SIT221: Data Structures and Algorithms

Lecture 9: Tree and Graph Algorithms

Week 8 recording for prac

	Recording Name	Session Name	Date	Duration
Week 8	Week 8 - prac - recording_1	Week 8 - prac	2/09/2017 4:08 pm	01:09:28
Week 7	Week 7 - prac - recording_1	Week 7 - prac	26/08/2017 4:11 pm	00:39:42
Week 4	SIT221 - Data Structures And Algorithms - recording_1	SIT221 - Data Structures And Algorithms	3/08/2017 11:22 am	00:42:34
Week 3	SIT221 - Data Structures And Algorithms - recording_1	SIT221 - Data Structures And Algorithms	27/07/2017 10:55 am	00:40:01
Week 2	Week 2 - prac - recording_1	Week 2 - prac	20/07/2017 11:01 am	00:31:17

Assignment 1 - Issues

- Comments are important
- You need to submit the whole solution
- Don’t change things that have been advised not to change
- Compilation error
- Plagiarism
- Sorting with indices should only swap indices
- Powerset → good to see other ways

What are we up to? 1. Data structures

We have covered the core data structures – can you tell the difference & when to use?

1. Vectors
2. Linked Lists
3. Stacks
4. Queues
5. Dictionaries
6. Trees
7. Graphs

What are we up to? 2. Algorithms

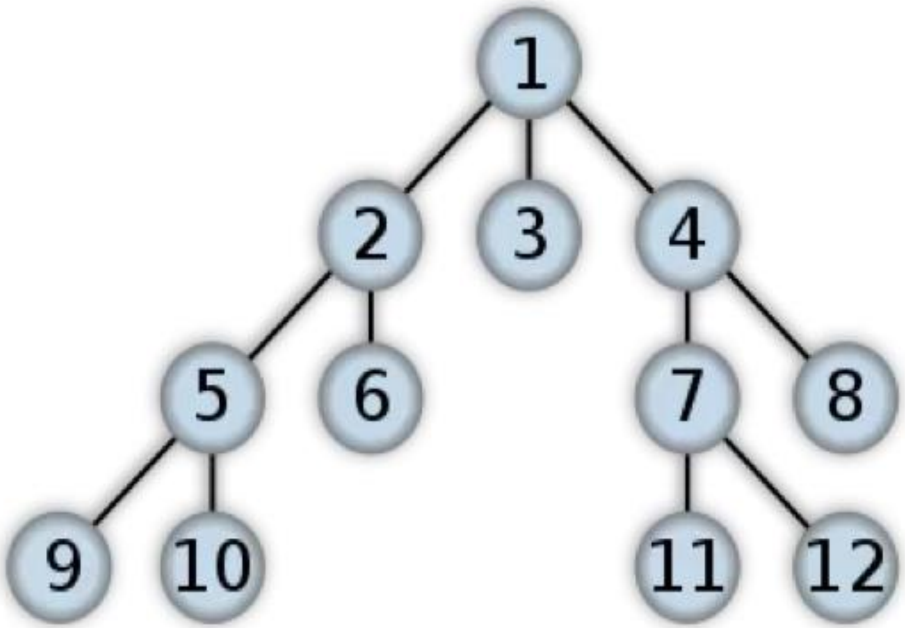
- Sorting [selection, insertion, merge, quick, Microsoft]
- Searching [linear, binary, key look up – dictionary]
- Basic operations in data structures: insert/add/traverse/delete/etc.
- Breadth first vs Depth first, Shortest Path, Minimum spanning tree.
- Dynamic Programming
- Greedy Algorithms

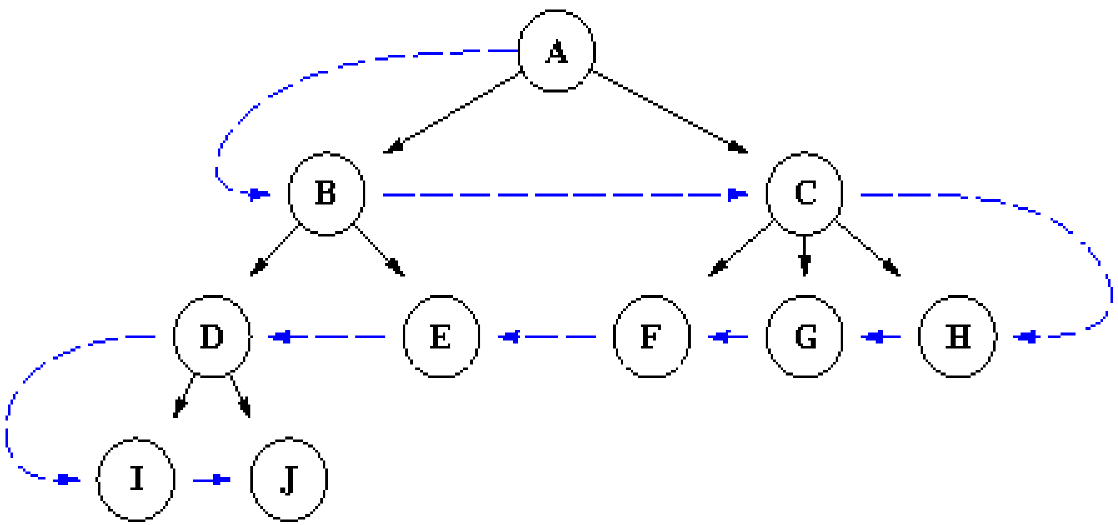
Graphs & Trees

- Every tree is a …
- But not every graph is a ...

Breadth-first Search - BFS

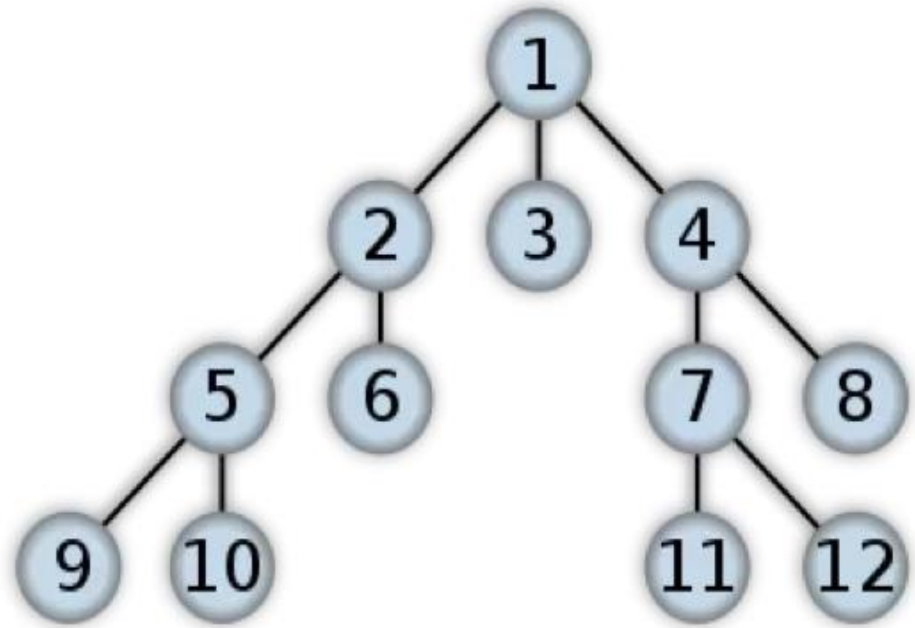
- Traversing graph or tree level by level





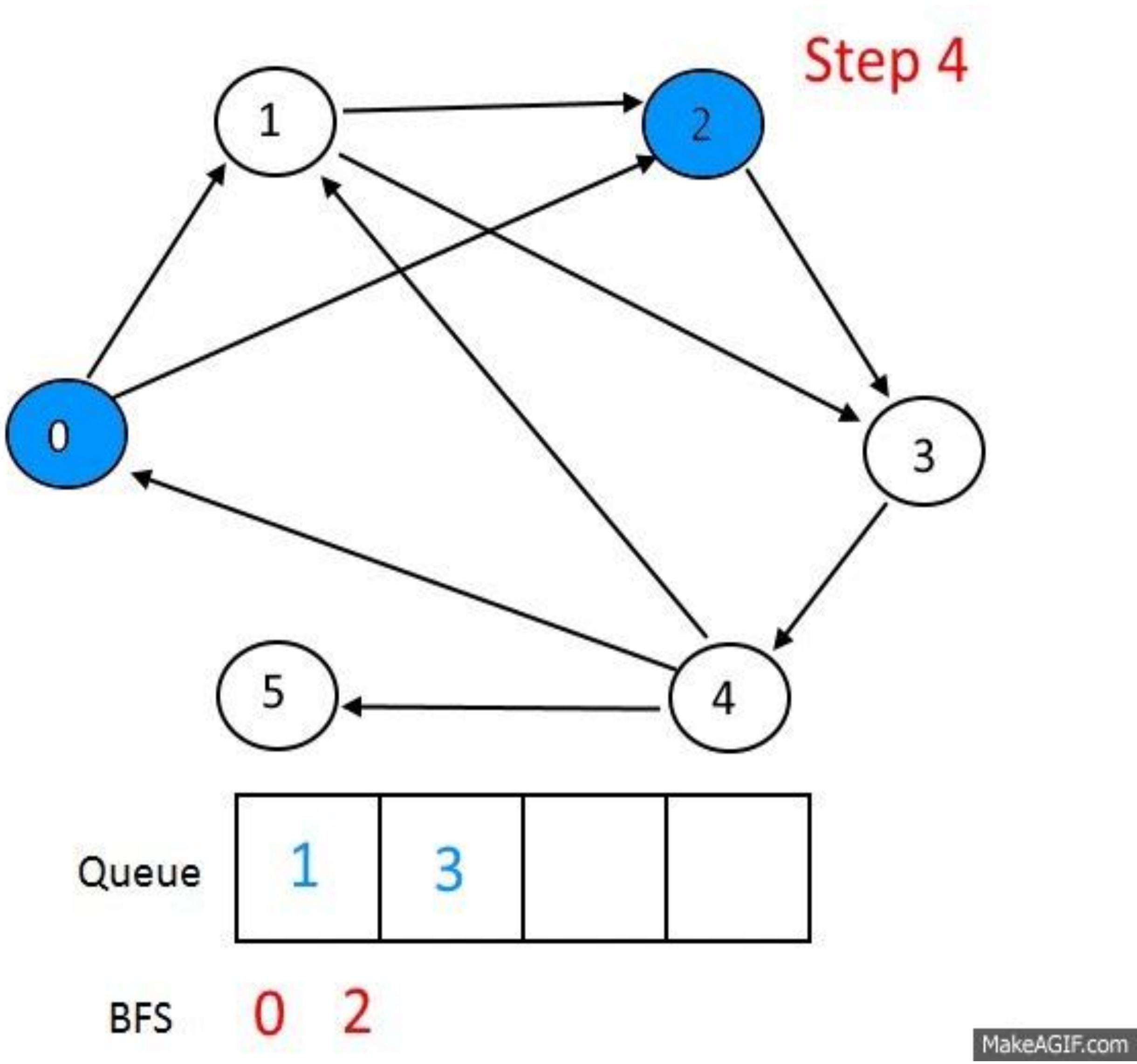
How to implement it?

1. Choose a starting node - in a tree, this will be the Root node.
2. Enque the node in a queue data structure
3. While Queue is not empty
    1. Dequeue a node from the queue
    2. Mark it as visited – avoid cycles/loops
    3. Enque all children nodes into the queue



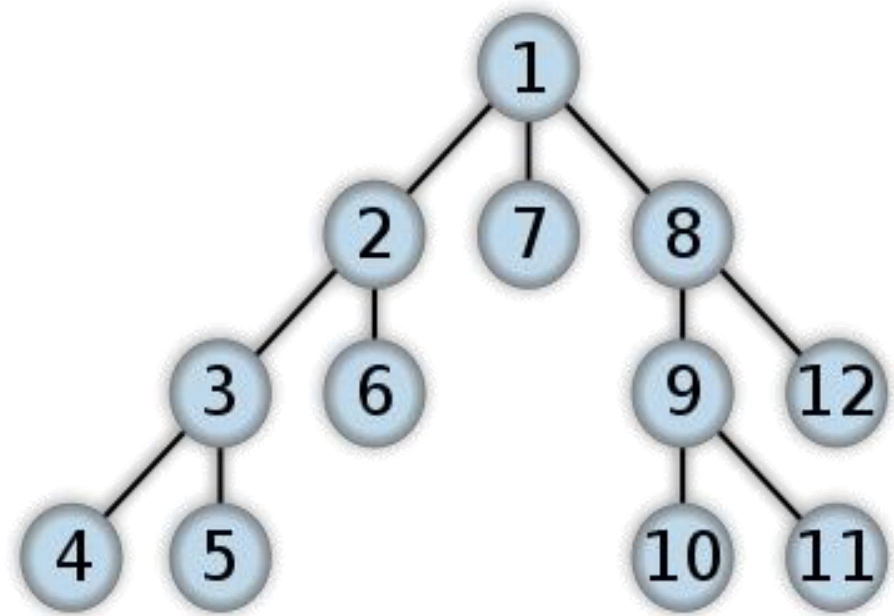
BFD in motion…

- Also try this: http://visualgo.net/dfsbfs



Graph traversal - Depth-first Search - DFS

1. Choose a starting node - in a tree, this will be the Root node.
2. Push the node in a stack data structure
3. While Stack is not empty
   1. Pop a node from the stack
   2. Mark it as visited – avoid cycles/loops
   3. Push all children nodes into the stack



Let's give it a go?

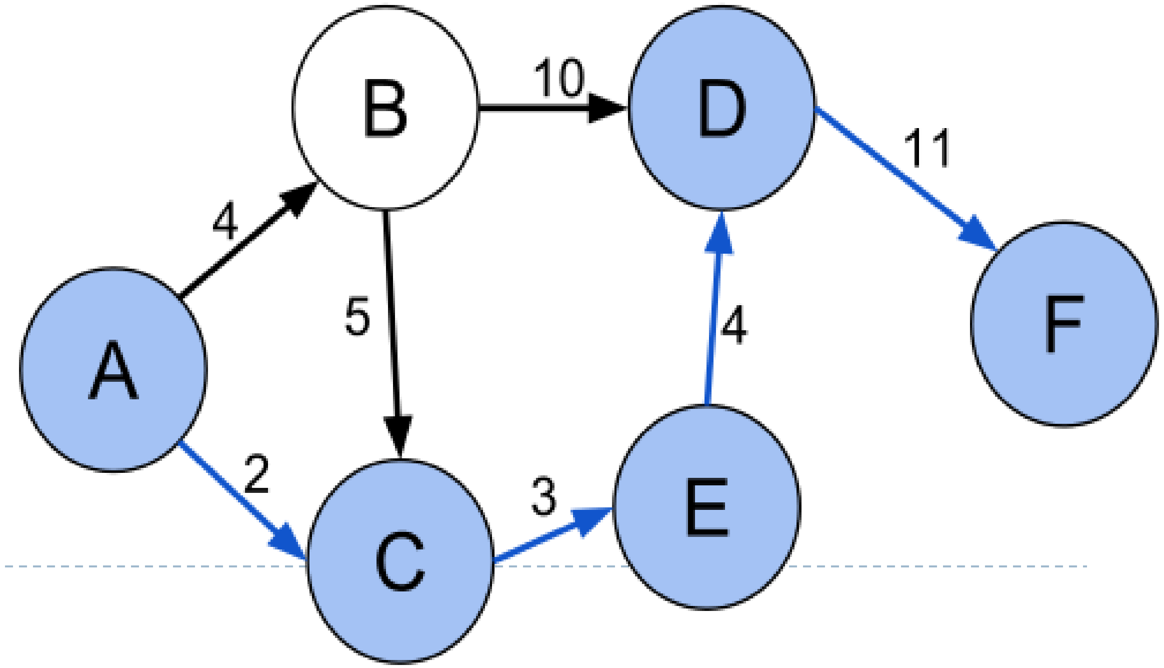
- Using our BinarySearchTree

Applications

- Social networks
- Web crawling
- Network broadcast
- Model checking

Shortest path

- What are the possible paths between A & F? which one is the shortest?
  - Path1: A → B → D → F [cost = 4 + 10 + 11]
  - Path2: A → B → C → E → D → F [cost = 4+5+3+4 + 11]
  - Path3: A→ C → E → D → F [cost = 2 + 3 + 4 + 11]



Dijkstra's algorithm

Input:

source node, destination node

Ouput:

- The min distance from the source node to the destination node
- The list of the nodes on the shortest path

Dijkstra's algorithm

Given a graph G(V,E)

- $\forall$ u, v $\in$ V

  If u and v are directly connected

  $c(u, v)$ = weight of (u,v)

  Else

  $c(u, v) = \infty$

- $\forall$ u $\in$ V, $c(u, u) = \infty$
- d(u) = the minimal distance from the source node to u.
- pre(u) = preceding of node u on the shortest path from the source node to u.

Dijkstra's algorithm

- Step 1. $V_T = \{v_{begin}\}$, $d(v_{begin}) = 0$, $E_T = \emptyset$
- Step 2. $\forall v \in V - \{v_{begin}\}$
  $d(v) = c(v_{begin}, v)$
  $pre(v) = v_{begin}$
- Step 3. If $v_{end} \in V_T$, stop. Otherwise, go to step 4.
- Step 4. $v^* = \mathrm{argmin}\{d(v)\}, v \in V - V_T$.
  $V_T = V_T \cup \{v^*\}$, $E_T = E_T \cup (pre(v^*), v^*)$
- Step 5. $\forall w \in V - V_T$, if $d(w) > d(v^*) + c(v^*, w)$, then
  $d(w) = d(v^*) + c(v^*, w)$
  $pre(w) = v^*$.

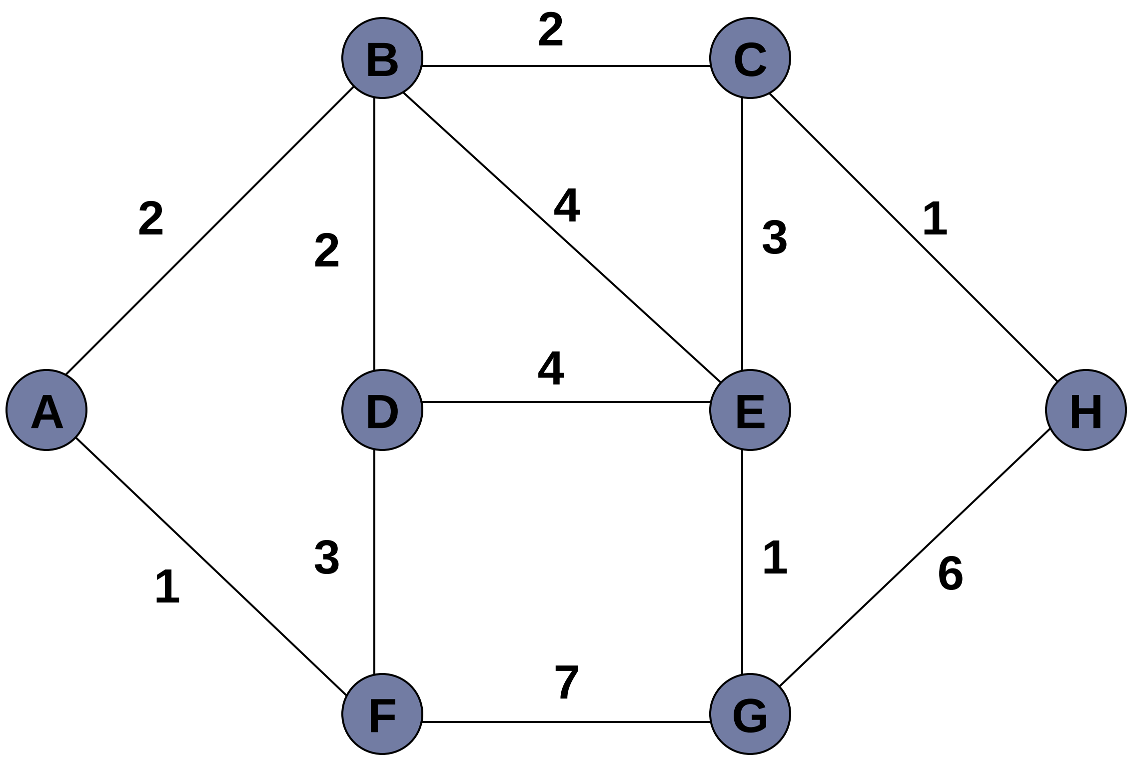
  Go to step 3.

Dijkstra's algorithm

Given a graph, how to find the shortest paths from a source node to all the remaining nodes?

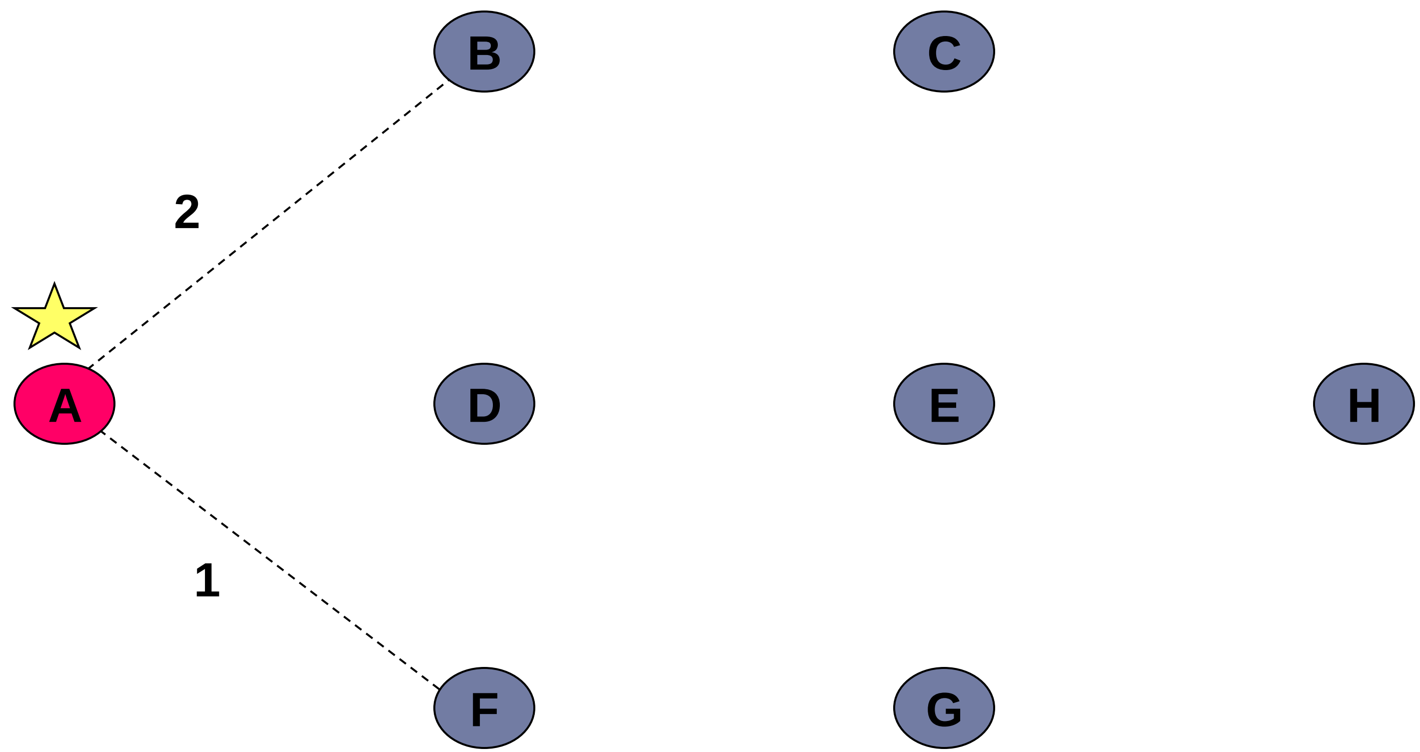
→ Replace Step 3 by condition $V_T = V$

Dijkstra's algorithm - Example

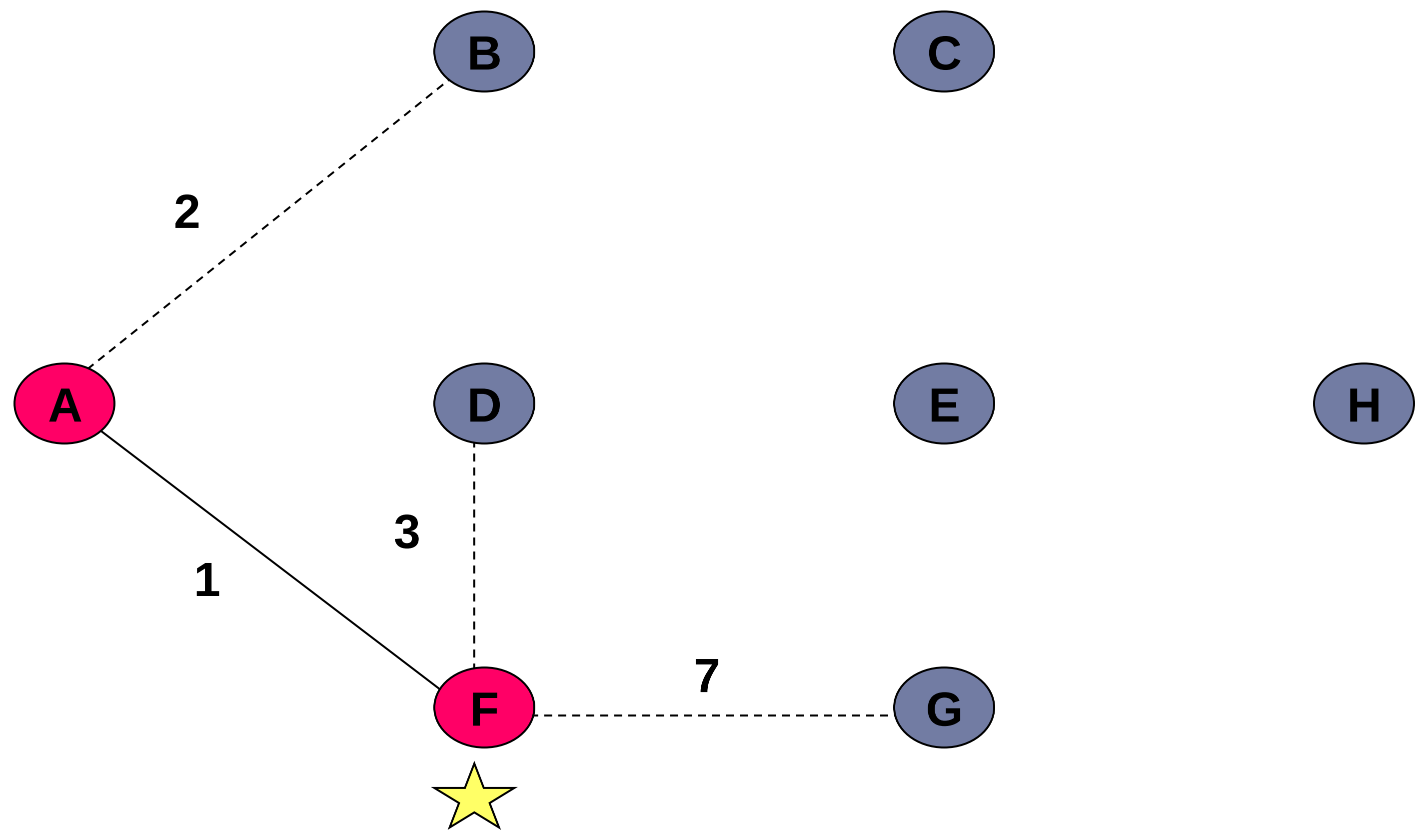


Find the shortest paths from A to all other nodes.

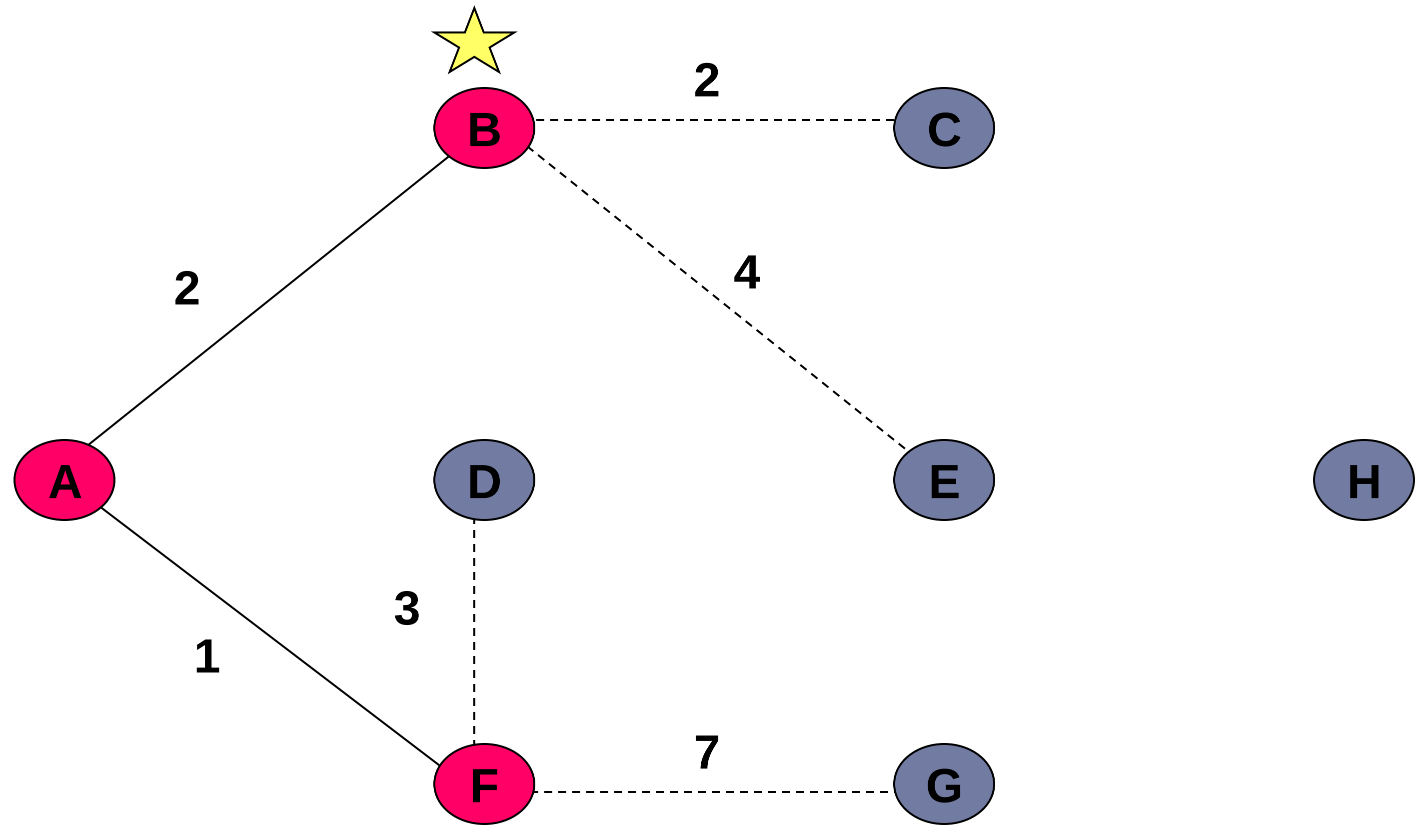
	V_T	d	pre
A	X	0	
B		2	A
C		∞	A
D		∞	A
E		∞	A
F		1	A
G		∞	A
H		∞	A



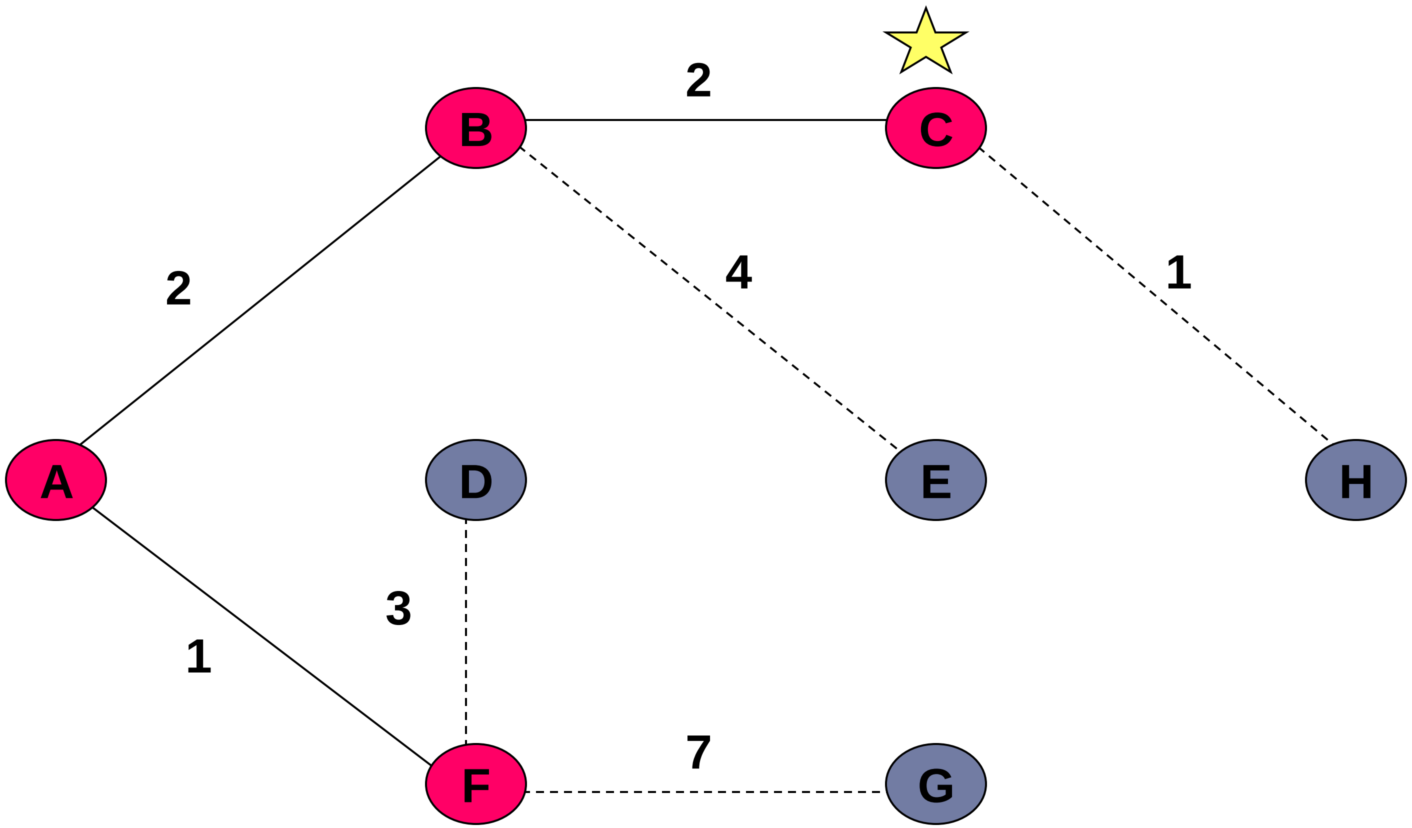
	V_T	d	pre
A	X	0	
B		2	A
C		∞	A
D		4	F
E		∞	A
F	X	1	A
G		8	F
H		∞	A



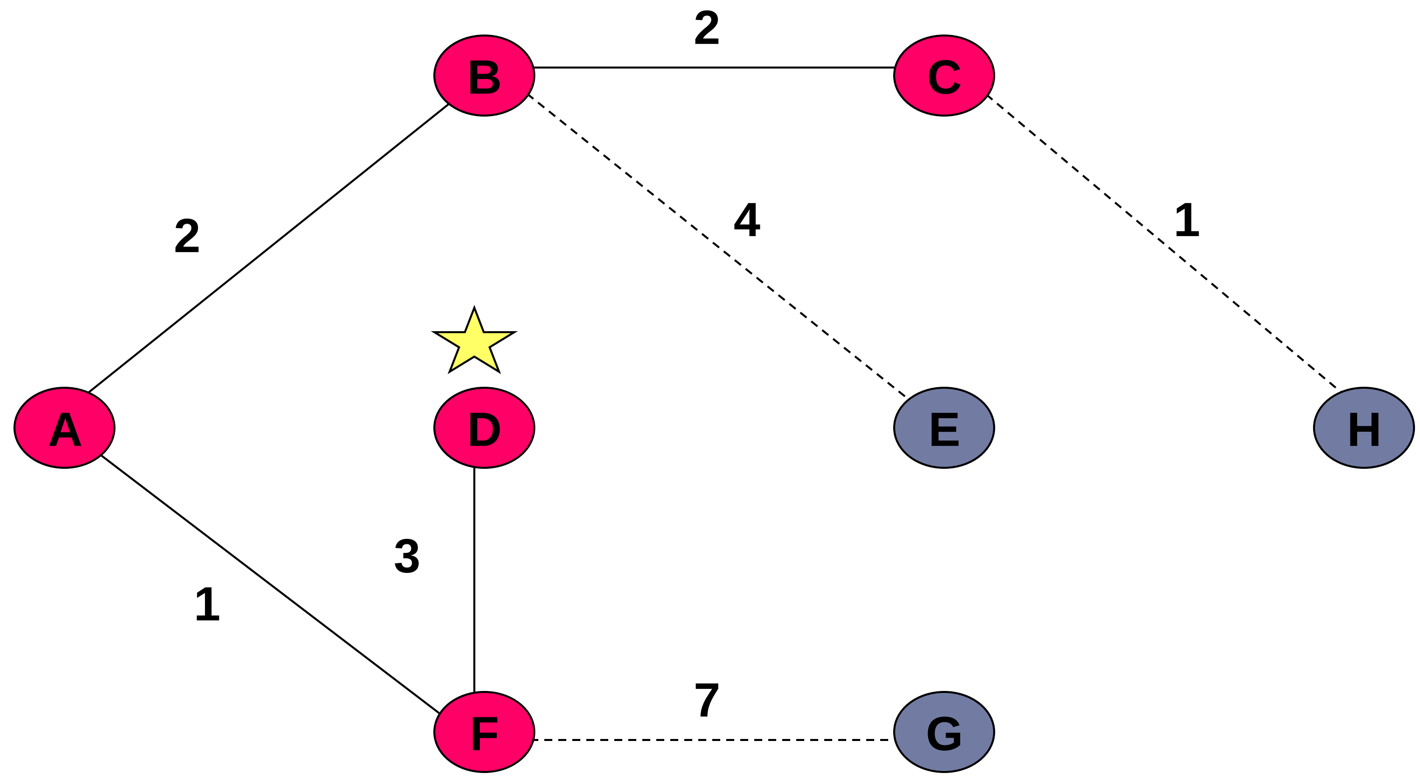
	V_T	d	pre
A	X	0	
B	X	2	A
C		4	B
D		4	F
E		6	B
F	X	1	A
G		8	F
H		∞	A



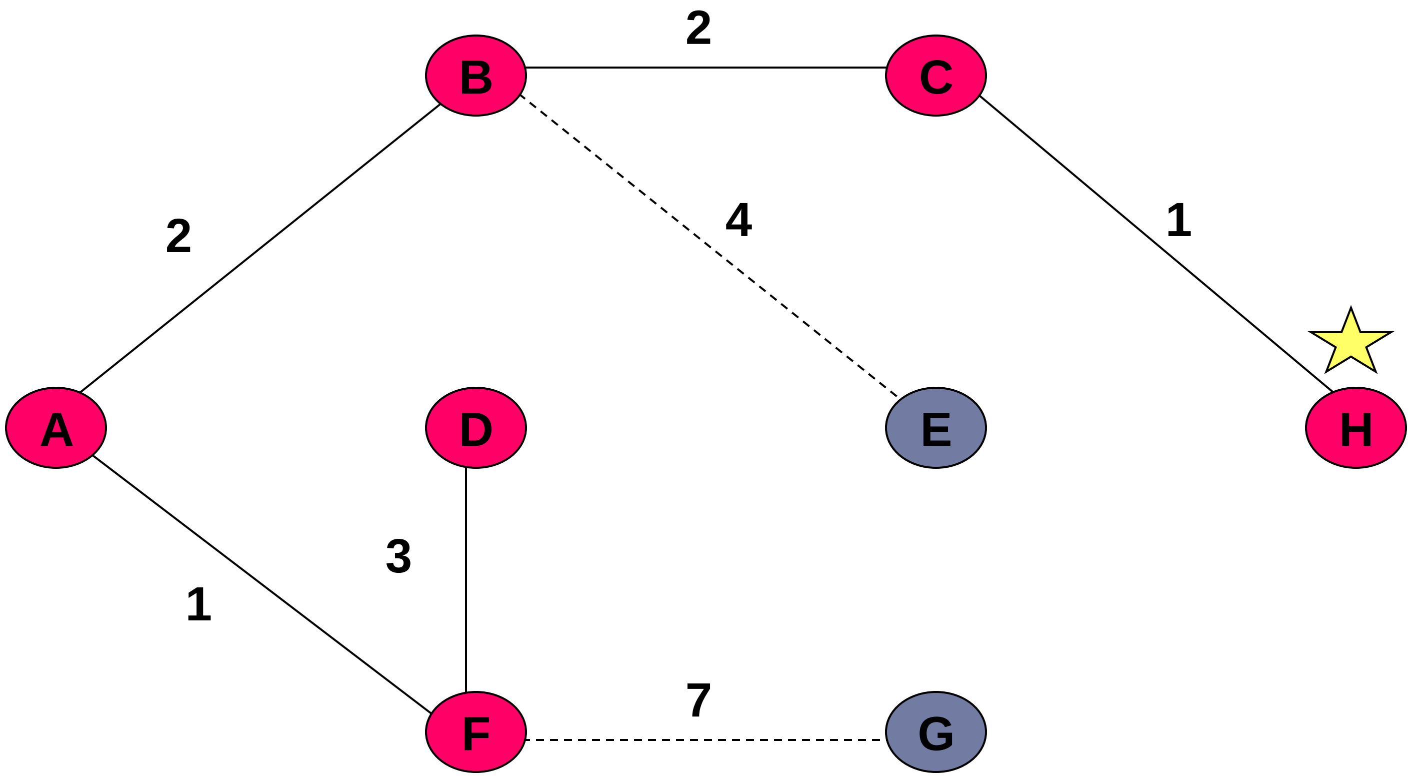
	V_T	d	pre
A	X	0	
B	X	2	A
C	X	4	B
D		4	F
E		6	B
F	X	1	A
G		8	F
H		5	C



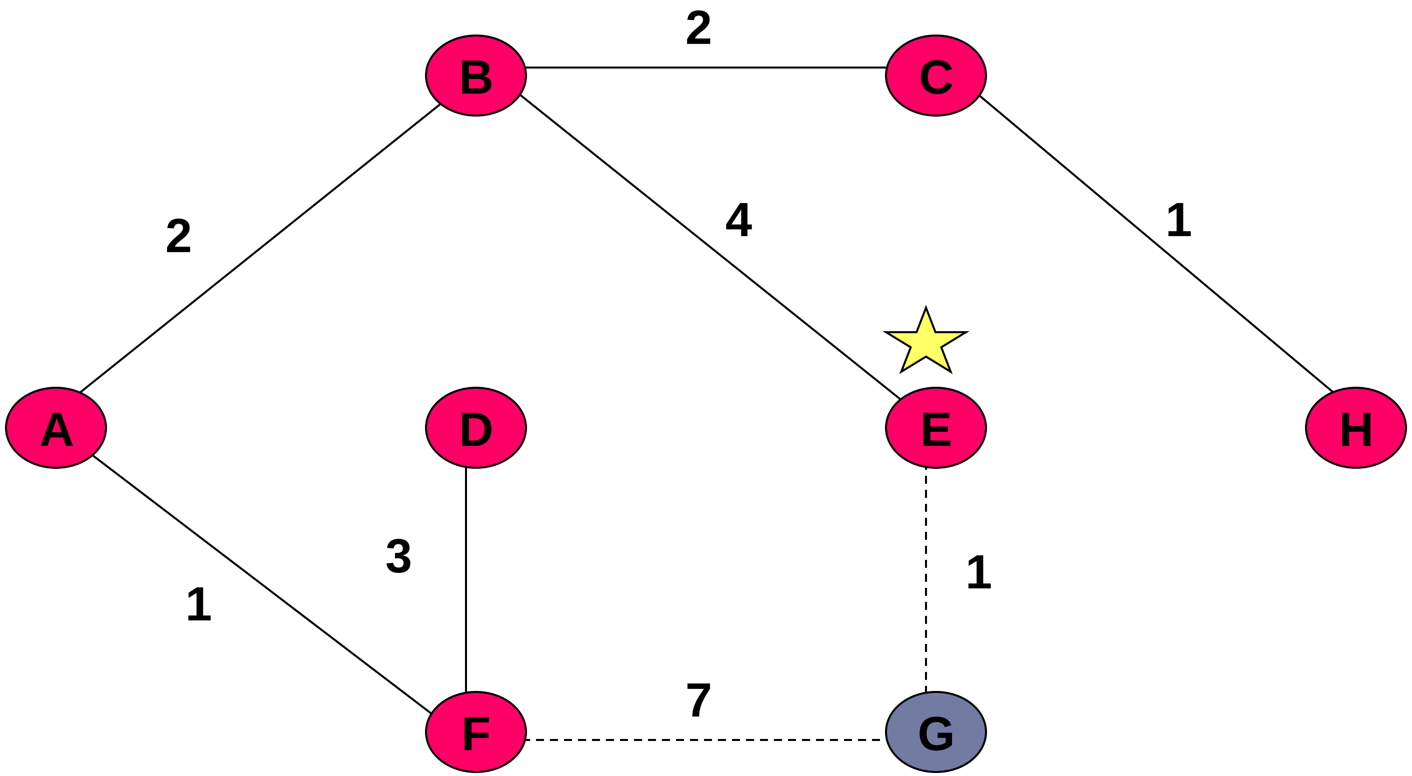
	V_T	d	pre
A	X	0	
B	X	2	A
C	X	4	B
D	X	4	F
E		6	B
F	X	1	A
G		8	F
H		5	C



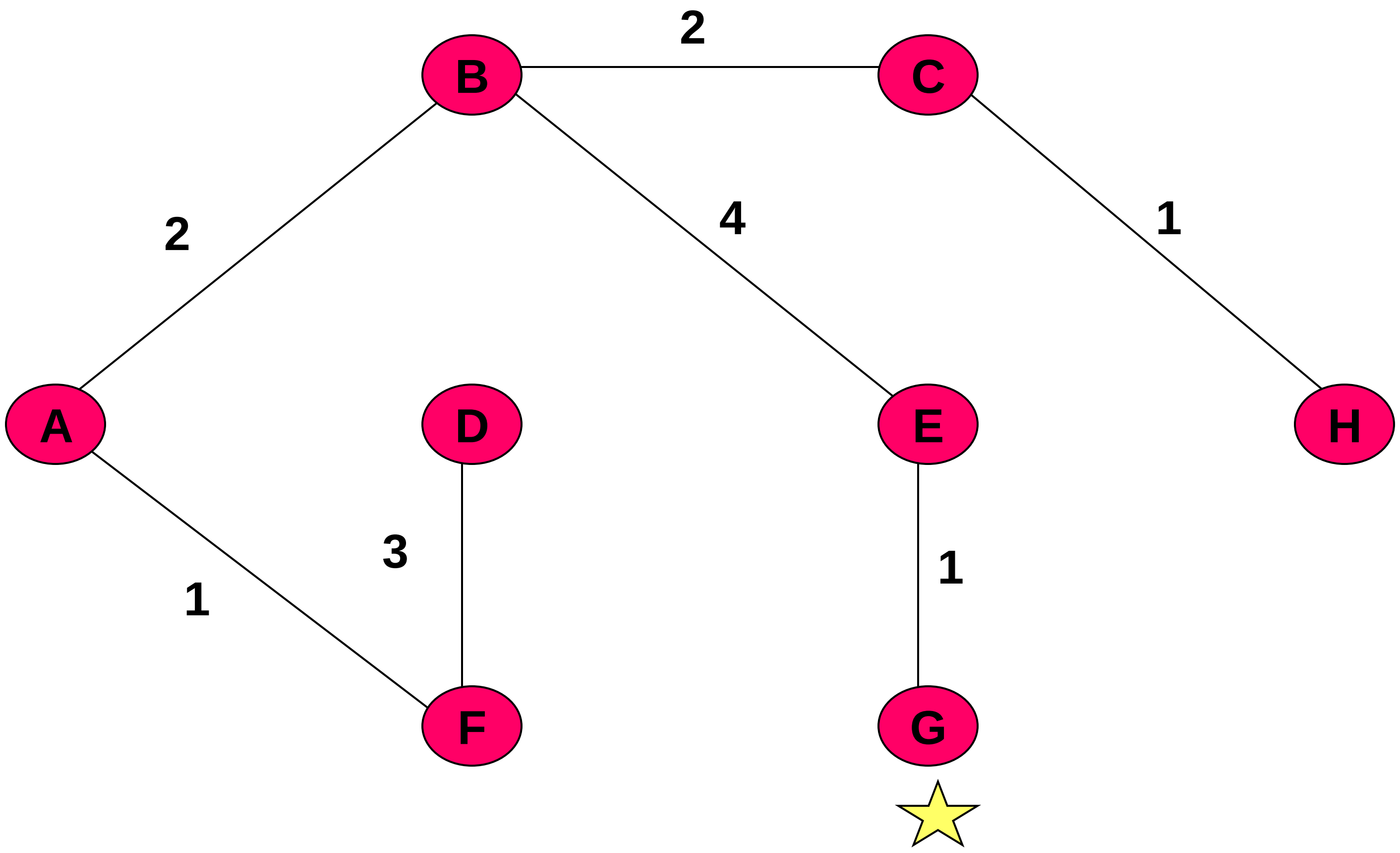
	V_T	d	pre
A	X	0	
B	X	2	A
C	X	4	B
D	X	4	F
E		6	B
F	X	1	A
G		8	F
H	X	5	C



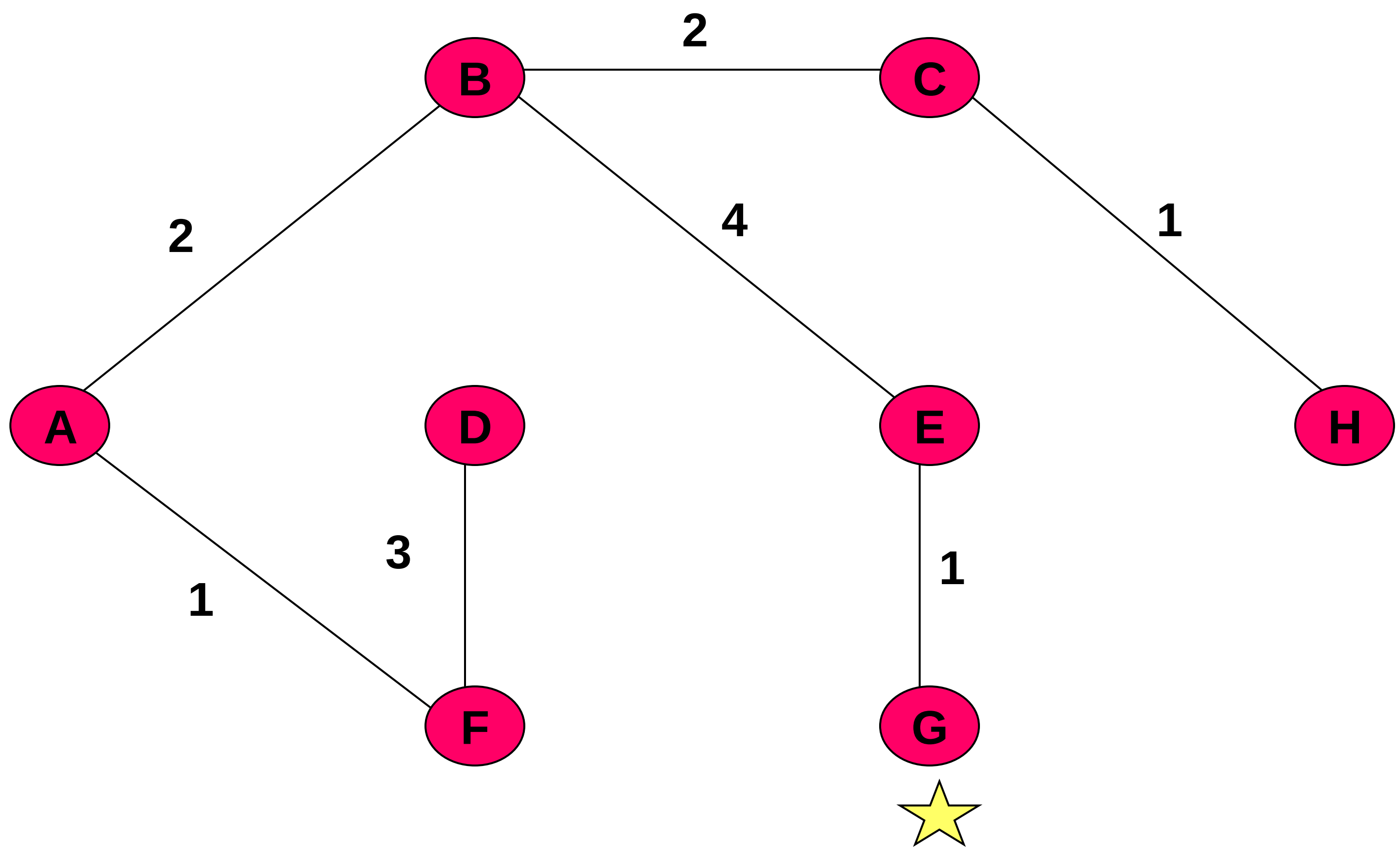
	V_T	d	pre
A	X	0	
B	X	2	A
C	X	4	B
D	X	4	F
E	X	6	B
F	X	1	A
G		7	E
H	X	5	C



	V_T	d	pre
A	X	0	
B	X	2	A
C	X	4	B
D	X	4	F
E	X	6	B
F	X	1	A
G	X	7	E
H	X	5	C



	V_T	d	pre
A	X	0	
B	X	2	A
C	X	4	B
D	X	4	F
E	X	6	B
F	X	1	A
G	X	7	E
H	X	5	C



The shortest path from A $\rightarrow$ G: G $\leftarrow$ E $\leftarrow$ B $\leftarrow$ A (7)

Dijkstra's algorithm

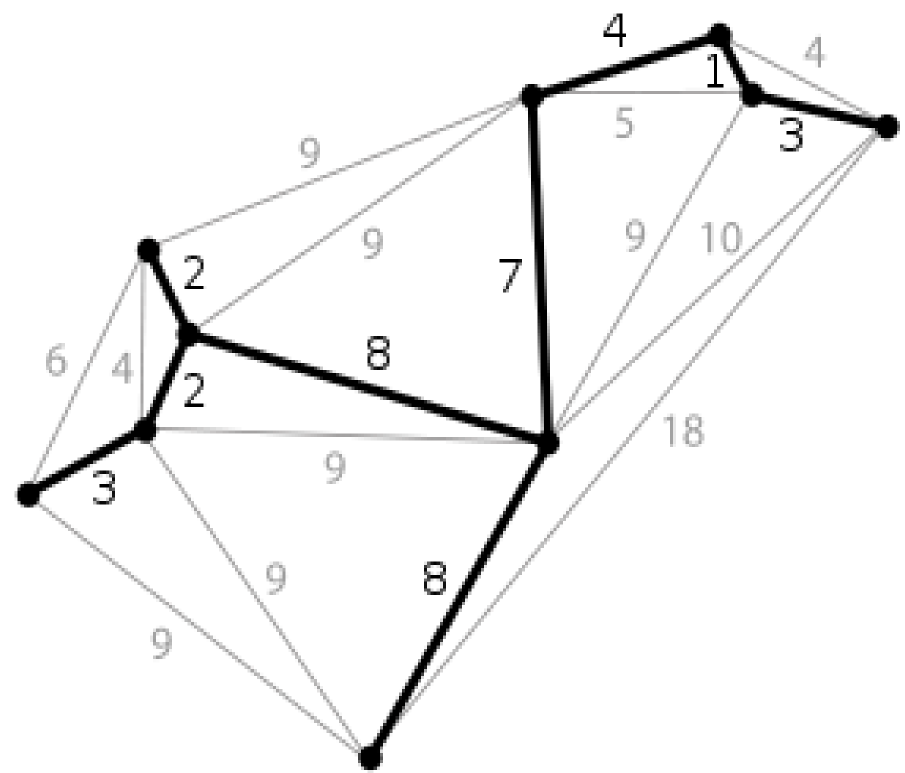
- Computational Complexity: $O(V^2)$
- Cannot be used for graphs with negative weights

Applications

- Packet routing in computer networks
- Vehicle routing in traffic networks
- Social networks – degree of separation – friendship relationships

Minimum spanning tree - MST

- Finding a low-cost tree connecting a set of nodes.
- Minimal total weighting for its edges.
- A graph with *n* vertices will have a spanning tree with *n*-1 edges.
- Prim's and Kruskal's algorithm

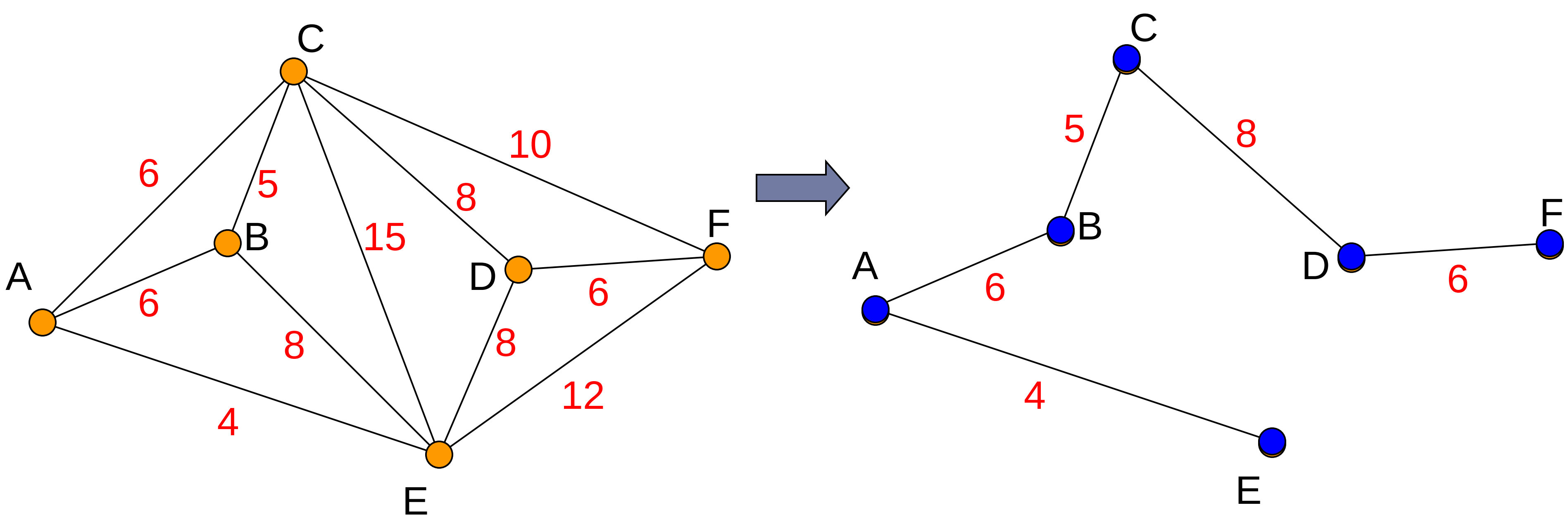


Prim's algorithm

1. $T = \emptyset$
2. Randomly select a vertex and add this vertex to T.
3. If every vertex of G is in T, then stop. Otherwise, go to step 4.
4. Find an edge which
   i) connects a vertex $\in$ T to a vertex $\notin$ T, and
   ii) has minimal weight.

   Add this edge to T and go back to step 2.

Prim's algorithm - Example



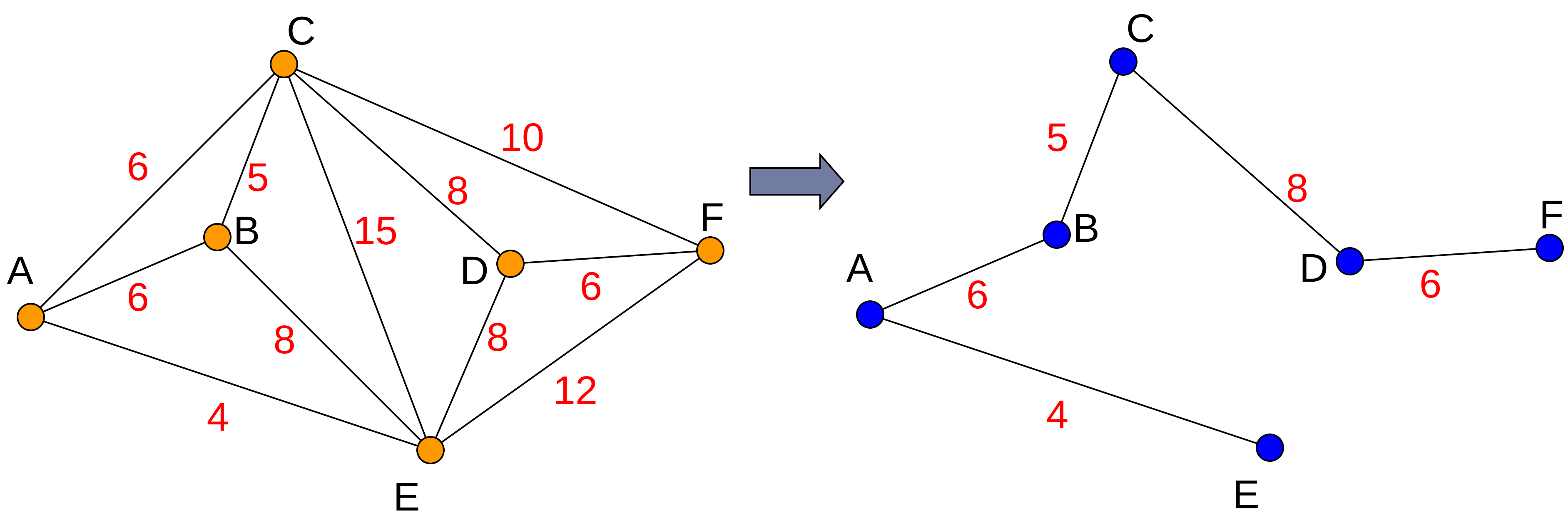
$$\sum weights = 29$$

Kruskal's algorithm

1. $T = (V, E_T)$ với $E_T = \emptyset$.
2. If T is connected*, then stop. Otherwise, go to Step 3.
3. Select an edge $\notin E_T$ with minimum weight such that this edge does not create any cycles in T when it is added into T. Go back to Step 2.

*A graph is connected if there always exists routes between any pair of nodes

Kruskal's algorithm - example



$$\sum weights = 29$$