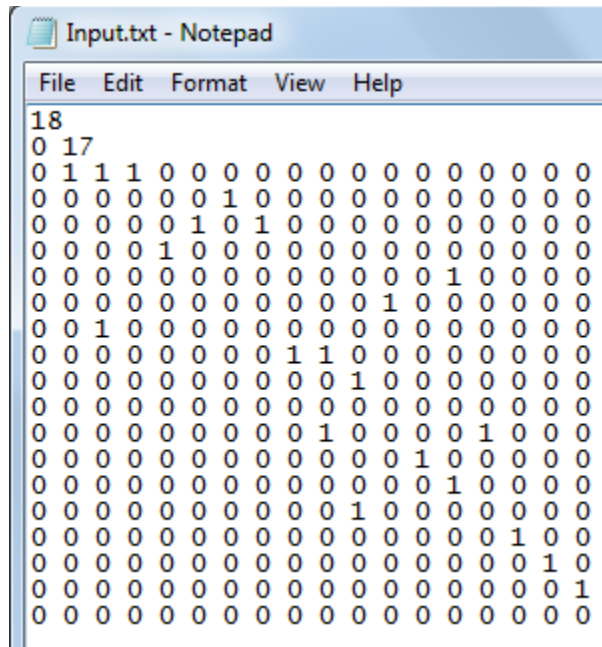


# BÁO CÁO BÀI THỰC HÀNH 1

## 1) Breadth-first Search, Depth-first Search:

### 1) Đọc file đầu vào .txt



Ý nghĩa ma trận: Với  $i$  và  $j$  chạy từ 0 đến 17,  $A_{ij} = 1$  nghĩa là có đường đi từ node  $i$  đến node  $j$ . Ngược lại,  $A_{ij} = 0$

```
2 with open("Input.txt", "r") as file:
3     nodeAmount = int(file.readline())
4     start, end = [int(num) for num in file.readline().split()]
5     adj_matrix = [[int(num) for num in line.split()] for line in file]
```

nodeAmount: Số lượng node của bài toán

start, end: Nod khởi đầu, kết thúc

adj\_matrix: Ma trận kề, kiểu dữ liệu "List của List của Int"

## 2) Ý tưởng Breadth-first Search

```
7   from queue import Queue
8   def bfs(graph, start, end):
9       frontier = Queue()
10      frontier.put([start])
11      explored = []
12      while True:
13          if frontier.empty():
14              raise Exception("No way found Exception")
15          path = frontier.get()
16          current_node = path[-1]
17          #print(type(path), "Current path =", path)
18          #print(type(current_node), "Current node =", current_node, '\n')
19          explored.append(current_node)
20          if current_node == end:
21              print("Solution for BFS:", path)
22              return
23          i = -1
24          for node in graph[current_node]:
25              i = i + 1
26              if (node == 1):
27                  if i not in explored:
28                      new_path = list(path)
29                      new_path.append(i)
30                      frontier.put(new_path)
```

“Uncomment các dòng comment để thấy rõ *path* và *current\_node* trong mỗi lần lặp”

Hàng đợi *frontier* chứa “List của Int” với phần tử đầu tiên là *start*.  
*frontier* có nhiệm vụ chứa các CON ĐƯỜNG cần mở rộng.

Danh sách *explored* để chứa các node đã được mở rộng.

Danh sách *path*, với mỗi lần lặp, sẽ được gán bằng *frontier.get()*, là con đường mà ta đang xét.

Node hiện tại *current\_node* sẽ là *path[-1]*, tức phần tử cuối cùng của *path*. Thêm *current\_node* vào *explored*. Ta bắt đầu xét.

Nếu *current\_node == end*, tức node kết thúc, ta sẽ in ra kết quả là *path*.  
Ý nghĩa ở đây là nếu con đường mà ta đang xét chứa đích đến mà ta hướng đến thì đó là con đường kết quả.

Nếu không, tiến hành mở rộng *current\_node* bằng cách duyệt *graph*, sẽ được gán bằng *adj\_matrix*. Biến đếm *i* để cho biết ta đang ở node thứ bao nhiêu. Nếu *node == 1*, tức có đường đi từ *current\_node* đến *node i*, và *node i* không nằm trong *explored*, khởi tạo *new\_path = list(path)* và thêm *i* vào, sau đó thêm *new\_path* vào *frontier*.

### 3) Ý tưởng Depth-first Search

```
32     from queue import LifoQueue
33     def dfs(graph,start,end):
34         frontier = LifoQueue()
35         frontier.put([start])
36         explored = []
37         while True:
38             if frontier.empty():
39                 raise Exception("No way found Exception")
40             path = frontier.get()
41             current_node = path[-1]
42             #print(type(path), "Current path =", path)
43             #print(type(current_node), "Current node =", current_node, '\n')
44             explored.append(current_node)
45             if current_node == end:
46                 print("Solution for DFS:", path)
47                 return
48             i = 18
49             for node in reversed(graph[current_node]):
50                 i = i - 1
51                 if (node == 1):
52                     if i not in explored:
53                         new_path = list(path)
54                         new_path.append(i)
55                         frontier.put(new_path)
```

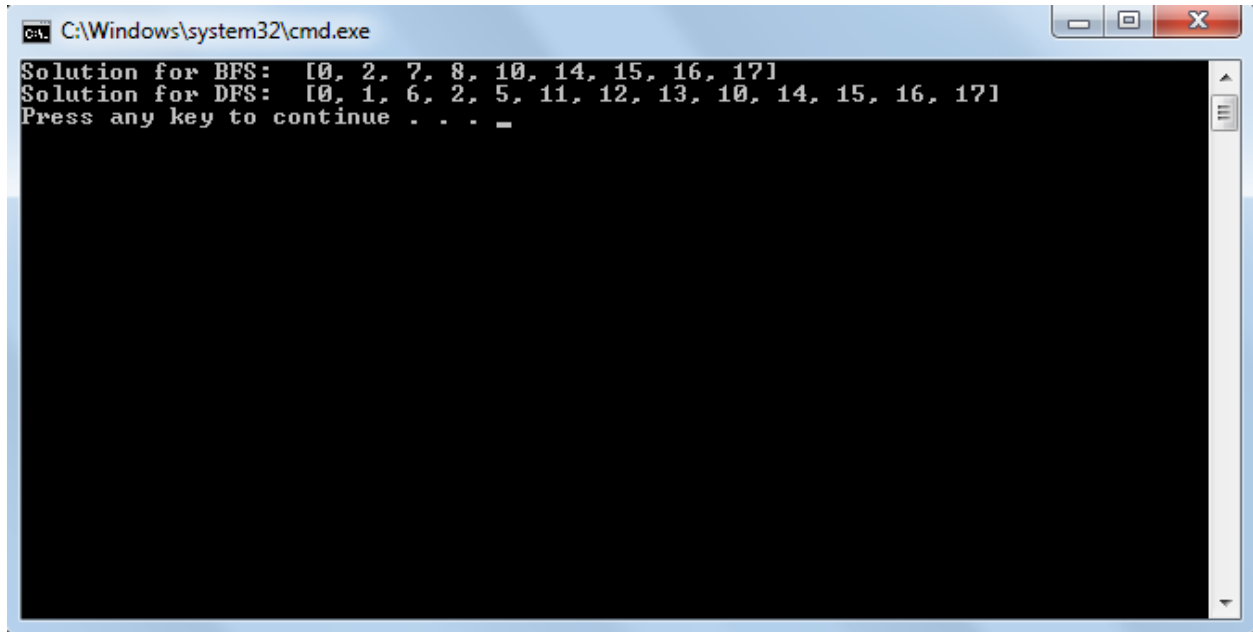
“Uncomment các dòng comment để thấy rõ *path* và *current\_node* trong mỗi lần lặp”

Tương tự BFS, nhưng ở đây *frontier* sẽ là một ngăn xếp. Khi duyệt *graph*, ta sẽ duyệt ngược lại so với BFS do tính chất của ngăn xếp.

Lưu ý: Trong file hướng dẫn, ví dụ minh họa cho thấy nếu 1 mở rộng thành 2 và 3, khi đó thuật toán sẽ ưu tiên mở rộng 3 tiếp theo. Tuy nhiên, cũng với ví dụ đó, thuật toán phía trên sẽ ưu tiên mở rộng 2. Nếu muốn chạy giống như hướng dẫn, thực hiện: thay *i = 18* thành *i = -1*; thay *i = i - 1* thành *i = i + 1*; bỏ *reversed()*.

#### 4) Chạy Code

```
53     bfs(adj_matrix, start, end)
54     dfs(adj_matrix, start, end)
```

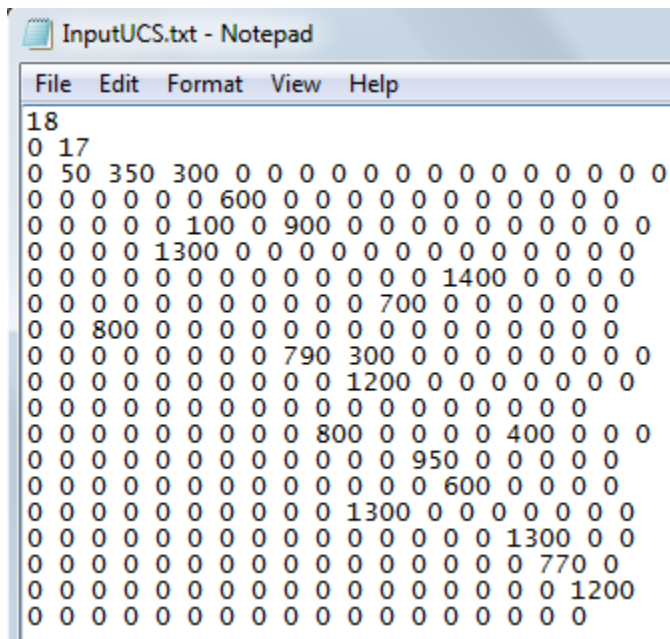


A screenshot of a Windows command prompt window titled "C:\Windows\system32\cmd.exe". The window displays the output of two graph traversal algorithms. The first line shows the solution for BFS: [0, 2, 7, 8, 10, 14, 15, 16, 17]. The second line shows the solution for DFS: [0, 1, 6, 2, 5, 11, 12, 13, 10, 14, 15, 16, 17]. The third line prompts the user to press any key to continue, followed by a series of dots and an underscore.

```
C:\Windows\system32\cmd.exe
Solution for BFS: [0, 2, 7, 8, 10, 14, 15, 16, 17]
Solution for DFS: [0, 1, 6, 2, 5, 11, 12, 13, 10, 14, 15, 16, 17]
Press any key to continue . . . _
```

## 2) Uniform cost Search:

### 1) Đọc file đầu vào .txt



Ý nghĩa ma trận: Với  $i$  và  $j$  chạy từ 0 đến 17,  $A_{ij} = x > 0$  nghĩa là có đường đi từ node  $i$  đến node  $j$  với chi phí là  $x$ . Ngược lại,  $A_{ij} = 0$

```
60 with open("InputUCS.txt", "r") as file:
61     nodeAmount = int(file.readline())
62     start, end = [int(num) for num in file.readline().split()]
63     adj_matrix = [[int(num) for num in line.split()] for line in file]
```

nodeAmount: Số lượng node của bài toán

start, end: Nod khởi đầu, kết thúc

adj\_matrix: Ma trận kề, kiểu dữ liệu "List của List của Int"

## 2) Ý tưởng Uniform cost Search

```
65 from queue import PriorityQueue
66 def ucs(graph, start, end):
67     frontier = PriorityQueue()
68     frontier.put([0, [start]]) # (priority, node)
69     explored = []
70     while True:
71         if frontier.empty():
72             raise Exception("No way found")
73         current_cost, path = frontier.get()
74         current_node = path[-1]
75         #print(type(current_cost), "Current cost =", current_cost)
76         #print(type(path), "Current path =", path)
77         #print(type(current_node), "Current node =", current_node, '\n')
78         explored.append(current_node)
79         if current_node == end:
80             print("Solution for UCS:", path, "- Cost:", current_cost)
81             return
82         i = -1
83         for cost in graph[current_node]:
84             i = i + 1
85             if (cost > 0):
86                 if i not in explored:
87                     new_path = list(path)
88                     new_path.append(i)
89                     frontier.put([current_cost + cost, new_path])
```

“Uncomment các dòng comment để thấy rõ *current\_cost*, *path* và *current\_node* trong mỗi lần lặp”

*frontier* là hàng đợi ưu tiên (PriorityQueue) chứa “List chứa 1 số nguyên là chi phí đường đi (cost) và CON ĐƯỜNG dưới dạng 1 list”.

Ví dụ: [150, [0,2,8]] nghĩa là để đi được đoạn đường từ node 0 đến node 2 đến node 8, chi phí sẽ là 150. Số nguyên đại diện cho chi phí này là yếu tố quyết định vị trí của phần tử trong hàng đợi ưu tiên, số nhỏ hơn xếp trước số lớn hơn.

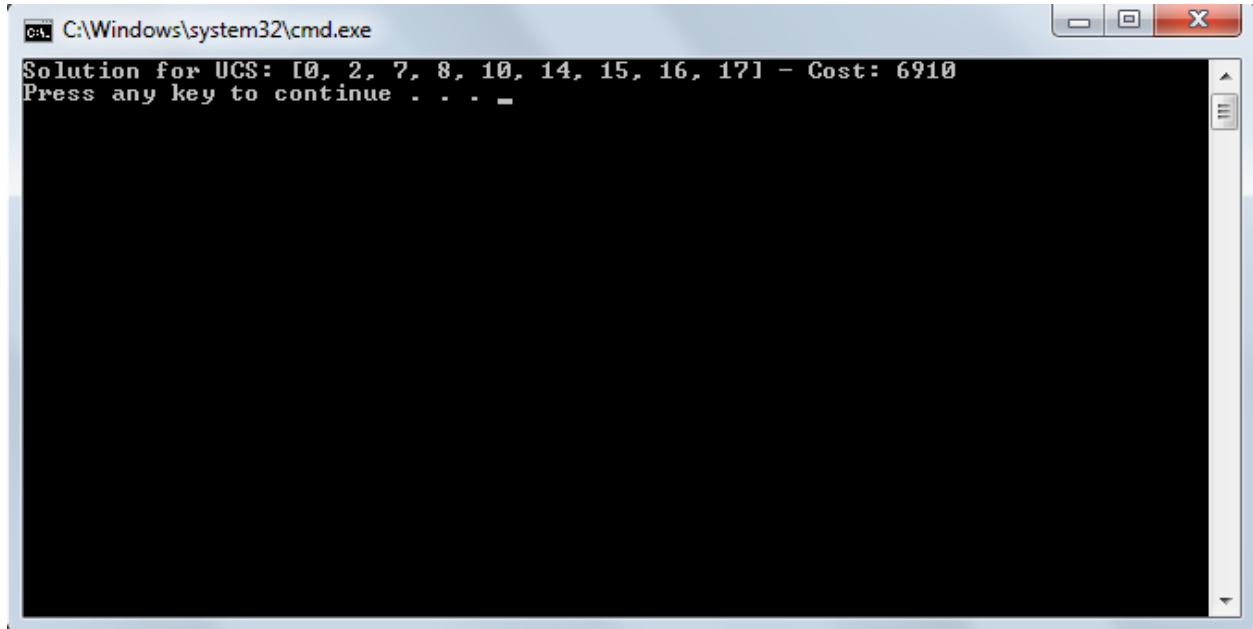
Trong mỗi lần lặp, *frontier.get()* sẽ trả ra chi phí và con đường, được gán đồng thời vào *current\_cost* và *path*. Tương tự BDS và DFS, đặt *current\_node = path[-1]*. Nếu *current\_node == end*, tức node kết thúc, ta sẽ in ra kết quả là *path*.

Graph mà ta truyền vào khi này sẽ chứa chi phí khi đi từ node này đến node khác, với 0 nghĩa là không có đường đi. Do đó biến *node* ở BFS và

DFS sẽ được thay bằng  $cost$  để đúng ý nghĩa. Điều kiện  $cost == 0$  khi này được thay bằng  $cost > 0$ . Khi điều kiện này thỏa, ta tiến hành tạo list  $new\_path$  với biến đếm  $i$  như BFS và DFS, nhưng khi này ta sẽ thêm  $new\_path$  vào  $frontier$  chung với  $current\_cost + cost$  trong một list, với  $current\_cost + cost$  là chi phí của  $new\_path$ .

### 3) Chạy Code

```
91 ucs(adj_matrix, start, end)
```



The screenshot shows a Windows command prompt window titled "C:\Windows\system32\cmd.exe". The output text is as follows:

```
Solution for UCS: [0, 2, 7, 8, 10, 14, 15, 16, 17] - Cost: 6910  
Press any key to continue . . . _
```