

EECS 281: 项目 4 - 口袋妖怪



截止时间为 2024 年 12 月 9 日星期一晚上 11:59

概述

您将获得一个图形，它是 xy 坐标（顶点）的集合，以及有关它们如何连接的规则（边）。这个项目分为三个部分：

- **A 部分**：计算**最小生成树 (MST)**以找到表示连接所有顶点的最低边权重成本的边子集。
- **B 部分**：研究并实施一种算法来近似解决**旅行商问题 (TSP)**；该解决方案不需要是最优的，但必须“足够接近”和“足够快”以满足自动评分器的要求。
- **C 部分**：使用分支定界算法计算最优 TSP 解决方案的权重。

项目目标

- 理解并实现MST算法
- 能够确定 Prim 算法或 Kruskal 算法在特定场景下是否更有效
- 理解并实现分支定界 (BnB) 算法
- 开发一种快速有效的 TSP 边界算法
- 探索各种启发式方法，快速实现近乎最优的解决方案
- 在线研究以识别、学习和实施各种“TSP 启发式方法”
- 使用图形可视化工具帮助调试

背景故事（有趣）

只带上一只皮卡丘，你，一名神奇宝贝训练师，就可以离开真新镇去填满你的神奇宝贝图鉴并捕捉所有的神奇宝贝！神奇宝贝遍布各地，你可以步行、冲浪或飞行来从一只神奇宝贝到另一只神奇宝贝。在你的旅途中，你可能会遇到其他神奇宝贝训练师，他们会让旅行变得更加困难，但一旦你与训练师战斗，你就再也不需要与他/她战斗了。你的任务是找出最有效的旅行路线。

在训练师逃避模式（A 部分）中，我们假设您没有遇到过任何训练师，并且您必须与其中一些训练师战斗，以便可以接触到该国的所有神奇宝贝。您需要通过与尽可能少的训练师战斗并行最短的距离来创建无训练师路径（一个指标将同时考虑这两者）。

在冠军模式（B 和 C 部分）中，我们假设您已经游览了整个国家，击败了所有训练师，并且可以使用飞行，这意味着您可以完全跳过与训练师的战斗，从一只小精灵飞到另一只小精灵。您将构建一条路径（更确切地说，当您返回家时，是一个循环）来捕捉每只小精灵，同时尽量减少行进距离。

需要明确的是：这些场景是分开的；程序将为其中一个场景创建一个计划，但不会在同一次运行中同时为两个场景创建一个计划（尽管您可能会发现一种模式的算法有助于另一种模式）。

在这个项目中，您的输出不需要与我们的完全相同才是正确的。只要它提供有效的答案，并遵循输出格式的规则，它可以是不同的顺序，但仍然是正确的。

常见项目元素


该项目的具体部分将在下面详细描述，但有一些共同的元素将在项目的所有部分中使用。

命令行参数

您的程序 `poke` 应该采用以下区分大小写的命令行选项：

- `-m`, `--模式{MST|FASTTSP|OPTTSP}`
 - `MST`：查找地图的最小生成树 (MST)
 - `FASTTSP`：找到一个快速但不一定是最优的 TSP 解决方案
 - `OPTTSP`：寻找 TSP 问题的最优解

此命令行选项是必需的，并且具有必需的参数。如果未正确包含，则将有用的错误消息打印到标准错误（`cerr`）和 `exit(1)`。程序输出格式由给定的指定 `mode`。

 所有模式都使用相同的输入格式，但每次只会测试一种模式。您的程序应该能够处理所有三种模式，但每次调用您的程序只会测试一种模式。您可能会发现一种模式在实现另一种模式时会很有用，但您只需为每种模式提供一个最终答案。

- `-h`, `--帮助`

打印该程序及其参数的简短描述 `exit(0)`。

如何执行程序的有效示例：

有效的命令行示例

```
1  $ ./poke --mode MST                (可以，但您必须手动输入)
2  ...
3  $ ./poke -h < inputFile.txt         ( OK, 在我们意识到没有-m之前-h已经发生了)
4  ...
5  $ ./poke -m OPTTSP < inputFile.txt  ( OK, 从磁盘上的文件读取)
6  ...
```

使用输入重定向时，它由 `shell` 处理，既不影响命令行，也不被命令行看到。重定向输入会通过 将指定文件的内容发送给程序 `cin`。您不必修改代码即可实现此功能；操作系统会处理它。

我们不会专门检查您的命令行处理错误，但是我们希望您的程序符合的默认行为 `getopt_long()`。不正确的命令行处理可能会导致各种难以诊断的问题。

输入文件格式

启动时，您的程序 `poke` 会从标准输入 () 读取图形顶点 `cin`。这些顶点是口袋妖怪的位置（与上面描述的背景故事相符）。地图可以表示为笛卡尔平面，您将获得一个整数， N ，表示图中有多少个顶点。接下来是 N 描述笛卡尔平面上点的有序整数对，每行一个，由单个空格分隔。

顶点由整数索引标识，这些索引对应于读取它们的顺序（第一个顶点位置是位置 0，第二个顶点是位置 1，等等）。输入将始终格式正确并符合此格式，因此无需进行错误检查。可能会有额外的空白行，但仅限于文件末尾。


```
地图输入示例 (p4-pokemon/spec-test.txt)

1 5
2 6 1
3 二 三
4 -5 -4
5 -1 6
6 0 -1
```

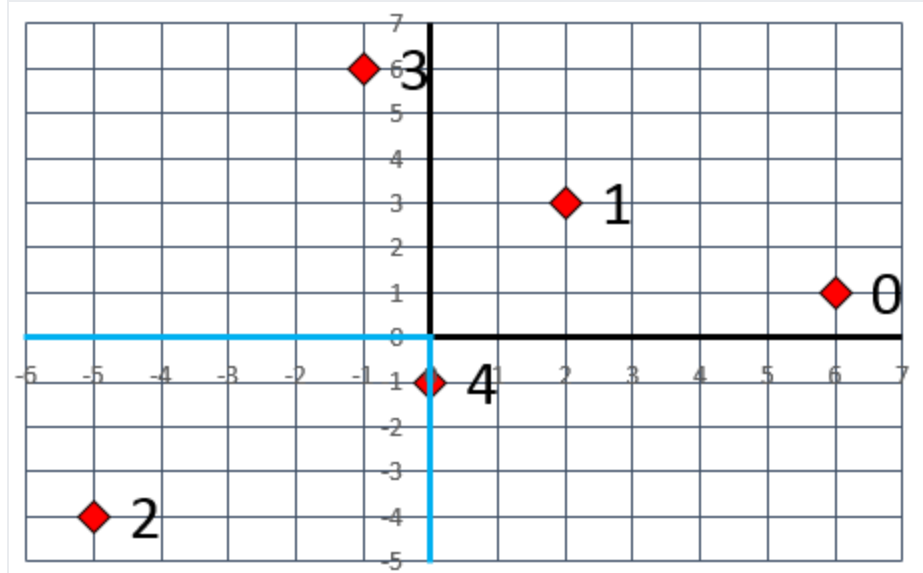
对于 A 部分，至少有 2 分。对于 B 和 C 部分，至少有 3 分。

地图区域

地图将分为三个区域：海洋（笛卡尔平面中的 QIII）、陆地（QI、QII 和 QIV）以及它们之间的海岸线（原点加上负 x 轴和 y 轴）。口袋妖怪可以出现在这些区域中的任何一个，并且只能出现在一个区域中。当口袋妖怪的区域很重要时（[第 A 部分](#)），如果操作有效，可以将其与坐标一起存储。选择这样做是可选的，也是空间和时间之间权衡的一个很好的例子，用于存储口袋妖怪区域的额外空间可以帮助在评估两个口袋妖怪的区域时节省时间。

 **专业提示：**对顶点进行分类时，请记住二分搜索的分析，其中最早的比较用于获取最大信息量，而最不可能的结果则保存到最后，而无需任何特定条件。这是一个二维图，因此需要多次比较和逻辑组合，但同样的想法也适用，可用于简化代码。

上述示例可以如下图所示进行可视化，其中显示的数字是位置索引，蓝线表示将陆地与海洋分开的海岸线。位置 2 在海上，位置 4 在海岸线上，其余位置在陆地上。



在您的程序中，有多种方式可以内部表示此配置，这将影响您的运行时。请明智地选择您的数据结构！

输出和自动评分器

在这个项目中，您的输出不需要与我们的输出完全相同才是正确的。只要您的输出提供了有效的树或路径权重并遵循格式规则，它可以是不同的树或路径，但仍然是正确的。或者，您的解决方案可能会找到与自动评分器上的树或路径相同的树或路径，并以不同的顺序打印它，但仍然是正确的。任何被判定为正确的内容都将获得满分，而不会因为与自动评分器不同而受到惩罚。

距离计算

要计算两点之间的距离，请使用[欧几里得距离](#)， $d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$ ，并将距离存储为 `double s`。在计算距离时，请务必小心舍入误差；自动评分器将允许您有 0.01 的误差幅度以考虑舍入，但您应避免在中间步骤进行舍入。

专业提示：

1. 使用 `pow()` 来计算数字的平方是不必要的，可以使用乘法运算符来避免。例如，`pow(x, 2)` 可以写成 `x * x`，这样 `x * x` 执行速度更快。
2. `int` 在将中间计算存储到输出之前，请仔细考虑可能溢出范围的中间计算 `double`。
3. `sqrt()` 是一个相对较慢的函数，如果考虑以下几点，可以避免： $a < b$ 然后 $\sqrt{a} < \sqrt{b}$ 。当涉及到总数时，情况就不一样了，所以 $a + b < c + d$ 并不保证 $\sqrt{a} + \sqrt{b} < \sqrt{c} + \sqrt{d}$ ，因此不应使用。

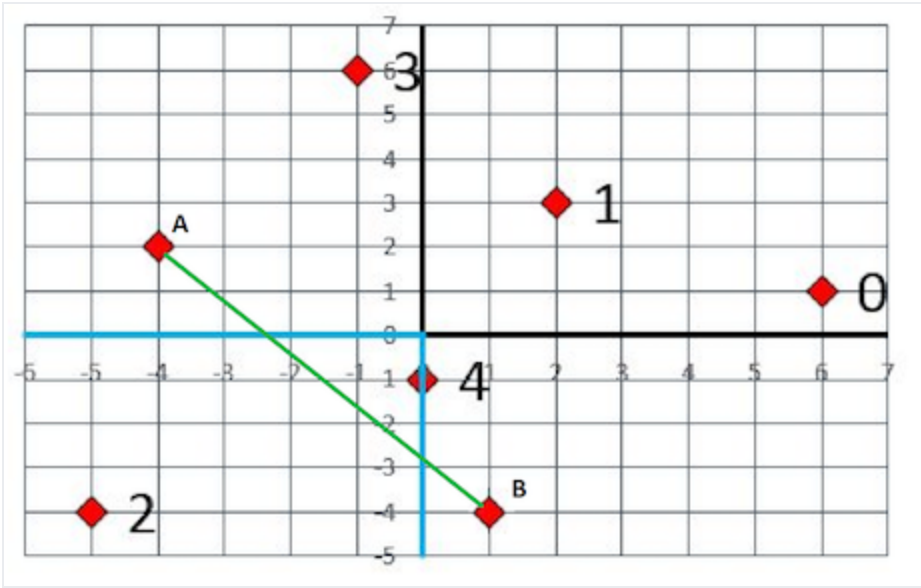
当您寻找路线时（在 B 部分和 C 部分中），您可以直接从海上地点前往陆地地点，反之亦然；换句话说，您可以忽略海岸线的限制。

A 部分：MST 模式

在此模式下，您将设计一个口袋妖怪网络 (MST)，连接每个口袋妖怪的位置，同时最小化所需的连接总长度。您可以使用任何 MST 算法来连接所有位置。提示：除非您想同时实现两者并进行比较，否则请考虑图形的性质（它有多少个顶点和边？）。您可以自由地调整讲座幻灯片中的代码以适合这个项目，但您需要仔细考虑完成每个部分所需的数据结构（存储不必要的数据可能会超出内存限制）。您的程序必须始终为每个输入生成一个有效的 MST。

在 A 部分中连接地点时，仅允许在同一区域内的地点之间轻松移动（参见[地图区域](#)）。从陆地前往海洋必须经过海岸线上的一个地点，并且海岸线也必须用于从陆地前往海洋。例如，您不能直接从 (-5, -4) 前往 (6, 1)。您必须先从 (-5, -4) 前往 (0, -1)，然后从 (0, -1) 前往 (6, 1)。

为简单起见，假设两个陆地位置之间有直接连接且跨越水面的任何旅行都将通过空中（跳跃或飞行）完成。下面的示例显示了两个有效连接的陆地位置 A 和 B。



A 部分距离

同一区域内任意两点之间的距离，或任意地点与海岸线上任意地点之间的距离，都是两点之间的欧几里得距离。陆地上任意地点与海洋上任意地点之间的距离实际上是无限的，因为您无法直接在两地之间旅行。

A 部分 图形表示

MST 模式可能至少有两个顶点，但自动评分器也会对数万个位置进行测试。距离矩阵 $|V|^2$ `double` 值不能保证适合内存。由于您无法提供足够的空间来查找边缘权重，因此您必须根据需要计算位置之间的距离。

A 部分输出格式

对于 MST 模式，您应该在一行上打印您自己生成的 MST 的总权重；此权重是 MST 中所有边的权重之和（以[欧几里得距离](#)为单位）。然后，您应该打印 MST 中的所有边。所有输出都应打印到标准输出（`cout`）。

输出应采用以下格式：

样本输出格式	
1	重量

```
2  节点 节点
3  节点 节点
4  .....
```

这里，`node` 是对应于 MST 顶点的位置索引，输出中给定行上的一对节点描述 MST 中从第一个节点到第二个节点的一条边。权重应格式化为双精度（2 个小数点精度就足够了 - 参见附录 A），打印时节点号应全部为整数值。例如，给定上面的示例输入文件，您的 MST 模式输出可能是：

MST 示例输出 (p4-pokemon/spec-test-MST-out.txt)

```
1  19.02
2  0 1
3  二 四
4  1 3
5  1 4
```

您还应始终打印描述边的顶点对，使得左侧索引的整数值小于右侧索引的整数值。换句话说：

```
1 2
```

是输出的可能有效边沿，但是

```
2 1
```

不是有效输出。

如果无法为给定位置构建 MST，因为陆地和海上都有位置而没有海岸线位置，则程序应将消息“无法构建 MST”打印到 `cerr` 和 `exit(1)`。

B 部分：FASTTSP 模式

在此模式下，您将弄清楚如何前往每个口袋妖怪，然后返回您的起始位置。您将实施[旅行商问题 \(TSP\)](#)的解决方案。路线将始终从文件中的第一个位置（索引 0）开始，访问其他每个位置一次，然后返回起点。您必须解决 TSP 并选择前往各个位置的路径，以最小化总行程距离。

在 FASTTSP 模式下，可能有大量位置需要访问，找到一种访问所有位置的最佳方法可能太慢，而且在您完成任务之前太阳可能已经冷却。您可以使用启发式方法来找到近乎最佳的路径。启发式是一种解决问题的方法（一种算法），它可以产生一个好的答案，但不一定是最好的答案。例如，您可以推测性地跳过一个分支，而不是等待知道它可以被跳过的事实。还有许多其他简单的启发式技术，例如从随机路径开始，并尝试通过小的更改来改进它。在线搜索“TSP 启发式”。有几种类型，有些更容易实现，有些具有更好的路径长度，有些两者兼而有之。确保您选择并实现的算法 $O(n^2)$ 。

您应该为旅行推销员问题 (TSP) 提出一个接近最佳旅行长度的解决方案。考虑到时间限制，您将无法可靠地找到最佳解决方案，尽管在幸运的情况下（取决于所选的图表和算法），偶尔可以找到最佳解决方案。

您可以使用我们在课堂上讲过的此部分的任意算法组合，包括您为 A 部分编写的 MST 算法。

在为本节设计算法时，你需要发挥创造力。你可以自由地实现你选择的任何其他算法，只要它满足时间和内存限制。但是，你不应该使用任何与课堂上所讲内容有显著不同的高级算法或公式（例如模拟退火、遗传算法和禁忌搜索 - 它们太慢了）。相反，创造性地结合你已经知道的算法，并提出简洁的优化。你的启发式算法很可能在某种程度上是贪婪的，但贪婪的方式有很多种！

B 部分距离

任意两个位置之间的距离都是该两个位置之间的欧几里得距离。

旅程的长度定义为所有成对行进距离的总和 - 即所有沿途边的长度总和。

B 部分图形表示

FASTTSP 模式可能至少有三个顶点，但自动评分器也会运行数万个位置的测试。距离矩阵 $|V|^2$ double 值不能保证适合内存。由于您无法提供足够的空间来查找边缘权重，因此您必须根据需要进行计算位置之间的距离。

B 部分输出格式

您应该通过在一行上打印整个行程的长度来开始输出。在下一行中，按访问节点的顺序输出节点。初始节点应为起始位置索引，最后一个节点应为返回到第 0 个位置之前的位置编号。行程中的节点应以单个空格分隔的方式打印。列出的最后一个位置后可以有一个空格。所有输出都应打印到标准输出 (`cout`)。

例如，如果给定上述输入文件，您的程序可以产生以下任一正确输出：

```
FASTTSP  示例输出 (p4-pokemon/spec-test-FASTTSP-out.txt)

1  31.64
2  0 4 2 3 1
```

或者

```
FASTTSP  备用样本输出

1  31.64
2  0 1 3 2 4
```

C 部分：OPTTSP 模式

有关 TSP 问题的描述，请参阅[B 部分](#)。在此模式下，您将实现一个保证最佳的 TSP 解决方案。您将使用 `genPerms()` 来查找一条始终从位置索引 0 开始、访问每个其他位置一次并返回起点的路线。您必须解决 TSP 并选择连接位置的边，以最小化总行驶距离。

要找到最佳路径，你可以从穷举法开始，这种方法会评估每条路径并挑选一条最小的路径。通过巧妙地构造这种枚举，你可以确定搜索的某些分支无法得出最佳解决方案。例如，你可以计算给定分支中可以找到的任何完整路径的长度的下限。如果这样的下限超过了你之前找到的完整解决方案的成本，你可以跳过这个分支，因为它没有希望。如果正确实施，这种分支定界法应该总能产生最佳解决方案。它不会像排序或搜索算法那样适用于其他问题，但它应该适用于少数位置。巧妙的优化（尽早识别无望的搜索分支）可以使你的算法快一百倍。在纸上绘制

TSP 路径并手动解决小位置配置以达到最优应该非常有用。请记住，运行边界函数所需的时间和边界允许你修剪的分支数量之间存在权衡。

确保您使用的 `genPerms()` 是使用 `swap()` 单个容器的版本，而不是速度慢得多的双容器版本（堆栈和队列）。

给定一个输入集 N 整数坐标定义的位置，您的任务是使用分支定界算法生成最佳路径。您的程序应始终生成尽可能短的路径作为解决方案，即使计算该解决方案非常耗时。您将有 35 秒的 CPU 时间限制来生成解决方案。如果您的程序没有生成有效的解决方案，它将无法通过测试用例。您的解决方案还将根据以前的项目通过时间和空间预算进行评判。

C 部分 距离

与B 部分距离相同。

C 部分图形表示

OPTTSP 模式可能至少有三个顶点，但由于时间限制，自动评分器将仅运行几十个位置的测试。距离矩阵 $|V|^2$ `double` 值将适合内存，因此您可以提供足够的空间来预先计算所有边缘权重。然而，在测试中，这并不比根据需要计算它们快得多，并且在某些情况下会使编码更加困难。我们在有和没有距离矩阵的情况下都对此进行了编码，两个版本都获得了满分。

C 部分输出格式

与B 部分输出格式相同。

测试和调试

该项目的一部分是准备几个测试文件，这些文件将揭露有缺陷的解决方案（无论是您自己的还是其他人的）。由于这对于测试和调试您的程序非常有用，我们建议您在完成解决方案之前，先尝试使用测试文件捕获我们故意制造的几个错误解决方案。自动评分器还会告诉您是否有自己的测试文件暴露了解决方案中的错误。

您提交的每个测试都应包含一个输入文件。当我们在故意出现错误的项目解决方案之一上运行您的测试文件时，我们会将输出与正确的项目解决方案进行比较。如果输出不同，则测试文件被认为暴露了该错误。

测试文件应命名为 `test-n-MODE.txt`，其中 $1 \leq n \leq 10$ 。自动评分器的错误解决方案将以指定的模式运行您的测试文件。模式必须是 MST、FASTTSP 或 OPTTSP。

您的测试文件在任何一个文件中都不得超过 10 个坐标。您最多可以提交 10 个测试文件（尽管使用较少的测试文件也可以获得满分）。自动评分器对您的解决方案运行的测试不限于每个文件中 10 个坐标；您的解决方案不应施加任何大小限制（只要有足够的系统内存）。

提交至自动评分器

将所有工作（包括所有需要的源代码文件以及测试文件）放在主目录以外的某个目录中。这将是您的“提交目录”。在提交代码之前，请确保：

- 每个源代码和头文件在文件顶部的注释中包含以下项目标识符：

```
// Project Identifier: 5949F553E20B650AB0FB2266D3C0822B13D248B0
```
- Makefile 也必须有这个标识符（在第一个 TODO 块中）
- 您已删除所有 .o 文件和可执行文件。输入“make clean”即可完成此操作。
- 您的 makefile 名为 Makefile。输入“make -R -r”可无错误地构建您的代码并生成一个名为 poke 的可执行文件。命令行选项 -R 和 -r 禁用自动构建规则，这在自动评分器上不起作用。
- 您的 Makefile 指定您正在使用 gcc 优化选项 -O3 进行编译。这对于获得所有性能点非常重要，因为 -O3 可以将代码速度提高一个数量级。
- 您的测试文件名为 test-n-MODE.txt，并且没有其他以 test 开头的项目文件名。最多可提交 10 个测试文件。文件名的“模式”部分必须是 MST、OPTTSP 或 FASTTSP。
- 您的程序和测试文件的总大小不超过 2MB。
- 你的提交目录（即 git 源代码管理使用的 .git 文件夹）中没有任何不必要的文件（包括你的文本编辑器和编译器等创建的临时文件）或子目录。
- 使用 g++ 编译器 6.2.0 版，您的代码可以正确编译和运行。这在 CAEN Linux 系统上可用（您可以通过 login.engin.umich.edu 访问）。即使一切似乎都可以在另一个操作系统或不同版本的 GCC 上运行，课程工作人员也不会支持在 Linux 上运行除 GCC 6.2.0 之外的任何版本（使用其他编译器和操作系统的学生确实观察到了不兼容性）。要在 CAEN 上使用 g++ 6.2.0 版进行编译，您必须将以下内容放在 Makefile 的顶部：

```
1  PATH := /usr/um/gcc-6.2.0/bin:$ (路径)
2  LD_LIBRARY_PATH := /usr/um/gcc-6.2.0/lib64
3  LD_RUN_PATH := /usr/um/gcc-6.2.0/lib64
```

提交以下所有文件：

- 项目的所有 .h 和/或 .cpp 文件
- 你的 Makefile
- 您的测试文件

您必须准备一个压缩的 tar 存档 (.tar.gz 文件)，其中包含要提交给自动评分器的所有文件。一种方法是将所有要提交的文件放在一个目录中。在此目录中，运行

```
1  $ tar -czvf submit.tar.gz Makefile * .cpp * .h test- * .txt
2  Makefile
3  一个main.cpp
4  ...
```

这将在您的工作目录中准备一个合适的文件。或者，281 Makefile 有有用的目标 fullsubmit 和 partialsubmit 可以为您完成此操作。使用命令 make help 了解它还能做什么！

将您的项目文件直接提交至 281 自动评分器：<https://eecs281ag.eecs.umich.edu/>。您每天最多可以提交四次（春季提交次数更多）。为此，每天的开始和结束时间均为午夜（安娜堡当地时间）。我们只会计算您提交的最佳成绩。如果您希望我们使用您的最后一次提交，请参阅自动评分器常见问题解答页面，或 [使用此表单](#)。

请务必阅读自动评分结果顶部显示的所有消息！这些消息通常有助于解释您遇到的一些问题（例如，由于 Makefile 错误而丢分，或者为什么会出现分段错误）。此外，请务必检查自动评分是否显示您自己的测试文件之

一暴露了解决方案中的错误（在底部）。在自动评分结果中搜索单词“提示”（不带引号）以获取可能有用的信息。

库和限制

我们强烈建议在本项目中使用 STL，但有几个禁止使用的功能除外。请勿使用：

- C++11 正则表达式库
- STL 线程/原子库（破坏运行时测量）
- 共享或唯一的指针。
- 其他库（例如，boost、pthreads 等）。

等级

80 分 – 您的成绩将根据正确性和性能（运行时间）得出。这将由自动评分器决定。对于这个项目，我们期望运行时间比以前的项目更广泛。与所有项目一样，用于最终评分的测试用例可能会有所不同。

10 分 – 您的程序没有内存泄漏。确保每次提交之前在 valgrind 下运行您的代码。这也是一个好主意，因为它会让您知道是否有未定义的行为（例如读取未初始化的变量），这可能会导致您的代码在自动评分器上崩溃。

10 分——学生测试文件覆盖率（揭露有缺陷的解决方案的有效性）。

运行时质量权衡

在这个项目中，没有单一的正确答案（与以前的项目不同）。因此，您的问题的评分不会像“差异”那么简单，而是对您的输出进行评估的结果。例如，如果我们为您提供了 A 部分的正方形，您可以选择 4 条边中的任意 3 条，并以任何顺序打印它们，这意味着有 24 个可能的正确输出文件！

尤其是对于 B 部分，我们预计学生的产出会存在较大差异。B 部分要求您解决一个难题，在给定的时间限制内，我们实际上并不期望您的产出在所有情况下都是最优的。您的解决方案的质量甚至可能因情况而异。我们希望您能够快速得出接近最优的解决方案。这不可避免地会在解决方案最优性和运行时间之间产生权衡。

您可能会发现，实现在 B 部分中使用的多种算法或启发式方法很有用，并且进行一些测试并使用自动评分器来确定哪种方法效果最好。

B 部分的成绩将根据您与最佳解决方案的接近程度（以百分比计算）来确定。

脚注 2：本项目中的“路径”可以理解为两个地点之间的路线。

脚注 3：此类启发式算法也称为近似算法。