新内容

读取空格

四、搜索

4. 记忆化搜索

经典的滑雪 https://www.luogu.com.cn/problem/P1434

R*C的格子，只能从当前格子去比它矮的格子。起点任选，问最长能滑几步。

朴素的搜索：

枚举每个顶点作为起点 搜最大深度

记忆化搜索：

```
#include <bits/stdc++.h>
using namespace std;
int dx[4] = {0,0,1,-1},dy[4] = {1,-1,0,0};
int n,m,a[201][201],s[201][201] = {0},ans;
bool use[201][201];
int dfs(int x, int y)
{
    if(s[x][y]) return s[x][y]; // s[x][y]表示以[x][y]为起点  最大深度是多少

    s[x][y] = 1;
    for(int i=0;i<4;i++)
    {
        int xx = x+dx[i];
        int yy = y+dy[i];
        if(xx>=1&&yy>=1&&xx<=n&&yy<=m&&a[x][y]>a[xx][yy])//合法的移动
        {
            dfs(xx,yy); //
            // s[xx][yy]
            s[x][y] = max(s[x][y],s[xx][yy]+1); //更新s[x][y]答案
        }
    }
    return s[x][y];
}
int main(void)
{
    cin >> n >> m;
    for(int i=1;i<=n;i++)
        for(int j=1;j<=m;j++)
            cin >> a[i][j];
    for(int i=1;i<=n;i++) // 枚举每个格子为起点
        for(int j=1;j<=m;j++)
            ans = max(ans,dfs(i,j)); //
    cout << ans;
    return 0;
}
```

dfs(int now) --> x,y res[x] [y]++

牛 https://www.luogu.com.cn/problem/P1535

N*M的格子，牛起初在（R1,C1), T秒后在（R2,C2)。地图上有障碍物。问可能的移动方式有多少种

T 次移动

DFS 的写法：

从起点往4个方向搜，深度为T，搜到最后停在终点就加答案

记忆化搜索的写法：

```
#include<iostream>
#include<cstring>

using namespace std;

const int N=110;

char g[N][N];
int n,m,t;
int r1,c1,r2,c2;
int f[N][N][N]; //f[x][y][t]表示以(x,y)为起点,在时间为t时可以到达(r2,c2)的路线数
// f[r1][c1][T]

int dx[]={0,1,0,-1};
int dy[]={1,0,-1,0};

int dfs(int x,int y,int t){ // f[x][y][t]

    if(f[x][y][t]!=-1)  return f[x][y][t];// 已经搜索过这个状态 直接用当时的结果

    //
    if(t==0){//注意关注t在f数组中的意义 t=0时
        if(x==r2&&y==c2)    return f[x][y][t]=1; // 已经在终点就=1
        return f[x][y][t]=0; // 不在终点就=0
    }

    int mx,my;
    int tmp=0; //
    for(int i=0;i<4;i++){ // 正常dfs 注意判断障碍物
        mx=x+dx[i];
        my=y+dy[i];
        if(mx>=0&&mx<n&&my>=0&&my<m&&g[mx][my]!='*')
            tmp+=dfs(mx,my,t-1);
    }
    return f[x][y][t]=tmp;
}

int main(){
    cin>>n>>m>>t;
    memset(f,-1,sizeof f); // -1
    for(int i=0;i<n;i++)
```

```
        for(int j=0;j<m;j++)
            cin>>g[i][j]; // 地图
    cin>>r1>>c1>>r2>>c2;  // 起点终点
    // 这个代码用的0下标
    r2--;c2--;
    cout<<dfs(r1-1,c1-1,t); // f[r1-1][c1-1][t]
}
```

五、动态规划（DP）

1. 线性DP

过河卒 https://www.luogu.com.cn/problem/P1002

卒从（0，0）到（n,m）。中间有马的坐标，该马所在的点和所有跳跃一步可达的点不能走。问一共多少种方法?

```
#include <bits/stdc++.h>
using namespace std;
int n,m,x_m,y_m;
long long dp[30][30] = {0}; // dp[i][j] = dp[i-1][j] + dp[i][j-1];
int horse[30][30] = {0};
int main(void)
{
    cin >> n >> m >> x_m >> y_m;
    // 偏移2位，相当于把原点移到[2][2]
    // 方便后续预处理马控制的位置，不用判断是否出界
    n+=2;
    m+=2;
    x_m+=2;
    y_m+=2;
    // 马控制的9个位置
    horse[x_m][y_m] = 1;//0
    horse[x_m+1][y_m+2] = 1;//1
    horse[x_m+2][y_m+1] = 1;//2
    horse[x_m+1][y_m-2] = 1;//3
    horse[x_m+2][y_m-1] = 1;//4
    horse[x_m-1][y_m+2] = 1;//5
    horse[x_m-2][y_m+1] = 1;//6
    horse[x_m-1][y_m-2] = 1;//7
    horse[x_m-2][y_m-1] = 1;//8
    // 初始化 到原本位置的方案数为1
    dp[2][2] = 1;
    // 对于其他所有位置
    // 从上到下更新行，从左到右更新列
    for(int i=2;i<=n;i++)
        for(int j=2;j<=m;j++)
        {
        // dp[i][j]
            // 马控制的位置不更新 dp[i][j]永远为0
            if(horse[i][j]) continue; // dp[2][4] = 0;
            // dp[2][5] = max(dp[1][5]+dp[2][4], dp[i][j]);
            // 否则 可以从上一行 / 左一行跳过来
```

```
            dp[i][j] = max(dp[i-1][j] + dp[i][j-1],dp[i][j]);
        }
    cout << dp[n][m] << endl;
    return 0;
}
```

牛吃草 https://www.luogu.com.cn/problem/P1868

n个区间的牧草，随便选但不能有重复。求最多能吃到的牧草数量。

之前一个问题 相当于每堆牧草占一个区间，但贡献都是1 ，所以可以排序右端点贪心求解

```
#include <bits/stdc++.h>
using namespace std;
vector<int>v[3000010];
int n,mx,dp[3000010]; // dp[i]
// mx 代表最大的 y
// dp[i] 表示坐标上前i个草地 最多能吃到多少堆牧草
int main(){
    cin >> n;
    for(int i=1;i<=n;i++){
        int x,y;
        cin >> x >> y;
        // v[y]: 所有以y为右端点的区间的左端点
        v[y].push_back(x-1); // 选择区间[x,y] 不影响 x-1 之前的部分
        mx=max(mx,y); // dp[mx]
    }
    for(int i=1;i<=mx;i++){
        dp[i]=dp[i-1];//至少能吃到前i-1个格子能吃到的
        for(int j=0;j<v[i].size();j++){ // v[i];
        // 到y这个位置，新增可选的区间为所有的 [last,y]
            int last=v[i][j]; // [last,y] dp[last] + i-last
            dp[i]=max(dp[i],dp[last]+i-last);// 取最优的
        }
    }
    cout << dp[mx];
    return 0;
}
```

dp[i]: 选择第i个人的所有情况中 最长的上升子序列长度是多少

1

j in [1 i-1]

dp[j] i a[j] < a[i]

dp[i] = max(dp[i], dp[j]+1);

合唱队形 https://www.luogu.com.cn/problem/P1091

n个同学，分别给定身高。需要去掉一些同学，使剩下同学身高满足严格先升后降，允许整个序列严格上升 / 严格下降。求最少去掉几个同学?

问题类似于求最长上升/下降序列

```
#include <bits/stdc++.h>
```

```
using namespace std;
int n;
int a[111] = {0};
int f[2][111] = {0}; // f[0][i]
// f[0][i] 用于存放从前往后看 以i结尾的最长上升子序列长度
// f[1][i] 用于存放从后往前看 以i结尾的最长下降子序列长度
int main(void)
{
    cin >> n;
    for(int i=1;i<=n;i++) cin >> a[i];
    a[0] = 0;
    for(int i=1;i<=n;i++)
        for(int j=0;j<i;j++)
            if(a[i]>a[j]) f[0][i] = max(f[0][i],f[0][j]+1);


    a[n+1] = 0;
    // i ..... n+1
    for(int i=n;i>=1;i--)
        for(int j=n+1;j>i;j--)
            if(a[i]>a[j]) f[1][i] = max(f[1][i],f[1][j]+1);

    int res = 0;
    // 枚举中心点，两段加起来
    for(int i=1;i<=n;i++) res = max(f[0][i]+f[1][i]-1,res);
    cout << n-res; // 剩下的人为需要去掉的人数
    return 0;
}
```

导弹拦截 https://www.luogu.com.cn/problem/P1020

输入若干个整数，是导弹依次飞来的高度

有一个拦截系统，可以拦截任意高度的导弹，但每次拦截都不能高于前一次的高度

即：每次拦截的导弹是一个单调递减子序列。

第一问：一套系统最多拦截多少个导弹?

第二问：如果要拦截所有导弹，至少需要多少套系统?

先考虑NOIP原题的数据，允许O(n^2) 时间复杂度做法：

```
#include <bits/stdc++.h>
using namespace std;
#define int long long
int a[100010] = {0};
int dp[100010] = {0}; // dp[i]:i是最后一个 可以拦截多少个
int n = 0;
int tail[100010] = {0}; // 维护一些序列的末尾 最优的做法是接在末尾最小的合法序列后面
signed main(void)
{
    int temp, idx = 0;
    while(cin >> temp){
        idx++;
```

```
        a[idx] = temp;
    }
    n = idx;
    // 1.
    for(int i=1;i<=n;i++){
        dp[i] = 1; // 它自己
        for(int j=1;j<i;j++){
            if(a[j] >= a[i]) dp[i] = max(dp[i], dp[j]+1);
        }
    }
    int res1 = 0;
    for(int i=1;i<=n;i++) res1 = max(res1, dp[i]);
    cout << res1 << '\n';

    // 2.
    int cnt = 0;// 套数
    for(int i=1;i<=n;i++){
        int now = a[i];
        int find = 0;

        for(int j=1;j<=cnt;j++){
            if(tail[j] >= now){ // end 数组维护成单调上升的
                find = 1;
                tail[j] = now;
                break;
            }
        }
        if(!find){
            cnt++;
            tail[cnt] = now;
        }
    }
    int res2 = cnt;
    cout << res2;
    return 0;
}
```

然后先考虑第二问O(nlogn) 时间复杂度的做法：

```
#include <bits/stdc++.h>
using namespace std;
#define int long long
int a[100010] = {0};
int dp[100010] = {0}; // dp[i]:i是最后一个 可以拦截多少个
int n = 0;
int tail[100010] = {0};
signed main(void)
{
    int temp, idx = 0;
    while(cin >> temp){
        idx++;
        a[idx] = temp;
    }
    n = idx;
    // 1.
```

```
    for(int i=1;i<=n;i++){
        dp[i] = 1; // 它自己
        for(int j=1;j<i;j++){
            if(a[j] >= a[i]) dp[i] = max(dp[i], dp[j]+1);
        }
    }
    int res1 = 0;
    for(int i=1;i<=n;i++) res1 = max(res1, dp[i]);
    cout << res1 << '\n';

    // 2.
    int cnt = 0;
    for(int i=1;i<=n;i++){
        int now = a[i];
        // int find = 0;
        // for(int j=1;j<=cnt;j++){
        //     if(tail[j] >= now){ // end 数组维护成单调上升的
        //         find = 1;
        //         tail[j] = now;
        //         break;
        //     }
        // }
        // if(!find){
        //     cnt++;
        //     tail[cnt] = now;
        // }
        int low = 1, high = cnt; // tail[i]: val
        while(low < high){
            int mid = (low+high)/2;
            if(tail[mid] >= now) high = mid;
            else low = mid+1;
        }
        //  tail[high] >=now --> now
        if(tail[high] >= now){
            tail[high] = now;
        }
        else{
            cnt++;
            tail[cnt] = now;
        }
    }
    int res2 = cnt;
    cout << res2;
    return 0;
}
```

最后看第一问的O(nlogn)解法：

Dilworth 定理（记住就可以）

把序列分成不上升子序列的最少个数，等于序列的最长上升子序列长度。

把序列分成不降子序列的最少个数，等于序列的最长下降子序列长度。

第二问：不上升子序列的最少个数 ---> 最长上升子序列长度

第一问：最长不上升子序列的长度 ---> 不下降子序列的最少个数

所以第一问其实是第二问反过来的问题 二分条件改一下即可

```
#include <bits/stdc++.h>
using namespace std;
#define int long long
int a[100010] = {0};
int dp[100010] = {0}; // dp[i]:i是最后一个 可以拦截多少个
int n = 0;
int tail[100010] = {0};
signed main(void)
{
    int temp, idx = 0;
    while(cin >> temp){
        idx++;
        a[idx] = temp;
    }
    n = idx;
    // 1.
    int cnt = 0;
    tail[0] = 2e9;
    for(int i=1;i<=n;i++){
        int now = a[i];
        if(now <= tail[cnt]){ // 没有二分中寻找的情况
            cnt++;
            tail[cnt] = now;
        }
        int low = 1, high = cnt;
        while(low < high){ // 维护末尾高度下降的子序列
            int mid = (low+high)/2;
            if(tail[mid] < now) high = mid;
            else low = mid+1;
        }
        if(tail[high] < now){
            tail[high] = now;
        }
    }
    int res1 = cnt;
    cout << res1 << '\n';

    // 2.
    cnt = 0;
    tail[0] = -2e9;
    for(int i=1;i<=n;i++){
        int now = a[i];
        // int find = 0;
        // for(int j=1;j<=cnt;j++){
        //     if(tail[j] >= now){ // end 数组维护成单调上升的
        //         find = 1;
        //         tail[j] = now;
        //         break;
        //     }
        // }
        // if(!find){
        //     cnt++;
        //     tail[cnt] = now;
```

```
        // }
        int low = 1, high = cnt;
        while(low < high){
            int mid = (low+high)/2;
            if(tail[mid] >= now) high = mid;
            else low = mid+1;
        }
        if(tail[high] >= now){
            tail[high] = now;
        }
        else{
            cnt++;
            tail[cnt] = now;
        }
    }
    int res2 = cnt;
    cout << res2;
    return 0;
}
```