

DFS（搜索）



引入

DFS 为图论中的概念，详见 [DFS（图论）](#) 页面。在 **搜索算法** 中，该词常常指利用递归函数方便地实现暴力枚举的算法，与图论中的 DFS 算法有一定相似之处，但并不完全相同。

解释

考虑这个例子：

例题



把正整数 n 分解为 3 个不同的正整数，如 $6 = 1 + 2 + 3$ ，排在后面的数必须大于等于前面的数，输出所有方案。

对于这个问题，如果不知道搜索，应该怎么办呢？

当然是三重循环，参考代码如下：

实现

C++

```
1 for (int i = 1; i <= n; ++i)
2     for (int j = i; j <= n; ++j)
3         for (int k = j; k <= n; ++k)
4             if (i + j + k == n) printf("%d = %d + %d + %d\n", n, i, j, k);
```

Python

```
1 for i in range(1, n + 1):
2     for j in range(i, n + 1):
3         for k in range(j, n + 1):
4             if i + j + k == n:
5                 print("%d = %d + %d + %d" % (n, i, j, k))
```

Java

```
1 for (int i = 1; i < n + 1; i++) {
2     for (int j = i; j < n + 1; j++) {
3         for (int k = j; k < n + 1; k++) {
4             if (i + j + k == n) System.out.printf("%d = %d + %d + %d\n", n, i, j, k);
5         }
6     }
7 }
```

那如果是分解成四个整数呢？再加一重循环？

那分解成小于等于 m 个整数呢？

这时候就需要用到递归搜索了。

该类搜索算法的特点在于，将要搜索的目标分成若干「层」，每层基于前几层的状态进行决策，直到达到目标状态。

考虑上述问题，即将正整数 n 分解成小于等于 m 个正整数之和，且排在后面的数必须大于等于前面的数，并输出所有方案。

设一组方案将正整数 n 分解成 k 个正整数 a_1, a_2, \dots, a_k 的和。

我们将问题分层，第 i 层决定 a_i 。则为了进行第 i 层决策，我们需要记录三个状态变量：
 $n - \sum_{j=1}^i a_j$ ，表示后面所有正整数的和；以及 a_{i-1} ，表示前一层的正整数，以确保正整数递增；以及 i ，确保我们最多输出 m 个正整数。

为了记录方案，我们用 `arr` 数组，第 i 项表示 a_i 。注意到 `arr` 实际上是一个长度为 i 的栈。

代码如下：



实现

C++

```
1  int m, arr[103]; // arr 用于记录方案
2
3  void dfs(int n, int i, int a) {
4      if (n == 0) {
5          for (int j = 1; j <= i - 1; ++j) printf("%d ", arr[j]);
6          printf("\n");
7      }
8      if (i <= m) {
9          for (int j = a; j <= n; ++j) {
10             arr[i] = j;
11             dfs(n - j, i + 1, j); // 请仔细思考该行含义。
12         }
13     }
14 }
15
16 // 主函数
17 scanf("%d%d", &n, &m);
18 dfs(n, 1, 1);
```

Python

```
1  arr = [0] * 103 # arr 用于记录方案
2
3
4  def dfs(n, i, a):
5      if n == 0:
6          print(arr[1:i])
7      if i <= m:
8          for j in range(a, n + 1):
9              arr[i] = j
10             dfs(n - j, i + 1, j) # 请仔细思考该行含义。
11
12
13 # 主函数
14 n, m = map(int, input().split())
15 dfs(n, 1, 1)
```

Java

```
1  static int m;
2
3  // arr 用于记录方案
4  static int[] arr = new int[103];
5
6  public static void dfs(int n, int i, int a) {
7      if (n == 0) {
```

```
8         for (int j = 1; j <= i - 1; j++) System.out.printf("%d ",
9 arr[j]);
10        System.out.println();
11    }
12    if (i <= m) {
13        for (int j = a; j <= n; ++j) {
14            arr[i] = j;
15            dfs(n - j, i + 1, j); // 请仔细思考该行含义。
16        }
17    }
18 }
19
20 // 主函数
21 final int N = new Scanner(System.in).nextInt();
22 m = new Scanner(System.in).nextInt();
23 dfs(N, 1, 1);
```

例题



```
1  #include <iomanip>
2  #include <iostream>
3  using namespace std;
4  int n;
5  bool vis[50]; // 访问标记数组
6  int a[50];    // 排列数组，按顺序储存当前搜索结果
7
8  void dfs(int step) {
9      if (step == n + 1) { // 边界
10         for (int i = 1; i <= n; i++) {
11             cout << setw(5) << a[i]; // 保留5个场宽
12         }
13         cout << endl;
14         return;
15     }
16     for (int i = 1; i <= n; i++) {
17         if (!vis[i]) { // 判断数字i是否在正在进行的全排列中
18             vis[i] = true;
19             a[step] = i;
20             dfs(step + 1);
21             vis[i] = false; // 这一步不使用该数 置0后允许下一步使用
22         }
23     }
24     return;
25 }
26
27 int main() {
28     cin >> n;
29     dfs(1);
30     return 0;
31 }
```



本页面最近更新：2024/5/8 20:33:33, [更新历史](#)



发现错误？想一起完善？ [在 GitHub 上编辑此页！](#)



本页面贡献者：Ir1d, H-J-Granger, partychicken, StudyingFather, countercurrent-time, Enter-tainer, ksyx, NachtgeistW, iamtwz, AngelKitty, Anyxyz, CCXXI, ChungZH, cjsoft, diauweb, Early0v0, ezoixx130, GekkaSaori, greyqz, Henry-ZHR, Konano, loader3229, LovelyBuggies, Makkiy, mgt, minghu6, ouuan, P-Y-Y, PotassiumWings, SamZhangQingChuan, sshwy, Suyun514, weiyong1024, Acfboy, GavinZhengOI, Gesrua, kenlig, kxccc, lychees, Menci, ouuan, Peanut-Tang, shawllew, SukkaW, Tiphereth-A, TrisolarisHD, vincent-163, wysunrise2, Xeonacid, Yue-plus, zyouxam



本页面的全部内容都在 [CC BY-SA 4.0](#) 和 [SATA](#) 协议之条款下提供，附加条款亦可能应用