

Competitive Programming Algorithms

Extracted from CP3 and December Algorithmics IOI Training Resources

January 2015

1 Data Structures

1.1 Stack

```
1 stack<int> s;  
2 s.push(1); s.push(2); s.push(3);  
3 while (!s.empty()) {  
4     cout << s.top() << endl;  
5     s.pop();  
6 } // prints 3 2 1
```

1.2 Sets

A set will contain only distinct elements. $O(\log n)$ insert, delete, search.

```
1 set<int> s;  
2 s.insert(1); s.insert(2); s.insert(3); s.insert(3); // set contains 1 2 3  
3 s.erase(1); // set contains 2 3  
4 if (s.find(1) == s.end()) cout << "Can't find 1\n";  
5 if (s.find(2) != s.end()) cout << "Can find 2\n";
```

1.3 Maps

Associative maps: get a value by a unique key. Like a set (unique key) with data attached. $O(\log n)$ insert, delete, search

```
1 map<int, int> m;  
2 m[5] = 34; m[33] = 1234;  
3 m.size(); // 2: only contains the elements you added  
4 map<string, int> ms;  
5 ms["hello_world"] = 42;  
6 ms["leeroy_jenkins"] = 12345;  
7 // iterating through elements  
8 map<int, int>::iterator it;  
9 for (it = ms.begin(); it != ms.end(); ++it){  
10     cout << it->first << "->" << it->second << "\n";  
11 }
```

1.4 Bitmasks

1. To set/turn on the j -th item (0-based indexing) of the set, use the bitwise OR operation $S | = (1 << j)$.
2. To check if the j -th item of the set is on, use the bitwise AND operation $T = S \& (1 << j)$.
If $T = 0$, then the j -th item of the set is off.
If $T \neq 0$ (to be precise, $T = (1 << j)$), then the j -th item of the set is on.

3. To clear/turn off the j -th item of the set, use the bitwise AND operation.

$S \&= \sim(1 \ll j)$ // \sim is the bitwise NOT operation

4. To toggle the j -th item of the set, use the bitwise XOR operation

$S \wedge= (1 \ll j)$

5. To get the value of the least significant bit that is on (first from the right), use $T = (S \& (-S))$.

6. To turn on all bits in a set of size n , use $S = (1 \ll n) - 1$.

1.5 Union-Find Disjoint Sets

```
1 class UnionFind{
2     private: vi p, rank;
3     public:
4         UnionFind(int N) {
5             rank.assign(N,0); p.assign(N,0);
6             for(int i = 0; i < N; i++) p[i] = i;
7         }
8         int findSet(int i){
9             return (p[i] == i) ? i : (p[i] = findSet(p[i]));
10        }
11        bool isSameSet(int i, int j){
12            return findSet(i) == findSet(j);
13        }
14        void unionSet(int i, int j){
15            if(!isSameSet(i,j)){
16                int x = findSet(i), y = findSet(j);
17                if(rank[x] > rank[y]) p[y] = x;
18                else{
19                    p[x] = y;
20                    if(rank[x] == rank[y]) rank[y]++;
21                }
22            }
23        }
24    }
```

1.6 Fenwick Trees

1.6.1 Implementation theory

Querying To query the range from 1 to i , add the buckets at position:

$p_0 = i$,

$p_1 = p_0 - \text{size of bucket } p_0$,

$p_2 = p_1 - \text{size of bucket } p_1, \text{ etc}$

Subtract size of bucket until 0

Updating To update, the ranges that contain i are:

$p_0 = i$,

$p_1 = p_0 + \text{size of bucket } p_0$

$p_2 = p_1 + \text{size of bucket } p_1, \text{ etc}$

```

1 long long ft[N + 1]; // note: Fenwick tree must be 1-indexed.
2 int ls(int x) { return x & (-x); }
3
4 void fenwick_update(int p, long long v){
5     for (; p <= N; p += ls(p)) ft[p] += v;
6 }
7
8 long long fenwick_query(int p){
9     long long sum = 0;
10    for (; p; p -= ls(p)) sum += ft[p];
11    return sum;
12 }

```

2 Sorts

sort, $O(n \log n)$ - sorts entire array

stable_sort, $O(n \log n)$ - keeps original order between equal elements

partial_sort, $O(n \log k)$ - sorts the k smallest entries

3 Conversions

```

1 string stlstr = "hello";
2 printf("%s", stlstr.c_str());
3
4 char cstr[] = "world";
5 cout << string(cstr) << endl;

```

4 Dynamic Programming

4.1 2D-Maxsum

For every pair of rows (eg. x_1, x_2):

- Sum each column between them (inclusive) into an 1D- array
 - Use W columns of 1D static sum
 - Or 2D static sum works too
- Perform 1D-Maxsum on this array

Complexity: $O(H^2W)$

```

1 int G[H+1][W+1], S[H+1][W+1], ans; /* 1-indexed */
2 /* W rows of 1D Static Sum */
3 for (int i = 1; i <= H; i++)
4     for (int j = 1; j <= W; j++)
5         S[i][j] = S[i-1][j] + G[i][j];
6 for (int x1 = 1; x1 <= H; x1++) {
7     for (int x2 = x1; x2 <= H; x2++) {
8         int cursum = S[x2][1] - S[x1-1][1];
9         for (int y = 2; y <= W; y++) {
10             cursum += max(cursum, 0) + S[x2][y] - S[x1-1][y];
11             ans = max(cursum, ans);
12         }
13     }
14 }

```

4.2 Lowest Common Substring

To recover LCS: start from `lcs[N][M]` and work backwards.

```

1 int lcs[1001][1001], A[1001], B[1001]; // A and B are 1-indexed here
2 for (int i = 0; i <= N; ++i) {
3     for (int j = 0; j <= M; ++j) {
4         if (i == 0 || j == 0) lcs[i][j] = 0;
5         if (A[i] == B[j]) lcs[i][j] = 1 + lcs[i-1][j-1];
6         else lcs[i][j] = max(lcs[i][j-1], lcs[i-1][j]);
7     }
8 }

```

5 Graphs

5.1 Topological Sort

```

1 void dfs(int vertex_id) {
2     if (visited[vertex_id]) return;
3     visited[vertex_id] = true;
4     for (auto i: adjList[vertex_id]) {
5         dfs(i);
6     }
7     topo.push_back(vertex_id);
8 }
9
10 for (int i = 0; i < V; ++i)
11     if (!visited[i]) dfs(i);
12
13 reverse(topo.begin(), topo.end());

```

5.2 Kruskal's

```

1 vector<pair<int,ii>> EdgeList; // (weight, two vertices) of the edge
2 for(int i = 0; i < E; i++){
3     scanf("%d_%d_%d", &u, &v, &w);
4     EdgeList.push_back(make_pair(w, ii(u, v)));
5 }
6 sort(EdgeList.begin(), EdgeList.end());
7
8 int mst_cost = 0;
9 UnionFind UF(V);
10 for(int i = 0; i < E; i++){
11     pair<int, ii> front = EdgeList[i];
12     if(!UF.isSameSet(front.second.first, front.second.second)){
13         mst_cost += front.first;
14         UF.unionSet(front.second.first, front.second.second);
15     }
16 } // note: number of disjoint sets must eventually be 1 for a valid MST
17 printf("MST_cost = %d", mst_cost);

```

5.3 Dijkstra's

$O((V+E)\log V)$, best for weighted graphs, works for negative weights (slower), unable to detect negative cycle.

```

1 vi dist(V, INF); dist[s] = 0; // INF = 1B to avoid overflow
2 priority_queue<ii, vector<ii>, greater<ii>> pq; pq.push(ii(0,s));
3 while(!pq.empty()){
4     ii front = pq.top(); pq.pop();
5     int d = front.first, u = front.second;
6     if(d > dist[u]) continue;
7     for(int j = 0; j < (int)AdjList[u].size(); j++){
8         ii v = AdjList[u][j];
9         if(dist[u] + v.second < dist[v.first]){
10             dist[v.first] = dist[u] + v.second;
11             pq.push(ii(dist[v.first], v.first));
12         }
13     }
14 }

```

5.4 Bellman Ford's

$O(VE)$, works for negative weight.

```

1 vi dist(V, INF); dist[s] = 0;
2 for(int i = 0; i < V - 1; i++){ // relax all E edges V-1 times
3     for(int u = 0; u < V; u++){
4         for(int j = 0; j < (int)AdjList[u].size(); j++){
5             ii v = AdjList[u][j];
6             dist[v.first] = min(dist[v.first], dist[u] + v.second);
7         } //relax
8     }
9 }

```

```

1 // after running Bellman Ford's algorithm shown above
2 bool hasNegativeCycle = false;
3 for(int u = 0; u < V; u++){
4     for(int j = 0; j < (int) AdjList[u].size(); j++){
5         if(v = AdjList[u][j];
6             if(dist[v.first] > dist[u] + v.second)
7                 hasNegativeCycle = true;
8     }
9 }
10 printf("Negative_Cycle_Exist?_\\%s\\n", hasNegativeCycle ? "Yes" : "No");

```

5.5 Floyd Warshall's

$V \leq 400$

```

1 // inside int main()
2 // precondition: AdjMat [i][j] contains the weight of edge (i,j)
3 // or INF (1B) if there is no such edge
4 // AdjMat is a 32-bit signed integer array
5 for (int k = 0; k < V; k++) // remember that loop order is k->i->j
6     for (int i = 0; i < V; i++)
7         for (int j = 0; j < V; j++)
8             AdjMat [i][j] = min(AdjMat [i][j] , AdjMat [i][j] + AdjMat[k][j]);

```

6 Mathematics

6.1 Modular Arithmetic

$$a \equiv b(\text{mod } m) \leftrightarrow a - b \equiv 0(\text{mod } m),$$

$$k * m \equiv 0(\text{mod } m),$$

$$a * c \equiv b * c(\text{mod } m) \leftrightarrow a \equiv b(\text{mod } m)$$

if m and c are coprime because of Euclid's lemma: If a prime divides the product of two numbers, it must divide at least one of those numbers.

6.2 Fast Exponentiation

```

1 int fastExp(ll base, ll p){
2     if(p==0) return 1;
3     else if(p==1) return base;
4     else{
5         ll res = fastExp(base, p/2);
6         res *= res;
7         res %= MOD;
8         if(p%2==1) res *= base%MOD;
9         return res%MOD;
10    }
11 }

```

6.3 GCD

Euclid's algorithm, $O(\log N)$ time.

```
1 ll gcd(ll a, ll b){
2     if(b == 0) return a;
3     if(a > b) return gcd(b, a%b);
4     if(b > a) return gcd(b, a);
5 }
```

6.4 LCM

$$lcm(m, n) = \frac{|m \cdot n|}{gcd(m, n)}$$

LCM can also be found by merging the prime decompositions of both m and n .

6.5 Sieve of Eratosthenes

```
1 ll sieveSize;
2 bitset<10000010> bs;
3 vi primes; // list of primes
4
5 void sieve(ll upperbound){ // create list of primes in [0..upperbound]
6     sieveSize = upperbound + 1; // add 1 to include upperbound
7     bs.set(); // set all bits to 1
8     bs[0] = bs[1] = 0; // except index 0 and 1
9     for(ll i = 2; i <= sieveSize; i++) if (bs[i]){
10         // cross out multiples of i starting from i * i
11         for(ll j = i * i; j <= sieveSize; j += i) bs[j] = 0;
12         primes.push_back(i);
13     }}
14
15 bool isPrime(ll N){
16     if(N <= sieveSize) return bs[N];
17     for(int i = 0; i < (int)primes.size(); i++)
18         if(N % primes[i] == 0) return false; // a good enough deterministic prime tester
19     return true;
20 } // only works for N <= (last prime in vi "primes")^2
21
22 int main(){
23     sieve(10000000);
24     printf("%d\n", isPrime(2147483647)); // return true
25     printf("%d\n", isPrime(136117223861)); // return false
26 }
```