E8-2021级航类-第八次练习赛题解

A 要结课啦!

难度	考点
1	无

题目分析

这题还需要分析?

示例代码

```
//
// Created by moc85 on 2022/5/14.
//

#include <stdio.h>

int main()
{
    int n = 0;
    scanf("%d", &n);
    if (n == 1) {
        printf("基础运算");
    } else if (n == 2) {
        printf("基本数据处理");
    } else if (n == 3) {
        printf("程序结构");
    } else if (n == 4) {
        printf("函数");
    } else if (n == 5) {
        printf("数组");
    } else if (n == 6) {
        printf("指针");
    } else if (n == 7) {
        printf("结构与联合");
    } else if (n == 8) {
        printf("综合应用");
    }

    return 0;
}
```

B 简单字符2.0

难度	考点
1	字符串

题目分析

本题的题目要求十分清晰，题目的输入是一行字符串，且只包含大小写英文字母。输入的字符串在符合以下三种情况时，认为字符串的大写用法是正确的：

- 全部字母都是大写，比如 "BUAA" ；
- 全部字母都是小写，比如 "accoding"；
- 首字母大写，其他字母小写， 比如 "Ceremony"

我们只需要读入字符串，然后判断字符串是否属于三种情况之一就可以了。本题情况不多，我们可以采用函数式编程，使得程序整体的思路更加清晰，具体做法参考以下的示例代码。

示例代码

```
#include <stdio.h>
#include <ctype.h> // islower, isupper这两个函数在该头文件中定义

// 判断字符串s是否所有字母都是大写
int all_capitalized(char *s) {
    // 用指针的方式遍历整个字符串，注意字符'\0'（字符串结尾标志）的ascii码为0
    for (char *c = s; *c != 0; c++) {
        if (islower(*c)) {
            return 0;
        }
    }
    return 1;
}

// 判断字符串s是否所有字母都是小写
int no_captialized(char *s) {
    for (char *c = s; *c != 0; c++) {
        if (isupper(*c)) {
            return 0;
        }
    }
    return 1;
}

// 判断字符串s是否只有首字母大写
int first_capitalized(char *s) {
    if (islower(*s)) {
        return 0;
```

```
    } else {
        for (char *c = s + 1; *c != 0; c++) {
            if (isupper(*c)) {
                return 0;
            }
        }
    }
    return 1;
}

int main() {

    char s[107];
    gets(s); // 读入一行字符串
    if (all_capitalized(s) || no_captialized(s) || first_capitalized(s)) {
        printf("true"); // 符合三种情况之一，输出true，注意拼写
    } else {
        printf("false"); // 否则输出false
    }

    return 0;
}
```

C 冰雹猜想2022

难度	考点
2	数组

题目分析

题意较为明确，只需按照流程生成一个关于数字N的变化序列，使用数组存储起来，并倒序输出，最大值在生成变化序列过程中即可求解完成。（可以思考一下如何不通过排序，线性求出求前两个最大值，求前三个最大值，之前上机题有过类似的题目）。
但题目也指出，冰雹猜想的变化过程异常剧烈，带有极大的随机性，因此我们**无法提前预知数组元素数量上界**，只能凭借感觉大致猜测。因此数组解法存在着数组越界的隐患。
倒序输出是递归的经典实例，题解给出了递归解法。递归解法相较于数组解法节省了大量的内存空间，同时也避免了因估计错误而导致数组越界的问题。

示例代码-递归解法

```
#include<stdio.h>
long long c,max,count;
void rua(long long x){
    if(x==1){
        printf("1 ");
        return;
    }
    long long n=x;
    if(n%2==0){
        n=n/2;
    }else{
        n=3*n+1;
    }
    if(n>max) max=n;
    rua(n);
    printf("%lld ",x);
}
int main(){
    scanf("%lld",&c);
    rua(c);
    printf("\n");
    printf("%lld\n",max);
    return 0;
}
```

示例代码-数组解法

```
#include<stdio.h>
long long n,max,count;
long long arr[10000];
int main(){
    scanf("%lld",&n);
    while(n!=1){
        arr[count++]=n;
        if(n>max){
            max=n;
        }
        if(n%2==0){
            n=n/2;
        }else{
            n=3*n+1;
        }
    }
    arr[count]=1;
    for(int i=count;i>=0;i--){
        printf("%lld ",arr[i]);
    }
    printf("\n");
    printf("%lld\n",max);
    return 0;
}
```

D 301 Moved Permanently 2022

难度	考点
2	文件

题目解析

题目难度不大，按照题目要求重定向输出即可。

示例代码

```
#include <stdio.h>
int main()
{
    freopen("301.txt", "w", stdout);
    int n, op, i;
    long long a1, b1;
    double a2, b2;
    scanf("%d", &n);
    for (i = 1; i <= n; i++)
    {
        scanf("%d", &op);
        if (op == 1 || op == 2)
        {
            scanf("%lld%lld", &a1, &b1);
            printf("%lld\n", a1 + b1);
        }
        else
        {
            scanf("%lf%lf", &a2, &b2);
            printf("%.5lf\n", a2 + b2);
        }
    }
    return 0;
}
```

E 简单的种田

难度	考点
2	递归

题目分析

一道基础的递归题。

先给出一个结论：对于一块田地，我们每次种的面积越大，那么最终消耗的总体力越少。例如一块2×2的田，我们种一次2×2只需要花8的体力，而种4次1×1则需要花16的体力。

定义函数$work(x, y)$表示长为x宽为y的田地所需的体力值。

设$x > y$，很多同学一下能想到这样的做法：

$$work(x,y) = 4*y+work(x-y,y).$$

假设有一块长为1，宽为10^{16}的田，这样做肯定会超时的。

正确的做法是：

$$work(x,y) = 4*y*(x/y) + work(x\%y, y).$$

示例代码

```
#include<stdio.h>

long long x, y;

long long work(long long x, long long y){
    if(!x || !y)    return 0;
    if(x < y)   return 4*x*(y/x) + work(x, y%x);
    else    return 4*y*(x/y) + work(x%y, y);
}

int main(){
    scanf("%lld%lld", &x, &y);
    printf("%lld", work(x, y));
    return 0;
}
```

F 蒙达鲁克硫斯伯古比奇巴勒城的名册2.0

难度	考点
4	结构体、指针

题目分析

本题是考察结构体及其指针的应用。题目的意思很简单，当我们在进行结构体的交换时，是将结构体变量所对应内存上的所有内容进行交换，也就是说，在结构体内容比较大时，我们需要交换的内容也非常的多，效率也会更加的低。

那为什么利用指针可以更快的进行交换呢？我们可以这样理解，将结构体想象成一个沉重的大箱子，而指针是贴在箱子上的一个标签，当我们需要进行交换时，你是选择费力地挪动箱子，还是简单的交换标签呢？挪动箱子就是我们上面说的交换结构体本身，交换标签指的是改变结构体指针所指向的内存，就是说指针指向的地址变了，但是地址上的内容并不发生改变，这样的话，无论结构体多大，我们改变的数据量都只是指针变量所占的字节数（8字节）

我们用n个结构体指针分别指向n个结构体，当进行交换时，每次交换两个指针的值即可。

参考代码

```
#include <stdio.h>
#include <stdlib.h>
#include<time.h>
struct People
{
    int age, index;
    char name[1005];
} data[200007], *dataP[200007];//结构体和相应的指针数组
int main()
{
    int n,m,k, t,i;

    scanf("%d %d %d", &n,&m,&k);
    for (i = 1; i <= n; ++i)
    {
        scanf("%d %s", &data[i].age, data[i].name);
        dataP[i] = data + i;
        data[i].index = i;//初始化指针
    }

    int l,r;
    for(int i=0;i<m;i++)
    {
        scanf("%d%d",&l,&r);
        while(l<r)
        {
```

```
            struct People *temp;//每次进行交换时，交换指针
            temp=dataP[l];
            dataP[l]=dataP[r];
            dataP[r]=temp;
            l++;r--;
        }
    }

    for (i = 0; i < k; ++i)
    {
        scanf("%d",&t);
        printf("%d %s\n", dataP[t]->age, dataP[t]->name);
    }

}
```

G 灭火

难度	考点
4	排序

题目分析

将着火房屋进行快速排序之后，比较相邻两个着火房屋的间隔即可；注意首尾房屋的特殊情况；

如果按照 $O(n^2)$ 模拟的话，只能过 60% 的数据点。

示例代码

```
#include <stdio.h>
#include <stdlib.h>
int max(int a,int b)
{
    if(a<b)return b;
    return a;
}
int cmp(const void *p1,const void *p2)
{
    return *(int *)p1 - *(int *)p2;
}
int a[1000005];
int n,m;
int t=0;
int main(void)
{
    scanf("%d%d",&n,&m);
    for(int i=0;i<m;i++)
    {
        scanf("%d",&a[i]);
    }


    qsort(a,m,sizeof(a[0]),cmp);

    t=max(t,a[0]-1);
    t=max(t,n-a[m-1]);
    for(int i=0;i<m-1;i++)
    {
        t=max(t,(a[i+1]-a[i])/2);
    }
    printf("%d\n",t+1);

    return 0;
```

```
}
```

H 找内鬼2.0

难度	考点
5	位运算

题目分析

假设大家已经学会通过异或运算去找唯一不同数。如果不会的话请参看 E2 - E题 找内鬼 题解

不同本题的难点在于判断两个内鬼时的情况。既然输入数量已经给到大家，所以当输入数据个数为偶数时，要么无内鬼，要么存在两个内鬼！存在两个内鬼的时候，他们的值一定不同，因此二进制表示也一定不同，对两个不同的数进行异或运算时，不同的位结果为1，假设异或结果是：

$$0\,0\,0\,0\,0\cdots0\,1\,0\cdots0\,0$$
$$\uparrow$$

箭头所指处，两个内鬼表示的值在这一位一定不相同。不妨假设内鬼 a 这一位为1，内鬼 b 这一位为0。我们在此时直接重新遍历一遍这一位为 0 的所有数据，即得这一位为 0 的内鬼 b ，再将确定好的内鬼 b 的值和之前的异或值再进行一次异或，即有关系：

$$= b \bigoplus (a \bigoplus b)$$

$$= 0 \bigoplus a$$

$$= a$$

此时内鬼 a 和内鬼 b 的值就得到了。

参考代码

```
#include<stdio.h>
int array[6000005];
int main(){
    int i, n, temp = 0, count = 0, temp2 = 0, a, b;
    scanf("%d", &n);
    for(i = 0; i < n; i++){
        scanf("%d", &array[i]);
        temp = temp ^ array[i];
    }
    if(n % 2)
        printf("%d", temp);
    else{
        if(temp == 0)
            printf("False Alarm.");
        else{
            i = temp;
            while((i & 1) == 0){
                i >>= 1;
                count++;
```

```
            }
            for(i = 0; i < n; i++)
                if(((array[i] >> count) & 1) == 1)
                    temp2 = array[i] ^ temp2;
            a = temp2;
            b = temp ^ temp2;
            if(b < a)
                printf("%d %d", b, a);
            else
                printf("%d %d", a, b);
        }
    }
    return 0;
}
```

I 数据类型

难度	考点
5	字符串

题目解析

观察小数、整数、字符串之间的区别，可以很容易得出几种数据类型满足的条件：

类型	小数点数量	数字数量	字母数量
小数	1	>0	0
整数	0	>0	0
字符串	≥ 0	≥ 0	≥ 0

观察上表，发现字符串中包含了小数和整数的条件，因此字符串需要最后判断。所以我们需要遍历输入的字符串，并根据上表对应类型输出。

示例代码

```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
#include<math.h>
#include<ctype.h>

int main()
{
    char in[50],try_out[50];
    double try_double;
    int try_int;
    long long try_ll;
    while(gets(in)!=NULL)
    {
        int dot=0,ch=0,num=0;
        for(int i=0;i<strlen(in);i++)
        {
            if(in[i]=='.') dot++;
            if(isalpha(in[i])) ch++;
            if(isdigit(in[i])) num++;
        }
        //try double: only one dot and no character
        if(dot==1&&ch==0&&num>0)
        {
            printf("double\n");
```

```
            continue;
        }
        //try int and long long: no dot nor character
        if(dot==0&&ch==0&&num>0)
        {
            sscanf(in,"%d",&try_int);
            sprintf(try_out,"%d",try_int);
            if(strcmp(in,try_out)==0)
            {
                printf("int\n");
                continue;
            }
            else
            {
                sscanf(in,"%lld",&try_ll);
                sprintf(try_out,"%lld",try_ll);
                if(strcmp(in,try_out)==0)
                {
                    printf("long long\n");
                    continue;
                }
            }
        }
        //string: other
        printf("string\n");
    }
    return 0;
}
```

J rlx不教你写代码（二）

难度	考点
6	综合

题目解析

本题较为复杂，对初学者来说难度较高。本题主要考察的是词法分析。词法分析的目的是将代码中的语法元素提取出来，语法元素如同题目中介绍的那样，包含关键字、标识符和运算符等内容。C语言词法分析的基本流程如下图所示：

对于大部分同学，词法分析的流程仅作了解即可，本题的具体实现可参考示例代码。

示例代码中，`token`结构体的定义借助了`enum`（枚举）数据类型，用于记录`token`的类型。简单介绍定义的函数的作用：

`get_char`：从代码中读取一个字符；

`peek_char`：偷看代码中的下一个字符是什么；

`get_token`：读取一个token。在函数内部，通过看下一个非空白字符就可以判断下一个token的类型；

`get_preprocessor`：读取一个预处理器类型的token；

`get_number`：读取一个数字类型的token；

`get_identifier_or_keyword`：读取一个标识符或关键字类型的token；

`get_character`：读取一个字符类型的token；

`get_string`：读取一个字符串类型的token；

`get_operator`：读取一个运算符类型的token；

`reformat_code`：格式化代码。

示例代码

仅供参考

```
#include <stdio.h>
#include <string.h>
#include <ctype.h>

struct file {
    char data[100007];
    char *pos;
};
struct token {
    enum {
```

```
        preprocessor,
        integer, float_point,
        character, string,
        identifier, keyword,
        unary_operator, binary_operator,
        left_parenthese, right_parenthese, left_bracket, right_bracket, left_brace,
right_brace, semicolon, comma,
        START, END
    } type;
    char name[50];
};

struct file code;

void input_file(struct file *f) {
    f->pos = f->data;
    char line[1007];
    while (fgets(line, 1007, stdin) != NULL) {
        strcat(f->data, line);
    }
}
char get_char(struct file *f) {
    char *p = f->pos;
    if (*p != 0) {
        f->pos = p + 1;
    }
    return *p;
}
char peek_char(struct file *f) {
    return *f->pos;
}

void get_preprocessor(struct file *f, struct token *save) {
    save->type = preprocessor;
    char *p = save->name;
    while (peek_char(f) != 'e') {
        *p++ = get_char(f);
    }
    *p++ = get_char(f);
    while (peek_char(f) == ' ') {
        get_char(f);
    }
    *p++ = ' ';
    *p++ = get_char(f);
    while (peek_char(f) != '>') {
        *p++ = get_char(f);
    }
    *p++ = get_char(f);
    *p = 0;
```

```
}

void get_number(struct file *f, struct token *save) {
    save->type = integer;
    char *p = save->name;
    while (isdigit(peek_char(f)) || peek_char(f) == '.') {
        if (peek_char(f) == '.') {
            save->type = float_point;
        }
        *p++ = get_char(f);
    }
    *p = 0;
}

void get_identifier_or_keyword(struct file *f, struct token *save) {
    char *keywords[] = {"int", "long", "short", "float", "double", "char", "unsigned",
"signed", "const", "void", "if", "else", "do", "while", "for", "continue", "break",
"sizeof", "return", NULL};
    save->type = identifier;
    char *p = save->name;
    while (isalpha(peek_char(f)) || isdigit(peek_char(f)) || peek_char(f) == '_') {
        *p++ = get_char(f);
    }
    *p = 0;
    for (int i = 0; keywords[i] != NULL; i++) {
        if (strcmp(save->name, keywords[i]) == 0) {
            save->type = keyword;
        }
    }
}

void get_character(struct file *f, struct token *save) {
    save->type = character;
    char *p = save->name;
    *p++ = get_char(f);
    while (peek_char(f) != '\'') {
        if (peek_char(f) == '\\') {
            *p++ = get_char(f);
            if (peek_char(f) == '\'') {
                *p++ = get_char(f);
            }
        } else{
            *p++ = get_char(f);
        }
    }
    *p++ = get_char(f);
    *p = 0;
}
```

```
void get_string(struct file *f, struct token *save) {
    save->type = string;
    char *p = save->name;
    *p++ = get_char(f);
    while (peek_char(f) != '"') {
        if (peek_char(f) == '\\') {
            *p++ = get_char(f);
            if (peek_char(f) == '"') {
                *p++ = get_char(f);
            }
        } else{
            *p++ = get_char(f);
        }
    }
    *p++ = get_char(f);
    *p = 0;
}

void get_operator(struct file *f, struct token *save) {
    char *p = save->name;
    if (peek_char(f) == '{') {
        save->type = left_brace;
        *p++ = get_char(f);
    } else if (peek_char(f) == '}') {
        save->type = right_brace;
        *p++ = get_char(f);
    } else if (peek_char(f) == ';') {
        save->type = semicolon;
        *p++ = get_char(f);
    } else if (peek_char(f) == '(') {
        save->type = left_parenthese;
        *p++ = get_char(f);
    } else if (peek_char(f) == ')') {
        save->type = right_parenthese;
        *p++ = get_char(f);
    } else if (peek_char(f) == '[') {
        save->type = left_bracket;
        *p++ = get_char(f);
    } else if (peek_char(f) == ']') {
        save->type = right_bracket;
        *p++ = get_char(f);
    } else if (peek_char(f) == ',') {
        save->type = comma;
        *p++ = get_char(f);
    } else {
        save->type = binary_operator;
        if (strchr("+-*/%^&|!<>=", peek_char(f)) != NULL) {
            char first = get_char(f), second = 0;
            *p++ = first;
```

```
            if (peek_char(f) == '=') {
                second = get_char(f);
                *p++ = second;
            } else if (strchr("+-&|<>=", first) != NULL && peek_char(f) == first) {
                second = get_char(f);
                *p++ = second;
            }
            if ((first == '<' && second == '<') || (first == '>' && second == '>')) {
                if (peek_char(f) == '=') {
                    *p++ = get_char(f);
                }
            }
        } else {
            *p++ = get_char(f);
        }
    }
    *p = 0;
    char *unary_operators[] = {"++", "--", "~", "!", NULL};
    for (int i = 0; unary_operators[i] != NULL; i++) {
        if (strcmp(unary_operators[i], save->name) == 0) {
            save->type = unary_operator;
            break;
        }
    }
}

void get_token(struct file *f, struct token *save) {
    while (peek_char(f) == '\n' || peek_char(f) == '\t' || peek_char(f) == ' ' ||
peek_char(f) == '\r') {
        get_char(f);
    }
    if (peek_char(f) == '#') {
        get_preprocessor(f, save);
    } else if (isdigit(peek_char(f))) {
        get_number(f, save);
    } else if (isalpha(peek_char(f)) || peek_char(f) == '_') {
        get_identifier_or_keyword(f, save);
    } else if (peek_char(f) == '\'') {
        get_character(f, save);
    } else if (peek_char(f) == '"') {
        get_string(f, save);
    } else if (peek_char(f) == 0) {
        save->type = END;
        save->name[0] = 0;
    } else {
        get_operator(f, save);
    }
}
```

```
int change_line(int level, struct token next_token) {
    printf("\n");
    level -= next_token.type == right_brace;
    for (int i = 0; i < level; i++) {
        printf("    ");
    }
    return 0;
}

void reformat_code(struct file *f) {
    struct token last_token = {}, cur_token = {START}, next_token = {};
    int level = 0, in_for = 0, new_line = 1;

    while (cur_token.type != END) {

        last_token = cur_token;
        if (last_token.type == START) {
            get_token(f, &cur_token);
        } else {
            cur_token = next_token;
        }
        get_token(f, &next_token);

        if (cur_token.type == right_parenthese && last_token.type == keyword) {
            cur_token.type = unary_operator;
        }
        if (cur_token.type == binary_operator) {
            if (last_token.type != integer && last_token.type != float_point &&
last_token.type != character && last_token.type != identifier && last_token.type !=
right_parenthese && last_token.type != right_bracket) {
                if (!(last_token.type == unary_operator && (strcmp(last_token.name,
"++") == 0 || strcmp(last_token.name, "--") == 0))) {
                    cur_token.type = unary_operator;
                }
            }
        }
        if (cur_token.type == left_brace) {
            level++;
        }
        if (cur_token.type == right_brace) {
            level--;
        }
        if (cur_token.type == keyword && strcmp(cur_token.name, "for") == 0) {
            in_for = 1;
        }
        if (cur_token.type == left_brace && in_for == 1) {
            in_for = 0;
        }
```

```
        if (cur_token.type == preprocessor) {
            new_line = printf("%s", cur_token.name);
            new_line = change_line(level, next_token);
        } else if (cur_token.type == binary_operator) {
            new_line = printf(" %s ", cur_token.name);
        } else if (cur_token.type == comma) {
            new_line = printf("%s ", cur_token.name);
        } else if (cur_token.type == keyword) {
            if (strcmp(cur_token.name, "return") == 0) {
                if (next_token.type != semicolon) {
                    new_line = printf("%s ", cur_token.name);
                } else {
                    new_line = printf("%s", cur_token.name);
                }
            } else if (strcmp(cur_token.name, "continue") == 0 ||
strcmp(cur_token.name, "break") == 0 || strcmp(cur_token.name, "sizeof") == 0) {
                new_line = printf("%s", cur_token.name);
            } else if (next_token.type == right_parenthese) {
                new_line = printf("%s", cur_token.name);
            } else {
                new_line = printf("%s ", cur_token.name);
            }
        } else if (cur_token.type == semicolon) {
            new_line = printf("%s", cur_token.name);
            if (in_for == 0) {
                new_line = change_line(level, next_token);
            } else {
                new_line = printf(" ");
            }
        } else if (cur_token.type == left_brace) {
            if (new_line) {
                new_line = change_line(level, (struct token){right_brace});
            }
            new_line = printf("%s", cur_token.name);
            new_line = change_line(level, next_token);
        } else if (cur_token.type == right_brace) {
            if (new_line) {
                new_line = change_line(level, (struct token){left_brace});
            }
            new_line = printf("%s", cur_token.name);
            if (next_token.type != semicolon && next_token.type != comma) {
                new_line = change_line(level, next_token);
            }
        } else {
            new_line = printf("%s", cur_token.name);
        }
    }

}
```

```c
int main(int argc, const char * argv[]) {

    input_file(&code);
    reformat_code(&code);

    return 0;
}
```

K 六宫数局

难度	考点
8	综合

题目分析

首先，每个节点都有多路选择的路径问题，考虑DFS。

接下来，难点在于如何存储这个数字阵列，提供一种“斜坐标系”的方式，将蜂巢如图所示存入二维数组中。

这样，原先的6个方向在这里变为：

方向	变化
`LU`	`y = y + 1`
`U`	`y = y + 1` `x = x + 1`
`RU`	`x = x + 1`
`RD`	`y = y - 1`
`D`	`y = y - 1` `x = x - 1`
`LD`	`x = x - 1`

接下来，用正常的DFS思路，将每到一个点时搜索可能的六个方向，从而找到终点。

示例代码

```
#include <stdio.h>

typedef struct {
    char dir[5];
    int length;
} path;

int hive[20][20];
int via[20][20];
path route[1000];

int arrived = 0;
int routeLength = 0;
```

```c
void fillHive(int n);
int getPower(int n, int a);
int inHive(int n, int x, int y);
void goOn(int n, int x, int y);
void printRoute();

int main()
{
    int n = 0;
    scanf("%d", &n);

    fillHive(n);

    goOn(n, 0, 0);

    return 0;

}

void fillHive(int n)
{
    int i = 0, j = 0;

    for (i = 0; i < n; i++) {
        for (j = 0; j < i + n; j++) {
            scanf("%d", &hive[j][i]);
        }
    }

    for (i = n; i < 2 * n - 1; i++) {
        for (j = i - n + 1; j < 2 * n - 1; j++) {
            scanf("%d", &hive[j][i]);
        }
    }

    return;
}

int getPower(int n, int a)
{
    int k = 0;
    while (n != 1) {
        if (n % a != 0) {
            break;
        }
        n /= a;
        k++;
    }
```

```
    return k;
}

int inHive(int n, int x, int y)
{
    if (x < n) {
        if (y < x + n && y >= 0) {
            return 1;
        } else {
            return 0;
        }
    } else if (x < 2 * n - 1) {
        if (y > x - n && y < 2 * n - 1) {
            return 1;
        } else {
            return 0;
        }
    } else {
        return 0;
    }
}

void goOn(int n, int x, int y)
{
    if (!inHive(n, x, y)) {
        return;
    }

    if (arrived) {
        return;
    }

    if (via[x][y] == 1) {
        return;
    }

    // printf("%d %d\n", x, y);

    if (x == 2 * n - 2 && y == 2 * n - 2) {
        arrived = 1;
        printRoute();
        return;
    }

    via[x][y] = 1;
    int num = hive[x][y];

    // printf("%d\n", num);
```

```
    int p_2 = getPower(num, 2);
    if (p_2 != 0) {
        route[routeLength] = (path){"LU", p_2};
        routeLength++;
        goOn(n, x, y + p_2);
        routeLength--;
    }

    int p_3 = getPower(num, 3);
    if (p_3 != 0) {
        route[routeLength] = (path){"U", p_3};
        routeLength++;
        goOn(n, x + p_3, y + p_3);
        routeLength--;
    }

    int p_5 = getPower(num, 5);
    if (p_5 != 0) {
        route[routeLength] = (path){"RU", p_5};
        routeLength++;
        goOn(n, x + p_5, y);
        routeLength--;
    }

    int p_7 = getPower(num, 7);
    if (p_7 != 0) {
        route[routeLength] = (path){"RD", p_7};
        routeLength++;
        goOn(n, x, y - p_7);
        routeLength--;
    }

    int p_11 = getPower(num, 11);
    if (p_11 != 0) {
        route[routeLength] = (path){"D", p_11};
        routeLength++;
        goOn(n, x - p_11, y - p_11);
        routeLength--;
    }

    int p_13 = getPower(num, 13);
    if (p_13 != 0) {
        route[routeLength] = (path){"LD", p_13};
        routeLength++;
        goOn(n, x - p_13, y);
        routeLength--;
    }

    via[x][y] = 0;
```

```
    return;
}

void printRoute()
{
    int i = 0;
    for (i = 0; i < routeLength; i++) {
        printf("%s %d\n", route[i].dir, route[i].length);
    }

    return;
}
```