记忆化搜索

简介&思想

在进行搜索的时候如果对某一个状态在不同的搜索过程中会搜索多次并且得到的结果只与状态本身的参数有关而与搜索过程无关，我们就可以用数组将第一次搜索这个状态所得到的结果储存，从而减小了复杂度。

例题

引水入城

来源：洛谷 p1514

概述：NxM的网格，第一行的格子可以建蓄水厂，其他各自可以建造输水厂，建造输水厂的条件是某个格子存在高度更高的相邻格子；需要令最后一行每一格都有输水厂，判断是否能做到，如果能则求出最少需要的蓄水厂个数。

答案：进行dfs搜索每一个格子建造输水厂/蓄水厂之后可以覆盖的最后一行的左右范围，并用数组存储。将第一行每个格子能覆盖的范围根据左边从小到大排序，转成区间覆盖问题。

```
#include <algorithm>
#include <cstdio>
#include<map>
using namespace std;
int
n,m,i,j,h[505][505],valid[505],vist[505],past[505][505],dx[4]={-1,1,0,0},dy
[4]={0,0,-1,1};
struct unit{
    int l,r;
    bool operator <(const unit&others) const
    {
        return this->l<others.l;
    }
}edge[505],que[505],point[505][505];
void dfs(int x,int y,int z)
{
    if(past[y][x])
    {
        edge[z].l=min(edge[z].l,point[y][x].l);
        edge[z].r=max(edge[z].r,point[y][x].r);
        return;
    }
    past[y][x]=1;
    if(y==n)
    {
        valid[z]=1;
```

```
        vist[x]=1;
        edge[z].l=min(edge[z].l,x);
        edge[z].r=max(edge[z].r,x);
        point[y][x].l=point[y][x].r=x;
    }
    for(int i=0,x1,y1;i<4;i++)
    {
        x1=x+dx[i];
        y1=y+dy[i];
        if(x1&&x1<=m&&y1&&y1<=n&&h[y][x]>h[y1][x1])
        {
            dfs(x1,y1,z);
            point[y][x].l=min(point[y][x].l,point[y1][x1].l);
            point[y][x].r=max(point[y][x].r,point[y1][x1].r);
        }
    }
}
int main()
{
    scanf("%d%d",&n,&m);
    for(i=1;i<=n;i++)
        for(j=1;j<=m;j++)
        {
            point[i][j].l=505;
            scanf("%d",&h[i][j]);
        }
    for(int i=1;i<=m;i++)
        edge[i].l=505;
    for(i=1;i<=m;i++)
        if(h[1][i]>=h[1][i-1]&&h[1][i]>=h[1][i+1])
            dfs(i,1,i);
    int aim=0;
    for(i=1;i<=m;i++)
        if(!vist[i])
            aim++;
    if(aim>0)
        printf("0\n%d",aim);
    else
    {
        int tot=0;
        for(int i=1;i<=m;i++)
            if(valid[i])
                que[++tot]=edge[i];
        sort(que+1,que+tot+1);
        int pos=1,iter=1,next,ans=0;
        while(pos<=m)
        {
            next=iter;
            while(iter<=tot&&que[iter].l<=pos)
            {
                if(que[iter].r>que[next].r)
```

```
                    next=iter;
                iter++;
            }
            pos=que[next].r+1;
            ans++;
        }
        printf("1\n%d",ans);
    }
    return 0;
}
```

游荡的奶牛

来源：洛谷 p1535

概述：一片草地上有一只奶牛，在T时间内每秒都会水平或竖直移动一格，给出奶牛的初始位置和一个目标位置，求T秒内从初始位置移动到目标位置的方案数。

答案：进行dfs搜索每一个格子剩余一定时间到达目标位置的方案数，同时用数组存值之，加以剪枝减少时间。

```
#include<cstdio>
using namespace std;
int
n,m,t,r1,c1,r2,c2,f[105][105][20],vist[105][105][20],dr[4]={-1,1,0,0},dc[4]
={0,0,-1,1};
char lawn[105][105];
int dfs(int r,int c,int time)
{
    if(!vist[r][c][time])
    {
        vist[r][c][time]=1;
        if(!time||r2-r+c2-c>time) return 0;
        for(int i=0,r3,c3;i<4;i++)
        {
            r3=r+dr[i];
            c3=c+dc[i];
            if(r3&&r3<=n&&c3&&c3<=m&&lawn[r3][c3]=='.')
                f[r][c][time]+=dfs(r3,c3,time-1);
        }
    }
    return f[r][c][time];
}
int main()
{
    scanf("%d%d%d",&n,&m,&t);
    for(int i=1;i<=n;i++)
```

```
        scanf("%s",lawn[i]+1);
    scanf("%d%d%d%d",&r1,&c1,&r2,&c2);
    f[r2][c2][0]=vist[r2][c2][0]=1;
    printf("%d",dfs(r1,c1,t));
    return 0;
}
```

逛公园

来源：洛谷 p3959

概述：一张有向图，无自环和重边、负边，存在0边，求从起点到终点长度不超过最短路+K的方案数；可能存在无穷种方案，判断这种情况。

答案：进行dfs搜索从每个点距离起点长度为其最短路+i$(0\le i\ge K)$到终点总距离符合要求的方案数，用数组记录之，并加以剪枝；无穷种方案即存在某个点位于一条合法的途径中，这个点连出一条0环，判断的方法就是在dfs中加入拓扑排序的成分。

```
#include<cstdio>
#include<cstring>
#include<queue>
#include<vector>
#define inf 0x3f3f3f3f
using namespace std;
int n,m,k,p,T;
int index1[100005],index2[100005];
struct unit{int to,next,w;}edge1[200005],edge2[200005];
int dis1[100005],dis2[100005],valid[100005],D,vist[100005];
int f[100005][55],vis[100005][55];
struct unit2
{
    int x,d;
    bool operator < (const unit2 &x) const
    {
        return x.d<d;
    }
};
void init()
{
    memset(index1,0,sizeof(index1));
    memset(index2,0,sizeof(index2));
    memset(dis1,0x3f,sizeof(dis1));
    memset(dis2,0x3f,sizeof(dis2));
    memset(valid,0,sizeof(valid));
    memset(f,0,sizeof(f));
    memset(vis,0,sizeof(vis));
    for(int i=0;i<=k;i++)
        f[n][i]=1;/*
```

```
    printf("%d: ",n);
    for(int i=0;i<=k;i++)
        printf("%d ",f[n][i]);
    printf("\n");*/
}
void dijkstra()
{
    priority_queue<unit2>q1;
    dis1[1]=0;
    q1.push({1,0});
    int surplus=1,head;
    memset(vist,0,sizeof(vist));
    while(surplus)
    {
        head=q1.top().x;
        q1.pop();
        if(vist[head]) continue;
        vist[head]=1;
        surplus--;
        for(int i=index1[head],update,son;i;i=edge1[i].next)
        {
            update=dis1[head]+edge1[i].w;
            son=edge1[i].to;
            if(update<dis1[son])
            {
                surplus+=(dis1[son]==inf);
                dis1[son]=update;
                q1.push({son,update});
            }
        }
    }
    D=dis1[n]+k;
    priority_queue<unit2>q2;
    dis2[n]=0;
    q2.push({n,0});
    surplus=1;
    memset(vist,0,sizeof(vist));
    while(surplus)
    {
        head=q2.top().x;
        q2.pop();
        if(vist[head]) continue;
        surplus--;
        vist[head]=1;
        valid[head]=((dis1[head]+dis2[head])<=D);
        for(int i=index2[head],update,son;i;i=edge2[i].next)
        {
            update=dis2[head]+edge2[i].w;
            son=edge2[i].to;
            if(update<dis2[son])
            {
```

```
                surplus+=(dis2[son]==inf);
                dis2[son]=update;
                q2.push({son,dis2[son]});
            }
        }
    }
}
int dfs(int x,int extra)
{
    if(vis[x][extra]<0) return -1;
    if(!vis[x][extra])
    {
        vis[x][extra]=-1;
        for(int i=index1[x],son,ex,temp;i;i=edge1[i].next)
        {
            son=edge1[i].to;
            if(valid[son])
            {
                ex=dis1[x]+extra+edge1[i].w-dis1[son];
                if(ex<=k&&ex+dis1[son]+dis2[son]<=D)
                {
                    temp=dfs(son,ex);
                    if(temp<0) return -1;
                    f[x][extra]=(f[x][extra]+temp)%p;
                }
            }
        }
        vis[x][extra]=1;
    }
    //printf("%d %d %d\n",x,extra,f[x][extra]);
    return f[x][extra];
}
int main()
{
    scanf("%d",&T);
    while(T--)
    {
        scanf("%d%d%d%d",&n,&m,&k,&p);
        init();
        for(int i=1,a,b,c;i<=m;i++)
        {
            scanf("%d%d%d",&a,&b,&c);
            edge1[i]={b,index1[a],c};
            index1[a]=i;
            edge2[i]={a,index2[b],c};
            index2[b]=i;
        }
        dijkstra();
        printf("%d\n",dfs(1,0));
    }
    return 0;
```

```
}
```

总结

搜索中常用的技巧，与dp相似。