奶牛的代码聚会【8.5】

这次的题目比上次的来说

确实是难了一些～

不过思想呀做题思路呀还是比较明显的啦

主要难在你有了思路之后怎么下手呢～

对细节和一些技巧方面有了一些要求哦

A# P5831: Cow Gymnastics B

```
/*
    做题思路：瞄一眼数据范围，很小，然后就可以暴力枚举啦，三重循环甚至四重循环都行，签到题一般都不会超时的
    时间复杂度：O(KN^3)
*/

#include <iostream>
#include <algorithm>
#define ll long long
using namespace std;

int main() {
    int i,j,k,l,n,m,ans=0;
    cin >> n >> m;
    int a[n][m];
    for (i=0; i<n; i++)
        for (j=0; j<m; j++)
            cin >> a[i][j];
    for (i=0; i<m; i++) { // 双重循环枚举第一行每两头奶牛
        for (j=i+1; j<m; j++) {
            for (k=1; k<n; k++) { // 枚举其他行
                for (l=0; l<m; l++) { // 枚举其他行的位置
                    if (a[k][l] == a[0][i]) break; // 如果先出现第一头牛，则正常～
                    else if (a[k][l] == a[0][j]) goto th; // 如果先出现第二头牛，则不一致
                }
            }
            ans++;
            th:;
        }
    }
    cout << ans << endl;
}
```

B# P6207: Cows on Skates G

```
/*
    做题思路：这道题和迷宫差不多，因此也可以用 DFS+回溯 或者 BFS 来做。不过有个小小的坑点，如果用 DFS+回溯的话，只需要回溯搜索的路径，访问过的点不需要回溯 visit[i] = 0，因为我们只需要输出任意一条路径，因此 DFS 到了终点就可以停止深搜。如果回溯代码执行的话，说明回溯的那些点不是通往终点的路，因此不需要 visit[i] = 0，如果回溯的话还会导致产生无用的搜索然后 TLE 一个点（引出专业名词：剪枝）
    so 此题用 BFS 写的话可能就简单粗暴一点（什么时候适合深搜什么时候适合广搜，可以自己思考对比一下，当然也可以和姐姐交流～）
    时间复杂度：均为O(rc)
*/

// DFS
#include <iostream>
#include <algorithm>
#include <vector>
#define ll long long
using namespace std;

int n,m,j,dx[4] = {1,-1,0,0},dy[4] = {0,0,1,-1}; // 四个方向
string s[113]; // 地图
vector<int> v;
void dfs(int x,int y) {
    if (x == n-1 and y == m-1) {
        for (int i=0; i<v.size(); i+=2) cout << v[i]+1 << " " << v[i+1]+1 <<
endl;
        cout << x+1 << " " << y+1 << endl; // 输出路径
        j = 1; // 结束深搜
    }
    s[x][y] = '*';
    v.push_back(x); // 记录路径
    v.push_back(y);
    for (int i=0; i<4; i++) {
        if (j == 0 and x+dx[i] >= 0 and x+dx[i] < n and y+dy[i] >= 0 and
y+dy[i] < m and s[x+dx[i]][y+dy[i]] == '.')
            dfs(x+dx[i],y+dy[i]);
    }
    // s[x][y] = '.'; // 加这个回溯的话，会多出很多无用的搜索然后 TLE
    v.pop_back(); // 回溯路径
    v.pop_back();
}

int main() {
    int i,j;
    cin >> n >> m;
    for (i=0; i<n; i++) cin >> s[i];
```

```
    dfs(0,0);
}

// BFS
#include <iostream>
#include <algorithm>
#include <vector>
#include <queue>
#define ll long long
using namespace std;

int main() {
    int i,j,r,c,x,y;
    int dx[4] = {1,0,-1,0},dy[4] = {0,1,0,-1};
    cin >> r >> c;
    string s[r];
    for (i=0; i<r; i++) cin >> s[i];
    queue<int> q1,q2; // x, y坐标
    queue<vector<int>> q3; // 路径
    q1.push(0);
    q2.push(0);
    q3.push(vector<int>(2,0));
    while (q1.size()) {
        x = q1.front();
        y = q2.front();
        if (x == r-1 and y == c-1) break; // 到终点啦~
        q1.pop();
        q2.pop();
        for (i=0; i<4; i++) {
            if (x+dx[i] >= 0 and x+dx[i] < r and y+dy[i] >= 0 and y+dy[i] < c
and s[x+dx[i]][y+dy[i]] == '.') {
                q1.push(x+dx[i]);
                q2.push(y+dy[i]);
                s[x+dx[i]][y+dy[i]] = '*';
                vector<int> v = q3.front();
                v.push_back(x+dx[i]);
                v.push_back(y+dy[i]);
                q3.push(v); // 写路径
            }
        }
        q3.pop(); // 注意最后再pop之前的路径
    }
    for (i=0; i<q3.front().size(); i+=2)
        cout << q3.front()[i]+1 << " " << q3.front()[i+1]+1 << endl;
}
```

C# P4379: Lemonade Line S

```
/*
    做题思路：非常简单的贪心，降序排序即可，然后比较 a[i] 和 i，就知道这头奶牛愿不愿意排队。可以证明的是，当某一头奶牛不想排队了，它之后的奶牛都不想排队
    时间复杂度：O(NlogN)
*/

#include <iostream>
#include <algorithm>
#define ll long long
using namespace std;

int main() {
    int i,j,n,ans=0;
    cin >> n;
    int a[n];
    for (i=0; i<n; i++) cin >> a[i];
    sort(a,a+n,greater<int>()); // 降序
    for (i=0; i<n; i++)
        if (a[i] >= i) ans++;
        // else break; // 加入这句话会快，不加也无所谓
    cout << ans << endl;
}
```

D# P1460: 健康的荷斯坦奶牛 Healthy Holsteins

```
/*
    做题思路：DFS+回溯，这道题就很不适合 BFS 了。常见的深搜板子，如果当前的维他命够吃了就停止当前深搜并更新最优值，如果不够吃就继续深搜饲料
    时间复杂度：O(v*2^g)
*/

#include <iostream>
#include <algorithm>
#define ll long long
using namespace std;

int v,g,ans = 99999999;
int a[15][25],b[25],c[25];
string ans2; // string当数组
void dfs(int x,int num,string s) {
    for (int i=x; i<g; i++) {
        int n = 0;
        for (int j=0; j<v; j++) c[j] += a[i][j]; // 加饲料
        for (int j=0; j<v; j++) {
            if (c[j] < b[j]) { // 还存在不够的
                n = 1; // 标志位
```

```
                dfs(i+1,num+1,s+(char)(i+1)); // 继续深搜
                break;
            }
        }
        if (n == 0 and ans > num) { // 已经够吃了，更新最优值
            ans = num;
            ans2 = s+(char)(i+1);
        }
        for (int j=0; j<v; j++) c[j] -= a[i][j]; // 回溯
    }
}

int main() {
    int i,j;
    cin >> v;
    for (i=0; i<v; i++) cin >> b[i];
    cin >> g;
    for (i=0; i<g; i++)
        for (j=0; j<v; j++)
            cin >> a[i][j];
    dfs(0,1,"");
    cout << ans;
    for (i=0; i<ans2.length(); i++) cout << " " << (int)ans2[i];
}
```

E# P1209: 修理牛棚 Barn Repair

```
/*
    做题思路：贪心，转换题意为：m块板->一块很长的板再做m-1次分割，每次分割掉间隔最大的空隙就可以啦
    时间复杂度：O(nlogn)
*/

#include <iostream>
#include <algorithm>
#define ll long long
using namespace std;

int main() {
    int i,j,m,s,c,ans;
    cin >> m >> s >> c;
    int a[c],b[c-1];
    for (i=0; i<c; i++) cin >> a[i];
    sort(a,a+c); // 输入不一定是有序的
    for (i=1; i<c; i++) b[i-1] = a[i]-a[i-1]-1; // 两头奶牛之间的间隙
    sort(b,b+c-1,greater<int>()); // 降序
    ans = a[c-1]-a[0]+1;
```

```
    for (i=0; i<min(m-1,c-1); i++) ans -= b[i]; // 最大提供木板数可能会大过奶牛数
    cout << ans << endl;
}
```

F# P6183: The Rock Game S

```
/*
    做题思路：还是深搜，其实一般能用深搜就不用广搜。深搜变化状态，记录路径，如果之前没有这种变化状态就可以继续深搜，最后搜到满足的情况后结束所有深搜。
    特殊方式：用一个数代表一种状态，类似之前的状态压缩，原来的'O'当作二进制位0，'X'当作二进制位1，因此可以把某种状态用一个int类型的数来表示，而状态的变换就可以直接用异或来执行。
    时间复杂度：O(N*2^N)
*/

#include <iostream>
#include <algorithm>
#include <set>
#define ll long long
using namespace std;

int n,ans,a[1<<15],c;
set<int> s;
void dfs(int num) { // num：深搜次数
    s.insert(ans); // set可以在O(logn)时间复杂度内判断之前有没有这种变化状态，当然你用vector记录然后遍历也是可以滴
    a[num] = ans;
    if (num == (1<<n)) { // 深搜次数够啦：输出结果
        for (int j=0; j<=(1<<n); j++) {
            for (int k=0; k<n; k++) cout << ((a[j]/(1<<k)%2) == 0 ? 'O' : 'X');
            cout << endl;
        }
        c = 1; // 结束所有深搜
        return;
    }
    for (int i=0; i<n; i++) { // 对每一个位进行变换
        ans ^= (1<<i); // 异或，只改变一个位置
        if (s.count(ans) == 0 or (num == (1<<n)-1 and ans == 0)) { // count函数 == 0：之前没有这种状态；num == (1<<n)-1 and ans == 0：最末尾变回全'O'
            if (c == 0) dfs(num+1);
        }
        ans ^= (1<<i); // 回溯
    }
    s.erase(ans); // 回溯
}

int main() {
```

```
    int i,j;
    cin >> n;
    dfs(0);
}
```

G# P3074: Milk Scheduling S

```
/*
    做题思路：高级的 BFS：拓扑排序，略微超纲，但其实数据结构课有讲过拓扑排序：构造有向图，如果x要在y之前挤完奶，则构造有向边x->y。然后对图跑一次拓扑排序即可
    时间复杂度：O(M)
*/

#include <iostream>
#include <algorithm>
#include <vector>
#include <queue>
#define ll long long
using namespace std;

int a[10005],start[10005],in[10005],ans; // a：挤奶时间；start：排队时间；in：有向图节点的入度
int main() {
    int i,j,n,m,x,y,t;
    cin >> n >> m;
    vector<int> v[n+1]; // vector保存图联通关系
    for (i=1; i<=n; i++) cin >> a[i];
    for (i=0; i<m; i++) {
        cin >> x >> y;
        v[x].push_back(y); // 存图
        in[y]++; // 入度+1
    }
    queue<int> q;
    for (i=1; i<=n; i++)
        if (in[i] == 0) // 入度为0的点，即可以直接开始挤奶的奶牛
            q.push(i);
    while (q.size()) { // 跑拓扑排序啦
        t = q.front();
        q.pop();
        for (i=0; i<v[t].size(); i++) { // 看看那些奶牛需要在它后面
            in[v[t][i]]--;
            start[v[t][i]] = max(start[v[t][i]],start[t]+a[t]); // 更新排队时间
            if (in[v[t][i]] == 0) q.push(v[t][i]); // 入度为0后，即得到了它的排队时间，可以开始挤奶啦～
        }
    }
    for (i=1; i<=n; i++) ans = max(ans,start[i]+a[i]); // 找到最久的那只奶牛
```

```
    cout << ans << endl;
}
```