

Data Structures and Algorithms

Exercise 10

Markus Innerebner, Romans Kasperovics
dsa(a)inf.unibz.it

Free University of Bozen - Bolzano, Italy

Thursday, April 30th, 2009

Assignment 10

Task1

Write a solution that tests if the two vertexes v, u are connected unreliably. The graph traversal must be based on Breadth-First-Search (**BFS**). Use as data structure for the connectivity of vertexes an *adjacency matrix*. Use as test data the undirected graph structure illustrated in figure 1.

- 1a) Implement a function
`bool isUnreliablyConnected(struct Vertex* u, struct Vertex* v);`
- 1b) Print all critical vertexes.
- 1c) What is the complexity of your implementation?

Assignment 10

Wireless Network

A wireless network can be modelled as a graph where the vertexes are computing devices located in physical space. The graph contains an edge (u, v) if device v is close enough to device u to receive and to transmit signals. Two devices are connected unreliably, if the connection can be destroyed by removing just one device from the network. Such a device is called *critical vertex*.

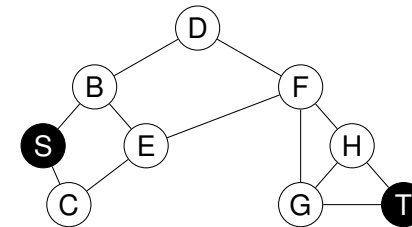


Figure: Graph representing a wireless network.

Assignment 10

Task1

Write a solution that tests if the two vertexes v, u are connected unreliably. The graph traversal must be based on Depth-First-Search (**DFS**). Use as data structure for the connectivity of vertexes an *adjacency list*. The complexity of your implementation must be less than $O(V * (V + E))$. Use as test data the undirected graph structure illustrated in figure 1.

- 2a) Implement a function
`bool isUnreliablyConnected(struct Vertex* u, struct Vertex* v);`
- 2b) Print all critical vertexes.
- 2c) What is the complexity of your implementation?

Hints

Vertex

Defining the colors:

```
typedef enum {WHITE, GRAY, BLACK} Color;
```

Structure of the vertex:

```
struct Vertex {
    char label; // label
    Color clr; // color
    struct Vertex* adj[SIZE_E]; // adjacency list
    struct Vertex* pred; // predecessor
    bool critical; // flag if critical vertex
    int dTime; // discovery time
};
```

Hints

Tasks 1,2 - Queue

Signature of queue, if queue is a global variable.

```
bool isEmpty();
void enqueue(Vertex* v);
Vertex* dequeue();
```

Hints

Tasks 1 - Adjacency matrix

```
//
int Adj[N][N] = {
    {0, 1, 1, 0, 0, 0, 0, 0, 0}, // S
    {1, 0, 0, 1, 1, 0, 0, 0, 0}, // B
    {1, 0, 0, 0, 1, 0, 0, 0, 0}, // C
    {0, 1, 0, 0, 0, 1, 0, 0, 0}, // D
    {0, 1, 1, 0, 0, 1, 0, 0, 0}, // E
    {0, 0, 0, 1, 1, 0, 1, 1, 0}, // F
    {0, 0, 0, 0, 0, 1, 0, 1, 1}, // G
    {0, 0, 0, 0, 0, 1, 1, 0, 1}, // H
    {0, 0, 0, 0, 0, 0, 1, 1, 0}, // T
};
```

Hints

Tasks 1 - Adjacency list

Use as adjacency list for each vertex an array of pointers (pointing to a vertex) with the size equals to the total number of vertexes