

Competitive Programming

Bjarki Ágúst Guðmundsson

Trausti Sæmundsson

Ingólfur Eðvarðsson

October 14, 2012

Contents

1	Data Structures	2
1.1	Union-Find	2
1.2	Segment Tree	2
1.3	Fenwick Tree	2
1.4	Interval Tree	2
2	Graphs	2
2.1	Breadth-First Search	2
2.2	Depth-First Search	3
2.3	Single Source Shortest Path	3
2.3.1	Dijkstra's algorithm	3
2.3.2	Bellman-Ford algorithm	3
2.4	All Pairs Shortest Path	3
2.4.1	Floyd-Warshall algorithm	3
2.5	Connected Components	3
2.5.1	Modified Breadth-First Search	3
2.6	Strongly Connected Components	3
2.6.1	Kosaraju's algorithm	3
2.6.2	Tarjan's algorithm	3
2.7	Topological Sort	3
2.7.1	Modified Breadth-First Search	3
2.8	Articulation Points/Bridges	3
2.8.1	Modified Depth-First Search	3
3	Number Theory	3
3.1	Binomial Coefficients	3

1 Data Structures

1.1 Union-Find

```
1 void uf_init(int* arr, int n) {
2     for (int i = 0; i < n; i++) {
3         arr[i] = i;
4     }
5 }
6
7 int uf_find(int* arr, int i) {
8     return arr[i] == i ? i : (arr[i] = uf_find(arr, arr[i]));
9 }
10
11 void uf_union(int* arr, int i, int j) {
12     arr[uf_find(arr, i)] = uf_find(uf_find(arr, j));
13 }
```

1.2 Segment Tree

1.3 Fenwick Tree

1.4 Interval Tree

2 Graphs

2.1 Breadth-First Search

An example of a breadth-first search that counts the number of edges on the shortest path from the starting vertex to the ending vertex. Note that it assumes that the two vertices are connected.

```
1 int bfs(int start, int end, vector<vi> adj_list) {
2     queue<ii> Q;
3     Q.push(ii(start, 0));
4
5     while (true) {
6         ii cur = Q.top(); Q.pop();
7
8         if (cur.first == end) {
9             return cur.second;
10        }
11
12        vi& adj = adj_list[cur.first];
13        for (vi::iterator it = adj.begin(); it != adj.end(); it++) {
14            Q.push(ii(*it, cur.second + 1));
15        }
16    }
17 }
```

2.2 Depth-First Search

2.3 Single Source Shortest Path

2.3.1 Dijkstra's algorithm

```
1  #define MAXEDGES 20000
2  bool done[MAXEDGES];
3
4  int dijkstra(int start, int end, vvii& adj_list) {
5      memset(done, 0, MAXEDGES);
6      priority_queue<ii, vii, greater<ii> > pq;
7      pq.push(ii(0, start));
8
9      while (!pq.empty()) {
10         ii current = pq.top(); pq.pop();
11         done[current.second] = true;
12
13         if (current.second == end)
14             return current.first;
15
16         vii &vtmp = adj_list[current.second];
17         for (vii::iterator it=vtmp.begin(); it != vtmp.end(); it++)
18             if (!done[it->second])
19                 pq.push(ii(current.first + it->first, it->second));
20     }
21     return -1;
22 }
```

2.3.2 Bellman-Ford algorithm

2.4 All Pairs Shortest Path

2.4.1 Floyd-Warshall algorithm

2.5 Connected Components

2.5.1 Modified Breadth-First Search

2.6 Strongly Connected Components

2.6.1 Kosaraju's algorithm

2.6.2 Tarjan's algorithm

2.7 Topological Sort

2.7.1 Modified Breadth-First Search

2.8 Articulation Points/Bridges

2.8.1 Modified Depth-First Search

3 Number Theory

3.1 Binomial Coefficients

The binomial coefficient $\binom{n}{k} = \frac{n!}{k!(n-k)!}$ is the number of ways to choose k items out of a total of n items.

```
1 int factorial(int n) {
2     int res = 1;
3     while (n) {
4         res *= n--;
5     }
6
7     return res;
8 }
9
10 int nck(int n, int k) {
11     return factorial(n) / factorial(k) / factorial(n - k);
12 }
13
14 void nck_precompute(int** arr, int n) {
15     for (int i = 0; i < n; i++)
16         arr[i][0] = arr[i][i] = 1;
17
18     for (int i = 1; i < n; i++)
19         for (int j = 1; j < i; j++)
20             arr[i][j] = arr[i - 1][j - 1] + arr[i - 1][j];
21 }
```

```
1 int nck(int n, int k) {  
2     if (n - k < k)  
3         k = n - k;  
4  
5     int res = 1;  
6     for (int i = 1; i <= k; i++)  
7         res = res * (n - (k - i)) / i;  
8  
9     return res;  
10 }
```