搜索与剪枝

冉雨杭

2023 年 8 月 5 日

搜索与剪枝

- 在搜索中，可以通过一些技巧来加快程序速度
- 这类方法往往比较玄学，很难准确说出会对速度有多大提升，很多时候更看经验
- 在比赛中，一般正解都是直接能够直接通过的，但会一些优化方法能够给你更多的选择！

P1562 N 皇后

- 给定一个 $N \times N$ 的国际象棋棋盘，有些位置可以放皇后，有多少种放 N 个不能互相攻击的皇后方案
- $1 \leq N \leq 14$

N 皇后

- 之前我们讲过普通的 N 皇后问题
- 假设用 p_i 表示第 i 个皇后放的位置
- 对于任意 i, j
  - $p_i \neq p_j$
  - $i - p_i \neq j - p_j$
  - $i + p_i \neq j + p_j$

N 皇后

```
void dfs(int u) {
    if(u == n + 1) {
        res++;
        return;
    }
    for (int i = 1; i <= n; i++) {
        if(!a[i] && !b[i + u] && !c[i + n - u] && can_vis[u][i]) {
            a[i] = b[i + u] = c[i + n - u] = 1;
            dfs(u + 1);
            a[i] = b[i + u] = c[i + n - u] = 0;
        }
    }
}
```

N 皇后

- 这个会超时通过不了
- 考虑每次需要枚举一下当前这一行放在哪个位置，越到后面可行的空间越少，每次枚举会有 $O(n)$ 的复杂度
- 可以通过一些技巧来保证每次枚举到的都是可行的方案

N 皇后

- 可以把哪些列能放进行状态压缩，用一个整数来表示这些信息
- 每次只需要找到下一个二进制为 1 的位置进行枚举即可
- 可以用 lowbit 快速找二进制下一个 1 的位置！

N 皇后

```
int lowbit(int x) {
    return x & (-x)
}

void dfs(int u, int cmask, int diag1_mask, int diag2_mask) {
    if(u == n + 1) {
        res++;
        return;
    }
    int mask = cmask & diag1_mask & diag2_mask;
    while(mask) {
        int x = lowbit(mask);
        mask -= x;
        if(!can_vis[u][lg[x]]) continue;
        dfs(u + 1, cmask - x, (diag1_mask - x << 1) + 1, (diag2_mask - x >> 1) + (1 << n - 1))
    }
}
```

N 皇后

- 可以发现这样我们每次枚举到的除非不能走，不然都是一种可行的方案
- 这样成功加速了程序，最后可以通过

分组

- 给定 n 个人，每个人有一个价值 a_i
- 需要把这 n 个人分成两组，同组内两人 x, y 之间带来的价值是 $a_x \times a_y$
- 求最大的划分价值
- $n \leq 25$

分组

- 先二进制枚举划分方案，然后每种方案 $O(n)$ 计算答案
- 复杂度 $O(n2^n)$
- 会超时

```
int res = 0;
for (int mask = 0; mask < (1 << n); mask++) {
    int s[2] = {0, 0};
    int sum = 0;
    for (int i = 0; i < n; i++) {
        res += s[mask >> i & 1] * a[i];
        s[mask >> i & 1] += a[i];
    }
    res = max(res, sum);
}
```

分组

- 可以通过 dfs 改变枚举顺序来降低复杂度
- 复杂度 $O(2^n)$

```
void dfs(int u, int s1, int s2, int sum) {
    if(u == 1) {
        res = max(res, sum);
        return;
    }
    dfs(u + 1, s1, s2 + a[u], sum + a[u] * s2);
    dfs(u + 1, s1 + a[u], s2, sum + a[u] * s1);
}
```

剪枝

- 剪枝通过通过增加一些判断条件来减少枚举空间
- 常用的几个剪枝技巧
  - 最优性剪枝
  - 可行性剪枝

P1120 小木棍

- 乔治有一些同样长的小木棍，他把这些木棍随意砍成几段，直到每段的长都不超过 50。
- 现在，他想把小木棍拼接成原来的样子，但是却忘记了自己开始时有多少根木棍和它们的长度。
- 给出现在小木棍的数量 n 以及每段小木棍的长度 a_i，找出原始木棍的最小可能长度。
- $1 \leq n \leq 65, 1 \leq a_i \leq 50$

P1120 小木棍

- 原来木棍长度一定是总和的因子，所以只需要枚举因子长度
- 剩下的问题就是想办法检验某个长度是否能被合成出来
- 因子从大到小枚举，这样只要我们找到答案就可以输出
- 对于每一个因子，我们只能搜索来验证是否可行

P1120 小木棍

- 将木棍大小从大到小排序，这样越到后面枚举的东西越少（尽量让不合法的情况靠前）
- 如果剩下木棍最短长度大于当前剩余长度直接返回
- 枚举一个木棍失败后下一个木棍不能和目前木棍长度相同
- 加上这些优化即可通过

折半搜索

- 折半搜索顾名思义就是将搜索空间折半
- 先处理其中一半，并用一些数据结构维护这些信息
- 再处理另一半，并用之前的数据结构快速统计答案

P4799 世界冰球锦标赛

- Bobek 有 M 元钱，共有 N 场比赛，每场比赛的门票都有一个价格 C_i
- 问在总票价不超过 M 元钱的情况下，Bobek 共有多少种不同的观赛方案
- $N \leq 40, M \leq 10^{18}, C_i \leq 10^{17}$

P4799 世界冰球锦标赛

- 直接二进制枚举
- 复杂度 $O(2^n)$
- 复杂度太高

P4799 世界冰球锦标赛

- 先二进制枚举前一半，然后将答案存入 vector 中
- 对 vector 进行排序
- 再枚举后一半，每次查询可以直接在 vector 上二分
- 复杂度 $O(2^{\frac{n}{2}}n)$

P3067 Balanced Cow Subsets G

- 给定一个长度为 n 的数组 $a_1, a_2 \cdots, a_n$
- 现在要将数组分成三部分，满足第一部分的和等于第三部分的和
- 问划分的方案数是多少
- $n \leq 20, a_i \leq 10^8$

网格路径

- 有一个 $n \times m$ 的网格，每个格子都有一个数字
- 现在要从 $(1, 1)$ 走到 (n, m)。每次只能往右走或者往下走
- 问有多少种走法，使得路径的数字和不超过 M
- $n, m \leq 20, a_{i,j} \leq 10^8, M \leq 10^9$

网格路径

- 直接搜索的复杂度是 $O(2^{n+m})$
- 复杂度太高，怎么优化？

网格路径

- 考虑折半搜索，先搜前一半的结果，把和塞入终点对应的 vector 里
- 每个 vector 里排序
- 再倒着搜后一半的结果，在终点对应的 vector 里去二分查找满足条件的个数
- 复杂度 $O((n+m)2^{\frac{n+m}{2}})$

P3067 Balanced Cow Subsets G

- 直接三进制枚举
- 复杂度 $O(3^n)$
- 会 TLE，想办法优化

P3067 Balanced Cow Subsets G

- 对前一半元素进行三进制枚举，然后用一个 map 记录和的方案数
- 对后一半元素进行三进制枚举，然后配合 map 统计答案
- 复杂度 $O(3^{\frac{n}{2}}n)$

记忆化搜索

- 记忆化搜索严格意义上不算搜索，是一种动态规划
- 记忆化搜索用一个数组来记录结果，这样每个状态不会重复搜索
- 记忆化搜索的好处是不需要你自己考虑状态枚举的顺序

递推

- $f(n) = f(\frac{n}{2}) + f(\frac{n+1}{2}) + f(\frac{n}{2}) * f(\frac{n+1}{2})$
- $f(1) = 1$
- **求** $f(n) \mod 998244353$
- $n \leq 10^{18}$

递推

- 直接递推显而易见的会超时
- 好像涉及的位置不会太多，我们可以把它们找出来再递推

递推

- 直接递推显而易见的会超时
- 好像涉及的位置不会太多，我们可以把它们找出来再递推
- 更好的方法是直接记忆化搜索！

递推

```
i64 solve(i64 n) {
    if(dp.count(n)) {
        return dp[n];
    }
    if (n == 1) {
        return 1;
    }
    return dp[n] = (solve(n / 2) + solve((n + 1) / 2) + 1ll * solve(n / 2) * solve((n + 1) / 2)
         ) % mod;
}
```

纸牌游戏

- 给定 n 张牌，玩家 A 和玩家 B 依次拿走每张纸牌（可以看见所有的牌），规定玩家 A 先拿，玩家 B 后拿。但是每个玩家每次只能拿走最左或最右的纸牌
- 最后的分数是 A 的卡牌分数和 - B 的卡牌分数和
- $n \leq 2000$

纸牌游戏

- 直接搜索应该怎么写？

```
int dfs(int l, int r) {
    if(l == r) return a[l];
    int res = max(a[l] - dfs(l + 1, r), a[r] - dfs(l, r - 1));
    return res;
}
```

- 复杂度是 $O(2^n)$

纸牌游戏

- 加两行，记忆化搜索

```
int dfs(int l, int r) {
    if (vis[l][r]) return dp[l][r];
    vis[l][r] = 1;
    if(l == r) return a[l];
    int res = max(a[l] - dfs(l + 1, r), a[r] - dfs(l, r - 1));
    return dp[l][r] = res;
}
```

- 复杂度是 $O(n^2)$

有向无环图的最长链

- 给定一个有向无环图，每个点有一个权值
- 定义一条链的价值为链上所有点的权值和
- 求最长链长度
- $n \leq 5 \times 10^5$

有向无环图的最长链

- 给定一个有向无环图，每个点有一个权值
- 定义一条链的价值为链上所有点的权值和
- 求最长链长度
- $n \leq 5 \times 10^5$

有向无环图的最长链

- 可以反向建图，然后用拓扑排序，在拓扑排序的时候顺便计算
- 也可以使用记忆化搜索，更简单

```
int dfs(int u) {
    if(vis[u]) return dp[u];
    vis[u] = 1;
    int res = 0;
    for (auto v : adj[u]) {
        res = max(res, dfs(v) + a[u]);
    }
    return dp[u] = res;
}
```

HDU 1078

- 在一个 $n \times m$ 二维平面上，每一个格子都有一些奶酪，一只老鼠从 (1,1) 出发，每次只能在同一正交方向上行走，至多走 k 步
- 老鼠吃了奶酪会变胖，为了有足够的能量继续行走，它每次都需要吃比上一次多的奶酪，即只能走向那些数字比原本格子数字大的格子。
- 给出二维平面和每个格子的奶酪数，问老鼠最多能吃多少奶酪。
- $n, m, k \leq 100$

HDU 1078

- 用 $dp[x][y]$ 表示从位置 (x, y) 开始走到终点，最多能吃到多少奶酪
- 答案是 $dp[1][1]$

HDU 1078

```
int dfs(int x, int y) {
    if(vis[x][y]) return dp[x][y];
    vis[x][y] = 1;
    dp[x][y] = num[x][y];
    for(int i = 1; i <= n; i++) {
        if(num[x][i] > num[x][y] && abs(i - y) <= k) {
            dp[x][y] = max(dp[x][y], dfs(x, i) + num[x][y]);
        }
        if(num[i][y] > num[x][y] && abs(i - x) <= k) {
            dp[x][y] = max(dp[x][y], dfs(i, y) + num[x][y]);
        }
    }
    return dp[x][y];
}
```

题目列表

- P1443
- P1135
- P1433
- P1605
- P1101
- P2404
- P1379
- P1078
- P3067

谢谢!