计基第二次上机题解

0x00 前言

给出的所有题解都以**容易理解**和**不使用还没学到的知识**为标准, 可能会有很大的优化空间.(无论是运行时间上还是代码美观程度上.) 望大佬轻喷.

应该是写的非常非常详细了.

个人对题目难度排下序: D < A < C < B.

0x01 A题 2的幂(powerof2.c)

虽说是签到题, 题目看上去十分简单, 样例也十分弱智.

但实际上做起来还是有一点难度的.

如果你因为没有仔细读完题而写出下面的代码的话, 肯定是不可能过的.

```
printf("%lld", (long long) pow(2, n));
```

因为... 题目最后写了 0 <= n <= 100000.

即使我们使用 C 中最大的数据类型 unsigned long long (无符号 64 bit 整数), 也只能用于计算 2^64 -1 以内的数. (如果算出来的话是 18446744073709551615, 20 位.) 这远远不足 2^100000. (你可以用数学方法发现它是一个 30103 位的数字).

你可能会想, 为什么 C 语言里不能发明一个可以存下更大数字的数据类型呢?

~~比如发明一个 long long long long long long long long, 可以直接存下 2^100000, 这样这题就可以直接秒杀了.~~

这里可以联系一下计基的知识. 我们写的 C 语言的语句最后都会被 C 语言编译器编译成汇编语言(比如 DLX), 然后汇编器将汇编语言生成对应的机器码. 最后直接丢给 CPU 执行.

但是 DLX 是 32 位的, 也就是一个寄存器中最多储存 32 个比特, 也恰好就是 int 的大小. 如今我们使用的 Intel / AMD 的 CPU 的 x86_64 指令集是 64 位的, 对应的也就是 long long 的大小.

所以 CPU 的硬件电路限制了 C 中整数的大小, 而为 CPU 设计添加电路的成本是极高的, 所以也不可能设计出一个 100000 位的 CPU 专门用来算数.

但是, 的确有很多时候我们需要计算大于 2^64 - 1 的数字, 这么大的数字又不可能手算, ~~要不然发明计算机干嘛.~~

我们只能想个办法, 用**不超过 2^64 - 1 的数字最终表示出 2^100000 的计算结果**.

自然地, 我们会想到, 既然不能直接使用 C 内置的计算直接算出最终结果, 我们只能自己想办法实现指数的运算. 计算方法其实我们小学就已经学过了 - **列竖式**.

如果要计算 2^100000, 也就是 2 * 2 * 2 * 2 * ..... (共计 10w 个 2 相乘). 那我们只要写一段程序能将一个数字乘 2, 把这个过程重复 n 次, 得到的结果就是 2^n 了.

我们计算一个数乘二的过程就是 - **每一位都乘二, 如果超过 10 就进一位.**

```
#define LEN 30104
int result[LEN];  // 开一个数组用来存每一位数，开在 main 外面会自动填 0.

// 乘 2
for(int i = 0; i < LEN; i++) {
    result[i] *= 2;
}
// 进位
for(int i = 0; i < LEN; i++) {
    if (result[i] >= 10) {
        result[i] -= 10;
        result[i + 1] += 1;
    }
}
```

没错，代码的核心部分就上面这一点... 所以说它是签到题，其实也不过分.

加上输入输出，就可以得到一份 **AC 代码**: (其他需要注意的小细节都写在注释里了.)

```
#include <stdio.h>
#define LEN 30104
int result[LEN];  // 开一个数组用来存每一位数，开在 main 外面会自动填 0.

int main() {
    int n;
    scanf("%d", &n);

    result[0] = 1;  // 别忘了开始设置个 1

    for (int j = 0; j < n; j++) {  // 重复 n 次
        // 乘 2
        for(int i = 0; i < LEN; i++) {
            result[i] *= 2;
        }
        // 进位
        for(int i = 0; i < LEN; i++) {
            if (result[i] >= 10) {
                result[i] -= 10;
                result[i + 1] += 1;
            }
        }
    }

    // 输出：由于这个数字其实是"倒着"的，所有从下标大的向下标小的遍历，遇到第一个不是 0 的就输出剩下所有的数.
    for (int i = LEN - 1; i >= 0; i--) {
        if (result[i] > 0) {
            for (int j = i; j >= 0; j--) {
                printf("%d", result[j]);
            }
            return 0;  // 输出完成直接结束即可
        }
    }

    return 0;  // 理论上不会执行到这里了
}
```

以上的这种操作被称为**高精度计算**. 是一种在计算较大的数字中要使用的方法.

0x02 B题 人工智能(ai.c)

这题题意其实不难理解, 就是把数字转换为英语中的表示.

想要过应该不难 (~~大不了把 100 以内都打表~~), 但是想把代码写的好看还挺难的. 所以下面会介绍两个简单又好用的函数来解决这个问题.

思路: 因为英文中的 thousand, million, billion 都是三位一分隔的, 所以可以先把输入的数字三个三个分为一段. 比如 `1234567891`, 写成 `1,234,567,891`.

然后就成了 `1 Billion 234 Million 567 Thousand 891`. 接下来只要想办法把一个三位数转成英文表示, 用 234 567 和 891 的英文表示替换掉原来的数字就行了.

先写一个表示三位数的方法:

这里不得不介绍一个很好用的函数: sprintf, 看名字和 printf 很像, 其实功能也很像.

就是把 printf 原本要直接输出的内容保存到一个指定的字符串里. 这样就可以简单的实现字符串的拼接了. 具体用法看代码就明白了.

```
// 这里不得不打表了, 毕竟电脑又不认识英文. (注意: 字符串本来就是 char[], 所以存字符串的数组需要是 char[][].)
char t1[20][20] = {"Zero", "One", "Two", "Three", "Four", "Five", "Six",
"Seven", "Eight", "Nine", "Ten", "Eleven", "Twelve", "Thirteen", "Fourteen",
"Fifteen", "Sixteen", "Seventeen", "Eighteen", "Nineteen"};

// 前两个字符串只是占个位置, 为了使后面的下标对应.
char t2[10][20] = {"???", "???", "Twenty", "Thirty", "Forty", "Fifty", "Sixty",
"Seventy", "Eighty", "Ninty"};


// 把一个不超过 2 位的 num 的英文表示存到 target 字符串里.
// 表示 3 位的方法中重复使用到表示 2 位, 所以单独提取成一个方法.(而不是复制一遍.)
void sprint_2digit_num(int num, char target[]) {
    // 下面这段就是根据英语的规则直接输出.
    // 注意判断 末尾的 0 无需输出.
    if (num >= 1 && num <= 19) {
        sprintf(target, "%s", t1[num]);  // sprintf 只比 printf 多了第一个参数, 表示储存结果的位置.
    } else if (num >= 20 && num <= 99) {
        if (num % 10 == 0) {
            sprintf(target, "%s", t2[num / 10]);
        } else {
            sprintf(target, "%s %s", t2[num / 10], t1[num % 10]);
        }
    }
}

// 把一个不超过 3 位的 num 的英文表示存到 target 字符串里.
void sprint_3digit_num(int num, char target[]) {
    if (num >= 1 && num <= 99) {  // 是个两位数
        sprint_2digit_num(num, target);
    } else if (num >= 100) {
        if (num % 100 == 0) {
            sprintf(target, "%s Hundred", t1[num / 100]);
        } else {
```

```
            char s1[100];  // 缓存一下结果
            sprint_2digit_num(num % 100, s1);
            sprintf(target, "%s Hundred %s", t1[num / 100], s1);
        }
    }
}
```

然后就是把整个数字切割之后再合并输出的方法了.

关于如何合并两个字符串, 这里再介绍一个字符串处理的非常好用的函数: strcat(str1, str2); 可以把字符串 str1 和 str2 拼接的结果保存到 str1 中.

```
void sprint_english_num(int num, char target[]) {
    // 这里当然用数组更好, 但我也想偷个懒.
    int part1 = 0, part2 = 0, part3 = 0, part4 = 0;
    part1 = num % 1000;
    num /= 1000;
    part2 = num % 1000;
    num /= 1000;
    part3 = num % 1000;
    part4 = num / 1000;

    // 把各个部分变成英文的
    char s1[200], s2[200], s3[200], s4[200];
    sprint_3digit_num(part1, s1);
    sprint_3digit_num(part2, s2);
    sprint_3digit_num(part3, s3);
    sprint_3digit_num(part4, s4);

    // 加上 Billion, Million, Thousand (注意结尾的空格.)
    strcat(s4, " Billion ");
    strcat(s3, " Million ");
    strcat(s2, " Thousand ");

    // 连起来 (注意: 像 1,000,123 中间一段如果全是 0, 直接跳过即可, 无需输出 Zero
Thousand.)
    if (part4 > 0) {
        strcat(target, s4);
    }
    if (part3 > 0) {
        strcat(target, s3);
    }
    if (part2 > 0) {
        strcat(target, s2);
    }
    if (part1 > 0) {
        strcat(target, s1);
    }
}
```

AC 代码: 加个输入输出就好了. 不再复制一遍. 别忘了 strcat 需要 `#include <string.h>`.

0x03 C题 快乐星球上的蚂蚁大战(antwar.c)

这道题需要仔细耐心读题, 理解题目描述的过程.

不难发现, 因为所有的蚂蚁行进速度相同, 所以只有**相邻的**蚂蚁**有可能**发生战斗.

那我们根据题目意思可以先写出如下的判断过程.

```
for (int i = 0; i < num; i++) {
    if (a[i] > 0 && a[i + 1] < 0 && i + 1 < num) {
        if (a[i] > -a[i + 1]) {
            // 第 i + 1 号蚂蚁死了
        } else if (a[i] < -a[i + 1]) {
            // 第 i 号蚂蚁死了
        } else {
            // 第 i 和第 i + 1 号蚂蚁一起死了
        }
    }
}
```

接下来问题就变成了, 如何描述 "一只蚂蚁死了" 这件事.

我们可以选择把蚂蚁的数值设置为 0, 代表这是一个空位.

但这样的话我们判断相邻的蚂蚁时就需要考虑到空位的影响, 也就需要修改上面的 for 循环, 比较麻烦.

所以我的解决方法是: **把这只蚂蚁之后的蚂蚁全部向前移动一格, 让蚂蚁的排列仍然是"紧凑的".**

写成 C 语言代码就是这样的.

```
void delete_ant(int ants[], int index, int num) {
    for (; index < num - 1; index++) {
        ants[index] = ants[index + 1];
    }
}
```

上面的部分完成了一轮作战, 再加个 for 循环让所有蚂蚁都作战完毕, 就可以得到 **AC 代码**:

```
#include <stdio.h>
#define MAX 1000000

int a[MAX];

void delete_ant(int ants[], int index, int num) {
    for (; index < num - 1; index++) {
        ants[index] = ants[index + 1];
    }
}

int main() {
    // 输入
    int num;
    scanf("%d", &num);
    for (int i = 0; i < num; i++) {
        scanf("%d", &a[i]);
    }
```

```
    for (int j = 0; j < 10000; j++) {  // 这里最多有 10000 只蚂蚁，所以最多作战 10000 次.
        for (int i = 0; i < num; i++) {
            if (a[i] > 0 && a[i + 1] < 0 && i + 1 < num) {
                if (a[i] > -a[i + 1]) {  // 第 i + 1 号蚂蚁死了
                    delete_ant(a, i + 1, num);
                    num--;  // 蚂蚁数量已经减少了 1
                } else if (a[i] < -a[i + 1]) {  // 第 i 号蚂蚁死了
                    delete_ant(a, i, num);
                    num--;
                } else {  // 第 i 和第 i + 1 号蚂蚁一起死了
                    delete_ant(a, i + 1, num);  // 注意这里如果先删除第 i 号蚂蚁的话，i + 1 会代替原来 i 的位置. 所以先调用 delete(i + 1) 比较直观.
                    num--;
                    delete_ant(a, i, num);
                    num--;
                }
            }
        }
    }

    // 输出
    for (int i = 0; i < num; i++) {
        printf("%d ", a[i]);
    }
    return 0;
}
```

0x04 D题 DiverseString(diversestring.c)

题目是英文的, 但也不难理解, 翻译成中文就一句话: 输入的字符串的所有字母要在字母表中**连续但不重复**.

比如 `fced`, 这四个字母排序后在字母表中是连续出现的: `cdef`.

特殊情况是, 如果只有一个字母, 也算是连续不重复.

那我们就用最朴素的想法, 直接统计一下字符串中出现的字符, 再判断一下是否符合题目条件即可.

```
int is_diverse(const char str[]) {
    // 先统计，和 C 语言那道字母统计图一样
    int stat[26];
    for (int i = 0; i < 26; i++) stat[i] = 0;  // 初始化
    int p = 0;
    char c = str[0];
    while (c != '\0') {
        stat[c - 'a']++;
        p++;
        c = str[p];
    }

    int flag = 0;  // flag 用于记录当前的状态
    for (int i = 0; i < 26; i++) {
        if (stat[i] > 1) return 0;  // 重复
        if (flag == 0 && stat[i] == 1) flag = 1;  // 有字母开始出现，将 flag 设置为 1. (代表当前已经开始连续.)
```

```
        if (flag == 1 && stat[i] == 0) flag = 2;  // 连续部分已经结束, 将 flag 设置为
2.
        if (flag == 2 && stat[i] == 1) return 0;  // flag 已经为 2 而又出现下一个连续
部分. 代表不符题意.
    }

    return 1;  // 符合题意
}
```

再加上输入输出, 就可以得到 **AC 代码**:

```
#include <stdio.h>

int is_diverse(const char str[]) {
    // 同上, 略
}

int main() {
    int len;
    scanf("%d", &len);
    for (int i = 0; i < len; i++) {
        char s[101];
        scanf("%s", s);
        printf("%s\n", is_diverse(s) ? "Yes" : "No");  // 注意是 Yes, 不是 YES 也不
是 yes!
    }
    return 0;
}
```

0x05 附加题

其实附加题比较简单. 但来做附加题的一定都是大佬吧.

我们来欣赏一个一行秒杀的版本.

```
print(*map(lambda x:' '.join(map(lambda
y:str(y),x)),sorted(set((__import__("itertools")).permutations([int(i) for i in
[input(),input().split(' ')][1]])))),sep='\n')
```