Belgian Olympiad in Informatics

Weekend Training 28-29 March 2015 — Day 2 and 3

Quiz solutions

Trainer:
Floris Kint

April 15, 2015

Contents

1 Datastructures

1.1 Students in a classroom

```
1 #include <iostream>
2 #include <set>
3 #include <string>
```

```
4
5  using namespace std;
6
7  struct classroom{
8    set<string> students;
9    void enter(string student){
10     if(in_classroom(student))
11       cout << "Student is already in classroom" << endl;
12     students.insert(student);
13   }
14   void leave(string student){
15     if(!in_classroom(student))
16       cout << "Student is not in classroom" << endl;
17     students.erase(student);
18   }
19   bool in_classroom(string student){
20     return students.count(student) > 0;
21   }
22 };
23
24 int main(){
25   std::ios::sync_with_stdio(false);
26   int N;
27   cin >> N;
28   classroom room;
29   for(int i = 0; i < N; ++i){
30     string name;
31     string action;
32     cin >> name >> action;
33     if(action == "enter"){
34       room.enter(name);
35     }else if(action == "leave"){
36       room.leave(name);
37     }else{
38       cout << "Invalid action, choose enter/leave" <<
             endl;
39     }
40   }
41   return 0;
42 }
```

1.2 Children and cookies

```
1  #include <string>
2  #include <iostream>
3  #include <map>
4
5  using namespace std;
```

```
6
7  struct cookies{
8    map<string, int> pocket;
9    void receive_cookies(string child, int amount){
10     if(amount <= 0)
11       cout << "Why are you so cruel as to give a non-
             positive number of cookies to this child?" <<
             endl;
12     this->pocket[child] += amount;
13   }
14   void eat_cookies(string child, int amount){
15     if(amount <= 0)
16       cout << "This is just disgusting..." << endl;
17     if(amount > get_cookies(child))
18       cout << "The child can only eat " << get_cookies(
             child)<< " cookies" << endl;
19     this->pocket[child] -= amount;
20   }
21   int get_cookies(string child){
22     return this->pocket[child];
23   }
24 };
25
26 int main(){
27   std::ios::sync_with_stdio(false);
28   int N;
29   cin >> N;
30   cookies cc;
31   for(int i = 0; i < N; ++i){
32     string name, action;
33     int amount;
34     cin >> name >> action >> amount;
35     if(action == "receive"){
36       cc.receive_cookies(name, amount);
37     }else if(action == "eat"){
38       cc.eat_cookies(name, amount);
39     }else{
40       cout << "Invalid action, choose either receive/eat"
              << endl;
41     }
42     cout << name << " has now " << cc.get_cookies(name)
         << endl;
43   }
44   return 0;
45 }
```

1.3 Containers

```cpp
#include <cstdio>
#include <algorithm>
#include <vector>

using namespace std;

struct container{
  int index;
  int l, h, d;
  container(int index, int l, int h, int d){
    this->index = index;
    this->l = l;
    this->h = h;
    this->d = d;
  }
  int volume() const{
    return this->l * this->h * this->d;
  }
  bool operator<(const container &c) const{
    return this->volume() < c.volume();
  }
};

int main(){
  int N;
  scanf("%d", &N);
  vector<container> containers;
  containers.reserve(N);
  for(int i = 0; i < N; ++i){
    int l, h, d;
    scanf("%d%d%d", &l, &h, &d);
    containers.push_back(container(i, l, h, d));
  }
  sort(containers.begin(), containers.end());
  for(vector<container>::iterator it = containers.begin()
      ; it < containers.end(); ++it){
    printf("%d: %d\n", it->index, it->volume());
  }
  return 0;
}
```

2 Graphs

2.1 What is a tree?

An acyclic graph. It is often used to represent a hierarchical structure.

2.2 How many edges are there in a tree?

Let n be the number of nodes in the tree. Then there are $n-1$ edges. This can easily be seen when the tree is drawn with a root node on top, and the children below their parents. Every node except for the root has exactly one edge to their parent, so there are $n-1$ edges.

2.3 Sparse graph with string identifiers

```
map<string, map<string, int> >; // weighted graph
map<string, set<string> >; // unweighted graph
```

2.4 Dense graph

```
// weight[i][j] is -1 if no edge from i to j exists
// if an edge from i to j exists, weight[i][j] is the weight of that edge
int weight[MAXN][MAXN];
```

2.5 What is a BST

A Binary Search Tree. It is a tree where every node has at most 2 children (binary tree). All nodes at the left of a node u are smaller than u. All nodes at the right of u are larger than u. Note that the term BST does NOT imply that the tree is balanced.

2.6 Find the distance between two elements in a BST

```
1  #include <cstdio>
2
3  #define LEFT 0
4  #define ME 1
5  #define RIGHT 2
6
7  struct node{
8    int value;
9    node* left;
10   node* right;
11   node(){
12     this->value = -1;
13     this->left = NULL;
14     this->right = NULL;
15   }
16   void add_value(int value){
17     if(this->value == -1){
18       this->value = value;
19       this->left = new node();
20       this->right = new node();
21     }else if(this->value < value){
22       this->right->add_value(value);
```

```
23      }else if(this->value > value){
24        this->left->add_value(value);
25      }
26    }
27    int side(int v){
28      if(this->value == v)
29        return ME;
30      if(this->value < v)
31        return RIGHT;
32      return LEFT;
33    }
34    node* node_at_side(int side){
35      switch(side){
36        case LEFT:
37          return left;
38        case RIGHT:
39          return right;
40        default:
41          return this;
42      }
43    }
44    int distance_to(int v){
45      if(v == this->value)
46        return 0;
47      if(v < this->value)
48        return 1+this->left->distance_to(v);
49      else
50        return 1+this->right->distance_to(v);
51    }
52  };
53
54  int distance_between(node *root, int a, int b){
55    node *current = root;
56    while(current->side(a) == current->side(b)){
57      current = current->node_at_side(current->side(a));
58    }
59    return current->distance_to(a) + current->distance_to(b
        );
60  }
61
62  int main(){
63    int N;
64    scanf("%d", &N);
65    node* root =  new node();
66    for(int i = 0; i < N; ++i){
67      int t;
68      scanf("%d", &t);
69      root->add_value(t);
70    }
71    int M;
```

```
72    scanf("%d", &M);
73    for(int i = 0; i < M; ++i){
74      int a, b;
75      scanf("%d%d", &a, &b);
76      printf("Distance from a to b is: %d\n",
            distance_between(root, a, b));
77    }
78    return 0;
79  }
```

2.7 What does Dijkstra's algorithm compute

Dijkstra's algorithm computes the shortest path length from a single source to all nodes in the graph.

2.8 What does an iteration of Dijkstra's algorithm consist of? What is the invariant?

In every iteration, Dijkstra adds the closest node that has not been visited yet to the tree consisting of visited nodes. Next, it adds all neighbours of the newly added node to a priority queue.

The invariant: we know the shortest path length from the source node to all nodes that have been visited.

2.9 Implement Dijkstra's algorithm

```
1  #include <cstdio>
2  #include <queue>
3  #include <list>
4  #include <vector>
5
6
7  using namespace std;
8
9  struct node{
10   int index;
11   int distance;
12   bool operator<(const node &n) const{
13     return this->distance > n.distance;
14   }
15   node(int index, int distance){
16     this->index = index;
17     this->distance = distance;
18   }
19 };
20
```

```
21 int dijkstra(vector<list<pair<int, int> > > &neighbours,
       int from, int to){
22   priority_queue<node> pq;
23   pq.push(node(from, 0));
24   vector<bool> visited(neighbours.size(), false);
25   while(!pq.empty()){
26     node n = pq.top();
27     pq.pop();
28     if(to == n.index)
29       return n.distance;
30     if(visited[n.index])
31       continue;
32     visited[n.index] = true;
33     for(list<pair<int, int> >::iterator it = neighbours[n
           .index].begin(); it != neighbours[n.index].end();
           ++it){
34       if(visited[it->first])
35         continue;
36       pq.push(node(it->first, it->second + n.distance));
37     }
38   }
39   return -1;
40 }
41
42 int main(){
43   int N;
44   scanf("%d", &N);
45   vector<list<pair<int, int> > > neighbours(N);
46   int M;
47   scanf("%d", &M);
48   for(int i = 0; i < M; ++i){
49     int a, b, d;
50     scanf("%d%d%d", &a, &b, &d);
51     neighbours[a].push_back(make_pair(b, d));
52   }
53   int start, end;
54   scanf("%d%d", &start, &end);
55   printf("%d\n", dijkstra(neighbours, start, end));
56   return 0;
57 }
```

2.10 Modify Dijkstra's algorithm to return the path

```
1 #include <cstdio>
2 #include <stack>
3 #include <queue>
4 #include <list>
5 #include <vector>
```

```
6
7
8  using namespace std;
9
10 struct node{
11   int index;
12   int distance;
13   int from;
14   bool operator<(const node &n) const{
15     return this->distance > n.distance;
16   }
17   node(int index, int distance, int from){
18     this->index = index;
19     this->distance = distance;
20     this->from = from;
21   }
22 };
23
24 pair<int, vector<int> > dijkstra(vector<list<pair<int,
       int> > > &neighbours, int from, int to){
25   priority_queue<node> pq;
26   pq.push(node(from, 0, -1));
27   vector<bool> visited(neighbours.size(), false);
28   vector<int> prev(neighbours.size(), -1);
29   while(!pq.empty()){
30     node n = pq.top();
31     pq.pop();
32     if(visited[n.index])
33       continue;
34     visited[n.index] = true;
35     prev[n.index] = n.from;
36     if(to == n.index){
37       stack<int> p;
38       int curr = n.index;
39       while(curr != -1){
40         p.push(curr);
41         curr = prev[curr];
42       }
43       vector<int> path;
44       while(!p.empty()){
45         path.push_back(p.top());
46         p.pop();
47       }
48       return make_pair(n.distance, path);
49     }
50     for(list<pair<int, int> >::iterator it = neighbours[n
           .index].begin(); it != neighbours[n.index].end();
          ++it){
51       if(visited[it->first])
52         continue;
```

```
53          pq.push(node(it->first, it->second + n.distance, n.
                index));
54        }
55      }
56      return make_pair(-1, vector<int>());
57  }
58
59  int main(){
60      int N;
61      scanf("%d", &N);
62      vector<list<pair<int, int> > > neighbours(N);
63      int M;
64      scanf("%d", &M);
65      for(int i = 0; i < M; ++i){
66          int a, b, d;
67          scanf("%d%d%d", &a, &b, &d);
68          neighbours[a].push_back(make_pair(b, d));
69      }
70      int start, end;
71      scanf("%d%d", &start, &end);
72      pair<int, vector<int> > result = dijkstra(neighbours,
            start, end);
73      printf("%d\n", result.first);
74      for(vector<int>::iterator it = result.second.begin();
            it != result.second.end(); ++it){
75          printf("%d ", *it);
76      }
77      printf("\n");
78      return 0;
79  }
```

2.11 DFS

```
1   #include <cstdio>
2   #include <list>
3   #include <vector>
4   #include <stack>
5
6   using namespace std;
7
8   bool dfs(vector<list<int> > &neighbours, int target){
9       stack<int> st;
10      st.push(0);
11      vector<bool> open(neighbours.size(), false);
12      open[0] = true;
13      while(!st.empty()){
14          int curr = st.top();
15          if(curr == target)
```

```
16        return true;
17      st.pop();
18      for(list<int>::iterator it = neighbours[curr].begin()
            ; it != neighbours[curr].end(); ++it){
19        if(open[*it])
20          continue;
21        st.push(*it);
22        open[*it] = true;
23      }
24    }
25    return false;
26  }
27
28  int main(){
29    int N;
30    scanf("%d", &N);
31    vector<list<int> > neighbours(N);
32    int M;
33    scanf("%d", &M);
34    for(int i = 0; i < M; ++i){
35      int a, b;
36      scanf("%d%d", &a, &b);
37      neighbours[a].push_back(b);
38    }
39    int to_find;
40    scanf("%d", &to_find);
41    printf("%d\n", dfs(neighbours, to_find)?1:0);
42    return 0;
43  }
```

2.12 BFS

```
1   #include <cstdio>
2   #include <list>
3   #include <vector>
4   #include <queue>
5
6   using namespace std;
7
8   bool dfs(vector<list<int> > &neighbours, int target){
9     queue<int> st;
10    st.push(0);
11    vector<bool> open(neighbours.size(), false);
12    open[0] = true;
13    while(!st.empty()){
14      int curr = st.front();
15      if(curr == target)
16        return true;
```

```
17      st.pop();
18      for(list<int>::iterator it = neighbours[curr].begin()
            ; it != neighbours[curr].end(); ++it){
19        if(open[*it])
20          continue;
21        st.push(*it);
22        open[*it] = true;
23      }
24    }
25    return false;
26  }
27
28  int main(){
29    int N;
30    scanf("%d", &N);
31    vector<list<int> > neighbours(N);
32    int M;
33    scanf("%d", &M);
34    for(int i = 0; i < M; ++i){
35      int a, b;
36      scanf("%d%d", &a, &b);
37      neighbours[a].push_back(b);
38    }
39    int to_find;
40    scanf("%d", &to_find);
41    printf("%d\n", dfs(neighbours, to_find)?1:0);
42    return 0;
43  }
```

2.13 When does an Euler path exist?

When the graph is connected and either two or zero nodes have an odd degree.

2.14 When does an Euler cycle exist?

When the graph is connected and all nodes have an even degree.

2.15 Algorithm to find Euler path

2.16 What should be added to a DFS to implement Toposort?

Taken from the course notes of the second weekend training.

Listing 1: DFS algorithm

```
1  int UNVISITED = 0, OPEN = 1, CLOSED = 2;
2
```

```
3  void dfsVisit(LinkedList<Integer>[] adj_list, int node,
       int[] labels)
4  {
5    labels[node] = OPEN;
6    for(int new_node : adj_list[node])
7      if(labels[new_node] == UNVISITED)
8        dfsVisit(adj_list, new_node, labels);
9    labels[node] = CLOSED;
10 }
11
12 void dfs(LinkedList<Integer>[] adj_list)
13 {
14   int[] labels = new int[adj_list.length];
15   Arrays.fill(labels, UNVISITED);
16   for(int node = 0; node < adj_list.length; node++)
17     if(labels[node] == UNVISITED)
18       dfsVisit(adj_list, node, labels);
19 }
```

Listing 2: Toposort algorithm

```
1  int UNVISITED = 0, OPEN = 1, CLOSED = 2;
2
3  void dfsVisit(LinkedList<Integer>[] adj_list, int node,
       int[] labels, Stack<Integer> stack)
4  {
5    labels[node] = OPEN;
6    for(int new_node : adj_list[node])
7      if(labels[new_node] == UNVISITED)
8        dfsVisit(adj_list, new_node, labels, stack);
9    labels[node] = CLOSED;
10     stack.push(node);
11 }
12
13 Stack<Integer> toposort(LinkedList<Integer>[] adj_list)
14 {
15   int[] labels = new int[adj_list.length];
16     Stack<Integer> stack = new Stack<Integer>();
17
18   Arrays.fill(labels, UNVISITED);
19   for(int node = 0; node < adj_list.length; node++)
20     if(labels[node] == UNVISITED)
21       dfsVisit(adj_list, node, labels, stack);
22     return stack;
23 }
```

2.17 List all rooms in the cave

```
#include <cstdio>

#define MAXR 1000
#define MAXC 1000

int directions[4][2] = {{1,0},{-1,0},{0,1},{0,-1}};
int R, C;
char grid[MAXR][MAXC];
bool visited[MAXR][MAXC];

bool valid_coordinate(int r, int c){
  if(!(r >= 0 && r < R && c >= 0 && c < C))
    return false;

  return grid[r][c] == '.';
}

void explore_room(int r, int c){
  if(!valid_coordinate(r, c))
    return;
  if(visited[r][c])
    return;
  visited[r][c] = true;
  printf(" %d %d\n", r, c);
  for(int i = 0; i < 4; ++i){
    int nr = r + directions[i][0];
    int nc = c + directions[i][1];
    explore_room(nr, nc);
  }
}

int main(){
  scanf("%d%d", &R, &C);
  for(int i = 0; i < R; ++i){
    while(fgetc(stdin) != '\n'){}
    for(int j = 0; j < C; ++j){
      grid[i][j] = fgetc(stdin);
    }
  }
  for(int i = 0; i < R; ++i){
    for(int j = 0; j < C; ++j){
      if(!valid_coordinate(i, j))
        continue;
      if(!visited[i][j]){
        printf("Found a new room: \n");
        explore_room(i, j);
      }
    }
  }
```

```
50    return 0;
51  }
```

2.18 Prim's algorithm with string identifiers

```
1   #include <iostream>
2   #include <set>
3   #include <string>
4   #include <map>
5   #include <list>
6   #include <queue>
7
8   using namespace std;
9
10  struct node{
11    string identifier;
12    int distance;
13    string coming_from;
14    node(string id, int d, string coming_from){
15      this->identifier = id;
16      this->distance = d;
17      this->coming_from = coming_from;
18    }
19    bool operator<(const node &n)const{
20      return this->distance > n.distance;
21    }
22  };
23
24  void prim(map<string, list<pair<string, int> > > &
        neighbours, set<string> cities){
25    priority_queue<node> pq;
26    pq.push(node(*cities.begin(), 0, ""));
27    set<string> visited;
28    int total_cost = 0;
29    while(!pq.empty()){
30      node n = pq.top();
31      pq.pop();
32      if(visited.count(n.identifier))
33        continue;
34      if(n.coming_from != ""){
35        cout << n.identifier << " to " << n.coming_from <<
              endl;
36        total_cost += n.distance;
37      }
38      visited.insert(n.identifier);
39      for(list<pair<string, int> >::iterator it =
            neighbours[n.identifier].begin(); it != neighbours
```

```
          [n.identifier].end(); ++it){
40        if(visited.count(it->first))
41          continue;
42        pq.push(node(it->first, it->second, n.identifier));
43      }
44    }
45    cout << "Total cost: " << total_cost << endl;
46  }
47
48  int main(){
49    std::ios::sync_with_stdio(false);
50    int N;
51    cin >> N;
52    set<string> cities;
53    for(int i = 0; i < N; ++i){
54      string name;
55      cin >> name;
56      cities.insert(name);
57    }
58    int M;
59    cin >> M;
60    map<string, list<pair<string, int>>> neighbours;
61    for(int i = 0; i < M; ++i){
62      string name1, name2;
63      int distance;
64      cin >> name1 >> name2 >> distance;
65      neighbours[name1].push_back(make_pair(name2, distance
          ));
66      neighbours[name2].push_back(make_pair(name1, distance
          ));
67    }
68    prim(neighbours, cities);
69    return 0;
70  }
```

2.19 Kruskal's algorithm with string identifiers

```
1   #include <iostream>
2   #include <string>
3   #include <vector>
4   #include <algorithm>
5   #include <map>
6   #include <set>
7
8   using namespace std;
9   struct union_find{
10    map<string, string> parent;
```

```
11    map<string, int> rank;
12    union_find(set<string> &cities){
13      for(set<string>::iterator it = cities.begin(); it !=
          cities.end(); ++it){
14        rank[*it] = 1;
15        parent[*it] = *it;
16      }
17    }
18    void merge(string a, string b){
19      string parent_a = find_parent(a);
20      string parent_b = find_parent(b);
21      if(rank[parent_a] > rank[parent_b]){
22        parent[parent_b] = parent_a;
23      }else{
24        parent[parent_a] = parent_b;
25        if(rank[parent_a] == rank[parent_b]){
26          rank[parent_b]++;
27        }
28      }
29    }
30    bool is_same_tree(string a, string b){
31      return find_parent(a) == find_parent(b);
32    }
33    string find_parent(string id){
34      if(parent[id] != id)
35        parent[id] = find_parent(parent[id]);
36      return parent[id];
37    }
38  };
39  struct edge{
40    string city1, city2;
41    int distance;
42    edge(string city1, string city2, int distance){
43      this->city1 = city1;
44      this->city2 = city2;
45      this->distance = distance;
46    }
47    bool operator<(const edge &e)const{
48      return this->distance < e.distance;
49    }
50  };
51  void kruskal(vector<edge> &edges, set<string> &cities){
52    sort(edges.begin(), edges.end());
53    union_find uf(cities);
54    int total_cost = 0;
55    for(vector<edge>::iterator it = edges.begin(); it !=
        edges.end(); ++it){
56      if(uf.is_same_tree(it->city1, it->city2))
57        continue;
58      uf.merge(it->city1, it->city2);
```

```
59      total_cost += it->distance;
60      cout << it->city1 << " " << it->city2 << endl;
61    }
62    cout << "Total cost: " << total_cost << endl;
63  }
64
65  int main(){
66    std::ios::sync_with_stdio(false);
67    int N;
68    cin >> N;
69    set<string> cities;
70    for(int i = 0; i < N; ++i){
71      string name;
72      cin >> name;
73      cities.insert(name);
74    }
75    int M;
76    cin >> M;
77    vector<edge> edges;
78    for(int i = 0; i < M; ++i){
79      string city1, city2;
80      int distance;
81      cin >> city1 >> city2 >> distance;
82      edges.push_back(edge(city1, city2, distance));
83    }
84    kruskal(edges, cities);
85    return 0;
86  }
```

2.20 UnionFind with string identifiers

Same as question above

```
1   #include <iostream>
2   #include <map>
3   #include <string>
4   #include <set>
5
6   using namespace std;
7   struct union_find{
8     map<string, string> parent;
9     map<string, int> rank;
10    union_find(set<string> &cities){
11      for(set<string>::iterator it = cities.begin(); it !=
            cities.end(); ++it){
12        rank[*it] = 1;
13        parent[*it] = *it;
14      }
```

```
15    }
16    void merge(string a, string b){
17      string parent_a = find_parent(a);
18      string parent_b = find_parent(b);
19      if(rank[parent_a] > rank[parent_b]){
20        parent[parent_b] = parent_a;
21      }else{
22        parent[parent_a] = parent_b;
23        if(rank[parent_a] == rank[parent_b]){
24          rank[parent_b]++;
25        }
26      }
27    }
28    bool is_same_tree(string a, string b){
29      return find_parent(a) == find_parent(b);
30    }
31    string find_parent(string id){
32      if(parent[id] != id)
33        parent[id] = find_parent(parent[id]);
34      return parent[id];
35    }
36  };
```

3 Other

3.1 Recursive relation steps on a line with steps of size 1/2/3

$$
\begin{aligned}
ways_0 &= 1 \\
ways_1 &= 1 \\
ways_2 &= 2 \\
ways_n &= ways_{n-1} + ways_{n-2} + ways_{n-3} \quad \forall n > 3
\end{aligned}
$$

3.2 Implementation of steps on al ine with size 1/2/3

```
1  #include <cstdio>
2
3  int get_ways(int N){
4    switch(N){
5      case 0:
6        return 1;
7      case 1:
8        return 1;
9      case 2:
```

```
10        return 2;
11      default:
12        int curr = 4;
13        int prev = 2;
14        int prev2 = 1;
15        for(int i = 3; i < N; ++i){
16          int tmp = prev2;
17          prev2 = prev;
18          prev = curr;
19          curr = tmp + prev + prev2;
20        }
21        return curr;
22    }
23  }
24
25  int main(){
26    int N;
27    scanf("%d", &N);
28    printf("%d\n", get_ways(N));
29    return 0;
30  }
```

3.3 Range sum of static array

```
1   #include <cstdio>
2   #include <vector>
3   using namespace std;
4
5   int main(){
6     int N;
7     scanf("%d", &N);
8     vector<int> v;
9     v.reserve(N);
10    vector<int> sum;
11    sum.reserve(N+1);
12    sum.push_back(0);
13    for(int i = 0; i < N; ++i){
14      int t;
15      scanf("%d", &t);
16      v.push_back(t);
17      sum.push_back(sum.back() + t);
18    }
19    int M;
20    scanf("%d", &M);
21    for(int i = 0; i < M; ++i){
22      int a, b;
23      scanf("%d%d", &a, &b);
```

```
24          printf("%d\n", sum[b+1]- sum[a]);
25      }
26      return 0;
27  }
```