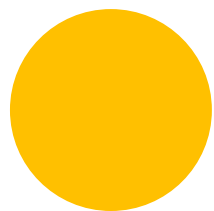
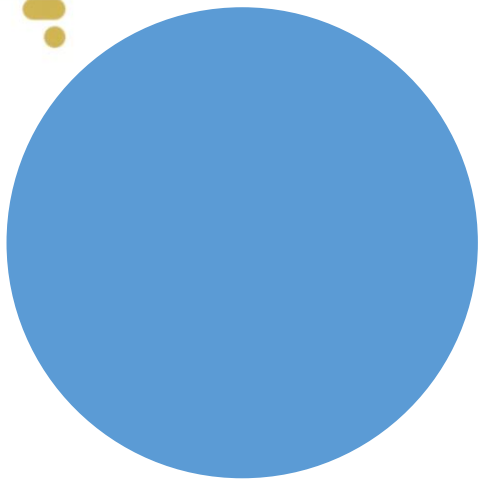


马上开始

35tang-C++竞赛系列三阶课程





《 35tang-C++竞赛系列三阶课程 》




本节目标

搜索与回溯

A small green seedling with several leaves is growing out of a crack in a dark, textured rock surface. The background is a soft, out-of-focus light color.

全局变量

- 位于所有函数外面
- 无论是main函数还是自定义函数都可以访问修改



DFS：一步选一个，一条路走到底就选好了（一个路径，一个排列...）

DFS都是类似模板程序，不难。

搜索就是枚举全部可能的一种手段。相比多重循环，他并没有降低时间复杂度。

但是请大家在后面的学习中多多注意参数是如何设计的，如何传递的，如何变化的？包括全局变量，这些代表了DFS的状态，代表了DFS如何枚举下一步。这些是DFS的精华部分。

DFS算法核心

当前一步怎么走
(当前怎么选)

下一步往哪走
(下一个怎么选)

课堂练习回 忆迷宫问题 DFS

- 比如我们定义一个二维数组（矩阵结构）来表示迷宫。

```
int maze[5][5] = {  
0, 0, 0, 0, 0,  
0, 0, 0, 1, 0,  
0, 0, 0, 0, 0,  
0, 1, 1, 1, 1,  
0, 0, 0, 1, 0};
```

其中的1表示墙壁，0表示可以走的路，只能横着走或竖着走，不能斜着走，请判断是否存在路线可以从左上角走到右下角走出迷宫。

DFS如何枚举路线的

grid[][]

		row				
		0	1	2	3	4
col	0	0	0	0	0	0
	1	0	1	0	1	1
	2	0	1	0	1	1
	3	0	1	1	1	1
	4	0	0	0	0	0

关键：

注意左面的层次关系。

DFS枚举是一条路走到黑，比如(0,0)格子出发，有两个相邻节点，但是先走(0,1)，接着走(0,1)的相邻节点，一直走到出口或者没有格子可以走了，再回来继续访问(0,0)格子的下一个相邻节点，比如(1,0)。

访问过的不能再访问。

BFS利用队列。DFS利用递归。都是状态的保存。也就是说访问下一次之前要记住刚才到哪了。

DFS其实就是利用了函数的递归栈来做回溯。当我们访问(0,0)格子的时候，我们发现了两个相邻节点，但是这个时候要先去处理(0,1)格子一直到底，(0,1)格子这条路走到底回来之后，怎么才能知道应该搜索(1,0)格子呢？DFS就是利用函数栈把搜索(0,1)格子之前的状态保存起来，搜索完(0,1)这条线路后取出状态，访问(1,0)。这就是回溯。

DFS模板函数

{

处理当前节点



迷宫DFS函数

{

当前行，列位置是否终点

进入下一个合法节点：
递归调用相邻合法节点



选择下一个位置：递归调用（当前
位置的四个没有走过的相邻位置）

}

}

```
bool dfs(int row,int col)
{
    visited[row][col]=true;
    //visited数组进入本次函数调用后修改为true，那么当前的这个row，col所对应的visited数组里面的元素永远是true，即使本次函数调用结束返回
    //所以无论那一条路线，在哪个递归函数里面，只要是在本次调用之后，都会发现该位置是true，从而避免再次访问
    if (row==4 && col==4) return true;
    //下面尝试4个方向取找，哪条找到了就返回true，都找不到就返回false
    for (int i=0;i<4;i++)
    {
        //函数栈的回溯功能，每次循环dfs(newrow,newcol)函数返回后，row，col并不会被修改，所以我们计算newrow，newcol不会出错。
        int newrow=row+rowdir[i];
        int newcol=col+coldir[i];
        if (newrow<0 || newrow>4 || newcol<0 || newcol>4 || visited[newrow][newcol] || maze[newrow][newcol]==1)
            continue;
        if (dfs(newrow,newcol)==true) return true;
    }
    return false;
}
```

注意，每次递归调用下一个dfs函数，也就是说尝试走到下一步的时候，下一步需要知道前面哪些位置是走过的：这个通过全局变量visited数组（因为全局变量全局可以访问，即使是下一个函数内部也可以访问），还需要知道当前走到哪个格子了，也就是row和col，这个是通过参数传递的。

为什么访问过了就不需要再访问了？因为如果一个位置往下走过一次，无论从这个位置是否可以最终到达出口，从其他位置到这个位置再走一次，结果是一样的。

思考

- 如果我们修改为寻找一共多少条不同的路径可以走出去（同一个位置在同一条路线只能访问一次）？
 - 刚才是找到一条就退出。现在需要找到所有。所以肯定需要一个计数器，当找到一条路的时候计数器+1，可以作为全局变量。
 - 更加特殊的，现在每一个位置可能会被访问多次，他可能位于不同路径上，所以这次走完一条路就需要把visited数组的状态back回去。看蓝色线走到红色0，黄色线再过来原来是不用再走了，因为这个位置visited设置为true，已经知道这个点走过了。但是现在这算两条路，要都能走。
- 一起来修改一下刚才的程序

```
int maze[5][5] = {
0, 0, 0, 0, 0,
0, 1, 0, 1, 0,
0, 0, 0, 0, 0,
0, 1, 1, 1, 0,
0, 0, 0, 1, 0};
```

手工回溯比普通DFS多了什么？

分支路线都尝试完毕后，也就是当前节点访问后，或者说函数返回时。当前状态需要“回溯”（还原），这样退到上一层走其他路线的时候这个位置就可以再次访问了。

```
void dfs(int row,int col)
```

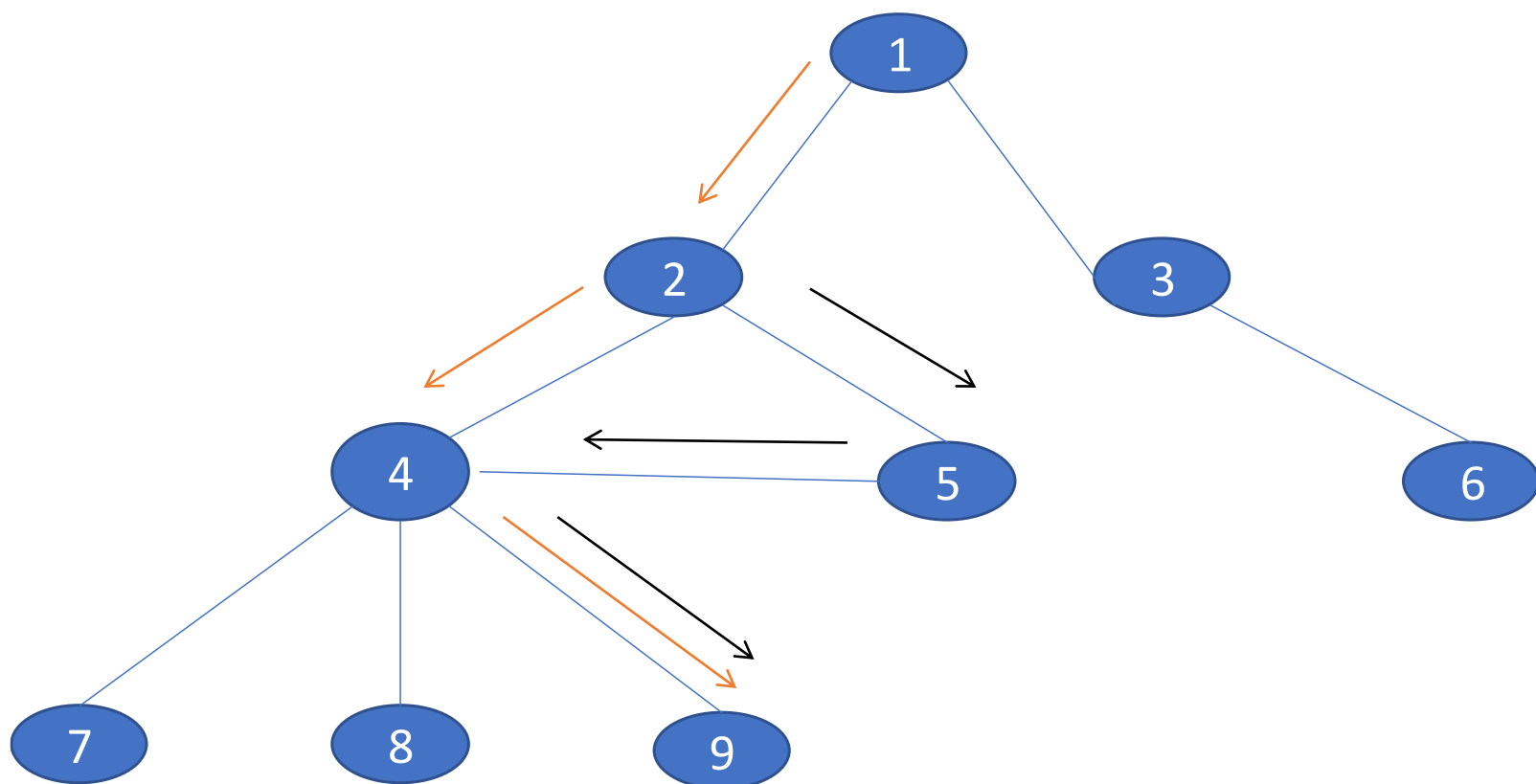
```
{  
    //visited[row][col]=true;  
    //放在这里会出错，因为如果有一条路到了终点就return了，来不及手工回溯了  
    if (row==4 && col==4)  
    {  
        ans++;  
        return;  
    }  
    visited[row][col]=true;  
    //尝试4个方向  
    for (int i=0;i<4;i++)  
    {  
        int newrow=row+        rowdir[i];  
        int newcol=col+        coldir[i];  
        if (newrow<0 || newrow>4 || newcol<0 || newcol>4  
            || visited[newrow][newcol] || maze[newrow][newcol]==1)  
            continue;  
        dfs(newrow,newcol); //每条路都要走到底，这些相邻点走下去的路线上的每一个为hi的函数中，本节点的visited是true。不会被再次访问的。因为这些路线不返回不会执行后面的visited[row][col]=false;语句  
    }  
    visited[row][col]=false;  
    //手工回溯，当前的row col这个位置可能要在其他路径使用，抹去row col走过的“痕迹”，或者说还原到原来的状态  
    return;  
}
```

回溯到底还原的是哪一层的状态其实取决于你什么时候需要这个状态是干净的。比如右面，如果我们在访问每一次相邻节点的时候设置相邻节点的visited标记为true，那么回溯就应该是每一个相邻节点访问过后，而不是函数最后。或者说你修改了什么，就要还原什么。注意这种方法，在主函数调用dfs(0,0)之前需要设置visited[0][0]为true，因为起点只能访问一次。

```
void dfs(int row,int col) //回溯法实现
```

```
{
void dfs(int row,int col)
{
    if (row==4 && col==4)
    {
        ans++;
        return ;
    }
    //尝试4个方向
    for (int i=0;i<4;i++)
    {
        int newrow=row+    rowdir[i];
        int newcol=col+    coldir[i];
        if (newrow<0 || newrow>4 || newcol<0 || newcol>4
            || visited[newrow][newcol] || maze[newrow][newcol]==1)
            continue;
        visited[newrow][newcol]=true;
        dfs(newrow,newcol); //每条路都要走
        visited[newrow][newcol]=false;
        //手工回溯，当前的newrow newcol这个位置可能要在其他路径
        //使用，甚至被其他的当前row col的相邻节点访问，抹去newrow
        //newcol走过的“痕迹”，或者说还原到原来的状态
    }
    return ;
}
```

```
}
```



左面状态树，比如每一个节点可能对应迷宫里面的一个位置。为什么需要手工回溯访问状态：如果我们是要搜索节点1到节点9有几个不同的路径，那么这个visited状态设置回false的操作就是必须的，比如我们第一次通过节点2访问了节点4，一直下去到节点9，节点4访问返回后，需要把节点4设置visited回false，这样我们在访问节点5的时候，就还是可以访问到节点4，但是这个和刚才才是不同的路径了。

关键：visited是全局变量。

回溯和手工回溯

是一种选优搜索法，又称为试探法，按选优条件向前搜索，以达到目标。但当探索到某一步时，发现原先选择并不优或达不到目标，就退回一步重新选择，这种走不通就退回再走的技术为回溯法。

DFS搜索利用递归其实就是回溯，记不记得函数栈，每一次递归下一层之前上一层的状态其实是保存下来的，等访问完下一层（相邻节点状态）系统会从栈取出刚才保存的状态，这就是回溯。但是，函数栈只会保存参数和局部变量，不会保存全局变量。这个时候如果有全局变量在函数调用下一层之前被修改了，就可能要手工回溯全局变量的状态，所谓手工回溯，就是自己把变量的状态还原到刚才调用之前。如果我们把状态全部作为参数传递，可能就不需要单独手工回溯了。

手工回溯 不是必须的

有时候手工回溯的不一定是visited是否访问过的状态。可以回溯其他任何的状态信息。

回溯的是一般是全局变量。因为系统栈回溯的是下一层函数调用前的状态，而调用下一层函数之前，全局变量有可能为了下次调用修改了（这个全局变量不在系统栈，所以系统无法自动回溯）。手工回溯就是要把这个修改在调用之后给还原了，彻底还原到调用之前的状态，从而为其他调用做好准备。

课堂讲解： 数字排列(选数)-有序枚举

- 给出 n 个数字 ($1 \leq n \leq 100$)，从里面选 k 个数字 ($1 \leq k \leq 80$) 请给出从小到大的所有排列(输出不重复，比如输出了4 5不会输出5 4)。格式如下，第一行 n 和 k 空格隔开，第二行 n 个数字空格隔开：

5 2

1 2 3 4 5

- 输出

1 2

1 3

1 4

1 5

2 3

2 4

2 5

3 4

3 5

4 5

第一步:有序枚举，先把输入数据排序，保证输入数据是类似 1 2 3 4 5 的顺序，这样，数组下标小的在前面就是数字小的在前面。

所谓的一个排列其实就是如何从输入数组中选择若干个数字

- 如果给定了k很小，比如是2或者3，我们显然可以循环枚举出所有可能取判断。但是现在k很大，没有办法枚举，就只有搜索了。
`inputnum[]={1,2,3,4,5};`
- 搜索就是另外一种枚举方法：先枚举输出的第一个，可以是输入的1到5任意一个，接下来递归枚举第二个，有序枚举，这一次枚举刚才选择的之后的任意一个，比如第一次如果选择输入的第1个元素，找第二个就从输入的第2个开始枚举。
- 搜索的核心：每一次选择一个输入的数到输出数组。选够了路径就到头了，也就是选择好了一个排列。
- 搜索的路径就是每一次从输入数组中选择一个当前输出的数，然后走到下一步（相邻节点）选择下一个输出的数。
- 和迷宫对比，迷宫中每一次处理的当前节点就是当前的行和列，这里每一次处理的当前节点就是当前要选择输出的第几个和上次选择了输入的第几个。迷宫的相邻节点是4个方向的4个位置，这里的可能的相邻节点就是依次选择上次输入的数之后的每一个数。

DFS模板函数

{

处理当前节点



当前行，列位置是否终点

进入下一个合法节点：
递归调用相邻合法节点



选择下一个位置：递归调用（当前
位置的四个没有走过的相邻位置）

}

}

DFS模板函数

{

处理当前节点



当前输出数组是否选够了

进入下一个合法节点：
递归调用相邻合法节点 分



选择下一个数：递归调用（下一个数
别选择当前已经选的数的后面的任
意一个可选的数）

}

}

迷宫DFS函数

{

选数DFS函数

{

抽象的DFS不是搜索图的路径，搜索的是状态，和图搜索思路类似，图搜索需要把当前走到的位置状态传递下取，而抽象的DFS一样要把当前选择的状态传递下取，比如当前选择了几个数，比如当前选到第几个数了，这些都是状态信息。DFS其实就是特殊的枚举，就是把每一种可能都列出来，或者说把每一种组合都列出来，当DFS函数返回的时候，就表示一条路到头了，也就是一种可能的组合完成了。

- 算法的核心是DFS搜索，每次选择当前输入数组元素（排过序）的后面的每一个元素进行尝试，选择某一个元素，就类似图搜索中找到了一个相邻节点，进入下一步。每条路上选择的数保存在输出数组outputnum。
- 这里的状态（类似迷宫的当前位置信息）：数组outputnum和长度j保存当前形成的输出数列，参数i是在inputnum序列尝试到了哪一个位置。在每一个分支路线dfs(i,j+1,outputnum);之后，i和j都不会修改。所以不需要手工回溯。

```
void dfs(int i,int j,int outputnum[101])
//i表示处理到了输入数组哪个元素，j表示当前选第几个
{
    if(j==k+1) { //找到k个了,输出
        for (int l=1;l<=k;l++) cout<<outputnum[l];
        cout<<endl;
        return;
    }
    for (int l=i+1;l<=n;l++) //当前j位置分别尝试i后面的每一个元素
    {
        outputnum[j]=inputnum[l];
        dfs(l,j+1,outputnum); //没有修改i和j
    }
    return;
}

int main()
{
    cin>>n>>k;
    for (int i=1;i<=n;i++) fin>>inputnum[i];
    sort(inputnum+1,inputnum+n+1); //排序
    int outputnum[101]={0};
    dfs(0,1,outputnum);
    return 0;
}
```

为什么不
需要visited
数组？

有序枚举

```
for (int l=i+1;l<=n;l++)  
//当前j位置分别尝试i后面的每一个元素  
{  
    outputnum[j]=inputnum[l];  
    dfs(l,j+1,outputnum); //没有修改i和j  
}
```

每一次选择的是上一次选择的数字后面的一个，不会往回选，所以不会重复。

```

int n, k;
int inputnum[101];
int outputnum[101];
int outlen=0;
void dfs(int i,int j)
{
    if(j==k+1) {
        for (int l=1;l<=k;l++)    cout<<outputnum[l];
        cout<<endl;
        return;
    }
    for (int l=i+1;l<=n;l++)    {

```

//需要保证这个循环内每一次执行下面这句话的时候
的outlen都是本次函数进来的时候的outlen

```

        outputnum[++outlen]=inputnum[l];
        dfs(l,j+1);
        outlen--;
    }
    return;
}

```

```

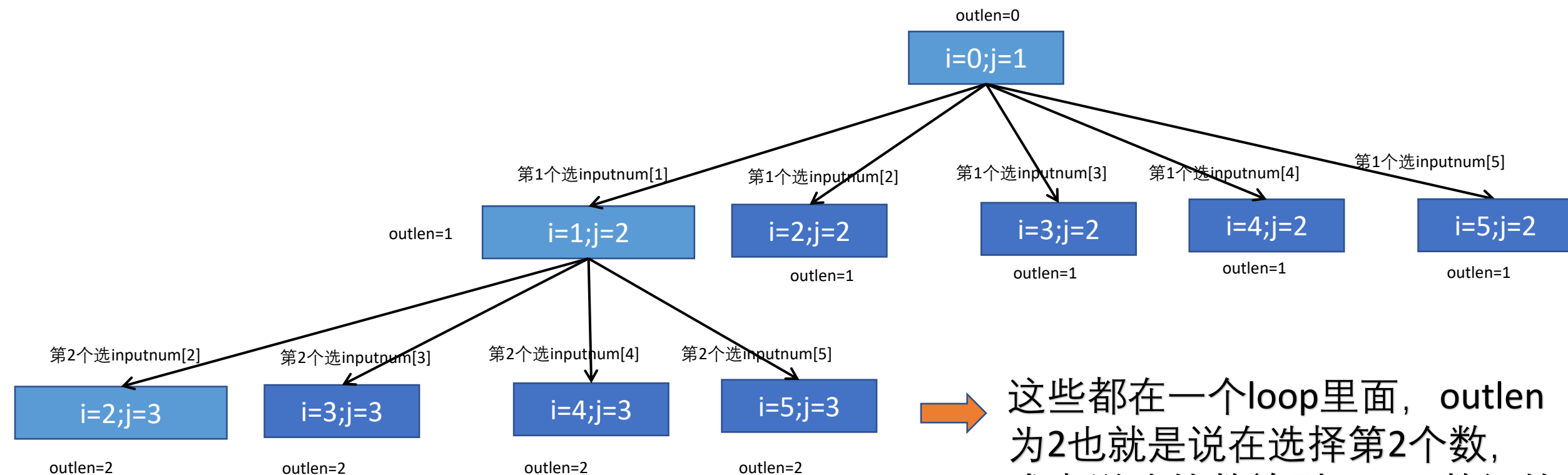
int main()
{
    cin>>n>>k;
    for (int i=1;i<=n;i++) cin>>inputnum[i];
    sort(inputnum+1,inputnum+n+1);
    dfs(0,1);

    return 0;
}

```

//把outputnum作为全局变量，用outlen表示当前位置，手工回溯outlen

为什么要手工回溯？看outlen的值的变化，在一条路走下去的时候outlen是不断增加的，比如在第一个位置，选择原数组第一个元素给outputnum，这个时候outlen是1，然后递归取选择outputnum的第2个元素。当这一次递归全部结束以后，需要尝试选择原数组第二个元素给outputnum，这个时候的outlen需要还原回到1。



➡ 这些都在一个loop里面，outlen为2也就是说在选择第2个数，或者说选的数放到output数组的第2个位置，需要保证outlen都是2

其实这个输出数组用
vector更方便

- 这里的输出数组一直在传递他的数组下标，或者说当前插入到输出数组的位置。
- 其实用vector的push_back()和pop_back()方法就不用自己控制了。
- 修改一下看看

```
vector<int> outputnum;
void dfs(int i)
//i表示从input数组已经选择了第几个
{
    if(outputnum.size()==k) {
        for (int l=0;l<outputnum.size();l++)    cout<<outputnum[l]<<" ";
        cout<<endl;
        return;
    }
    for (int l=i+1;l<=n;l++) {
        //outputnum[++outlen]=inputnum[l];
        outputnum.push_back(inputnum[l]);
        dfs(l);
        //outlen--;
        outputnum.pop_back();
    }
    return;
}
```

学到了什么？

- DFS函数一般算法类似模板程序
 - 判断是否退出
 - 尝试后面的每一个可能节点（或者说每一个选择）
 - 尝试新节点之前可能需要手工回溯状态，这些状态一定是在调用下一层函数之前被修改过了
- 抽象的DFS不是搜索图的路径，搜索的是状态，和图搜索思路类似，图搜索需要把当前走到的位置状态传递下取，而抽象的DFS一样要把当前选择的状态传递下取，比如当前选择了几个数，比如当前选到第几个数了，这些都是状态信息，就是你当前的一个选择。迷宫的当前状态是选择了哪行哪列，往下走，所以参数需要传递这个行和列，而选数的例题，当前状态是选择了第几个输入数字，所以参数传递是这个序号，或者说第几个。为什么传递这些状态？因为下一步必须知道当前在哪条路上或者说上一步是什么选择，才能选择下一步走下去。
- DFS其实就是特殊的枚举，就是把每一种可能都列出来，或者说把每一种组合都列出来，当DFS函数返回的时候，就表示一条路到头了，也就是一种可能的组合完成了。或者说DFS就是做一个选择，然后走下去。第一步就要确定怎么走下去或者说怎么枚举，比如迷宫，就是要从一个位置走4个相邻位置，比如选数，就是要选择一个数，后面的数从他后面的位置选，比如8皇后，就是选了一行，后面选择他后面的一行。复杂的就是要去设计走下去需要传递什么参数？修改什么状态，是否需要手工回溯等等。
- 一般对于访问状态visited数组，手工回溯的机会最大，因为同一个状态下一步的几个分支状态，一个个访问，访问的时候如果每一个分支状态希望都是“干净”的，就需要手工回溯。
- 简单一点，不用太刻意，无论局部变量还是全局变量，是否需要手工回溯，关键是看这个变量是否在下次调用之前是“干净”的，是否被修改过。如果修改过，下次需要时“干净”的，就需要手工回溯。主要还是一个逻辑判断，自己去判断下一次调用如果不手工回溯，那么这个变量的值对不对。

下节课例题预习

- 尝试理解题目，思考可能的算法，不需要做
- 2017 NOIP棋盘



作业1selectm.cpp

给定 n （不超过100）个不同数字如何选出若干个（1到 n ）数字，使得他们的和是 m 。列出所有的可能。每种可能一行输出，数字之间用空格分开。输出数字按照从小到大顺序排列。题目保证所有输入数据的和不超过10的8次方。

示例输入

5 7

5 2 3 4 1

示例输出：

1 2 4

2 5

3 4

示例说明：这个例子中要求选择的和是7，显然有3种可能，都列出来。



作业1提示

- 也是有序去枚举，这样可以避免重复选择数字
- 这道题也是用DFS搜索，和课堂例题不同的在于退出条件不同：搜到了一条路径的退出条件是数字和为 m ，这个时候输出并且退出。
- 需要一个output数组保存当前选择的数。搜索的主要思路和课堂的例子“数字排列”基本一致。dfs函数中的相邻节点(下一步)就是选择上次选择数字后面的任意一个数。但是不是选出 k 个输出，而是选择的排列和为 m 的输出。
- 当然，也可以用二阶介绍的方法，dfs函数每一步是当前元素选和不选两个分支（相邻节点）走下去。

挑战作业2

自然数的拆分问题

- <https://www.luogu.com.cn/problem/P2404>
 - 提示：非常类似上一个题目。 n 拆分，dfs每一次可能选择 $1, 2, 3 \dots$ 作为当前拆分的数字，然后下一次应该再拆分出来的数字只能大于等于上一次拆分的数字并且小于 n -上次拆分的数字。比如上一次拆分出来的是 i ，那么下一次拆分就应该从 $i, i+1 \dots n-i$ 去选择。dfs函数传递下去的参数应该是当前选择了哪个数以及拆分剩下的和，并且把选择的数保存在数组。

超级挑战作业3

- USACO training Section 2.1 [PROB Healthy Holsteins](https://www.luogu.com.cn/problem/P1460) <https://www.luogu.com.cn/problem/P1460>

给出牛所需的最低的维他命量和每种饲料包含的各种维他命的量，输出喂给牛需要几种饲料，以及他们的编号，要求饲料种类最少。如果出现重复，输出编号最小的一组。维他命量以整数表示，每种饲料最多只能对牛使用一次，数据保证存在解。比如给出：

4

100 200 300 400


3

50 50 50 50

200 300 200 300

900 150 389 399

上述输入表示奶牛需要4种维他命，第二行的四个整数表示这4种维他命的量。后面给出3组饲料，每种饲料给出的4个整数分别表示该饲料包含的维他命的量。显然，可以用后面给出3组饲料里面的第1组和第3组凑够100 200 300 400。所以，输出 2 1 3



作业3提示

- 思路不难，但是很繁琐。
- 与作业1非常像，不同的现在不是选择若干个数字求和判断，而是选择多组数字分别求和。DFS就是要枚举出所有的可能选择的组合。
- 从前往后选择，对于每一组数据，都有选择不选择两种可能。需要保存当前选择了哪几种饲料，当发现某一次搜索结束（也就是选择的饲料的几个维他命的量都大于等于要求的总量的时候），需要打擂台法判断该次搜索包含的饲料数目是否是最小的，如果是，就需要保存最新的选择。

由易到难，思维体系训练
实战结合，创新协作培养
兴趣导向，未来职业引领

<https://www.35tang.com>



扫码关注公众号

<https://www.三五堂.com>



添加辅导老师