Minimal Spanning Tree

เรื่อง การเดินทางภายในราชวิทยาลัยจุฬาภรณ์

จัดทำโดย

1.นางสาวธนพร	ป้อมสถิตย์	รหัสนักศึกษา 64070505206
2.นางสาวกรณปัทม์	นาคเทศ	รหัสนักศึกษา 64070505219
3.นางสาวพิมพ์ชนก	ธัญญเจริญ	รหัสนักศึกษา 64070505224
4.นางสาวเจษฎาพร	นันทะสิทธิ์	รหัสนักศึกษา 64070505226
5.นายธีรภัทธ์	ชำพาลี	รหัสนักศึกษา 64070505231

เสนอ

ผศ. สนั่น สระแก้ว

รายงานนี้เป็นส่วนหนึ่งของรายวิชา Discrete Mathematics (CPE103)

คณะวิศวกรรมศาสตร์ ภาควิชาวิศวกรรมคอมพิวเตอร์ สาขาวิทยาศาสตร์ข้อมูลสุขภาพ

มหาวิทยาลัยเทคโนโลยีพระจอมเกล้าธนบุรี

ภาคเรียนที่ 1 ปีการศึกษา 2564

คำนำ

รายงานฉบับนี้เป็นส่วนหนึ่งของวิชา CPE103 Discrete Mathematics ภาคเรียนที่ 1 ปีการศึกษา 2564 คณะวิศวกรรมศาสตร์ ภาควิชาวิศวกรรมคอมพิวเตอร์ สาขาวิทยาศาสตร์ข้อมูลสุขภาพ มหาวิทยาลัยเทคโนโลยีพระจอมเกล้าธนบุรี จัดทำเพื่อใช้โปรแกรมคอมพิวเตอร์มาหาเส้นทางที่สั้นที่สุดในการเดินทางภายในวิทยาลัยแพทยศาสตร์ศรีสวางควัฒน ราชวิทยาลัยจุฬาภรณ์

รายงานเล่มนี้สำเร็จลุล่วงได้ด้วยความกรุณาและความอนุเคราะห์จาก ผศ.สนั่น สระแก้ว และคณะผู้สอนรายวิชา Discrete Mathematics ที่กรุณาสละเวลาอันมีค่าให้คำปรึกษาและข้อเสนอแนะต่าง ๆ ตั้งแต่เริ่มต้นจนสำเร็จเรียบร้อย ผู้จัดทำขอขอบพระคุณเป็นอย่างสูง

ทางคณะผู้จัดทำหวังว่า รายงานฉบับนี้จะเป็นประโยชน์ต่อผู้อ่าน หรือนักเรียน นักศึกษาที่กำลังหาข้อมูลเรื่องนี้อยู่หากมีข้อแนะนำหรือข้อผิดพลาดประการใด คณะผู้จัดทำขอน้อมรับไว้และข้ออภัยมา ณ ที่นี้ด้วย

คณะผู้จัดทำ

สารบัญ

เหตุผล

ด้วยสถานการณ์โควิด-19 ทำให้พวกเราไม่ได้เข้าไปเรียนที่ ราชวิทยาลัยจุฬาภรณ์ ทำให้พวกเราไม่คุ้นเคยกับเส้นทางและอาคารเรียนภายในราชวิทยาลัยจุฬาภรณ์ พวกเราจึงอยากศึกษาเส้นทางของราชวิทยาลัยจุฬาภรณ์ และหาเส้นทางที่สั้นที่สุดในการเดินทาง

ปัญหาในชีวิตจริง

จากสถานการณ์การแพร่ระบาดของไวรัสโควิด-19 ในปัจจุบัน ทำให้นักศึกษาจำนวนมากต้องเรียนออนไลน์อยู่ที่บ้าน และมีนักศึกษาชั้นปีที่ 1 ที่ไม่เคยเข้าไปเรียนหรือใช้ชีวิตจริงในรั้วมหาวิทยาลัยเลย

ทำให้เราไม่คุ้นเคยกับสถานที่และเส้นทางภายในมหาวิทยาลัย ดังนั้นกลุ่มของพวกเราจึงเล็งเห็นปัญหาข้อนี้ เลยได้นำเอาความรู้ที่เรียนมาจากวิชา Discrete Mathematics หรือ CPE 103 มาเขียนโปรแกรมขึ้น เพื่อที่จะได้คุ้นเคยกับเส้นทางและหาเส้นทางที่เร็วที่สุดในการเดินทางไปยังสถานที่ต่าง ๆ ในมหาวิทยาลัย

วัตถุประสงค์

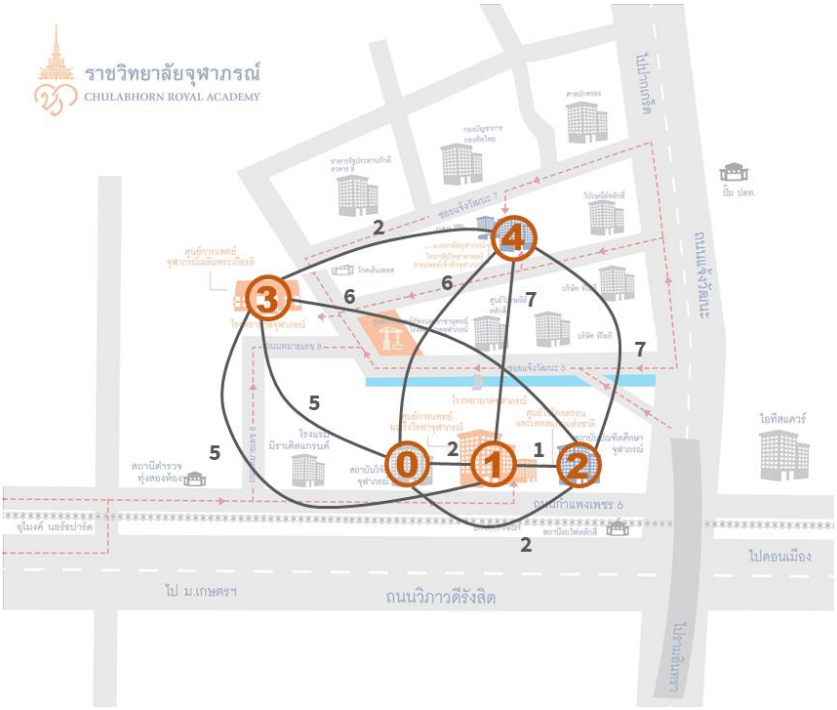
1.เพื่อค้นหาเส้นทางที่ประหยัดที่สุดสำหรับการเดินทางด้วยรถยนต์ภายในวิทยาลัยแพทยศาสตร์ศรีสวางควัฒน ราชวิทยาลัยจุฬาภรณ์

2.เพื่อนำความรู้ที่ได้จากการเรียนวิชา Discrete Mathematics มาประยุกต์ใช้ในชีวิตจริง

การเปลี่ยนจากปัญหาให้เป็นกราฟ

1.เปลี่ยนสถานที่ต่าง ๆ ในราชวิทยาลัยจุฬาภรณ์ให้เป็น Vertices

2.เปลี่ยนระยะเวลาการเดินทางไปยังสถานที่ต่าง ๆ ให้เป็น Edge



จากกราฟจะเห็นได้ว่า

Vertices ที่ 0 คือ สถาบันวิจัยจุฬาภรณ์

Vertices ที่ 1 คือ โรงพยาบาลจุฬาภรณ์

Vertices ที่ 2 คือ สถาบันบัณฑิตศึกษาจุฬาภรณ์

Vertices ที่ 3 คือ ศูนย์การแพทย์จุฬาภรณ์เฉลิมพระเกียรติ

Vertices ที่ 4 คือ วิทยาลัยวิทยาศาสตร์การแพทย์เจ้าฟ้าจุฬาภรณ์

การเปลี่ยนจากกราฟเป็นเมทริกซ์

1.กำหนดเมทริกซ์ให้เป็นสี่เหลี่ยมจัตุรัสให้มีขนาดเท่ากับ จำนวน Vertices คูณ จำนวน Vertices

2.ให้แสดงระยะเวลาการเดินทางจาก Vertices หนึ่งไปยัง Vertices หนึ่งถ้า Vertices นั้นไม่มีความสัมพันธ์กันให้ใช้เลข 0

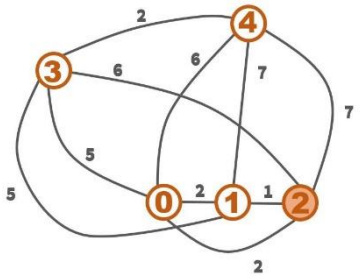
	0	1	2	3	4
0	0	2	2	5	6
1	2	0	1	5	7
2	2	1	0	6	8
3	5	5	6	0	2
4	6	7	8	2	0

การใช้ Prim’s Algorithm มาประยุกต์ในการแก้ปัญหา

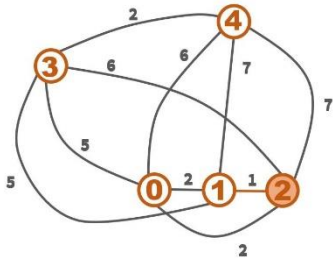
Prim’s Algorithm เป็นหนึ่งใน greedy algorithm ที่ใช้ค้นหา minimal spanning tree สำหรับกราฟแบบถ่วงน้ำหนักที่ โดยจะต้องหาเส้นทางที่สามารถไปครบทุก vertices และมีน้ำหนักรวมของทั้งหมดในต้นไม้ให้น้อยที่สุด Algorithm นี้ทำงานโดยการพิจารณา tree ย่อย ๆ จากจุดยอดเริ่มต้นแบบสุ่มและต้องไม่เกิดวงจรขึ้น

ขั้นตอนของ Prim’s Algorithm

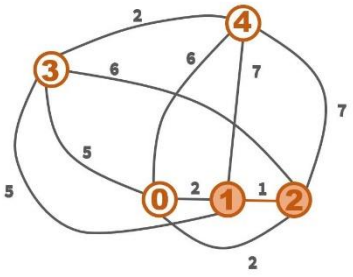
1.เลือก vertices แบบสุ่ม



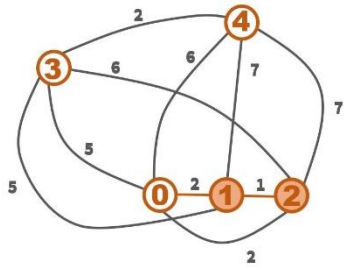
2.เลือกเส้นทางที่มีน้ำหนักน้อยที่สุดที่ติดกับ vertices ที่เลือกไว้



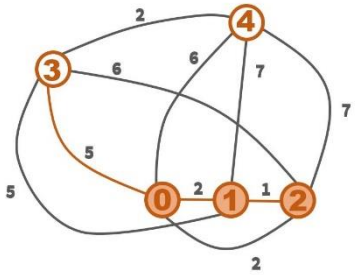
3.เส้นทางที่เลือกจะนำไปสู่ vertices ใหม่ ให้เรามองทั้งหมดนั้นเป็น tree

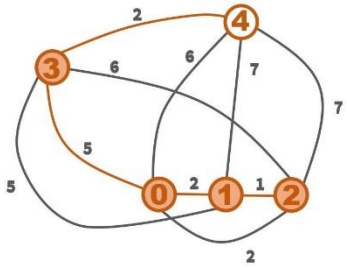


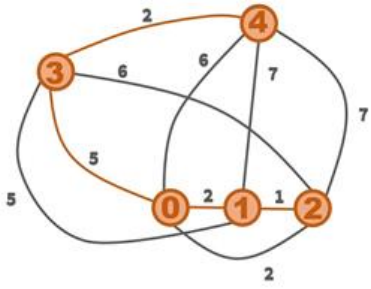
4.เลือกเส้นทางใหม่อีกครั้งโดยเลือกเส้นทางที่น้ำหนักน้อยที่สุดที่เชื่อมต่อกับ tree ในข้อ 3



5.ทำข้อ 3 และ 4 ซ้ำจนกว่าจะเชื่อมต่อกันครบทุก vertices







โปรแกรม

```python
# source code : https://github.com/ElHurta/prims-algorithm-python/blob/main/prims-algorithm.py
def primsAlgorithm(vertices):

    # Creating the adjacency Matrix with n x n dimensions filled with zeros, where n is the graph number of vertices:
    adjacencyMatrix = [[0 for column in range(vertices)]
                       for row in range(vertices)]

    # Creating another adjacency Matrix for the Minimum Spanning Tree:
    mstMatrix = [[0 for column in range(vertices)]
                       for row in range(vertices)]
```

สร้างฟังก์ชัน prim’s Algorithm เพื่อใช้ในการประมวณผลหาเส้นทางที่คุ้มค่าที่สุด โดยมี parameter เป็นจำนวน vertices ภายในฟังก์ชันจะมีการสร้างเมทริกซ์ปรับค่าได้ จำนวน 2 เมทริกซ์ ได้แก่ เมทริกซ์สำหรับรับค่า และ เมทริกซ์สำหรับการทำ minimal spanning tree แต่ละเมทริกซ์จะมีขนาด เท่ากับ จำนวน vertices * จำนวน vertices สมาชิกทุกตัวเริ่มต้นจะเป็น 0

```python
    # Filling the adjacency matrix:
    for i in range(0,vertices):
        # Since the adjacency matrix is Symmetric we don't have to fill the whole matrix, only the upper half:
        for j in range(0+i,vertices):
            # Asking for the edges weights:
            adjacencyMatrix[i][j] = int(input('Enter the path weight between the vertices: ({}, {}):  '.format(i,j)))

            # Again, we use the Symmetric Matrix as an advantage:
            adjacencyMatrix[j][i] = adjacencyMatrix[i][j]
```

ใช้ for loop ในการรับค่าน้ำหนัก edge ใส่ลงในเมทริกซ์สำหรับรับค่าเริ่มต้นจากผู้ใช้ ให้ครบทุกสมาชิก และจากการใช้ประโยชน์เรื่องความสมมาตรของเมทริกซ์ จะทำให้เราไม่ต้องกรอกน้ำหนักเองทุกค่า ยกตัวอย่างเช่น ค่าน้ำหนักในตำแหน่ง (2,4) และตำแหน่ง (4,2) จะมีค่าเท่ากัน จึงกรอกแค่ค่าในตำแหน่ง (2,4) ครั้งเดียวก็เพียงพอ

```
# Once the Adjacency Matrix is filled, we can start looking for the MST:
# Defining a really big number:
positiveInf = float('inf')

# This is a list showing which vertices are already selected so we don't pick the same vertex twice and we can actually know
selectedVertices = [False for vertex in range(vertices)]
```

เมื่อกรอกครบทุกสมาชิกแล้ว กำหนดค่า infinity ซึ่งให้เป็นค่าน้อยที่สุดชั่วคราว ไว้ในตัวแปร positiveInf สำหรับใช้ในการเปรียบเทียบในขั้นตอนต่อๆ ไป และสร้าง list ชื่อว่า selectedVertices ภายในมีสมาชิกเป็น False จำนวนเท่ากับ vertices เช่น vertices = 5 จะได้ว่า selectedVertices = [False, False, False, False, False] เพื่อใช้ในการตรวจสอบว่าได้ใช้ vertices นั้นไปแล้วหรือยัง ป้องกันการเลือก vertices ซ้ำ

```
# While there are vertices that are not included in the MST, keep looking:
while(False in selectedVertices):
    # We use the big number we created before as the possible minimum weight
    minimum = positiveInf

    # The starting vertex
    start = 0

    # The ending vertex
    end = 0

    for i in range(0,vertices):
        # If the vertex is part of the MST, look its relationships
        if selectedVertices[i]:
            # Again, we use the Symmetric Matrix as an advantage:
            for j in range(0+i,vertices):
                # If the vertex analized have a path to the ending vertex AND its not included in the MST (to avoid cycles)
                if (not selectedVertices[j] and adjacencyMatrix[i][j]>0):
                    # If the weight path analized is less than the minimum of the MST
                    if adjacencyMatrix[i][j] < minimum:
                        # Defines the new minimum weight, the starting vertex and the ending vertex
                        minimum = adjacencyMatrix[i][j]
                        start, end = i, j

    # Since we added the ending vertex to the MST, it's already selected:
    selectedVertices[end] = True


    # Filling the MST Adjacency Matrix fields:
    mstMatrix[start][end] = minimum

    if minimum == positiveInf:
        mstMatrix[start][end] = 0

    mstMatrix[end][start] = mstMatrix[start][end]

# Show off:
print(mstMatrix)
```

โค้ดส่วนนี้เป็นส่วนประมวลผลมีการวนซ้ำแบบ while loop จนกว่า selectedVertice จะกลายเป็น True ทั้งหมดซึ่งหมายความว่า เลือกใช้ครบทุก vertices และมีการกำหนดตัวแปร start และ end ซึ่งแทนการเชื่อมต่อและยังใช้ในการเติมค่าลงไปในเมทริกซ์ minimal spanning tree

ภายในจะประกอบไปด้วย 2 ลูป ได้แก่ ลูป rows และ ลูป columns โดยจะใช้ข้อมูลจาก เมทริกซ์ที่รับค่ามาจากผู้ใช้ โดยลูปที่ 1 จะทำหน้าที่ในการตรวจสอบว่า vertices ที่เลือกอยู่ใน minimal spanning tree หรือไม่ ซึ่งในการทำงานครั้งแรกสมาชิกทุกตัวใน selectedVertice เป็น False ทำให้ออกจากลูปก่อนที่จะเข้าทำงานลูป 2 ซึ่งเป็นเหตุผลที่กำหนดตัวแปรเริ่มต้น start และ end เป็น 0 เมื่อออกจากลูปจะกำหนดให้สมาชิก selectedVertice ตำแหน่งนั้นเป็น True และจะมีการเติมค่าสมาชิกในใน minimal spanning tree เมทริกซ์ด้วยค่า minimum และใช้ประโยชน์จากการสมมาตรของเมทริกซ์ เพื่อเติมค่าในตำแหน่งที่ตรงกัน

ลูป 2 มีการตรวจสอบเงื่อนไข 2 ครั้ง เงื่อนไขที่ 1 เพื่อตรวจสอบว่ามีเส้นทางให้ไปต่อและไม่มีการเลือก vertices ซ้ำ ซึ่งจะทำให้เกิดวงจร เมื่อผ่านการตรวจสอบเงื่อนไขที่ 1 จะเข้าสู่การเปรียบเทียบน้ำหนัก หากน้ำหนักนั้นน้อยกว่าค่า minimum จะประกาศค่านั้นให้เป็น minimum ตัวใหม่ โดยตัวแปร start และ end จะถูกอัพเดตด้วยค่า i และ j ตามลำดับ หากค่าน้ำหนักมีค่ามากกว่า minimum ก็จะทำการค้นหาจาก column ที่เหลือต่อไป ในสุดท้ายค่า start และ end จะกลายเป็นสมาชิกให้แก่ minimal spanning tree เมทริกซ์ และทำซ้ำจนกว่าทุก vertices จะถูกใช้จนหมด

```
#Call to the function:
primsAlgorithm(int(input('Enter the vertices number: ')))
```

เรียกใช้ฟังก์ชัน prim’s Algorithm โดยมีการรับค่าจำนวน vertices จากผู้ใช้และส่งไปเป็น parameter ให้แก่ฟังก์ชัน

การนำผลลัพธ์ที่ได้ มาแก้ปัญหา

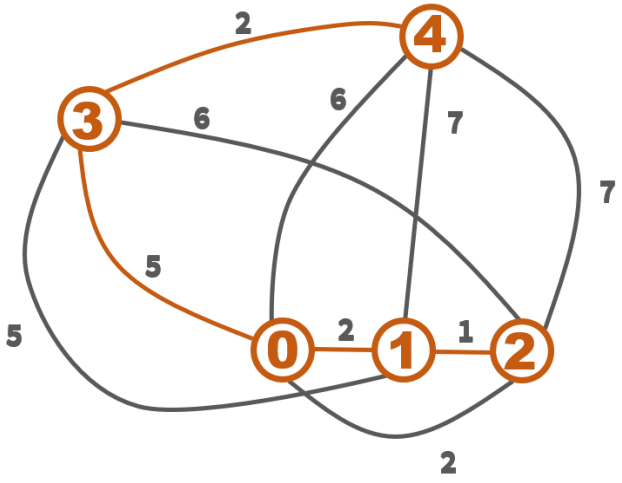
```
In [6]: #เรียกใช้ฟังก์ชั่น
        primsAlgorithm(int(input('Enter the vertices number: ')))

Enter the vertices number: 5
Enter the path weight between the vertices: (0, 0):  0
Enter the path weight between the vertices: (0, 1):  2
Enter the path weight between the vertices: (0, 2):  2
Enter the path weight between the vertices: (0, 3):  5
Enter the path weight between the vertices: (0, 4):  6
Enter the path weight between the vertices: (1, 1):  0
Enter the path weight between the vertices: (1, 2):  1
Enter the path weight between the vertices: (1, 3):  5
Enter the path weight between the vertices: (1, 4):  7
Enter the path weight between the vertices: (2, 2):  0
Enter the path weight between the vertices: (2, 3):  6
Enter the path weight between the vertices: (2, 4):  8
Enter the path weight between the vertices: (3, 3):  0
Enter the path weight between the vertices: (3, 4):  2
Enter the path weight between the vertices: (4, 4):  0
[[0, 2, 0, 5, 0], [2, 0, 1, 0, 0], [0, 1, 0, 0, 0], [5, 0, 0, 0, 2], [0, 0, 0, 2, 0]]
```

จากผลลัพธ์จะเห็นได้ว่าเราจะได้ค่า Minimal Spanning ออกมาในรูปของ list ซึ่งสามารถนำผลลัพธ์ที่ได้นั้นมาแปลงเป็นเมทริกซ์ได้ดังนี้ โดยค่าที่ได้จาก listตำแหน่งที่ 0 มาใส่ในเมทริกซ์แถวที่ 0 ค่าจากlistตำแหน่งที่ 1 มาใส่ในเมทริกซ์แถวที่1 ค่าที่ได้จากlistตำแหน่งที่ 2 มาใส่ในเมทริกซ์แถวที่2 ค่าที่ได้จากlistตำแหน่งที่ 3 มาใส่ในเมทริกซ์แถวที่ 3 และ ค่าที่ได้จาก listตำแหน่งที่ 4 มาใส่ในเมทริกซ์แถวที่ 4 ซึ่งจะได้เมทริกซ์ตามตารางด้านล่างนี้

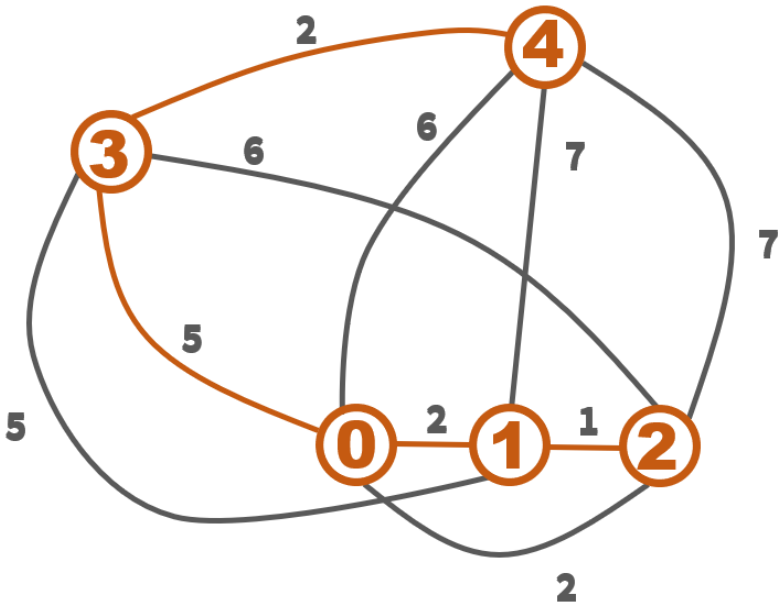
	0	1	2	3	4
0	0	2	0	5	0
1	2	0	1	0	0
2	0	1	0	0	0
3	5	0	0	0	2
4	0	0	0	2	0

ซึ่งจากเมทริกซ์ด้านบนนั้นสามารถหาเวลาที่สั้นที่สุดในการเดินทางภายในราชวิทยาลัยจุฬาภรณ์ได้ ซึ่งเส้นทางการเดินทางจะได้กราฟที่ประหยัดเวลาที่สุดจะได้ดังกราฟด้านล่างนี้



สรุป

เราสามารถหาระยะเวลาการเดินทางภายในราชวิทยาลัยจุฬาภรณ์ที่น้อยที่สุดได้ จากการหา Minimal Spanning Tree โดยที่เราต้องแปลงสถานที่ต่าง ๆ ให้เป็น Vertices ต่าง ๆ ระยะเวลาให้เป็น Edge และนำกราฟที่ได้แปลงเป็นเมทริกซ์ จากนั้นนำ Prim’s Algorithm มาประยุกต์ในการแก้ปัญหาเพื่อที่หาระยะเวลาที่สั้นที่สุดในการเดินทาง ซึ่งจากการแก้ปัญหาพบว่า ถ้าเราเดินทางตามรูปภาพเราจะประหยัดเวลาในการเดินทางภายในราชวิทยาลัยจุฬาภรณ์ได้ โดยเราเดินทางครบทุกสถานที่



อ้างอิง

Arnon Rokprai.2 May 2019.”การหา Shortest Path และ MST โดยใช้ R- Studio #Social Network Analysis”[online system] from https://medium.com/data-growing/(4 Nov 2021)

Dan Nelson.29 June 2021.”Graphs in Python:Minimum Spanning Trees- Prim’ s Algorithm”.[online system] from https://stackabuse.com/graphs-in-python-minimum-spanning-trees-prims-algorithm/ (4 Nov 2021).

Fan page : Algorithm.4 Aug 2010.”Minimum Spanning Tree”[online system] from Minimum Spanning Tree | Design and Analysis of Algorithm (wordpress.com) (4 Nov 2021)