

Data Structures and Algorithms

Exercise 10

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

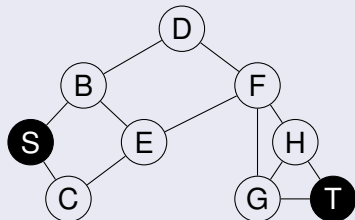
Free University of Bozen - Bolzano, Italy

Thursday, May 7th, 2009

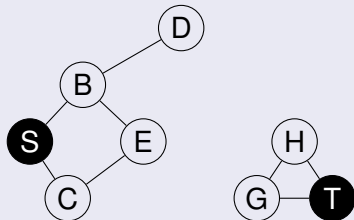
Solutions 10

Communication Networks

Original wireless network



Deleting node F

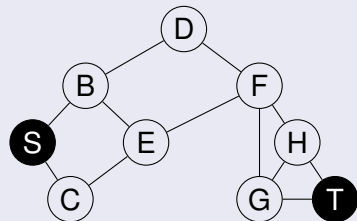


Solutions 10

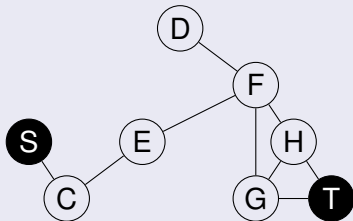
Task 1 - BFS-based solution

Idea: We remove a vertex (e.g. B) from the graph and test vertex T is still reachable from vertex S .

Original wireless network



Deleting node B



We repeat this step $|V|$ times.

Solutions 10

Task 1

```
#include <stdio.h>
#include <stdlib.h>
#define MAX 9
#define SIZE_V 9
typedef enum {FALSE, TRUE} bool;
typedef enum {WHITE, GRAY, BLACK} color;

typedef struct vertex {
    char label; color clr;
    struct vertex* pred; bool critical; int pos;
} Vertex;

Vertex* G[SIZE_V]; Vertex* Q[MAX];
int head = 0, tail = 0;
```

Solutions 10

Task 1 - Adjacency matrix - initialization

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

void init(){
    char labels[] = {'B', 'C', 'D', 'E', 'F', 'G', 'H', 'S', 'T'}; int i;
    for (i=0; i< SIZE_V; i++) {
        Vertex* v = malloc(sizeof(Vertex)); v->clr = WHITE; v->critical = FALSE;
        v->label = labels[i]; v->pos = i; G[i] = v;
    }
}
```

Solutions 10

Task 1 - Queue functions

```
void Enqueue(Vertex* v) {  
    Q[tail] = v;  
    tail = (tail==MAX-1) ? 0 : tail+1;  
}  
  
Vertex* Dequeue() {  
    Vertex* x = Q[head]; Q[head]=NULL;  
    head = (head == MAX-1) ? 0 : head + 1;  
    return x;  
}  
  
bool isEmpty() {  
    return head==tail;  
}
```

Solutions 10

Task 1 - Unreliability check

```
bool isUnreliablyConnected(Vertex* s, Vertex* t, Vertex* exclude) {
    if(s->label==exclude->label || t->label==exclude->label) return FALSE;
    int i;
    for (i = 0; i < SIZE_V; i++) {
        G[i]->clr = WHITE; G[i]->pred = NULL;
    }
    head = tail = 0;
    s->clr = GRAY; s->pred = NULL; exclude->clr = GRAY;
    Enqueue(s);
    while (!isEmpty()) {
        Vertex* u = Dequeue();
        if (u->label==t->label) return FALSE;
        for (i = 0; i < SIZE_V; i++) {
            if (G[i]->clr == WHITE && Adj[u->pos][i] > 0 ) {
                G[i]->clr = GRAY; G[i]->pred = u; Enqueue(G[i]);
            }
        }
        u->clr = BLACK;
    }
    return TRUE;
}
```

Solutions 10

Task 1 - Main

```
int main() {
    init(); int i; bool unreliable = FALSE;
    for(i = 0; i < SIZE_V; i++) { // test reliability of nodes S and T
        if(isUnreliablyConnected(G[7],G[8],G[i])){
            printf("Node %c is critical!\n",((Vertex*)G[i])->label);//1.b
            unreliable = TRUE; break;
        }
    }
    char* s = (unreliable) ? "" : " NOT";
    printf("Vertex %c and %c are%s unreliably connected.\n", G[7]->label, G[8]->label, s);
    unreliable = FALSE;
    for(i = 0; i < SIZE_V; i++) { // test reliability of nodes S and F
        if(isUnreliablyConnected((Vertex*)G[7],(Vertex*)G[4],(Vertex*)G[i])){
            printf("Node %c is critical!\n",G[i]->label); //1.b
            unreliable = TRUE; break;
        }
    }
    s = (unreliable) ? "" : " NOT";
    printf("Vertex %c and %c are%s unreliably connected.\n", G[7]->label, G[4]->label, s);
    return 0;
}
```

```
$ ./task1
```

```
Node F is critical!
```

```
Vertex S and T are unreliably connected.
```

```
Vertex S and F are NOT unreliably connected.
```

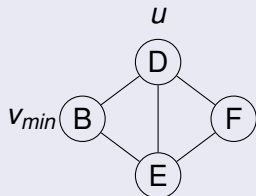
Complexity: $O(|V| * (|V| + |E|))$

Solutions 10

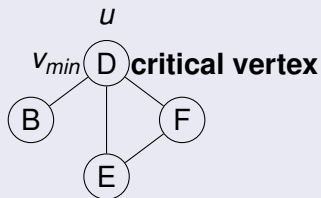
Task 2 - DFS-based solution

Idea: For any vertex u we determine from any vertex of u 's subtrees the vertex v_{min} with the smallest discovery time. If one of this vertices points to a vertex with a smaller discovery time than u , we obtain a cycle and the connection is reliable.

Reliability between B and E



Unreliability between B and E



Solutions 10

Task 2 - structure

```
#include <stdio.h>
#include <stdlib.h>
typedef enum {BLACK, GRAY, WHITE} color;
typedef enum {FALSE, TRUE} bool;

#define SIZE_V 9

typedef struct vertex {
    char label;           // label of the vertex
    color clr;           // color of the vertex
    struct vertex* adj[SIZE_V]; // we use an adjacency list
    struct vertex* pred; // predecessor
    bool critical;       // flag if critical point or not
    int dTime;           // discovery time
} Vertex;

Vertex* G[SIZE_V];      // graph
int time = 0;
```

Solutions 10

Task 2 - Initialization

```
void init(){
    time = 0; int i,j; char labels[] = {'B', 'C', 'D', 'E', 'F', 'G', 'H', 'S', 'T'};
    for (i=0; i< SIZE_V; i++) {
        Vertex* v = malloc(sizeof(Vertex));
        v->clr = WHITE; v->critical = FALSE; v->label = labels[i];
        v->dTime = -1; v->pred = NULL;
        for(j=0; j< SIZE_V; j++) v->adj[j] = NULL;
        G[i] = v;
    }
    G[0]->adj[0]=G[2]; G[0]->adj[1]=G[3]; G[0]->adj[2]=G[7]; // |B|->| D,E,S |
    G[1]->adj[0]=G[0]; G[1]->adj[1]=G[3]; G[1]->adj[2]=G[7]; // |C|->| B,E,S |
    G[2]->adj[0]=G[0]; G[2]->adj[1]=G[4]; // |D|->| B,F |
    G[3]->adj[0]=G[0]; G[3]->adj[1]=G[1]; G[3]->adj[2]=G[4]; // |E|->| B,C,F |
    G[4]->adj[0]=G[2]; G[4]->adj[1]=G[3]; G[4]->adj[2]=G[5]; G[4]->adj[3]=G[6];
    // |F|->| D,E,G,H |
    G[5]->adj[0]=G[4]; G[5]->adj[1]=G[6]; G[5]->adj[2]=G[8]; // |G|->| F,H,T |
    G[6]->adj[0]=G[4]; G[6]->adj[1]=G[5]; G[6]->adj[2]=G[8]; // |H|->| F,G,T |
    G[7]->adj[0]=G[2]; G[7]->adj[1]=G[5]; G[7]->adj[2]=G[8]; // |S|->| B,C |
    G[8]->adj[0]=G[5]; G[8]->adj[1]=G[6]; // |T|->| G,H |
}
```

Solutions 10

Task 2 - Visiting vertex

```
int visit(Vertex* u) {
    u->clr = GRAY; u->dTime = ++time;
    int min = time; int i;
    for (i = 0; i < SIZE_V; i++) {
        Vertex* v = u->adj[i];
        if(v != NULL){
            if(v->clr == WHITE){
                v->pred=u; int m = visit(v);
                if(m < min) min=m;
                if(m >= u->dTime) u->critical=TRUE;
            } else {
                if(v->dTime < min) min= v->dTime;
            }
        }
    }
    return min;
}
```

Solutions 10

Task 2 - Unreliability check

```
bool isUnreliablyConnected(Vertex* s, Vertex* t) {
    if(s==t) return TRUE;
    visit(s);
    if (t->clr == WHITE) {
        printf ("%c and %c are not connected\n", s->label, t->label);
        return FALSE;
    }
    Vertex* v = t;
    while(v != s){
        if (v->critical) {
            printf("Node %c is critical!\n",v->label);
            return TRUE;
        }
        v=v->pred;
    }
    return FALSE;
}
```

```
int main() {
    init(); char* s=""; // test reliability of nodes S and T
    if(!isUnreliablyConnected(G[7],G[8]))
        s=" NOT";
    printf("Vertex %c and %c are%s unreliably connected.\n",
        G[7]->label, G[8]->label, s);
    s=""; // test reliability of nodes S and C
    if(!isUnreliablyConnected(G[7],G[1]))
        s=" NOT";
    printf("Vertex %c and %c are%s unreliably connected.\n",
        G[7]->label, G[1]->label,s);
    return 0;
}
```

```
$ ./task2
```

```
Node F is critical!
```

```
Vertex S and T are unreliably connected.
```

```
Vertex S and C are NOT unreliably connected.
```

Complexity: $O(|V| + |E|)$