COMP3711 Assignment 4

TANUWIJAYA, Randy Stefan
(20582731)
rstanuwijaya@connect.ust.hk

Department of Physics - HKUST
Department of Computer Science and Engineering - HKUST

April 10, 2021

Problem 1: MSTs

Let $G = (V, E)$ be a weighted undirected connected graph inputted as an adjacency list. Assume that all edges have distinct weights so the *Minimum Spanning Tree* T is unique and that you have already run a MST algorithm that has found and stored T.

Now arbitrarily increase the weight of any one edge $e = (u, v)$ in E. You should assume that the edges in the graph still all have distinct weights, i.e, the new weight is different than the weights of all of the other edges.

Let T' be the MST of the graph with the new weights. i.e., after replacing $w(u, v)$ with its new weight $w'(u, v)$.

(a) Prove that either $T' = T$ or that T and T' differ by at most one edge.

Equivalent to proof: if $e' \neq e$ was an edge in MST before e was changed, then e' will also be in the MST after the weight is changed.

If $e \notin T$, then $T = T'$ as increasing e does not affect the MST.

If $e = (u, v) \in T$, then we can separate V into S and $V - S$, where $u \in S$ and $v \in V - S$, then T' and T differ by at most only one edge, e.

Proof:

- Consider running Prim's Algorithm for S and $V - S$ to build each respective MST separately. By assumption, the weight of each edge is unique, then the MST is unique.
- Suppose we run the Prim's algorithm for the subset S, and obtained the MST for S, namely T_S. By repeatedly applying cut lemma, we can expand the MST of S, T_S to obtain the MST of V, both before and after the weight increase of e, T and T'. Without loss of generality, we can also find the MST for $V - S$, T_{S-V} is also in T and T'.
- In other words, we can split T into three trees, which are T_S, e, and T_{V-S}. and T' into T_S, e^*, and T_{V-S}, where e^* can be the same as e or another edge connecting S and $V - S$. Then, union the MSTs is $MST(S) \cup MST(V - S) = T - e = T' - e^*$.
- In either cases, T_S and T_{V-S} is in T'
- Therefore, at most only one node, e will be changed.

(b) Give an $O(|E|)$ time algorithm for finding T':

After describing your algorithm, you must then explicitly justify why your algorithm is correct and why it runs in $O(|E|)$ time.

(1) Disconnect the edge $e = (u, v)$ from T, where that $v.p = u$. There are two separate trees $u \in T_S$ and $v \in T_{V-S}$, and name the subsets of the nodes in one of the trees as S and $V - S$.
(2) Suppose repeatedly checking the parents of u would return the *root*, r, ie. $r.p = r$. Set $v.p \leftarrow v$. Such that repeatedly checking the parents of the items in S would return r and the items in $V - S$ would return v. Preprocess the nodes by categorizing the subset of each node by using dynamic programming to check the parent of each node, i.e. the parent of nodes in S is r and the parent of nodes in $V - S$ is v. By using stack data structure and dynamic programming, each node will be processed only once, thus the running time is $O(V)$ time.
(3) For each edges (u, v), check if $u \in S$ and $v \in\in V - S$. If so, store the minimum value and the corresponding edge connecting S and $V - S$. Suppose the minimum edge connecting S and $V - S$ is e' after checking all the edges. This procedure checks the edges connecting S and $T - S$ once only, therefore the running time is $O(E)$
(4) Connect back T_S and T_{V-S} with e', return it as T'.

Running time analysis: step (1) takes $O(1)$ amount of time, step (2) takes $O(V)$ amount of time, and step (3) takes $O(E)$ amount of time and step (4) takes $O(1)$ amount of time. Therefore, the total running time is $O(E + V) = O(E)$ because $|E| \geq |V| + 1$.

Problem 2: Graph Algorithms

Modified from [KT] Chapter 4.

Security analysts are trying to track the spread of an online virus in a collection of networked computers.

- There are n computers in the collection, labelled as $C_1, \ldots, C_n$.
- The input to the problem is a collection of m *trace-data* triples.
- Each triple is of the form $(C_{j_i}, C_{k_i}, t_i) = 1, \ldots, m$ where $j_i \neq k_i$. A triple indicates that computer C_{j_i} and computer C_{k_i} exchanged bits at time t_i.
- You may assume that the triples are sorted in non-decreasing order of time, i.e., $t_i \leq t_{i+1}$

The analysts would like to be able to answer the following type of question: *If the virus was inserted into computer C_a at time x could it possibly have infected computer C_b by time y?*

The mechanics of infection are that: if an infected computer C_i exchanges bits with computer C_j at time t (i.e., either (C_i, C_j, t) or (C_j, C_i, t) appear in the trace data) then C_j becomes immediately infected.

This means that infection spreads from one computer to another through a sequence of communications that occur in non-decreasing order of time. The input to your algorithm also includes the extra information that a specific computer C_a has been infected at time t_{start}

Describe an $O(m+n)$ time algorithm which determines, for every other computer, the earliest time at which it can become infected.

More specifically, your algorithm should determine the set C of computers that can be infected by a sequence of communications starting at C_a on or after time t_{start} and then create a directed infection tree T for C.

- The nodes of T are the computers in C. The root of T is C_a.
- T is represented by two arrays $P[1 \ldots n]$ and $Time[1 \ldots n]$.
- $Time[i]$ should be the earliest time that computer C_i can be infected
- Each edge $C_i \rightarrow C_j$ in the tree corresponds to one existing trace-data triple (C_i, C_j, t) or (C_j, C_i, t). $P[j] = i$ and $Time[j] = t$.
- Note that $P[a]$ is undefined and $Time[a] = t_{start}$.
  If $C_i \notin C$ then $P[i] = Time[i] = \infty$.
- A consequence of the definitions is that $Time[i] \geq Time[P[i]]$ for all $i \neq a$.

(a) (i) Give documented pseudocode for creating T and the associated arrays.

Algorithm 1: Infect(n, $trace$, C_a, t_{start})

```
   // Initialize the arrays
 1 let V[1...n] as an array containing [1...n]
 2 let u.color ← white for all u in V ;                    // white for uninfected
 3 let P[1...n] as an array of all ∞
 4 let Time[1...n] as an array of all ∞
   // set the first infection
 5 C_a.color ← red
 6 P[C_a] = nil, Time[C_a] = t_start
 7 E = [], t = t_start
 8 foreach m in trace do
       // suppose m[0,1] is the edge and m[2] is the time, skip until the
          t_start
 9     if m[2] < t_start then
10         continue
11     end
       // end of a sequence of trace having the same time
12     if m[2] ≠ t then
           // run DFS
13         DFS(E)
           // reset E and t
14         t = m[2]
15         E = []
16     end
17     append (m[0], m[1]) to E
18 end
   // run DFS one last time
19 DFS(E)
20 return P[1 ... n], Time[1 ... n]
```

Algorithm 2: DFS(E)

```
 1 Build(E) ; // initialize adj[1..n] as empty linked lists with size of
     n and u.visited ← false for all u ∈ V with running time of O(E)
 2 foreach (u, v) in E do
 3     append v to adj[u]
 4     append u to adj[v]
 5 end
   // call DFS if to spread infection, if a node has not been visited
      previously
 6 foreach (u, v) in E do
 7     if u.color = red and u.visited = false then
 8         DFS-Visit(u)
 9     end
10     if v.color = red and v.visited = false then
11         DFS-Visit(v)
12     end
13 end
```

Algorithm 3: DFS-Visit(u)

```
  // Spread the virus infection to node u
1 u.visited ← true
2 u.color ← red
3 for v in adj[u] do
4 |   if v.color = white then
5 |   |   P[v] ← u
6 |   |   Time[v] ← t
7 |   |   DFS-Visit(v)
8 |   end
9 end
```

(ii) Explain what your code does.

The code starts by initializing the $V[1..n]$ by it's label, $P[1..n]$ by all nil, and $Time[1 \dots n]$ by all ∞. It also label if a computer has been infected or not by using a *color* label, where *red* represent if a computer is already infected and *white* represent if a computer is not infected yet.

Then, it initialize the virus infection by setting the color of C_a, the origin of the virus, as *red*, and setting the initial infection time of $Time[C_a]$, as t_{start}.

Define E as an linked list of all edges with the same bit exchange time, and t as the last *trace* time processed in the following for loop.

Begin the for loop. As the *trace* is sorted by the bit-exchange time non-descending, we can skip the first few traces before the first infection time t_{start}. Then, there would be two cases:

- If the trace time is the same as the previous one, we simply append the edges to E
- If the trace time is different than the previous one, we run DFS algorithm to see how the virus infection spreads within the tree with the edges E by running the DFS algorithm to update the infection, and fill the P and $Time$ arrays. Then we reset the t to be last trace time, and clear the E.

Then we run the DFS for one last time, as E might not be empty. Finally, return the P and $Time$ arrays.

(b) Prove that your algorithm is correct.

Please remember to be clear.

Put every new idea and short argument in a separate paragraph with sufficient space between the paragraphs. We can't understand what you've written if we can't read it.

The algorithm will check all traces starting from the least time. If some traces shares the same time, the edges will be checked together using the DFS algorithm. The DFS will visit the adjacent nodes if the node is already infected and have not been visited previously in the current DFS call (line 8 and 11 Algorithm 2).

The parent of each node $P[1..n]$ is correct, but not unique. **Proof:** Note that the *trace* is sorted by the bit-exchange time in non-decreasing order. For the group of *trace* with the same time, the algorithm use DFS to track how the infection spreads and set the parent of a node as the other node in the edge where the infection spreads from. In case there are multiple infected nodes connected to a single uninfected node, then the parent of the uninfected node would be chosen by the order of the traces, but choosing the parent of the uninfected node to be other infected node is also correct. Thus, the parent is not unique. Therefore, the algorithm return a correct $P[1..n]$ parent array by considering the correctness of DFS algorithm.

The earliest infection time of each node $Time[1..n]$ is correct and unique. **Proof:** Note that the *trace* is sorted by the bit-exchange time in non-decreasing order. For the group of *trace* with the same time, the algorithm use DFS to track how the infection spreads and set the parent of a node as the other node in the edge where the infection spreads from. Therefore, the algorithm return a correct $Time[1..n]$ parent array. The $Time[1..n]$ array is unique as there must be only one earliest infection time for each node. Therefore the $Time[1..n]$ is correct as each edges is processed order of by the non-decreasing time of the traces, i.e. by uniqueness of earliest infection time, node that is infected earlier must be processed before the node that is infected later, and nodes with the same time will be processed within the same DFS tree.

Therefore, the algorithm return the correct parent array $P[1..n]$ and earliest infection time array $Time[1..n]$

(c) Explain why your algorithm runs in $O(m + n)$ time.

The initialization of Infect() procedure takes $O(n)$.

The most outer for loop in line 10 of Infect algorithm runs for all m, and call the DFS algorithm. The DFS algorithm will check for all E and run DFS Visit at most once for all nodes in E (Build(E) procedure takes O(E) time). The DFS Visit will visit a node if only the node is never visited previously in the current DFS call. In other word, the running time of DFS(V, E) is $O(|E|)$, as the 'visiting' (line 5-7 of DFS-Visit(u) algorithm) will happen at most once for each node. Suppose E_i is the list of edges in the i-th call of DFS algorithm, then $\sum |E_i| \leq m$, as the $|E_i|$ will be cleared everytime DFS is run.

Therefore the total running time is $O(n + \sum |E_i|) = O(n + m)$

Problem 3: BFS and DFS

In this problem you will have to describe the breadth and depth first search trees calculated for particular graphs.

(A) Consider the following graph $G = (V, E)$ with 16 vertices

$V = \{u_0, \ldots, u_7\} \cup \{v_0, \ldots, v_7\}$ and the edges E as given by the following adjacency lists:

Each node u_i in $\{u_0, \ldots, u_7\}$ is connected, in the following order, to its four neighbors: $u_{(i+1) \mod 8}$, $u_{(i-1) \mod 8}$, $v_{(i+1) \mod 8}$, $v_{(i-1) \mod 8}$.

Each node v_i in $\{v_0, \ldots, v_7\}$ is connected, in the following order, to its four neighbors: $u_{(i+1) \mod 8}$, $u_{(i-1) \mod 8}$, $v_{(i+3) \mod 8}$, $v_{(i-3) \mod 8}$.

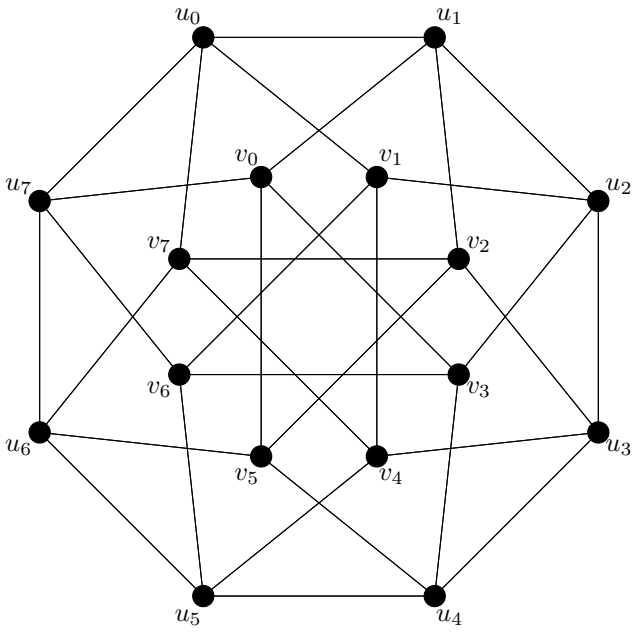


Figure 3.1: Graph with all edges

(i) Recall that BFS is *Breadth First Search*. Draw the edges of the BFS tree of G that results when starting from the node $\mathbf{u_1}$, with G represented using the adjacency lists described above.

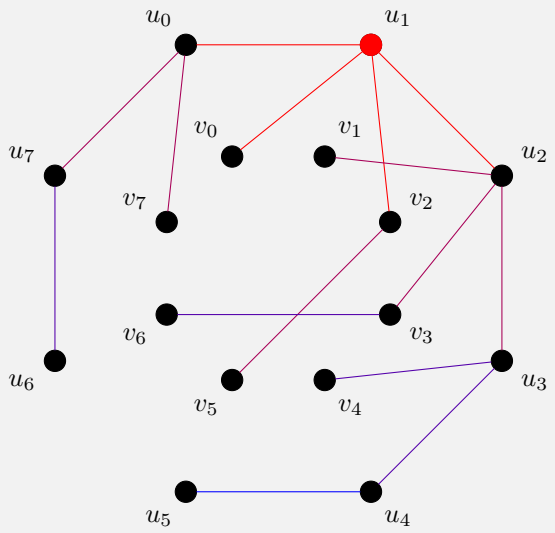


Figure 3.2: BFS Tree

(ii) Recall that DFS is *Depth First Search*. Draw the edges of the BFS tree of G that results when starting from the node $\mathbf{u_1}$, with G represented using the adjacency lists described above.

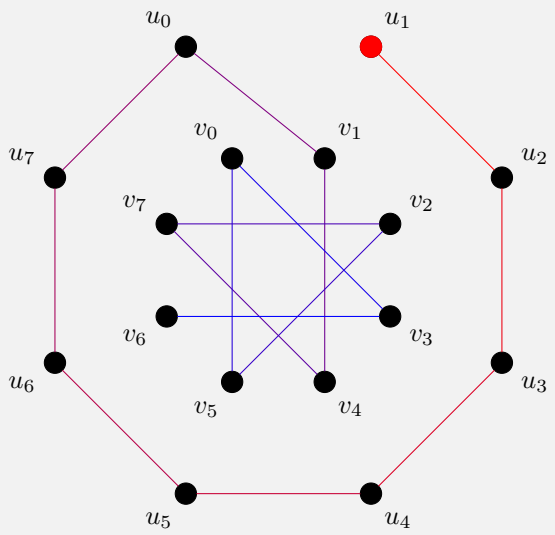


Figure 3.3: DFS Tree

(B) Consider the same graph $G = (V, E)$ shown above and the same adjacency lists as illustrated in (A).

(i) Recall that BFS is *Breadth First Search*. Draw the edges of the BFS tree of G that results when starting from the node $\mathbf{v_1}$, with G represented using the adjacency lists described above.

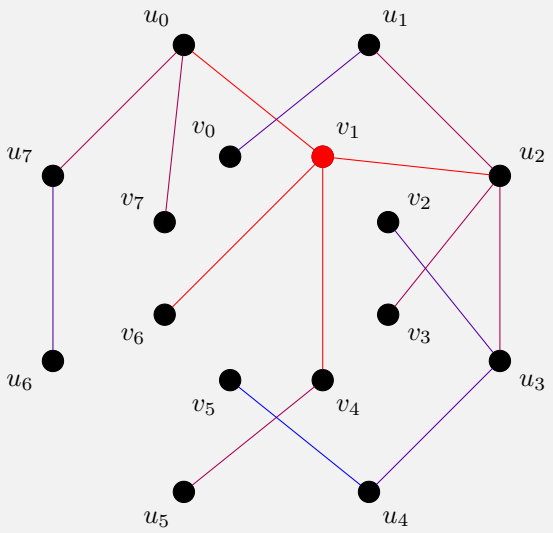


Figure 3.4: BFS Tree

(ii) Recall that DFS is *Depth First Search*. Draw the edges of the BFS tree of G that results when starting from the node $\mathbf{v_1}$, with G represented using the adjacency lists described above.

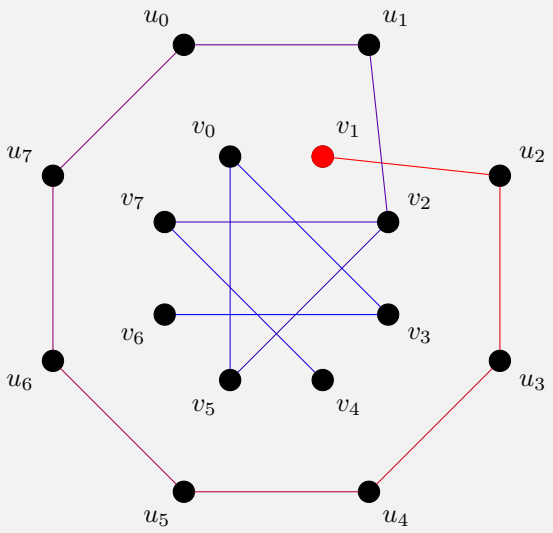


Figure 3.5: DFS Tree

Problem 4: Find Minimum Spanning Tree (MST) and Shortest Path

The tourist map of Venice can be modeled as a connected, edge-weighted undirected graph with nodes being its major attractions and edges being tourist routes connecting two attractions.

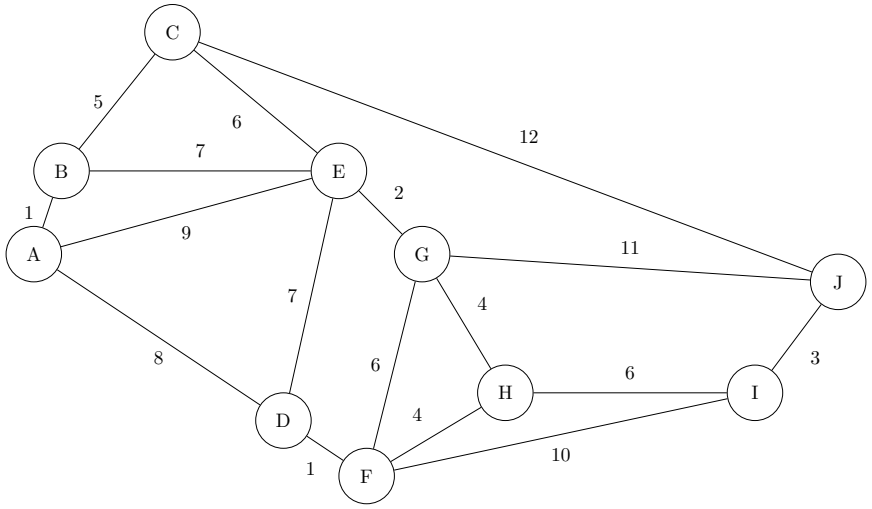


Figure 4.1: Undirected graph of the map

(A) The following graph depicts the tourist routes and their corresponding distance in terms of kilometers among 10 major Venice attractions shown in the following figures.

(a) Run **Prim's** algorithm taught in 16_MST on the graph, *starting from vertex A*, to find G's MST. (This minimizes the total distance covered, in kilometers, of the spanning tree.). provide the following information for each step:

(1) S that keeps the current explored nodes; initially $S = \{A\}$
(2) T that keeps the current selected MST edges; initially $T = \{\}$
(3) A table that tracks the key value and parent of each node.
Also, upon completion of the algorithm, please:
(4) draw the final MST, and
(5) provide the minimum total route length (in kilometers) achieved by this MST.

(1) Step 0:

$$S = \{A\}$$
$$T = \{\}$$

Node	A	B	C	D	E	F	G	H	I	J
Key	0	1	∞	8	9	∞	∞	∞	∞	∞
Parent	-	A	-	A	A	-	-	-	-	-

(2) Step 1:

$$S = \{A, B\}$$
$$T = \{(A, B)\}$$

Node	A	B	C	D	E	F	G	H	I	J
Key	0	1	5	8	7	∞	∞	∞	∞	∞
Parent	-	A	B	A	B	-	-	-	-	-

(3) Step 2:

$$S = \{A, B, C\}$$

$$T = \{(A, B), (B, C)\}$$

Node	A	B	C	D	E	F	G	H	I	J
Key	0	1	5	8	6	∞	∞	∞	∞	12
Parent	-	A	B	A	C	-	-	-	-	C

(4) Step 3:

$$S = \{A, B, C, E\}$$

$$T = \{(A, B), (B, C), (C, E)\}$$

Node	A	B	C	D	E	F	G	H	I	J
Key	0	1	5	7	6	∞	2	∞	∞	12
Parent	-	A	B	E	C	-	E	-	-	C

(5) Step 4:

$$S = \{A, B, C, E, G\}$$

$$T = \{(A, B), (B, C), (C, E), (E, G)\}$$

Node	A	B	C	D	E	F	G	H	I	J
Key	0	1	5	7	6	6	2	4	∞	11
Parent	-	A	B	E	C	G	E	G	-	G

(6) Step 5:

$$S = \{A, B, C, E, G, H\}$$

$$T = \{(A, B), (B, C), (C, E), (E, G), (G, H)\}$$

Node	A	B	C	D	E	F	G	H	I	J
Key	0	1	5	7	6	4	2	4	6	11
Parent	-	A	B	E	C	H	E	G	H	G

(7) Step 6:

$$S = \{A, B, C, E, G, H, F\}$$

$$T = \{(A, B), (B, C), (C, E), (E, G), (G, H), (H, F)\}$$

Node	A	B	C	D	E	F	G	H	I	J
Key	0	1	5	1	6	4	2	4	6	11
Parent	-	A	B	F	C	H	E	G	H	G

(8) Step 7:

$$S = \{A, B, C, E, G, H, F, D\}$$

$$T = \{(A, B), (B, C), (C, E), (E, G), (G, H), (H, F), (F, D)\}$$

Node	A	B	C	D	E	F	G	H	I	J
Key	0	1	5	1	6	4	2	4	6	11
Parent	-	A	B	F	C	H	E	G	H	G

(9) Step 8:

$$S = \{A, B, C, E, G, H, F, D, I\}$$

$$T = \{(A, B), (B, C), (C, E), (E, G), (G, H), (H, F), (F, D), (H, I)\}$$

Node	A	B	C	D	E	F	G	H	I	J
Key	0	1	5	1	6	4	2	4	6	3
Parent	-	A	B	F	C	H	E	G	H	I

(10) Step 9:

$$S = \{A, B, C, E, G, H, F, D, I, J\}$$

$$T = \{(A, B), (B, C), (C, E), (E, G), (G, H), (H, F), (F, D), (H, I), (I, J)\}$$

Node	A	B	C	D	E	F	G	H	I	J
Key	0	1	5	1	6	4	2	4	6	3
Parent	-	A	B	F	C	H	E	G	H	I

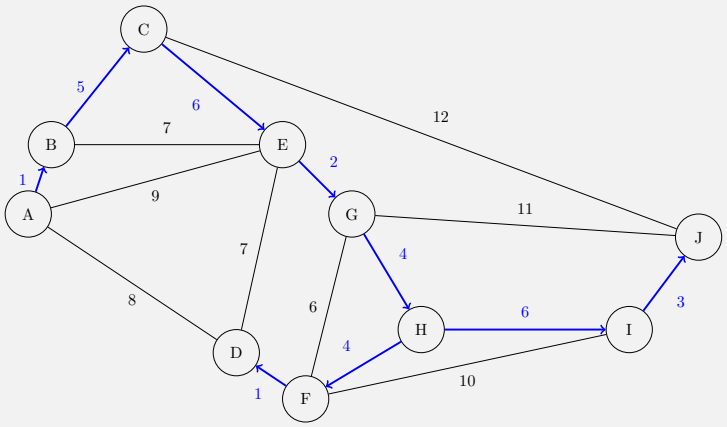


Figure 4.2: Prim's Minimum Spanning Tree, with cost of 32

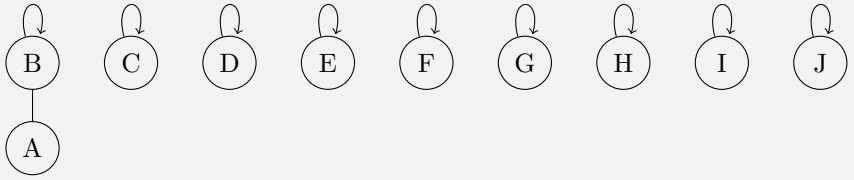
(b) Run **Kruskal's** algorithm taught in 16 MST on the same graph to again find its MST. To answer this question, you need to follow the example in 16c_Kruskal_UF and provide the following information for each step:

(1) The current edge that is being tested, and the decision *(i.e., "YES" if this edge is added to the MST, "NO" otherwise)*.

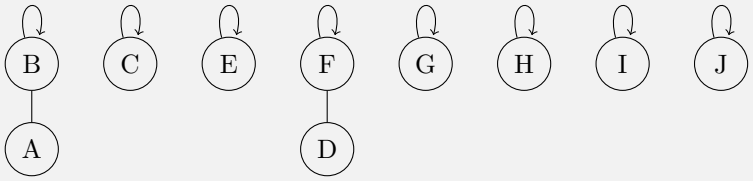
(2) The current Union-Find data structure after adding the edge to MST if the decision is "*YES*" Also, upon completion of the algorithm, please:

(3) draw the final MST, and
(4) provide the minimum total route length (in kilometers) achieved by this MST.

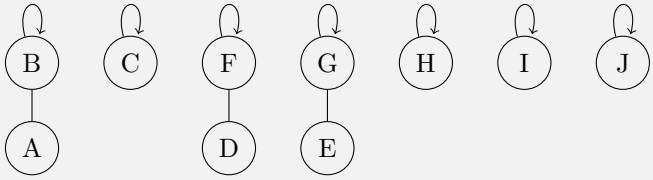
(1) Step 1: edge: $\{A, B\}$, decision: *YES*



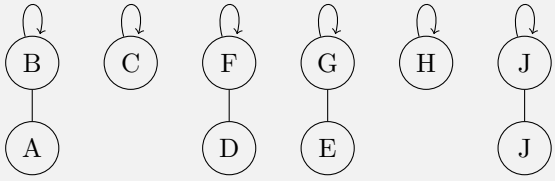
(2) Step 2: edge: $\{D, F\}$, decision: *YES*



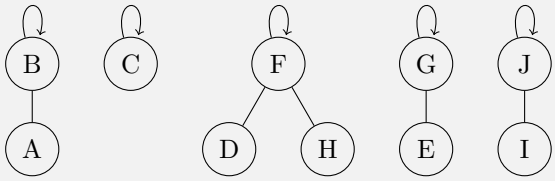
(3) Step 3: edge: $\{E, G\}$, decision: *YES*



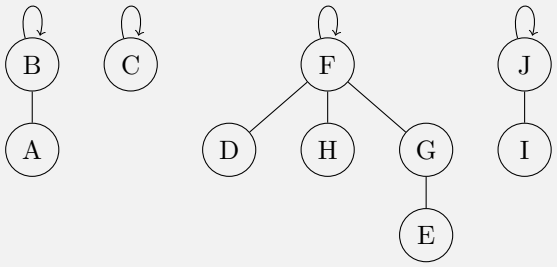
(4) Step 4: edge: $\{I, J\}$, decision: *YES*



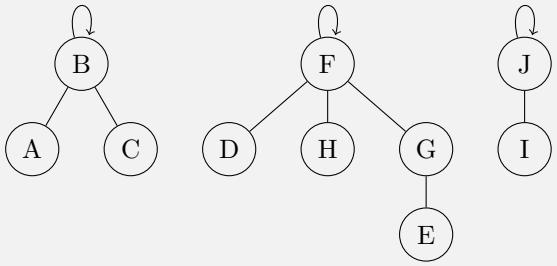
(5) Step 5: edge: $\{F, H\}$, decision: *YES*



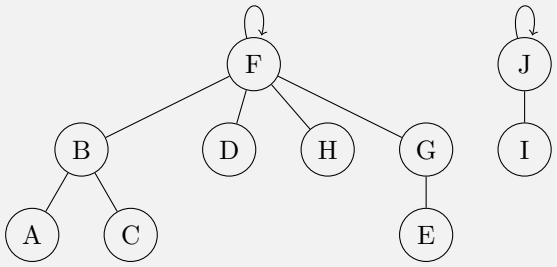
(6) Step 6: edge: $\{G, H\}$, decision: *YES*



(7) Step 7: edge: $\{B, C\}$, decision: *YES*

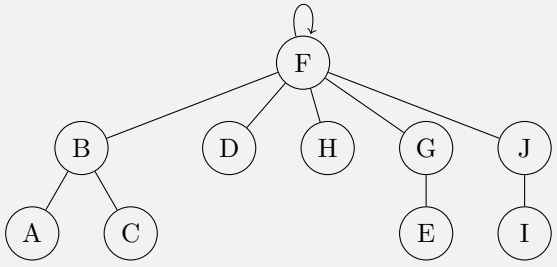


(8) Step 8: edge: $\{C, E\}$, decision: *YES*



(9) Step 9: edge: $\{F, G\}$, decision: *NO*

(10) Step 10: edge: $\{H, I\}$, decision: *YES*



(11) Step 11: edge: $\{B, E\}$, decision: *NO*

(12) Step 12: edge: $\{D, E\}$, decision: *NO*

(13) Step 13: edge: $\{A, D\}$, decision: *NO*

(14) Step 14: edge: $\{A, E\}$, decision: *NO*

(15) Step 15: edge: $\{F, I\}$, decision: *NO*

(16) Step 16: edge: $\{G, J\}$, decision: *NO*

(17) Step 17: edge: $\{C, J\}$, decision: *NO*

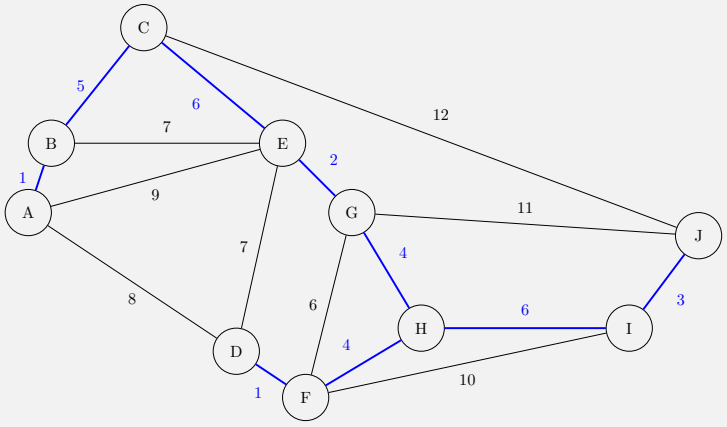


Figure 4.3: Kruskal's Minimum Spanning Tree, with cost of 32

(B) Run **Dijkstra's** Algorithm on the same graph to find the shortest path from Santa Lucia Train Station (node B) to all the other tourist destinations.

To answer this question, you need to follow the example in 17c_Dijkstra_Worked_Example and provide the following information for each step:

(1) A table that tracks the distance value ($d[u]$) and parent ($p[u]$) of each node. Also, upon completion of the algorithm, please:

(2) draw the final shortest-path tree.

(1) Step 1:

u	A	B	C	D	E	F	G	H	I	J
$d[u]$	1	0	∞	∞	∞	∞	∞	∞	∞	∞
$p[u]$	B	-	-	-	-	-	-	-	-	-

(2) Step 2:

u	A	B	C	D	E	F	G	H	I	J
$d[u]$	1	0	5	∞	∞	∞	∞	∞	∞	∞
$p[u]$	B	-	B	-	-	-	-	-	-	-

(3) Step 3:

u	A	B	C	D	E	F	G	H	I	J
$d[u]$	1	0	5	∞	7	∞	∞	∞	∞	∞
$p[u]$	B	-	B	-	B	-	-	-	-	-

(4) Step 4:

u	A	B	C	D	E	F	G	H	I	J
$d[u]$	1	0	5	9	7	∞	∞	∞	∞	∞
$p[u]$	B	-	B	A	B	-	-	-	-	-

(5) Step 5:

u	A	B	C	D	E	F	G	H	I	J
$d[u]$	1	0	5	9	7	∞	∞	∞	∞	17
$p[u]$	B	-	B	A	B	-	-	-	-	C

(6) Step 6:

u	A	B	C	D	E	F	G	H	I	J
$d[u]$	1	0	5	9	7	∞	9	∞	∞	17
$p[u]$	B	-	B	A	B	-	E	-	-	C

(7) Step 7:

u	A	B	C	D	E	F	G	H	I	J
$d[u]$	1	0	5	9	7	15	9	∞	∞	17
$p[u]$	B	-	B	A	B	G	E	-	-	C

(8) Step 8:

u	A	B	C	D	E	F	G	H	I	J
$d[u]$	1	0	5	9	7	15	9	13	∞	17
$p[u]$	B	-	B	A	B	G	E	G	-	C

(9) Step 9:

u	A	B	C	D	E	F	G	H	I	J
$d[u]$	1	0	5	9	7	10	9	13	∞	17
$p[u]$	B	-	B	A	B	D	E	G	-	C

(10) Step 10:

u	A	B	C	D	E	F	G	H	I	J
$d[u]$	1	0	5	9	7	10	9	13	20	17
$p[u]$	B	-	B	A	B	D	E	G	F	C

(11) Step 11:

u	A	B	C	D	E	F	G	H	I	J
$d[u]$	1	0	5	9	7	10	9	13	19	17
$p[u]$	B	-	B	A	B	D	E	G	H	C

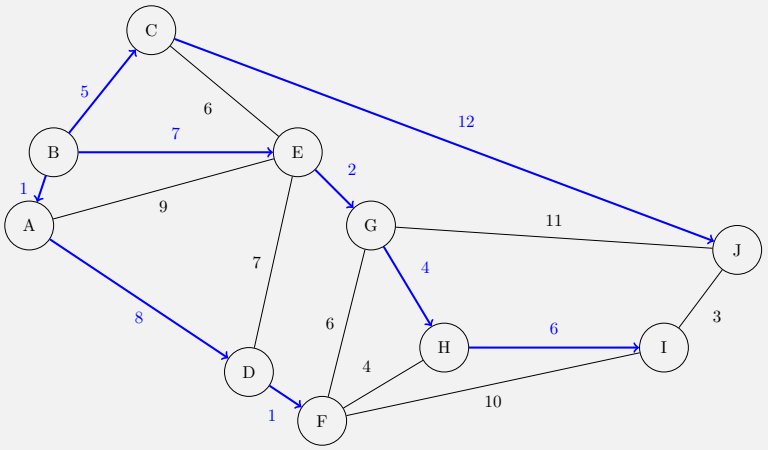


Figure 4.4: Djikstra's Shortest Path Tree