6-Recursion 题解

by 小作文 gg

2021 年 11 月 25 日

目录

本周题目较难，考虑到大多数同学是第一次学习递归，因此题解中将会给出一些（比较）详细的讲解。

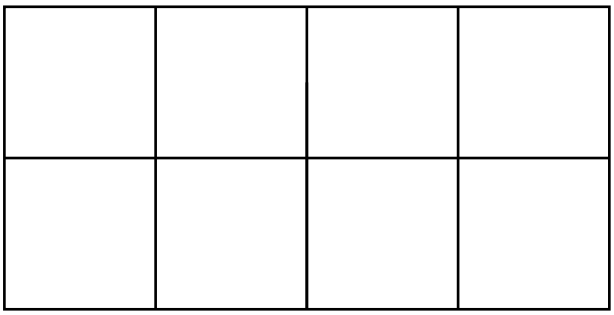
1 三角形 (triangle.c)

本周的较难题。

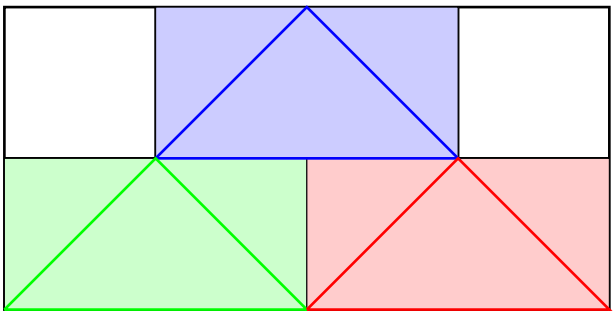
首先需要需要明确的一点是递归之所以困难，是因为你人脑中呈现出来的顺序和真正计算机执行时候的顺序是不太一样的，为了达成人脑与计算机的“一致”，我们需要从这些题目中学习多种“技巧”，而此题中的“技巧”就是开一个足够大的二维 char 数组 $map[][]$，我们在操作的时候仅仅对 map 中的某些位置填上合适的元素，最后再将整个地图输出。

那么现在我们有了一张足够大的画布，我们需要在这块画布中填上一个规模为 n 的三角形阵，那么接下来有两种情况：

- $n = 1$，此时我们认为问题的规模足够小，直接在这块画布中填上三角形即可。
- $n \geq 2$，此时我们可以利用一下这个图形的构成，假设下面是我们的画布。

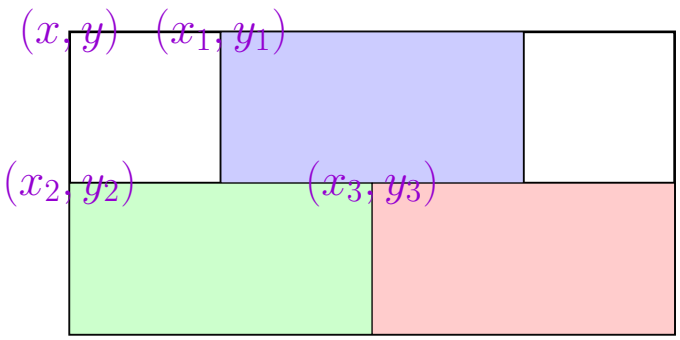


我们需要做的是利用这个图形的结构，当我们近似地把这块画布均等地分成 8 个部分的时候，我们会发现左上角和右上角的两个部分是不填任何东西的，而需要填东西的部分是下图中画出的浅蓝色、浅绿色和浅红色的三块较小的画布，而在这三块矩形的小画布上，问题的规模从 n 缩减为了 $n - 1$。



因为我们在上一步的操作中需要缩减画布的大小，将一块大的画布（不完全地）拆分为三个相同大小的画布，这时我们发现如果我们仅仅用一个问题的规模参数 n 来描述这个问题，是不够的。因为我们在做的过程中需要知道当前在二维数组的哪个区间里面在做画图的工作，因此我们还需要一个坐标指示当前的位置是在二维数组中是哪一块。

不妨设 $paint(n, x, y)$ 函数可以在以 (x, y) 为左上角的矩形中解决规模为 n 的画三角形问题，根据上面的讨论，我们可以写出 $n = 1$ 的边界情况；而 $n \geq 2$ 的非边界情况，我们可以根据左上角的坐标 (x, y) 和问题的规模 n 分别对于蓝色、绿色、红色的小矩形计算出其左上角的坐标 $(x_1, y_1), (x_2, y_2), (x_3, y_3)$ ，然后我们递归地去调用 $paint(n-1, x_1, y_1), paint(n-1, x_2, y_2), paint(n-1, x_3, y_3)$ 即可。



至于 (x, y) 和 (x_1, y_1) 这样的坐标之间到底差了多少，以及画布的规模具体需要多大这种问题。这不是我们在设计递归函数的过程中所需要特别关心的事情，显然是一个与 n 有关的函数 $f(n)$ ，大家自己动手计算一下即可。

```c
/* triangle.c */
#include <stdio.h>

char ans[2010][3010];

void solve(int n, int x, int y) {
    if (n == 1) {
        ans[x][y+1] = ans[x+1][y] = '/';
        ans[x][y+2] = ans[x+1][y+3] = '\\';
        ans[x+1][y+1] = ans[x+1][y+2] = '_';
        return;
    }
    solve(n - 1, x, y + (1 << (n - 1)));
    solve(n - 1, x + (1 << (n - 1)), y);
    solve(n - 1, x + (1 << (n - 1)), y + (1 << n));
}

int main() {
    int n;
    scanf("%d", &n);
    for (int i = 1; i <= (1 << n); i++)
        for (int j = 1; j <= (1 << (n+1)); j++)
            ans[i][j] = ' ';
```

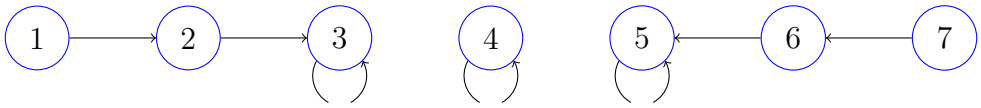
```
    solve(n, 1, 1);
    for (int i = 1; i <= (1 << n); i++) {
        for (int j = 1; j <= (1 << (n+1)); j++)
            printf("%c", ans[i][j]);
        printf("\n");
    }
    return 0;
}
```

一个需要注意的小细节是因为反斜杠无法正常打出，所以我们使用两个反斜杠连写表示转义运算。

2 不要抄袭 (plagiarize.c)

本周较简单的一道题。

我们用一个点来代表一个同学（1 号点代表第一位同学，2 号点代表第二位同学，以此类推），若 i 号同学抄袭了 j 号同学，则用一个箭头从 i 号点指向 j 号点，下图为样例。



上面这个东西其实已经被我们抽象为了一个有向图，关于有向图的定义，我们将在下学期离散数学的课程中学到，大家在此不必深究。

回到这道题本身，怎么找到那个抄袭的源头呢？如果要求解 i 的源头，我们可以从 i 号点出发，沿着箭头[1]在这个图中“走路”，直到我们走到某个点发现它就是一个源头（自己走回自己），当前所在的点就是 i 号的抄袭源头。因为题中已经保证了不存在箭头所组成的“环”，所以这种方法一定有解。

```
/* plagiarize.c */
#include <stdio.h>

#define MX 100010

int nxt[MX], n;

int f(int x) {
    if (nxt[x] == x)
        return x;
    return f(nxt[x]);
}
```

[1]题目中已经保证了一个点只会有一个可走的箭头。

```
13
14 int main() {
15     scanf("%d", &n);
16     for (int i = 1; i <= n; i++)
17         scanf("%d", &nxt[i]);
18     for (int i = 1; i <= n; i++)
19         printf("%d ", f(i));
20     return 0;
21 }
```

这份标程中使用了 nxt 来代替原来的 $next$ 单词，这是因为这个单词在一些库里面较为常见，使用 nxt 可以有效避免因为重新定义而编译错误的问题。

3 杨辉三角形 (yanghui.c)

简单题。

题意即为求组合数 $\binom{a-1}{b-1}$，根据提示中给出的公式直接进行递归，需要考虑一下的是递归的边界情况，因为一个求解的函数 $C(a,b)$ 会访问 $C(a-1,b-1)$ 和 $C(a-1,b)$，第一种情况下 a,b 会同步变小，而第二种情况只有 a 会逐渐变小，因为我们保证了一开始给出的 $b \le a$，所以有两种边界情况：

- $b=a$，此时 $C(a,b)=1$。
- $b=0$，此时 $C(a,b)=1$。

大家也可以写出很多不同的边界条件，正确的条件都能在数学上证明等价。

```
1  #include <stdio.h>
2  #define ll long long
3
4  ll sol(ll a, ll b) {
5      if (b == 0) return 1;
6      if (a == b) return 1;
7      return sol(a - 1, b - 1) + sol(a - 1, b);
8  }
9
10 int main() {
11     ll a, b;
12     scanf("%lld%lld", &a, &b);
13     printf("%lld\n", sol(a - 1, b - 1));
14     return 0;
15 }
```

本题需要使用 long long，因为组合数的增长速度很快。

4 填格子 (tile.c)

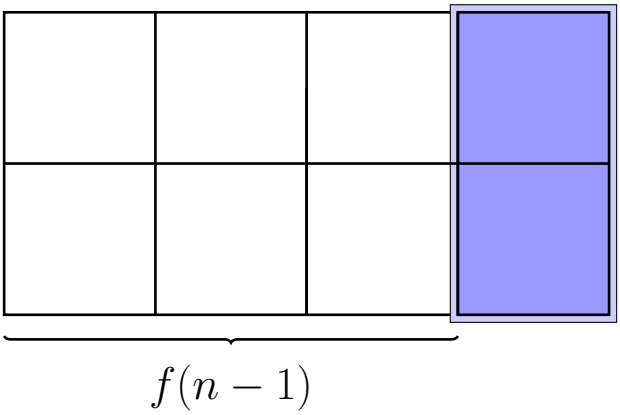
经典小学数学题，较简单。

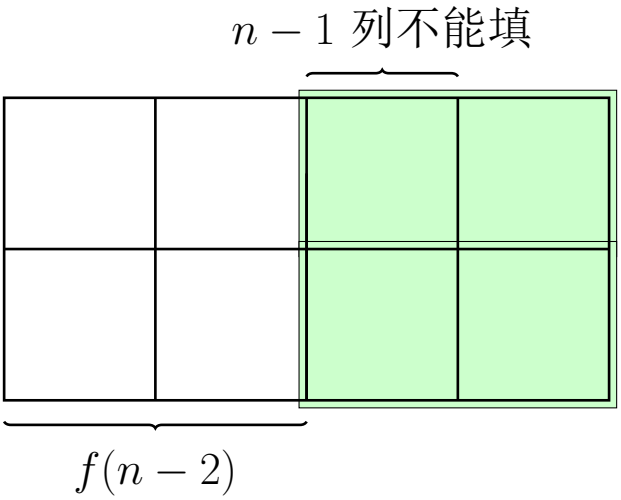
本题需要同学们进行一点数学推导，最后得出的公式非常简单，设 $f(n)$ 为填 $2 \times n$ 格子的方案数，则 $f(n)$ 满足

$$f(n) = f(n-1) + f(n-2) \qquad (n \geq 2)$$

边界条件为 $f(0) = f(1) = 1$。

简单推导过程如下，考虑一个 $2 \times n$ 的格子，考虑它的最后一列怎么填，只有一种方法，即在最后填上 2×1 的格子，这种情况的状态数为 $f(n-1)$；但是我们还需要考虑它的最后两列横着填了两个 1×2 的格子的情况，这种情况的方案数为 $f(n-2)$ 。






```c
/* tile.c */
#include <stdio.h>

int f(int n) {
    if (n == 0)
        return 1;
    if (n < 0)
        return 0;
```

```
    return f(n - 2) + f(n - 1);
}

int main() {
    int n;
    scanf("%d", &n);
    printf("%d\n", f(n));
    return 0;
}
```

5 最大值区间 (max.c)

可能算是本周的中档题?

仔细阅读题意之后，我们可以设计一个 $solve(l,r)$ 函数来处理闭区间 $[l,r]$ 中的情况。我们先假设这个区间合法，即 $l \le r$ ，按照题意，我们可以找到 $a[l:r]$ 中的最大值的下标 p ，然后我们就得到了第 p 个答案 $[l,r]$ ，接着递归调用 $solve(l,p-1)$ 和 $solve(p+1,r)$ 。

上面的过程都已经在题意中给出，问题在于如何方便地设计递归的边界，因为我们对于一个合法的区间 $[l,r]$ 拆解之后可能会出现：有两个合法的新区间、有一个合法的新区间、有零个合法的新区间三种情况，所以我们可以在进入 $solve$ 的第一步判断那些不合法的情况，即 $l > r$ 时直接 return 。

容易发现，执行一次“删数”的操作就能求得被删的那个数的答案，我们在 $main()$ 中只需要调用 $solve(1,n)$ 一次就可以求得所有答案，因为这些答案求出的顺序不一致，利用一个全局数组保存最后一起输出即可。

```
/* max.c */
#include <stdio.h>

#define N 100005

int n, a[N], tmp[N], ans[N][2];

void solve(int l, int r) {
    if (l > r) return;
    int p = l;
    for (int i = l + 1; i <= r; i++)
        if (a[i] > a[p])
            p = i;
    ans[p][0] = l, ans[p][1] = r;
    solve(l, p - 1), solve(p + 1, r);
}
```

```
17
18 int main() {
19     scanf("%d", &n);
20     for (int i = 1; i <= n; i++)
21         scanf("%d", &a[i]);
22     solve(1, n);
23
24     for (int i = 1; i <= n; i++)
25         printf("%d %d\n", ans[i][0], ans[i][1]);
26     return 0;
27 }
```

需要说明的是，有些同学对于每个数都调用一次 $solve()$ 函数计算答案，这样的做法会造成大量冗余的计算，在本题中也会超时。

6 数字分解 (decomposition.c)

递归的经典应用，可能是本周的最难题，成功被选为问求的某道例题（手动叹气）。

要解决这道题，我们从初中到高中学到的算术以上的数学知识可能只会成为阻碍，让我们返璞归真，回到小学或者幼儿园那个刚刚学会加法的年龄。现在思考一个问题：如果你对这个数的结构、性质一无所知，但是你想要分解它，你会怎么做?

我认为最简单的做法是“尝试”。面对要分解一个数 n 的问题，我们天然地先写下一个最小的数 1，假如这个数就是 1，那么分解已经完成；假如这个数比 1 稍大一点，我们接下来要做的事情就变成了分解 $n-1$ 这件事情。

这时我们会发现，我们尝试分解 n 和尝试分解 $n-1$ 的过程是非常近似的，只是要分解的这个数变了。从编程的视角去审视一下，我们发现这是原来问题的一个**子问题**。

考虑尝试枚举**一个数**的过程，对于每一个 $i \le n$，我们都可以尝试将 n 先分解为 i，然后继续分解 $n-i$ 这个数，将后面所得的答案（可能有多个）写在 i 的后面，就得到了分解 n 这个问题的所有答案。

我们尝试设计一个函数 $Devide(n)$ 可以帮我们解决“分解 n 这个数”的问题，根据上面的描述，我们枚举所有 $\le n$ 的 i，然后调用 $Devide(n-i)$ 就可以解决 n 的问题了。

到这里我们考虑两个细节问题：

- 容易发现 $Devide$ 是一个递归函数，既然是递归函数就一定有边界（出口），这个 $Devide$ 的出口到底在哪里？
- 怎么用 C 语言实现“先写下 i，再将 $Devide(i)$ 所给出的答案写在 i 的后面”这件事情？

对于第一个问题，我们很容易想到这个递归的边界就是 $n=0$，对于 $n=0$ 我们认为这个数不能再分解了，此时前面写下的所有数就构成了一个合法的答案；而当我们尝试解决第一个问题的时候，我们发现第二个问题的答案也呼之欲出了，“此时前面写下的所有数就构成了一个合法的答案”已经

告诉了我们怎么保存答案，我们考虑使用一个全局数组 $ans[]$ 和全局变量 len，约定当进入 $Devide(n)$ 的时候之前写下的答案保存在 $ans[0:len)$ [2]中，有了这样的约定，我们可以在每一次的“尝试”中都将自己当前尝试的数“写下”，保存到全局数组中，直到递归到 0 的时候，我们输出答案。

```
1  int ans[], len = 0;
2
3  void Devide(int n) {
4      if (n == 0) {
5          for (int i = 0; i < len; i++) printf("%d ", ans[i]);
6          printf("\n");
7          return;
8      }
9
10     for (int i = 1; i <= n; i++) {
11         ans[len] = i, len++;
12         Devide(n - i);
13     }
14 }
```

有了想法，我们写出以上代码，但是我们在实现的过程中忽略了一件重要的事情：我们是连续不断地在同一个位置进行多次尝试，而我们**在尝试过程中对全局数组的影响是只增加不消除的**。这其实是一件影响正确性的事情，我们在分解 n 的过程中，应当依次尝试 1，2，3 $\cdots$，但是这些数都应该填在同一个位置，所以当一次尝试结束之后，我们应当**把这次尝试的影响消除**，具体到代码实现上，我们应该在 11 行 $Devide(n-i)$ 返回之后，执行一次 $len--$，以便在 i 循环到下一个位置的时候可以在与上一次相同的位置填入此次需要尝试的下一个数。

```
1  int ans[], len = 0;
2
3  void Devide(int n) {
4      if (n == 0) {
5          for (int i = 0; i < len; i++) printf("%d ", ans[i]);
6          printf("\n");
7          return;
8      }
9
10     for (int i = 1; i <= n; i++) {
11         ans[len] = i, len++;
12         Devide(n - i);
13         len--;
```

[2] $ans[0,len)$ 这样的写法仅仅为了表示左闭右开，实际上并不规范。

```
    }
}
```

这样的代码已经可以跑了，大家可以自己尝试一下，上面的这段代码会跑出怎样的结果。我这里截取 $n = 5$ 的部分输出如下。

```
1 1 1 1 1
1 1 1 2
1 1 2 1
......
```

我们发现这并不是我们想要的输出，因为有诸如 1 1 2 1 这样的东西存在。我们很难说 1 1 2 1 不是 5 的一个合法的分解，只是在本题中，我们认为 1 1 2 1 和 1 1 1 2 是等价的。怎么办呢？我们可以人为地加入一个限制：**让我们后面填入的数一定要不小于前面填入的数**。容易发现，当执行到 $Devide(n)$ 时，前一个填入的数是 $ans[len-1]$，于是我们修改代码。

```
int ans[], len = 0;

void Devide(int n) {
    if (n == 0) {
        for (int i = 0; i < len; i++) printf("%d ", ans[i]);
        printf("\n");
        return;
    }

    int last = (len == 0) ? 1 : ans[len - 1];
    for (int i = last; i <= n; i++) {
        ans[len] = i, len++;
        Devide(n - i);
        len--;
    }
}
```

这个步骤也可以通过增加一个函数的参数来实现，不如说还是这样的实现比较多。

由于本题的核心代码已经给出，不再给出标程。还有一点点细节问题（比如：ans 这个数组要开多大?），请同学们自己思考。

7 积分 (integration.c)

本周的第二道阅读理解题，主要考察大家在复杂一点的情况下如何进行合理的抽象。如果大家读懂了题目应该不难，直接给出标程。

```c
/* integration.c */
#include <stdio.h>
#include <math.h>

#define SIZE 30

int n;
double p;
double a[SIZE];
double A, B;
double f(double x)
{
    double ans = a[0];
    double t = x;
    for (int i = 1; i <= n; i++)
    {
        ans += a[i] * t;
        t = t * x;
    }
    return pow(ans, p);
}
double sh(double l, double r)
{
    return (4.0 * f((l + r) / 2.0) + f(l) + f(r)) * (r - l) / 6.0;
}


double Deal(double l, double r, double ex)
{
    double mid = (l + r) / 2.0;
    double SL = sh(l, mid);
    double SR = sh(mid, r);
    double S = sh(l, r);
    if (fabs(SL + SR - S) <= 15.0 * ex)
        return SL + SR + (SL + SR - S) / 15.0;
    return Deal(l, mid, ex / 2.0) + Deal(mid, r, ex / 2.0);
}
int main()
{
```

```
40      scanf("%d %lf", &n, &p);
41      for (int i = 0; i <= n; i++)
42          scanf("%lf", &a[i]);
43      scanf("%lf %lf", &A, &B);
44      printf("%.6lf\n", Deal(A, B, 1e-5));
45      return 0;
46  }
```

这里我们对一些细节进行解说：

- 积分函数在分治处理的过程中精度是会发生变化的，具体来说是进入下一层递归函数的时候需要 /2 ，这一点在题目中已经给出，但是有不少同学没有注意到；另一件事是如果大家每次都调用 $eps/pow(2,k)$ 来计算误差的话会得到超时的结果，希望大家下次能采取更快一点的写法。
- 求浮点数的绝对值可以使用 $fabs()$ 这个函数，它定义在 $math.h$ 库中；**不能使用**求整数绝对值的 $abs()$ 函数，它的返回值是一个整型，严重影响答案的正确性。

8 写在最后

终于写完了，这周题解写死我了。qaq