实验舱
青少年编程
走近科学 走进名校

蛟龙四班

深度优先搜索-回溯

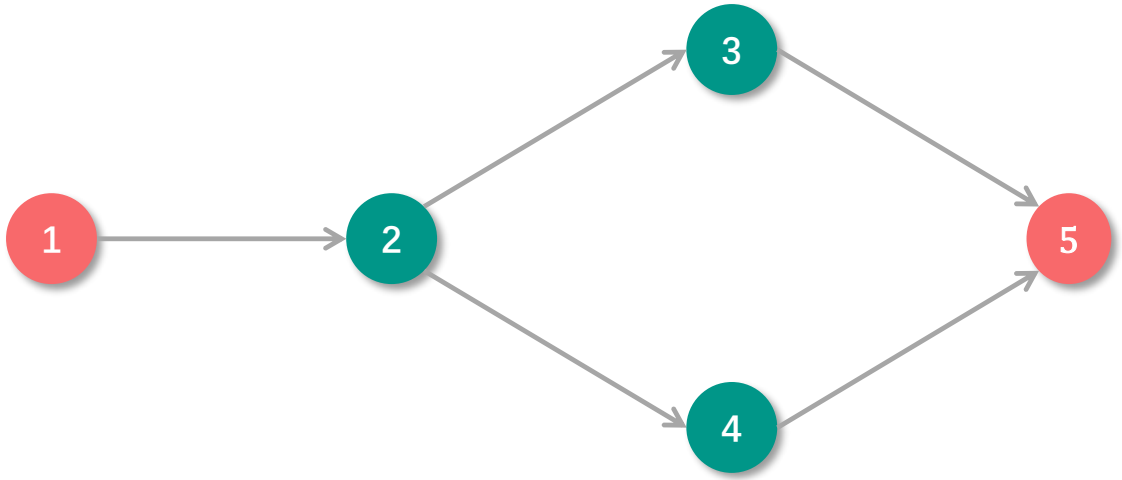
Mas

回溯

搜索与回溯($BackTracking$)是解题的常用方法,一些问题无法根据确定的计算法则来求解,只能利用搜索与回溯的求解

回溯是搜索算法中的一种控制策略它的基本思想:

为了求得问题的解,先选择某一种可能情况向前探索,在探索过程中,一旦发现原来的选择是错误的,就退回一步重新选择,继续向前探索,如此反复进行,直至得到解或证明无解



找出1 ~ 5的所有路径

节点1标记可走,回溯取消标记

节点2标记可走,回溯取消标记

节点3标记可走,回溯取消标记

节点5标记可走,回溯取消标记

#1118、迷宫问题

问题描述

设有一个 $N \times N(2 \leq N < 10)$ 方格的迷宫，入口和出口分别在左上角和右上角。
迷宫格子中分别放 0 和 1，0 表示可通，1 表示不能，入口和出口处肯定是 0。

迷宫走的规则如下所示：

> *从某点开始，有八个方向可走，前进方格中数字为 0 时表示可通过，为 1 时表示不可通过。*
>
> *找出所有从入口（左上角）到出口（右上角）的路径(不能重复)，输出路径总数，如果无法到达，则输出 0*

输入样例

```
3
0 0 0
0 1 1
1 0 0
```

输出样例

```
2
```

当前位置为(x,y)尝试枚举八个方向移动至下一步

为避免重复经过格子,可使用全局数组进行标记

当找到解或无路可走时,回溯取消标记

```
void dfs(int x, int y)
{
  if (x == 0 && y == n - 1)
  {
    ans++;
    return;
  }
  for (int i = 0; i < 8; i++)
  {
    int tx = x + dir[i][0], ty = y + dir[i][1];
    if (tx >= 0 && tx < n && ty >= 0 && ty < n && maze[tx][ty] && !vis[tx][ty])
    {
      vis[tx][ty] = true; //标记为已走过
      dfs(tx, ty);
      vis[tx][ty] = false; //回溯,恢复现场
    }
  }
}
```

#1156、和为T

问题描述

从一个大小为 n 的整数集中选取一些元素，使得它们的和等于给定的值 T 。
每个元素限选一次，不能一个都不选。

输入格式

第一行一个正整数 n ，表示整数集内元素的个数。
第二行 n 个整数，用空格隔开。
第三行一个整数 T ，表示要达到的和。

输出格式

输出有若干行，每行输出一组解，即所选取的数字，按照输入中的顺序排列。

若有多组解，优先输出不包含第 n 个整数的；

若都包含或都不包含，优先输出不包含第 $n-1$ 个整数的，依次类推。

最后一行输出总方案数。

> *观察输出样例,应该优先选择靠后的元素*

样例输入

```
5
-7 -3 -2 5 9
0
```

样例输出

```
-3 -2 5
-7 -2 9
2
```

数据规模和约定

对于全部数据 $1 \leq n \leq 22$， $-2^{63} \leq T < 2^{63}$

#578、n皇后问题

题目描述

要求 n 个国际象棋的皇后，摆在 $n \times n$ 的棋盘上，互相不能攻击，输出全部方案的个数。
当个皇后在同一行，或者同一列，或者同一条主对角线，或者同一条副对角线时，两个皇后会互相攻击。

输入

一个整数，皇后的个数。

输出

一个整数，表示方案数。

输入样例

```
4
```

输出样例

```
2
```

合法方案中一行只能放置一个皇后,逐行考虑放置皇后

对于每一行有n列可以选择放置,尝试放置至第j列

```
int dfs(int r)
{
  if (r == n + 1)
  {
    cnt++;
    return;
  }
  for (int c = 1; c <= n; c++) //枚举当前行放置的列
    if (check(r, c)) //检查是否发生冲突
    {
      ans[r] = c;
      dfs(r + 1);
    }
}
```

#578、n皇后问题

记ans_r为第r行皇后放置的列数

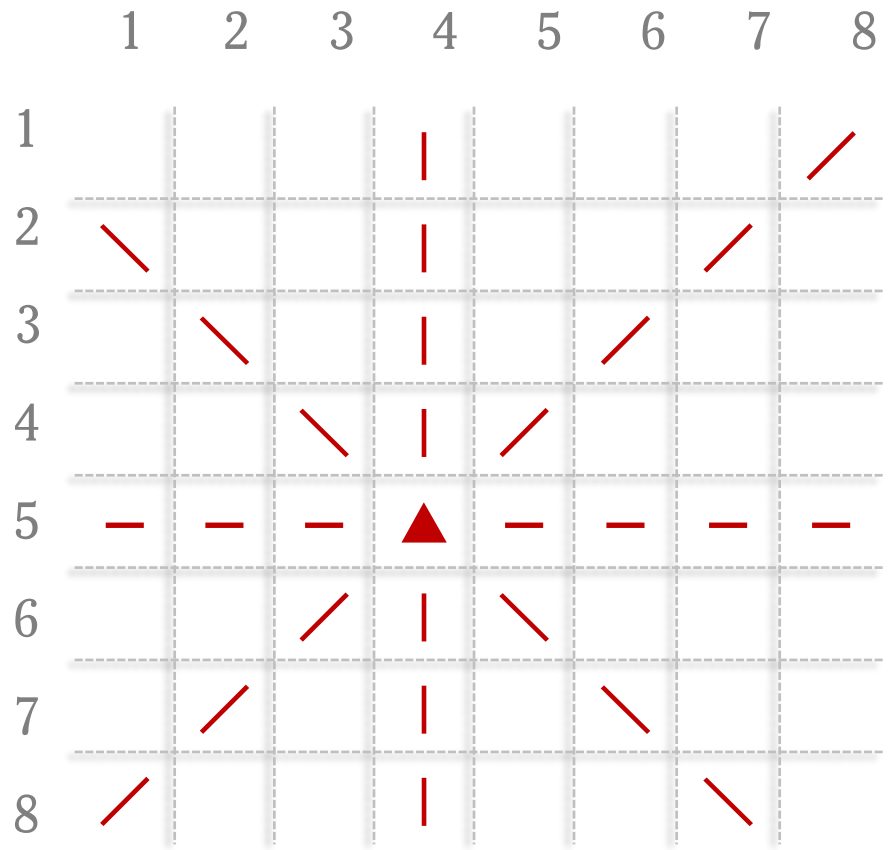
对于第r行放置在c列时

列上冲突仅需要考虑是否存在$t < r, ans_t = ans_r$

左上至右下对角线冲突仅需要考虑是否存在$t < r, ans_t - t = ans_r - r$

右上至左下对角线冲突仅需要考虑是否存在$t < r, ans_t + t = ans_r + r$

```
bool check(int r, int c)
{
  for (int i = 1; i < r; i++)
    if (ans[i] == c || ans[i] + i == r + c || ans[i] - i == c - r)
      return false;
  return true;
}
```



#578、n皇后问题

对于每一次$check$都需要线性的时间复杂度

不妨考虑使用数组进行标记,$ban1[i]$表示第i列已被使用

将每条左上到右下的对角线编号为$x-y+n$

将每条右上到左下的对角线编号为$x+y$

$ban2[x+y]$表示编号为$x+y$的对角线已被占用

$ban3[x-y+n]$表示编号为$x-y+n$的对角线已被占用

仅需要检查这三个数组即可

每次回溯时需要取消这三个数组的标记

```
void dfs(int r)
{
  if (r == n + 1)
  {
    cnt++;
    return;
  }
  for (int c = 1; c <= n; c++)                          //枚举当前行放置的列
    if (!ban1[c] && !ban2[r + c] && !ban3[c - r + n]) //检查是否发生冲突
    {
      ban1[c] = ban2[r + c] = ban3[c - r + n] = true;
      ans[r] = c;
      dfs(r + 1);
      ban1[c] = ban2[r + c] = ban3[c - r + n] = false;
    }
}
```

n皇后问题拓展

若每行正好放一个皇后的条件不变,每列、两个对角线改成“最多放置两个皇后”,该如何回溯?

用 $ban1[i], ban2[i], ban3[i]$ 记录 $true/false$ 已经不够,可以把它们都改成 int 类型。

每当枚举 c 时,把 $ban[i] = 0$ 改成 $ban[i] \leq 1$。

第 r 行决定放在 c 位置后, $ban[i]$ += 1。

回溯后 $ban[i]$ −= 1。

#2248、受伤的皇后

题目描述

有一个 $n \times n$ 的国际象棋棋盘（n 行 n 列的方格图），请在棋盘中摆放 n 个受伤的国际象棋皇后，要求：

- 任何两个皇后不在同一行
- 任何两个皇后不在同一列
- 如果两个皇后在同一条 45 度角的斜线上，这两个皇后之间行号的差值至少为 3 。
  请问一共有多少种摆放方案。

输入描述

输入的第一行包含一个整数 n 。

其中，$1 \leq n \leq 10$ 。

输入

```
4
```

输出

```
2
```

若出现在同一条对角线上

只有在行号差值小于3时才认为冲突

```
bool check(int r, int c)
{
  for (int i = 1; i < r; i++)
    if (
      ans[i] == c
    || (ans[i] + i == c + r && r - i < 3)
    || (ans[i] - i == c - r && r - i < 3)
    )
      return false;
  return true;
}
```

#1643、2n皇后

题目描述

给定一个 $n \times n$ 的棋盘，棋盘中有一些位置不能放皇后。

现在要向棋盘中放入 n 个黑皇后和 n 个白皇后，使任意的两个黑皇后都不在同一行、同一列或同一条斜线（包括正负斜线）上，任意的两个白皇后都不在同一行、同一列或同一条斜线（包括正负斜线）上。问总共有多少种放法？ n 小于等于 8 。

输入格式

输入的第一行为一个整数 n ，表示棋盘的大小。

接下来 n 行，每行 n 个 0 或 1 的整数，如果一个整数为 1 ，表示对应的位置可以放皇后，如果一个整数为 0 ，表示对应的位置不可以放皇后。

输出格式

输出一个整数，表示总共有多少种放法。

样例输入1

```
4
1 1 1 1
1 1 1 1
1 1 1 1
1 1 1 1
```

样例输入2

```
4
1 0 1 1
1 1 1 1
1 1 1 1
1 1 1 1
```

样例输出1

```
2
```

样例输出2

```
0
```

#1643、2n皇后

黑皇后仅与黑皇后冲突,白皇后仅与白皇后冲突

将放皇后拆成两个步骤,如先放置黑皇后再放置白皇后

在dfs中加入一个参数标记皇后颜色,当把黑皇后放完后从第一行开始尝试放白皇后

```
void dfs(int row, int f)
{
  if (row == n + 1)
  {
    if (f)
    {
      ans++;
      return;
    }
    dfs(1, !f); //放完黑皇后,从第一行开始放白皇后
  }
  for (int col = 1; col <= n; col++)
    if (borad[row][col] && !ban1[col][f] && !ban2[row + col][f] && !ban3[row - col + n][f])
    {
      borad[row][col] = 0, ban1[col][f] = ban2[row + col][f] = ban3[row - col + n][f] = true;
      dfs(row + 1, f);
      borad[row][col] = 1, ban1[col][f] = ban2[row + col][f] = ban3[row - col + n][f] = false;
    }
}
```

#1083、全排列问题

问题描述

输出自然数 1 到 n 所有不重复的排列，即 n 的全排列，要求所产生的任一数字序列中不允许出现重复的数字。

输入格式

一个数 $n(1 \leq n \leq 9)$

输出格式

由 $1 \sim n$ 组成的所有不重复的数字序列，每行一个序列。

【输入样例

```
3
```

输出样例

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

#1083、全排列问题

思路一

考虑做n层的枚举

每一层选取一个之前未选取过的数

回溯时取消标记,时间复杂度$O(n \times n!)$

思路二

对于i个元素考虑与$i \sim n$范围内元素进行交换

那么第i个位置元素已被确定

只需要考虑$i+1 \sim n$范围内元素的排列

回溯时应当将第i个元素交换回来,时间复杂度$O(n \times n!)$

```
void permutation(int deep)
{
  if (deep == n + 1)
  {
    for (int i = 1; i <= n; i++)
      printf("%d%c", ans[i], " \n"[i == n]);
    return;
  }
  for (int i = 1; i <= n; i++)
    if (!vis[i])
    {
      vis[i] = true;
      ans[deep] = a[i];
      permutation(deep + 1);
      vis[i] = false;
    }
}
```

```
void permutation(int idx)
{
  if (idx == n + 1)
  {
    for (int i = 1; i <= n; i++)
      printf("%d%c", a[i], " \n"[i == n]);
    return;
  }
  for (int i = idx; i <= n; i++)
  {
    swap(a[idx], a[i]);
    permutation(idx + 1);
    swap(a[idx], a[i]);
  }
}
```

#1083、全排列问题

思路三

使用STL中的next_permutation或prev_permutation

bool next_permutation(first, last)可求出下一个排列

bool pre_permutation(first, last)可求出上一个排列

若需要求出全部的排列,**需要对元素进行排序**

求出全部的排列,时间复杂度$O(n \times n!)$

```
sort(a + 1, a + 1 + n);
do
{
  for (int i = 1; i <= n; i++)
    printf("%d%c", a[i], " \n"[i == n]);
} while (next_permutation(a + 1, a + 1 + n));
```

```
sort(a + 1, a + 1 + n, greater<int>());
do
{
  for (int i = 1; i <= n; i++)
    printf("%d%c", a[i], " \n"[i == n]);
} while (prev_permutation(a + 1, a + 1 + n));
```

字典序法

设P是1 ~ n的一个全排列:p=p1 p2 ..... pn = p1 p2 ..... pj-1 pj pj+1 ..... pk-1 pk pk+1 ..... pn

1）从排列的右端开始，找出第一个比右边数字小的数字的序号j（j从左端开始计算），即 j=max{ i | pi<pi+1 }

2）在pj的右边的数字中，找出所有比pj大的数中最小的数字pk，即k=max{ i | pi>pj}（右边的数从右至左是递增的，因此k是所有大于pj的数字中序号最大者）

3）对换pi，pk

4）再将pj+1 ..... pk-1 pk pk+1 ..... pn倒转得到排列p'=p1 p2 ..... pj-1 pj pn ..... pk+1 pk pk-1 ..... pj+1，这就是排列p的下一个排列。

```cpp
#include <bits/stdc++.h>
using namespace std;
int n = 4, a[21] = {1, 2, 3, 4};
bool next_permutation()
{
    int pos = n - 1, i;
    while (pos - 1 >= 0 && a[pos - 1] > a[pos])
        pos--;
    pos--;
    if (pos < 0)
        return false;
    for (i = pos + 1; i < n && a[i] > a[pos]; i++)
        ;
    i--;
    swap(a[i], a[pos]);
    reverse(a + pos + 1, a + n);
}
int main()
{
    do
    {
        for (int i = 0; i < n; i++)
            cout << a[i] << " \n"[i == n - 1];
    } while (next_permutation());
    return 0;
}
```

#1828、优美连接

题目描述

给出 n 个 2 位整数 $(1 \le n \le 10)$，将这n个数拼成一个长 $2n$ 位长整数：$y = x_1x_2x_3.....x_{2n}$。

然后进行计算：$d = |x_1 - x_2| + |x_2 - x_3| + ... + |x_{2n-1} - x_{2n}|$

问题：当 n 个数给出之后，找出一种拼接方法，使 d 最小。

例如：$n = 3$ 时，三个数分别为：$26, 17, 34$ 拼接方法有：

$$261734 = |2-6| + |6-1| + |1-7| + |7-3| + |3-4| = 20$$
$$263417 = |2-6| + |6-3| + |3-4| + |4-1| + |1-7| = 17$$
$$.....$$
$$173426 = |1-7| + |7-3| + |3-4| + |4-2| + |2-6| = 17$$

其中最小 d 为 17

输入

第一行一个整数 n，第二行 n 个整数

输出

一个整数，即最小的 d

输入#1

```
3
26 17 34
```

输出#1

```
17
```

直接列举出所有的排列

对于每个排列计算出差值d

取最小值输出即可

```
sort(a, a + n);
do
{
  ans = min(ans, cal());
} while (next_permutation(a, a + n));
cout << ans;
```

谢谢观看