Laborator 11. Grafuri. Reprezentari, parcurgeri

Un graf este o structura de date format din noduri și muchii – legături intre noduri. Grafurile sunt folosite pentru a modela relațiile dintre perechi de obiecte , de exemplu drumurile dintr-o țara intre diferite orase (nodurile sunt orasele, muchiile sunt drumurile). Grafurile pot avea costuri pe muchii sau nu, în funcție de ceea ce se dorește să se reprezinte.

Grafurile pot fi neorientate (muchie intre x și y înseamnă și muchie intre y și x) sau orientate (cum e și cazul drumurilor unidirectionale într-un oraș).

Grafurile pot fi implementate în 2 moduri:

1. Matrice de adiacenta – dacă avem n noduri, se formeaza o matrice nxn, în care m[i][j] = 1 dacă avem muchie de la nodul i la nodul j. Pentru parcurgerea vecinilor unui nod trebuie parcursa toată linia din matrice (complexitate ridicată).

Graful va fi reprezentat în felul următor:

```
typedef struct
{
        int nn;
        int **Ma;
}       TGraphM;
```

2. Liste de adiacenta – fiecare nod are o lista cu vecini (nodurile către care exista muchii directe). Aceasta este metoda optima de reprezentare.

Graful va fi reprezentat în felul următor:

```
typedef struct node
{
        int v;
        TCost c;
        struct node *next;
} TNode, *ATNode;

typedef struct
{
        int nn;
        ATNode *adl;
}       TgraphL;
```

Exista 2 parcurgeri uzuale ale unui graf:

1. Parcurgerea în adâncime- DFS. Se pornește de la un nod, de exemplu nodul 1. Se vizitează toți vecinii acestuia, și în momentul în care am găsit un vecin continuam parcurgerea de la acesta. Se poate implementa recursiv (când găsim un vecin care nu a fost marcat apelam recursiv DFS din vecin) sau nerecursiv, folosind o stiva.

2. Parcurgerea în latime- BFS. Se pornește de la un nod de start, care este adaugat într-o coada. Se vizitează toți vecinii nodului, apoi nodul este șters din coada. Vecinii sunt adăugați în coada dacă nu au fost vizitati deja, apoi se reia parcurgerea cât timp exista noduri în coada.

În cadrul laboratorului, vom lucra cu un graf neorientat (muchiile sunt bidirectionale, când se adauga muchia x->y trebuie adaugata și muchia y->x).

Aveți de implementat următoarele funcții:

alloc_matrix – aloca matricea de adiacenta pentru un graf reprezentat cu matrice de adiacenta
alloc_list - aloca vectorul de liste pentru un graf reprezentat cu liste de adiacenta

insert_edge_matrix, insert_edge_list – adauga o muchie în graf, în funcție de reprezentare. Aveți grija ca muchia x->y înseamnă ca trebuie creata și legătura inversa, y->x
dfs_stack – parcurgere în adâncime cu stiva (nerecursiv), folosind matrice de adiacenta
dfs_rec- parcurgere în adâncime folosind recursivitate și matrice de adiacenta
dfs_rec_list – parcurgere în adâncime folosind recursivitate și liste
bfs_queue- parcurgere în latime folosind liste de adiacenta

delete_node_list, delete_node_matrix – stergerea unui nod și a muchiilor care îl contin la unul din capete dintr-un graf, în funcție de reprezentare

free_matrix, free_list – eliberarea memoriei pentru structurile folosite