深度优先搜索

search

第一课

搜索

搜索算法是利用计算机的高性能来**有目的**的**穷举**一个问题的部分或所有的**可能情况**，从而求出问题的解 。

深度优先搜索 Depth First Search （DFS）

广度优先搜索 Breadth First Search （BFS）

深度优先搜索Depth First Search （DFS）
由图灵奖得主 霍普克罗夫特与塔扬发明

John Hopcroft

康奈尔大学计算机科学系教授

美国国家科学院和工程院院士

Robert Tarjan

普林斯顿大学计算机系教授

LCA、强连通分量、splay、非波拉契堆、LCT、反阿克曼函数.....

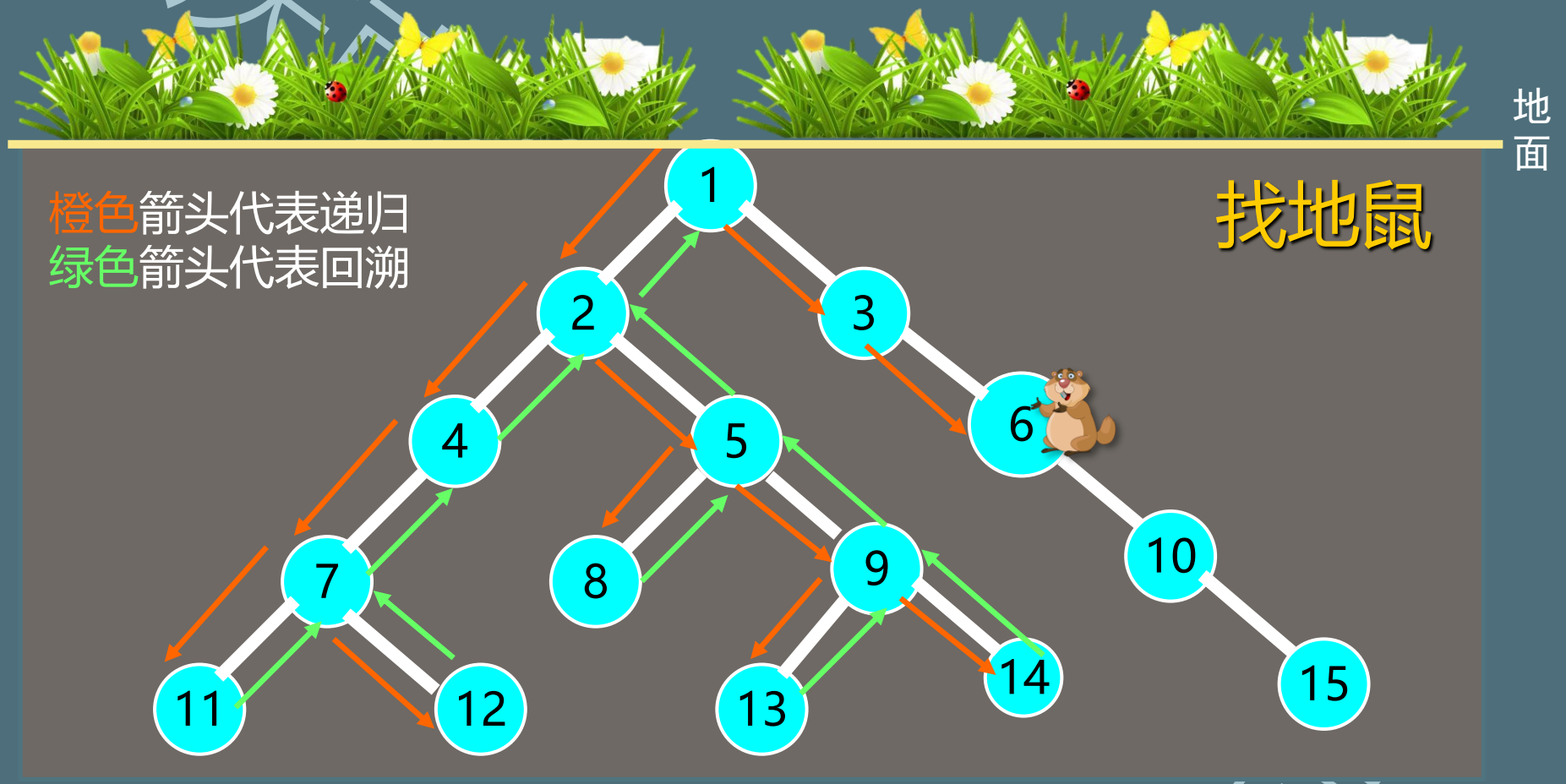
```
#include <bits/stdc++.h>
using namespace std;
int work(int x)
{
    int temp;
    if(x>0)
    {
        temp=x+work(x-1);
        return(temp);
    }
    else return 0;
}

int main()
{
    int a;
    cin>>a;
    cout<<work(a)<<endl;
}
```

输入：5
输出？15

可直接写成 return (x+work(x-1));

深度优先搜索

从问题的某一种可能出发，搜索从这种情况出发所能达到的所有可能，当这一条路走到“尽头”而没找到解时，再倒回上一个步，从另一个可能出发，继续搜索.



一直走到底，走不通就掉头试下一条路。

搜索例题一　逃离迷宫1 NKOJ3170

有一个方格迷宫，我们可以将它看作一个n*m的矩阵，每个方格表示一个房间，方格中有数字0和1，数字0表示该房间是空的，可以顺利通过，数字1表示该房间有怪兽，不能通过。

一开始何老板位于左上角的方格(坐标（1,1）位置)，他要走到右下角的出口(坐标（n,m）位置)，每一步何老板只能往下或往右走。

他想知道总共有多少条可行的线路？　　1<=n,m<=20

输入格式

第一行，两个整数n和m

接下来是一个有数字0和1构成的n*m的矩阵，表示迷宫

输出格式

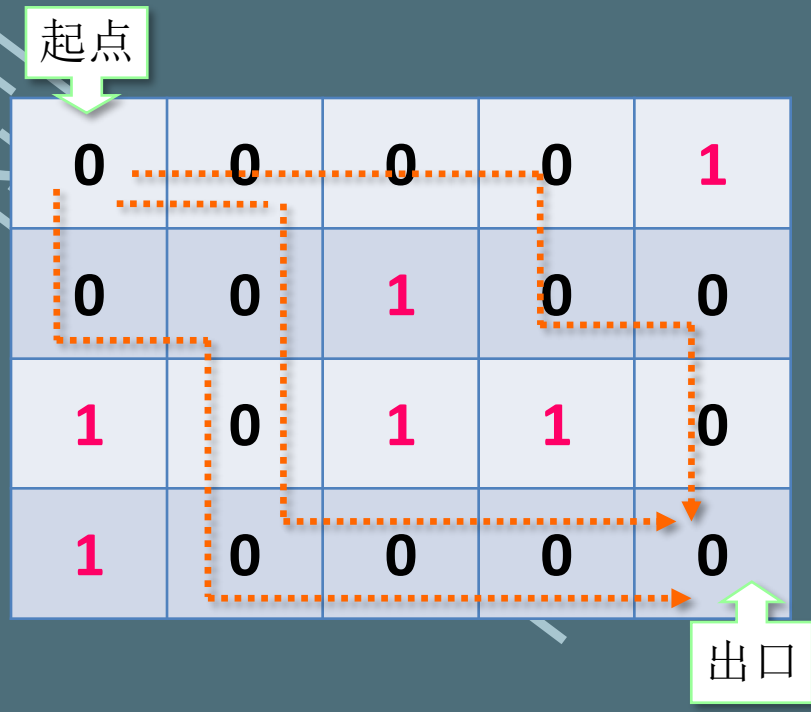
一个整数，表示可行路线的条数。

样例输入

```
4 5
0 0 0 0 1
0 0 1 0 0
1 0 1 1 0
1 0 0 0 0
```

样例输出

```
3
```



```
int n,m,ans=0,Map[21][21];
void FindWay(int x,int y)                    //搜索，从点（x,y）出发，到终点（n,m）的路径数
{
      if(x==n && y==m)ans++;                 //若当前位置就是终点，表示找到一条路
      else
      {
            if(x+1<=n && Map[x+1][y]==0)FindWay(x+1,y);   //往下走一步
            if(y+1<=m && Map[x][y+1]==0)FindWay(x,y+1);   //往右走一步
      }      //搜索时必须确保每一步都在地图范围内，且是安全的
}

int main()
{
      cin>>n>>m;                                          //输入地图尺寸，坐标(n,m)即是终点
      for(int i=1;i<=n;i++)
          for(int j=1;j<=m;j++)cin>>Map[i][j];            //读入地图
      if(Map[1][1]==0)FindWay(1,1);                       //起点为(1,1)，若该处没有怪兽，则搜索
      cout<<ans<<endl;
}
```

搜索例题二　逃离迷宫2　NKOJ3171

有一个方格迷宫，我们可以将它看作一个n*m的矩阵，每个方格表示一个房间，方格中有数字0和1，数字0表示该房间是空的，可以顺利通过，数字1表示该房间有怪兽，不能通过。

一开始何老板位于左上角的方格(坐标（1,1）位置)，他要走到右下角的出口(坐标（n,m）位置)，每一步何老板可以往上、下、左、右走。

他想知道**最少需要几步**就可以走出迷宫？　　1<=n,m<=20

输入格式

第一行，两个整数n和m

接下来是一个有数字0和1构成的n*m的矩阵，表示迷宫

输出格式

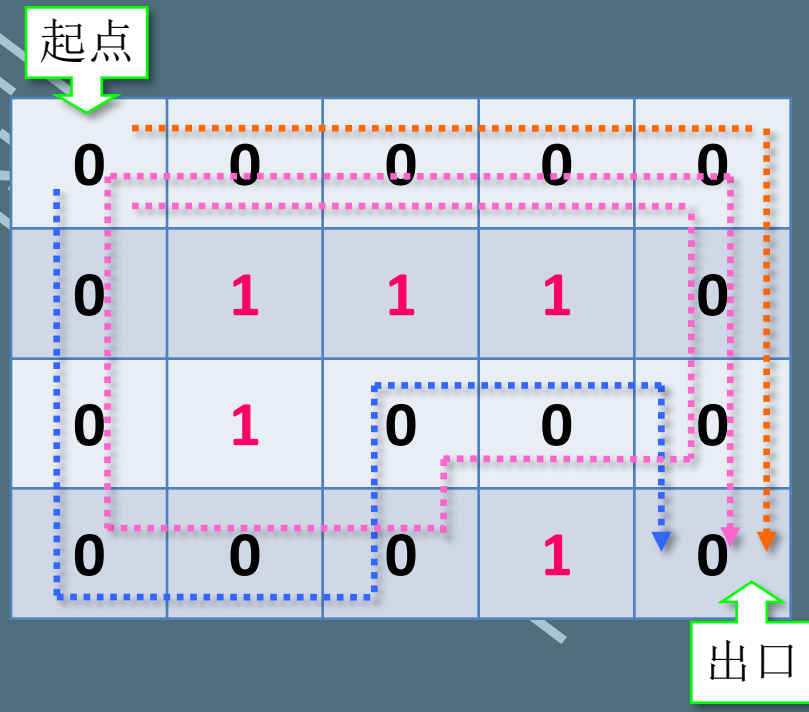
一个整数，表示可行路线的条数。

样例输入

```
4 5
0 0 0 0 0
0 1 1 1 0
0 1 0 0 0
0 0 0 1 0
```

样例输出

```
7
```



```
int n,m,Map[21][21],ans=1000000000;   //Map[ ][ ]数组记录地图

void FindWay(int x,int y,int dis)     //从起点走到(x,y)位置，步数为dis
{
        if(x==n && y==m)
        {
                if(dis<ans)ans=dis;
                return;                //只要到了终点，就结束搜索
        }

        if(x+1<=n && Map[x+1][y]==0)FindWay(x+1,y,dis+1);     //往下走一步
        if(y+1<=m && Map[x][y+1]==0)FindWay(x,y+1,dis+1);     //往右走一步
        if(x-1>=1 && Map[x-1][y]==0)FindWay(x-1,y,dis+1);     //往上走一步
        if(y-1>=1 && Map[x][y-1]==0)FindWay(x,y-1,dis+1);     //往左走一步
}

int main()
{
        ......                                 //读入地图
        if(Map[1][1]==0)FindWay(1,1,0);        //起点为[1,1]，若该处没有怪兽，则搜索
        cout<<ans<<endl;
}
```

这个搜索会“死”掉为什么呢?

因为每个点都可能被反复经过。怎样保证不重复搜索呢?

稍作优化

```
const int inf=1000000000;
int n,m,Map[21][21],Step[21][21];   //Step[x][y]记录从起点走到[x,y]所需最小步数

void FindWay(int x,int y,int dis)   //从起点走到(x,y)位置，步数为dis
{
        //如果dis<Step[x][y]表示当前方案比以前的方案更优，更新Step[x][y];否则不必继续搜索下去了，return;
        if(dis<Step[x][y])Step[x][y]=dis; else return;
        if(x==n && y==m)return; //如果到了终点，结束搜索

        if(x+1<=n && Map[x+1][y]==0)FindWay(x+1,y,dis+1);          //往下走一步
        if(y+1<=m && Map[x][y+1]==0)FindWay(x,y+1,dis+1);          //往右走一步
        if(x-1>=1 && Map[x-1][y]==0)FindWay(x-1,y,dis+1);          //往上走一步
        if(y-1>=1 && Map[x][y-1]==0)FindWay(x,y-1,dis+1);          //往左走一步
}

int main()
{
        ......                                          //读入地图
        for(i=1;i<=n;i++)
            for(j=1;j<=m;j++)Step[i][j]=inf;
        if(Map[1][1]==0)FindWay(1,1,0);       //起点为[1,1]，若该处没有怪兽，则搜索
        cout<<Step[n][m]<<endl;
}
```


深度优先搜索（DFS）

```
void DFS( Point P )
{
        for(所有P的邻接点K)
        {
                if(K未被访问)
                {
                        标记K;
                        DFS(K);

                }
        }
}
```

每次递归到一个点，则检查是否存在与它相邻，而且未被访问的点，有则递归访问这个点，无则返回上一层。

搜索例题三　分数(NKOJ1072)

有n个数（n<=20），让你分成两堆。使两堆和的差最小(绝对值)。

Input

两行，第一行，一个数n，表示数字总的个数。
第二行，n个数字(每个数字不超过20000)

Output

一个正整数，表示两堆数字和的差的最小值。

Sample Input

```
5
1   3   6   9   23
```

Sample Output

```
4
```


搜索分析：

所有数字的总和**sum**=a[1]+a[2]+...+a[n]

如果一堆数字的总和为**tot**，那么，两堆数字的差为？ **abs(sum-2*tot)**

只用讨论一堆数字的情况，讨论第x个数字是否选入该堆。

```
void DFS(int x,int tot)//当前讨论第x个数选，tot表示前面选的数字总和
{
    if(abs(sum-2*tot)<Min)Min=abs(sum-2*tot); //Min为全局变量，记录最优值
    if(x<=n)
    {
        DFS(x+1,tot+a[x]);//选x，下一次讨论第x+1个数，已选数字总和加上刚刚选的第x个数
        DFS(x+1,tot);     //不选x
    }
}
```

是否存在其他解法？DP

```
int main()
{
    scanf("%d",&n);
    for(int i=1;i<=n;i++)
    {
        scanf("%d",&a[i]);
        sum+=a[i];
    }
    Min=inf;
    DFS(1,0);
    printf("%d\n",Min);
}
```