Competitive Programming Reference

ngmh

December 2025

Contents

1 Data Structures **2**
1.1 Prefix Sums 2
1.1.1 1D 2
1.1.2 2D 2

2 Graph Theory **2**

3 Dynamic Programming **2**
3.1 Maxsum 2
3.1.1 1D 2
3.2 Longest Increasing Subsequence 3
3.2.1 N^2 DP 3
3.3 Coin Combinations 3
3.4 Coin Change 3

4 Math **3**

5 Algorithms **3**
5.1 Binary Search 3
5.2 Binary Search using Lifting 4

6 Miscellaneous **4**
6.1 Fast I/O 4
6.2 Superfast I/O 4

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

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

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

1.1.2 2D

```
//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 = 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

3 Dynamic Programming

3.1 Maxsum

3.1.1 1D

Kadane's Algorithm.

```
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

```
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

```
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];
```

4 Math

5 Algorithms

5.1 Binary Search

Find the cuberoot of 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.

```
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";
```

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;
}
```