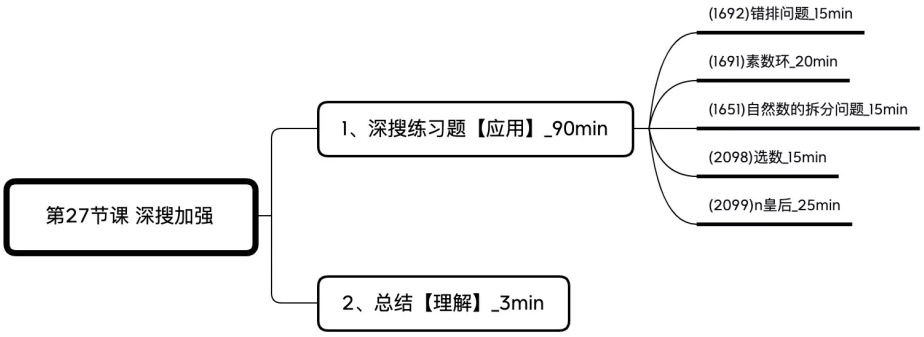
27 深搜加强_讲义



深搜加强练习

例1:(1692)错排问题

【题目描述】
输入n，输出 1~n 做排列的错排的所有方案。错排就是第 i 个位置不能放 i 的排列。

【输入】
一个整数n（n≤9）。

【输出】
按字典序从小到大输出方案。

【输入样例】
4

【输出样例】
2 1 4 3
2 3 4 1
2 4 1 3
3 1 4 2
3 4 1 2
3 4 2 1
4 1 2 3
4 3 1 2
4 3 2 1

分析

与(2094)全排列问题类似，只不过多了一个条件。

在排列型枚举中，枚举每一个填空中所有可能的选项，然后判断这种选项是否合法。如果这个选项合法的话，就填写下一个空，然后继续；如果这个填空中所有的选项都不合法，那就不用继续枚举下去了，而是去尝试更换上一个填空的选择，继续枚举。这种方式被称为**回溯算法**。

```
void dfs(int k){
    if(所有空都填完了){
        判断最优解/记录答案;
        return;
    }
    for(枚举这个空能填的项){
        if(这个选项是合法的){
            记录这个空(保存现场);
            dfs(k+1);
            取消这个空(恢复现场); //回溯算法，常常使用深搜来实现
        }
    }
}
```

答案

```
#include <bits/stdc++.h>
using namespace std;
int a[10],n,b[10];//b数组用来记录某个数是否被使用了
void dfs(int k){
    if(k==n+1){
        for(int i=1;i<=n;i++){
            cout<<a[i]<<" ";
        }
        cout<<'\n';
    }
    for(int i=1;i<=n;i++){
        if(b[i]==0 && k!=i){
            a[k]=i;
            b[i]=1;
            dfs(k+1);
            b[i]=0;
        }
    }
}
int main(){
    cin>>n;
    dfs(1);
    return 0;
}
```

练习1:(1691)素数环

【题目描述】
输入正整数n，把整数1,2,...,n组成一个环，使得相邻两个整数之和均为素数。输出时，从整数1开始逆时针排列。并且多个输出按照字典序从小到大排列。同一个环恰好输出一次。n≤16，保证一定有解。

【输入】
输入n。

【输出】
按字典序从小到大输出所有方案。

【输入样例】
8

【输出样例】
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

分析

属于排列型枚举，在填空时，维护当前空和上一空之和为素数，别忘了数组的首尾之和也要为素数。

答案

```
#include <bits/stdc++.h>
using namespace std;
int n,a[20],b[20]; //b[i]=1表示i已经用了
bool isPrime(int num){ //判断质数函数
    if(num<2) return false;
    for(int i=2;i*i<=num;i++){
        if(num%i==0) return false;
    }
    return true;
}
void dfs(int k){
    if(k==n+1){
        if(isPrime(a[1]+a[n])){
            for(int i=1;i<=n;i++) cout<<a[i]<<" ";
            cout<<endl;
        }
        return;
```

```
    }
    for(int i=2;i<=n;i++){
        if(b[i]==0 && isPrime(i+a[k-1])){
            b[i]=1;
            a[k]=i;
            dfs(k+1);
            b[i]=0;
        }
    }
}
int main(){
    cin>>n;
    a[1]=1;
    dfs(2);
    return 0;
}
```

例2:(2097)自然数的拆分问题

【题目描述】
任何一个大于 1 的自然数 n，总可以拆分成若干个小于 n 的自然数之和。现在给你一个自然数 n，要求你求出 n 的拆分成一些数字的和。每个拆分后的序列中的数字从小到大排序。然后你需要输出这些序列，其中字典序小的序列需要优先输出。

【输入】
待拆分的自然数 n(1≤n≤8)。

【输出】
若干数的加法式子。

【输入样例】
7

【输出样例】
1+1+1+1+1+1+1
1+1+1+1+1+2
1+1+1+1+3
1+1+1+2+2
1+1+1+4
1+1+2+3
1+1+5
1+2+2+2
1+2+4
1+3+3
1+6
2+2+3

2+5
3+4

分析

我们需要先枚举第一个数字，再枚举第二个数字...也就是函数需要一个参数k，表示第k层搜索，枚举第k个数字。
并且在这个过程中我们规定，下一个数字必须大于等于上一个数字。也需要知道上一层搜索剩下了多少数字，决定了下一层搜索的枚举范围。

构造深搜函数时，需要有2个参数：

```
int n,a[15]; //a数组存放每层递归的数字
void dfs(int k, int res){...} //res剩下了多少数字
```

答案

```
#include <bits/stdc++.h>
using namespace std;
int n,a[20];
void dfs(int k,int res){//res是剩下的数的大小，k是第几个数
    if(res==0){
        for(int i=1;i<k;i++){
            cout<<a[i];
            if(i<k-1) cout<<'+';

        }
        cout<<endl;
        return;
    }
    for(int i=a[k-1];i<=res;i++){//至少大于等于上一个数
        if(i<n){//排除n=n这个情形
            a[k]=i;
            dfs(k+1,res-i);
        }
    }
}
int main(){
    cin>>n;
    a[0]=1;//让第一个数从1开始枚举
    dfs(1,n);
    return 0;
}
```

例3:(2098)选数

【题目描述】

已知n 个整数 $x_1,x_2,\cdots,x_n$ ，以及 1 个整数 k（k<n）。从 n 个整数中任选 k 个整数相加，可分别得到一系列的和。例如当 n=4，k=3，4 个整数分别为 3,7,12,19 时，可得全部的组合与它们的和为：

3+7+12=22

3+7+19=29

7+12+19=38

3+12+19=34

现在，要求你计算出和为素数共有多少种。

例如上例，只有一种的和为素数：3+7+19=29。

【输入】

第一行两个空格隔开的整数 $n,k(1\le n\le 20$ ，$k<n)$ 。

第二行 n 个整数，分别为 $x_1,x_2,\cdots,x_n(1\le x_i\le 5\times 10^6)$ 。

【输出】

输出一个整数，表示种类数。

【输入样例】

4 3

3 7 12 19

【输出样例】

1

分析

属于组合型枚举，与(2096)k位异或这道题类似，n个数中选k个。

n个元素	3	7	12	19
选择方案1	1	1	1	0
选择方案2	1	1	0	1
选择方案3	1	0	1	1
选择方案4	0	1	1	1

1代表选，0代表不选。

每次向下层搜索，需要知道当前选择到了哪个数字，还需要知道已经选择了几个数字。

我们可以这样构造深搜函数：

```
int n,k,a[25],f[25],ans; //f数组用来标记数字是否被使用
void dfs(int cur, int cnt){
    if(确定了每个元素选/不选){
        判断最优解/记录答案; //还需要判断已选择的个数cnt是否达到k
        return;
    }
    f[cur]=1;
    dfs(cur+1,cnt+1);
    f[cur]=0;
    dfs(cur+1,cnt);
}
```

答案

```
#include <bits/stdc++.h>
using namespace std;
int n,k,a[25],f[25],ans;
bool isPrime(int x){
    if(x<2) return false;
    for(int i=2;i<=sqrt(x);i++){
        if(x%i==0) return false;
    }
    return true;
}
void dfs(int cur,int cnt){
    if(cur==n+1){
        if(cnt==k){
            int sum=0;
            for(int i=1;i<=n;i++){
                if(f[i]) sum+=a[i];
            }
            if(isPrime(sum)) ans++;
        }
        return;
    }
    f[cur]=1;
    dfs(cur+1,cnt+1);
    f[cur]=0;
    dfs(cur+1,cnt);
}
int main(){
    cin>>n>>k;
    for(int i=1;i<=n;i++) cin>>a[i];
    dfs(1,0);
    cout<<ans;
    return 0;
}
```

以上是第一种写法。我们还可以更为方便地构造深搜函数。

在搜索时，如果选择了某数，可以将这个数直接计算到总和sum里。到达边界后，直接判断sum是否是素数就可以了。递归返回上一层时，不加上这个数字即可。

那么sum随着递归而变化，把sum放进dfs函数的参数列表中。这样可以不需要f数组来标记了。以下是第二种写法：

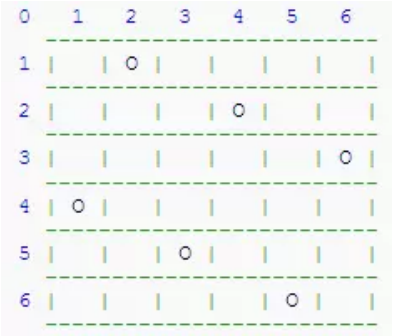
```
#include <bits/stdc++.h>
using namespace std;
int n,k,a[25],ans;
bool isPrime(int x){
```

```
    if(x<2) return false;
    for(int i=2;i<=sqrt(x);i++){
        if(x%i==0) return false;
    }
    return true;
}
void dfs(int cur,int cnt,int sum){
    if(cur==n+1){
        if(cnt==k){
            if(isPrime(sum)) ans++;
        }
        return;
    }
    dfs(cur+1,cnt+1,sum+a[cur]);
    dfs(cur+1,cnt,sum);
}
int main(){
    cin>>n>>k;
    for(int i=1;i<=n;i++) cin>>a[i];
    dfs(1,0,0);
    cout<<ans;
    return 0;
}
```

例4:(2099)n皇后

【题目描述】

一个如下的 6×6 的跳棋棋盘，有六个棋子被放置在棋盘上，使得每行、每列有且只有一个，每条对角线（包括两条主对角线的所有平行线）上至多有一个棋子。



上面的布局可以用序列 2 4 6 1 3 5 来描述，第 i 个数字表示在第 i 行的相应位置有一个棋子，如下：

行号 1 2 3 4 5 6

列号 2 4 6 1 3 5

这只是棋子放置的一个解。请编一个程序找出所有棋子放置的解。

并把它们以上面的序列方法输出，解按字典顺序排列。

请输出前 3 个解。最后一行是解的总个数。

【输入】

一行一个正整数 n(6≤n≤13)，表示棋盘是 n×n 大小的。

【输出】
前三行为前三个解，每个解的两个数字之间用一个空格隔开。
第四行只有一个数字，表示解的总数。

【输入样例】
6

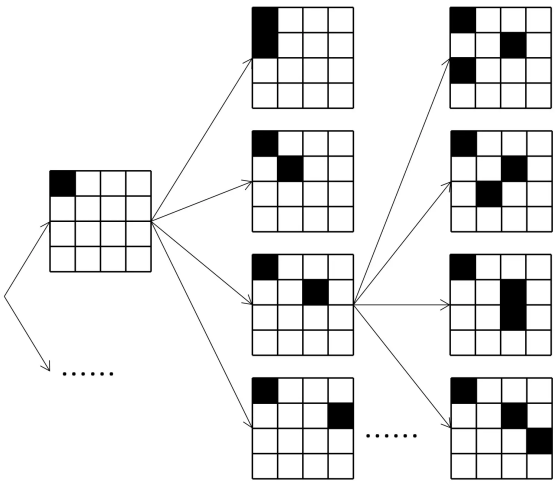
【输出样例】
2 4 6 1 3 5
3 6 2 5 1 4
4 1 5 2 6 3
4

分析

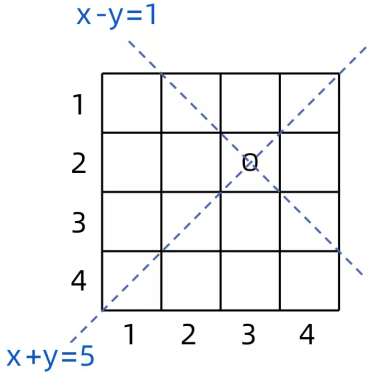
我们发现每行每列都恰好放一个皇后，所以确定搜索的框架为一行一行的去确定每一行的皇后放在哪一列。

这样就是个排列型搜索，在排列型搜索的基础上，我们再去判断新放上去的皇后是否和之前放的皇后在竖线和对角线上有冲突即可。

下图是4皇后的解答树：



(k,a[k])与(j,a[j])，若两者的列、主对角线、副对角线，其中一个有重叠，则不合法



我们可以发现，对于同一个对角线上的所有坐标点(x,y)中，x+y 或者 x−y 是定值。

```
int a[15],n,tot;
void dfs(int k){
    ......
    for(int i=1;i<=n;i++){
        int ok=1;
        a[k]=i; //尝试把第k行的皇后放在第i列
        for(int j=1;j<k;j++){ //判断是否和前面的冲突
            if(a[k]==a[j] || k-a[k]==j-a[j] || k+a[k]==j+a[j]){
                ok=0;
                break;
            }
        }
        if(ok) dfs(k+1); //合法，继续递归
    }
}
```

完整代码如下：

```
#include <bits/stdc++.h>
using namespace std;
int a[15],n,tot;
void dfs(int k){
    if(k==n+1){
        tot++;
        if(tot>3) return;
        for(int i=1;i<=n;i++) cout<<a[i]<<" ";
        cout<<endl;
        return;
    }
    for(int i=1;i<=n;i++){
        int ok=1;
        a[k]=i; //尝试把第k行的皇后放在第i列
        for(int j=1;j<k;j++){ //判断是否和前面的冲突
            if(a[k]==a[j] || k-a[k]==j-a[j] || k+a[k]==j+a[j]){
                ok=0;
                break;
            }
        }
        if(ok) dfs(k+1); //合法，继续递归
    }
}
int main(){
    cin>>n;
    dfs(1);
    cout<<tot<<endl;
    return 0;
}
```

但是这个方法会有数据点超时。因为我们在判断冲突时，用了一个循环。

我们可以用二维数组标记竖线、斜线的占用状态，查询占用的复杂度降到O(1)。

判断(k,i)这个位置的列、主对角线、副对角线是否被占用。注意，k-i 可能为负值，放在数组中会越界，所以存取的时候要加上n。

答案

```
#include <bits/stdc++.h>
using namespace std;
int a[15],n,tot,vis[3][35];
void dfs(int k){
    if(k==n+1){
        tot++;
        if(tot>3) return;
        for(int i=1;i<=n;i++) cout<<a[i]<<" ";
        cout<<endl;
        return;
    }
    for(int i=1;i<=n;i++){
        if(!vis[0][i] && !vis[1][k+i] && !vis[2][k-i+n]){
            a[k]=i;
            vis[0][i]=vis[1][k+i]=vis[2][k-i+n]=1;//标记相关的列、主对角线、副对角线已经被占用
            dfs(k+1);
            vis[0][i]=vis[1][k+i]=vis[2][k-i+n]=0;//恢复现场
        }
    }
}
int main(){
    cin>>n;
    dfs(1);
    cout<<tot<<endl;
    return 0;
}
```

总结

1. 严格来说，搜索算法也算是一种暴力枚举策略。DFS会尝试所有可能的解，沿着一条路径一直搜索下去，直到不能再搜索就退回到上一步。
2. 在算法竞赛中，如果无法找到一种高效求解的方法，使用搜索也可以解决一些规模较小的情况，而有的任务就必须使用搜索来完成，因此这是相当重要的策略。
3. 目前遇到的题目数据范围较小，不容易超时，但是如果枚举量过大，就需要优化掉过多的无效状态，降低问题规模。下节课将会介绍深搜的剪枝技巧。

作业

(1693)校园舞会
(1694)肥成选零食
(1695)燃烧我的卡路里