图的探索

时间限制： 60 秒

空间限制： 750 MB

相关文件： 题目目录

题目背景

在社交网络中，你和任何一个陌生人之间所间隔的人不会超过六个，也就是说，最多通过六个人你就能够认识任何一个陌生人。

这就是著名的六度分隔现象，我们可以将社交网络模拟为**图** 。

题目描述

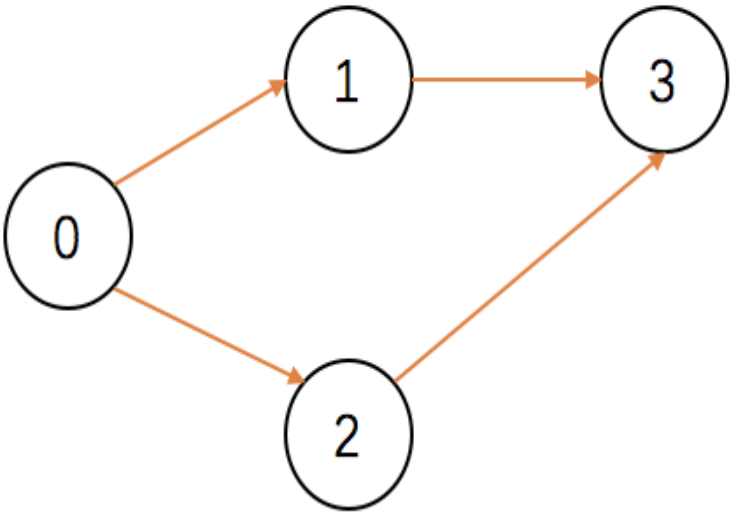
有向图 G 是由点与有向边构成的一种数据结构，记为 $G=(V, E)$ 。其中： V 为所有结点的集合， E 为所有有向边的集合。

点的总数为 $|V|$ （点的 id 范围为 0 到 $|V|-1$），边的总数为 $|E|$ 。

有向边 $e=(u, v)$ 为一条由点 u 出发到达点 v 的边， u 为源结点， v 为目标结点，表明 u 可以通过边 e 到达 v ，我们规定每条边的距离为 1 ，即 u 与 v 的距离为 1 。

对应到现实的社交网络中，结点对应到社交网络中的成员，有向边 $e=(u, v)$ 对应成员之间的关系： u 认识 v 。

下图是用有向图描述社交网络认识关系的例子：



其中：

V={0,1,2,3}

E={(0,1),(0,2),(1,3),(2,3)}

在社交网络中，一个人能通过多层关系找到最远的人和这个人通过任何方式都无法找到的人数能很好地刻画这个人的社交关系。在这个题目中，我们需要对指定的几个人求出这些数据。

具体地，我们会给出若干个社交网络成员对应的结点 s，你需要从 s 出发沿着有向边探索，找到能探索到的所有结点中，距离最远的结点 d 。你需要输出点 d 的 id，点 d 与出发结点 s 的距离。当存在多个符合标准的结点时，输出 id 最大的。此外，你还要统计从点 s 出发进行探索之后，探索不到的结点的数量，并输出。

在上图的例子中，假设我们关注的结点 s 为点 0。从点 0 出发，最远能探索到点为 3，距离为 2。该图中所有点均能从 0 出发被探索到。因此本例输出的结果为：`3 2 0`。

C/C++解题框架

你需要在 `solve.c/cpp` 中实现预处理的 `my_init()` 函数与探索的 `my_solve()` 函数。

程序开始运行时，会先调用你的预处理函数，此时你可以读写文件、修改变量或进行其他操作，但你并不能获知需要探索的结点 s；之后会调用 5 次 `my_solve()` 函数，每次调用会给你一个结点 s，你需要在函数中调用一次且仅一次 `MAIN_output()` 函数来报告此次探索的答案。

具体的，在所给的框架中，包含以下一些文件、文件夹：

- `Makefile`：用于编译，生成可执行文件 `my_run` 。
- `main.c/cpp`：程序的入口，其中：
    - 会调用预处理的 `my_init()` 函数与进行探索的 `my_solve()` 函数。
    - `MAIN_output()` 函数用于输出每次探索的结果：最远的结点、距离、未探索到的结点总数。
    - 实际测评使用的 `main.c/cpp` 会有所不同。
- `solve.h`：声明了 `my_init()` 、 `my_solve()` 、 `MAIN_output()` 。
- `solve.c/cpp`：你所需要填写的代码文件，其中：
    - 你需要实现 `my_init()` 、 `my_solve()` 函数。
    - 你可以在该文件中任意添加你要用到的变量、函数和其他代码。
- `run.sh`：运行程序的脚本。
- `case0`：文件夹。参赛者用于测试自己程序的一个较小的测试样例。其中包含图的信息文件、图数据的文本文件（**实际测评时不提供文本格式的文件**）、图数据的二进制文件、参考答案。

程序会自动统计你的预处理的用时、之后的 5 次探索的总用时。你各部分所用的时间会影响你的得分，将在下文的“评分方法”中介绍。

在答题中，有以下**注意事项**：

- 你只能修改并提交 `solve.c/cpp` 文件，修改其他文件是没有意义的。
- 你只能使用 `MAIN_output()` 函数告诉框架探索结果，提交的代码中不能用其他任何方式输出任何额外信息。
- 你的 `my_solve()` 函数每次被调用时，都必须调用恰好一次 `MAIN_output()`，不能多次调用，也不能不调用。

`make` 所使用的编译命令为 `gcc -O2 main.c solve.c -o my_run / g++ -O2 main.cpp solve.cpp -o my_run`

`run.sh` 中的运行命令为 `./my_run info.txt graph_binary_little_endian graph_binary_big_endian`

三个参数都是文件路径，具体意义见“输入格式”。

Java 解题框架

你需要在 `MainSolve.java` 中实现预处理的 `myInit()` 函数与探索的 `mySolve()`

函数。

程序开始运行时，会先调用你的预处理函数，此时你可以读写文件、修改变量或进行其他操作，但你并不能获知需要探索的结点 s；之后会调用 5 次 `mySolve()` 函数，每次调用会给你一个结点 s，你需要在函数中调用一次且仅一次 `mainOutput()` 函数来报告此次探索的答案。

具体的，在所给的框架中，包含以下一些文件、文件夹：

- `Makefile`：用于编译，生成 `.class` 文件。
- `MyMain.java`：程序的入口，其中：
    - 会调用预处理的 `myInit()` 函数与进行探索的 `mySolve()` 函数。
    - 实际测评使用的 `MyMain.java` 会有所不同。
- `MainOutput.java`：实现了 `mainOutput()` 函数。
    - `mainOutput()` 函数用于输出每次探索的结果：最远的结点、距离、未探索到的结点总数。
- `MainSolve.java`：你所需要填写的代码文件，其中：
    - 你需要实现 `myInit()` 、 `mySolve()` 函数。
    - 你可以在该文件中任意添加你要用到的变量、函数和其他代码。
- `run.sh`：运行程序的脚本。
- `case0`：文件夹。参赛者用于测试自己程序的一个较小的测试样例。其中包含图的信息文件、图数据的文本文件（**实际测评时不提供文本格式的文件**）、图数据的二进制文件、参考答案。

程序会自动统计你的预处理的用时、之后的 5 次探索的总用时。你各部分所用的时间会影响你的得分，将在下文的“评分方法”中介绍。

在答题中，有以下**注意事项**：

- 你只能修改并提交 `MainSolve.java` 文件，修改其他文件是没有意义的。
- 你只能使用 `mainOutput()` 函数告诉框架探索结果，提交的代码中不能用其他任何方式输出任何额外信息。
- 你的 `mySolve()` 函数每次被调用时，都必须调用恰好一次 `mainOutput()`，不能多次调用，也不能不调用。

`make` 所使用的编译命令为 `javac MyMain.java MainSolve.java MainOutput.java`

`run.sh` 中的运行命令为 `java MyMain info.txt graph_binary_little_endian graph_binary_big_endian`

三个参数都是文件路径，具体意义见“输入格式”。

输入格式

你编译生成的程序 `my_run` 可以从如下文件中读入：

- `info.txt` 为测试所用到的图的信息，两行，分别为：点数 $|V|$ 、边数 $|E|$ 。
- `graph_binary_little(big)_endian` 为测试所用到的图的数据，以小端（大端）模式二进制形式存放所有的边，每条边 (u,v) 中的每个点均使用一个 `int` 来表示。该文件中的边，均已经升序排好，先按照边的源结点排序，对每一个点的所有出边，也已按照目标结点升序排好。

如上面的图中，一共有 4 条边 {(0,1),(0,2),(1,3),(2,3)}

因此在 `graph_binary_big_endian` 会是：

```
0x00000000 0x00000001 0x00000000 0x00000002
0x00000001 0x00000003 0x00000002 0x00000003
```

在 `graph_binary_little_endian` 会是：

```
0x00000000 0x01000000 0x00000000 0x02000000
0x01000000 0x03000000 0x02000000 0x03000000
```

你实现的函数中，可以根据需要读取这两个文件。

一共有 5 个测试点，每个测试样例进行 5 次探索。保证 $|V|<5\times106$，$|E|<3\times108$。

样例

在 `case0` 文件夹中含有以下文件：

- `info.txt` 图的信息文件，两行，分别为：点数 $|V|$ 、边数 $|E|$ 。
- `graph.txt` 图数据的文本文件(**实际测评时不提供文本格式的文件**)，已升序排好。
- `graph_binary_little_endian` 图数据的二进制文件，将 `graph.txt` 中的每条边，以小端模式二进制形式存放。
- `graph_binary_big_endian` 图数据的二进制文件，将 `graph.txt` 中的每条边，以大端模式二进制形式存放。
- `answer.txt` 参考答案。

  `case0` 所使用的图数据为使用 `R-MAT` 生成的符合 `power-law` 的图。

调试方法

如果你想测试自己的图：

1. 按照要求自己生成 `info.txt`、`graph.txt`、`graph_binary_little_endian`、`graph_binary_big_endian` 等文件；
2. 相应修改 `run.sh` 运行命令中的文件路径。

如果你想测试自己的点 s ：

- C/C++：修改 `main.c/cpp` 中 传给 `my_solve()` 函数的第 3 个参数，重新编译生成新的可执行文件。
- Java：修改 `MyMain.java` 中 传给 `mySolve()` 函数的第 2 个参数，重新编译生成新的 `class` 文件。

子任务

测试点	$\|V\|$	$\|E\|$	基准时间(s)	来源或生成方式
1	9,996	99,933	R-MAT 生成图	
2	4,847,571	68,993,773	8	LiveJournal social network
3	3,072,626	117,185,083		Orkut social network(部分)
4		234,370,166	12	Orkut social network(全部)
5	3,999,983	266,903,057		R-MAT 生成图

其中 R-MAT 生成图均与样例为同一代码生成，边的分布服从 power-law。

评分方法

100=25（正确性）+25（基准性能）+50（全场性能评比）

其中 25 分为正确性得分：每个测试点满分为 5 分，每个测试点一共 5 次查询，一次 1 分，即：25=5×5×1。你的每个测试点需要在 **60*s*** 的时间内完成，包括框架用到的时间（**1*s* 内**）、预处理用的时间、5 次查询用的时间。

对于 case2、3、4、5 ，若该测试点的 5 次查询全部正确，就会进行基准性能评估和全场性能评比。

每个测试点会统计 5 次查询的**总用时 *t*** ，注意：**这里**不算入预处理所用时间。

25 分为基准性能得分：每个测试点满分为 6.25 分，当这个测试点的用时 t 小于该测试点

的基准时间时，一次性得到满分 6.25 分。

50 分为性能评比得分：每个测试点满分为 12.5 分，设 $t\text{min}$ 为所有选手本测试点的 t 的最小值，该测试点的性能得分为：

$$12.5\times\frac{t\text{min}}{t}$$

提示

- 本题中，一些测试点的图数据会非常大，请注意内存的使用。
- 社交网络中，两个人大多时候是相互认识或相互不认识，但也存在 u 认识 v 但 v 不认识 u 的情况。
- 社交网络中，有些人会认识很多人，有些人认识很少的人，甚至谁都不认识。
- 社交网络中，相邻的人经常会形成一些大大小小的团体，团体中的人往往相互认识。
- 社交网络中，一个人所认识的人可能分布比较广泛，如一个人的大学同学往往来自于全国不同城市，一个城市的人也可能生活在不同地区。
- 在打开二进制文件时使用"rb"而不是 “r” ，进而可以通过读取字符或字符串的方法来进行数据的读取。