Competitive Programming Teknikleri Kılavuzu

Algoritmalar, Veri Yapıları ve Optimizasyon Teknikleri

=blue!5=blue!50=0.8

Bu kılavuz competitive programming yarışmalarında başarılı olmak için gerekli olan temel ve ileri seviye teknikleri kapsamaktadır.

13 Temmuz 2025

İçindekiler

Giriş

Bu kılavuz, competitive programming dünyasında başarılı olmak isteyen programcılar için hazırlanmış kapsamlı bir referans kaynağıdır. Algoritmalardan veri yapılarına, optimizasyon tekniklerinden platform-spesifik stratejilere kadar geniş bir yelpazede konuları ele almaktadır.

Kılavuzun amacı, hem başlangıç seviyesindeki hem de ileri seviyedeki yarışmacılara pratik ve uygulanabilir teknikler sunmaktır. Her bölüm, gerçek yarışma problemlerinde karşılaşabileceğiniz durumları göz önünde bulundurarak tasarlanmıştır.

Bölüm 1

Temel Stratejiler

1.1 Problem Çözme Yaklaşımı

Başarılı competitive programming için sistematik bir yaklaşım benimsenmelidir. Aşağıdaki adımları takip etmek, problem çözme sürecinizi önemli ölçüde geliştirecektir:

Strateji

Problem Çözme Süreci:

1. Problemi dikkatli bir şekilde okuyun (2-3 kez)
2. Verilen örnekleri kağıt üzerinde manuel olarak çözün
3. Problem tipini belirleyin ve uygun algoritmayı seçin
4. Zaman ve alan karmaşıklığını hesaplayın
5. Kenar durumları (edge cases) düşünün
6. Kodu yazın ve test edin

1.2 Hızlı Başlangıç Template

Her yarışmada kullanabileceğiniz standart bir template hazırlamak, değerli zaman kazandıracaktır:

```
#include <bits/stdc++.h>
using namespace std;

// Type definitions
#define ll long long
#define vi vector<int>
#define vll vector<long long>
#define pii pair<int, int>
#define pll pair<long long, long long>

// Utility macros
#define all(x) x.begin(), x.end()
#define pb push_back
#define mp make_pair
#define fi first
#define se second
#define sz(x) (int)(x).size()

// Constants
const int MOD = 1e9 + 7;
const int INF = 1e9;
const ll LLINF = 1e18;
const double EPS = 1e-9;
```

```
24
25 // Fast I/O
26 void fastIO() {
27     ios_base::sync_with_stdio(false);
28     cin.tie(NULL);
29     cout.tie(NULL);
30 }
31
32 int main() {
33     fastIO();
34
35     // Ana kod buraya yaz l r
36
37     return 0;
38 }
```

Listing 1.1: C++ Competitive Programming Template

1.3 Input/Output Optimizasyonu

Büyük veri setleriyle çalışırken I/O hızı kritik öneme sahiptir:

```
1  // H zl  I/O ayarlar
2  ios_base::sync_with_stdio(false);
3  cin.tie(NULL);
4  cout.tie(NULL);
5
6  // Dosya I/O (gerekti inde)
7  freopen("input.txt", "r", stdin);
8  freopen("output.txt", "w", stdout);
9
10 // C-style I/O (bazen daha h zl )
11 scanf("%d", &n);
12 printf("%d\n", result);
13
14 // Buffer boyutu art rma
15 setvbuf(stdout, NULL, _IOFBF, 1000000);
```

Listing 1.2: I/O Optimizasyon Teknikleri

Bölüm 2

Algoritma Teknikleri

2.1 Greedy Algoritmaları

Greedy algoritmaları, her adımda yerel olarak en iyi seçimi yapan algoritmalarıdır.

Örnek

Activity Selection Problem: Maksimum sayıda aktivite seçmek için bitiş zamanına göre sıralama yapılır.

```
struct Activity {
    int start, end, id;

    bool operator<(const Activity& other) const {
        return end < other.end;  // Biti  zaman na g re  s rala
    }
};

int activitySelection(vector<Activity>& activities) {
    sort(activities.begin(), activities.end());

    int count = 1;
    int lastEnd = activities[0].end;

    for (int i = 1; i < activities.size(); i++) {
        if (activities[i].start >= lastEnd) {
            count++;
            lastEnd = activities[i].end;
        }
    }

    return count;
}
```

Listing 2.1: Activity Selection Greedy Algorithm

2.2 Dynamic Programming

Dynamic Programming (DP), alt problemlerin sonuçlarını saklayarak aynı hesaplamaları tekrar yapmayı önleyen bir tekniktir.

2.2.1 Memoization Yaklaşımı

```
map<int, ll> memo;

ll fib(int n) {
    if (n <= 1) return n;
```

```
    if (memo.count(n)) return memo[n];
    return memo[n] = fib(n-1) + fib(n-2);
}
```

Listing 2.2: Fibonacci - Memoization

2.2.2 Knapsack Problem

```
int knapsack(vector<int>& weights, vector<int>& values, int
    capacity) {
    int n = weights.size();
    vector<vector<int>> dp(n + 1, vector<int>(capacity + 1, 0));

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

    return dp[n][capacity];
}
```

Listing 2.3: 0/1 Knapsack Problem

2.3 Binary Search Variations

Binary search, sıralı dizilerde logaritmik zamanda arama yapmanızı sağlar.

```
int lowerBound(vector<int>& arr, int target) {
    int left = 0, right = arr.size();

    while (left < right) {
        int mid = left + (right - left) / 2;
        if (arr[mid] < target) {
            left = mid + 1;
        } else {
            right = mid;
        }
    }

    return left;
}
```

Listing 2.4: Lower Bound Implementation

2.3.1 Binary Search on Answer

```
bool canSolve(int x, vector<int>& data) {
    // x de eri ile    zm    m mk n m  kontrol et
    // Implementation problem-specific
```

```
    return true;
}

int binarySearchAnswer(int low, int high, vector<int>& data) {
    int answer = -1;

    while (low <= high) {
        int mid = low + (high - low) / 2;

        if (canSolve(mid, data)) {
            answer = mid;
            high = mid - 1;  // Daha k      k cevap ara
        } else {
            low = mid + 1;
        }
    }

    return answer;
}
```

Listing 2.5: Binary Search on Answer Pattern

2.4 Graph Algorithms

2.4.1 Depth-First Search (DFS)

```
vector<vector<int>> adj;
vector<bool> visited;

void dfs(int node) {
    visited[node] = true;

    for (int neighbor : adj[node]) {
        if (!visited[neighbor]) {
            dfs(neighbor);
        }
    }
}
```

Listing 2.6: DFS Implementation

2.4.2 Breadth-First Search (BFS)

```
vector<int> bfs(int start, int n) {
    vector<int> dist(n, -1);
    queue<int> q;

    q.push(start);
    dist[start] = 0;

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

        for (int neighbor : adj[curr]) {
            if (dist[neighbor] == -1) {
```

```
                dist[neighbor] = dist[curr] + 1;
                q.push(neighbor);
            }
        }
    }

    return dist;
}
```

Listing 2.7: BFS Implementation

2.4.3 Dijkstra's Algorithm

```
vector<ll> dijkstra(int start, int n, vector<vector<pii>>& adj) {
    vector<ll> dist(n, LLINF);
    priority_queue<pll, vector<pll>, greater<pll>> pq;

    dist[start] = 0;
    pq.push({0, start});

    while (!pq.empty()) {
        ll d = pq.top().first;
        int u = pq.top().second;
        pq.pop();

        if (d > dist[u]) continue;

        for (auto& edge : adj[u]) {
            int v = edge.first;
            ll w = edge.second;

            if (dist[u] + w < dist[v]) {
                dist[v] = dist[u] + w;
                pq.push({dist[v], v});
            }
        }
    }

    return dist;
}
```

Listing 2.8: Dijkstra's Shortest Path Algorithm

Bölüm 3

Veri Yapıları

3.1 Segment Tree

Segment Tree, aralık sorguları ve güncellemeleri için güçlü bir veri yapısıdır.

```
class SegmentTree {
private:
    vector<ll> tree;
    int n;

    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, start, mid);
            build(arr, 2*node+1, mid+1, end);
            tree[node] = tree[2*node] + tree[2*node+1];
        }
    }

    void updateHelper(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) {
                updateHelper(2*node, start, mid, idx, val);
            } else {
                updateHelper(2*node+1, mid+1, end, idx, val);
            }
            tree[node] = tree[2*node] + tree[2*node+1];
        }
    }

    ll queryHelper(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 queryHelper(2*node, start, mid, l, r) +
               queryHelper(2*node+1, mid+1, end, l, r);
    }

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

```
    }

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

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

Listing 3.1: Segment Tree Implementation

3.2 Union-Find (Disjoint Set)

```
class UnionFind {
private:
    vector<int> parent, rank;
    int components;

public:
    UnionFind(int n) : components(n) {
        parent.resize(n);
        rank.resize(n, 0);
        for (int i = 0; i < n; i++) {
            parent[i] = i;
        }
    }

    int find(int x) {
        if (parent[x] != x) {
            parent[x] = find(parent[x]);  // Path compression
        }
        return parent[x];
    }

    bool unite(int x, int y) {
        int px = find(x), py = find(y);
        if (px == py) return false;

        if (rank[px] < rank[py]) {
            parent[px] = py;
        } else if (rank[px] > rank[py]) {
            parent[py] = px;
        } else {
            parent[py] = px;
            rank[px]++;
        }

        components--;
        return true;
    }

    bool connected(int x, int y) {
        return find(x) == find(y);
    }

```

```
    int getComponents() {
        return components;
    }
};
```

Listing 3.2: Union-Find with Path Compression and Union by Rank

3.3 Trie (Prefix Tree)

```
struct TrieNode {
    TrieNode* children[26];
    bool isEndOfWord;
    int count;

    TrieNode() {
        isEndOfWord = false;
        count = 0;
        for (int i = 0; i < 26; i++) {
            children[i] = nullptr;
        }
    }
};

class Trie {
private:
    TrieNode* root;

public:
    Trie() {
        root = new TrieNode();
    }

    void insert(string word) {
        TrieNode* curr = root;
        for (char c : word) {
            int index = c - 'a';
            if (curr->children[index] == nullptr) {
                curr->children[index] = new TrieNode();
            }
            curr = curr->children[index];
            curr->count++;
        }
        curr->isEndOfWord = true;
    }

    bool search(string word) {
        TrieNode* curr = root;
        for (char c : word) {
            int index = c - 'a';
            if (curr->children[index] == nullptr) {
                return false;
            }
            curr = curr->children[index];
        }
        return curr->isEndOfWord;
    }

```

```
    int countWordsWithPrefix(string prefix) {
        TrieNode* curr = root;
        for (char c : prefix) {
            int index = c - 'a';
            if (curr->children[index] == nullptr) {
                return 0;
            }
            curr = curr->children[index];
        }
        return curr->count;
    }
};
```

Listing 3.3: Trie Data Structure

Bölüm 4

Matematiksel Teknikler

4.1 Modular Arithmetic

Büyük sayılarla çalışırken overflow problemini önlemek için modular arithmetic kullanılır.

```
const int MOD = 1e9 + 7;

// Modular exponentiation
ll power(ll base, ll exp, ll mod = MOD) {
    ll result = 1;
    base %= mod;

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

    return result;
}

// Modular inverse (when mod is prime)
ll modInverse(ll a, ll mod = MOD) {
    return power(a, mod - 2, mod);
}

// Modular multiplication
ll mulmod(ll a, ll b, ll mod = MOD) {
    return ((a % mod) * (b % mod)) % mod;
}

// Modular addition
ll addmod(ll a, ll b, ll mod = MOD) {
    return ((a % mod) + (b % mod)) % mod;
}

// Modular subtraction
ll submod(ll a, ll b, ll mod = MOD) {
    return ((a % mod) - (b % mod) + mod) % mod;
}
```

Listing 4.1: Modular Arithmetic Functions

4.1.1 Combinatorics with Modular Arithmetic

```
vector<ll> fact, invFact;

```

```
void precomputeFactorials(int n) {
    fact.resize(n + 1);
    invFact.resize(n + 1);

    fact[0] = 1;
    for (int i = 1; i <= n; i++) {
        fact[i] = (fact[i-1] * i) % MOD;
    }

    invFact[n] = modInverse(fact[n]);
    for (int i = n - 1; i >= 0; i--) {
        invFact[i] = (invFact[i+1] * (i+1)) % MOD;
    }
}

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

Listing 4.2: nCr Modulo p Calculation

4.2 Number Theory

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

// Least Common Multiple
ll lcm(ll a, ll b) {
    return (a / gcd(a, b)) * b;
}

// Extended Euclidean Algorithm
ll extgcd(ll a, ll b, ll& x, ll& y) {
    if (b == 0) {
        x = 1, y = 0;
        return a;
    }
    ll x1, y1;
    ll g = extgcd(b, a % b, x1, y1);
    x = y1;
    y = x1 - (a / b) * y1;
    return g;
}

// Sieve of Eratosthenes
vector<bool> sieve(int n) {
    vector<bool> isPrime(n + 1, true);
    isPrime[0] = isPrime[1] = false;

    for (int i = 2; i * i <= n; i++) {
        if (isPrime[i]) {
            for (int j = i * i; j <= n; j += i) {
                isPrime[j] = false;
            }
```

```
        }
    }

    return isPrime;
}

// Prime factorization
vector<int> primeFactors(int n) {
    vector<int> factors;

    for (int i = 2; i * i <= n; i++) {
        while (n % i == 0) {
            factors.push_back(i);
            n /= i;
        }
    }

    if (n > 1) {
        factors.push_back(n);
    }

    return factors;
}
```

Listing 4.3: Number Theory Functions

Bölüm 5

Optimizasyon Teknikleri

5.1 Bit Manipulation

```cpp
// Temel bit operations
int setBit(int n, int pos) { return n | (1 << pos); }
int clearBit(int n, int pos) { return n & ~(1 << pos); }
int toggleBit(int n, int pos) { return n ^ (1 << pos); }
bool checkBit(int n, int pos) { return (n & (1 << pos)) != 0; }

// Count set bits
int countSetBits(int n) {
    return __builtin_popcount(n);
}

// Check if power of 2
bool isPowerOfTwo(int n) {
    return n > 0 && (n & (n - 1)) == 0;
}

// Next power of 2
int nextPowerOfTwo(int n) {
    if (isPowerOfTwo(n)) return n;
    return 1 << (32 - __builtin_clz(n));
}

// Iterate through all subsets
void generateAllSubsets(vector<int>& arr) {
    int n = arr.size();

    for (int mask = 0; mask < (1 << n); mask++) {
        vector<int> subset;

        for (int i = 0; i < n; i++) {
            if (mask & (1 << i)) {
                subset.push_back(arr[i]);
            }
        }

        // Process subset
        for (int x : subset) {
            cout << x << " ";
        }
        cout << endl;
    }
}
```

Listing 5.1: Bit Manipulation Techniques

5.2 Two Pointers Technique

```cpp
// Two sum in sorted array
pair<int, int> twoSum(vector<int>& arr, int target) {
    int left = 0, right = arr.size() - 1;

    while (left < right) {
        int sum = arr[left] + arr[right];
        if (sum == target) {
            return {left, right};
        } else if (sum < target) {
            left++;
        } else {
            right--;
        }
    }

    return {-1, -1};
}

// Remove duplicates from sorted array
int removeDuplicates(vector<int>& arr) {
    if (arr.empty()) return 0;

    int writeIndex = 1;
    for (int i = 1; i < arr.size(); i++) {
        if (arr[i] != arr[i-1]) {
            arr[writeIndex++] = arr[i];
        }
    }

    return writeIndex;
}

// Container with most water
int maxArea(vector<int>& height) {
    int left = 0, right = height.size() - 1;
    int maxWater = 0;

    while (left < right) {
        int width = right - left;
        int currentWater = width * min(height[left], height[right])
            ;
        maxWater = max(maxWater, currentWater);

        if (height[left] < height[right]) {
            left++;
        } else {
            right--;
        }
    }

    return maxWater;
}
```

Listing 5.2: Two Pointers Examples

5.3 Sliding Window

```cpp
// Maximum sum subarray of size k
int maxSumSubarray(vector<int>& arr, int k) {
    int n = arr.size();
    if (n < k) return -1;

    //  lk  pencereyi hesapla
    int windowSum = 0;
    for (int i = 0; i < k; i++) {
        windowSum += arr[i];
    }

    int maxSum = windowSum;

    // Kayan pencere
    for (int i = k; i < n; i++) {
        windowSum = windowSum - arr[i-k] + arr[i];
        maxSum = max(maxSum, windowSum);
    }

    return maxSum;
}

// Longest substring with at most k distinct characters
int longestSubstringKDistinct(string s, int k) {
    if (k == 0) return 0;

    unordered_map<char, int> charCount;
    int left = 0, maxLen = 0;

    for (int right = 0; right < s.length(); right++) {
        charCount[s[right]]++;

        while (charCount.size() > k) {
            charCount[s[left]]--;
            if (charCount[s[left]] == 0) {
                charCount.erase(s[left]);
            }
            left++;
        }

        maxLen = max(maxLen, right - left + 1);
    }

    return maxLen;
}
```

Listing 5.3: Sliding Window Techniques

Bölüm 6

Debugging ve Test Stratejileri

6.1 Debug Makroları

```
#ifdef LOCAL
#define debug(x) cerr << #x << " = " << x << endl
#define debug2(x, y) cerr << #x << " = " << x << ", " << #y << " = " << y << endl
#define debug3(x, y, z) cerr << #x << " = " << x << ", " << #y << " = " << y << ", " << #z << " = " << z << endl
#else
#define debug(x)
#define debug2(x, y)
#define debug3(x, y, z)
#endif

// Array/vector yazd rma
template<typename T>
void printArray(const vector<T>& arr, string name = "Array") {
    cerr << name << ": ";
    for (const auto& x : arr) {
        cerr << x << " ";
    }
    cerr << endl;
}

// Matrix yazd rma
template<typename T>
void printMatrix(const vector<vector<T>>& matrix, string name = "Matrix") {
    cerr << name << ":\n";
    for (const auto& row : matrix) {
        for (const auto& x : row) {
            cerr << x << " ";
        }
        cerr << endl;
    }
}

// Timing macro
#ifdef LOCAL
#define TIME_START auto start_time = chrono::high_resolution_clock::now()
#define TIME_END auto end_time = chrono::high_resolution_clock::now(); \
    auto duration = chrono::duration_cast<chrono::milliseconds>(end_time - start_time); \
    cerr << "Execution time: " << duration.count() << " ms" << endl
#else
#define TIME_START
```

```
#define TIME_END
#endif
```

Listing 6.1: Debugging Macros

6.2 Test Case Oluşturma

```
#include <random>

class TestGenerator {
private:
    mt19937 rng;

public:
    TestGenerator() : rng(chrono::steady_clock::now().
       time_since_epoch().count()) {}

    int randInt(int min_val, int max_val) {
        uniform_int_distribution<int> dist(min_val, max_val);
        return dist(rng);
    }

    vector<int> randArray(int size, int min_val, int max_val) {
        vector<int> arr(size);
        for (int i = 0; i < size; i++) {
            arr[i] = randInt(min_val, max_val);
        }
        return arr;
    }

    string randString(int length, bool lowercase_only = true) {
        string result;
        for (int i = 0; i < length; i++) {
            if (lowercase_only) {
                result += 'a' + randInt(0, 25);
            } else {
                result += 'A' + randInt(0, 25);
            }
        }
        return result;
    }

    vector<pair<int, int>> randGraph(int vertices, int edges) {
        vector<pair<int, int>> graph;
        set<pair<int, int>> used;

        while (graph.size() < edges) {
            int u = randInt(0, vertices - 1);
            int v = randInt(0, vertices - 1);
            if (u != v && used.find({u, v}) == used.end()) {
                graph.push_back({u, v});
                used.insert({u, v});
                used.insert({v, u});
            }
        }

        return graph;
```

```
    }
};

//  rnek  kullan m
void generateTestCase() {
    TestGenerator gen;

    int n = gen.randInt(1, 100);
    vector<int> arr = gen.randArray(n, 1, 1000);

    cout << n << endl;
    for (int x : arr) {
        cout << x << " ";
    }
    cout << endl;
}
```

Listing 6.2: Random Test Case Generation

6.3 Assertions ve Validation

```
#define ASSERT(condition, message) \
    if (!(condition)) { \
        cerr << "Assertion failed: " << message << endl; \
        cerr << "File: " << __FILE__ << ", Line: " << __LINE__ <<
           endl; \
        exit(1); \
    }

#define ASSERT_RANGE(value, min_val, max_val) \
    ASSERT((value) >= (min_val) && (value) <= (max_val), \
           #value " is out of range [" #min_val ", " #max_val "]")

// Input validation   rnei
void validateInput(int n, vector<int>& arr) {
    ASSERT_RANGE(n, 1, 100000);
    ASSERT(arr.size() == n, "Array size mismatch");

    for (int x : arr) {
        ASSERT_RANGE(x, 1, 1000000);
    }
}
```

Listing 6.3: Assertion Macros

Bölüm 7

Zaman Yönetimi

7.1 Problem Sıralama Stratejisi

Strateji

Yarışma Sırasında Problem Sıralaması:

1. Tüm problemleri hızla gözden geçirin (5-10 dakika)
2. En kolay görünen problemden başlayın
3. Her problemi çözdükten sonra bir sonraki en kolay probleme geçin
4. Takıldığınız problemde 15-20 dakikadan fazla vakit harcamayın
5. Zor problemleri yarışmanın son kısmına bırakın

7.2 Zaman Tahsisi

Aktivite	2 Saat	3 Saat
Problem okuma	10 dk	15 dk
Kolay problemler (2-3 tane)	40 dk	60 dk
Orta problemler (2-3 tane)	60 dk	90 dk
Zor problemler	5 dk	30 dk
Buffer/Review	5 dk	5 dk

Tablo 7.1: Örnek Zaman Tahsisi

7.3 Hızlı Kodlama İpuçları

```
// D ng  k saltmalar
#define FOR(i, a, b) for (int i = (a); i < (b); i++)
#define RFOR(i, a, b) for (int i = (a); i >= (b); i--)
#define REP(i, n) FOR(i, 0, n)
#define RREP(i, n) RFOR(i, n-1, 0)

// Container k saltmalar
#define ALL(x) (x).begin(), (x).end()
#define SZ(x) (int)(x).size()
#define SORT(x) sort(ALL(x))
#define REVERSE(x) reverse(ALL(x))

// Template fonksiyonlar
template<typename T>
```

```
T gcd(T a, T b) { return b ? gcd(b, a % b) : a; }

template<typename T>
T lcm(T a, T b) { return a / gcd(a, b) * b; }

template<typename T>
T power(T base, T exp, T mod) {
    T result = 1;
    while (exp > 0) {
        if (exp % 2 == 1) result = (result * base) % mod;
        base = (base * base) % mod;
        exp /= 2;
    }
    return result;
}
```

Listing 7.1: Time-Saving Macros

Bölüm 8

Platform Spesifik İpuçları

8.1 Codeforces

Codeforces, en popüler competitive programming platformlarından biridir.

İpucu

Codeforces İpuçları:

- Çoğu problemde multiple test case vardır
- Modular arithmetic sık kullanılır (MOD = 1e9 + 7)
- Interactive problemlerde flush() kullanmayı unutmayın
- Hacking phase'de başkalarının çözümlerini test edebilirsiniz

```cpp
#include <bits/stdc++.h>
using namespace std;

#define ll long long
#define MOD 1000000007

void solve() {
    // Her test case i in     zm     buraya
}

int main() {
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);

    int t;
    cin >> t;

    while (t--) {
        solve();
    }

    return 0;
}
```

Listing 8.1: Codeforces Template

8.2 AtCoder

```cpp
// AtCoder'da  s  k  kullan lan  modular  values
const int MOD = 998244353;  // AtCoder' n   sevdi i  mod
```

```
const int MOD2 = 1000000007;  // Klasik mod

// AtCoder Beginner Contest template
#include <bits/stdc++.h>
using namespace std;

int main() {
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);

    // Tek test case genellikle

    return 0;
}
```

Listing 8.2: AtCoder Specific Constants

8.3 Google Code Jam

```
#include <bits/stdc++.h>
using namespace std;

void solve(int test_case) {
    //     zm     burada

    cout << "Case #" << test_case << ": " << answer << endl;
}

int main() {
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);

    int T;
    cin >> T;

    for (int test = 1; test <= T; test++) {
        solve(test);
    }

    return 0;
}
```

Listing 8.3: Google Code Jam Format

Bölüm 9

Sık Kullanılan Patterns

9.1 Frequency Counter

```
// Karakter frekans
unordered_map<char, int> charFreq;
for (char c : str) {
    charFreq[c]++;
}

// Array element frekans
map<int, int> freq;
for (int x : arr) {
    freq[x]++;
}

// En sık geçen element
int mostFrequent = max_element(freq.begin(), freq.end(),
    [](const pair<int,int>& a, const pair<int,int>& b) {
        return a.second < b.second;
    })->first;
```

Listing 9.1: Frequency Counting Patterns

9.2 Prefix Sum

```
// 1D prefix sum
vector<ll> prefix(n + 1, 0);
for (int i = 0; i < n; i++) {
    prefix[i + 1] = prefix[i] + arr[i];
}

// Range sum query [l, r] (0-indexed)
ll rangeSum(int l, int r) {
    return prefix[r + 1] - prefix[l];
}

// 2D prefix sum
vector<vector<ll>> prefix2D(n + 1, vector<ll>(m + 1, 0));
for (int i = 1; i <= n; i++) {
    for (int j = 1; j <= m; j++) {
        prefix2D[i][j] = matrix[i-1][j-1] +
                         prefix2D[i-1][j] +
                         prefix2D[i][j-1] -
                         prefix2D[i-1][j-1];
    }
}

```

```
// 2D range sum query
ll rangeSum2D(int r1, int c1, int r2, int c2) {
    return prefix2D[r2+1][c2+1] - prefix2D[r1][c2+1] -
           prefix2D[r2+1][c1] + prefix2D[r1][c1];
}
```

Listing 9.2: Prefix Sum Patterns

9.3 Difference Array

```
// Range update i in difference array
class DifferenceArray {
private:
    vector<ll> diff;
    int n;

public:
    DifferenceArray(vector<int>& arr) {
        n = arr.size();
        diff.resize(n + 1, 0);

        // Initialize difference array
        for (int i = 0; i < n; i++) {
            diff[i] = arr[i] - (i > 0 ? arr[i-1] : 0);
        }
    }

    // Range update: add val to [l, r]
    void rangeUpdate(int l, int r, int val) {
        diff[l] += val;
        if (r + 1 < n) {
            diff[r + 1] -= val;
        }
    }

    // Get final array after all updates
    vector<ll> getFinalArray() {
        vector<ll> result(n);
        result[0] = diff[0];

        for (int i = 1; i < n; i++) {
            result[i] = result[i-1] + diff[i];
        }

        return result;
    }
};
```

Listing 9.3: Difference Array for Range Updates

Bölüm 10

Karmaşıklık Analizi

10.1 Zaman Karmaşıklığı Tablosu

Algoritma	**Zaman Karmaşıklığı**	**Alan Karmaşıklığı**	**Kullanım Alanı**
Linear Search	O(n)	O(1)	Unsorted array arama
Binary Search	O(log n)	O(1)	Sorted array arama
Merge Sort	O(n log n)	O(n)	Stable sorting
Quick Sort	O(n log n) avg	O(log n)	In-place sorting
Heap Sort	O(n log n)	O(1)	Priority queue
Counting Sort	O(n + k)	O(k)	Integer sorting
DFS/BFS	O(V + E)	O(V)	Graph traversal
Dijkstra	O(V log V + E)	O(V)	Shortest path
Bellman-Ford	O(VE)	O(V)	Negative weights
Floyd-Warshall	$O(V^3)$	$O(V^2)$	All pairs shortest
Union-Find	O((n))	O(n)	Disjoint sets
Segment Tree	O(log n)	O(n)	Range queries
Fenwick Tree	O(log n)	O(n)	Range sum queries
KMP	O(n + m)	O(m)	String matching

10.2 Karmaşıklık Hesaplama

Algoritma

Karmaşıklık hesaplarken dikkat edilecek noktalar:

- İç içe döngülerde çarpım kuralı
- Recursive fonksiyonlarda master theorem
- Amortized analysis durumları
- Cache efficiency ve constant factors

Bölüm 11

Son İpuçları ve Kaynaklar

11.1 Yapılacaklar ve Yapılmayacaklar

Yapılacaklar	Yapılmayacaklar
Her problemde edge case'leri kontrol et	Gereksiz optimizasyonlara takılma
Modular arithmetic kullanırken overflow'a dikkat et	Zor problemde çok fazla zaman harcama
Template kodunu ezberle	Template olmadan yarışmaya girme
Hızlı typing pratiği yap	Test etmeden submit etme
Farklı algoritmaları kombinle	Panic yapma!
Debugging araçlarını kullan	Soru okumadan kod yazmaya başlama
Düzenli pratik yap	Sadece kolay problemler çözme

11.2 Pratik Kaynakları

11.2.1 Online Platformlar

- **Codeforces**: En aktif community ve düzenli contest'ler
- **AtCoder**: Kaliteli problemler ve educational content
- **LeetCode**: Interview preparation ve algorithm practice
- **HackerRank**: Çeşitli zorluk seviyeleri
- **SPOJ**: Klasik algoritmalar için
- **TopCoder**: SRM'ler ve marathon matches
- **CodeChef**: Long contest'ler

11.2.2 Kaynak Kitaplar

- **"Competitive Programming"** by Steven Halim
- **"Algorithm Design Manual"** by Steven Skiena
- **"Introduction to Algorithms"** by CLRS
- **"Programming Pearls"** by Jon Bentley
- **"Concrete Mathematics"** by Graham, Knuth, Patashnik

11.2.3 YouTube Kanalları

- **Errichto**: Problem solving techniques
- **Colin Galen**: Contest strategies
- **William Fiset**: Data structures and algorithms
- **Algorithms Live!**: Live problem solving

11.3 Gelişim Yol Haritası

Strateji

Beginner to Expert Road Map:

1. **Beginner (0-6 ay):** STL, basic algorithms, greedy
2. **Intermediate (6-12 ay):** DP, graphs, number theory
3. **Advanced (1-2 yıl):** Segment trees, advanced DP, geometry
4. **Expert (2+ yıl):** Flow networks, string algorithms, advanced math

Sonuç

Bu kılavuz, competitive programming dünyasında başarılı olmak için gereken temel ve ileri seviye teknikleri kapsamlı bir şekilde ele almaktadır. Unutmayın ki, teorik bilgi tek başına yeterli değildir - sürekli pratik yaparak bu teknikleri ustalaştırmanız gerekmektedir.

Her algoritma ve teknik için düzenli olarak problemler çözün, farklı yaklaşımları deneyin ve hatalarınızdan öğrenin. Competitive programming bir maraton değil, sprint'tir - hızlı düşünme, doğru algoritma seçimi ve hatasız implementation başarının anahtarıdır.

Başarılar dileriz!