A Competitive Programming Cheat Sheet

October 24, 2015

1 Binary Search

```
int search(int n, function<bool(int)> lt) {
        int i = 0, k = n;
        while (i < k) {
                int m = i + (k - i) / 2;
                if (lt(m)) { // max(lt(m)) s.t. lt(m) is true
                        i = m + 1;
                } else {
                        k = m;
                }
        }
        return i; // returns where needle SHOULD be!
}
```

```
remember lower_bound and upper_bound(begin, end) in stdlib
```

2 C++11 Features

```
#include <tuple>
#include <iostream>
using namespace std;
typedef tuple<int, int, int> Tuple;
int main() {
        Tuple t = Tuple(1, 2, 3);
        cout << get<0>(t) << " " << get<1>(t) << " " << get<2>(t) << endl;
        return 0;
}
```

```
#include <functional>
#include <iostream>
using namespace std;
int main(){
        function<void(int, int)> prnt = [](int a, int b){cout << a << " and " << b << endl;};
        int the_int = 42;
        function<void(int)> prnt2 = [&the_int](int a){cout << a << " THE_INT: " << the_int <<
        int leet = 1337;
        function<void()> prnt3 = [&](){cout << the_int << " vs. " << leet << endl;};
        function<void()> prnt4 = [=](){cout << "Captured by value, not reference" << the_int <
}
```

3 Disjoint Set

```
int findRoot(vector<int>& parent, int i) {
        if (parent[i] != i) parent[i] = findRoot(parent, parent[i]);
        return parent[i];
}

bool join(vector<int>& parent, int i, int j) {
        int p = findRoot(parent, i), q = findRoot(parent, j);
        if (p != q) {parent[q] = p; return true;}
        return false;
}
```

```
does not include rank -> probably O(log(n)). Tofix: empty tree has rank 0. join(a, b) where a.
```

4 Fenwick

```
void update(vector <T> &tree, int i, T amount) {
        for (; i < tree.size(); i |= i + 1) tree[i] += amount;
}

T sum(const vector <T> &tree, int i) {
        T s = T();
        for (; i > 0; i &= i - 1) s += tree[i - 1];
        return s;
}
```

```
Range query. Log n insert, log n lookup.
```

5 Geometry

```
// TODO(Unimplemented)
```

```
// TODO(Unimplemented)
```

6 Graph

```
typedef int ValueT;
const ValueT INF = 1<<29;

struct Edge {
        int from, to;
        ValueT weight;
        Edge(int a, int b, ValueT c) {from=a; to=b; weight=c;}
};

struct Node {
        ValueT dist;
        int idx;
        bool visited;
        Node() {dist=INF; idx=-1; visited=false;}
        Node(int i) {dist=INF; idx=i; visited=false;}
};
```

```
// TODO(Unimplemented)
```

```
// TODO(Unimplemented)
```

```
// TODO(Unimplemented)
```

```
const int INF = 1<<31;
const int MAX_V = 101;
const int MAX_E = 10000;
int dist[MAX_V][MAX_V];
Edge edges[MAX_E];
void floydWarshall(int size, int num_edge) {
        for (int i = 0; i < size; ++i)
                for (int j = 0; j < size; ++j)
                        dist[i][j] = INF;
        for (int k = 0; k < size; ++k) {dist[k][k] = 0;}
        for (int e = 0; e < num_edge; ++e) {
                Edge e = edges[e];
                dist[e.from][e.to] = min(dist[e.from][e.to], e.weight);
        }
        for (int k = 0; k < size; ++k) {
                for (int i = 0; i < size; i++) {
                        for (int j = 0; j < size; ++j) {
                                if (dist[i][k] == INF || dist[k][j] == INF) continue;
                                dist[i][j] = min(dist[i][j], dist[i][k] + dist[k][j]);
                                /* If solving with negative cycles copy this loop and change t
                                        'if (dist[k][k] < 0) dist[i][j] = -INF;'*/
                        }
                }
        }
}
```

```
const int MAX_V = 10005;
Node nodes[MAX_V];
vector<Edge> adjList[MAX_V];
typedef pair<ValueT, int> State;
inline ValueT dist(const State& s) { return s.first; }
inline int node(const State& s) { return s.second; }

struct MyCompare {
        bool operator() (const State& lhs, const State& rhs) { return dist(lhs) > dist(rhs); }
};

void dijkstra(int start) {
        priority_queue<State, vector<State>, MyCompare> pq;
        pq.push(State(0, start));
        nodes[start].dist = 0;
        while(!pq.empty()) {
           State state = pq.top(); pq.pop();
                if (nodes[node(state)].visited) continue;
                nodes[node(state)].visited = true;
                for (int i = 0; i < (int) adjList[node(state)].size(); ++i){
                        const Edge& e = adjList[node(state)][i];
                        ValueT next_dist = nodes[e.from].dist + e.weight;
                        if (next_dist < nodes[e.to].dist) {
                           nodes[e.to].dist = next_dist;
                                pq.push(State(next_dist, e.to));
                        }
                }
        }
}
```

```
// TODO(Unimplemented)
```

```
#include dijkstra
5 inline ValueT dist(const State& s) { return s.first + h(s.first, goal); }
where h does not overestimate distance to goal, and h is preferably monotonic
A* is polynomial if h is within O(log n) of real cost
```

```
const int MAX_V = 1005;
const int MAX_E = 5005;
Node nodes[MAX_V];
Edge edgeList[MAX_E];

void bellmanFord(int start, int num_nodes, int num_edges) {
        nodes[start].dist = 0;
        for (int i = 1; i < num_nodes; ++i) {
                for (int k = 0; k < num_edges; ++k) {
                        const Edge& e = edgeList[k];
                        if (nodes[e.from].dist != INF)
                            nodes[e.to].dist = min(nodes[e.to].dist, nodes[e.from].dist + e.we
                }
        }

        for (int i = 1; i < num_nodes; ++i) {
            for (int k = 0; k < num_edges; ++k) {
                    const Edge& e = edgeList[k];
                    if (nodes[e.from].dist == INF) continue;

                    ValueT dist = nodes[e.from].dist + e.weight;
                    ValueT other = nodes[e.to].dist;

                    if (dist < other)
                            nodes[e.to].dist = -INF;
            }
        }
}
```

7 Math

```
int gcd(int a, int b) {
        if (b == 0) return a;
        return gcd(b, a%b);
}

int lcm(int a, int b) {
        return abs(a*b) / gcd(a, b);
}

using i3tuple = tuple<int, int, int>;
using i2tuple = tuple<int, int>;

i3tuple extendedEuclid(int a, int b) {
        if (a == 0) return i3tuple(b, 0, 1);
        Tuple t = extendedEuclid(b % a, a);
        int _gcd = get<0>(t), x = get<1>(t), y = get<2>(t);
        return i3tuple(_gcd, x - (b/a) * y, y);
}

i2tuple diophanticEquation(int a, int b, int c) {
        if (c%gcd(a, b) != 0) return i2tuple(-1, -1); // Unsolvable
        i3tuple t = extendedEuclid(a, b);
        int _gcd = get<0>(t), x = get<1>(t), y = get<1>(t);
        y *= c; x *= c;
        return i2tuple(x, y);
}
```

```
vector<bool> sieve(int limit) {
        vector<bool> sieve(limit+1, true);
        sieve[0] = sieve[1] = false;
        sqlim = sqrt(limit) + 1;
        for (int i = 2; i <= sqlim; ++i) {
                if (sieve[i]) {
                        for (int j = i * i; j <= limit; j += i) {
                                sieve[j] = false;
                        }
                }
        }
        return sieve;
}
```

```
GCD, LCM and Extended Euclid is untested
```

8 String

```
// TODO(Unimplemented)
```

```
// TODO(Unimplemented)
// I (drathier) would suggest using a suffix array instead of a suffix tree, since SA is usual
```