冲刺 Noip2021 模拟赛 A 组 day2

题目名称	矩阵赋值	统计数字	摘果问题	第三题
存盘文件名	matrix.cpp	rabbit.cpp	clever.cpp	grid.cpp
输入文件名	matrix.in	rabbit.in	clever.in	grid.in
输出文件名	matrix.out	rabbit.out	clever.out	grid.out
时限	1s	1s	1s	1s
内存限制	512M	512M	512M	512M

【注意事项】：请自行完成题目，切勿讨论。

1．矩阵赋值

【算法分析】

我们记录每行/列最后一次修改的值与时间，对于每个位置 (i,j)，取决于第 i 行/第 j 列中最晚的修改。

【核心代码】

```
int main()
{
    freopen("matrix.in","r",stdin);
    freopen("matrix.out","w",stdout);
    n=read();m=read();q=read();
    for(int i=1;i<=q;i++)
    {
        int t=read(),x=read(),y=read();
        //记录修改后的值与时间
        if(t==1)id1[x]=y,t1[x]=i;
        else id2[x]=y,t2[x]=i;
    }
    for(int i=1;i<=n;i++)
    {
        for(int j=1;j<=m;j++)
        //以最后修改为准
        if(t1[i]>t2[j])cout<<id1[i]<<" ";
        else cout<<id2[j]<<" ";
        cout<<endl;
    }
}
```

【参考程序】

```
#include<bits/stdc++.h>
using namespace std;
const int N=1e6;
int n,m,q;
int id1[N],id2[N],t1[N],t2[N];
int read()
```

```
{
    int res=0,f=1;char c=getchar();
    while(c<'0'||c>'9'){if(c=='-')f=-1;c=getchar();}
    while(c>='0'&&c<='9'){res=res*10+c-'0';c=getchar();}
    return res*f;
}

int main()
{
    freopen("matrix.in","r",stdin);
    freopen("matrix.out","w",stdout);
    n=read();m=read();q=read();
    for(int i=1;i<=q;i++) {
        int t=read(),x=read(),y=read();
        //记录修改后的值与时间
        if(t==1)id1[x]=y,t1[x]=i;
        else id2[x]=y,t2[x]=i;
    }
    for(int i=1;i<=n;i++) {
        for(int j=1;j<=m;j++)
        //以最后修改为准
        if(t1[i]>t2[j])cout<<id1[i]<<" ";
        else cout<<id2[j]<<" ";
        cout<<endl;
    }
}
```

2．统计数字(rabbit)

【算法分析】

rabbit number 各位数字一定 ≤ 3。

若某数字 x 的一位，$a \geq 4$，那么它在该位的贡献是 a^2，而在 x 中该位自乘进了一位。故贡献为 $a\text{^}2-10+1$。导致 $s(x^2)<s(x)*s(x)$ 所以一定不是 rabbit number。

既然各位数字<=3，枚举各位数字爆搜，时间复杂度 $O(4^9)$。

【核心代码】

```
inline int S(long long x) {//统计各位之和
    int res = 0;
    while (x) {
        res += x % 10;
        x /= 10;
    }
    return res;
}
```

```
inline void dfs(int k, long long cur, int sum) {
//第几位/此数/各位之和
    if (k > 10) {
        if (cur >= l && cur <= r)//判是否越界
            ans += S(cur * cur) == sum * sum;//符合条件就+1
        return;
    }
    for (int i = 0; i <= 3; i++)
        dfs(k + 1, cur * 10 + i, sum + i);//暴搜
}
```

【参考程序】

```
#include <bits/stdc++.h>
using namespace std;
int l, r, ans;

inline int S(long long x) {//统计各位之和
    int res = 0;
    while (x) {
        res += x % 10;
        x /= 10;
    }
    return res;
}

inline void dfs(int k, long long cur, int sum) {
    //第几位/此数/各位之和
    if (k > 10) {
        if (cur >= l && cur <= r)//判是否越界
            ans += S(cur * cur) == sum * sum;//符合条件就+1
        return;
    }
    for (int i = 0; i <= 3; i++)
        dfs(k + 1, cur * 10 + i, sum + i);//暴搜
}

int main() {
    freopen("rabbit.in", "r", stdin);
    freopen("rabbit.out", "w", stdout);
    cin >> l >> r;
    dfs(1, 0, 0);
    cout << ans << endl;
}
```

3．摘果问题

【算法分析】

计算出通过每条边所需的时间，$O(n^2)$ 枚举 i,j 判断 i 是否能跳到 j 与所需时间。

最后 Floyd 或者其他最短路算法即可求出从 1 到 n 的最短路。

时间复杂度 $O(n^3)$ or $O(n^2)$。

【核心代码】

```
int main()
{
    freopen("clever.in","r",stdin);
    freopen("clever.out","w",stdout);
    memset(f,0x7f,sizeof(f));
    cin>>n>>v;
    for(int i=1;i<=n;i++)
    {
        cin>>x[i]>>y[i]>>s;
        if(i==1)continue;
        //加入树边
        f[i][s]=f[s][i]=
sqrt((x[i]-x[s])*(x[i]-x[s])+(y[i]-y[s])*(y[i]-y[s]))/v*1.0;
    }
    for(int i=1;i<=n;i++)
    for(int j=1;j<=n;j++)
    if(x[i]==x[j]&&y[i]>y[j])//加入通过跳的边
    f[i][j]=min(f[i][j],sqrt((y[i]-y[j])*2/10.0));
    for(int k=1;k<=n;k++)//最短路
    for(int i=1;i<=n;i++)
    if(i!=k)
    for(int j=1;j<=n;j++)
    if(i!=j&&j!=k)f[i][j]=min(f[i][j],f[i][k]+f[k][j]);
    printf("%.2lf\n",f[1][n]);
    return 0;
}
```

【参考程序】

```
#include<bits/stdc++.h>
using namespace std;
const int N=105;
double f[N][N];
int x[N],y[N];
int n,v,s;
int main()
{
    freopen("clever.in","r",stdin);
    freopen("clever.out","w",stdout);
```

```
    memset(f,0x7f,sizeof(f));
    cin>>n>>v;
    for(int i=1;i<=n;i++)
    {
        cin>>x[i]>>y[i]>>s;
        if(i==1)continue;
        //加入树边
        f[i][s]=f[s][i]=
sqrt((x[i]-x[s])*(x[i]-x[s])+(y[i]-y[s])*(y[i]-y[s]))/v*1.0;
    }
    for(int i=1;i<=n;i++)
    for(int j=1;j<=n;j++)
    if(x[i]==x[j]&&y[i]>y[j]) //加入通过跳的边
    f[i][j]=min(f[i][j],sqrt((y[i]-y[j])*2/10.0));
    for(int k=1;k<=n;k++)      //最短路
    for(int i=1;i<=n;i++)
    if(i!=k)
    for(int j=1;j<=n;j++)
    if(i!=j&&j!=k)f[i][j]=min(f[i][j],f[i][k]+f[k][j]);
    printf("%.2lf\n",f[1][n]);
    return 0;
}
```

4．网格连通

【算法分析】

我们考虑用并查集将每个点属于的连通块编号，以及每个连通块的大小求出来。

可以考虑将答案转化成求，与子正方形相接的连通块大小之和的最大值（这里不包括子正方形内部的点）。

因此我们可以在枚举子正方形的时候，将子正方形内的所有位置，在其对应的连通块大小中扣除，然后再枚举子正方形外圈的连通块，求它们的大小之和，取最大值即为所求。

这样转化后，我们的信息就有可加减的性质。

上面算法的时间复杂度瓶颈在于，每次要枚举子正方形中的每个位置。

实际上，枚举子正方形的时候，把一个子正方形向右移，影响的只有两个列，只需要枚举这两个列的所有位置即可。

时间复杂度 $O(n^3)$。

【核心代码】

```
void add(int i,int j){if(!in[c[i][j]]++)ans+=sum[c[i][j]];} //加入
void del(int i,int j){if(!--in[c[i][j]])ans-=sum[c[i][j]];} //删除
void dfs(int x,int y,int d) //找连通块
{
    if(s[x][y]!='.'||c[x][y])return;
    c[x][y]=d;sum[d]++;
    dfs(x+1,y,d);dfs(x-1,y,d);
```

```
    dfs(x,y+1,d);dfs(x,y-1,d);
}

int main()
{
    freopen("grid.in","r",stdin);
    freopen("grid.out","w",stdout);
    n=read();k=read();
    for(int i=1;i<=n;i++)cin>>s[i]+1;

    for(int i=1;i<=n;i++)
    for(int j=1;j<=n;j++)
    if(!c[i][j])dfs(i,j,++cnt);

    for(int x=1;x+k-1<=n;x++)
    {
        int y=x+k-1;
        int res=0;ans=0;
        for(int i=1;i<=cnt;i++)in[i]=0;//初始化

        for(int i=x;i<=y;i++)
        for(int j=1;j<=k;j++)
        if(s[i][j]=='.')add(i,j);//加入
        else res++;//统计 #

        //加入外围
        for(int i=x;i<=y;i++)
        if(s[i][k+1]=='.')add(i,k+1);
        for(int j=1;j<=k;j++)
        {
            if(s[x-1][j]=='.')add(x-1,j);
            if(s[y+1][j]=='.')add(y+1,j);
        }
        maxx=max(maxx,res+ans);
        for(int l=2;l+k-1<=n;l++)//移动
        {
            int r=l+k-1;
            for(int i=x;i<=y;i++)
            {
                if(s[i][r]!='.')res++;
                if(s[i][l-1]!='.')res--;
                if(s[i][l-2]=='.')del(i,l-2);
                if(s[i][r+1]=='.')add(i,r+1);
            }
```

```
            if(s[x-1][r]=='.')add(x-1,r);
            if(s[y+1][r]=='.')add(y+1,r);
            if(s[x-1][l-1]=='.')del(x-1,l-1);
            if(s[y+1][l-1]=='.')del(y+1,l-1);
            maxx=max(maxx,res+ans);
        }
    }
    cout<<maxx<<endl;
}
```

【参考程序】

```
#include<bits/stdc++.h>
using namespace std;
const int N=505;
int n,k,cnt,maxx,ans;
int fa[N*N],bo[N*N];
int c[N][N];

char s[N][N];
int in[N*N],sum[N*N];
int read()
{
    int res=0,f=1;char c=getchar();
    while(c<'0'||c>'9'){if(c=='-')f=-1;c=getchar();}
    while(c>='0'&&c<='9'){res=res*10+c-'0';c=getchar();}
    return res*f;
}

void add(int i,int j){if(!in[c[i][j]]++)ans+=sum[c[i][j]];} //加入
void del(int i,int j){if(!--in[c[i][j]])ans-=sum[c[i][j]];} //删除
void dfs(int x,int y,int d) //找连通块
{
    if(s[x][y]!='.'||c[x][y])return;
    c[x][y]=d;sum[d]++;
    dfs(x+1,y,d);dfs(x-1,y,d);
    dfs(x,y+1,d);dfs(x,y-1,d);
}

int main()
{
    freopen("grid.in","r",stdin);
    freopen("grid.out","w",stdout);
    n=read();k=read();
    for(int i=1;i<=n;i++)cin>>s[i]+1;
```

```
    for(int i=1;i<=n;i++)
    for(int j=1;j<=n;j++)
    if(!c[i][j])dfs(i,j,++cnt);

    for(int x=1;x+k-1<=n;x++)
    {
        int y=x+k-1;
        int res=0;ans=0;
        for(int i=1;i<=cnt;i++)in[i]=0;//初始化

        for(int i=x;i<=y;i++)
        for(int j=1;j<=k;j++)
        if(s[i][j]=='.')add(i,j);//加入
        else res++;//统计 #

        //加入外围
        for(int i=x;i<=y;i++)
        if(s[i][k+1]=='.')add(i,k+1);
        for(int j=1;j<=k;j++)
        {
            if(s[x-1][j]=='.')add(x-1,j);
            if(s[y+1][j]=='.')add(y+1,j);
        }
        maxx=max(maxx,res+ans);
        for(int l=2;l+k-1<=n;l++)//移动
        {
            int r=l+k-1;
            for(int i=x;i<=y;i++)
            {
                if(s[i][r]!='.')res++;
                if(s[i][l-1]!='.')res--;
                if(s[i][l-2]=='.')del(i,l-2);
                if(s[i][r+1]=='.')add(i,r+1);
            }
            if(s[x-1][r]=='.')add(x-1,r);
            if(s[y+1][r]=='.')add(y+1,r);
            if(s[x-1][l-1]=='.')del(x-1,l-1);
            if(s[y+1][l-1]=='.')del(y+1,l-1);
            maxx=max(maxx,res+ans);
        }
    }
    cout<<maxx<<endl;
}
```