2022 蓝桥杯 B 组省赛题解

Kenshin2438

2023 年 1 月 16 日

目录

前言

题目地址: http://oj.daimayuan.top/course/18/problems

代码仅供参考（已通过上述 OJ 测试）

1 九进制转十进制

简单做一个进制转换即可，

$$2 \times 9^3 + 0 \times 9^2 + 2 \times 9^1 + 2 \times 9^0 = 1478$$

2 顺子日期

官方的**顺子**未定义，存在歧义。按照代码源上的注释，应当是 14 天。分别为，

$$2022/01/2* \quad 2022/10/12 \quad 2022/11/23 \quad 2022/12/3*$$

3 刷题统计

一周一共能做 $5a + 2b$ 题，容易知道需要的整周数目。除却整周的部分，最后一周里判断是在前 5 天还是后 2 天做完即可。

```cpp
#include <iostream>
using namespace std;

int main() {
  cin.tie(nullptr)->sync_with_stdio(false);
  long long a, b, n;
  cin >> a >> b >> n;
  long long ans = n / (5 * a + 2 * b) * 7;
  n %= (5 * a + 2 * b);

  if (n >= 5 * a) {
    ans += 5;
    n -= 5 * a;
    ans += (n + b - 1) / b;
  } else {
    ans += (n + a - 1) / a; // 上取整的一种写法
  }

  cout << ans << '\n';
  return 0;
}
```

4　修剪灌木

先简单模拟一下过程：每棵灌木，被修剪时有两种可能——从左往右和从右往左。那么，实际上就有两种可能的高度 $2 \times (n - i)$ 和 $2 \times (i - 1)$。

```cpp
#include <iostream>
using namespace std;

int main() {
  cin.tie(nullptr)->sync_with_stdio(false);
  int n; cin >> n;
  for (int i = 1; i <= n; i++) {
    cout << max(2 * (n - i), 2 * (i - 1)) << '\n';
  }
  return 0;
}
```

5　X 进制减法

由于题目保证 $A \geq B$，那么自然是每个数位上的进制越小，两者的差值越小。注意细节：最小的进制为 2 进制。

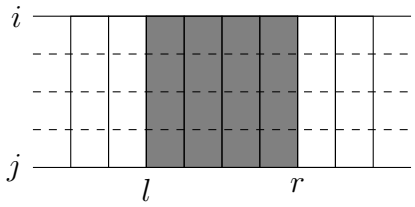
```cpp
#include <iostream>
#include <vector>
using namespace std;

const int mod = 1000000007;

int main() {
  cin.tie(nullptr)->sync_with_stdio(false);
  int N; cin >> N;
  int na; cin >> na;
  vector<int> A(na);
  for (int i = na - 1; i >= 0; i--) {
    cin >> A[i];
  }
  int nb; cin >> nb;
  vector<int> B(max(na, nb));
  for (int i = nb - 1; i >= 0; i--) {
    cin >> B[i];
  }
  long long ans = 0, pre = 1;
  for (int i = 0; i < na; i++) {
    int base = max(2, max(A[i], B[i]) + 1);
    ans = (ans + (A[i] - B[i]) * pre % mod + mod) % mod;
    pre = pre * base % mod;
  }
  cout << ans << '\n';
  return 0;
}
```

6 统计子矩阵

枚举子矩形的上下两侧的位置，不妨令子矩阵夹在第 i, j 行之间。然后则需要统计，有多少对不同的 $l, r (l \leq r)$ 使得子矩阵位于 l, r 列之间的时候总和小于 k ，显然这个问题被压缩到了一维——如下图，等价为找到满足求和不大于 k 的不同连续区间的个数。



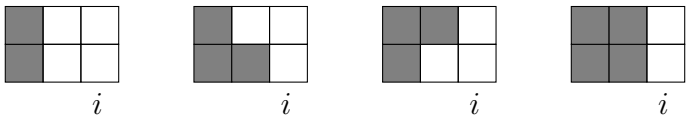
枚举 i, j 两行的时间复杂度为 $\mathcal{O}(n^2)$ ，如果此时再枚举 l, r 则在时间复杂度上 $\mathcal{O}(n^2m^2)$ 无法通过（仍可拿下 70% 的分）。该问题是**滑动窗口**的经典题，最后一步仅需要 $\mathcal{O}(m)$ 即可，详细实现见代码，练习可以去LeetCode。

```cpp
#include <iostream>
#include <vector>
using namespace std;

int main() {
  cin.tie(nullptr)->sync_with_stdio(false);
  int n, m, k; cin >> n >> m >> k;
  vector<vector<int>> grid(n, vector<int>(m));
  for (int i = 0; i < n; i++)
    for (int j = 0; j < m; j++)
      cin >> grid[i][j];
  for (int j = 0; j < m; j++) {
    for (int i = 1; i < n; i++) {
      grid[i][j] += grid[i - 1][j];
    } // 各列的前缀和，后续会使用
  }
  long long ans = 0;
  for (int i = 0; i < n; i++) {
    for (int j = i; j < n; j++) {
      vector<int> val(m);
      for (int k = 0; k < m; k++) {
        val[k] = grid[j][k] - (i ? grid[i - 1][k] : 0);
      } // 通过上面的前缀和优化
      for (int l = 0, r = 0, S = 0; r < m; r++) {
        S += val[r];
        while (S > k && l <= r) S -= val[l], l += 1;
        ans += max(0, r - l + 1);
      }
    }
  }
  cout << ans << '\n';
  return 0;
}
```

7 积木画

很明显，本题用到的算法为**动态规划**。由于使用的方块最多只能影响到 2 列，假设当前处理到第 i 列，并且前 $i-2$ 列均被覆盖，则第 $i-1$ 列上的方块有 4 中可能的状态。



上述四种状态，从左至右分别定义为 $0,1,2,3$，以 `dp[i][s]` 表示处理完前 i 列且状态为 s 的方案数。则可知答案为 `dp[n][3]` 。

```
#include <iostream>
#include <vector>
using namespace std;

const int mod = 1000000007;

int main() {
  cin.tie(nullptr)->sync_with_stdio(false);
  int n; cin >> n;
  if (n == 1) return cout << 1 << '\n', 0;
  vector<long long> dp = {1, 1, 1, 2};
  for (int i = 3; i <= n; i++) {
    vector<long long> ndp(4);
    ndp[0] = dp[3];
    ndp[1] = (dp[0] + dp[2]) % mod;
    ndp[2] = (dp[0] + dp[1]) % mod;
    ndp[3] = (dp[0] + dp[1] + dp[2] + dp[3]) % mod;
    dp = move(ndp);
  }
  cout << dp[3] << '\n';
  return 0;
}
```

8 扫雷

由于爆炸半径均不超过 10，可以直接建图，此时的边数目为 $\mathcal{O}(nR^2)$，其后使用 BFS 或者 DFS 遍历即可。同时，由于坐标范围比较大，这里需要 Hash 来储存（使用 `std::unordered_map` 实现，常数较大）。

```
#include <iostream>
#include <vector>
#include <unordered_map>
using namespace std;

const int N = 5e4 + 10;
```

```cpp
struct Bomb {
  long long x, y;
  int r, cnt;
} Bomb[N];

unordered_map<long long, int> idx;
inline long long myHash(long long x, long long y) {
  return x * 1e9 + y;
} // std::pair<long long, long long> 没有Hash

int ans = 0;
inline void dfs(long long X, long long Y, int R) {
  for (long long x = X - R; x <= X + R; x++) {
    if (x < 0 || x > 1e9) continue;
    for (long long y = Y - R; y <= Y + R; y++) {
      if (y < 0 || y > 1e9) continue;
      if (idx.find(myHash(x, y)) != idx.end()) {
        int nex = idx[myHash(x, y)];
        long long dx = Bomb[nex].x - X;
        long long dy = Bomb[nex].y - Y;
        long long dis2 = dx * dx + dy * dy;
        if (dis2 > R * 1LL * R) continue;
        if (Bomb[nex].cnt != 0) {
          ans += Bomb[nex].cnt;
          Bomb[nex].cnt = 0;
          dfs(Bomb[nex].x, Bomb[nex].y, Bomb[nex].r);
        }
      }
    }
  }
}

int main() {
  cin.tie(nullptr)->sync_with_stdio(false);
  int n, m; cin >> n >> m;
  int num = 0;
  for (int i = 0; i < n; i++) {
    long long x, y; int r;
    cin >> x >> y >> r;
    if (idx.find(myHash(x ,y)) == idx.end()) {
      idx[myHash(x, y)] = ++num;
      Bomb[num].x = x;
      Bomb[num].y = y;
      Bomb[num].r = r;
      Bomb[num].cnt = 1;
    } else { // 同一中心的只需要知道最大爆炸半径
      int id = idx[myHash(x, y)];
      Bomb[id].r = max(Bomb[id].r, r);
      Bomb[id].cnt += 1;
    }
  }
  while (m--) {
    long long X, Y; int R;
    cin >> X >> Y >> R;
```

```
    dfs(X, Y, R);
  }
  cout << ans << '\n';
  return 0;
}
```

9 李白打酒加强版

显然也是**动态规划**统计方案数，但本题的状态很多（剩余的酒的数目）。容易知道，酒的数目一定要小于等于剩下的花的数目，这样才能满足最终把酒喝完，以此可以排除许多的非法状态。

以 dp[i][j][k] 表示已经遇到了 i 次店，j 次花，当前还有 k 斗酒的方案数。那么最终答案就为 dp[n][m-1][1]。

```
#include <iostream>
#include <vector>
using namespace std;

const int mod = 1000000007;

int main() {
  cin.tie(nullptr)->sync_with_stdio(false);
  int n, m; cin >> n >> m;
  vector<vector<vector<int>>> dp(
    n + 1, vector<vector<int>>(m + 1, vector<int>(m + 1))
  );
  for (int i = 0; i <= n; i++) {
    for (int j = 0; j <= m; j++) {
      if (i == 0 && j == 0) {
        dp[0][0][2] = 1;
        continue;
      }
      for (int k = 0; k <= m; k++) {
        if (i > 0 && k % 2 == 0)
          dp[i][j][k] = (dp[i][j][k] + dp[i-1][j][k/2]) % mod;
        if (j > 0 && k + 1 <= m)
          dp[i][j][k] = (dp[i][j][k] + dp[i][j-1][k+1]) % mod;
      }
    }
  }
  cout << dp[n][m-1][1] << '\n';
  return 0;
}
```

10 砍竹子

容易发现，操作 $H \to \lfloor\sqrt{\lfloor\frac{H}{2}\rfloor+1}\rfloor$ 会使得 H 快速收敛至 1 。我们以这样一种思路来考虑：

假设已经砍了 $i-1$ 棵竹子，当前是第 i 颗。首先，当前的竹子一定会经过上述的操作到达 1 。这个过程中可能会出现 *当前竹子的高度与前一颗竹子在某一时刻的高度一致*的情况，一旦出现则表明此后的步骤可以同前一颗竹子一起进行。使用二分、`std::set` 或 `std::map` 等均可通过此题。

```
#include <algorithm>
#include <iostream>
#include <vector>
#include <cmath>
using namespace std;

int main() {
  cin.tie(nullptr)->sync_with_stdio(false);
  int n; cin >> n;
  int ans = 0;

  vector<long long> pre;
  for (int i = 0; i < n; i++) {
    long long H; cin >> H;
    vector<long long> now;
    for (bool has = false; H != 1; ) {
      now.push_back(H);
      has |= binary_search(pre.rbegin(), pre.rend(), H);
      ans += (has == false);
      H = sqrt(H / 2 + 1);
    }
    pre = move(now);
  }

  cout << ans << '\n';
  return 0;
}
```