Δομές Δεδομένων και Αλγόριθμοι (Γ’ εξάμηνο)

Τμήμα Μηχανικών Πληροφορικής ΤΕ, ΤΕΙ Ηπείρου

Γκόγκος Χρήστος

27/11/2014

Άσκηση εργαστηρίου #5 (Ο αλγόριθμος συντομότερης διαδρομής του Dijkstra)

Στην εργασία αυτή υλοποιείται ο αλγόριθμος εύρεσης της συντομότερης διαδρομής του Dijkstra.

Θα χρειαστεί να δηλωθούν οι ακόλουθες βιβλιοθήκες:

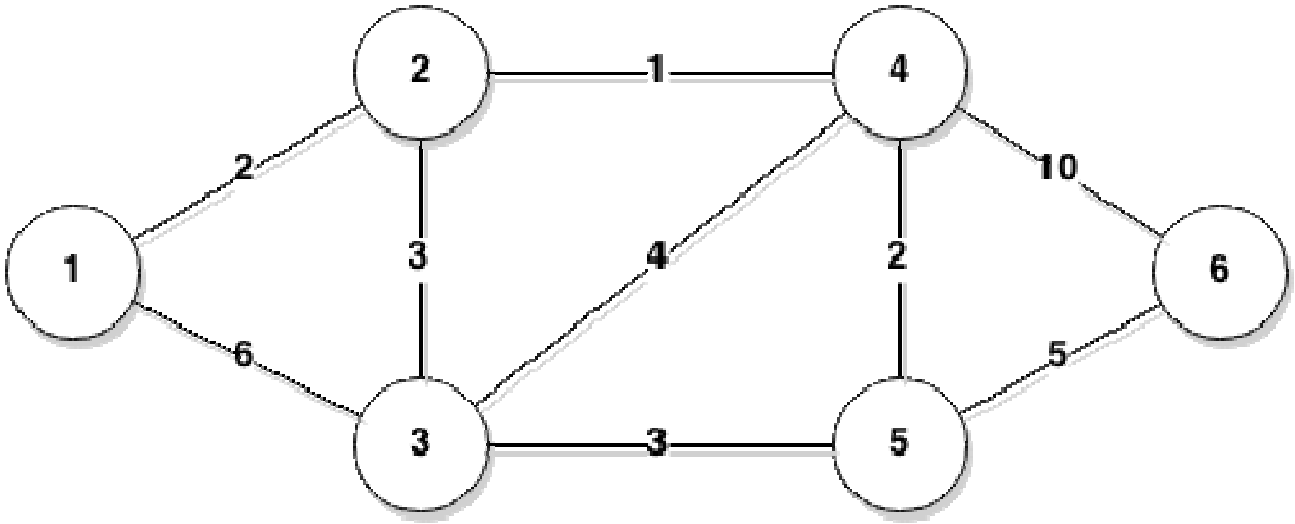
```
#include <iostream>
#include <sstream>
#include <fstream>
#include <set>
#include <vector>
#include <list>
```

Τα αρχεία που περιγράφουν το γράφημα (μη κατευθυνόμενο με βάρη) αποτελούνται από σειρές που ξεκινούν παραθέτοντας τον αριθμό της κορυφής X και εν συνεχεία μια σειρά από ζεύγη τιμών Y,Z όπου το Y είναι το βάρος της σύνδεσης ανάμεσα στην κορυφή X και στην κορυφή Z. Για παράδειγμα το ακόλουθο αρχείο

```
1 2,2 6,3
2 2,1 3,3 1,4
3 6,1 3,2 4,4 3,5
4 1,2 4,3 2,5 10,6
5 3,3 2,4 5,6
6 10,4 5,5
```

Αρχείο: TOY_GRAPH.txt

αντιστοιχεί στο παρακάτω γράφημα:



Η ανάγνωση του αρχείου και η δημιουργία μιας δομής με λίστες που περιέχει την πληροφορία του γραφήματος μπορεί να γίνει με τη συνάρτηση `read_data` που δίνεται στη συνέχεια όπου `fn` είναι το όνομα του αρχείου και `V` το πλήθος των κορυφών του γραφήματος.

```
list<pair<int, int>>* read_data(string fn, int V) {
	list<pair<int, int>>* graph = new list<pair<int, int>> [V + 1]();
	fstream filestr;
	string buffer;
	filestr.open(fn.c_str());
	if (filestr.is_open()) {
		while (getline(filestr, buffer)) {
			string buffer2;
			stringstream ss;
			ss.str(buffer);
			vector<string> tokens;
			while (ss >> buffer2) {
				tokens.push_back(buffer2);
			}
			int vertex1 = atoi(tokens[0].c_str());
			for (unsigned int i = 1; i < tokens.size(); i++) {
				int pos = tokens[i].find(",");

				int vertex2 = atoi(tokens[i].substr(0, pos).c_str());
				int weight =
						atoi(
								tokens[i].substr(pos + 1,
										tokens[i].length() -
1).c_str());
				graph[vertex1].push_back(make_pair(vertex2, weight));
			}
		}
	} else {
		cout << "Error opening file: " << fn << endl;
		exit(-1);
	}
	return graph;
}
```

Συνεπώς η κλήση της συνάρτησης θα γίνεται ως εξής:

```
int main(int argc, char **argv) {
	int V = 6;
	list<pair<int, int>>* graph = read_data("TOY_GRAPH.txt", V);

    ...

    return 0;
}
```

Ερώτημα 1

Να κατασκευάσετε συνάρτηση που να εμφανίζει όλες τις ακμές του γράφου με την μορφή X<--Y-->Z όπου X και Z είναι οι κορυφές και Y είναι το βάρος πάνω στην ακμή που τις συνδέει. Η συνάρτηση να έχει την ακόλουθη μορφή:

```
void print_graph(list<pair<int, int>>* g, int V) {
    ...
}
```

Για το πρόβλημα TOY_GRAPH που αναφέρθηκε παραπάνω οι τιμές που θα πρέπει να εμφανίζονται είναι:

```
1<--2-->2    1<--6-->3    2<--2-->1    2<--3-->3    2<--1-->4
3<--6-->1    3<--3-->2    3<--4-->4    3<--3-->5    4<--1-->2
4<--4-->3    4<--2-->5    4<--10-->6   5<--3-->3    5<--2-->4
```

```
5<--5-->6    6<--10-->4    6<--5-->5
```

Ερώτημα 2

Ο ακόλουθος κώδικας υπολογίζει τη συντομότερη διαδρομή από μια κορυφή προς όλες τις άλλες κορυφές του γραφήματος:

```
void compute_shortest_paths_to_all_vertices(list<pair<int, int>>* g, int V,
            int source, int* distance) {
      list<int> S;
      set<int> V_S;
      for (int i = 1; i <= V; i++) {
            if (i == source) {
                  S.push_back(i);
                  distance[i] = 0;
            } else {
                  V_S.insert(i);
                  distance[i] = INT_MAX;
            }
      }
      while (!V_S.empty()) {
            int min = INT_MAX;
            int pmin = -1;
            for (int v1 : S) {
                  for (pair<int, int> w_v : g[v1]) {
                        int weight = w_v.first;
                        int v2 = w_v.second;
                        bool is_in_V_S = V_S.find(v2) != V_S.end();
                        if ((is_in_V_S) && (distance[v1] + weight < min)) {
                              min = distance[v1] + weight;
                              pmin = v2;
                        }
                  }
            }
            // The graph might not be connected
            if (pmin == -1)
                  break;
            distance[pmin] = min;
            S.push_back(pmin);
            V_S.erase(pmin);
      }
}
```

και καλείται ως εξής:

```
int source=1;
int* shortest_path_distances = new int[V + 1];
compute_shortest_paths_to_all_vertices(graph, V, source, shortest_path_distances);
```

προκειμένου να τοποθετήσει στον πίνακα `shortest_path_distances` τα μήκη των συντομότερων διαδρομών από την κορυφή `source=1` προς όλες τις άλλες κορυφές. Τα δε αποτελέσματα μπορούν να εμφανιστούν με τον ακόλουθο κώδικα

```
for (int destination_vertex = 1; destination_vertex <= V;
            destination_vertex++) {
      if (shortest_path_distances[destination_vertex] == INT_MAX)
            printf("Οι κορυφές %d και %d δεν είναι συνδεδεμένες\n", 1,
                  destination_vertex);
      else
            printf("Η συντομότερη διαδρομή από την κορυφή %d στην κορυφή %d έχει μήκος
%d\n", 1,
```

```
                        destination_vertex,
                        shortest_path_distances[destination_vertex]);
}
```

και είναι για το γράφημα TOY_GRAPH:

```
Η συντομότερη διαδρομή από την κορυφή 1 στην κορυφή 1 έχει μήκος 0
Η συντομότερη διαδρομή από την κορυφή 1 στην κορυφή 2 έχει μήκος 2
Η συντομότερη διαδρομή από την κορυφή 1 στην κορυφή 3 έχει μήκος 5
Η συντομότερη διαδρομή από την κορυφή 1 στην κορυφή 4 έχει μήκος 3
Η συντομότερη διαδρομή από την κορυφή 1 στην κορυφή 5 έχει μήκος 5
Η συντομότερη διαδρομή από την κορυφή 1 στην κορυφή 6 έχει μήκος 10
```

Να καλέσετε τη συνάρτηση `compute_shortest_paths_to_all_vertices` επαναλήπτικά έτσι ώστε να εμφανίσετε τα μήκη των συντομότερων διαδρομών από κάθε κορυφή προς κάθε άλλη κορυφή του γραφήματος.

Ερώτημα 3

Χρησιμοποιήστε τον ίδιο κώδικα έτσι ώστε να φορτώσετε το αρχείο CAR91_GRAPH.txt που θα βρείτε στο https://github.com/chgogos/ett_ce_teiep_2014_2015 και εμφανίστε τη μεγαλύτερη συντομότερη διαδρομή μεταξύ 2 οποιονδήποτε κορυφών του γραφήματος (το γράφημα αυτό έχει 682 κορυφές).