20210821习题题解

A.数的划分

课堂例题，代码课上已经给出，略

B.function

课堂例题，给出了核心代码，现给出AC代码，代码截取自某同学，也有同学写了更简洁的版本：

```
#include <bits/stdc++.h>
using namespace std;
long long f[21][21][21], INF = 1e9;
long long w(long long a, long long b, long long c)
{
    if (a < 1 || b < 1 || c < 1)
        return 1;
    if (a > 20 || b > 20 || c > 20) {
        if (f[20][20][20] == -INF)
            f[20][20][20] = w(20, 20, 20);
        return f[20][20][20];
    }
    if (a < b && b < c) {
        if (f[a][b][c - 1] == -INF)
            f[a][b][c - 1] = w(a, b, c - 1);
        if (f[a][b - 1][c - 1] == -INF)
            f[a][b - 1][c - 1] = w(a, b - 1, c - 1);
        if (f[a][b - 1][c] == -INF)
            f[a][b - 1][c] = w(a, b - 1, c);
        return f[a][b][c - 1] + f[a][b - 1][c - 1] - f[a][b - 1][c];
    }
    if (f[a - 1][b][c] == -INF)
        f[a - 1][b][c] = w(a - 1, b, c);
    if (f[a - 1][b - 1][c] == -INF)
        f[a - 1][b - 1][c] = w(a - 1, b - 1, c);
    if (f[a - 1][b][c - 1] == -INF)
        f[a - 1][b][c - 1] = w(a - 1, b, c - 1);
    if (f[a - 1][b - 1][c - 1] == -INF)
        f[a - 1][b - 1][c - 1] = w(a - 1, b - 1, c - 1);
    return f[a - 1][b][c] + f[a - 1][b - 1][c] + f[a - 1][b][c - 1] - f[a -
1][b - 1][c - 1];
}
int main()
{
    long long a, b, c;
    for (int i = 0; i < 21; i++)
        for (int j = 0; j < 21; j++)
            for (int k = 0; k < 21; k++)
                f[i][j][k] = -INF;
    cin >> a >> b >> c;
    while (a != -1 || b != -1 || c != -1) {
        cout << "w(" << a << ", " << b << ",  " << c << ") = " << w(a, b, c)
<< endl;
```

```
        cin >> a >> b >> c;
    }
    return 0;
}
```

C.逃离迷宫

课堂例题，核心代码与正常的网格类题目搜索一致，加上奇偶性剪枝即可通过。

```
#include <bits/stdc++.h>
using namespace std;
const int N = 60;
char mp[N][N];
bool vis[N][N];
int dx[4] = { -1, 0, 1, 0 };
int dy[4] = { 0, -1, 0, 1 };
bool ok;
int n, m, T;
void dfs(int x, int y, int t)
{
    if (ok) {
        return;
    }
    if (t == T) {
        if (mp[x][y] == 'D') {
            ok = true;
        }
        return;
    }
    vis[x][y] = true;
    for (int i = 0; i < 4; ++i) {
        int tx = x + dx[i];
        int ty = y + dy[i];
        if (tx < 0 || tx >= n || ty < 0 || ty >= m || mp[x][y] == 'X' ||
vis[tx][ty]) {
            continue;
        }
        dfs(tx, ty, t + 1);
    }
    vis[x][y] = false;
}
int main()
{

    cin >> n >> m >> T;
    for (int i = 0; i < n; ++i) {
        cin >> mp[i];
    }
    int sx, sy, ex, ey; //sx，sy表示起点坐标，ex，ey表示终点坐标
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < m; ++j) {
            if (mp[i][j] == 'S') {
                sx = i, sy = j;
            }
```

```
            if (mp[i][j] == 'D') {
                ex = i, ey = j;
            }
        }
    }
    //奇偶性剪枝
    if ((sx + sy + ex + ey + T) % 2 != 0) {
        cout << "NO" << endl;
    } else {
        ok = false;
        dfs(sx, sy, 0);
        if (ok) {
            cout << "YES" << endl;
        } else {
            cout << "NO" << endl;
        }
    }
    return 0;
}
```

D.数字三角形

一道比较经典，多解的问题，除搜索还有其他解法。

主要搜索两个方向，搜索函数dfs(x,y,sum)，x表示行，y表示列，sum表示当前的和。

往左下搜索：$dfs(x+1,y,sum+a[x+1][y]);$

往右下搜索：$dfs(x+1,y+1,sum+a[x+1][y+1]);$

边界条件：$x>n$

参考代码：

```
#include <bits/stdc++.h>
using namespace std;
int a[25][25], n, ans = -1000000;
void dfs(int x, int y, int sum) //x表示行，y表示列，sum表示当前的和
{
    if (x == n + 1) {
        ans = max(ans, sum);
        return;
    }
    dfs(x + 1, y ,sum + a[x + 1][y]);  //两个方向搜索
    dfs(x + 1, y + 1 ,sum + a[x + 1][y + 1]);
    return;
}
int main()
{
    cin >> n;
    for (int i = 1; i <= n; i++)
        for (int j = 1; j <= i; j++)
            cin >> a[i][j];
    dfs(1,  1 , a[1][1]);
    cout << ans;
    return 0;
```

```
}
```

E.滑雪

按题目要求对四个方向进行搜索，移动时对移动的步数用$f[i][j]$进行记录，保存最优解，体现一个记忆化搜索的思想。

参考代码：

```
#include <bits/stdc++.h>
using namespace std;
const int N = 105;
int X[5] = { 0, 0, 0, 1, -1 };
int Y[5] = { 0, 1, -1, 0, 0 };
int map[N][N], f[N][N];
int n, m, ans = -1000;
int dfs(int x, int y)
{
    if(x>n||x<1||y>m||y<1) return ;
    if (f[x][y] == 1) {
        for (int i = 1; i <= 4; i++) {
            int dx = x + X[i];
            int dy = y + Y[i];
            if (map[dx][dy] < map[x][y]) {  //记录当前最优的步数，记忆化
                f[x][y] = max(f[x][y], dfs(dx, dy) + 1);
            }
        }
    }
    return f[x][y];
}
int main()
{
    cin >> n >> m;
    for (int i = 1; i <= n; i++)
        for (int j = 1; j <= m; j++)
            cin >> map[i][j], f[i][j] = 1;
    for (int i = 1; i <= n; i++)
        for (int j = 1; j <= m; j++)
            ans = max(ans, dfs(i, j));
    cout << ans;
    return 0;
}
```

F.棋盘

这道题目还是比较有意思的，在我们熟悉的棋盘搜索的基础上，加上一个可以使用魔法这样一个条件。

题目如果没有魔法可以使用，那么我们的搜索函数dfs就是这样：dfs(x,y,nowcoin)，表示当前的位置(x,y)，以及当前花费的金币

有了魔法，我们的状态就需要知道是否使用魔法，所以需要再加入一个参数mflag，表示是否使用魔法。

最终的函数dfs(x,y,nowcoin,mflag)

为了方便判断，搜索前规定一下颜色，0代表无色 1代表红色 2代表黄色

搜索时无非两种情况：

1、下一个格子无色，如果魔法未使用，那就使用魔法过去

2、下一个格子有色，如果颜色相同，直接过去；否则，就花钱过去

参考代码：

```
#include <bits/stdc++.h>
using namespace std;
const int N = 110;
int X[5] = { 0, 0, 0, 1, -1 };
int Y[5] = { 0, 1, -1, 0, 0 };
int n, m, ans = 2147482333;
int f[N][N]; //存储 当前坐标(i,j)的最优值
int map[N][N];
void mofa(int x, int y, int nowcoin, bool mflag)
{
    if (x > m || x < 1 || y > m || y < 1)
        return;
    if (map[x][y] == 0)
        return;
    if (nowcoin >= f[x][y])
        return; //最优剪枝
    f[x][y] = nowcoin;
    if (x == m && y == m) {
        ans = min(nowcoin, ans);
        return;
    }
    for (int i = 1; i <= 4; i++) {
        int dx = x + X[i];
        int dy = y + Y[i];
        if (map[dx][dy] != 0) { //如果下个格子不是无色
            if (map[dx][dy] == map[x][y])
                mofa(dx, dy, nowcoin, 0); //同色就过去
            else
                mofa(dx, dy, nowcoin + 1, 0); //不同色花金币过去
        } else if (map[dx][dy] == 0 && !mflag) { //如果下个格子是无色但没用魔法
            map[dx][dy] = map[x][y];
            mofa(dx, dy, nowcoin + 2, 1); //无色变有色，过去
            map[dx][dy] = 0;
        }
    }
}
int main()
{
    cin >> m >> n;
    for (int i = 1; i <= n; i++) {
        int x, y, c;
        scanf("%d %d %d", &x, &y, &c);
        map[x][y] = c + 1; //0无色 1红色 2黄色
    }
```

```
    memset(f, 0x7f, sizeof(f));
    mofa(1, 1, 0, 0);
    if (ans != 2147482333)
        cout << ans << endl;
    else
        cout << -1 << endl;
    return 0;
}
```

G.小木棍

根据课上分析的几种剪枝，选择3种左右效率高一点的剪枝即可，具体选择的剪枝条件请看代码。

具体参考代码注释，参考代码：

```
#include<bits/stdc++.h>
using namespace std;
const int maxn = 65;
int used[maxn], stick[maxn];
int n, sum, num, l;
bool cmp(int a, int b)
{
    return a > b;
}
bool dfs(int s, int le, int pos) //s表示已经拼好了几根木棒，le表示将要用来拼接的小木条长度，pos表示当前搜索的位置（下标）
{
    if (s == num)
        return true;
    int sign = le == 0 ? 1 : 0; //记录是否没有木条了
    for (int i = pos + 1; i < n; i++) {
        if (used[i])
            continue; //如果该木条被用过则跳过
        if (le + stick[i] == l) //如果刚好能够拼接成一根木棒进行下一根木棒的拼接
        {
            used[i] = 1; //置为已使用
            if (dfs(s + 1, 0, -1)) //下一根木棒也能拼接成功说明l为答案
                return true;
            used[i] = 0; //如果本次搜索失败应还原used
            return false; //下一次不能够拼成则失败了
        } else if (le + stick[i] < l) //如果小于则接着往下搜
        {
            used[i] = 1;
            if (dfs(s, le + stick[i], i))
                return true;
            used[i] = 0;
            if (sign)
                return false; //表示没有木条可拼接了意味着失败了（不能直接return false,假如还有木条还可继续拼接）
            while (stick[i] == stick[i + 1])
                i++; //如果木条长度相同不必再搜索
        }
    }
    return false;
}
```

```cpp
int main()
{
    ios::sync_with_stdio(0);
    cin.tie(0);
    while (cin >> n && n) {
        sum = 0;
        for (int i = 0; i < n; i++) {
            cin >> stick[i];
            sum += stick[i];
        }
        sort(stick, stick + n, cmp); //把木条从大到小排列
        int ans;
        for (l = stick[0]; l <= sum; l++) {
            if (sum % l == 0) {
                memset(used, 0, sizeof (used)); //每种l都应该重置used
                num = sum / l; //该情况下应该有的木棒数
                if (dfs(0, 0, -1)) {
                    ans = l;
                    break;
                }
            }
        }
        cout << ans << endl;
    }
    return 0;
}

```

H.生日蛋糕

做这道题，你首先要有点数学常识：

首先圆柱的各个公式：
$V = \pi R^2 H$ 体积
$S_{侧} = 2\pi RH$ 侧面积
$S_{底} = \pi R^2$ 底面积

表面积=侧面积+底面积

这道题的难点主要在于dfs函数的设计，即搜索状态的寻找。

根据题目意思，涉及到体积、表面积、高度等搜索信息，为了准确转移搜索的信息和剪枝，这道题需要涉及五个参数

dfs(int sum,int v,int dep,int h,int r) ， sum 表面涂的面积， v是蛋糕体积， dep是第几层， h是高度， r是半径

除了dep是搜索的层次以外，其余的参数引入是为了辅助搜索和剪枝。

为了尽可能不搜索无效的表面积、体积、高度，主要有以下几种剪枝

- **搜索顺序上的剪枝**：从体积大的搜到体积小，每次的半径至少减一。（这样的话若是方案不合法在一个大的的时候就可以直接扔掉不要，但是如果是小的的话，可能就要试很多个。）

- **可行性剪枝1**：当 当前的体积+之后预测到的自己认为的最优的体积>n 的时候就可以直接舍弃这一个不合法的方案。（当前的加上最优的都已经是不合法的了，还能怎么办？）
- **可行性剪枝2**：上下界剪枝。根据多年的数学学习把半径和高度的冗余的状态全部都丢掉不要。
- **最优性剪枝1**：在dfs的过程中可能会有很多次搜索到不合法的方案，若是不合法的方案用到的面积刚好等于n的时候（也就是说给定的面积全部都用完了），那么最优解的ans一定小于此时的答案，这个时候在ban掉这个方案之前可以对答案进行进一步的更新；
- **最优性剪枝2:** 若 当前的面积+之后预测到的自己认为的最优的面积>ans 的时候就可以直接舍弃这一个不优秀的方案，因为他没有答案小。

由于以上的剪枝涉及到大量表面积、体积的计算，为了简化代码避免重复计算，这一部分可以先预处理到$minv[]$和$minb[]$，节约搜索时间。

```
#include<bits/stdc++.h>
using namespace std;
int minv[30],minb[30];  //由题目可知,每一层的半径与高度是逐渐递减的,且最小是少1
int n,m,ans;              //所以提前把每一层的体积与表面积算出来
void dfs(int sum,int v,int dep,int h,int r)  //sum 表面涂的面积 v是蛋糕体积 dep 是第几层 h是高度 r是半径
{
    if(v>n)return ;  //如果堆蛋糕的体积已经大于所给体积了,那么就无法再向上搭建了
    if(dep==0)    //如果搭建完成了且此时蛋糕刚好用完,判断此时的表面积是否是最优的
    {
  //  printf("%d %d\n",v,sum);
        if(v==n&&sum<ans)
        {
            ans=sum;
        }
        return;
    }
   //    如果现在的体积加上下一层所用的最小体积依然大于规定的体积,那就返回上一层
   //     如果现在所求的表面积加上下一层的依然大于之前所求的的结果,则返回
   //     如果剩余体积的侧面积加上之前的大于之前求得的最优结果,返回上一层.
    if(v+minv[dep-1]>n||sum+minb[dep-1]>ans|| (n-v)/r*2+sum>=ans)return;
    for(int i=r-1;i>=dep;i--)   //从大到小枚举,所求的结果一定是最优的
    {
        if(dep==m)sum=i*i;  //到了最后一层的时候要算上顶部
        int th=min(h-1,(n-v-minv[dep-1])/(i*i));  //找到这一层的最小高度，上下界剪枝
        for(int j=th;j>=dep;j--)
        {
            dfs(sum+2*i*j,v+i*i*j,dep-1,j,i);
        }
    }
    return;
}
int main()
{
    for(int i=1;i<21;i++) //预处理每一层的体积和表面积便于搜索
    {
        minv[i]=minv[i-1]+i*i*i;
        minb[i]=minb[i-1]+i*i*2;
    }
    while(~scanf("%d%d",&n,&m))
    {
        ans=0x3f3f3f3f;
```

```
        dfs(0,0,m,n+1,n+1);
        if(ans==0x3f3f3f3f)
            ans=0;
        printf("%d\n",ans);
    }
    return 0;
}
```