Structuri de date

Rotaru Alexandru Andrei
rotarualexandruandrei94@gmail.com
2019-2020

University Politehnica of Bucharest

1 Resurse

Laborantul va preciza ce exercitii veti avea de rezolvat. Pentru rezolvarea fiecarui exercitiu veti folosi un fisier separat. **Optional**, va puteti verifica folosind checker-ul (daca este pus la dispozitie).

1.1 Vizualizare

Pentru a putea visualiza o parte din algoritmi si intelege mai bine cum functioneaza structurile de date accesati:
`https://www.cs.usfca.edu/~galles/visualization/Algorithms.html`

1.2 Link-uri catre platforme online cu probleme

- Hackerrank: `https://www.hackerrank.com/domains/data-structures`
- Leetcode: `https://leetcode.com/problemset/all/`

1.3 Tool-uri de debug

- GDB: `https://cs.baylor.edu/~donahoo/tools/gdb/tutorial.html`
- Valgrind: `http://valgrind.org/docs/manual/quick-start.html`

1.4 Feedback

Pentru a semnala probleme sau a oferi sugestii pentru laboratoarele urmatoare:
`https://forms.gle/8bRw7mPgqaJ9tRzd6`

2 Exerciții

2.1 Exercițiul 1 [2 puncte]

Fiind data structura de graf reprezentata prin matrice de adiacenta să se implementeze operațiile de adăugare/ștergere/verificare de muchii.

Listing 1: Graph base operations

```
typedef struct {
    size_t node_count;
    bool **adj_matrix;
} undirected_graph_t;

// se vor implementa in fisierul exercitiul1.cpp
void graph_insert_edge(undirected_graph_t * const,
    const size_t, const size_t);

void graph_remove_edge(undirected_graph_t * const,
    const size_t, const size_t);

bool graph_is_edge(undirected_graph_t * const,
    const size_t, const size_t);
```

2.2 Exercițiul 2 [3 puncte]

Să se implementeze algoritmii de explorare a grafurilor: DFS și BFS. Aceștia vor primii ca parametru o funcție ce trebuie apelată cu nodul curent dat ca parametru.

- dfs recursiv
- dfs iterativ, folosind o stiva
- bfs iterativ, folosind o coada

Listing 2: Semnăturile funcțiilor

```
1  void graph_traverse_DFS_recursive(
2      undirected_graph_t * const graph,
3      bool * const visited,
4      const size_t start_node_id,
5      visit_func_t func);
6
7  void graph_traverse_DFS_iterative(
8      undirected_graph_t * const graph,
9      const size_t start_node_id,
10     visit_func_t func);
11
12 void graph_traverse_BFS(
13     undirected_graph_t * const graph,
14     const size_t start_node_id,
15     visit_func_t func);
```

2.3 Exercițiul 3 [5 puncte]

Să se implementeze o funcție care determină cel mai scurt drum intre 2 noduri intr-un graf neorientat. Vectorul returnat trebuie să fie alocat dinamic, iar pentru a determina lungimea lui se va adăuga la final valoarea -1.

Pentru un vector de lungime 4:
$12, 2, 6, 3, -1$

Listing 3: Semnătura funcției

```
1  int *shortest_path(
2      undirected_graph_t * const graph,
3      const size_t source_id,
4      const size_t destination_id);
```

2.4 Exercițiul 4 [3 puncte]

Sa se implementeze o funcție ce număra cate componente conexe exista intr-un graf neorientat. O componenta conexă este un subgraf cu proprietatea că există cel puțin un drum intre oricare 2 noduri din acel subgraf.

Listing 4: Semnătura funcției

```
1 size_t connected_componenets(undirected_graph_t *graph);
```