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++

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

Python

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

Java

```java
for (int i = 1; i < n + 1; i++) {
    for (int j = i; j < n + 1; j++) {
        for (int k = j; k < n + 1; k++) {
            if (i + j + k == n) System.out.printf("%d = %d + %d + %d%n", n, i, j, k);
        }
    }
}
```

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 $a_1, a_2, \ldots, a_k$ 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++

```cpp
int m, arr[103];  // arr 用于记录方案

void dfs(int n, int i, int a) {
  if (n == 0) {
    for (int j = 1; j <= i - 1; ++j) printf("%d ", arr[j]);
    printf("\n");
  }
  if (i <= m) {
    for (int j = a; j <= n; ++j) {
      arr[i] = j;
      dfs(n - j, i + 1, j);  // 请仔细思考该行含义。
    }
  }
}

// 主函数
scanf("%d%d", &n, &m);
dfs(n, 1, 1);
```

Python

```python
arr = [0] * 103  # arr 用于记录方案


def dfs(n, i, a):
    if n == 0:
        print(arr[1:i])
    if i <= m:
        for j in range(a, n + 1):
            arr[i] = j
            dfs(n - j, i + 1, j)  # 请仔细思考该行含义。


# 主函数
n, m = map(int, input().split())
dfs(n, 1, 1)
```

Java

```java
static int m;

// arr 用于记录方案
static int[] arr = new int[103];

public static void dfs(int n, int i, int a) {
    if (n == 0) {
```

```
        for (int j = 1; j <= i - 1; j++) System.out.printf("%d ",
arr[j]);
        System.out.println();
    }
    if (i <= m) {
        for (int j = a; j <= n; ++j) {
            arr[i] = j;
            dfs(n - j, i + 1, j); // 请仔细思考该行含义。
        }
    }
}

// 主函数
final int N = new Scanner(System.in).nextInt();
m = new Scanner(System.in).nextInt();
dfs(N, 1, 1);
```

例题

Luogu P1706 全排列问题

```cpp
#include <iomanip>
#include <iostream>
using namespace std;
int n;
bool vis[50];  // 访问标记数组
int a[50];     // 排列数组，按顺序储存当前搜索结果

void dfs(int step) {
  if (step == n + 1) {  // 边界
    for (int i = 1; i <= n; i++) {
      cout << setw(5) << a[i];  // 保留5个场宽
    }
    cout << endl;
    return;
  }
  for (int i = 1; i <= n; i++) {
    if (!vis[i]) {  // 判断数字i是否在正在进行的全排列中
      vis[i] = true;
      a[step] = i;
      dfs(step + 1);
      vis[i] = false;  // 这一步不使用该数 置0后允许下一步使用
    }
  }
  return;
}

int main() {
  cin >> n;
  dfs(1);
  return 0;
}
```