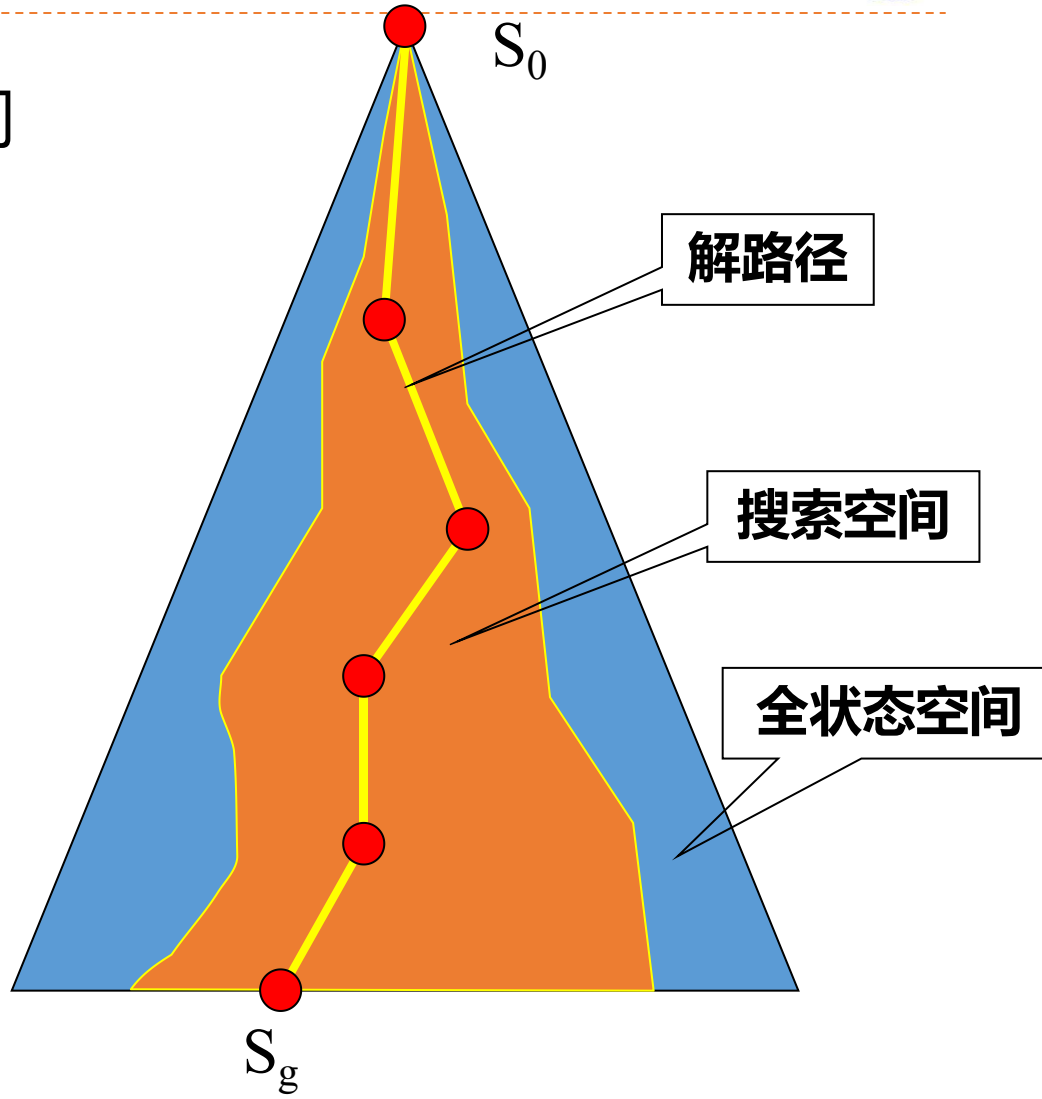
DFS实践

信智学院　陈琰宏

3.1.1 搜索算法

由此，可以看出这类问题的解，就是一个合法状态的序列，其中序列中第一个状态是问题的初始状态，而最后一个状态则是问题的结束状态。如图所示即搜索问题的示意图：



3.1.2 搜索算法常见分类

- 枚举算法
- 深度优先搜索(DFS)与回溯
- 广度优先搜索（BFS)
- 双向广度优先搜索
- A*算法
- 群智能搜索算法

3.1.3 回溯搜索概念

回溯法也称试探法.

它的基本思想是：从问题的某一种状态（初始状态）出发，搜索从这种状态出发所能达到的所有“状态”，当一条路走到“尽头”的时候（不能再前进），再后退一步或若干步，从另一种可能“状态”出发，继续搜索，直到所有的“路径”（状态）都试探过。

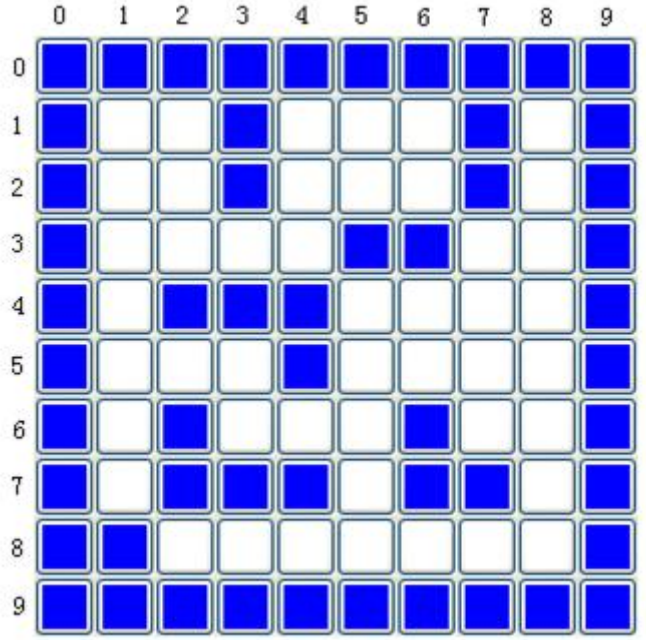
这种不断“前进”、不断“回溯”寻找解的方法，就称作“回溯法”。

搜索框架

递归回溯：由于回溯法是对解空间的深度优先搜索，因此在一般情况下可用递归函数来实现回溯法如下：

递归回溯法算法框架[一]

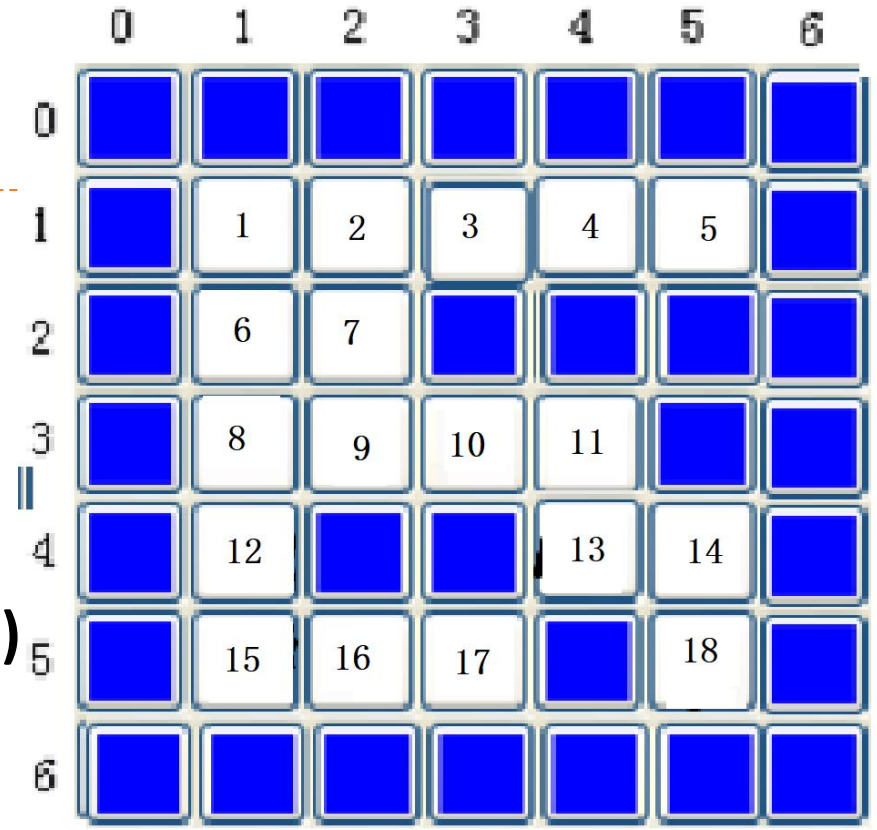
```
int Search(int k)
    {
    for (i=1;i<=算符种数;i++)
        if (满足条件)
            {
                保存结果
                if (到目的地) 输出解;
                    else Search(k+1);
                恢复：保存结果之前的状态{回溯一步}
            }
    }
```



搜索框架

递归回溯法算法框架[二]

```
int Search(int k)
   {
    if  (到目的地) 输出解;
          else
              for (i=1;i<=算符种数;i++)
                  if  (满足条件)
                        {
                           保存结果;
                        Search(k+1);
                           恢复：保存结果之前的状态{回溯一步}
                        }
   }
```



1[3302]素数环

素数环:从1到n这n个数摆成一个环，要求相邻的两个数的和是一个素数。
如，n=8是，素数环为：
1 2 3 8 5 6 7 4
1 2 5 8 3 4 7 6
1 4 7 6 5 8 3 2
1 6 7 4 3 8 5 2
总数为4
输入
输入n的值（n不大于15）
输出

样例输入
8
样例输出
4

算法分析

非常明显，这是一道回溯搜索的题目。从1开始，每个空位有20种可能，只要填进去的数合法：与前面的数不相同；与左边相邻的数的和是一个素数。第20个数还要判断和第1个数的和是否素数。

【算法流程】

1、数据初始化；

2、递归填数：判断第i个数填入是否合法；

A、如果合法：填数；判断是否到达目标（20个已填完）：是，打印结果；不是，递归填下一个；

B、如果不合法：选择下一种可能；

1[3370]字母有重复全排

对n个元素进行全排列。这n个元素由小写字母组成，这n个元素中的某些可能相同

[输入格式] 1<=n<=500

[输出格式]每行一个序列，按字典序输出最后一行输出方案数

[输入样例]

4

aacc

[输出样例]

aacc

acac

acca

caac

caca

ccaa

6

分析

考虑搜索的框架：

1. 目标状态是什么？
2. 每个位置的搜索空间是什么？

输入
4
aacc
[输出]
aacc
acac
acca
caac
caca
ccaa
6

1. 目标状态：

把n个字母放完

```
if(k==n+1){
    cnt++;
    for(int i=1;i<=n;i++){
        printf("%c",a[i]);
    }
    printf("\n");
    return;
}
```

2. 每个位置的搜索空间是什么？ 所有小写字母

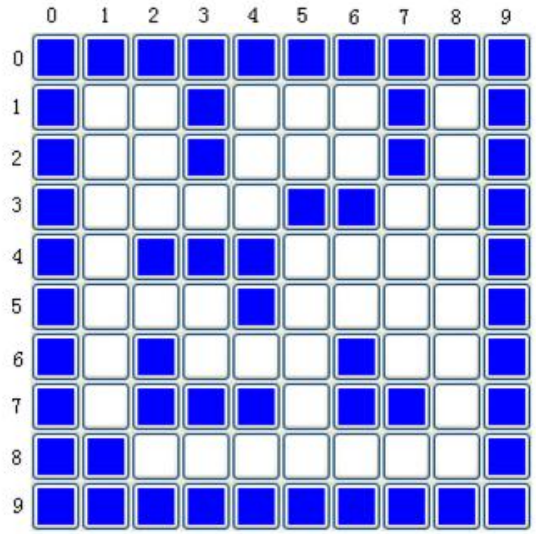
1[3437]字母有重复全排

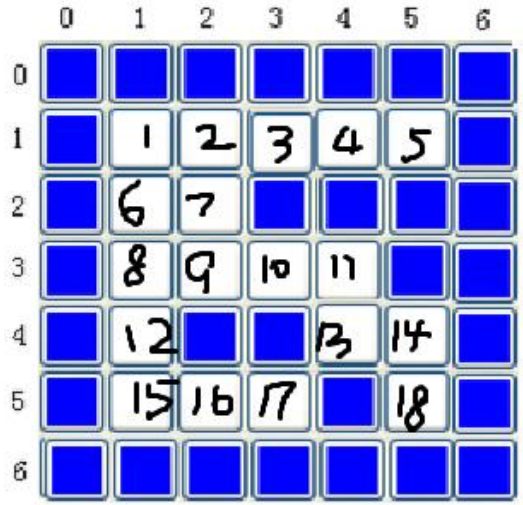
```
7  void dfs(int k){
8      if(k==n+1){
9          ____________
10         return;
11     }
12     for(char i=_____;i<=_____;_______){
13         if(____________){
14             a[k]=i;
15             ____________
16             dfs(k+1);
17             ____________;
18         }
19     }
20 }
```

1[3437]字母有重复全排

4 [3727]迷宫3求路径次数

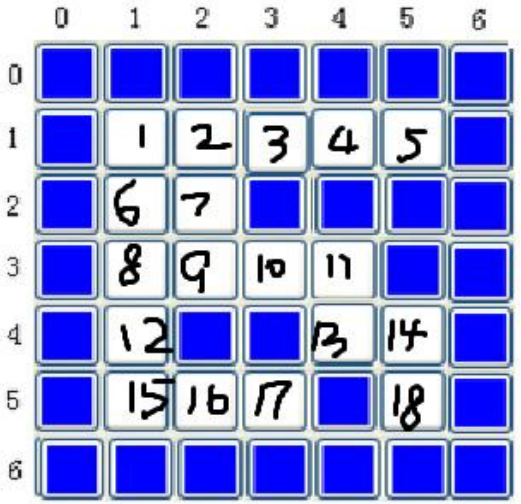
给定一个M×N的迷宫图，求从指定入口（1,1）到出口（M,N）有多少种走法。例如迷宫图如图所示（M=10，N=10），其中的方块图表示迷宫。对于图中的每个方块，用空白表示通道，用阴影表示墙。要求所求路径必须是简单路径，即在求得的路径上不能重复出现同一通道块。





分析

如图所示，蓝色的表示墙壁，白色的表示可走的方块，题目求解从入口位置1（坐标（1,1））出发，到达位置18（坐标（5,5））。



1. 确定搜索方向

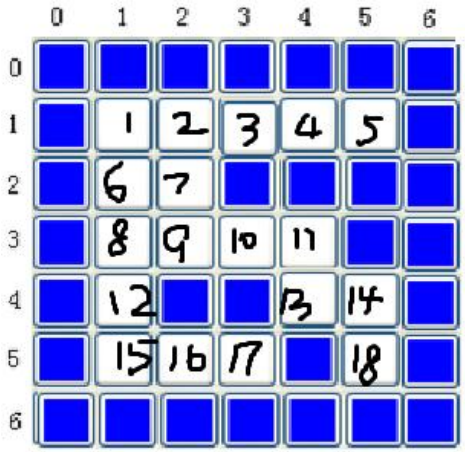
对于每一个方块（位置）而言，共有四种走法：上下左右。对于计算机而言所有的操作都是有序的，因此需要规定搜索过程中，寻路的顺序，如上下左右，左右上下等。

本解题采用的搜索顺序为左上右下，定义方向数组：

```
int dir[4][2]={{-1,0},{0,1},{1,0},{0,-1}};//定义方向
```

分析

2. 确定搜索"到达目的地"的条件



假设(xe,ye)表示迷宫出口位置，则到达迷宫出口的代码可以表示成：

```
if(xe==m&&ye==n){sum++;}
```

3. 记录搜索过程中走过位置的状态

当一个位置被走过以后，为了避免来回重复走，需要记录当前的位置有没有被走过，如果被走过，这个位置就不能再走。假设用vis[][]数组记录位置的状态，初始时vis数组的值全部初始化为0，vis[x][y]=0**表示该位置没有走过，vis[i][j]=1表示该位置已经被走过。**

实现代码如下：

```
vis[x][y]=1;//用1表示该位置没有被走过
```

分析

```
#include<iostream>
#include<cstring>
using namespace std;
int a[12][12];//定义迷宫
int dir[4][2]={{-1,0},{0,1},{1,0},{0,-1
int vis[12][12];//标记有没有走过
int sum=0,xe,ye;//xe,ye终点
```

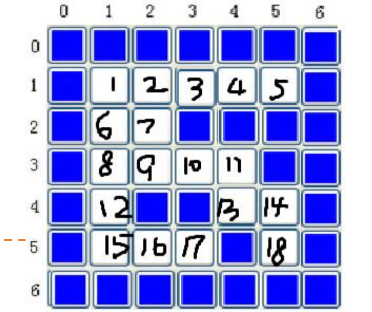
```
void dfs(int x,int y){
    if(_____1_____){
        sum++;//到达目的地
    }
    else{
        for(int i=0;i<4;i++){
            int xx=_____2_____;//下一个点
            int yy=_____2_____;
            if(_____3_____){
                _____4_____;
                _____5_____;
                _____6_____;
            }
        }
    }
}
```

```
int main(){
    int m,n,i,j;
    cin>>m>>n;
    xe=m,ye=n;
    memset(a,1,sizeof(a));
    for(i=1;i<=m;i++)
        for(j=1;j<=n;j++)
            cin>>a[i][j];

    vis[1][1] = 1;//记录第一点没访问过
    dfs(1,1);
    cout<<sum<<endl;
    return 0;
}
```

代码实现

```
#include<iostream>
#include<cstring>
using namespace std;
int a[12][12];//定义迷宫
int dir[4][2]={{-1,0},{0,1},{1,0},{0,-1}};
int vis[12][12];//标记有没有走过
int sum=0,xe,ye;//xe,ye终点
void dfs(int x,int y){
    if(x==xe&&y==ye){
        sum++;//到达目的地
    }
    else{
        for(int i=0;i<4;i++){
            int xx=x+dir[i][0];//下一个点
            int yy=y+dir[i][1];
            if(a[xx][yy]==0&&vis[xx][yy]==0){
                vis[xx][yy]=1;//记
                dfs(xx,yy);
                vis[xx][yy]=0;//退
            }
        }
    }
}
int main(){
    int m,n,i,j;
    cin>>m>>n;
    xe=m,ye=n;
    memset(a,1,sizeof(a));
    for(i=1;i<=m;i++)
        for(j=1;j<=n;j++)
            cin>>a[i][j];

    vis[1][1] = 1;//记录第一点没访问过
    dfs(1,1);
    cout<<sum<<endl;
    return 0;
}
```

代码实现（理解回溯）




```
for(int i=0;i<4;i++){
    int xx=x+dir[i][0];//下一个点
    int yy=y+dir[i][1];
    if(a[xx][yy]==0&&vis[xx][yy]==0){
        vis[xx][yy]=1;//记录走过
        dfs(xx,yy);
        vis[xx][yy]=0;//退格，下一次
    }
}
```

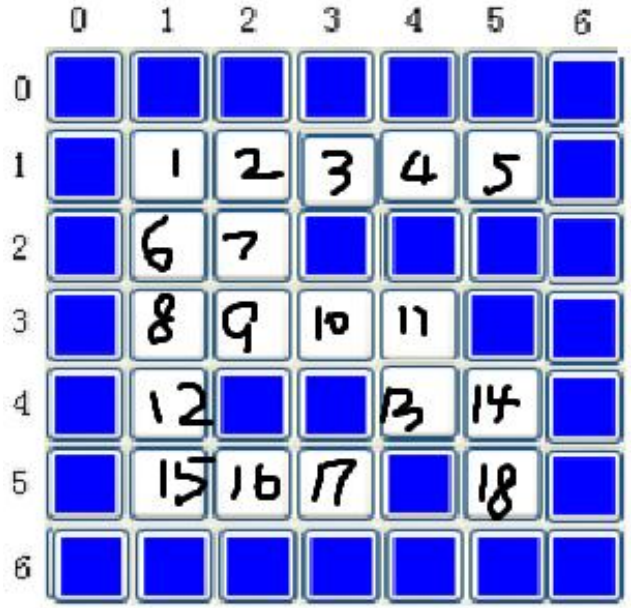

```
for(int i=0;i<4;i++){
    int xx=x+dir[i][0];//下一个点
    int yy=y+dir[i][1];
    if(a[xx][yy]==0&&vis[xx][yy]==0){
        vis[xx][yy]=1;//记录走过
        dfs(xx,yy);
        vis[xx][yy]=0;//退格，下一次
    }
}
```

```
for(int i=0;i<4;i++){
    int xx=x+dir[i][0];//下一个点
    int yy=y+dir[i][1];
    if(a[xx][yy]==0&&vis[xx][yy]==0){
        vis[xx][yy]=1;//记录走过
        dfs(xx,yy);
        vis[xx][yy]=0;//退格，下一次
    }
}
```

```
for(int i=0;i<4;i++){
    int xx=x+dir[i][0];//下一个点
    int yy=y+dir[i][1];
    if(a[xx][yy]==0&&vis[xx][yy]==0){
        vis[xx][yy]=1;//记录走过
        dfs(xx,yy);
        vis[xx][yy]=0;//退格，下一次
    }
}
```

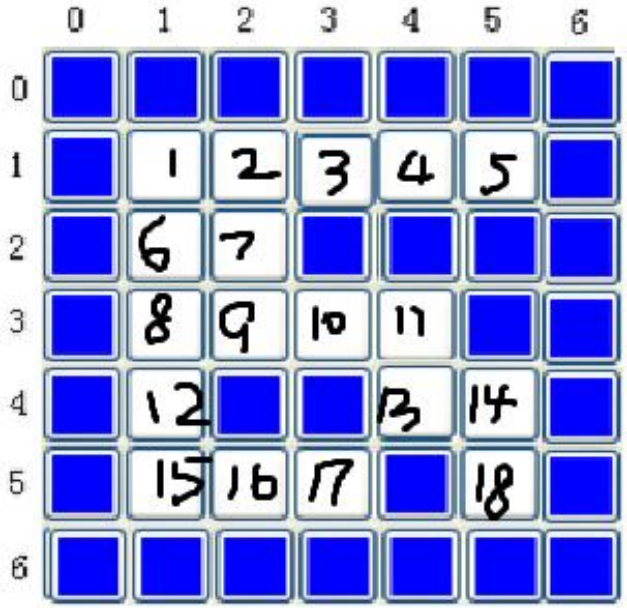
4 [3727] 迷宫3（二）栈模拟

```
1   #include<iostream>
2   #include<cstring>
3   #include<stack>
4   using namespace std;
5   int a[12][12];//定义迷宫
6   int dir[4][2]={{-1,0},{0,1},{1,0},{0,-1}};//定义方向
7   int vis[12][12];//标记有没有走过
8   int sum=0,xe,ye;//xe,ye终点
9
10  struct node{
11      int x,y;
12      int dir;
13  };
```



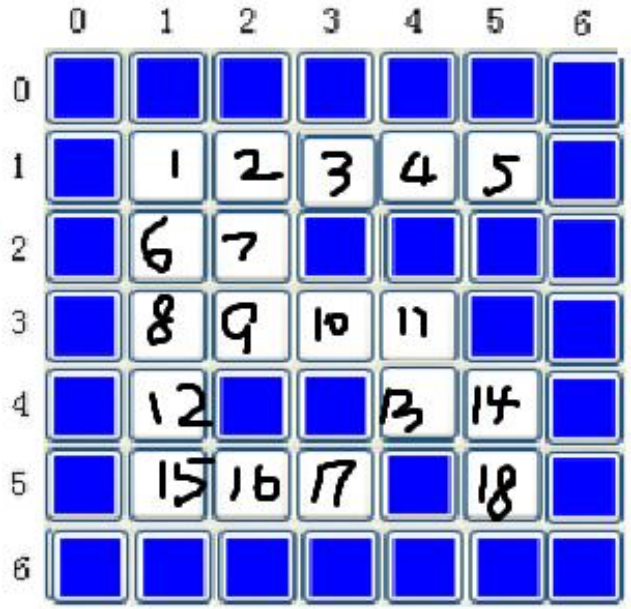
主函数

```
15  int main(){
16      int m,n,i,j;
17      cin>>m>>n;
18      int xe=m,ye=n;
19      memset(a,1,sizeof(a));
20      for(i=1;i<=m;i++)
21          for(j=1;j<=n;j++)
22              cin>>a[i][j];
23
24      node start,t;
25      start.x=1,start.y=1,start.dir=0;
26      stack<node>st;
27      st.push(start);
28      while(!st.empty())
29      {
   ------------------------------------
63      cout<<sum<<endl;
64      return 0;
65  }
```



栈模拟过程1

```
28      while(!st.empty())
29      {
30          node e=st.top();//获取栈点元素
31          vis[e.x][e.y]=1;//记录状态为已处理
32          if(e.x==xe&&e.y==ye)
33          {//到达出口
34              sum++;
35              vis[e.x][e.y]=0;//重新可用
36              st.pop();
37          }
38          while(!st.empty()&&st.top().dir==4)//走不通了回退
39              vis[st.top().x][st.top().y]=0,st.pop();
40          if(st.empty())
41              break;//处理完毕
42          e=st.top();
43          st.pop();//弹出栈点元素
44          for(int i=e.dir;i<4;i++)
```



栈模拟过程2

```
44          for(int i=e.dir;i<4;i++)
45          {//处理下一个点
46              int xx=e.x+dir[i][0];//下一个点
47              int yy=e.y+dir[i][1];
48              if(a[xx][yy]==0&&vis[xx][yy]==0){//可走
49                  node r;
50                  r.x=xx,r.y=yy,r.dir=0;
51                  e.dir=i+1;//更新e的方向
52                  st.push(e);//更新栈内e的值，因为修改了方向
53                  st.push(r);
54                  break;//找到一个可走点，退出继续搜索
55              }
56              if(i==3)
57              {//该方向搜索完毕
58                  e.dir=i+1;
59                  st.push(e);
60              }
61          }
62      }
63      cout<<sum<<endl;
64      return 0;
65  }
```

5 [2769] 迷宫

一天Extense在森林里探险的时候不小心走入了一个迷宫，迷宫可以看成是由n * n的格点组成，每个格点只有2种状态，.和#，前者表示可以通行后者表示不能通行。同时当Extense处在某个格点时，他只能移动到东南西北(或者说上下左右)四个方向之一的相邻格点上，Extense想要从点A走到点B，问在不走出迷宫的情况下能不能办到。如果起点或者终点有一个不能通行(为#)，则看成无法办到。

输入

第1行是测试数据的组数k，后面跟着k组输入。每组测试数据的第1行是一个正整数n (1 ≤ n ≤ 100)，表示迷宫的规模是n * n的。接下来是一个n * n的矩阵，矩阵中的元素为.或者#。再接下来一行是4个整数ha, la, hb, lb，描述A处在第ha行, 第la列，B处在第hb行, 第lb列。注意到ha, la, hb, lb全部是从0开始计数的。

输出

k行，每行输出对应一个输入。能办到则输出"YES"，否则输出"NO"。

5 [2769] 迷宫

样例输入
2
3
.##
..#
#..
0 0 2 2
5
.....
###.#
..#..
###..
...#.
0 0 4 0
样例输出
YES
NO

分析

```
#include <iostream>
using namespace std;
int n;
char a[105][105];
int ha,la,hb,lb;//起点、终点
int d[4][2]={{-1,0},{0,1},{1,0},{0,-1}};//方向数组
int flag=0;//用于标记是否走到出口
void dfs(int x,int y){
    if(x==hb&&y==lb){
        ____1_____________;
        return;
    }
    else{
        for(int i=0;i<4;i++){
            _____2_____________
            if(a[xx][yy]=='.'){
                __3_____________;
                dfs(xx,yy);
                ___4____________;
            }
        }
    }
}
```

代码

```cpp
#include <iostream>
using namespace std;
int n;
char a[105][105];
int ha,la,hb,lb;//起点、终点
int d[4][2]={{-1,0},{0,1},{1,0},{0,-1}};//方向数组
int flag=0;//用于标记是否走到出口
void dfs(int x,int y){
    if(flag==1)return; //找到了就return
    if(x==hb&&y==lb){
        flag=1;
        return;
    }
    else{
        for(int i=0;i<4;i++){
            int xx=x+d[i][0];
            int yy=y+d[i][1];
            if(a[xx][yy]=='.'){
                a[xx][yy]='#';//保存结果
                dfs(xx,yy);
                a[xx][yy]='.';
            }
        }
    }
}
int main(){
    int k;
    cin>>k;
    while(k--){
        flag=0;
        cin>>n;
        for(int i=0;i<n;i++)
            for(int j=0;j<n;j++)
                cin>>a[i][j];//输入地图
        cin>>ha>>la>>hb>>lb;//输入起点和终点
        dfs(ha,la);
        if(flag==1) cout<<"YES"<<endl;
        else cout<<"NO"<<endl;

    }
    return 0;
}
```

6 [2804] 走迷宫

一个迷宫由R行C列格子组成，有的格子里有障碍物，不能走；有的格子是空地，可以走。
给定一个迷宫，求从左上角走到右下角最少需要走多少步(数据保证一定能走到)。只能在水平方向或垂直方向走，不能斜着走。
输入
第一行是两个整数，R和C，代表迷宫的长和宽。（1≤R，C≤40)
接下来是R行，每行C个字符，代表整个迷宫。
空地格子用‘.’表示，有障碍物的格子用‘#’表示。
迷宫左上角和右下角都是‘.’。
输出
输出从左上角走到右下角至少要经过多少步（即至少要经过多少个空地格子）。计算步数要包括起点和终点。

6 [2804] 走迷宫

样例输入
5 5
..###
#....
#.#.#
#.#.#
#.#..
样例输出
9

思路

```
2   #include<cstring>
3   using namespace std;
4   char a[12][12];//定义迷宫
5   int dir[4][2]={{-1,0},{0,1},{1,0},{0,-1}};//定义方向
6   int vis[12][12];//标记有没有走过
7   int minstep=10000,xe,ye;//xe,ye终点
8   void dfs(int x,int y,int step){
9       if(x==xe&&y==ye){
10          _______1_____________;
11      }
12      else{
13          for(int i=0;i<4;i++){
14              ____2_____________________
15              if(__3_____________________________ ){
16                  vis[xx][yy]=1;//记录走过
17                  __4_____________________
18                  vis[xx][yy]=0;//退格，下一次还可以走
19              }
20          }
21      }
22  }
```

代码

```
#include<iostream>
#include<cstring>
using namespace std;
char a[12][12];//定义迷宫
int dir[4][2]={{-1,0},{0,1},{1,0},{0,-1}};//定义方向
int vis[12][12];//标记有没有走过
int minstep=10000,xe,ye;//xe,ye终点
void dfs(int x,int y,int step){
    if(x==xe&&y==ye){
        if(step<minstep)minstep=step;
    }
    else{
        for(int i=0;i<4;i++){
            int xx=x+dir[i][0];//下一个点
            int yy=y+dir[i][1];
            if(a[xx][yy]=='.'&&vis[xx][yy]==0){
                vis[xx][yy]=1;//记录走过
                dfs(xx,yy,step+1);
                vis[xx][yy]=0;//退格，下一次还可以走
            }
        }
    }
}
int main(){
    int m,n,i,j;
    cin>>m>>n;
    xe=m,ye=n;
    memset(a,'#',sizeof(a));
    memset(vis,0,sizeof(a));
    for(i=1;i<=m;i++)
        for(j=1;j<=n;j++)
            cin>>a[i][j];
    dfs(1,1,1);
    cout<<minstep<<endl;
    return 0;
}
```

7 [6922] zb的生日

今天是阴历七月初五，acm队员zb的生日。zb正在和C小加、never在武汉集训。他想给这两位兄弟买点什么庆祝生日，经过调查，zb发现C小加和never都很喜欢吃西瓜，而且一吃就是一堆的那种，zb立刻下定决心买了一堆西瓜。当他准备把西瓜送给C小加和never的时候，遇到了一个难题，never和C小加不在一块住，只能把西瓜分成两堆给他们，为了对每个人都公平，他想让两堆的重量之差最小。每个西瓜的重量已知，你能帮帮他么？

输入

多组测试数据（<=1500）。数据以EOF结尾第一行输入西瓜数量N（1 ≤ N ≤ 20)第二行有N个数，W1，…，Wn（1 ≤ Wi ≤ 10000)分别代表每个西瓜的重量

输出

输出分成两堆后的质量差

样例输入

5

5 8 13 27 14

样例输出

3

分析

因为只有20个西瓜，所以西瓜的组合为2^{20}，约10^6，可以考虑搜索，搜索的过程类似全排列。为了实现“**为了对每个人都公平，他想让两堆的重量之差最小**”，策略应该怎么考虑？

让每个人的重量无限接近v/2，就可以达到最小，因此问题可以转化为 “求min(ans-v/2)”

```cpp
#include <iostream>
#include <cstdio>
#include <string.h>
using namespace std;
int a[21], v, n, m;
void dfs(int i, int ans)
{
    if (ans>v) return;//超过一半退出
    if (i>n)
    {//判断完毕
        if (m<ans) m = ans;
        return;
    }//对于每个西瓜两种选择，取和不取
    dfs(i+1, ans+a[i]);
    dfs(i+1, ans);
}
```

```cpp
int main()
{
    int sum;
    while (~scanf("%d", &n))
    {
        sum = 0;
        for (int i=1; i<=n; i++)
        {
            scanf("%d", &a[i]);
            sum += a[i];
        }
        v = sum/2;//把一半体积作为边界
        m = 0;
        dfs(1, 0);
         cout<<sum-2*m<<endl;
    }
    return 0;
}
```

14 [3371]泉水

小f住在农村，离他的家不远有一口井，传说是小f的祖先开掘的。虽然小f的村子里通了自来水，但是由于这口井井水质量非常的好，因此小f仍然喝这口井里的水。小f非常喜欢这口井，所以他经常去打水。小f的家里有n（n 是偶数）只桶，这些桶虽然大小相等，但是由于很多都有些破损，所以认为它们是不同的。小f经常挑一根扁担（带两只空桶，必须是空的，且是2 只）去井边打水。小f每次去井旁都会把桶中的水装到极限（假设水量无穷，且小f都能够担得动）。设小f挑得是x、y 两只桶，则打水一趟需要走time[x,y]分钟。小f想要在最少的时间内用自己的力量把家里所有的空桶装满。小f觉得这是个难题，于是来找你帮忙。。

14 [3371]泉水

第一行有一个数字，是n（n<=20）。接下来n 行，每行n 个数字，代表了time 矩阵。time 矩阵中每一个数都是正整数，且time 矩阵中time[i,i]是没有用的。（注意time[i,j]=time[j,i]）

输出仅包含一行，就是最佳挑水方案的最少时间。

4
0 58 49 81
58 0 38 76
49 38 0 48
81 76 48 0

106

分析

此题可以抽象为在一个上（下）三角矩阵中选择两个受到约束的值，使得这两个值的和最大。

题目n<=20,最大选择范围2000，时间复杂度最大为10^6，可以用类似求组合数的枚举求解，采用DFS实现。

分析1：目标状态

目标状态：搜索完每一个桶，判断是否是最小值。

```
if(i>n){//目标状态是把所有的都遍历完
    if(sum<minn){
        minn=sum;
    }
    return;
}
```

分析2：搜索过程

一个扁担两个桶，先搜索第一个桶。如果当前桶不可用，直接继续搜索下一个。

确定一个可用后，找到第二个桶。确定两个桶后，记录状态。开始下一轮搜索。

```
15  if(b[i]==1)dfs(i+1,sum);
16      //如果当前i不可用要处理
17  else{
18      for(int j=i+1;j<=n;j++){
19          if(b[i]==0&&b[j]==0){
20              b[i]=1,b[j]=1;
21              dfs(i+1,sum+a[i][j]);
22              b[i]=0,b[j]=0;
23          }
24      }
25      return;
26  }
```

分析3：剪枝

1. sum值一旦超过已知的最小值，该搜索树分支就可以结束。

```
if(sum>minn)//剪枝
    return;
```

代码（主函数）

```
1   #include<bits/stdc++.h>
2   using namespace std;
3   int n,a[20][20],b[20]={0};
4   int minn=0x3f3f3f;
5   void dfs(int i,int sum){
30  int main(){
31      cin>>n;
32      for(int i=1;i<=n;i++){
33          for(int j=1;j<=n;j++){
34              cin>>a[i][j];
35          }
36      }          // 从第一个桶开始搜索
37      dfs(1,0);//初始时间为0
38      cout<<minn;
39      return 0;
40  }
```

代码（DFS1）

```
5  void dfs(int i,int sum){
6      if(sum>minn)//剪枝
7          return;
8
9      if(i>n){//目标状态是把所有的都遍历完
10         if(sum<minn){
11             minn=sum;
12         }
13         return;
14     }
```

代码（DFS2）

```
15          if(b[i]==1)dfs(i+1,sum);
16              //如果当前i不可用要处理
17          else{
18              for(int j=i+1;j<=n;j++){
19                  if(b[i]==0&&b[j]==0){
20                      b[i]=1,b[j]=1;
21                      dfs(i+1,sum+a[i][j]);
22                      b[i]=0,b[j]=0;
23                  }
24              }
25              return;
26          }
27  }
```

[6336] 小猫爬山

给定一组不同长度的木棍，有可能将它们端到端连接成一个正方形吗？

输入的第一行包含N，表示有n个测试用例。每个测试用例包括一个整数M, 4<=M<=20，表示木棒的数量。后面是M个整数，每一个数表示木棍的长度len, 1<=len<=10000之间的整数。

对于每组数据，如果可能形成正方形，则输出包含“是”的行；否则输出“否”。

输入样例

```
4 1 1 1 1
5 10 20 30 40 50
8 1 7 2 6 4 4 3 5
```

输出样例

```
yes
no
yes
```

分析

考虑搜索的框架：

1. 目标状态是什么？
2. 每个位置的搜索空间是什么？

[209] 四边形

给定一组不同长度的木棍，有可能将它们端到端连接成一个正方形吗？

输入的第一行包含N，表示有n个测试用例。每个测试用例包括一个整数M，4<=M<=20，表示木棒的数量。后面是M个整数，每一个数表示木棍的长度len，　1<=len<=10000之间的整数。

对于每组数据，如果可能形成正方形，则输出包含“是”的行；否则输出“否”。

输入样例

```
4 1 1 1 1
5 10 20 30 40 50
8 1 7 2 6 4 4 3 5
```

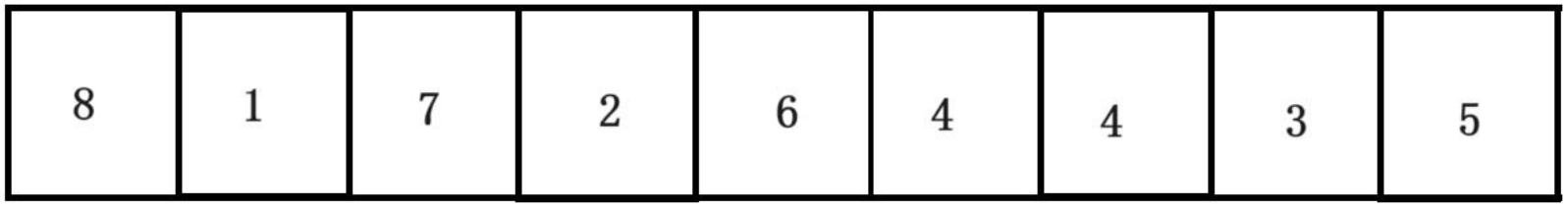
输出样例

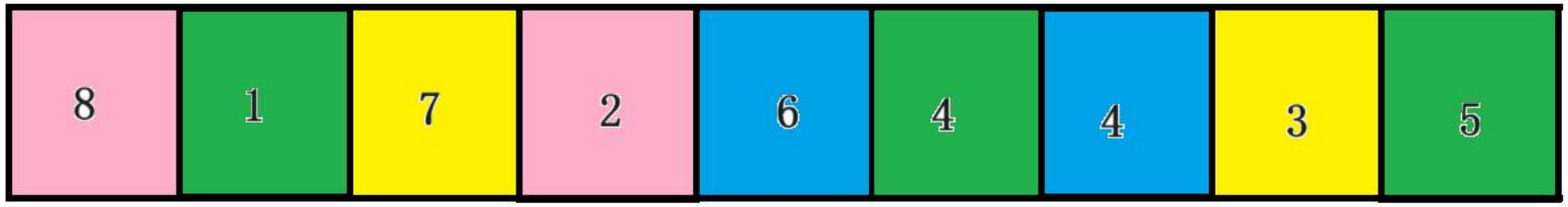
```
yes
no
yes
```

分析

m=9　8 1 7 2 6 4 4 3 5

sum=40 每条边的长度为10





分析

分析

分析

1143: 木瓜地

育才中学的小A同学在外春游， 不小心游荡到郊外，看到了一块木瓜地，木瓜是小A的最爱了，一看到木瓜胃口就会变得无限大。这块木瓜地被分割成一个 R行C列的网格(1 < = R < = 40, 1 < = C < = 40)。小A可以从一个格沿著一条跟X轴或 Y轴平行的直线走到邻接的另一个格。小A发现一开始她自己在木瓜林的(1,1)，也就是第 一行第一列慢悠悠地吃着木瓜。 小A总是用她最信赖地双筒望远镜去数每一个邻接的格的木瓜的数目。然后她就游 荡到那个有最多没有被吃掉的木瓜的邻接的格子（保证这样的格子只有一个）。 按照这种移动方法，最终小A总是会在(R,C)停止然后吃掉那里的木瓜。 给定这个木瓜地的大小及每个格的木瓜数F_ij(1 < = F_ij < = 100)，要求小A一共吃了 多少个木瓜。

今天的课程结束啦……

下课了…
同学们再见！