
void printGraph(struct Graph* graph)
{
    int i;
    for (i = 0; i < N; i++)
    {
        // print current vertex and all its neighbors
        struct Node* ptr = graph->head[i];
        while (ptr != NULL)
        {
            printf("(%d -> %d)\t", i, ptr->dest);
            ptr = ptr->next;
        }

        printf("\n");
    }
}

// Directed graph implementation in C
int main(void)
{
    // input array containing edges of the graph (as per the above diagram)
    // `(x, y)` pair in the array represents an edge from `x` to `y`
    struct Edge edges[] =
    {
        { 0, 1 }, { 1, 2 }, { 2, 0 }, { 2, 1 }, { 3, 2 }, { 4, 5 }, { 5, 4 }
    };

    // calculate the total number of edges
    int n = sizeof(edges)/sizeof(edges[0]);

    // construct a graph from the given edges
    struct Graph *graph = createGraph(edges, n);

    // Function to print adjacency list representation of a graph
    printGraph(graph);

    return 0;
}
```

Parcurgere Grafuri

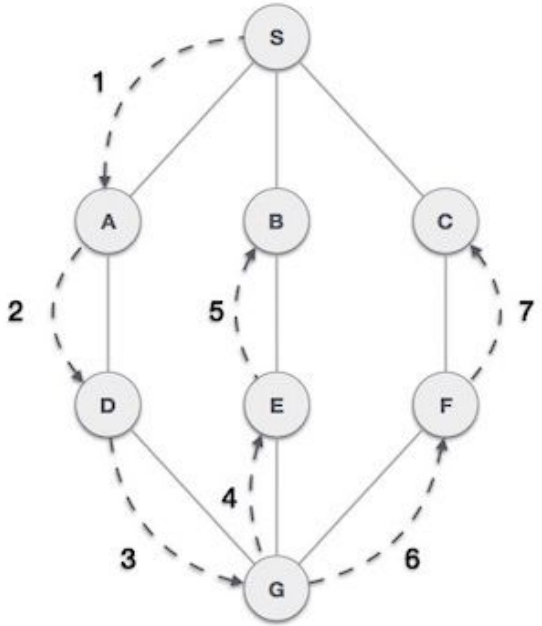
Parcurge în adâncime - Depth First Search

DFS este un algoritm care parcurge grafurile în adâncime, folosindu-se de o stivă pentru a ține minte următorul nod la care trebuie să ajungă.

Pas 1: alegem un nod de la care plecăm

Pas 2: vizităm nodurile adiacente nevizitate, le afișăm și le punem în stivă

Pas 3: dacă pentru un nod nu există vecini nevizitați facem pop din stivă și încercăm din nou până stiva este goală



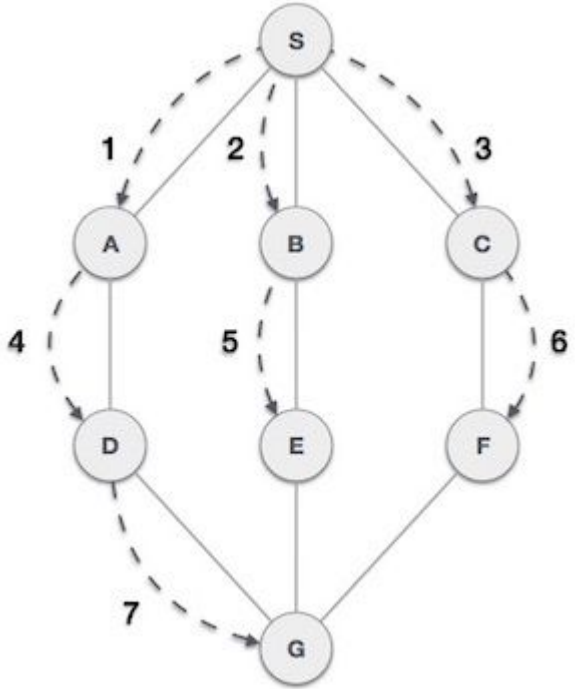
Parcurge în lățime - Breadth First Search

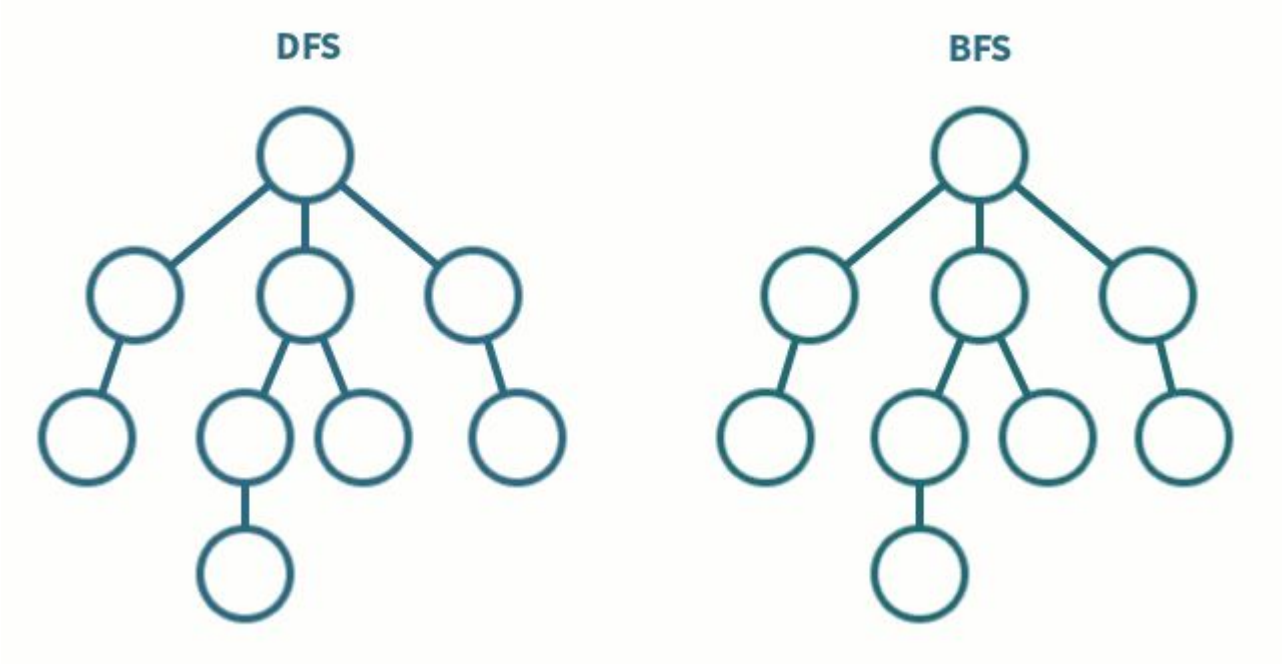
BFS este un algoritm care parcurge grafurile în lățime, folosindu-se de o coadă pentru a ține minte următorul nod la care trebuie să ajungă.

Pas 1: alegem un nod de la care plecăm, îl afișăm și îl punem în coadă

Pas 2: vizităm toți vecinii nodului sursă pe care îi punem în coadă. Când nu mai există scoatem nodul sursă din coadă

Pas 3: repetăm pasul 2 până golim coada





DFS bs BFS

Exerciții

Exerciții

1. Realizați un graf pe care să îl parcurgeți atât cu BFS cât și cu DFS. Separat să aveți și graful desenat și matricea de adiacență. 5p
2. Realizați un algoritm care să detecteze dacă într-un graf orientat există un drum între două noduri date. 4p

Exerciții FIIR

Realizați un graf pe care să îl parcurgeți atât cu BFS cât și cu DFS. Separat să aveți și graful desenat și matricea de adiacență.

Resurse: DFS in C BFS in C