

## BÀI 5: CÁC THUẬT TOÁN TÌM KIẾM (tiếp theo)

### I. MỤC TIÊU:

Sau khi thực hành xong, sinh viên nắm được:

- Áp dụng các thuật toán tìm kiếm vào các bài toán thực tế.

### II. TÓM TẮT LÝ THUYẾT:

#### 1. Traveling Salesperson Problem - TSP:

Cho trước  $n$  thành phố và các khoảng cách  $d_{ij}$  giữa mỗi cặp thành phố, tìm tour ngắn nhất sao cho mỗi thành phố được viếng thăm chỉ một lần.

#### 2. Cây khung nhỏ nhất (Minimum Spanning Tree – MST):

- Cho đồ thị  $G = (V, E)$  vô hướng với các cạnh  $d_{ij}$
- Một cây khung  $T$  là một đồ thị con của  $G$  mà nó là
  - 1 cây (đồ thị không tuần hoàn liên thông)
  - Mở rộng tất cả các đỉnh
- Mỗi cây khung có  $(n - 1)$  cạnh
- Chiều dài của mỗi cây khung  $T$  là  $\sum_{(i,j) \in T} d_{ij}$
- Bài toán cây khung nhỏ nhất là tìm 1 cây khung có chiều dài nhỏ nhất.

#### 3. Thuật toán cho MST:

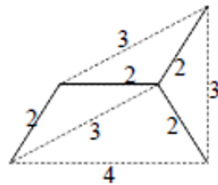
- **Bước 1:** tìm cạnh ngắn nhất trong đồ thị. Nếu có nhiều hơn 1 cạnh như vậy thì chọn 1 ngẫu nhiên 1 cạnh. Đánh dấu cạnh này và các đỉnh được kết nối.
- **Bước 2:** Chọn cạnh ngắn nhất tiếp theo, trừ khi nó tạo thành 1 chu trình với các cạnh đã được đánh dấu trước đó. Đánh dấu cạnh đó và các đỉnh được kết nối.
- **Bước 3:** Nếu tất cả các cạnh được kết nối thì khi đó ta đã hoàn thành. Ngược lại, lặp lại Bước 2.

#### 4. Cây khung nhỏ nhất dựa vào heuristic:

- **Bước 1:** Xây dựng một cây khung nhỏ nhất
- **Bước 2:** Chọn nút gốc là nút bất kỳ.
- **Bước 3:** Duyệt qua tất cả các đỉnh bằng tìm kiếm theo chiều sâu, ghi lại tất cả các đỉnh (đỉnh đã viếng thăm và đỉnh chưa viếng thăm)
- **Bước 4:** Sử dụng chiến lược nhanh chóng trực tiếp hơn để khởi tạo một tour khả thi.

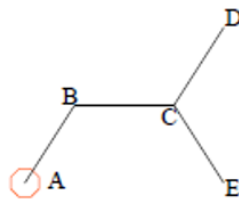
#### 5. Ví dụ:

- **Bước 1:** Xây dựng một cây khung nhỏ nhất

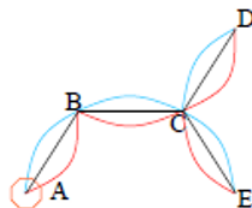


MST có thể được giải trong  $O(n^2)$ , cũng là chặn dưới cho TSP,  $W^* \leq L^*$ .

- **Bước 2:** Chọn nút gốc là 1 nút bất kỳ

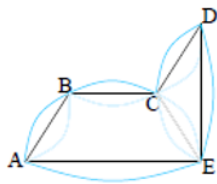


- **Bước 3:** Duyệt qua tất cả các đỉnh



Chuỗi là:  $A - B - C - D - C - E - C - B - A$ , chiều dài của tour là  $2W^*$

- **Bước 4:** Sử dụng chiến lược nhanh chóng trực tiếp hơn để khởi tạo một tour MST  
 $A - B - C - D - (C) - E - (C - B) - A$



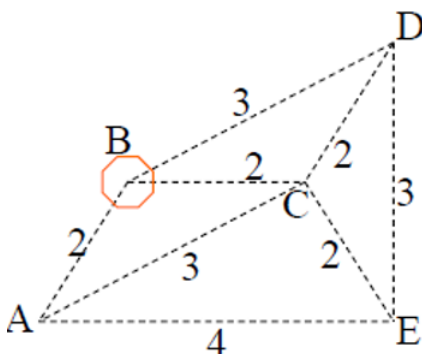
Tour MST là  $A - B - C - D - E - A$ , chiều dài của tour TSP nhỏ hơn bằng  $2W^*$ .

## 6. Heuristic chèn gần nhất:

- **Bước 1:** Chọn 1 nút  $v$  bất kỳ và cho chu trình  $C$  chỉ chứa  $v$
- **Bước 2:** Tìm một nút bên ngoài  $C$  gần nhất với 1 nút trong  $C$ , gọi là  $k$ .
- **Bước 3:** Tìm 1 cạnh  $\{ij\}$  trong  $C$  sao cho  $d_{ik} + d_{kj} - d_{ij}$  là tối thiểu.
- **Bước 4:** Xây dựng một chu trình  $C$  mới bằng việc thay thế  $\{ij\}$  với  $\{ik\}$  và  $\{k, j\}$ .
- **Bước 5:** Nếu chu trình  $C$  hiện tại chứa tất cả các đỉnh thì dừng. Ngược lại, quay lại Bước 2.

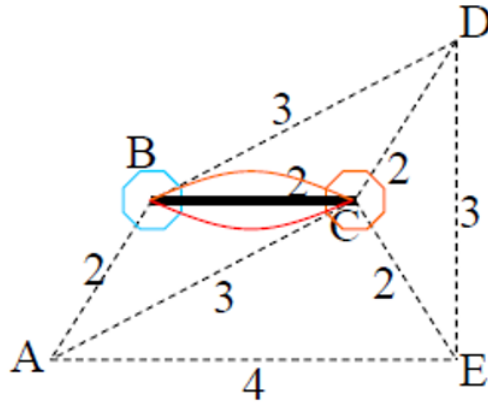
## 7. Ví dụ:

- **Bước 1:** Chọn 1 nút  $v$  bất kỳ và cho chu trình  $C$  chỉ chứa  $v$



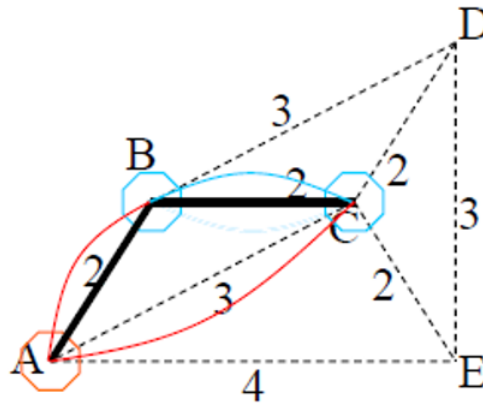
Chiều dài của tour hiện tại là 0.

- **Bước 2 - 3 - 4:** Lần lặp đầu tiên



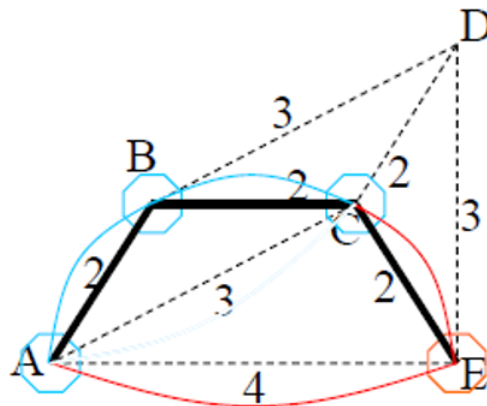
Chiều dài của C không lớn hơn gấp đôi chiều dài của đường in đậm (bằng nhau trong lần lặp này).

Lần lặp thứ 2:



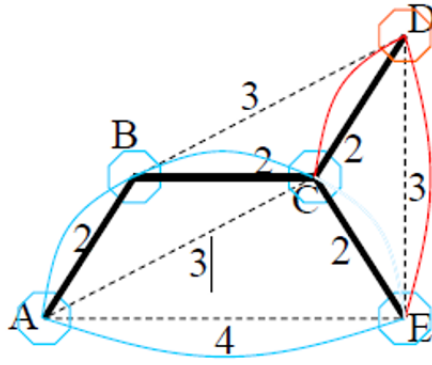
chiều dài của tour C hiện tại là không lớn hơn gấp đôi chiều dài đường in đậm.

Lần lặp thứ ba:



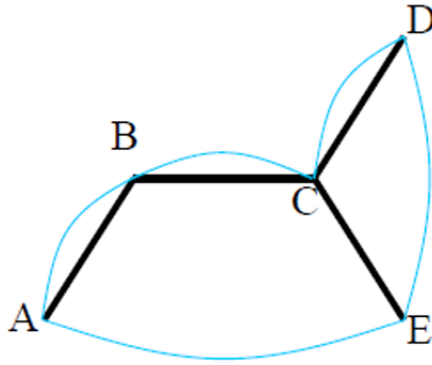
Chiều dài của tour C hiện tại không lớn hơn lần chiều dài đường in đậm.

Lần lặp thứ 4:



Chiều dài của tour C hiện tại không lớn hơn lần chiều dài đường in đậm.

- **Bước 5:** Kết quả cuối cùng là  $A - B - C - D - E - A$



Chiều dài của tour C hiện tại không lớn hơn lần chiều dài đường in đậm.

8. Thuật toán  $A^*$  cho việc giải bài toán TSP:

- **Trạng thái ban đầu:** Agent ở thành phố bắt đầu và không viếng thăm bất kỳ thành phố nào khác.
- **Trạng thái kết thúc:** Agent đã viếng thăm tất cả các thành phố và đến thành phố bắt đầu 1 lần nữa.
- **Hàm successor:** khởi tạo tất cả các thành phố chưa viếng thăm.
- **Chi phí cạnh:** khoảng cách giữa các thành phố được biểu diễn bởi các nút, sử dụng chi phí này để tính  $g(n)$ .

- $h(n)$ : khoảng cách tối thành phố chưa viếng thăm gần nhất ước lượng khoảng cách đi từ tất cả thành phố bắt đầu.

### III. NỘI DUNG THỰC HÀNH:

#### 1. Bài toán:

Sử dụng thuật toán  $A^*$  để giải bài toán TSP và heuristic được sử dụng là cây khung nhỏ nhất.

#### 2. Cài đặt:

```

1  from treelib import Node, Tree
2  import sys
3
4  # Structure to represent tree nodes in the A* expansion
5  class TreeNode(object):
6      def __init__(self, c_no, c_id, f_value, h_value, parent_id):
7          self.c_no = c_no
8          self.c_id = c_id
9          self.f_value = f_value
10         self.h_value = h_value
11         self.parent_id = parent_id
12
13     # Structure to represent fringe nodes in the A* fringe list
14     class FringeNode(object):
15         def __init__(self, c_no, f_value):
16             self.f_value = f_value
17             self.c_no = c_no
18
19
20     class Graph():
21
22         def __init__(self, vertices):
23             self.V = vertices
24             self.graph = [[0 for column in range(vertices)]
25                           for row in range(vertices)]
26
27         # A utility function to print the constructed MST stored in parent[]
28         def printMST(self, parent, d_temp, t):
29             #print("Edge \tWeight")
30             sum_weight = 0
31             min1 = 10000
32             min2 = 10000
33             r_temp = {} #Reverse dictionary
34             for k in d_temp:
35                 r_temp[d_temp[k]] = k
36
37             for i in range(1, self.V):
38                 #print(parent[i], "-", i, "\t", self.graph[i][parent[i]])
39                 sum_weight = sum_weight + self.graph[i][parent[i]]
40                 if (graph[0][r_temp[i]] < min1):
41                     min1 = graph[0][r_temp[i]]
42                 if (graph[0][r_temp[parent[i]]] < min1):
43                     min1 = graph[0][r_temp[parent[i]]]
44                 if (graph[t][r_temp[i]] < min2):

```

```

45         min2 = graph[t][r_temp[i]]
46         if graph[t][r_temp[parent[i]]] < min2:
47             min2 = graph[t][r_temp[parent[i]]]
48
49     return (sum_weight + min1 + min2)*10000
50
51
52     # A utility function to find the vertex with
53     # minimum distance value, from the set of vertices
54     # not yet included in shortest path tree
55     def minKey(self, key, mstSet):
56
57         # Initilaize min value
58         min = sys.maxsize
59
60         for v in range(self.V):
61             if key[v] < min and mstSet[v] == False:
62                 min = key[v]
63                 min_index = v
64
65         return min_index
66
67     # Function to construct and print MST for a graph
68     # represented using adjacency matrix representation
69     def primMST(self, d_temp, t):
70
71         # Key values used to pick minimum weight edge in cut
72         key = [sys.maxsize] * self.V
73         parent = [None] * self.V # Array to store constructed MST
74         # Make key 0 so that this vertex is picked as first vertex
75         key[0] = 0
76         mstSet = [False] * self.V
77         sum_weight = 10000
78         parent[0] = -1 # First node is always the root of
79
80         for c in range(self.V):
81
82             # Pick the minimum distance vertex from the set of vertices not yet processed.
83             # u is always equal to src in first iteration
84             u = self.minKey(key, mstSet)
85
86             # Put the minimum distance vertex in the shortest path tree
87             mstSet[u] = True
88
89             # Update dist value of the adjacent vertices of the picked vertex only if the
90             # current distance is greater than new distance and
91             # the vertex is not in the shortest path tree
92             for v in range(self.V):
93                 # graph[u][v] is non zero only for adjacent vertices of u
94                 # mstSet[v] is false for vertices not yet included in MST
95                 # Update the key only if graph[u][v] is smaller than key[v]
96                 if self.graph[u][v] > 0 and mstSet[v] == False and key[v] > self.graph[u][v]:
97                     key[v] = self.graph[u][v]
98                     parent[v] = u
99
100         return self.printMST(parent, d_temp, t)
101
102
103     # Idea here is to form a graph of all unvisited nodes and make MST from that.
104     # Determine weight of that mst and connect it with the visited node and 0th node
105     # Prim's Algorithm used for MST (Greedy approach)
106     def heuristic(tree, p_id, t, V, graph):
107         visited = set() # Set to store visited nodes
108         visited.add(0)
109         visited.add(t)
110         if p_id != -1:
111             tnode = tree.get_node(str(p_id))
112             # Find all visited nodes and add them to the set
113             while (tnode.data.c_id != 1):
114                 visited.add(tnode.data.c_no)
115                 tnode = tree.get_node(str(tnode.data.parent_id))
116         l = len(visited)
117         num = V - l # No of unvisited nodes
118         if (num != 0):
119             g = Graph(num)
120             d_temp = {}
121             key = 0
122             # d_temp dictionary stores mappings of original city no as (key) and
123             # new sequential no as value for MST to work
124             for i in range(V):
125                 if i not in visited:
126                     d_temp[i] = key
127                     key = key + 1
128
129             i = 0
130             for i in range(V):
131                 for j in range(V):
132                     if ((i not in visited) and (j not in visited)):

```

```

133         g.graph[d_temp[i]][d_temp[j]] = graph[i][j]
134
135     #print(g.graph)
136     mst_weight = g.primMST(d_temp, t)
137     return mst_weight
138 else:
139     return graph[t][0]
140
141
142 def checkPath(tree, toExpand, V):
143     tnode=tree.get_node(str(toExpand.c_id)) # Get the node to expand from the tree
144     list1 = list() # List to store the path
145     # For 1st node
146     if(tnode.data.c_id == 1):
147         #print("In If")
148         return 0
149     else:
150         #print("In else")
151         depth = tree.depth(tnode) # Check depth of the tree
152         s = set() # Set to store nodes in the path
153         # Go up in the tree using the parent pointer and add all
154         # nodes in the way to the set and list
155         while(tnode.data.c_id != 1):
156             s.add(tnode.data.c_no)
157             list1.append(tnode.data.c_no)
158             tnode=tree.get_node(str(tnode.data.parent_id))
159         list1.append(0)
160         if(depth == V and len(s) == V and list1[0]==0):
161             print("Path complete")
162             list1.reverse()
163             print(list1)
164             return 1
165         else:
166             return 0
167
168 def startTSP(graph,tree,V):
169     goalState = 0
170     times = 0
171     toExpand = TreeNode(0,0,0,0,0) # Node to expand
172     key = 1 # Unique Identifier for a node in the tree
173     heu = heuristic(tree,-1,0,V,graph) # Heuristic for node 0 in the tree
174     tree.create_node("1", "1", data=TreeNode(0,1,heu,heu,-1)) # Create 1st node in the tree i.e. 0th city
175     fringe_list = {} # Fringe List(Dictionary) (FL)
176     fringe_list[key] = FringeNode(0, heu) # Adding 1st node in FL
177     key = key + 1
178     while(goalState == 0):
179         minf = sys.maxsize
180         # Pick node having min f_value from the fringe list
181         for i in fringe_list.keys():
182             if(fringe_list[i].f_value < minf):
183                 toExpand.f_value = fringe_list[i].f_value
184                 toExpand.c_no = fringe_list[i].c_no
185                 toExpand.c_id = i
186                 minf = fringe_list[i].f_value
187
188         h = tree.get_node(str(toExpand.c_id)).data.h_value # Heuristic value of selected node
189         val=toExpand.f_value - h # g value of selected node
190         path = checkPath(tree, toExpand, V) # Check path of selected node if it is complete or not
191         # If node to expand is 0 and path is complete, we are done
192         # We check node at the time of expansion and not at the time of generation
193         if(toExpand.c_no==0 and path==1):
194             goalState=1;
195             cost=toExpand.f_value # Total actual cost incurred
196         else:
197             del fringe_list[toExpand.c_id] # Remove node from FL
198             j=0
199             # Evaluate f_values and h_values of adjacent nodes of the node to expand
200             while(j<V):
201                 if(j!=toExpand.c_no):
202                     h = heuristic(tree, toExpand.c_id, j, V, graph) # Heuristic calc
203                     f_val = val + graph[j][toExpand.c_no] + h # g(parent) + g(parent->child) + h(child)
204                     fringe_list[key] = FringeNode(j, f_val)
205                     tree.create_node(str(toExpand.c_no), str(key),parent=str(toExpand.c_id), \
206                                     data=TreeNode(j,key,f_val,h,toExpand.c_id))
207                     key = key + 1
208             j=j+1
209         return cost
210 if __name__ == '__main__':
211     V=4
212     graph=[[0,5,2,3],[5,0,6,3],[2,6,0,4],[3,3,4,0]]
213
214     tree = Tree()
215     ans = startTSP(graph,tree,V)
216     print("Ans is "+str(ans))
217

```



### 3. Yêu cầu:

- Cài đặt và thực thi chương trình. Nếu chương trình bị báo lỗi thì lỗi ở dòng nào và sửa lại như thế nào?
- Viết báo cáo trình bày lại tất cả những gì em hiểu liên quan tới bài thực hành.  
Nhận xét?