# 信息学竞赛常用优化技巧

高效算法与代码优化指南

By 奇思妙学

# 目录

1. 时间复杂度分析
2. 空间复杂度优化
3. 输入输出优化
4. 数据结构优化
5. 算法策略优化
6. 数学优化技巧
7. 实战案例分析

# 时间复杂度分析

算法效率的核心指标

# 大O表示法理解

```cpp
// O(1) - 常数时间复杂度
int getFirst(vector<int>& arr) {
    return arr[0];
}

// O(n) - 线性时间复杂度
int sumArray(vector<int>& arr) {
    int sum = 0;
    for (int num : arr) {
        sum += num;
    }
    return sum;
}
```

**关键点**：关注最坏情况下的增长趋势

# 常见时间复杂度对比

| 复杂度 | 数据规模 n=10^6 | 可行性 |
| --- | --- | --- |
| O(1) | 任意 | 优秀 |
| O(log n) | 任意 | 优秀 |
| O(n) | 10^6 | 良好 |
| O(n log n) | 10^6 | 可接受 |
| $O(n^2)$ | 10^3 | 临界 |
| $O(2^n)$ | 20 | 不可行 |

# 空间复杂度优化

内存使用的艺术

# 内存优化策略

```cpp
// 优化前：O(n) 额外空间
vector<int> temp(n);
for (int i = 0; i < n; i++) {
    temp[i] = arr[n - i - 1];
}

// 优化后：O(1) 额外空间
for (int i = 0; i < n / 2; i++) {
    swap(arr[i], arr[n - i - 1]);
}
```

**技巧**：原地操作、重复利用空间

# 数据结构内存考量

- **vector** vs **array**：动态 vs 静态
- **unordered_map**：哈希表开销
- **bitset**：位级压缩
- **链式前向星**：图的紧凑存储

```
// 位集优化布尔数组
bitset<1000000> visited;
// 仅占用 125KB，而非 1MB
```

# 输入输出优化

竞赛中的速度瓶颈突破

# C++ 输入输出加速

```cpp
#include <ios>
#include <iostream>

// 关闭同步，显著加速
ios::sync_with_stdio(false);
cin.tie(nullptr);
cout.tie(nullptr);

// 使用 '\n' 而非 endl
cout << "Hello" << '\n';

// 手动解析数字（极端优化）
int read() {
    int x = 0;
    char c = getchar();
    while (c < '0' || c > '9') c = getchar();
    while (c >= '0' && c <= '9') {
        x = x * 10 + c - '0';
        c = getchar();
    }
    return x;
}
```

# 数据结构优化

选择合适的数据结构

# STL 容器选择指南

| 操作需求 | 推荐容器 | 时间复杂度 |
| --- | --- | --- |
| 随机访问 | vector | O(1) |
| 频繁插入删除 | list | O(1) |
| 键值查询 | unordered_map | O(1)平均 |
| 有序遍历 | map | O(log n) |
| 优先级队列 | priority_queue | O(log n) |

# 自定义数据结构优化

```cpp
// 并查集路径压缩
struct DSU {
    vector<int> parent;
    int find(int x) {
        return parent[x] == x ? x : parent[x] = find(parent[x]);
    }
};

// 单调队列优化滑动窗口
deque<int> dq;
for (int i = 0; i < n; i++) {
    while (!dq.empty() && arr[dq.back()] >= arr[i]) {
        dq.pop_back();
    }
    dq.push_back(i);
}
```

# 算法策略优化

思维层面的优化

# 分治与剪枝

```
// 二分答案框架
int binarySearch(int left, int right) {
    while (left <= right) {
        int mid = left + (right - left) / 2;
        if (check(mid)) {
            left = mid + 1;
        } else {
            right = mid - 1;
        }
    }
    return right;
}

// DFS 剪枝示例
void dfs(int depth, int current) {
    if (current >= best) return; // 最优性剪枝
    if (depth == n) {
        best = min(best, current);
        return;
    }
    // ... 递归搜索
}
```

# 动态规划优化

| 优化技术 | 适用场景 | 效果 |
| --- | --- | --- |
| 滚动数组 | 状态只依赖前几个 | 降维 |
| 斜率优化 | 决策单调性 | $O(n^2) \rightarrow O(n)$ |
| 四边形不等式 | 区间DP | 降低枚举范围 |
| 状态压缩 | 状态可二进制表示 | 减少状态数 |

# 数学优化技巧

数值计算的智慧

# 数论与组合优化

```cpp
// 快速幂模运算
long long pow_mod(long long a, long long b, long long mod) {
    long long res = 1;
    while (b) {
        if (b & 1) res = res * a % mod;
        a = a * a % mod;
        b >>= 1;
    }
    return res;
}

// 质数筛法优化
vector<bool> is_prime(n + 1, true);
for (int i = 2; i * i <= n; i++) {
    if (is_prime[i]) {
        for (int j = i * i; j <= n; j += i) {
            is_prime[j] = false;
        }
    }
}
```

# 位运算技巧

```
// 常用位操作
x & (x - 1); // 清除最低位的1
x & -x;      // 获取最低位的1
(x + 7) & ~7; // 向上对齐到8的倍数

// 状态压缩DP
for (int mask = 0; mask < (1 << n); mask++) {
    for (int j = 0; j < n; j++) {
        if (!(mask >> j & 1)) {
            dp[mask | (1 << j)] = min(
                dp[mask | (1 << j)],
                dp[mask] + cost[j]
            );
        }
    }
}
```

# 实战案例分析

## 综合运用优化技巧

# 案例：最大子段和问题

**问题**：给定数组，求连续子数组的最大和

```
// 暴力解法 O(n³)
int maxSum = INT_MIN;
for (int i = 0; i < n; i++) {
    for (int j = i; j < n; j++) {
        int sum = 0;
        for (int k = i; k <= j; k++) {
            sum += arr[k];
        }
        maxSum = max(maxSum, sum);
    }
}

// Kadane算法 O(n)
int maxSum = arr[0], current = arr[0];
for (int i = 1; i < n; i++) {
    current = max(arr[i], current + arr[i]);
```

# 优化思维总结

1. **分析瓶颈**：时间/空间复杂度分析
2. **选择结构**：合适的数据结构
3. **改进算法**：更优的算法策略
4. **代码级别**：输入输出、循环等优化
5. **数学工具**：数论、组合数学应用
6. **常数优化**：位运算、缓存友好

# 进阶学习资源

- 《算法导论》- 理论基础
- 《挑战程序设计竞赛》- 实战指南
- OI Wiki - 最新竞赛知识
- Codeforces, AtCoder - 练习平台

# Q & A

问题与讨论