信息学奥赛笔记01

回溯算法

递归复习——汉诺塔问题https://www.luogu.com.cn/problem/U410585

汉诺塔

题目背景

汉诺塔问题源自印度一个古老的传说，印度教的“创造之神”梵天创造世界时做了 3 根金刚石柱，其中的一根柱子上按照从小到大的顺序摞着 64 个黄金圆盘。梵天命令一个叫婆罗门的门徒将所有的圆盘移动到另一个柱子上，移动过程中必须**遵守以下规则**：

- 每次只能移动柱子最顶端的一个圆盘；
- 每个柱子上，小圆盘永远要位于大圆盘之上；

题目描述

请你写一个递归程序，输出挪圆盘的过程。（假设要将圆盘从 A 柱子上挪到 C 柱子上。）

输入格式

输入一个数字 n，表示需要挪动的圆盘数量。

输出格式

第一行输出一个数字，表示需要挪动圆盘的步数。

之后输出若干行，每一行表示一次挪圆盘的步骤。

样例 #1

样例输入 #1

```
2
```

样例输出 #1

```
3
A->B
A->C
B->C
```

提示

【数据范围】

对于100%的数据，有$1 \leq n \leq 10$

思路分析

对于汉诺塔问题，我们可以把一个大的问题，划分成小的问题，核心思路是，我需要将n个盘子从A柱移动到C柱上，我只需要把它划分成子问题。

- 第一步，把$n-1$个盘子从A柱借助C柱移动到B柱。
- 第二步，把第n号盘子从A柱移动到C柱
- 回到步骤一

那么对于第$n-1$个盘子来说只需要重复这一步骤，只要当前剩余的盘子大于`1`，都重复去执行，最终的结果就会变成只剩下`1`个盘子，只需要直接将他从`A`移动到`C`就可以了，这就是我们在把一个大问题化成一个小问题来解决，只需要我们反复去执行小的问题，最终就能实现大问题，这就是递归。

```cpp
#include <bits/stdc++.h>
using namespace std;
void hanoi(int n, char a, char b, char c) {
    if (n == 1) {
        cout << a << "->" << c << endl;
        return;
    }
    hanoi(n - 1, a, c, b);  //步骤1
    cout << a << "->" << c << endl;  //步骤二
    hanoi(n - 1, b, a, c);  //继续下一步递归
}
int main() {
    int n;
    cin >> n;
    cout << pow(2, n) - 1 << endl;
    hanoi(n, 'A', 'B', 'C');
    return 0;
}
```

回溯算法——排列组合类

> 回溯算法是一种暴力搜索的算法。对于我们平时做题的时候采用的for循环的暴力搜索来说，他是一种横向的搜索，什么是横向的搜索呢？比如说从1-10，我们可以把它排在数轴上，依次寻找，依次检验。而回溯是一种纵向的搜索，我先往深走，走到不能走为止的时候，回头，回到上一层，如果上一层还有别的路通往下一层，那就继续走，直到把所有的路径全部遍历完为止。

我们可以写出回溯算法的基本通式

```cpp
void dfs(/*当前这一层的状态*/) {
    if (/*到达终止条件*/) {
        //判断是否需要保存答案
        return;
    }
    for (/*在当前这一层的情况下，横向的去找寻所有可能得路径*/) {
        if (/*当前路径已经被寻找过*/) {
            continue;
        }
        //将当前选择的情况做上标记/保存
        dfs(/*下一层*/);
```

```
            //回溯，将标记取消
        }
    }
```

回溯算法有一个基本的三要素：

- 传入参数——当前这层的状态
- 终止条件——递归结束的边界
- 循环方式——当前这一层到下一层的方案

对于一个dfs问题来说，搞清楚这三个条件后，代码就很容易能写出来了，整个dfs的过程也清晰明了了。

实质上来说，回溯问题其实只需要关注：**上一层对这一层的影响——传参；这一层对下一层的影响——递归调用的实际的值；这一层内该做的事——循环**

全排列问题

[U410629] 排列组合问题——全排列(E)https://www.luogu.com.cn/problem/U410629

题目描述

输入一个正整数n，求`1 - n`的组成的整数的全排列。

`3`的全排列有

`[1, 2, 3]`

`[1, 3, 2]`

`[2, 1, 3]`

`[2, 3, 1]`

`[3, 1, 2]`

`[3, 2, 1]`

输入格式

一行，一个正整数n。

输出格式

若干行，一行输出一个排列，用空格隔开。

请按照字典序输出！！

样例 #1

样例输入 #1

```
3
```

样例输出 #1

```
1 2 3
1 3 2
2 1 3
2 3 1
3 1 2
3 2 1
```

提示

对于100%的数据，有$1 \leq n \leq 10$。

思路分析

对于全排列问题来说，我们要考虑dfs的三要素：

当前这一层的状态——正在填第几个数

终止条件——已经填完了n个数

循环方式——从1-n循环，如果发现这个数已经被使用过，则跳过

```cpp
#include <bits/stdc++.h>
using namespace std;
int n, ans[11], vis[11];
void dfs(int idx) {   //idx表示当前这一层是正在填第idx个数
    if (idx == n) {    //终止条件，发现已经填完了idx个数，也就是当idx==n时，退出递归
            for (int i = 0; i < n; i++) {
                cout << ans[i] << " ";
            }
            cout << endl;
            return;
        }
        for (int i = 1; i <= n; i++) {   //在填第idx个数的时候，我们把1-n所有的数都找一遍
            if (vis[i]) continue;
            vis[i] = 1;
            ans[idx] = i;
            dfs(idx + 1);
            vis[i] = 0;
        }
}
int main() {
    cin >> n;
    dfs(0);
    return 0;
}
```

组合问题

[U410638]排列组合问题——全组合(E)https://www.luogu.com.cn/problem/U410638

题目描述

输入两个正整数n，k，求`1 - n`的数中选出k个数组合的结果。

输入格式

一行，两个正整数n, k。

输出格式

若干行，一行输出一个排列，用空格隔开。

请按照字典序输出！！

样例 #1

样例输入 #1

```
4 2
```

样例输出 #1

```
1 2
1 3
1 4
2 3
2 4
3 4
```

提示

对于100%的数据，有$1 \leq n, k \leq 30$。

思路分析

和排列问题不同的是，组合问题的终止条件需要变化，由于`(1, 2)(2, 1)`需要被认为是两种不同的结果，如果我们按照排列问题，把循环的内容从1-n全循环一遍的话，必然会导致有重复答案出现的情况，为了避免这个情况，我们可以从上次选的数后面开始作为当前这一层的填数起始点，这样的话，我们最终所有的组合结果都是从小到大递增的，不会出现重复的结果。

```
#include<bits/stdc++.h>
using namespace std;
int n, k;
int ans[50];
```

```cpp
void dfs(int x, int y) {    //对于当前这一层x表示正在填第x个数，y表示这一次可以填的数必须从y开始
    if(x == k) {            //如果填了k个数了，退出递归
        for(int i = 0; i < k; i++) {
            cout<< ans[i] <<" ";
        }
        cout << endl;
        return;
    }
    for(int i = y; i <= n; i++) {    //从y-n开始循环
        ans[x] = i;        //保存当前的数
        dfs(x + 1, i + 1);  //下一层的状态x是当前的x + 1，已经使用了数y，所以下一层可以填的数必须是y + 1开始
    }
}
int main() {
    cin >> n >> k;
    dfs(0, 1);  //最初的时候，我们要开始填第0个数，并且我们可以填的数是从1开始的
    return 0;
}
```

求子集问题

[U410624] 排列组合问题——求子集https://www.luogu.com.cn/problem/U410624

题目描述

输入一个正整数n，求`1 - n`的组成的整数集的子集。

一个整数集`[1, 2, 3]`的子集有

`[1]`

`[2]`

`[3]`

`[1, 2]`

`[1, 3]`

`[2, 3]`

`[1, 2, 3]`

输入格式

一行，一个正整数n。

输出格式

若干行，一行输出一个子集，用空格隔开。

请按照字典序输出！！

样例 #1

样例输入 #1

```
3
```

样例输出 #1

```
1
1 2
1 2 3
1 3
2
2 3
3
```

提示

对于100%的数据，有$1 \leq n \leq 16$。

思路分析

和排列组合问题不同的是，求子集问题的当前状态不再是正在填第i个了，因为排列问题和组合问题有一个明确的终止条件，找到k(组合)/n(排列)个数的时候就停止了，子集问题需要我们每次往答案数组里补入一个数的时候立刻将答案输出出来，其他部分都没有变化。

由于求子集问题需要我们对这个数组频繁的尾插和尾删，所以使用动态数组$vector$求解会更好。

```cpp
#include <bits/stdc++.h>
using namespace std;
vector<int> ans;
int n;
void dfs(int now) {
    if (now == n + 1) {       //最多的时候，我们填写了n个数，终止
        return;
    }
    for (int i = now; i <= n; i++) {    //从当前的now开始往后遍历，依次填数
        ans.push_back(i);     //把当前的i补入数组末尾
        for (int i = 0; i < ans.size(); i++) {    //立刻输出答案
            cout << ans[i] << " ";
        }
        cout << endl;
        dfs(i + 1);    //继续填写下一个数
        ans.pop_back();
    }
}
int main() {
    cin >> n;
    dfs(1);
    return 0;
}

```