



浙江财经大学

Zhejiang University Of Finance & Economics



# 搜索初步

信智学院 陈琰宏






# 1 搜索基础

---

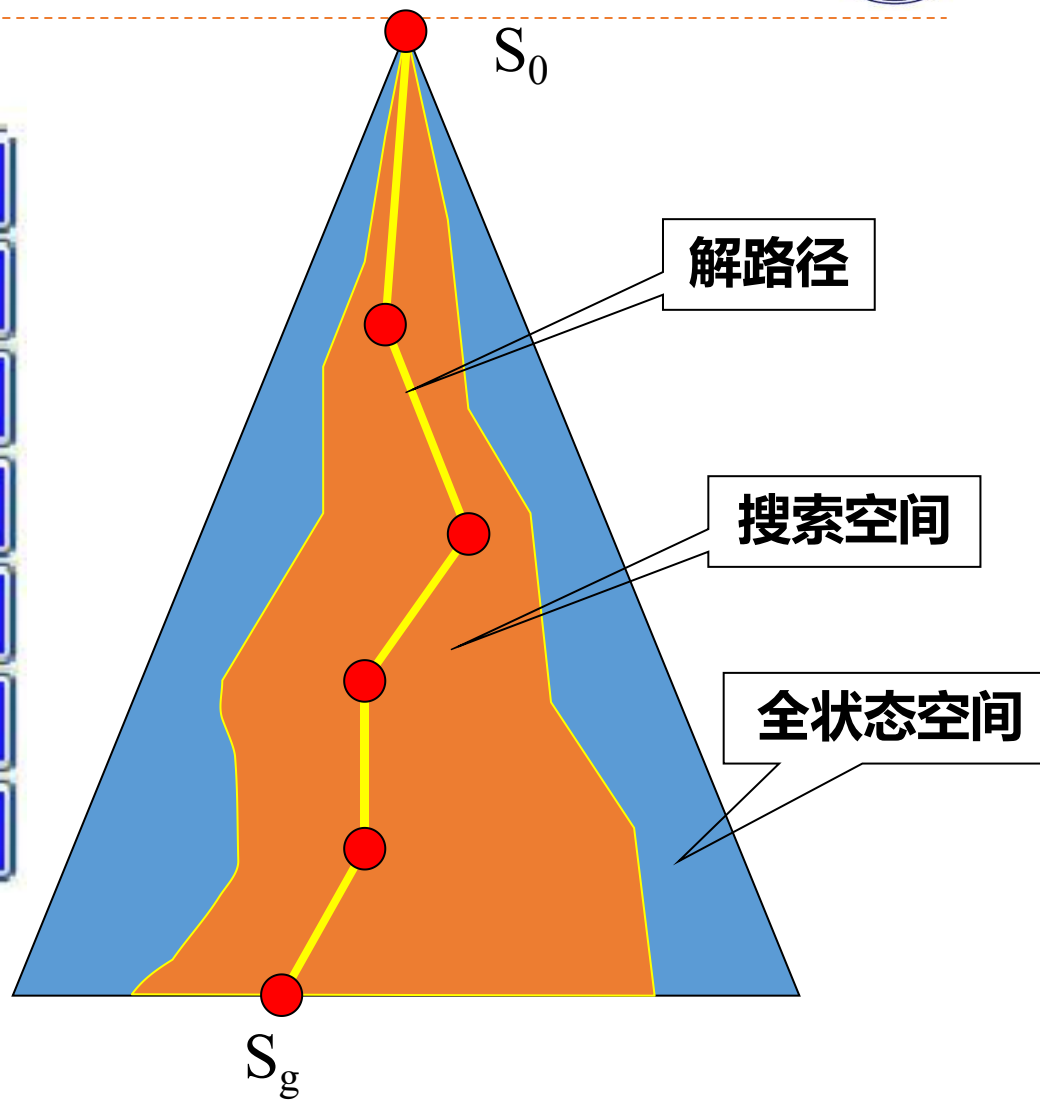
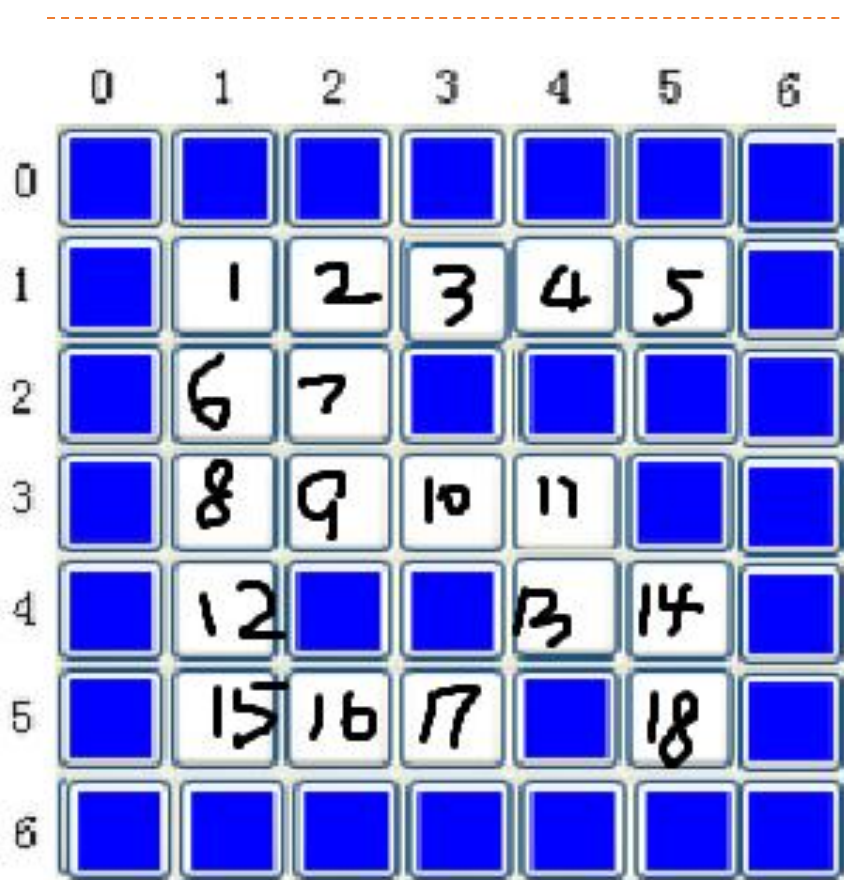
搜索算法是利用计算机的高性能来有目的地穷举一个问题的部分或所有的可能情况，从而求出问题的解的一种方法。很多问题无法根据某种确定的计算法则来求解，可以利用搜索技术求解。

搜索过程实际上是根据初始条件和扩展规则构造一棵状态集合，并在集合中寻找符合目标状态节点的过程。

---



# 1.1 基本概念





# 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





## 1.2 搜索算法常见分类

---

- **枚举算法**: 最直接的搜索方法, 列举问题的所有状态. 然后从中找出问题的解
  - **深度优先搜索**: 从初始结点开始扩展, 按照某中顺序不断的向下扩展, 直到找到目标状态或者是无法继续扩展
  - **广度优先搜索**: 从初始状态开始, 通过规则来生成第一层结点, 同时检查生成结点中是否有目标结点. 若没有则生成下一层接点, 并检查是否有目标结点...
- 





深度优先搜索（Depth First Search, DFS），简称深搜，其状态“退回一步”的顺序符合“后进先出”的特点，所以采用“栈”存储状态。深搜适用于要求所有解方案的题目。

深搜可以采用直接递归的方法实现，其算法框架如下：



# 1.3 深度优先搜索的算法框架

```
void dfs(int dep, 参数表 ){  
    自定义参数 ;  
    if( 当前是目标状态 ){  
        输出解或作相关处理 ;  
    }else  
    {  
        for(i = 1; i <= 状态的拓展可能数 ; i++)  
            if( 满足条件, 第 i 种状态拓展可行 ){  
                维护自定义参数 ;  
                dfs(dep+1, 参数表 );  
            }  
    }  
}
```

	0	1	2	3	4	5	6
0	■	■	■	■	■	■	■
1	■	1	2	3	4	5	■
2	■	6	7	■	■	■	■
3	■	8	9	10	11	■	■
4	■	12	■	■	13	14	■
5	■	15	16	17	■	18	■
6	■	■	■	■	■	■	■



## 2.1 [3369] 全排列问题

输出自然数 $1\sim n$ 所有不重复的排列，即 $n$ 的全排列，要求所产生的任一数字序列中不允许出现重复数字。

[输入格式]

$$1 \leq n \leq 9$$

[输出格式]

由 $1\sim n$ 组成的所有不重复的数字序列。每行一个序列

[输入样例]

3

[输出样例]

1 2 3

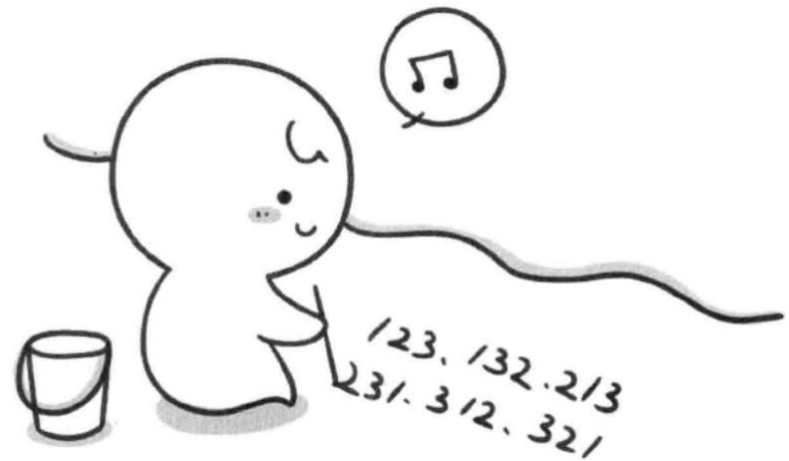
1 3 2

2 1 3

2 3 1

3 1 2

3 2 1





# 暴力枚举

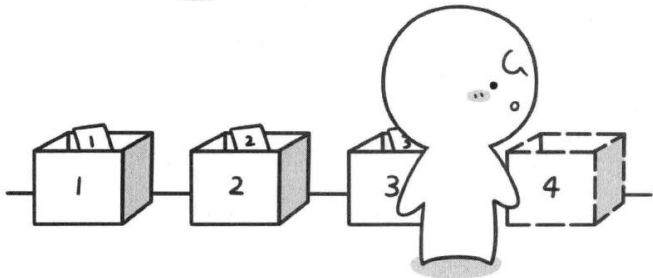
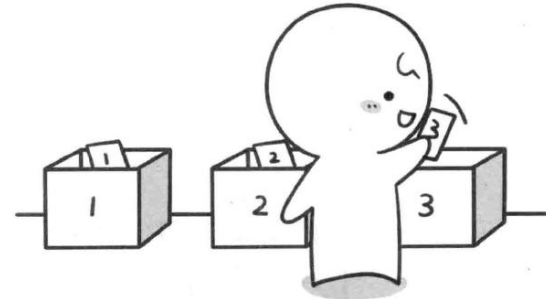
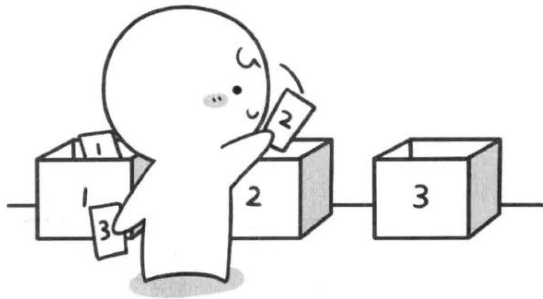
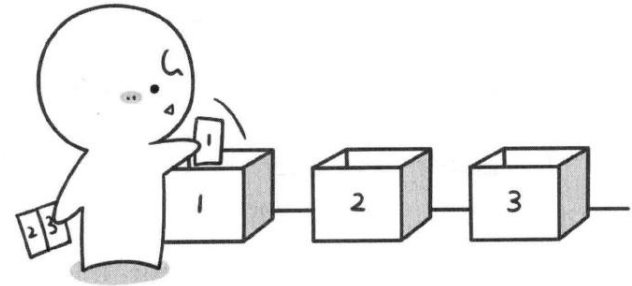
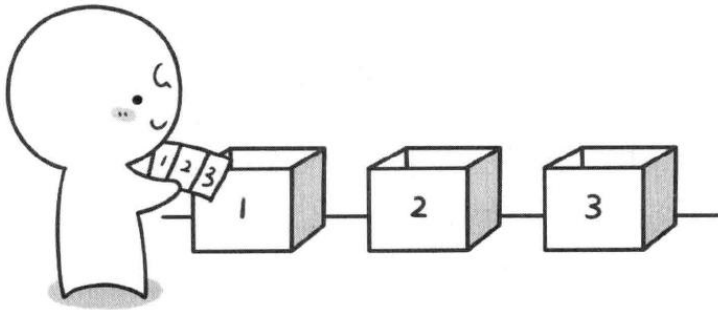


```
1  for(int a=1;a<=3;a++)
2      for(int b=1;b<=3;b++)
3          for(int c=1;c<=3;c++)
4              if(a!=b&&a!=c&&b!=c)
5                  printf("%d%d%d\n",a,b,c);
```

```
1  for(int a=1;a<=4;a++)
2      for(int b=1;b<=4;b++)
3          for(int c=1;c<=4;c++)
4              for(int d=1;d<=4;d++)
5                  if(a!=b&&a!=c&&b!=c&&...)
6                      printf("%d%d%d%d\n",a,b,c,d);
```

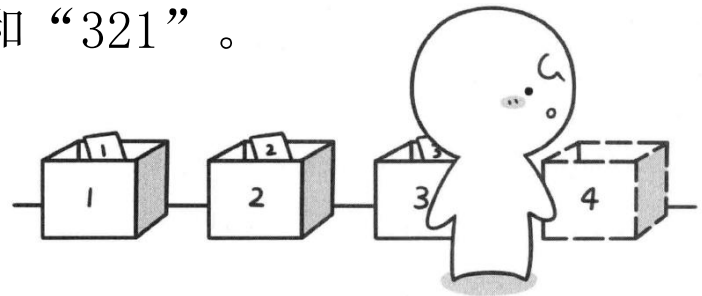
## a. 过程模拟

现在需要将这3张扑克牌分别放到3个盒子里面，并且每个盒子有且只能放一张扑克牌，那么一共有多少种不同的放法呢？



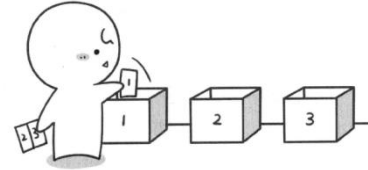
## b. 思路分析

产生了一种排列之后小哼需要立即返回。现在小哼需要退一步重新回到3号盒子面前，需要取回之前放在3号盒子中的扑克牌，再去尝试看看还能否放别的扑克牌，从而产生一个新的排列。于是小哼取回了3号扑克牌。当小哼再想往3号盒子放别的扑克牌的时候，**却发现手中仍然只有3号扑克牌，没有别的选择。于是小哼不得不再往回退一步，回到2号盒子面前。**小哼回到2号盒子后，收回了2号扑克牌。现在小哼手里面有两张扑克牌了，分别是2号和3号扑克牌。按照之前约定的顺序，现在需要往2号盒子中放3号扑克牌（上一次放的是2号扑克牌）。放好之后小哼又向后走一步，再次来到了3号盒子面前。小哼再次来到3号盒子后，将手中仅剩的2号扑克牌放入了3号盒子。又来到4号盒子面前。当然了，这里并没有4号盒子。此时又产生了一个新的排列“**132**”。接下来按照刚才的步骤去模拟，便会依次生成所有排列：“213”、“231”、“3 12”和“321”。



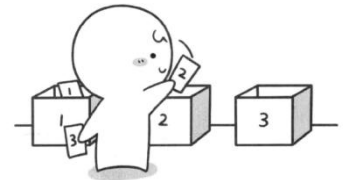
## c. 步骤

### c.1 往第step号盒子放扑克牌



```
1 for(i=1;i<=n;i++)
2 {
3     a[step]=i; //将i号扑克牌放到step号盒子中
4 }
```

### c.2 往第step号盒子牌前判断该牌是否存在并记



```
1 for(i=1;i<=n;i++)
2     .....
3     if(book[i]==0) //如果i号扑克牌还在手上
4     {
5         a[step]=i; //将i号扑克牌放到step号盒子中
6         book[i]=1; //i号扑克牌已经不在手中
7     }
8 }
```

## c. 步骤

### C.3 进入下一轮搜索

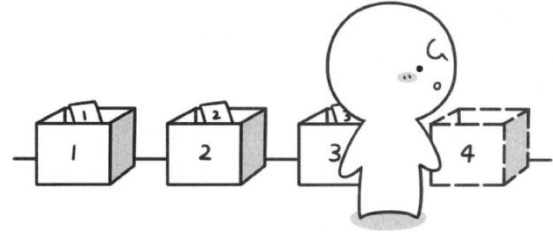
```
24 |  
25 | □  
26 |  
27 | □  
28 |  
29 |  
30 |  
31 |  
32 |  
33 |
```

```
for(i=1;i<=n;i++)  
{//如果未到达边界, 就开始尝试每一种可能  
    if(book[i]==0)//如果i号扑克牌还在手上  
    {  
        a[step]=i;//将i号扑克牌放到step号盒子中  
        book[i]=1;//i号扑克牌已经不在手中  
        dfs(step+1);//处理下一个盒子, 即“下一步和这一步是一样的”  
        book[i]=0;//收回放入的扑克牌, 保证正常返回  
    }  
}
```

book[i]=0这条语句非常重要, 这句话的作用是将小盒子中的扑克牌收回, 因为在一次摆放尝试结束返回的时候, 如果不把刚才放入小盒子中的扑克牌收回, 那将无法再进行下一次摆放。

## c. 步骤

### C.4完成一轮搜索 (到达目的状态)



```
14  if(step==n+1)//判断边界, 如果到了第n+1个盒子, 就说明前n个盒子都已经放好
15  {
16      for(i=1;i<=n;i++)
17      {
18          printf("%d",a[i]); //打印一个排列
19      }
20      printf("\n");
21      return;//返回上一层
22  }
```

# c. 步骤

```
36 void dfs1(int k){
37     for(int i=1;i<=n;i++){
38         if(b[i]==0){
39             a[k]=i,b[i]=1;
40             dfs2(k+1);
41             b[i]=0;
42         }
43     }
44 }
45 }
```

```
25 void dfs2(int k){
26     for(int i=1;i<=n;i++){
27         if(b[i]==0){
28             a[k]=i,b[i]=1;
29             dfs3(k+1);
30             b[i]=0;
31         }
32     }
33 }
34 }
```

```
5 void dfs4(int k){
6     if(k==n+1){
7         for(int i=1;i<=n;i++)
8             printf("%d ",a[i]);
9         printf("\n");
10        return;
11    }
12 }
```

```
14 void dfs3(int k){
15     for(int i=1;i<=n;i++){
16         if(b[i]==0){
17             a[k]=i,b[i]=1;
18             dfs4(k+1);
19             b[i]=0;
20         }
21     }
22 }
23 }
```





# DFS函数

```
11 void dfs(int step)
12 {
13     int i;
14     if(step==n+1)//判断边界, 如果到了第n+1个盒子, 就说明前n个盒子都已经放好
15     {
16         for(i=1;i<=n;i++)
17         {
18             printf("%d",a[i]); //打印一个排列
19         }
20         printf("\n");
21         return;//返回上一层
22     }
23
24     for(i=1;i<=n;i++)
25     { //如果未到达边界, 就开始尝试每一种可能
26         if(book[i]==0)//如果i号扑克牌还在手上
27         {
28             a[step]=i;//将i号扑克牌放到step号盒子中
29             book[i]=1;//i号扑克牌已经不在手中
30             dfs(step+1);//处理下一个盒子, 即“下一步和这一步是一样的”
31             book[i]=0;//收回放入的扑克牌, 保证正常返回
32         }
33     }
34     return;//循环结束也返回上一层
35 }
```



# 填空



```
1 void dfs(int step)
2 {
3     int i;
4     if(____ 1 ____)//判断边界
5     {
6         for(i=1;i<=n;i++)
7             printf("%d ",a[i]); //打印一个排列
8         printf("\n");
9         return;//返回上一层
10    }
11    for(i=1;i<=n;i++)
12    { //如果未到达边界, 就开始尝试每一种可能
13        if(____ 2 ____)//如果i号扑克牌还在手上
14        {
15            ____ 3 ____; //将i号扑克牌放到step号盒子中
16            book[i]= ____ 4 ____; //i号扑克牌已经不在手中
17            dfs(____ 5 ____); //处理下一个盒子, 即“ 下一步和这一步是一样的”
18            book[i]= ____ 7 ____; //收回放入的扑克牌, 保证正常返回
19        }
20    }
21    return;//循环结束也返回上一层
22 }
```

# 代码



```
1  #include <stdio.h>
2  #include <stdlib.h>
3  int a[10],book[10],n;
4  //a数组表示盒子, book数组为标记数组, n为数字个数
5  void dfs(int step);
6  int main()
7  {
8      scanf("%d",&n);
9      dfs(1);
10     return 0;
11 }
12 void dfs(int step)
13 {
37
11 void dfs(int step)
12 {
13     int i;
14     if(step==n+1)//判断边界, 如果到了第n+1个盒子, 就说明前n个盒子都已经放好
15     {
16         for(i=1;i<=n;i++)
17         {
18             printf("%d",a[i]); //打印一个排列
19         }
20         printf("\n");
21         return;//返回上一层
22     }
23
24     for(i=1;i<=n;i++)
25     { //如果未到达边界, 就开始尝试每一种可能
26         if(book[i]==0)//如果i号扑克牌还在手上
27         {
28             a[step]=i;//将i号扑克牌放到step号盒子中
29             book[i]=1;//i号扑克牌已经不在手中
30             dfs(step+1);//处理下一个盒子, 即“ 下一步和这一步是一样的”
31             book[i]=0;//收回放入的扑克牌, 保证正常返回
32         }
33     }
34     return;//循环结束也返回上一层
35 }
```



## 3.1 [2997]数塔问题

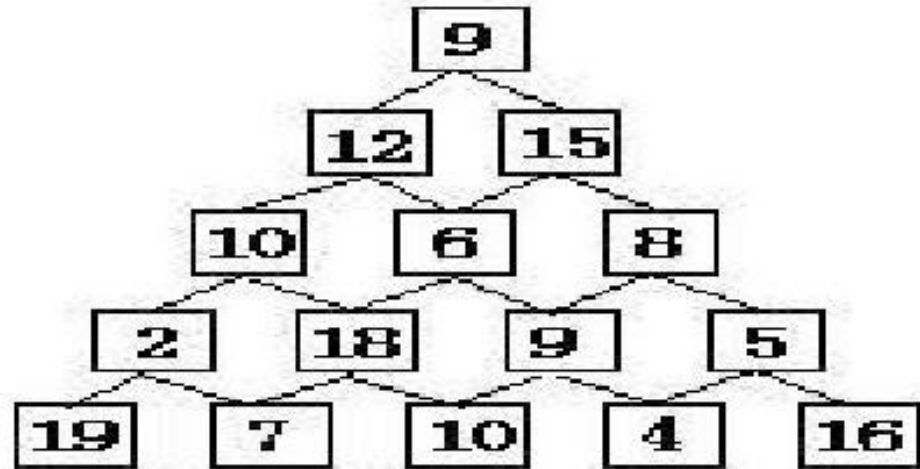
有如下所示的数塔，要求从顶层走到底层，若每一步只能走到相邻的结点，则经过的结点的数字之和最大是多少？  
输入数据首先包括一个整数C,表示测试实例的个数，每个测试实例的第一行是一个整数N( $1 \leq N \leq 20$ )，表示数塔的高度，接下来用N个数字表示数塔，其中第i行有个i个整数，且所有的整数均在区间[0,99]内。

对于每个测试实例，输出可能得到的最大和，每个实例的输出占一行。

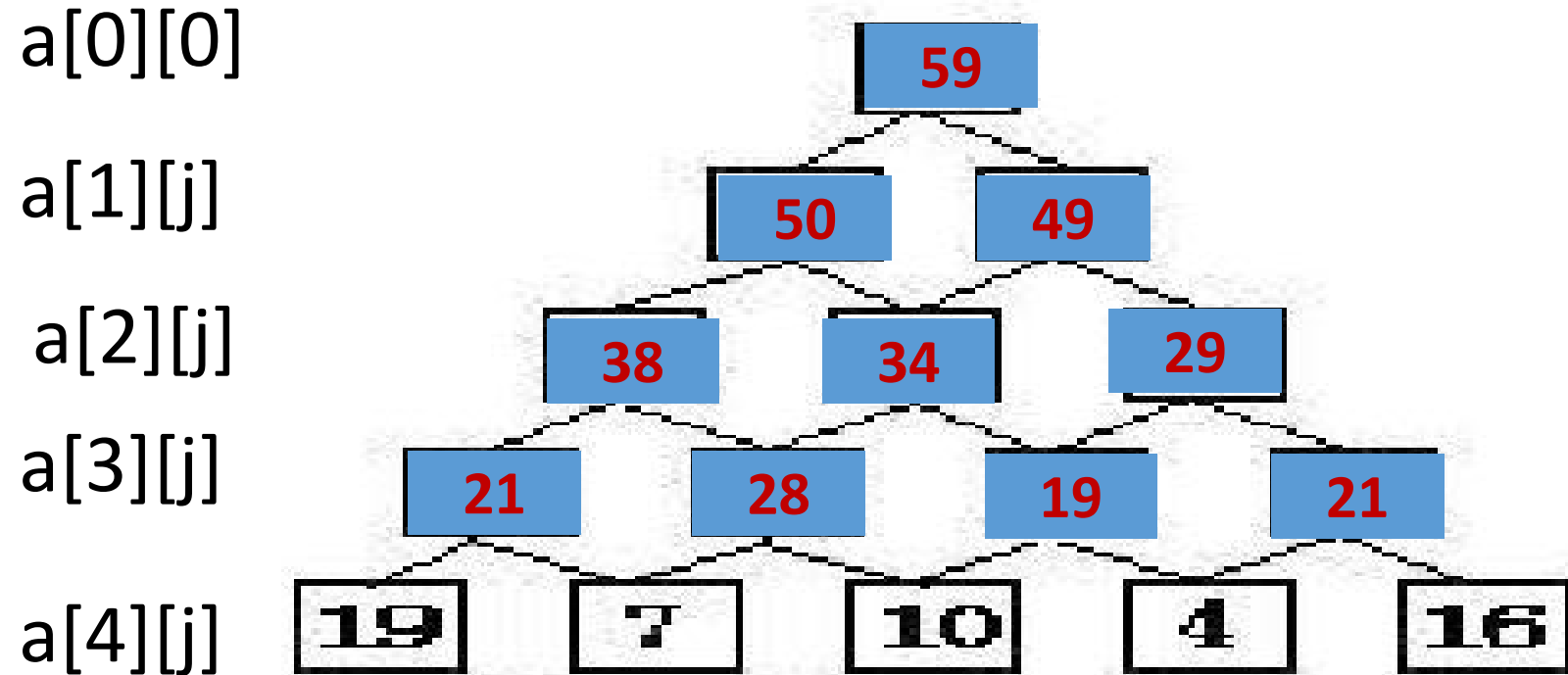
样例输入

1

5 9 12 15 10 6 8 2 18 9 5 19 7 10 4 16



# a. 数塔问题(递推法)





## a. 数塔问题(递推法)

```
1  #include<bits/stdc++.h>
2  using namespace std;
3  int main(){
4      int a[105][105],n;
5      memset(a,0,sizeof(a));
6      cin>>n;
7      for(int i=0;i<n;i++)
8          for(int j=0;j<=i;j++)
9              cin>>a[i][j];
10     for(int i=n-1;i>=0;i--)
11         for(int j=0;j<=i;j++)
12             a[i][j]=a[i][j]+max(a[i+1][j],a[i+1][j+1]);
13     cout<<"max="<<a[0][0]<<endl;
14     return 0;
15 }
```



## b. 数塔问题(DFS)

---

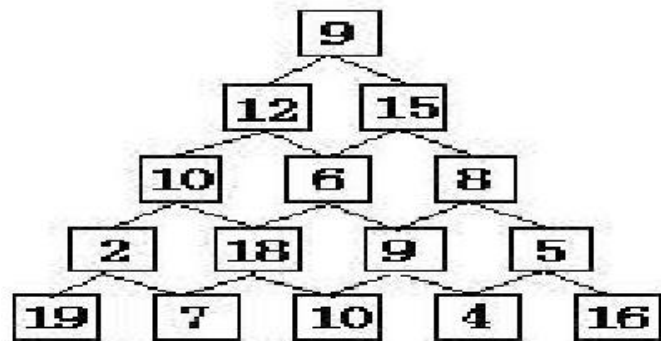
问题要求的是从最高点按照规则走到最低点的路径的最大的权值和，路径起点终点固定，走法规则明确，可以考虑用搜索来解决。

定义递归函数void Dfs(int x,int y,int Curr)，其中x,y表示当前已从(1,1)走到(x,y)，目前已走路径上的权值和为Curr。

当x=N时，到达递归出口，如果Curr比Ans大，则把Ans更新为Curr；否则向下一行两个位置行走，即递归执行

Dfs(x+1,y,Curr+A[x+1][y])和

Dfs(x+1,y+1,Curr+A[x+1][y+1])。



## b. 数塔问题(DFS)代码

```
1  #include<iostream>
2  using namespace std;
3  //5
4  //9 12 15 10 6 8 2 18 9 5 19 7 10 4 16
5  int a[25][25];
6  int n,ans;
7  void dfs(int x,int y, int curr){
8      // (x,y)表示的位置, curr表示的是当前位置对应的权重和
9      if(x==n){ // 到达目标, 也就是最后一行的时候
10         if(curr>ans) ans=curr; // 求最大的权值和
11         return; // 1求解完一轮, 返回到上一层
12     }
13     dfs(x+1,y,curr+a[x+1][y]);
14     dfs(x+1,y+1,curr+a[x+1][y+1]);
15 }
16 int main(){
17     cin>>n; // scanf("%d",&n);
18     for(int i=1;i<=n;i++)
19         for(int j=1;j<=i;j++)
20             cin>>a[i][j];
21     ans=0;
22     dfs(1,1,a[1][1]);
23     cout<<"max="<<ans<<endl; // printf("max=%d",ans);
24     return 0;
25 }
```

该方法实际上是把所有路径都走了一遍, 由于每一条路径都是由 $N-1$ 步组成, 每一步有“左”、“右”两种选择, 因此路径总数为 $2^{N-1}$ , 所以该方法的时间复杂度为 $O(2^{N-1})$ , 超时。



# 填空

```
2 using namespace std;
3 int a[25][25];
4 int n,ans;
5 void dfs(int x,int y, int curr){
6     //(x,y)表示的位置, curr表示的是当前位置对应的权重和
7     if(____1____){//到达目标, 也就是最后一行的时候
8         if(curr>ans)ans=curr;//求最大的权值和
9         return;//1求解完一轮, 返回到上一层
10    }
11    dfs(____2____);
12    dfs(____3____);
13 }
14 int main(){
15     cin>>n;//scanf("%d",&n);
16     for(int i=1;i<=n;i++)
17         for(int j=1;j<=i;j++)
18             cin>>a[i][j];
19     ans=0;
20     dfs(____4____);
21     cout<<"max="<<ans<<endl;//printf("max=%d",ans);
22     return 0;
23 }
```



## c. 数塔问题(记忆化搜索)


---

普通DFS之所以会超时，是因为进行了重复搜索，如样例中从(1, 1)到(3, 2)有“左右”和“右左”两种不同的路径，也就是说搜索过程中两次到达(3, 2)这个位置，那么从(3, 2)走到终点的每一条路径就被搜索了两次，我们完全可以在第一次搜索(3, 2)到终点的路径时就记录下(3, 2)到终点的最大权值和，下次再次来到(3, 2)时就可以直接调用这个权值避免重复搜索。我们把这种方法称为记忆化搜索。

记忆化搜索需要对方法一中的搜索进行改装。由于需要记录从一个点开始到终点的路径的最大权值和，因此我们重新定义递归函数Dfs。

定义Dfs(x, y)表示从(x, y)出发到终点的路径的最大权值和，答案就是Dfs(1, 1)。计算Dfs(x, y)时考虑第一步是向左还是向右，我们把所有路径分成两大类：

---



# 数塔问题(记忆化搜索)

---

①第一步向左：那么从 $(x, y)$ 出发到终点的这类路径就被分成两个部分，先从 $(x, y)$ 到 $(x+1, y)$ 再从 $(x+1, y)$ 到终点，第一部分固定权值就是 $A[x][y]$ ，要使得这种情况的路径权值和最大，那么第二部分从 $(x+1, y)$ 到终点的路径的权值和也要最大，这一部分与前面的 $Dfs(x, y)$ 的定义十分相似，仅仅是参数不同，因此这一部分可以表示成 $Dfs(x+1, y)$ 。综上，第一步向左的路径最大权值和为 $A[x][y]+Dfs(x+1, y)$ ；

②第一步向右：这类路径要求先从 $(x, y)$ 到 $(x+1, y+1)$ 再从 $(x+1, y+1)$ 到终点，分析与上面一样，这类路径最大权值和为 $A[x][y]+Dfs(x+1, y+1)$ ；

为了避免重复搜索，我们开设全局数组 $F[x][y]$ 记录从 $(x, y)$ 出发到终点路径的最大权值和，一开始全部初始化为-1表示未被计算过。在计算 $Dfs(x, y)$ 时，首先查询 $F[x][y]$ ，如果 $F[x][y]$ 不等于-1，说明 $Dfs(x, y)$ 之前已经被计算过，直接返回 $F[x][y]$ 即可，否则计算出 $Dfs(x, y)$ 的值并存储在 $F[x][y]$ 中。

---



```

1  #include <iostream>
2  #include <algorithm>
3  using namespace std;
4  const int MAXN = 505;
5  int A[MAXN][MAXN], F[MAXN][MAXN], N;
6  int Dfs(int x,int y)
7  {
8      if (F[x][y]== -1) //没有搜索过
9      {
10         if (x==N)F[x][y]=A[x][y];
11         else F[x][y]=A[x][y]+max(Dfs(x+1,y),Dfs(x+1,y+1));
12     }
13     return F[x][y];//搜索过就直接返回,
14     //相当于返回根据原来记忆的值
15 }

```

由于 $F[x][y]$ 对于每个合法的 $(x,y)$ 都只计算过一次，而且计算是在 $O(1)$ 内完成的，因此时间复杂度为 $O(N*N)$ 。可以通过本题。

```

18 int main() {
19     cin >> N;
20     for(int i = 1;i <= N;i ++)
21         for(int j = 1;j <= i;j ++)
22             cin >> A[i][j];
23     for(int i = 1;i <= N;i ++)
24         for(int j = 1;j <= i;j ++)
25             F[i][j] = -1;
26     Dfs(1,1);
27     cout << "max="<<F[1][1] << endl;
28     return 0;
29 }

```

# 填空

```
6  const int MAXN = 505;
7  int A[MAXN][MAXN], F[MAXN][MAXN], N;
8  int Dfs(int x, int y)
9  {
10     if ( _____ 1 _____ ) //没有搜索过
11     {
12         if (x==N) _____ 2 _____ ;
13         else F[x][y]=A[x][y]+ _____ 3 _____ );
14     }
15     return _____ ; //搜索过就直接返回,
16     //相当于返回根据原来记忆的值
17 }
```



## 3.2 [4020] 体积

### 【问题描述】

给出  $n$  件物品，每件物品有一个体积  $V_i$ ，求从中取出若干件物品能够组成的不同的体积和有多少种可能。

### 【输入格式】

第 1 行 1 个正整数，表示  $n$ 。

第 2 行  $n$  个正整数，表示  $V_i$ ，每两个数之间用一个空格隔开。

### 【输出格式】

一行一个数，表示不同的体积和有多少种可能。

样例输入

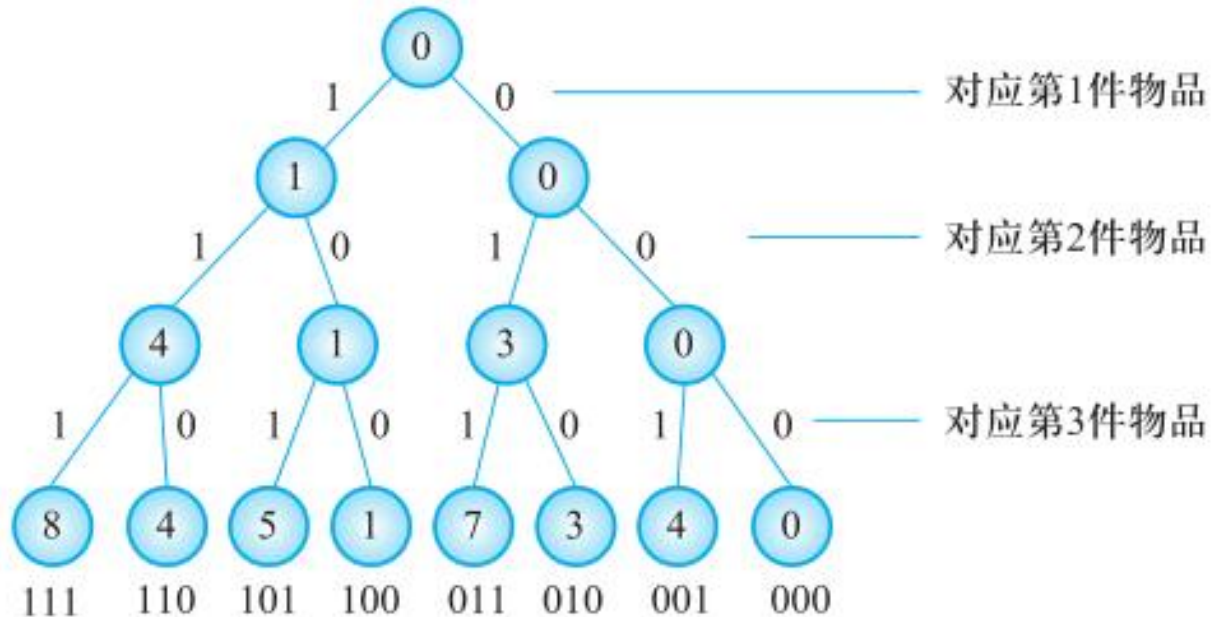
3 1 3 4

样例输出

6

# 问题分析

典型的搜索问题，对于每一件物品只有取和不取两种情况，给如如下搜索树，表示搜索的状态。





# 问题分析

假设ans表示目前方案的体积和, 初始化为0。设计一个数组  $a[i]$ , 表示体积*i*是否出现过, 初始化为0。

先考虑取第一件物品, 等到这个情况处理完, 再退回到这一步处理不取第一件物品。所以, 对于样例来说, 这时 $V=1$ ;

然后再递归处理第二件物品, 同样先处理取的情况, 处理完再回来处理不取的情况, 所以此时再递归处理第三件物品, 同样先处理取的情况, 此时 $V=8$ , 三件物品已经处理完毕, 得到一个体积8,  $a[8]=\text{true}$ . 再考虑不取第三件物品的情况, 此时得到一个体积4,  $a[4]=\text{true}$ 。处理完第三件物品的两种状态, 退回到上一步处理第二件物品不取的情况, 此时体积 $V=1$ , 再往前递归一步, 考虑第三件物品的取与不取, 此时分别得到2个体积和 $V=5$  ,  $V=1$ ,  $a[5]=\text{true}$ ,  $a[1]=\text{true}$ . . 再连续退回二次, 回到第一件物品不取的情况, 分别处理, 最后, 统计 $a[i]$ 为true的个数。

# 代码



```
1  #include<iostream>
2  using namespace std;
3  int v[21]; // 每件物品的体积
4  bool a[1001]; // n ≤ 20, vi ≤ 50, 最大和为1000
5  int n; // n个数
6  void dfs(int dep, int sum){ // sum表示和, dep表示取多少个数
7      if(dep > n){
8          a[sum] = true;
9          return;
10     }
11     dfs(dep+1, sum+v[dep]); // 取当前的数
12     dfs(dep+1, sum); // 不取当前的数
13 }
14 int main(){
15     cin >> n;
16     for(int i=1; i<=n; i++) cin >> v[i];
17     dfs(1, 0);
18     int ans=0;
19     for(int i=1; i<=1000; i++)
20         if(a[i]) ans++;
21     cout << ans << endl;
22 }
```







# 填空

```
1  #include<iostream>
2  using namespace std;
3  int v[21];
4  bool a[1001]; //n ≤ 20, V i ≤ 50, 最大和为1000
5  int n; //n个数
6  void dfs(int dep, int sum){ //sum表示和, dep表示取多少个数
7      if(dep > n){
8          _____ 1 _____;
9          return;
10     }
11     dfs(_____ 2 _____); //取当前的数
12     dfs(_____ 3 _____); //不取当前的数
13 }
14 int main(){
15     cin >> n;
16     for(int i=1; i<=n; i++) cin >> v[i];
17     dfs(_____ 5 _____);
18     int ans=0;
19     for(int i=1; i<=1000; i++)
20         if(a[i]) ans++;
21     cout << ans << endl;
22 }
```



## 3.3 [2747]判断元素是否存在

有一个集合M是这样生成的：

- (1) 已知 $k$ 是集合M的元素；
- (2) 如果 $y$ 是M的元素，那么， $2y+1$ 和 $3y+1$ 都是M的元素；
- (3) 除了上述二种情况外，没有别的数能够成为M的一个元素。

问题：任意给定 $k$ 和 $x$ ，请判断 $x$ 是否是M的元素。这里的 $k$ 是无符号整数， $x$ 不大于100000，如果是，则输出YES，否则，输出NO。

输入整数  $k$  和  $x$ ，逗号间隔。输入整数  $k$  和  $x$ ，逗号间隔。

如果是，则输出 YES，否则，输出NO。





## 3.3 [2747]判断元素是否存在

```
1  #include<iostream>
2  using namespace std;
3  int dfs(int x,int y)
4  {
5      if(x==y)return 1;
6      if(x>y)return 0;
7      if(dfs(x*2+1,y))return 1;
8      if(dfs(x*3+1,y))return 1;
9      return 0;
10 }
11 int main()
12 {
13     int a,b;
14     scanf("%d,%d",&a,&b);
15     if(dfs(a,b))cout<<"YES"<<endl;
16     else cout<<"NO"<<endl;
17     return 0;
18 }
```



## 3.4[2757] 移动线路

X桌子上有一个 $m$ 行 $n$ 列的方格矩阵，将每个方格用坐标表示，行坐标从下到上依次递增，列坐标从左至右依次递增，左下角方格的坐标为 $(1,1)$ ，则右上角方格的坐标为 $(m,n)$ 。

小明是个调皮的孩子，一天他捉来一只蚂蚁，不小心把蚂蚁的右脚弄伤了，于是蚂蚁只能向上或向右移动。小明把这只蚂蚁放在左下角的方格中，蚂蚁从

左下角的方格中移动到右上角的方格中，每步移动一个方格。蚂蚁始终在方格矩阵内移动，请计算出不同的移动路线的数目。

对于1行1列的方格矩阵，蚂蚁原地移动，移动路线数为1；对于1行2列（或2行1列）的方格矩阵，蚂蚁只需一次向右（或向上）移动，移动路线数也为1.....对于一个2行3列的方格矩阵，





## 3.4[2757] 移动线路

```
1  #include<iostream>
2  using namespace std;
3  int m,n;
4  int sum=0;
5  void dfs(int x,int y){
6      if(x>m||y>n)return; //不出边界
7      if(x==m&& y==n){ //到达终点
8          sum=sum+1;
9          return;
10     }
11     dfs(x+1,y); //搜索
12     dfs(x,y+1);
13 }
14 int main(){
15     cin>>m>>n;
16     dfs(1,1);
17     cout<<sum<<endl;
18     return 0;
19 }
```





## 2.4 回溯搜索

回溯搜索是深度优先搜索（DFS）的一种，DFS的一种改进，是更实用的搜索求解方法。

### 与DFS的区别

扩展结点时，DFS将所有子结点全部扩展出来，再选取最新的一个结点进行扩展。

回溯搜索只扩展所有子结点的其中一个，然后再以这一子结点去扩展下一个子结点，当某个结点不能再扩展出新结点的时候，就删除这个结点，用其父结点来扩展新的结点。

DFS最后寻找解的路径，回溯搜索路径即解路径

### 回溯法的最大优点

占用内存少，只需存取当前的一条搜索路径。

### 适用范围

要求所有解方案的问题

试探性求解问题中





## 2.4.1 回溯搜索概念


---

回溯法也称试探法。

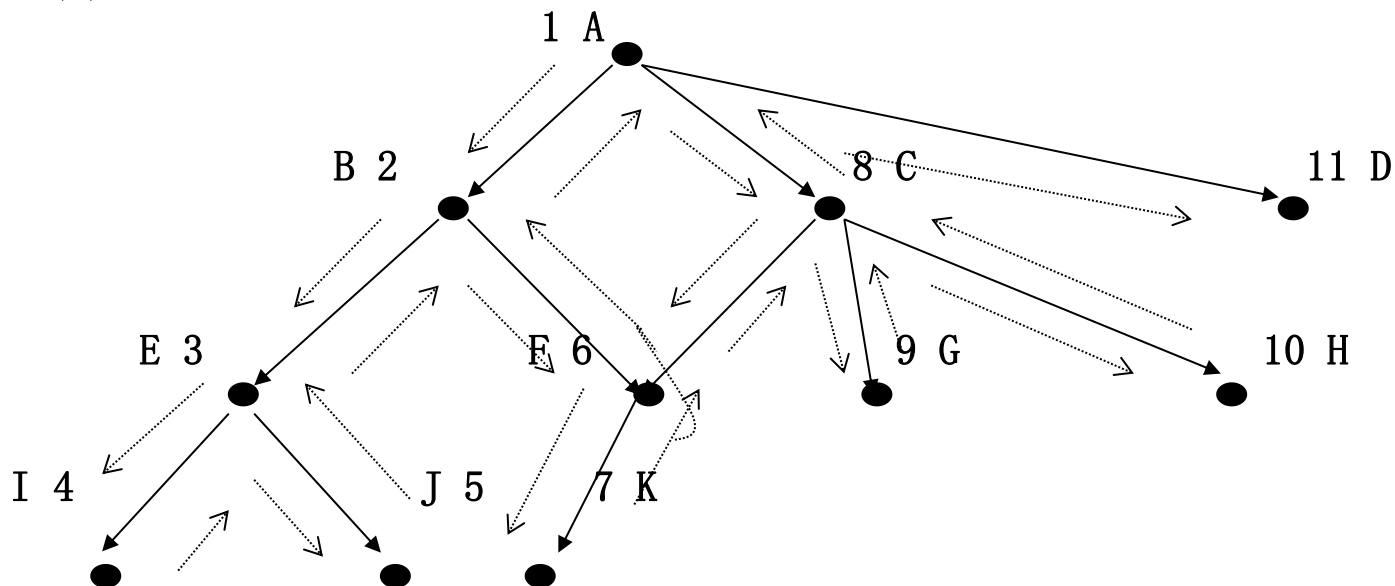
它的基本思想是：从问题的某一种状态（初始状态）出发，搜索从这种状态出发所能达到的所有“状态”，当一条路走到“尽头”的时候（不能再前进），再后退一步或若干步，从另一种可能“状态”出发，继续搜索，直到所有的“路径”（状态）都试探过。

这种不断“前进”、不断“回溯”寻找解的方法，就称作“回溯法”。

---



## 回溯策略 回溯搜索的示意



状态空间中回溯搜索 图中虚线箭头的方向表明了搜索的**轨迹**，  
结点边的数字表明了被搜索到的**次序**






# 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}
                }
}
```

---



## [3302]组合的输出

---

排列与组合是常用的数学方法，其中组合就是从 $n$ 个元素中抽出 $r$ 个元素(不分顺序且 $r \leq n$ )，我们可以简单地将 $n$ 个元素理解为自然数 $1, 2, \dots, n$ ，从中任取 $r$ 个数。

现要求你用递归的方法输出所有组合。

例如 $n=5, r=3$ ，所有组合为：

1 2 3   1 2 4   1 2 5   1 3 4   1 3 5   1 4 5   2 3 4   2 3 5   2  
4 5   3 4 5

# 分析

---

1	2	3
1	2	4
1	2	5
1	3	4
1	3	5
1	4	5
2	3	4
2	3	5
2	4	5
3	4	5

组合问题与排列问题很相近，差别在于组合数不能重复。如何保证不重复的，观察可以得到，对于组合数而言，如果当前元素为 $a[i]$ ，后一个元素 $a[i+1] > a[i] + 1$ ，如何保证？

## 2.4.2 [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

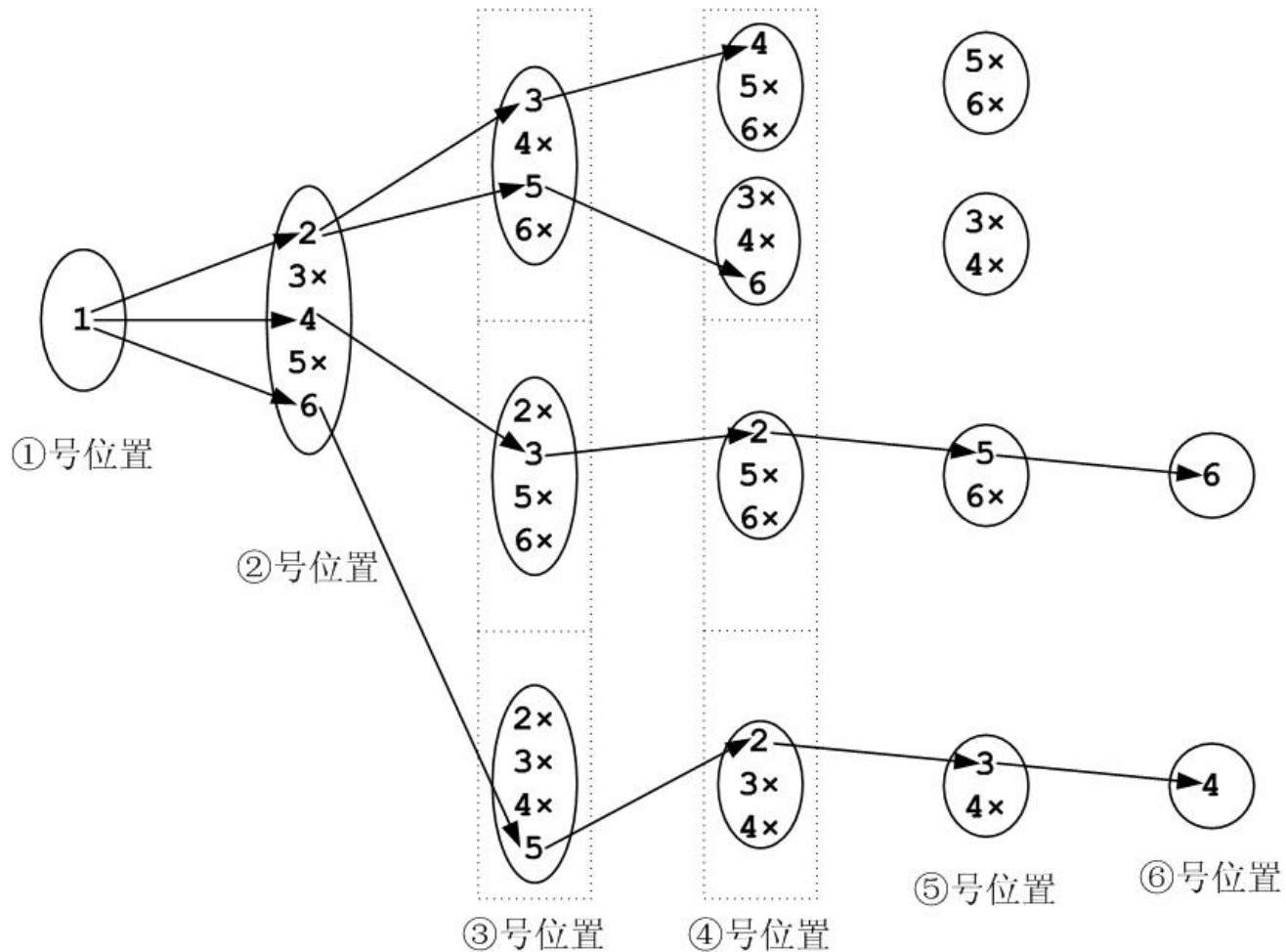
---

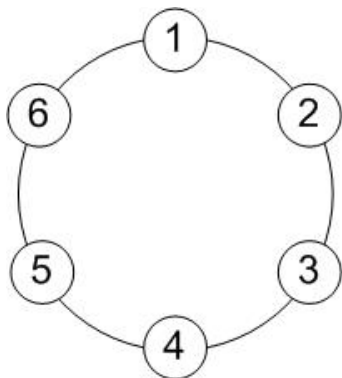
非常明显，这是一道回溯的题目。从1开始，每个空位有20种可能，只要填进去的数合法：与前面的数不相同；与左边相邻的数的和是一个素数。第20个数还要判断和第1个数的和是否素数。

## 【算法流程】

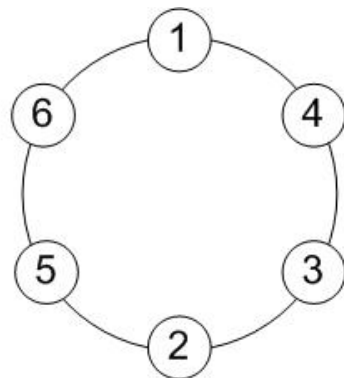
- 1、数据初始化；
  - 2、递归填数：判断第*i*个数填入是否合法；
    - A、如果合法：填数；判断是否到达目标（20个已填满）：是，打印结果；不是，递归填下一个；
    - B、如果不合法：选择下一种可能；
- 
- 

# 搜索策略 (n=6)





(a) 环上的 $n$ 个位置



(b) 在 $n$ 个位置放置自然数 $1\sim n$

在①号位置上放置数1。在②号位置上可供选择的数为 $2\sim 6$ ，其中3和5是不可行的，对2、4和6一一试探，先试探2。

②号位置上放置2以后，③号位置上可供放置的数为 $3\sim 6$ ，其中4和6是不可行的，对3和5也是一一试探，先试探3。



---

③号位置上放置3以后，④号位置上可供放置的数为4~6，其中只有4是可行的，所以放置4。这时可供⑤号位置上放置的数为5和6，均不可行。这些方案都不可行。

再考虑③号位置上放置5，④号位置上只能放置6，此后⑤号位置上放置3和4，均不可行。这些方案都排除。

再考虑②号位置上放置4，③号位置上只能放置3，④号位置上只能放置2，⑤号位置上只能放置5，⑥号位置只能放置6，这个方案是可行的。

...

如此搜索，直到试探所有方案为止。

---

```
1 //3302 素数环
2
3 #include <iostream>
4 #include<iomanip>
5 #include<cmath>
6 using namespace std;
7 int a[100]={0},b[100]={0};
8 //数组a用来存储环中的每个元素 b表示当前的值i有没有没使用过
9 int n,sum=0;
10 □ int prime(int a,int b){
11     int c=a+b;
12     for(int i=2;i<=sqrt(c);i++ )
13         if(c%i==0)return 0;
14     return 1;
15 }
```

```
18 int search(int k)//k表示处理的位置
19 {
20     if(k==n+1&&prime(a[1],a[n]))sum++;// if (到目的地) 输出解;
21     else{
22         for(int i=2;i<=n;i++){//for (i=1;i<=算符种数;i++)
23             if(prime(i,a[k-1])&&b[i]==0){//if (满足条件)
24                 a[k]=i,b[i]=1;//保存结果
25                 search(k+1);
26                 b[i]=0;//恢复
27             }
28         }
29     }
30 }
31
32 int main()
33 {
34     //memset(a,0,sizeof(a));
35     //memset(b,0,sizeof(b));
36     cin>>n;
37     a[1]=1,b[1]=1;
38     search(2);
39     cout<<sum<<endl;
40     return 0;
41 }
```



# 今天的课程结束啦.....

---



下课了...  
同学们再见!