问题求解 I - OJ期末考试试题

考试说明：

1. 考试时长为 3 小时，请注意把控时间。
2. 每道题都有相当多的部分分可以在未想出标准解法的情况下得到，请注意不要在一道题上吊死。
3. 考题未必按照难度顺序排列，每个人对难度的感知也不同，不必按照顺序攻坚。
4. 对于每道题，我们只会关注你的最后一次提交，并在不是满分的情况下阅读你书写的注释酌情给分(请不要在注释上内卷，我们不会关注你的注释的美观度/详细度，只关注思想)。

1. BrainF**k Syntax Checker

题面描述

关于 BF 程序的基础知识请参照第二题。

给定一个 BF 程序，请你判断它是否是合法的。合法请输出 `Yes`，否则输出 `No`。

第二题中有关问题简化的假设对本题并不成立。具体地，在本题中，

- BF 的机器模型拥有向两端无限延伸的内存单元格，你不需要担心 `<` 和 `>` 超出边界的问题。
- 你可以认为输入流中有无限的字符可以读取，即程序不会在执行到 `,` 的时候因没有输入而终止。
- 你不需要检测死循环——死循环并不意味着不合法，但如果导致死循环的语句后有明显不符合语法的语句，你仍应当检测出并输出 No。例如：
  - `[+]hahaha` 会陷入死循环，但 `[+]` 后出现了非法字符，所以它仍是不合法的。
  - `[>][` 会陷入死循环，但最后单独出现了左中括号，所以它仍是不合法的。

输入格式

每个测试点的输入格式如下：

```
S
```

其中 S 是待检测的 BF 程序。我们保证 S 中不会出现对读入不友好的不可见字符。
我们强烈建议你使用如下的模板进行该题的作答：

```cpp
#include <bits/stdc++.h>
using namespace std;

char program[1000];

bool bf_checker()
{
    // your checker
}

int main ()
{
    scanf("%s", program);
    if (bf_checker())
        puts("Yes");
    else
        puts("No");
    return 0;
}
```

输出格式

对于每个测试点，输出一行 Yes 或 No。

输入样例

```
样例1:
+++[+-],.,....

样例2:
ProblemSolving
```

输出样例

```
样例1:
Yes

样例2:
No
```

数据范围

对于 100% 的分数：BF 程序长度不超过 1000。

提示

如果你仔细阅读 BF 程序 8 种符号的定义，你就会发现只有两种符号可能导致语法层面的错误。除了明显不合法的字符，你只需要重点关注这两种字符。

2. BrainF**k Interpreter

题面描述

BrainFuck (简称 BF) 是一种极小化的计算机语言，于 1993 年由 Urban Müller 发明。它虽然极其简单，但却是图灵完备的——换言之，理论上它与大家常使用的 C, C++, Python 等通用高级语言有着相同的表达能力！在该题中，我们将带你了解 BF 的语法规则并要求你写出一个 BF 的解释器。

BF 基于一个简单的机器模型：除了 BF 程序本身，这个机器还包括：内存——一个被**初始化为0**的数组 (理论上无限长)，一个可以指向数组中某个元素的指针，以及输入、输出流。

BF 程序由以下 8 种符号构成：

- `+`：将指针指向的单元格中的数字+1。
- `-`：将指针指向的单元格中的数字+1。
- `>`：将指针向右挪一个。
- `<`：将指针向左挪一个。
- `,`：读入一个数，将该数记录在指针指向的单元格。
- `.`：输入当前单元格指向的数。
- `[`：循环开始标识符，它和与之匹配的 `]` 构成了一个循环，`[]` 中的内容是循环体，进入循环的充要条件是当前指针指向的单元格中的数不等于 0。如果不满足该条件，程序会跳转到匹配的 `]` 的下一个字符继续执行。
- `]`：循环结束标识符，遇到它程序将跳转到与之匹配的 `[` 标志符。

你可能对以上定义感到有些困惑 (尤其是最后两个)，我们下面将通过若干个例子来阐述 BF 程序的语义 (以下的解释中，我们假设“数组”的下标从 0 开始，指针最初指向下标 0)：

- `>>+`，无输入。
  - 解释：指针向右移动到 [2]，然后给该单元格+1。
  - 最终内存：`(0, 0, 1, ...)`。
- `,[-]`，输入 5。
  - 解释：读入一个数存入 M[0]，然后在 M[0] 不为 0 时执行循环体，给 M[0] 减一，执行 5 次后 M[0]=0，于是退出。
  - 最终内存：`(0, ...)`。

- `+++++[>>+<<-]`，无输入。
  - 解释：最初给 M[0] 加五次1，然后进入循环。循环的语义为：先把指针向右移动到 M[2]，+1，然后移回 M[0]，-1。这个过程可以很形象地理解为“搬砖”：M[0] 有 5 块砖，每轮循环搬一块到 M[2]，直到搬完退出循环。
  - 最终内存：`(0, 0, 5, ...)`。

2.1 是该问题的简单版本。你需要阅读以下 BF 程序片段并按要求分析输出内容或内存状态 (若有多个输出，用空格隔开)：

- `+++++`，请输出 M[0] 的值。
- `,.,.`，输入 `4 8`，请给出输出结果。
- `+++++>+++[<+>-]`，请输出 M[0] 的值。
- `+++[>+++++<-]`，请输出 M[1] 的值。
- `>,[>,]<[<]>[.>]`，输入 `1 5 3 2 4 0`，请给出输出结果。

2.2 是该问题的困难版本。你需要用 C/C++ 语言编写一个 BrainF**k 的解释器。具体来说，给定一段程序和该程序中所有的输入值，你需要打印：

- 程序执行阶段所有 `.` 操作的输出值。
- 程序执行结束后的内存状态。

此外，为了简化问题，我们做如下额外约定：

- 给定的 BF 程序一定合法 (你在第一题中已经做过合法性检查了) 且不会发生死循环。
- 理论上 BF 的内存是可以向两端无限延伸的，但为了简单，我们假设内存的长度只有 10 (即下标 0...9)，刚开始指针指向下标 0。我们保证给定的 BF 程序指针移动过程中不会超出该范围。
- 理论上 BF 存储的是字符的 ASCII 码，但为了简单，本题中我们所有的读入，输出和内存中存储的值都是非负整数。

输入格式

每个测试点输入格式如下:

```
type
S
N
a1 a2 ... an
```

其中

- `type=1, 2, 3, 4, 5` 代表该测试点用于测试上面给定的对应号码的问题。`type=0` 表示这是一个需要运行你编写的解释器获得答案的测试点。剩下的输入内容在所有测试点中均存在，但仅在

`type=0` 的测试点中有意义。

- `S` 是 BF 程序。
- `N` 是该 BF 程序对应的输入个数。
- 接下来 N 个数代表 BF 程序的输入。

我们强烈建议你使用如下的模板进行该题的作答:

```
#include <bits/stdc++.h>
using namespace std;

int type;
char program[1000];
int input[1000], input_count;

void concrete_examples()
{
    switch (type)
    {
        case 1: printf("你的第1题解答\n"); break;
        case 2: printf("你的第2题解答\n"); break;
        case 3: printf("你的第3题解答\n"); break;
        case 4: printf("你的第4题解答\n"); break;
        case 5: printf("你的第5题解答\n"); break;
    }
}

void general_solve()
{
    // 你的 BrainF**k 解释器实现
}

int main ()
{
    scanf("%d", &type);
    scanf("%s", program);
    scanf("%d", &input_count);
    for (int i = 0; i < input_count; i++)
        scanf("%d", input + i);
    if (type > 0)
        concrete_examples();
    else
        general_solve();
    return 0;
}
```

根据输入格式，你应该很容易理解该模板的含义，因此如果你有自己的编程喜好 (如倾向于使用 C++ 风格的 string 类型而不是 C 风格的 char 数组，或者倾向于使用 1-base 数组)，你完全可以自行修改该模板。

输出格式

对于 `type=1, 2, 3, 4, 5` 的测试点，你的输出应当形如

```
ans1 ans2 ans3...
```

每个 ansi 代表一个需要输出的数。

对于 `type=0` 的测试点，你的输出应当形如

```
N
a1 a2 ... an
m0 m1 ... m9
```

其中

- `N` 表示 BF 程序执行过程中的输出个数。
- `ai` 表示 BF 程序执行过程中的第 i 个输出，如果 N 为 0 则该行省略。
- `mi` 表示 BF 程序执行结束后下标为 i 的单元格的值。注意我们保证 BF 程序的执行不会使用超过 10 个单元格，你只需要打印前 10 个单元格的值即可。

输入样例

```
0
+++.>,[>,]
6
7 8 9 2 3 0
```

输出样例

```
1
3
3 7 8 9 2 3 0 0 0 0
```

数据范围

部分测试点已经在题目中给出。每答对一个问题可获得本题 10% 的分数。

剩下的 50% 的分数中：

- 对于 20% 的分数：程序中不包含 `[]`。

- 对于 40% 的分数：程序不包含嵌套的 `[]`。
- 对于 100% 的分数：程序包含所有符合 BF 语法的符号；BF 程序的长度小于 1000，输入的非负整数个数小于 1000 且保证在 int 范围内。

提示

1. 虽然上面的解释看起来非常复杂，用 BrainF**k 写出满足一定功能要求的程序也十分挑战智商，但实现一个解释器，让机器来执行 BF 其实非常容易。我们在这里给出一台真正的计算机执行机器语言指令的过程，你或许可以从中获得启发：
   - 取指：取出下一条执行的指令。【在 BF 中一条“指令”对应了什么？】
   - 译码：根据指令的格式确定该指令的类型。
   - 执行：根据指令类型，执行该指令定义的动作。
   - 跳转：根据指令类型和执行过程，确定下一条应当执行的指令的位置。【在 BF 中什么“指令”的“下一条指令”的位置是不平凡的？】

   真正的计算机 (或者说, CPU) 就是在永不停止地执行上面的四步骤循环。
2. 如果你有自信可以实现正确的解释器，你可以用你实现的解释器运行给定的5道题目直接获得所需答案，不需要手算浪费时间。

3. DXY's Graph Problem

题面描述

DXY和`生鱼片`在NFT市场上获得了一个十分稀有的 n 个点，m 条边的无向连通图，对于如此珍贵的东西，DXY和`生鱼片`提出希望将这张图分成两个点集，由二人分别保管，同时，为了防止对方将属于自己的那一部分损毁，他们提出了一个要求，希望可以从自己的那一部分完整地恢复出原图。
DXY认为这很容易做到，只要他知道每个边对应的一个顶点，他就可以对这一条边进行修复，即，如果记某人拥有的顶点构成的集合为V_d，该集合满足对于任意原图中的边u,v，都有$u \in V_d$ 或者 $v \in V_d$，则DXY可以从V_d中完整地恢复出原图，
但笨笨的DXY只会修复，并不知道如何将这张图进行划分，于是邪恶的DXY威胁你们，要你们写一个程序帮忙给出划分方案，并将其作为OJ考试题目，特别地，DXY对图进行从1到n的编号，如果可以进行划分，DXY希望能获得编号为1的点，如果不能进行这样的划分，也请你告知DXY。

输入格式

每组测试数据的格式如下：

```
n m
u1 v1
u2 v2
...
um vm
```

测试数据的第一行输入两个数字 $n, m(2 \leq n \leq m \leq 100000)$，分别表示图的顶点数和边数。
接下来 m 行，每行两个数字$u_i, v_i(1 \leq u_i, v_i \leq n)$，表示$u_i, v_i$之间存在一条边。
数据保证图连通，且不含重边和自环。

输出格式

如果可以进行划分，则输出"Yes"，并继续输出三行，第一行输出D, S，表示DXY和生鱼片在划分中能获得的顶点个数，第二行**从小到大**输出D个数字，表示DXY获得的顶点编号（1号顶点应该按照DXY的意愿被分配给DXY，否则DXY不让你通过！），第三行从小到大输出S个数字，表示生鱼片获得的顶点编号。
如果不能进行划分，则输出"No"。

输入样例

```
样例1
3 3
1 2
2 3
3 1

样例2
4 4
1 2
2 3
3 4
4 1
```

输出样例

```
样例1
No

样例2
Yes
2 2
1 3
2 4
```

数据范围

对于60%的数据，满足$2 \leq n \leq 20, n-1 \leq m \leq 50$
对于80%的数据，满足$2 \leq n \leq 100, n-1 \leq m \leq \frac{(n-1)*n}{2}$
对于100%的数据，满足$2 \leq n \leq m \leq 100000$

对于一个合理的分配方案，要将每个节点分配过DXY或者生鱼片。

提示

1. 本题并没有说明“如果有多种方案，输出任意一个即可”，这说明在 Yes 的情况下划分方案是唯一的。你不妨从思考成功划分的充要条件入手解决这个问题。(~~虽然这是一种“小镇做题家”的解题思路，但我们还是写在这里供大家参考。~~)
2. 大家可能会在本题的图的存储上犯难。一个简单的想法是对于每个节点存储它的所有邻居节点，但不同的节点邻居数量不同，难以定义一个统一的二维数组。这里我们给大家介绍用 vector 建立图的邻接表的做法：

```
int N, M; // N 是顶点数
vector<int> v[N + 10];
...
for (int i = 1; i <= M; i++)
{
    scanf("%d%d", &x, &y);
    v[x].push_back(y);
    v[y].push_back(x);
}
```

访问节点 x 的所有邻居节点：

```
for (int neighbor : v[x])
    printf("%d\n", neighbor);
```