ΟΝΟΜΑ: Μηλτιάδης
ΕΠΩΝΥΜΟ: Μαντές
ΑΜ: 1084661
ΕΤΟΣ: 2

6

ΑΣΚΗΣΗ ΕΡΓΑΣΤΗΡΙΟΥ : 6

- **ΣΧΟΛΙΑΣΜΟΣ ΛΥΣΗΣ**

Αρχικά, ορίζουμε μια συνάρτηση create_arrays( ) με ορίσματα μια int μεταβλητή v που συμβολίζει τον κόμβο του γραφήματος από όπου εξέρχεται η ακμή και μια λίστα graph[ ] τύπου <int> που συμβολίζει όλο το κατευθυνόμενο γράφημα. Στο σώμα της συνάρτησης δημιουργούμε πρώτα τα δύο ζητούμενα διανύσματα nodeOutEdges και graphOutEdges τύπου <int> καθώς και έναν iterator it για να μπορούμε να διατρέξουμε τη λίστα που αντιπροσωπεύει το γράφημα. Επίσης, αφήνουμε τη θέση 0 του nodeOutEdges κενή περνώντας σαν όρισμα στην push_back( ) το -1. Με ένα βρόχο while ξεκινάμε να διατρέχουμε τη λίστα όσο ο μετρητής i είναι μικρότερος από το v για να εξασφαλίσουμε ότι ο κόμβος στον οποίο βρισκόμαστε κάθε φορά ανήκει στο γράφημα. Έπειτα, με ένα βρόχο for ξεκινάμε από τη πρώτη θέση του nodeOutEdges και αρχίζουμε να προσθέτουμε στις διαδοχικές θέσεις του διανύσματος τη τιμή του j η οποία κάθε φορά αντιπροσωπεύει τη θέση που βρίσκεται η πρώτη εξερχόμενη ακμή του εκάστοτε κόμβου εντός του graphOutEdges. Με έναν εσωτερικό βρόχο for ξεκινάμε να διατρέχουμε τη λίστα - γράφημα και να αποθηκεύουμε στις διαδοχικές θέσεις του διανύσματος graphOutEdges τη τιμή του δείκτη it η οποία κάθε φορά αντιπροσωπεύει τον κόμβο, στον οποίο καταλήγουν οι εξερχόμενες ακμές από τον εκάστοτε κόμβο στον οποίο βρισκόμαστε. Ακόμα, εκχωρούμε στη μεταβλητή j μέσα στον ίδιο βρόχο το μέγεθος του πίνακα graphOutEdges, το οποίο μεταβάλλεται κάθε φορά που προσθέτουμε ένα νέο κόμβο και έξω από τον βρόχο while εκχωρούμε στη θέση 0 του nodeOutEdges που είχαμε αφήσει κενή τη θέση που βρίσκεται ο τελευταίος κόμβος στον graphOutEdges (j - 1). Στο τέλος, εκτυπώνουμε και τους δύο πίνακες ολοκληρωμένους.

- **ΚΩΔΙΚΑΣ ΛΥΣΗΣ**

```
#include<iostream>
#include<list>
#include<iterator>
#include <algorithm>
#include <vector>
using namespace std;
```

```
void displayGraph(list<int> graph[], int v)  // display
the Graph
{
   for(int i = 1; i<v; i++) {
      cout << i << "--->";
      list<int> :: iterator it;
      for(it = graph[i].begin(); it != graph[i].end();
++it) {
         cout << *it << " ";
      }
      cout << endl;
   }
}

void insert_edge(list<int> graph[], int u, int v)  //add
vertices (u v), (v u)
{
   graph[u].push_back(v);
}

void create_arrays(int v, list<int> graph[])
{
    vector<int> graphOutEdges;
    vector<int> nodeOutEdges;
    list<int> :: iterator it;

    int i;
    int j = 0;

    nodeOutEdges.push_back(-1);
    while(i<v)
    {
        for(i=1; i<v; i++)
        {
            nodeOutEdges.push_back(j);
            for(it = graph[i].begin(); it !=
graph[i].end(); ++it) graphOutEdges.push_back(*it);

            j=graphOutEdges.size();
        }
    }

    nodeOutEdges[0]=j-1;

    cout << "\nnodeOutEdges : ";
    for (i = 0; i < nodeOutEdges.size(); i++) cout <<
nodeOutEdges[i] << " ";

    cout << "\ngraphOutEdges : ";
    for (i = 0; i < graphOutEdges.size(); i++) cout <<
graphOutEdges[i] << " ";
```

```
}

main(int argc, char* argv[])
{
    int v = 9;    //number of vertices in graph +1
    //create an array of lists whose size is v
    list<int> graph[v];
    insert_edge(graph, 1, 2);
    insert_edge(graph, 1, 3);
    insert_edge(graph, 1, 4);
    insert_edge(graph, 1, 5);
    insert_edge(graph, 2, 3);
    insert_edge(graph, 3, 1);
    insert_edge(graph, 3, 5);
    insert_edge(graph, 4, 5);
    insert_edge(graph, 5, 2);
    insert_edge(graph, 6, 7);
    insert_edge(graph, 7, 8);
    insert_edge(graph, 8, 6);
    insert_edge(graph, 8, 7);

    displayGraph(graph, v);

    create_arrays(v,graph);
}
```

- **ΑΠΟΤΕΛΕΣΜΑΤΑ ΕΚΤΕΛΕΣΗΣ**

```
1--->2 3 4 5
2--->3
3--->1 5
4--->5
5--->2
6--->7
7--->8
8--->6 7


nodeOutEdges : 12 0 4 5 7 8 9 10 11
graphOutEdges : 2 3 4 5 3 1 5 5 2 7 8 6 7
```