Question 1: Dijkstra's Weighted Shortest Path Trace

We trace Dijkstra's algorithm on the graph in Figure 1, starting from vertex **E**.

- **Start Vertex:** E
- **Color Code:** **Visited (S)**, **Shortest Path Tree / Distances**.
- **Tie-Breaking:** Alphabetical order for vertices with equal current distances.

Step 0: Initialization

- Set $d[E] = 0$, all others ∞.
- Priority Queue: $\{(E, 0)\}$.

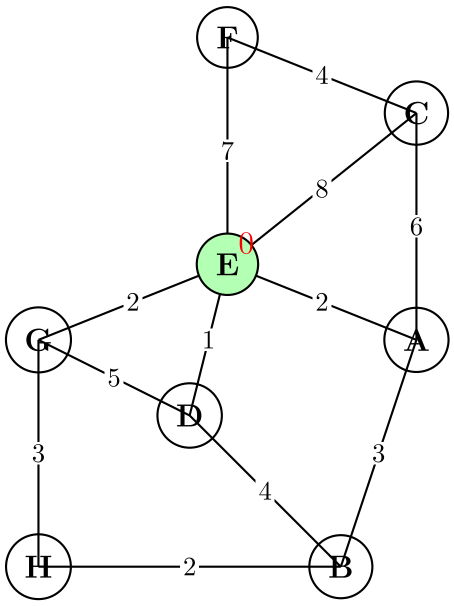


Figure 1: Start at E.

Step 1: Process E

Extract **E** (dist 0). Update neighbors:

- D: $0 + 1 = 1$. $(p[D] = E)$
- A: $0 + 2 = 2$. $(p[A] = E)$
- G: $0 + 2 = 2$. $(p[G] = E)$
- F: $0 + 7 = 7$. $(p[F] = E)$
- C: $0 + 8 = 8$. $(p[C] = E)$

Queue: $\{D : 1, A : 2, G : 2, F : 7, C : 8\}$. Min is **D**.

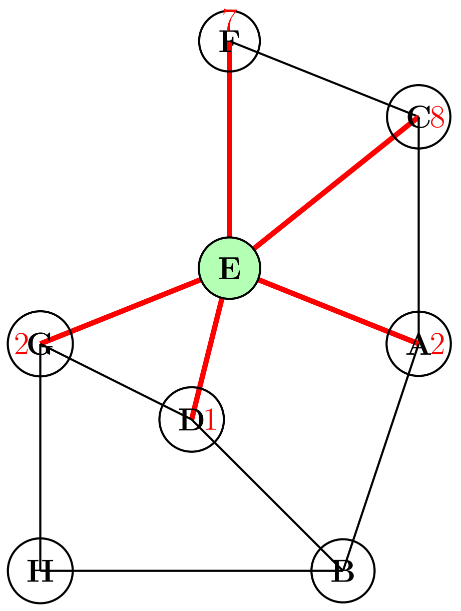


Figure 2: E processed. Neighbors updated.

Step 2: Process D

Extract **D** (dist 1). Add edge **(E, D)** to tree. Update neighbors:

- B: $d[D] + 4 = 1 + 4 = 5 < \infty$. Update $d[B] = 5, p[B] = D$.
- G: $d[D] + 5 = 6 > 2$. No update.

Queue: $\{A : 2, G : 2, B : 5, F : 7, C : 8\}$. Min is **A** or **G**. Pick **A**.

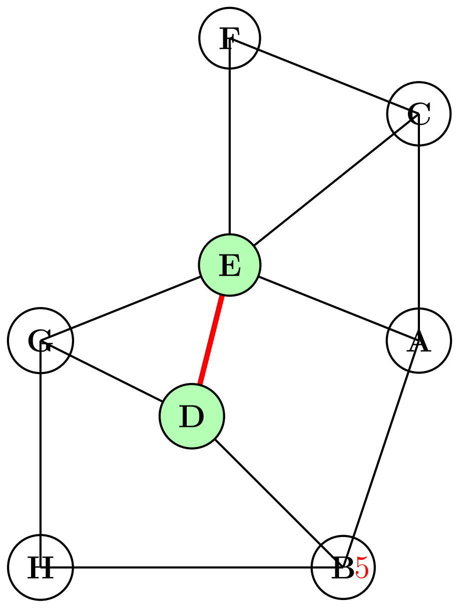


Figure 3: D processed. B discovered.

Step 3: Process A

Extract **A** (dist 2). Add edge **(E, A)** to tree.

- B: $d[A] + 3 = 2 + 3 = 5$. Tie with current. No change.

- C: $d[A] + 6 = 2 + 6 = 8$. Tie with current. No change.

Queue: $\{G : 2, B : 5, F : 7, C : 8\}$. Min is **G**.

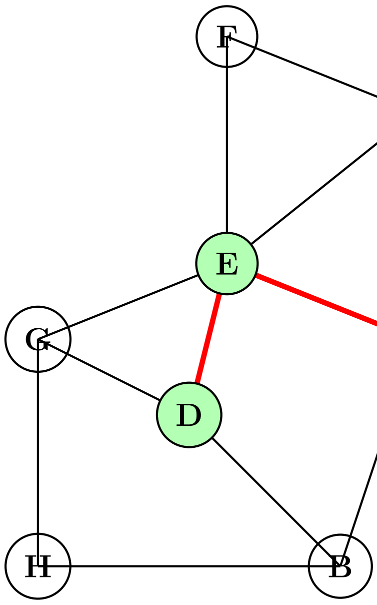


Figure 4: A processed.

Step 4: Process G

Extract **G** (dist 2). Add edge **(E, G)** to tree.

- H: $d[G] + 3 = 2 + 3 = 5 < \infty$. Update $d[H] = 5, p[H] = G$.

Queue: $\{B : 5, H : 5, F : 7, C : 8\}$. Min is **B** or **H**. Pick **B**.

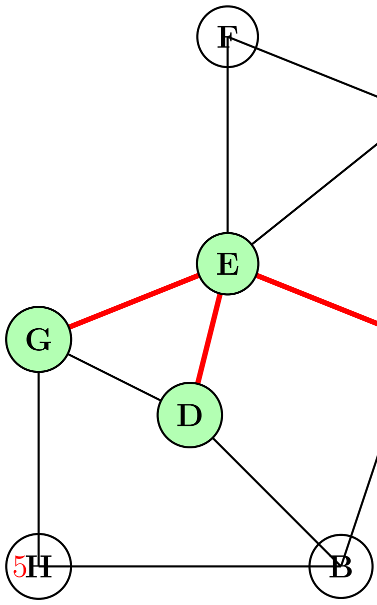


Figure 5: G processed. H discovered.

Step 5: Process B

Extract **B** (dist 5). Parent is D. Add edge **(D, B)** to tree.

- H: $5 + 2 = 7 > 5$. No change.

Queue: $\{H : 5, F : 7, C : 8\}$. Min is **H**.

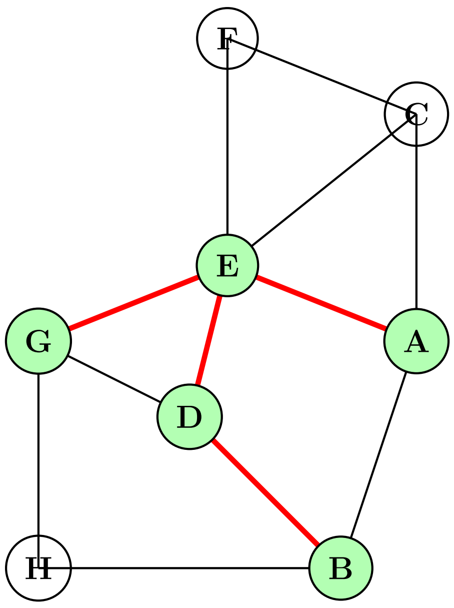


Figure 6: B processed via D.

Step 6: Process H

Extract **H** (dist 5). Parent is G. Add edge **(G, H)** to tree. Queue: $\{F : 7, C : 8\}$. Min is **F**.

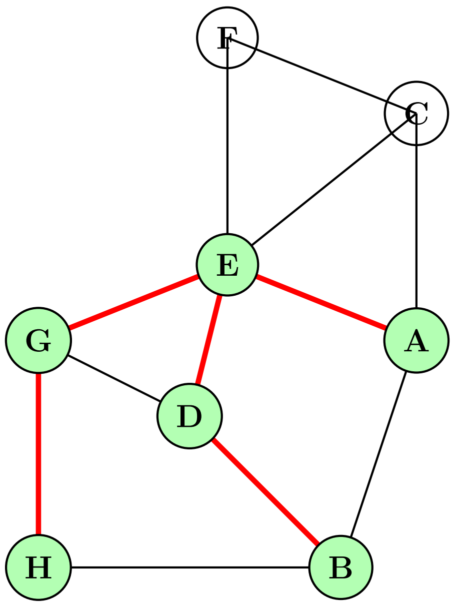


Figure 7: H processed via G.

Step 7 & 8: Process F and C

- Extract **F** (dist 7). Add **(E, F)**. C updated? $7 + 4 = 11 > 8$. No.
- Extract **C** (dist 8). Add **(E, C)**.

Queue Empty.

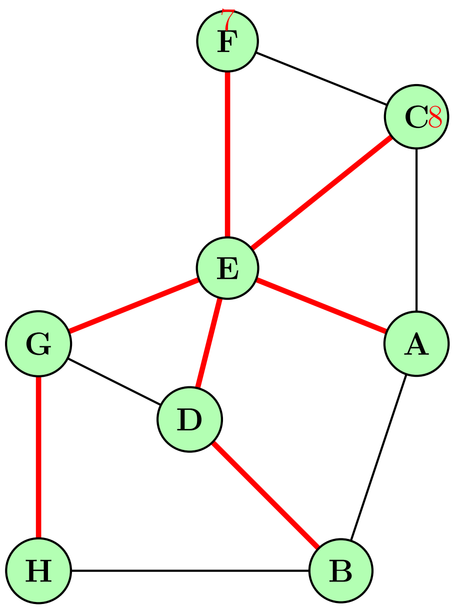


Figure 8: Final Shortest Path Tree (Red).

Vertex	**Distance**	**Parent**
E	0	-
D	1	E
A	2	E
G	2	E
B	5	D
H	5	G
F	7	E
C	8	E

Question 2: Prim's MST Algorithm Trace (Start: E)

We trace Prim's Minimum Spanning Tree (MST) algorithm starting from vertex **E**.

- S: The set of vertices included in the MST so far.
- **Cut Edges**: Edges connecting a vertex in S to a vertex outside S. We always pick the minimum weight edge from this set.

Step 0: Initialization

Start with vertex **E**.

- $S = \{E\}$.
- Available edges from E: $(E, D, 1), (E, A, 2), (E, G, 2), (E, F, 7), (E, C, 8)$.

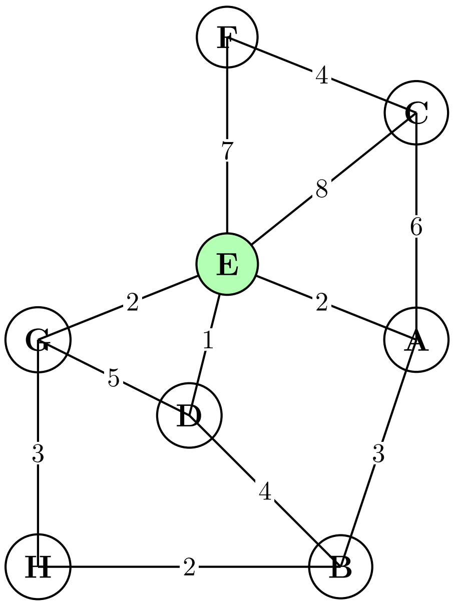


Figure 9: Start at E.

Step 1: Select Edge (E, D)

The minimum weight edge connected to $S = \{E\}$ is **(E, D)** with weight **1**.

- Add D to S. $S = \{E, D\}$.
- Add edge (E, D) to MST.
- New candidate edges from D: $(D, B, 4), (D, G, 5)$.

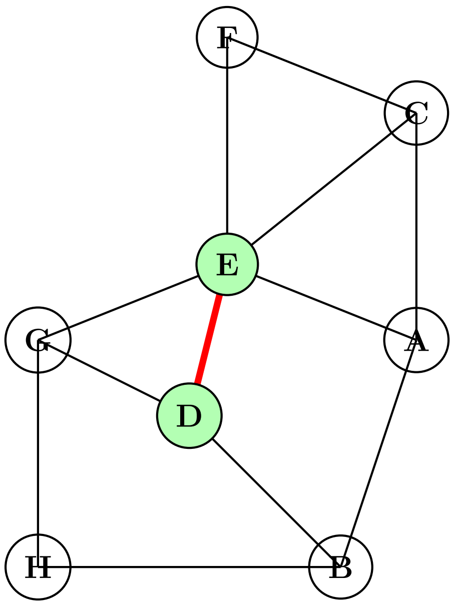


Figure 10: Edge (E, D) added.

Step 2: Select Edge (E, A)

Candidates: $(E, A, 2), (E, G, 2), (D, B, 4), (D, G, 5)$ (plus expensive ones). **Tie:** (E, A) and (E, G) are both 2. We pick **(E, A)** arbitrarily.

- Add A to S. $S = \{E, D, A\}$.
- Add edge (E, A) to MST.

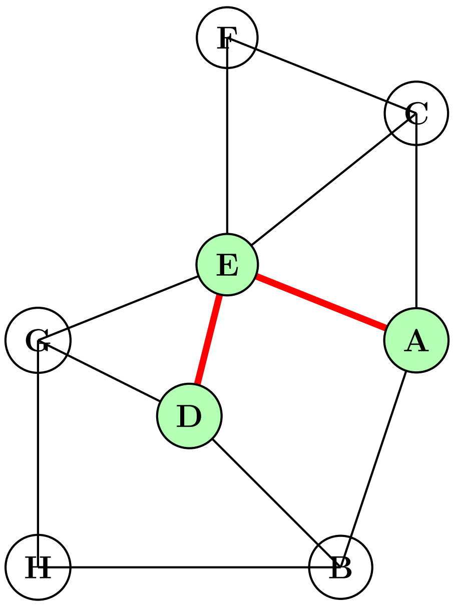


Figure 11: Edge (E, A) added.

Step 3: Select Edge (E, G)

Candidates: $(E, G, 2), (A, B, 3), (A, C, 6), (D, B, 4)$. Min is **(E, G)** with weight **2**.

- Add G to S. $S = \{E, D, A, G\}$.
- Add edge (E, G) to MST.

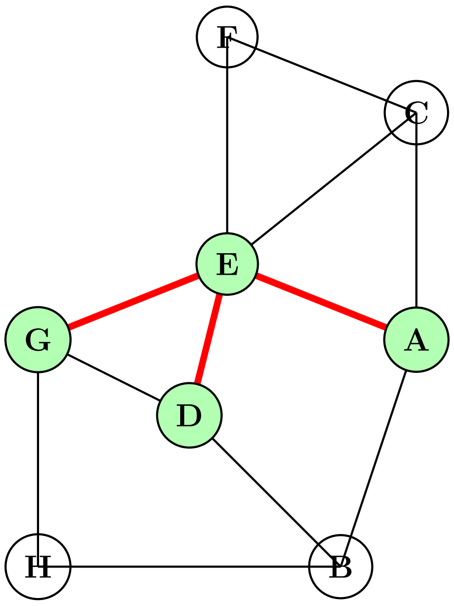


Figure 12: Edge (E, G) added.

Step 4: Select Edge (A, B)

Candidates: $(A, B, 3), (G, H, 3), (D, B, 4), (A, C, 6)$. **Tie:** (A, B) and (G, H) are both 3. Pick **(A, B)**.

- Add B to S. $S = \{E, D, A, G, B\}$.
- Add edge (A, B) to MST.

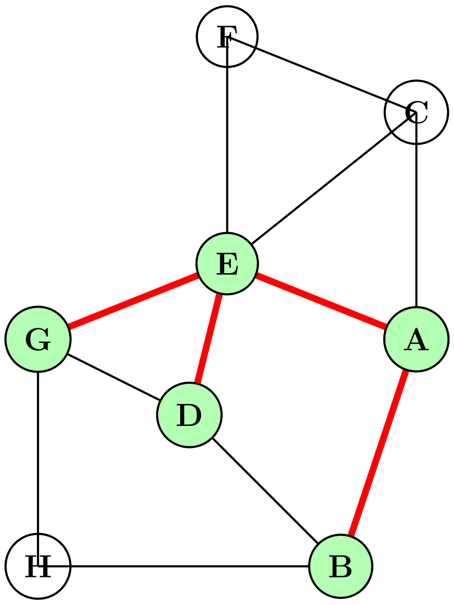


Figure 13: Edge (A, B) added.

Step 5: Select Edge (B, H)

Candidates: $(B, H, 2), (G, H, 3), (A, C, 6)$. Min is **(B, H)** with weight **2**.

- Add H to S. $S = \{E, D, A, G, B, H\}$.
- Add edge (B, H) to MST.

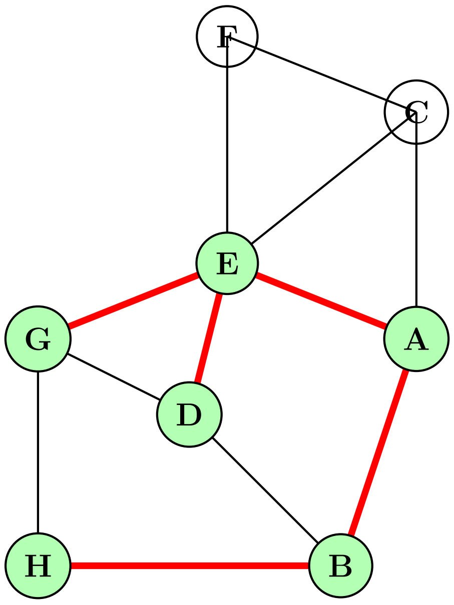


Figure 14: Edge (B, H) added.

Step 6: Select Edge (A, C)

Candidates from S to V-S: $(A, C, 6), (E, C, 8), (E, F, 7)$. Min is **(A, C)** with weight **6**.

- Add C to S. $S = \{E, D, A, G, B, H, C\}$.
- Add edge (A, C) to MST.

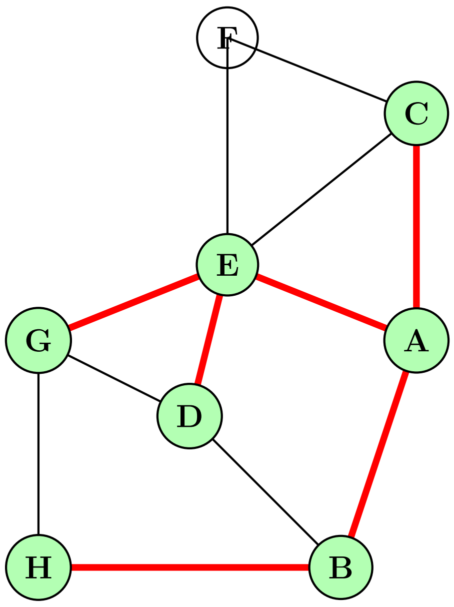


Figure 15: Edge (A, C) added.

Step 7: Select Edge (C, F)

Candidates: $(C, F, 4), (E, F, 7)$. Min is **(C, F)** with weight **4**.

- Add F to S. All vertices visited.
- Add edge (C, F) to MST.

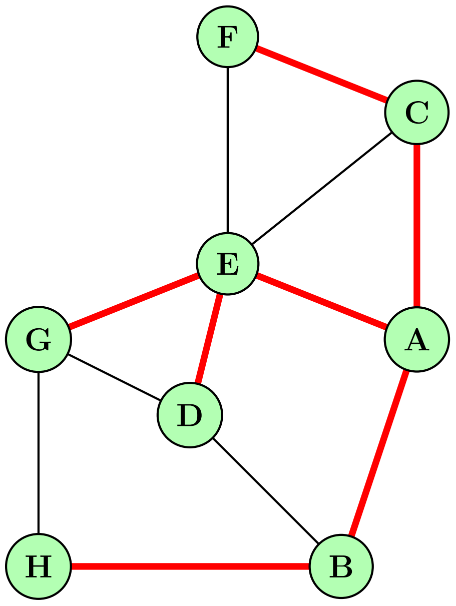


Figure 16: Final MST. Total Weight $= 1 + 2 + 2 + 3 + 2 + 6 + 4 = 20$.

Final MST Edges

Edge	**Weight**
(E, D)	1
(E, A)	2
(E, G)	2
(B, H)	2
(A, B)	3
(C, F)	4
(A, C)	6
Total	**20**

Question 3: Kruskal's MST Algorithm Trace

We trace Kruskal's algorithm on the graph in Figure 1.

- **Strategy:** Sort all edges by weight. Add edge if it doesn't form a cycle.
- **Goal:** Select $V - 1 = 7$ edges.

Step 0: Sorted Edges & Initialization

Sorted Edges: $(E, D, 1), (E, A, 2), (E, G, 2), (B, H, 2), (A, B, 3), (G, H, 3), (C, F, 4), \ldots$

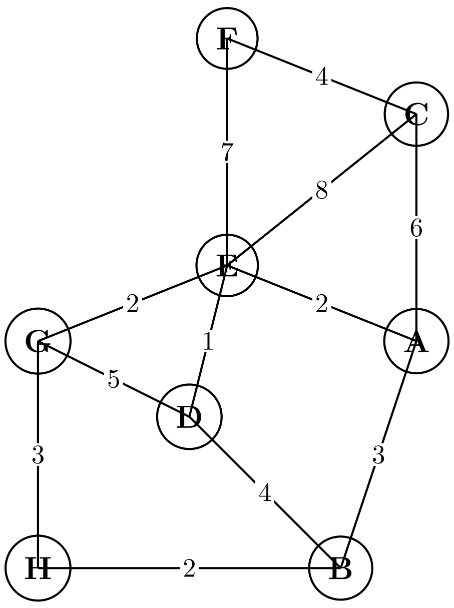


Figure 17: Initial Graph. No edges selected.

Step 1: Select (E, D)

Weight 1. No cycle. **Accept**. Sets: $\{E, D\}, \{A\}, \{B\}, \{C\}, \{F\}, \{G\}, \{H\}$.

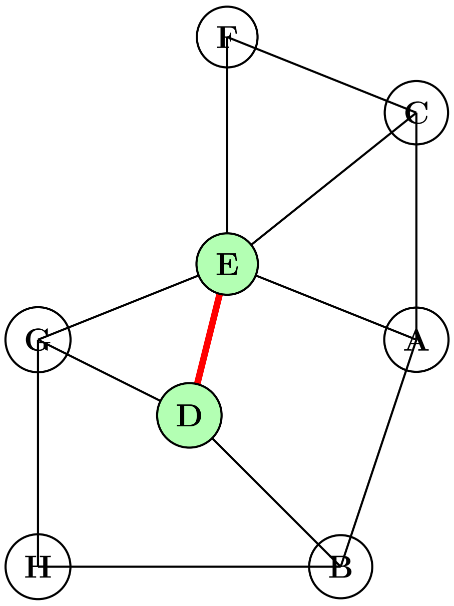


Figure 18: Added (E, D).

Step 2: Select (E, A)

Weight 2. No cycle. **Accept**. Sets: $\{E, D, A\}, \{B\}, \dots$

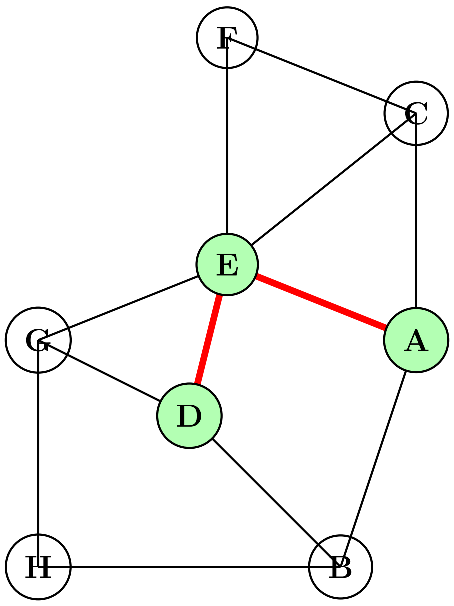


Figure 19: Added (E, A).

Step 3: Select (E, G)

Weight 2. No cycle. **Accept**. Sets: $\{E, D, A, G\}, \dots$

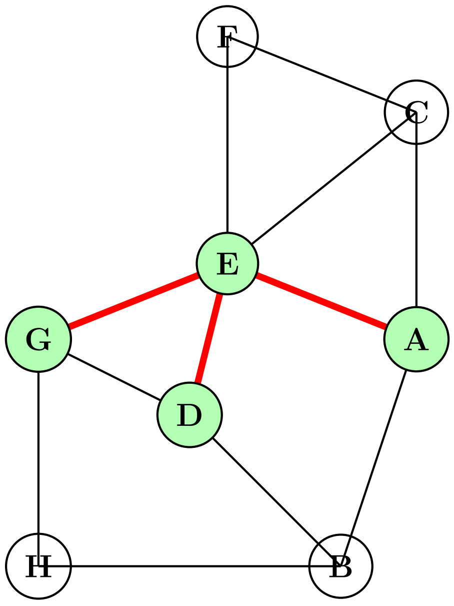


Figure 20: Added (E, G).

Step 4: Select (B, H)

Weight 2. No cycle (H and B are disjoint from E-D-A-G). **Accept**. Sets: $\{E, D, A, G\}, \{B, H\}, \dots$

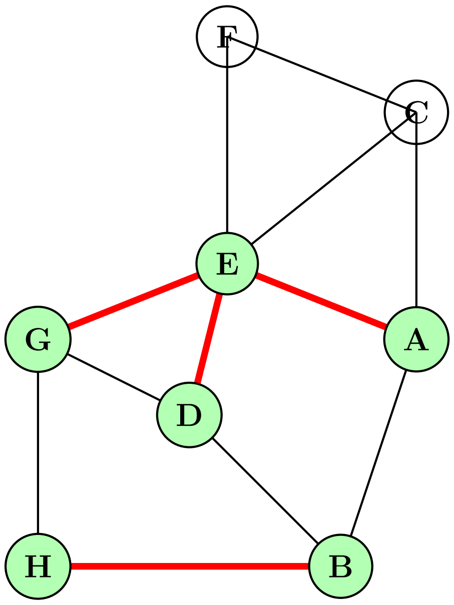


Figure 21: Added (B, H).

Step 5: Select (G, H)

Weight 3. Connects $\{E, D, A, G\}$ and $\{B, H\}$. No cycle. **Accept**. Sets: $\{E, D, A, G, B, H\}, \ldots$

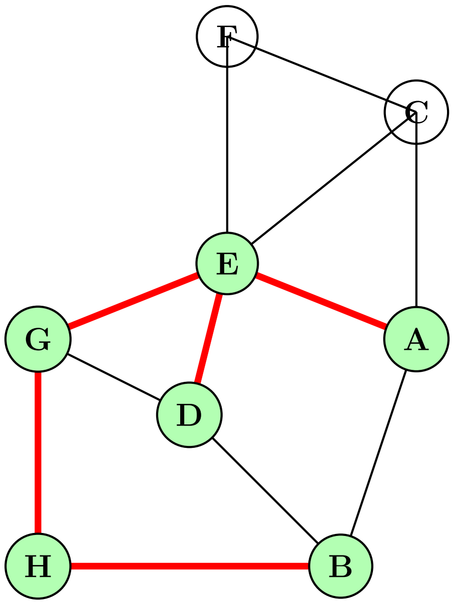


Figure 22: Added (G, H).

Step 6: Check (A, B)

Weight 3. A and B are already connected (via E-G-H). **Reject** (Cycle).

Step 7: Select (C, F)

Weight 4. C and F are disjoint from the main component. **Accept**. Sets: $\{E, D, A, G, B, H\}, \{C, F\}$.

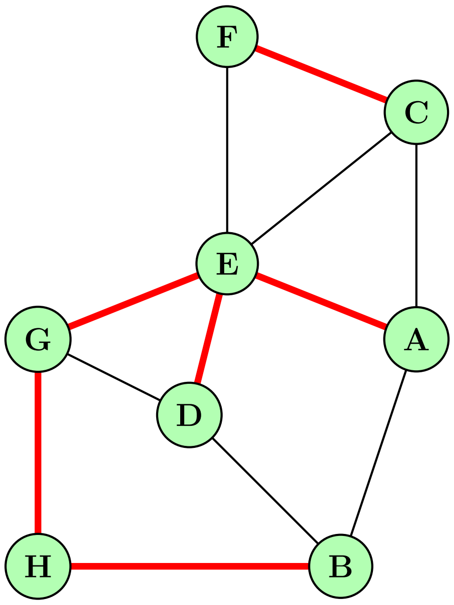


Figure 23: Added (C, F).

Step 8: Check (D, B)

Weight 4. D and B already connected. **Reject**.

Step 9: Check (G, D)

Weight 5. G and D already connected. **Reject**.

Step 10: Select (A, C)

Weight 6. Connects $\{E...\}$ and $\{C, F\}$. **Accept**. All vertices connected. 7 edges selected.

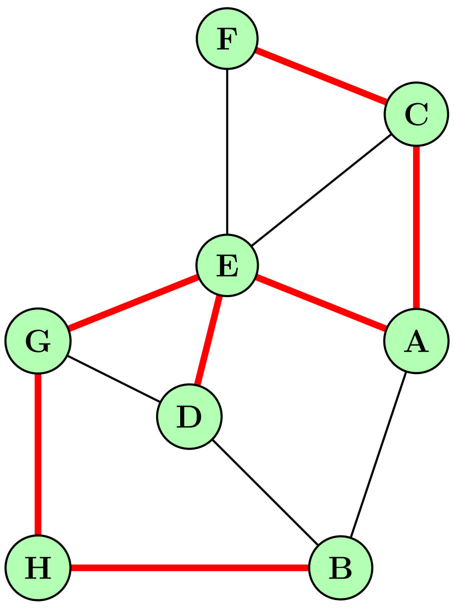


Figure 24: Final MST. Added (A, C).

Final Edges	**Weight**
(E, D)	1
(E, A)	2
(E, G)	2
(B, H)	2
(G, H)	3
(C, F)	4
(A, C)	6
Total Weight	**20**

Question 4: Breadth-First Search Trace (Start: E)

We trace the BFS traversal starting from vertex **E**.

- **Queue:** FIFO structure to manage vertices to visit.
- **Tie-Breaking:** Neighbors are processed in alphabetical order.
- **Color Code:** **Visited** nodes, **Tree Edges**.

Step 0: Initialization

Start at **E**. Mark E as visited. **Queue:** $[E]$

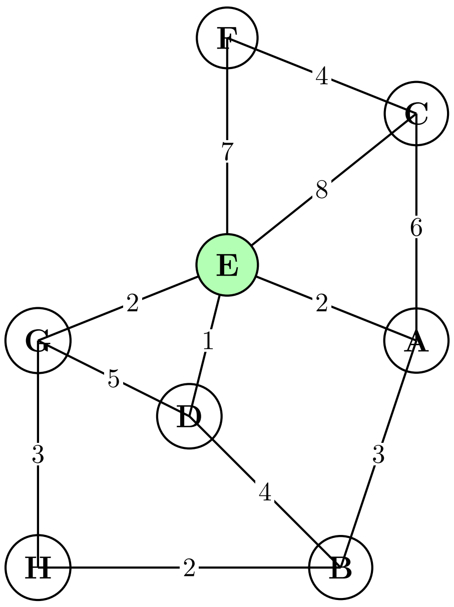


Figure 25: Start at E. Queue: [E]

Step 1: Expand E (Layer 1)

Dequeue **E**. Neighbors of E (Alphabetical): **A, C, D, F, G**. All are unvisited. Mark them visited and enqueue them. Add edges: (E,A), (E,C), (E,D), (E,F), (E,G). **Queue:** $[A, C, D, F, G]$

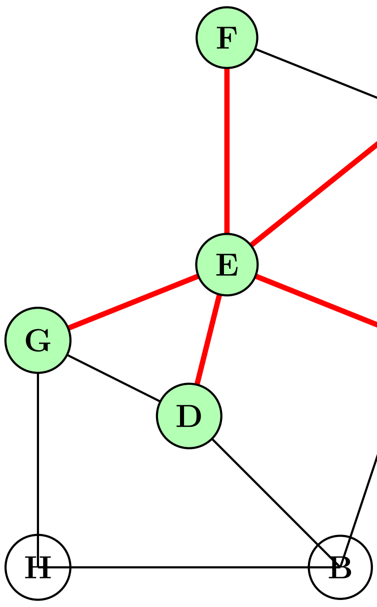


Figure 26: E processed. Layer 1 visited.

Step 2: Expand A (Start of Layer 2)

Dequeue **A**. Neighbors: C (visited), E (visited), **B** (Unvisited). Mark **B** visited. Enqueue B. Add edge (A, B). **Queue:** $[C, D, F, G, B]$

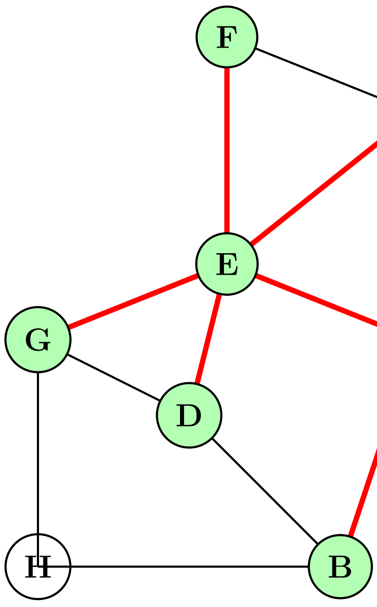


Figure 27: A processed. B discovered.

Step 3: Expand C, D, F

- Dequeue **C**. Neighbors F, E, A are all visited. No change.
- Dequeue **D**. Neighbors E, G, B are all visited. No change.
- Dequeue **F**. Neighbors C, E are all visited. No change.

Queue: $[G, B]$

Step 4: Expand G

Dequeue **G**. Neighbors: E (visited), D (visited), **H** (Unvisited). Mark **H** visited. Enqueue H. Add edge (G, H). **Queue:** $[B, H]$

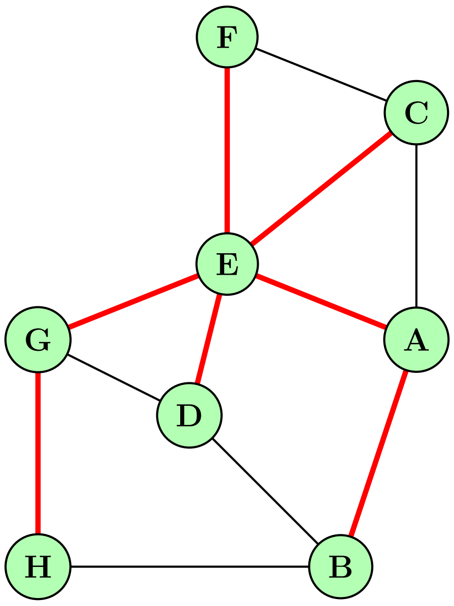


Figure 28: G processed. H discovered via G.

Step 5: Process Remaining Queue (B, H)

- Dequeue **B**. Neighbors A, D, H all visited.
- Dequeue **H**. Neighbors G, B all visited.

Queue empty. BFS Complete.

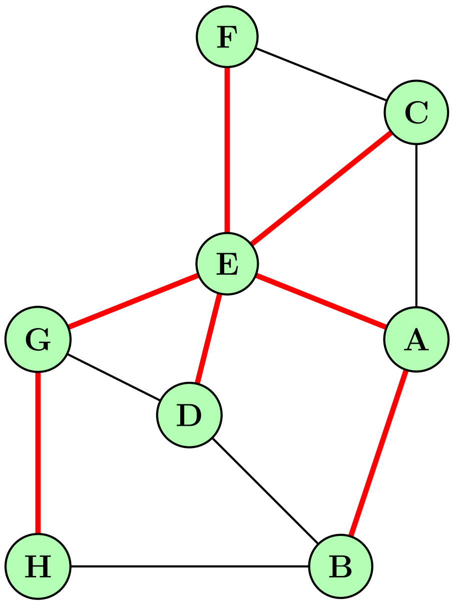


Figure 29: Final BFS Tree.

Vertex	**Discovery Order**	**Parent**
E	1	-
A	2	E
C	3	E
D	4	E
F	5	E
G	6	E
B	7	A
H	8	G

Question 5: Topological Sort (Kahn's Algorithm)

We find a topological ordering by iteratively removing vertices with **In-Degree 0**.

- **Tie-Breaking:** Alphabetical order (e.g., if A and B are both available, pick A).
- **Color Code:** **Active** (In-Degree 0, ready to process), Processed.

Step 1: Initialization

Calculate initial in-degrees. **In-Degree 0: s List:** []

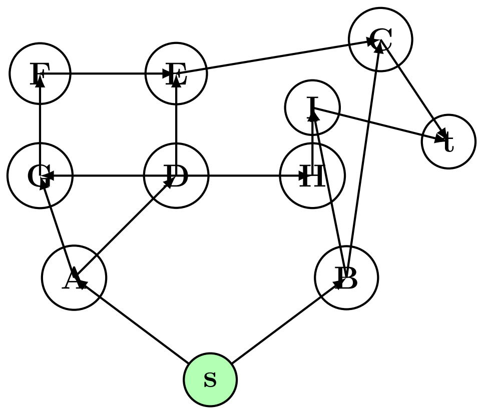
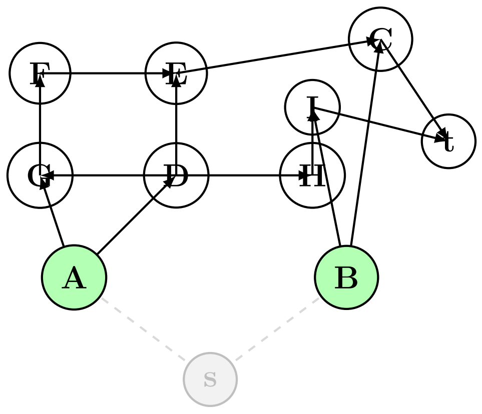


Figure 30: Start. 's' has in-degree 0.

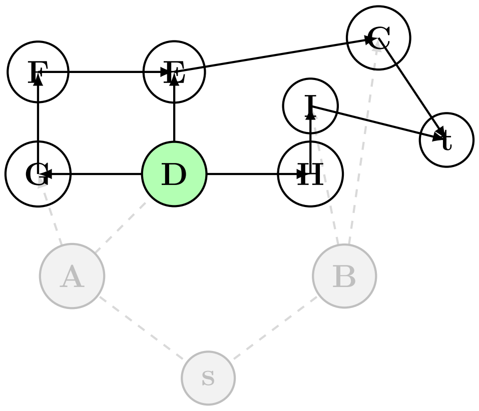
Step 2: Process 's'

Remove **s**. Decrement in-degrees of neighbors (A, B). New In-Degrees 0: **A, B**. **Sorted List:** $[s]$



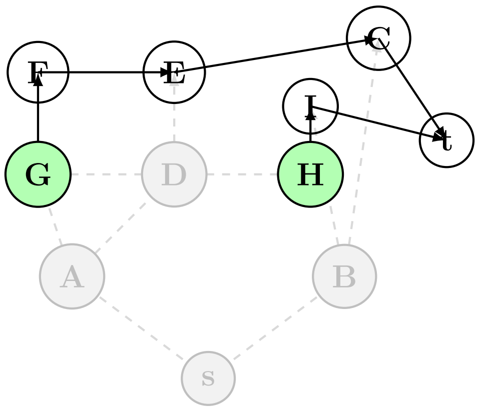
Step 3: Process 'A' and 'B'

1. Pick **A** (Alphabetical). Decrement G, D. - D only depended on A (Indegree becomes 0). - G depends on A and D (Indegree not 0 yet). 2. Pick **B**. Decrement I, C. - I depends on B, H. C depends on B, E. (Neither becomes 0). Available set: **D**. **Sorted List:** $[s, A, B]$



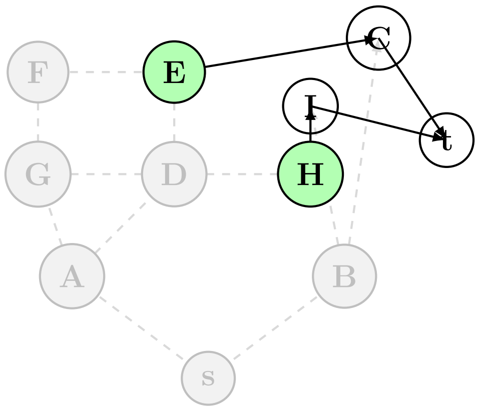
Step 4: Process 'D'

Remove **D**. Decrement G, E, H. - G depended on A (done) and D (done) → **G becomes 0**. - H depended on D (done) → **H becomes 0**. - E depends on D and F (F not done). Available set: **G, H**. **Sorted List:** $[s, A, B, D]$



Step 5: Process 'G' and 'F'

1. Pick **G**. Unlock F. Set: $\{H, F\}$. 2. Pick **F** (Alphabetical before H). Unlock E (E needed D, F). - D was done, F is now done → **E becomes 0**. Available set: **E, H**. **Sorted List:** $[s, A, B, D, G, F]$



Step 6: Process 'C', 'E', 'H' sequence

1. Set $\{E, H\}$. Pick **E**. Unlock C (C needed B, E). - B done, E done → **C becomes 0**. Set $\{C, H\}$. 2. Pick **C**. Unlock t (t needs I, C). I not ready. Set $\{H\}$. 3. Pick **H**. Unlock I (I

needed B, H). - B done, H done → **I becomes 0**. Set $\{I\}$. **Sorted List:** $[..., G, F, E, C, H]$

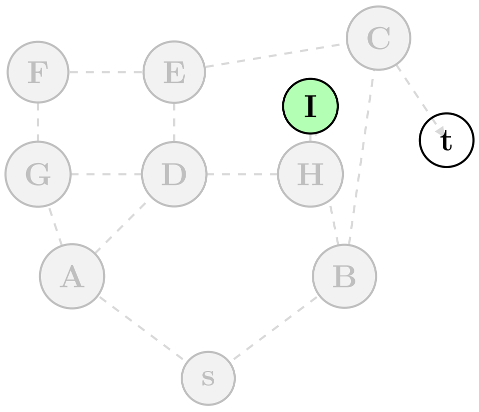


Figure 31: Processing E, C, H. Only I remains.

Step 7: Finalize

1. Process **I**. Unlock t. **t** becomes 0. 2. Process **t**. Queue Empty.

Final Topological Order	s, A, B, D, G, F, E, C, H, I, t