Lab 2：过定点最短路

一、 实验介绍

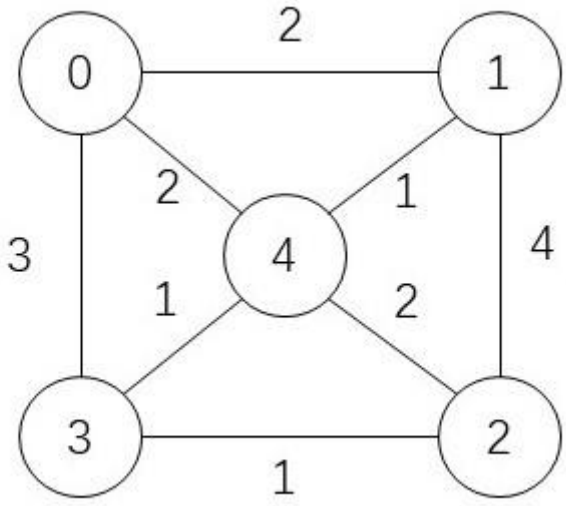
最短路问题是图论理论的一个经典问题，就是在指定图网络中找一条从起点到终点间权重最小的一条路径，解决这类问题的经典算法有 Dijkstra、Floyd 等。

最短路问题在实际生活中有非常广泛的应用。考虑这样一个具体的实际场景：幸运的上海交通大学学生源源在某天同时抢到了麦当劳、超市、小眷村等的购买机会，于是他需要从宿舍出发去购买对应的食物，如果只考虑路上的距离，哪条路径可以从宿舍出发，经过以上所有地点后再回到宿舍，且需要走的路最短呢？

二、 实验要求

Part 1 图计算部分

上述问题可以建模为一个加权无向图中的经过定点的最短路问题。在加权无向图 G = (V, E) 中，有 $X \in V$ 为起点和终点，点集 $M = \{M_1, M_2, \dots, M_n \mid M_i \in V\}$ 为必须要经过的节点。以下图为例， $X = 0$ 既是起点，也是终点，$M = \{1, 2, 3\}$ 是必经点集合。



1. 寻找定点最短路

下面给出解决该问题的一种思路：

1. 用最短路径算法（如 Dijkstra）计算出源节点 X 和 n 个必经节点所组成的点集中两两节点之间的最短距离和路径。
2. 对要经过的 n 个中间节点进行全排列，可以生成 $n!$ 个中间节点序列。
3. 对于每一个上述中间节点序列 $Q_1, Q_2, \dots, Q_n$ ，将 X 增加到首尾，可得到路径

   $X \to Q_1 \to Q_2 \to \cdots \to Q_n \to X$，

   该路径中相邻两点之间的最短路之和，即为选用该中间节点序列时的最短距离和路径。

   取所有情况中的最优值作为答案。

对于上述例子，我们可以有以下结果：

源节点及必经节点之间的最短距离（只列了正向，无向图中反向距离相同，路径方向相反）：

① 0->1：最短路径 0->1，最短距离为 2

② 0->2：最短路径 0->4->2 或 0->4->3->2 或 0->3->2，最短距离为 4

③ 0->3：最短路径 0->3 或 0->4->3，最短距离为 3

④ 1->2：最短路径 1->4->2 或 1->4->3->2，最短距离为 3

⑤ 1->3：最短路径 1->4->3，最短距离为 2

⑥ 2->3：最短路径 2->3，最短距离为 1

中间节点序列的全排列：

① (1,2,3)：中间节点距离和为 3+1=4，总距离和为 4+2+3 = 9

② (1,3,2)：中间节点距离和为 2+1=3，总距离和为 3+2+4 = 9

③ (2,1,3)：中间节点距离和为 3+2=5，总距离和为 5+4+3 = 12

④ (2,3,1)：中间节点距离和为 1+2=3，总距离和为 3+4+2 = 9

⑤ (3,1,2)：中间节点距离和为 2+3=5，总距离和为 5+3+4 = 12

⑥ (3,2,1)：中间节点距离和为 1+3=4，总距离和为 4+3+2 = 9

由上述结果，我们可以得出最短距离就是 9 了，并可按照对应的方案选择出一条最短路径，

如 0->1->4->3->2->3->0。

2. 输入输出

(1) 输入部分

本次实验的输入部分仍通过文件给出，内容如下图所示（以上面的图为例）：

```
5
0 2 @ 3 2
2 0 4 @ 1
@ 4 0 1 2
3 @ 1 0 1
2 1 2 1 0
0
1 2 3 $
```

第一行为节点数 N。

之后为 N×N 的邻接矩阵用来表示图信息，图是简单图，无需考虑自环和平行边。**值= 0 表示某个节点自己，(0,1) = 2 代表 0 和 1 之间的边权值为 2，(0,2) =@表示 0 和 2 之间没有直接相连的边**，在代码中会把这个符号解析成一个很大的数 INF（在.h 中有提供），所有测试数据中的边权值都为正数且远小于 INF。

之后每两行为一组测试：

两行中的第一行 0 表示问题中的起点和终点。

第二行 1,2,3 表示必须要经过的中间节点集, $ 表示一组节点集的结束。

每一行中的所有输入用空格隔开。

(2) 输出部分

在初始的 .h 文件的 FixedSP 类中，定义了一个 getFixedPointShortestPath 接口，接收起(终)点序号，以及中间节点集作为参数，该函数会在每组输入结束后调用，要求返回一个 int 数组，代表**任意**一条最短路径（如例子中，这个数组的内容可以为{0,1,4,3,2,3,0}），我们会对这个数组检查，打印其长度并验证其正确性，因此你不需要添加任何输出，只需要根据要求把路径放回这个数组即可。**特别注意，我们给的图可能不连通，因此可能不存在一条路径，此时数组应为空，对应的最短距离在参考的输出文件中用 INF 的数值（1e8）表示。**

Part 2 并行化编程

在之前的 lab 中，我们所写的都是单线程程序，或者说是“串行”执行的，同一时刻只有一条语句在执行，这样其实并没有充分利用机器上的计算资源（如多核 CPU 计算机上，只有一个核在运行）。在课上大家已经了解到了并行计算的基本概念，它的主要目的就是将一个计算量繁重的大任务分成几个子任务交由多份计算资源同时计算，来提高整体的执行效率。所以在 Part 2 中你要在 Part 1 内容的基础上做简单的并行化。

这一部分你需要按惯例完成一份 pdf 书面报告，应包括但不限于以下内容：

① 你的并行化思路

简要描述你将对于哪一部分计算任务进行并行化以及理由，至少对一个计算任务进行并行化。比如可能的并行化思路有：(1) 对于一个节点数为 V，中间定点个数为 N 的图，在没有预计算的情况下，使用 Dijkstra 算法计算一个全排列序列上相邻两点间最短距离的时间复杂度为 $O(N \times V^2)$，那么当图的规模变大时，计算的时间会变久，你可以将总的计算量分给多个线程去共同完成来提高这部分的执行效率；(2) 遍历所有全排列找最短距离是一个阶乘级的暴力解法，当 N 大到一定值时，会明显发现上述算法的执行效率变慢，因此这也是一个可以通过并行计算来提高执行速度的角度

② 如何具体实施并行化

详细描述你为了并行化①中的计算任务在代码层上的实现细节。

③ 性能评测与结论

对你多线程版本的过定点最短路算法进行性能评测。这部分应包括完成评测的**硬件环境、评测方案设计（输入数据设计和评测内容）、评测结果（图表）。**要评测的内容至少包括**不同数据规模和不同并发程度（线程数，必须包括单线程）下的执行效率对比。**在输入数据上请注意：本问题中的邻接矩阵是一个对称矩阵，且主对角线为 0。

根据你的评测结果给出对应的实验结论。如果不符合预期，请分析可能的原因。

提醒： 此书面报告相当于 Lab3 并行编程，因此占有整个 Lab3 的分数（比此前的书面报告分数更高），请认真对待！

三、 实验评分

1. 代码运行

我们提供了 Makefile 工具，在主目录下执行 **make** 指令生成可执行程序 main。你可以通过下述指令查看自己的实现情况：

```
./main d 输入文件的路径
```

例如

```
./main d ./input/fsp_input1
```

将使用 input 文件夹下的 fsp_input1 文件中的内容根据你的实现将输出打印到标准输出中作为 debug 参考。

2. 代码评分

你可以在主目录下执行 **make grade** 指令（或者形如./main g 的指令）查看自己实现是否正确，如果不正确，会打印出你的输出与答案的差异，请检查是否正确实现以及按输出部分的要求完成，如果正确实现，你应该看到如下输出：

```
<<<<<<<< grade test >>>>>>>>
  TEST1:    OK
  TEST2:    OK
  TEST3:    OK
  TEST4:    OK
  grade: 100/100
<<<<<<<< grade test over >>>>>>>>
```

通过它们你将拿到 Part1 的分数。

3. 实验上交

Part1 部分你可以随意修改 FixedSP.cpp 和 FixedSP.h，但不能修改 getFixedPointShortestPath 接口声明。**提交实验请将 FixedSP.cpp 和 FixedSP.h 打包上传 Canvas，命名使用“学号+姓名+lab2_Part1”，如“520123456789+张三**

+lab2_Part1.zip”。

Part2 部分可任意修改，**提交实验请将去掉输入数据集的所有代码以及 pdf 书面报告打包上传 Canvas 平台（会有另外的作业用于提交），命名使用“学号+姓名+lab2_Part2”，如“520123456789+张三+lab2_Part2.zip”**。

Part1 代码请提交能够通过 grade test 的单线程版本，Part2 的代码请提交你并行化之后的版本，不对 Part2 的 grade test 做测试。

Lab2 截止日期为 2022.05.26 21:59PM。

四、 注意事项

1. 切勿**抄袭**！如若发现明显雷同现象，该实验评分将以 0 分计。
2. 注意实验截止日期，迟交或不交会视情况扣分，因疫情防控原因无法按时上交的请及时与助教联系说明原因，会酌情适当放宽期限。
3. 请注意按照实验要求完成，避免因为自动脚本评分造成的判分失误。
4. 有问题随时通过邮件或微信群联系负责 Lab 的助教姬浩迪、陈沛东。