Heap Thrust Reference 2026

ICPC Reference (Date 1 version)

Team Members:

Enrique Job Calderón (ksobrenat32)
Luis Daniel Salazar (luisdakan)
Gustavo Valenzuela (GusTimeTraveler)

Institution:
Facultad de Ingeniería, UNAM - CPCFI

November 13, 2025

Contents

1 Graph Theory **2**
1.1 Depth First Search (DFS) 2
1.2 Breadth First Search (BFS) 2
2 Data Structures **2**
2.1 Segment Tree 2
2.2 Fenwick Tree (BIT) 3
3 Range Queries **3**
3.1 Sparse Table 3
4 Mathematics **3**
4.1 Number Theory 3
4.1.1 GCD and LCM 3
4.1.2 Extended Euclidean Algorithm 4
4.1.3 Modular Arithmetic 4
4.2 Combinatorics 4
4.2.1 Binomial Coefficients 4
4.2.2 Catalan Numbers 5
4.3 Probability 5
4.3.1 Basic Formulas 5
4.3.2 Expected Value 5
4.4 Algebra 5
4.4.1 Matrix Operations 5
4.4.2 System of Linear Equations 5
4.5 Calculus 5
4.5.1 Derivatives 5
4.5.2 Integration 6
4.5.3 Series 6
5 String Algorithms **6**
5.1 KMP (Knuth-Morris-Pratt) 6
5.2 Z-Algorithm 6
6 Computational Geometry **7**
6.1 Basic Formulas 7
6.1.1 Distance and Dot Product 7
6.1.2 Cross Product 7
6.2 Convex Hull 7
7 Dynamic Programming **8**
7.1 Longest Increasing Subsequence 8
7.2 Knapsack 8

1 Graph Theory

1.1 Depth First Search (DFS)

Example implementation:

```
// Depth First Search
vector<int> adj[MAXN];
bool visited[MAXN];

void dfs(int u) {
    visited[u] = true;
    for (int v : adj[u]) {
        if (!visited[v]) {
            dfs(v);
        }
    }
}
```

Listing 1: DFS Implementation

1.2 Breadth First Search (BFS)

Example implementation:

```
// Breadth First Search
vector<int> adj[MAXN];
int dist[MAXN];

void bfs(int start) {
    queue<int> q;
    memset(dist, -1, sizeof(dist));
    dist[start] = 0;
    q.push(start);

    while (!q.empty()) {
        int u = q.front();
        q.pop();

        for (int v : adj[u]) {
            if (dist[v] == -1) {
                dist[v] = dist[u] + 1;
                q.push(v);
            }
        }
    }
}
```

Listing 2: BFS Implementation

2 Data Structures

2.1 Segment Tree

Example implementation:

```
// Segment Tree for Range Sum Query
class SegmentTree {
    vector<long long> tree;
    int n;

public:
    SegmentTree(vector<int>& arr) {
        n = arr.size();
        tree.resize(4 * n);
        build(arr, 0, 0, n - 1);
    }

    void build(vector<int>& arr, int node, int start, int end) {
        if (start == end) {
            tree[node] = arr[start];
        } else {
            int mid = (start + end) / 2;
            build(arr, 2*node+1, start, mid);
            build(arr, 2*node+2, mid+1, end);
            tree[node] = tree[2*node+1] + tree[2*node+2];
        }
    }

    void update(int node, int start, int end, int idx, int val) {
        if (start == end) {
            tree[node] = val;
        } else {
            int mid = (start + end) / 2;
            if (idx <= mid) {
                update(2*node+1, start, mid, idx, val);
            } else {
                update(2*node+2, mid+1, end, idx, val);
            }
            tree[node] = tree[2*node+1] + tree[2*node+2];
        }
    }

    long long query(int node, int start, int end, int l, int r) {
        if (r < start || end < l) return 0;
        if (l <= start && end <= r) return tree[node];
        int mid = (start + end) / 2;
        return query(2*node+1, start, mid, l, r) +
               query(2*node+2, mid+1, end, l, r);
    }

    void update(int idx, int val) { update(0, 0, n-1, idx, val); }
```

```
    long long query(int l, int r) { return query(0, 0, n-1, l, r); }
};
```

Listing 3: Segment Tree

2.2 Fenwick Tree (BIT)

Example implementation:

```
// Fenwick Tree (Binary Indexed Tree)
class FenwickTree {
    vector<long long> tree;
    int n;

public:
    FenwickTree(int n) : n(n), tree(n + 1, 0) {}

    void update(int idx, int delta) {
        for (++idx; idx <= n; idx += idx & -idx) {
            tree[idx] += delta;
        }
    }

    long long query(int idx) {
        long long sum = 0;
        for (++idx; idx > 0; idx -= idx & -idx) {
            sum += tree[idx];
        }
        return sum;
    }

    long long query(int l, int r) {
        return query(r) - (l > 0 ? query(l - 1) : 0);
    }
};
```

Listing 4: Fenwick Tree

3 Range Queries

3.1 Sparse Table

Example implementation:

```
// Sparse Table for Range Minimum Query
class SparseTable {
    vector<vector<int>> table;
    vector<int> log;
    int n;

public:
    SparseTable(vector<int>& arr) {
        n = arr.size();
        int maxLog = log2(n) + 1;
        table.assign(n, vector<int>(maxLog));
        log.assign(n + 1, 0);

        // Precompute logarithms
        for (int i = 2; i <= n; i++) {
            log[i] = log[i/2] + 1;
        }

        // Build sparse table
        for (int i = 0; i < n; i++) {
            table[i][0] = arr[i];
        }

        for (int j = 1; j < maxLog; j++) {
            for (int i = 0; i + (1 << j) <= n; i++) {
                table[i][j] = min(table[i][j-1],
                                  table[i + (1 << (j-1))][j-1]);
            }
        }
    }

    int query(int l, int r) {
        int j = log[r - l + 1];
        return min(table[l][j], table[r - (1 << j) + 1][j]);
    }
};
```

Listing 5: Sparse Table

4 Mathematics

4.1 Number Theory

4.1.1 GCD and LCM

$$\gcd(a, b) = \gcd(b, a \bmod b)$$

$$\text{lcm}(a, b) = \frac{ab}{\gcd(a, b)}$$

```
// Greatest Common Divisor
int gcd(int a, int b) {
    return b == 0 ? a : gcd(b, a % b);
}

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

Listing 6: GCD Implementation

4.1.2 Extended Euclidean Algorithm

For $ax + by = \gcd(a, b)$:

```
// Extended Euclidean Algorithm
// Returns gcd(a, b) and finds x, y such that ax + by = gcd(a, b)
int extgcd(int a, int b, int &x, int &y) {
    if (b == 0) {
        x = 1;
        y = 0;
        return a;
    }
    int x1, y1;
    int g = extgcd(b, a % b, x1, y1);
    x = y1;
    y = x1 - (a / b) * y1;
    return g;
}
```

Listing 7: Extended GCD

4.1.3 Modular Arithmetic

$$a^{-1} \bmod m \text{ exists iff } \gcd(a, m) = 1$$
$$a^{\phi(m)} \equiv 1 \pmod{m} \text{ (Euler's theorem)}$$
$$a^{p-1} \equiv 1 \pmod{p} \text{ (Fermat's little theorem)}$$

```
// Modular Exponentiation
long long mod_pow(long long base, long long exp, long long mod) {
    long long result = 1;
    base %= mod;
    while (exp > 0) {
        if (exp & 1) result = (result * base) % mod;
        base = (base * base) % mod;
        exp >>= 1;
    }
    return result;
}

// Modular Inverse using Fermat's Little Theorem
// Works when mod is prime
long long mod_inv(long long a, long long mod) {
    return mod_pow(a, mod - 2, mod);
}
```

Listing 8: Modular Exponentiation

4.2 Combinatorics

4.2.1 Binomial Coefficients

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}$$
$$\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}$$
$$\sum_{k=0}^{n} \binom{n}{k} = 2^n$$

```
// Binomial Coefficients with Modular Arithmetic
const int MOD = 1e9 + 7;
const int MAXN = 1e6 + 5;
long long fact[MAXN], inv_fact[MAXN];

long long mod_pow(long long base, long long exp, long long mod) {
    long long result = 1;
    while (exp > 0) {
        if (exp & 1) result = (result * base) % mod;
        base = (base * base) % mod;
        exp >>= 1;
    }
    return result;
}

void precompute() {
    fact[0] = 1;
    for (int i = 1; i < MAXN; i++) {
        fact[i] = (fact[i-1] * i) % MOD;
    }
    inv_fact[MAXN-1] = mod_pow(fact[MAXN-1], MOD - 2, MOD);
```

```
    for (int i = MAXN-2; i >= 0; i--) {
        inv_fact[i] = (inv_fact[i+1] * (i+1)) % MOD;
    }
}

long long nCr(int n, int r) {
    if (r < 0 || r > n) return 0;
    return (fact[n] * inv_fact[r] % MOD) * inv_fact[n-r] % MOD;
}
```

Listing 9: Binomial Coefficients

4.2.2 Catalan Numbers

$$C_n = \frac{1}{n+1}\binom{2n}{n} = \frac{(2n)!}{(n+1)!n!}$$
$$C_n = \sum_{i=0}^{n-1} C_i C_{n-1-i}$$
$$C_0 = 1, \quad C_{n+1} = \frac{2(2n+1)}{n+2} C_n$$

4.3 Probability

4.3.1 Basic Formulas

$$P(A \cup B) = P(A) + P(B) - P(A \cap B)$$
$$P(A|B) = \frac{P(A \cap B)}{P(B)}$$
$$P(A \cap B) = P(A|B) \cdot P(B) = P(B|A) \cdot P(A)$$

4.3.2 Expected Value

$$E[X] = \sum_i x_i \cdot P(X = x_i)$$
$$E[aX + b] = aE[X] + b$$
$$E[X + Y] = E[X] + E[Y]$$

4.4 Algebra

4.4.1 Matrix Operations

Matrix multiplication complexity: $O(n^3)$

$$(AB)_{ij} = \sum_{k=1}^{n} A_{ik} B_{kj}$$
$$(AB)^T = B^T A^T$$
$$\det(AB) = \det(A)\det(B)$$

4.4.2 System of Linear Equations

Gaussian elimination: $O(n^3)$

$$Ax = b$$
$$\text{If } \det(A) \neq 0, \text{ unique solution exists}$$

4.5 Calculus

4.5.1 Derivatives

$$(x^n)' = nx^{n-1}$$
$$(e^x)' = e^x$$
$$(\ln x)' = \frac{1}{x}$$
$$(\sin x)' = \cos x$$
$$(\cos x)' = -\sin x$$

4.5.2 Integration

$$\int x^n\,dx = \frac{x^{n+1}}{n+1} + C \quad (n \neq -1)$$
$$\int e^x\,dx = e^x + C$$
$$\int \frac{1}{x}\,dx = \ln|x| + C$$
$$\int \sin x\,dx = -\cos x + C$$
$$\int \cos x\,dx = \sin x + C$$

4.5.3 Series

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$$
$$\sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}$$
$$\sum_{i=0}^{n} x^i = \frac{1-x^{n+1}}{1-x} \quad (x \neq 1)$$

5 String Algorithms

5.1 KMP (Knuth-Morris-Pratt)

Example implementation:

```
// Knuth-Morris-Pratt Algorithm
vector<int> computeLPS(string pattern) {
    int m = pattern.length();
    vector<int> lps(m);
    int len = 0;
    lps[0] = 0;

    for (int i = 1; i < m; ) {
        if (pattern[i] == pattern[len]) {
            len++;
            lps[i] = len;
            i++;
        } else {
            if (len != 0) {
                len = lps[len - 1];
            } else {
                lps[i] = 0;
                i++;
            }
        }
    }
    return lps;
}

vector<int> KMP(string text, string pattern) {
    vector<int> lps = computeLPS(pattern);
    vector<int> matches;
    int n = text.length();
    int m = pattern.length();
    int i = 0, j = 0;

    while (i < n) {
        if (text[i] == pattern[j]) {
            i++;
            j++;
        }

        if (j == m) {
            matches.push_back(i - j);
            j = lps[j - 1];
        } else if (i < n && text[i] != pattern[j]) {
            if (j != 0) {
                j = lps[j - 1];
            } else {
                i++;
            }
        }
    }
    return matches;
}
```

Listing 10: KMP Algorithm

5.2 Z-Algorithm

Example implementation:

```
// Z-Algorithm
vector<int> z_function(string s) {
    int n = s.length();
    vector<int> z(n);
    int l = 0, r = 0;
```

```

    for (int i = 1; i < n; i++) {
        if (i <= r) {
            z[i] = min(r - i + 1, z[i - l]);
        }
        while (i + z[i] < n && s[z[i]] == s[i + z[i]]) {
            z[i]++;
        }
        if (i + z[i] - 1 > r) {
            l = i;
            r = i + z[i] - 1;
        }
    }
    return z;
}

// Pattern matching using Z-algorithm
vector<int> patternMatch(string text, string pattern) {
    string combined = pattern + "$" + text;
    vector<int> z = z_function(combined);
    vector<int> matches;
    int m = pattern.length();

    for (int i = m + 1; i < combined.length(); i++) {
        if (z[i] == m) {
            matches.push_back(i - m - 1);
        }
    }
    return matches;
}
```

Listing 11: Z-Algorithm

6 Computational Geometry

6.1 Basic Formulas

6.1.1 Distance and Dot Product

$$d(P_1, P_2) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$
$$\vec{a} \cdot \vec{b} = a_x b_x + a_y b_y = |\vec{a}||\vec{b}| \cos\theta$$

6.1.2 Cross Product

$$\vec{a} \times \vec{b} = a_x b_y - a_y b_x$$
$$|\vec{a} \times \vec{b}| = |\vec{a}||\vec{b}| \sin\theta$$

6.2 Convex Hull

Example implementation:

```
// Convex Hull using Graham Scan
struct Point {
    long long x, y;
    bool operator<(const Point& p) const {
        return x < p.x || (x == p.x && y < p.y);
    }
};

long long cross(Point O, Point A, Point B) {
    return (A.x - O.x) * (B.y - O.y) - (A.y - O.y) * (B.x - O.x);
}

vector<Point> convexHull(vector<Point> points) {
    int n = points.size(), k = 0;
    if (n <= 3) return points;

    sort(points.begin(), points.end());
    vector<Point> hull(2 * n);

    // Build lower hull
    for (int i = 0; i < n; i++) {
        while (k >= 2 && cross(hull[k-2], hull[k-1], points[i]) <=
            0) k--;
        hull[k++] = points[i];
    }

    // Build upper hull
    for (int i = n - 2, t = k + 1; i >= 0; i--) {
        while (k >= t && cross(hull[k-2], hull[k-1], points[i]) <=
            0) k--;
        hull[k++] = points[i];
    }

    hull.resize(k - 1);
    return hull;
}
```

Listing 12: Convex Hull (Graham Scan)

7 Dynamic Programming

7.1 Longest Increasing Subsequence

Example implementation:

```
// Longest Increasing Subsequence in O(n log n)
int lis(vector<int>& arr) {
    vector<int> dp;

    for (int x : arr) {
        auto it = lower_bound(dp.begin(), dp.end(), x);
        if (it == dp.end()) {
            dp.push_back(x);
        } else {
            *it = x;
        }
    }

    return dp.size();
}

// If you need to reconstruct the sequence
vector<int> lisSequence(vector<int>& arr) {
    int n = arr.size();
    vector<int> dp, parent(n, -1), indices;

    for (int i = 0; i < n; i++) {
        auto it = lower_bound(dp.begin(), dp.end(), arr[i]);
        int pos = it - dp.begin();

        if (it == dp.end()) {
            dp.push_back(arr[i]);
            indices.push_back(i);
        } else {
            *it = arr[i];
            indices[pos] = i;
        }

        if (pos > 0) {
            parent[i] = indices[pos - 1];
        }
    }

    vector<int> result;
    int curr = indices.back();
    while (curr != -1) {
        result.push_back(arr[curr]);
        curr = parent[curr];
    }
    reverse(result.begin(), result.end());
    return result;
}
```

Listing 13: LIS in O(n log n)

7.2 Knapsack

Example implementation:

```
// 0-1 Knapsack Problem
int knapsack(int W, vector<int>& weights, vector<int>& values) {
    int n = weights.size();
    vector<vector<int>> dp(n + 1, vector<int>(W + 1, 0));

    for (int i = 1; i <= n; i++) {
        for (int w = 0; w <= W; w++) {
            dp[i][w] = dp[i-1][w];
            if (weights[i-1] <= w) {
                dp[i][w] = max(dp[i][w],
                               dp[i-1][w - weights[i-1]] + values[i-1]);
            }
        }
    }

    return dp[n][W];
}

// Space-optimized version
int knapsackOptimized(int W, vector<int>& weights, vector<int>& values) {
    int n = weights.size();
    vector<int> dp(W + 1, 0);

    for (int i = 0; i < n; i++) {
        for (int w = W; w >= weights[i]; w--) {
            dp[w] = max(dp[w], dp[w - weights[i]] + values[i]);
        }
    }

    return dp[W];
}
```

Listing 14: 0-1 Knapsack