

CSE 100

Minimum Spanning Trees

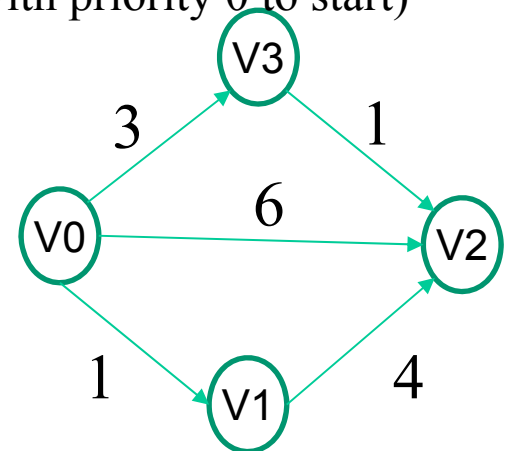
Prim's and Kruskal

Your Turn

The array of vertices, which include dist, prev, and done fields (initialize dist to 'INFINITY' and done to 'false'):

V0:	dist=	prev=	done=	adj: (v1,1), (v2,6), (v3,3)
V1:	dist=	prev=	done=	adj: (v2,4)
V2:	dist=	prev=	done=	adj:
V3:	dist=	prev=	done=	adj: (v2,1)

The priority queue (set start vertex dist=0, prev=-1, and insert it with priority 0 to start)

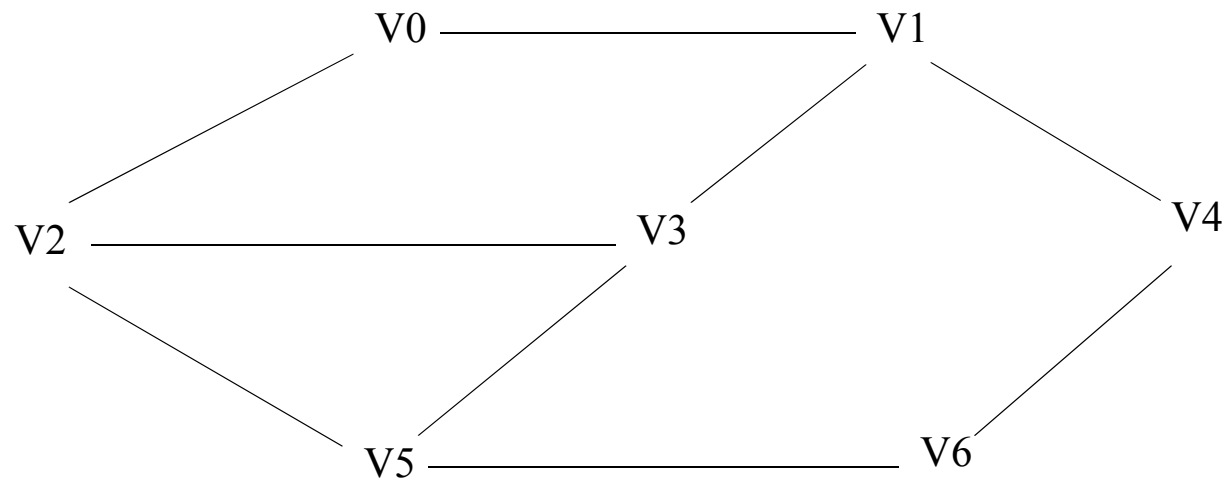


Spanning trees

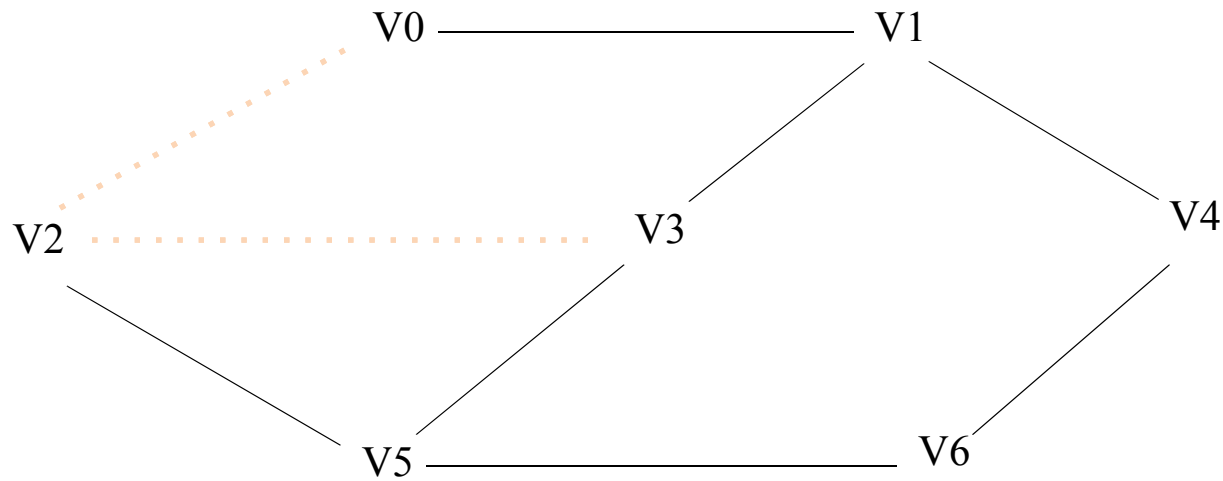
- We will consider spanning trees for *undirected* graphs
- A spanning tree of an undirected graph G is an undirected graph that...
 - contains all the vertices of G
 - contains only edges of G
 - has no cycles
 - is connected
- A spanning tree is called “spanning” because it connects all the graph’s vertices
- A spanning tree is called a “tree” because it has no cycles (recall the definition of *cycle* for undirected graphs)
- What is the root of the spanning tree?
 - you could pick any vertex as the root; the vertices adjacent to that one are then the children of the root; etc.

Spanning trees: examples

- Consider this undirected graph G:



Spanning tree? Ex. 1

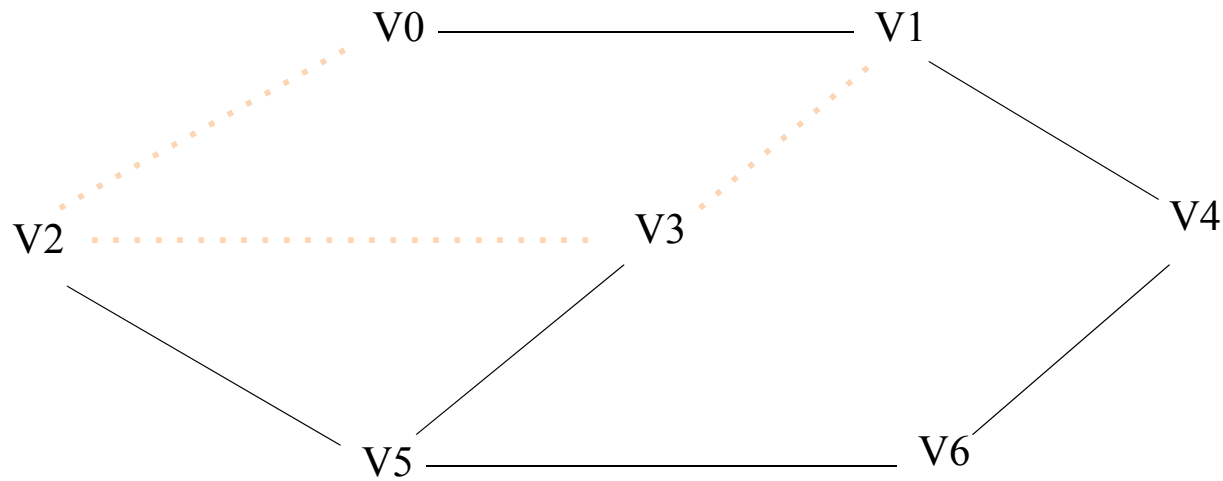


Is this graph a spanning tree of G?

A. Yes

☒ B. No

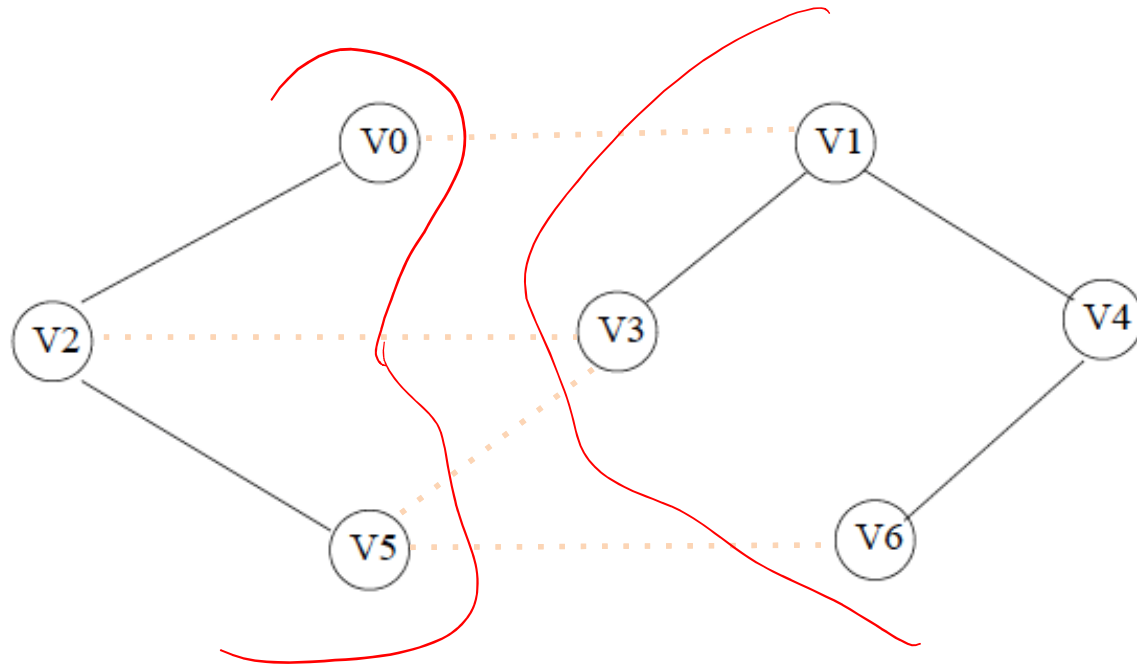
Spanning tree? Ex. 2



Is this graph a spanning tree of G?

- A. Yes
- B. No

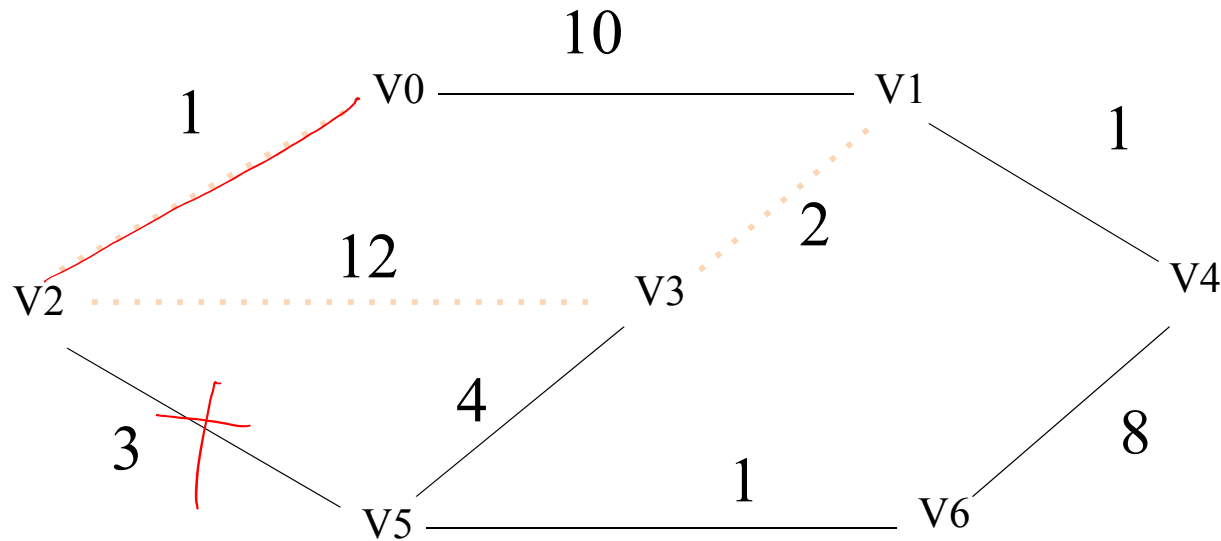
Spanning tree? Ex. 3



Is this graph a spanning tree of G?

- A. Yes
- ☒ B. No

Minimum Spanning tree: Spanning tree with minimum total cost



Is this graph a minimum spanning tree of G ?

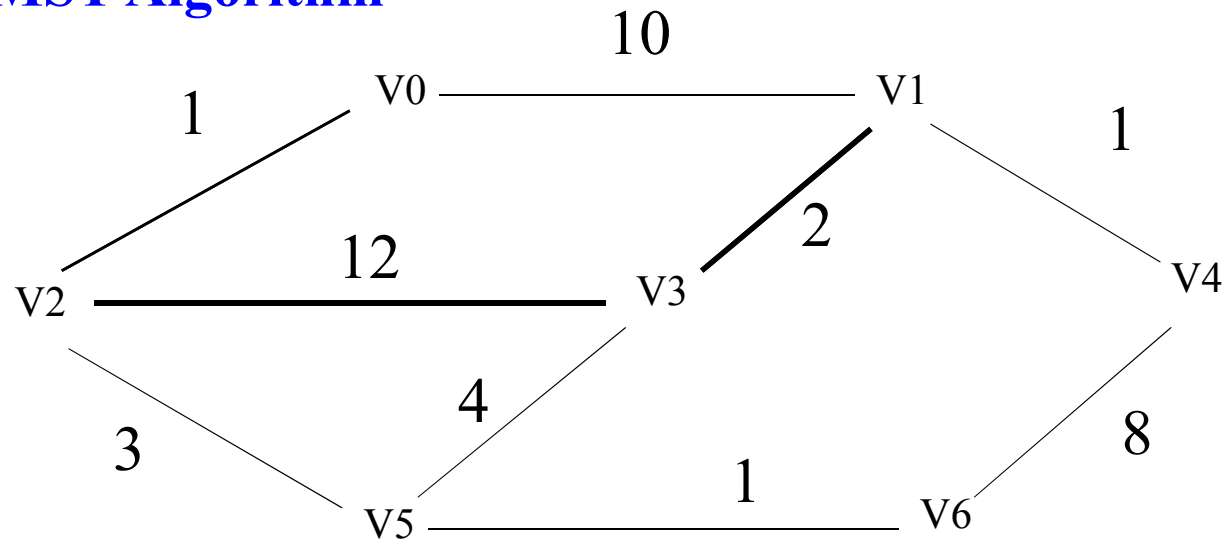
A. Yes

☒ B. No

Minimum spanning trees in a weighted graph

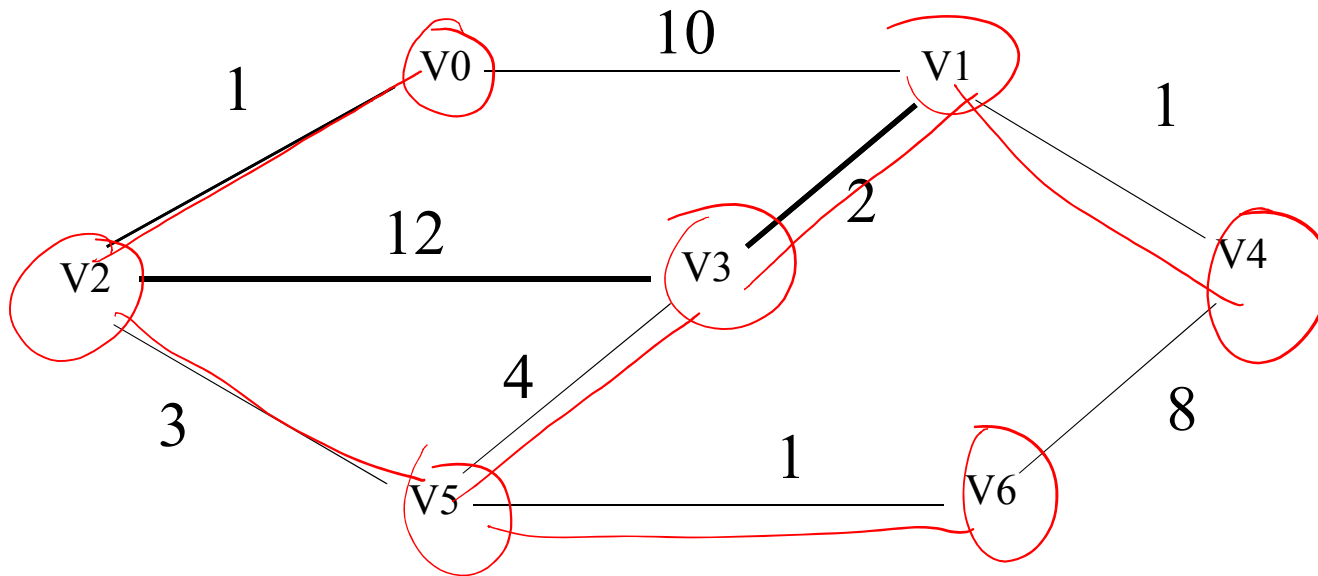
- A single graph can have many different spanning trees
- They all must have the same *number* of edges, but if it is a weighted graph, they may differ in the *total weight* of their edges
- Of all spanning trees in a weighted graph, one with the least total weight is a *minimum* spanning tree (MST)
- It can be useful to find a minimum spanning tree for a graph: this is the least-cost version of the graph that is still connected, i.e. that has a path between every pair of vertices
- How to do it?

Prim's MST Algorithm



- Start with any vertex and grow like a mold, one edge at a time
- Each iteration choose the cheapest crossing edge

Fast implementation of Prim's MST Algorithm



- Iteration 1

Finding a minimum spanning tree: Prim's algorithm

- As we know, minimum weight paths from a start vertex can be found using Dijkstra's algorithm
- At each stage, Dijkstra's algorithm extends the best path from the start vertex (priority queue ordered by total path cost) by adding edges to it
- To build a minimum spanning tree, you can modify Dijkstra's algorithm slightly to get Prim's algorithm
- At each stage, Prim's algorithm adds the edge that has the least cost from any vertex in the spanning tree being built so far (priority queue ordered by single edge cost)
- Like Dijkstra's algorithm, Prim's algorithm has worst-case time cost $O(|E| \log |V|)$
- We will look at another algorithm: Kruskal's algorithm, which also is a simple greedy algorithm
- Kruskal's has the same big-O worst case time cost as Prim's, but in practice it can be made to run faster than Prim's, if efficient supporting data structures are used

Fast Implementation of Prim's MST Algorithm

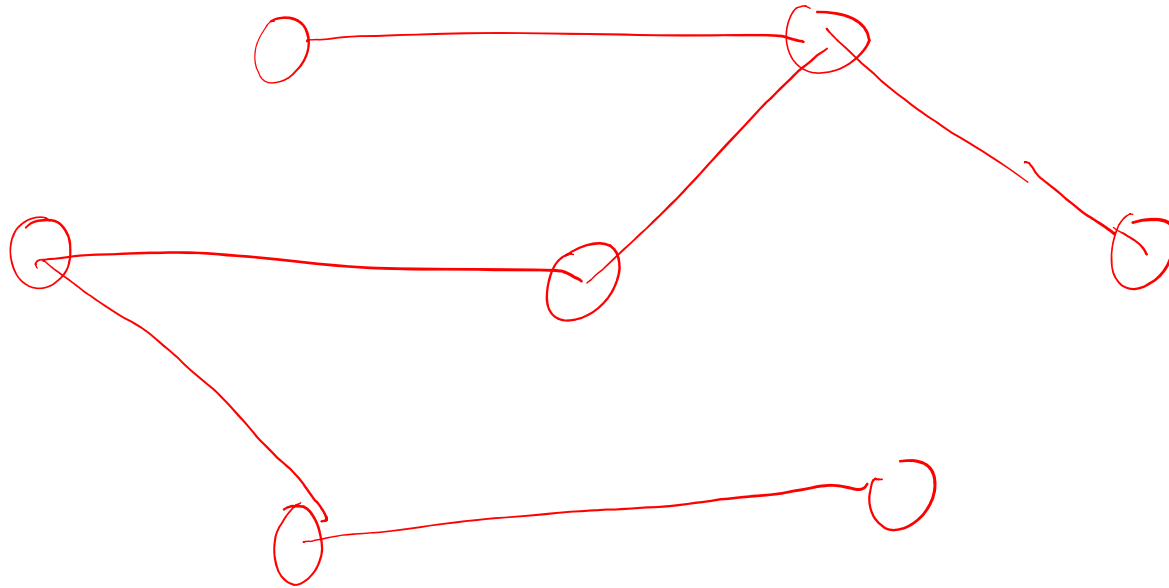
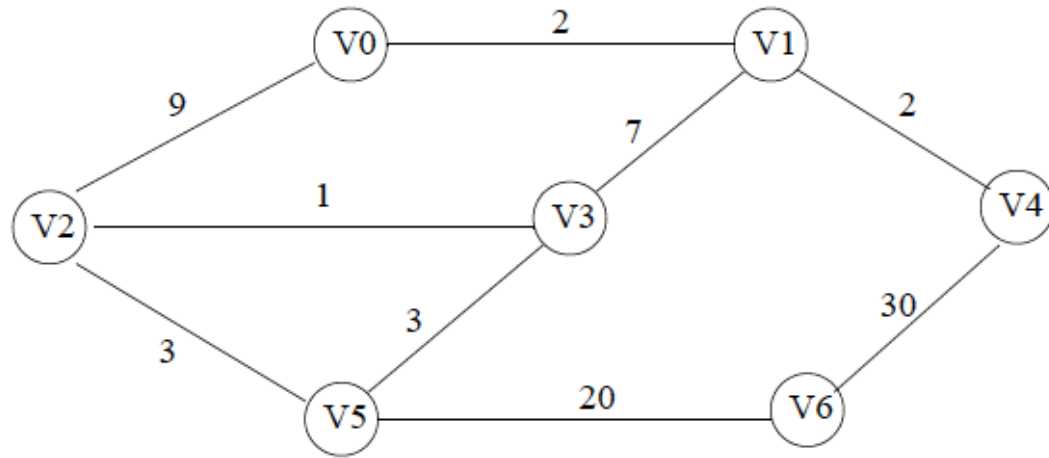
1. Initialize the vertex vector for the graph. Set all “done” fields to false. Pick an arbitrary start vertex s . Set its “prev” field to -1, and its “done” field to true. Iterate through the adjacency list of s , and put those edges in the priority queue.
 2. Is the priority queue empty? Done!
 3. Remove from the priority queue the edge (v, w, cost) with the smallest cost.
 4. Is the “done” field of the vertex w marked true?
If Yes: this edge connects two vertices already connected in the spanning tree, and we cannot use it. Go to 1.
 5. Accept the edge: Mark the “done” field of vertex w true, and set the “prev” field of w to indicate v .
 6. Iterate through w 's adjacency list, putting each edge in the priority queue.
 7. Go to 1.
- The resulting spanning tree is then implicit in the values of “prev” fields in the vertex objects.

Weighted minimum spanning tree: Kruskal's algorithm

- Prim's algorithm starts with a single vertex, and grows it by adding edges until the MST is built:
- Kruskal's algorithm starts with a forest of single-node trees (one for each vertex in the graph) and joins them together by adding edges until the MST is built;

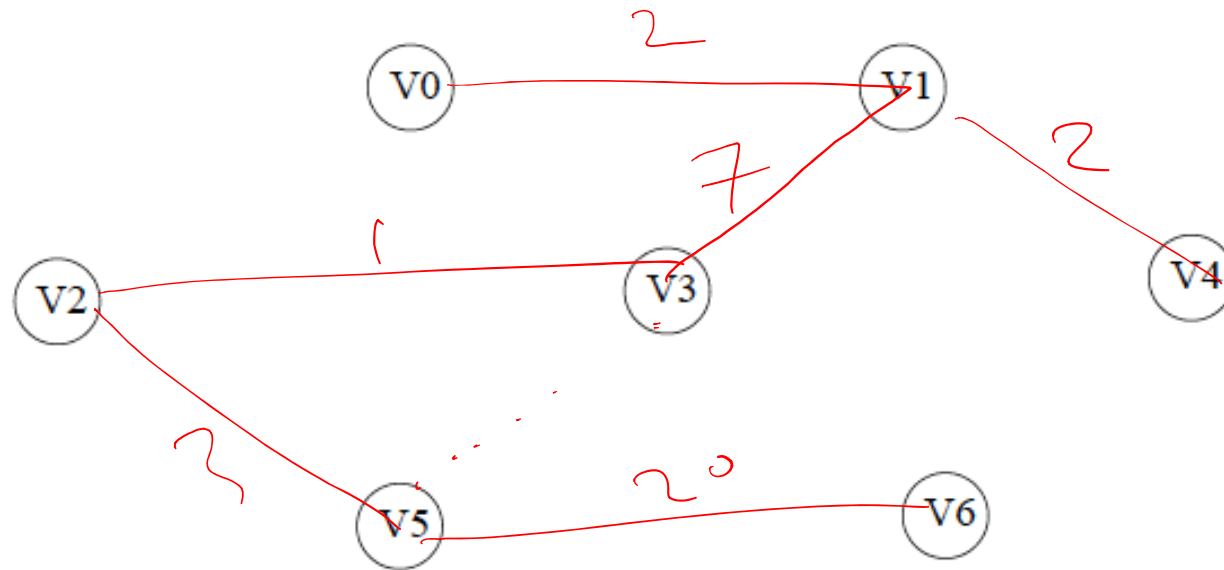
Why Kruskal?

Kruskal's algorithm:



Kruskal's algorithm: output

- Show the result here:



- What is the total cost of this spanning tree? 35
- Is there another spanning tree with lower cost? With equal cost? yes

Naïve Implementation of Kruskal's algorithm:

1. Sort edges in increasing order of cost
2. Set of edges in MST, $T = \{\}$
3. For $i = 1$ to $|E|$

 If $T \cup \{e_i\}$ has no cycles

 Add e_i to T

Ref: Tim Roughgarden (stanford)

Running Time of Naïve Implementation of Kruskal's algorithm:

1. Sort edges in increasing order of cost
2. Set of edges in MST, $T = \{\}$
3. For $i = 1$ to $|E|$

If $T \cup \{e_i\}$ has no cycles

Add e_i to T

DFS or
BFS

Towards a fast implementation for Kruskal's Algorithm

- What is the work that we are repeatedly doing with Kruskal's Algo?

Towards a fast implementation for Kruskal's Algorithm

- What is the work that we are repeatedly doing with Kruskal's Algo?
 - Checking for cycles: Linear with BFS, DFS
 - Union-Find Data structure allows us to do this in constant time! (Next time)

Graph algorithm time costs: a summary

- The graph algorithms we have studied have fast algorithms:
 - Find shortest path in unweighted graphs
 - Solved by basic breadth-first search: $O(|V|+|E|)$ worst case
 - Find shortest path in weighted graphs
 - Solved by Dijkstra's algorithm: $O(|E| \log|V|)$ worst case
 - Find minimum-cost spanning tree in weighted graphs
 - Solved by Prim's or Kruskal's algorithm: $O(|E| \log|V|)$ worst case
- The “greedy” algorithms used for solving these problems have polynomial time cost functions in the worst case
 - since $|E| \leq |V|^2$, Dijkstra's, Prim's and Kruskal's algorithms are $O(|V|^3)$
- As a result, these problems can be solved in a reasonable amount of time, even for large graphs; they are considered to be ‘tractable’ problems
- However, many graph problems do not have any known polynomial time solutions...!