Competitive Programming

Bjarki Ágúst Guðmundsson Trausti Sæmundsson Ingólfur Eðvarðsson

October 29, 2012

Contents

1 Data Structures

1.1 Union-Find

```
1  class union_find {
2  private:
3          int* parent;
4          int cnt;
5
6  public:
7          union_find(int n) {
8                  parent = new int[cnt = n];
9                  for (int i = 0; i < cnt; i++)
10                         parent[i] = i;
11         }
12
13         union_find(const union_find& other) {
14                 parent = new int[cnt = other.cnt];
15                 for (int i = 0; i < cnt; i++)
16                         parent[i] = other.parent[i];
17         }
18
19         ~union_find() {
20                 if (parent) {
21                         delete[] parent;
22                         parent = NULL;
23                 }
24         }
25
26         int find(int i) {
27                 assert(parent != NULL);
28                 return parent[i] == i ? i : (parent[i] = find(parent[i]));
29         }
30
31         bool unite(int i, int j) {
32                 assert(parent != NULL);
33
34                 int ip = find(i),
35                         jp = find(j);
36
37                 parent[ip] = jp;
38                 return ip != jp;
39         }
40 };
```

1.2 Segment Tree

1.3 Fenwick Tree

1.4 Interval Tree

2 Graphs

2.1 Breadth-First Search

An implementation of a breadth-first search that counts the number of edges on the shortest path from the starting vertex to the ending vertex in the specified unweighted graph (which is represented with an adjacency list). Note that it assumes that the two vertices are connected.

```
int bfs(int start, int end, vvi adj_list) {
        queue<ii> Q;
        Q.push(ii(start, 0));

        while (true) {
                ii cur = Q.front(); Q.pop();

                if (cur.first == end)
                        return cur.second;

                vi& adj = adj_list[cur.first];
                for (vi::iterator it = adj.begin(); it != adj.end(); it++)
                        Q.push(ii(*it, cur.second + 1));
        }
}
```

Another implementation that doesn't assume the two vertices are connected. If there is no path from the starting vertex to the ending vertex, a `-1` is returned.

```
int bfs(int start, int end, vvi adj_list) {
        set<int> visited;
        queue<ii> Q;
        Q.push(ii(start, 0));
        visited.insert(start);

        while (!Q.empty()) {
                ii cur = Q.front(); Q.pop();

                if (cur.first == end)
                        return cur.second;

                vi& adj = adj_list[cur.first];
                for (vi::iterator it = adj.begin(); it != adj.end(); it++)
                        if (visited.find(*it) == visited.end()) {
                                Q.push(ii(*it, cur.second + 1));
                                visited.insert(*it);
```

```
18                          }
19          }
20
21          return -1;
22  }
```

2.2 Depth-First Search

2.3 Single Source Shortest Path

2.3.1 Dijkstra's algorithm

An implementation of Dijkstra's algorithm that returns the length of the shortest path from the starting vertex to the ending vertex.

```
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
22          return -1;
23  }
```

3 Mathematics

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) res *= n--;
4         return res;
5 }
6
7 int nck(int n, int k) {
8         return factorial(n) / factorial(k) / factorial(n - k);
9 }
```

```
1 void nck_precompute(int** arr, int n) {
2         for (int i = 0; i < n; i++)
3                 arr[i][0] = arr[i][i] = 1;
4
5         for (int i = 1; i < n; i++)
6                 for (int j = 1; j < i; j++)
7                         arr[i][j] = arr[i - 1][j - 1] + arr[i - 1][j];
8 }
```

```
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  }
```