Competitive Programming Reference

ngmh

Last Updated: 07/01/2026

Contents

1 Data Structures

Data Structure	Precomputation / Update	Query	Memory	Notes
Prefix Sum	O(N) / X	O(1)	O(N)	Associative Functions (+, XOR)
Sparse Table	O(N log N) / X	O(1)	O(N log N)	Non-Associative Functions (max, gcd)
Fenwick Tree	X / O(log N)	O(log N)	O(N)	Prefix Sum with Updates
Segment Tree	X / O(log N)	O(log N)	O(4N)	Allows more Information

Table 1: Quick Summary of Data Structures

1.1 Prefix Sums

1.1.1 1D

$O(N)$ precomputation, $O(1)$ query.

```
//Query - 1-Indexed
int query(int s, int e){
    return ps[e]-ps[s-1];
}

//Precomputation
ps[0] = 0;
for(int i = 1; i <= n; i++) ps[i] = ps[i-1]+a[i];
```

1.1.2 2D

$O(R \cdot C)$ precomputation, $O(1)$ query.

```
//Query - 1-Indexed
int query(int x1, int y1, int x2, int y2){
    return ps[x2][y2]-ps[x1-1][y2]-ps[x2][y1-1]+ps[x1-1][y1-1];
}

//Precomputation
for (int i = 0; i <= r; i++) ps[i][0] = 0;
for (int j = 0; j <= c; j++) ps[0][j] = 0;
for (int i = 1; i <= r; i++) {
    for (int j = 1; j <= c; j++) {
        ps[i][j] = ps[i-1][j]+ps[i][j-1]-ps[i-1][j-1]+a[i][j];
    }
}
```

2 Graph Theory

2.1 Depth First Search

Runs in $O(V)$.

```
// For adjacency lists
void dfs(int x, int p) {
    for (int y : adj[x]) {
        if (y != p) {
            dist[y] = dist[x]+1;
            dfs(y, x);
        }
    }
}
```

2.2 Breadth First Search

Runs in $O(V)$.

```
// For adjacency lists
visited[s] = 1;
dist[s] = 0;
q.push(s);
while (!q.empty()) {
    int f = q.front(); q.pop();
    for (int i : adjlist[f]) {
        if (!visited[i]) {
            q.push(i);
            visited[i] = 1;
            dist[i] = dist[f] + 1;
        }
    }
}
```

3 Dynamic Programming

3.1 Maxsum

3.1.1 1D

Kadane's Algorithm. Runs in $O(N)$.

```
int ans = nums[0], cur = nums[0];
for (int i = 1; i < nums.size(); i++) {
    if (cur < 0) cur = 0;
    cur += nums[i];
    ans = max(ans, cur);
}
```

3.2 Longest Increasing Subsequence

3.2.1 N^2 DP

```
int ans = 0, dp[n];
memset(dp, 0, sizeof(dp));
for (int i = 0; i < n; i++) {
    for (int j = 0; j < i; j++) {
        if (a[j] < a[i]) {
            dp[i] = max(dp[i], dp[j]);
        }
    }
    dp[i]++;
    ans = max(ans, dp[i]);
}
```

3.3 Coin Combinations

Runs in $O(N \cdot V)$.

```
int ways[v+1];
memset(ways, 0, sizeof(ways));
ways[0] = 1;
for (int i = 0; i < n; i++) {
    int c = coins[i];
    for (int sum = c; sum <= v; sum++) {
        ways[sum] = (ways[sum] + ways[sum - c]) % MOD;
    }
}
cout << ways[v];
```

3.4 Coin Change

Runs in $O(N \cdot V)$.

```
const int INF = 1e9;
vector<int> dp(v + 1, INF);
dp[0] = 0;
for (int i = 1; i <= v; i++) {
    for (int j = 0; j < n; j++) {
        if (i >= c[j] && dp[i - c[j]] != INF) {
            dp[i] = min(dp[i], dp[i - c[j]] + 1);
        }
    }
}
cout << dp[v];
```

3.5 Knapsack

3.5.1 0-1

Runs in $O(N \cdot S)$.

```
for (int i = 0; i < n; i++) {
    for (int j = s; j >= w[i]; j--) {
        dp[j] = max(dp[j], dp[j - w[i]] +v[i]);
    }
}
cout << dp[s];
```

3.6 Digit DP

Runs in $O(D)$.

```
// Example - Numbers
// Compute the number of palindrome free numbers in a given range

vector<int> num;
ll dp[20][11][11][2][2]; // idx, last1, last2, tight, hasStarted

ll derp(int pos, int last1, int last2, bool tight, bool hasStarted) {
    if(pos == num.size()) return 1; // successfully populated whole number

    if(dp[pos][last1][last2][tight][hasStarted] != -1) {
        // state already visited
        return dp[pos][last1][last2][tight][hasStarted];
    }

    ll res = 0;
    int limit = tight ? num[pos] : 9; // do we need to keep to the range
    for(int d = 0; d <= limit; d++) { // try all next digits
        bool newHasStarted = hasStarted || (d != 0);
        bool newTight = tight && (d == limit);
        // skip palindromes only if the number has started
        if(newHasStarted) {
            if(d == last1) continue; // palindrome length 2
            if(d == last2) continue; // palindrome length 3
        }
        int newLast1 = newHasStarted ? d : 10;
        int newLast2 = hasStarted ? last1 : 10;
        res += derp(pos+1, newLast1, newLast2, newTight, newHasStarted);
    }
    return dp[pos][last1][last2][tight][hasStarted] = res;
}

// convert number to digits
void dcmp(ll x){
    num.clear();
    if(x == 0) num.push_back(0);
    while(x > 0) {
        num.push_back(x % 10);
        x /= 10;
    }
    reverse(num.begin(), num.end());
}


// Total Valid with value <= x
// Use PIE to get number within [a, b]
ll solve(ll x){
    dcmp(x);
    memset(dp, -1, sizeof(dp));
    return derp(0, 10, 10, true, false);
}

// To compute the kth string satisfying
// Either binary search or build character by character:
int count = 0;
vector<int> ans;
for (int i = 0; i < n; i++) {
    int x = 0;
    for (int j = 1; j < 10; j++) { // adjust to size of alphabet
        if (count + dp(i, j) > k) {
            break;
        }
        x = j;
    }
    count += dp(i, x); // position i is bounded by x
    ans.push_back(x);
}
```

4 Math

5 Algorithms

5.1 Binary Search

Find the cuberoot of n. Runs in $O(\log N)$.

```
long long n; cin >> n;
long long mini = 0, maxi = 1e6, medi;
while (mini < maxi) {
    medi = mini+(maxi-mini)/2;
    if (medi * medi * medi >= n) maxi = medi;
    else mini = medi+1;
}
cout << mini << "\n";
```

5.2 Binary Search using Lifting

Find the cuberoot of n. Runs in $O(\log N)$.

```
long long n; cin >> n;
long long cur = 0, gap = 1e6, next;
while (gap > 0) {
    while (next = cur + gap, next * next * next < n) {
        cur = next;
    }
    gap >>= 1;
}
cout << cur+1 << "\n";
```

5.3 Sliding Set

Speeds up DP from $O(N^2)$ to $O(N \log N)$.

```
// Example - Candymountain
// Jump across with minimax candies
// Populate with initial window
for(int i = 0; i < k; i++) {
    dp[i] = candies[i];
    s.insert(candies[i]);
}
// Sliding Set
for(int i = k; i < n; i++){
    dp[i] = max(candies[i], *s.begin());
    s.erase(s.find(dp[i-k]));
    s.insert(dp[i]);
}
```

6 Miscellaneous

6.1 Fast I/O

Cannot use with `scanf`, `printf`.

```
ios_base::sync_with_stdio(false);
cin.tie(0);
```

6.2 Superfast I/O

Only for non-negative integer input.

```
inline ll ri () {
    ll x = 0;
    char ch = getchar_unlocked();
    while (ch < '0' || ch > '9') ch = getchar_unlocked();
    while (ch >= '0' && ch <= '9') {
        x = (x << 3) + (x << 1) + ch - '0';
        ch = getchar_unlocked();
    }
    return x;
}
```