CS 6212 DESIGN AND ANALYSIS OF ALGORITHMS

LECTURE: THE GREEDY METHOD – PART II

Instructor: Abdou Youssef

OBJECTIVES OF THIS LECTURE (PART A)

By the end of Part A of this lecture, you will be able to:

- Prove the optimality of the greedy solution of the Minimum Spanning Tree (MST) problem
- Prove the optimality of the greedy solution of the Single-Source Shortest Paths (SSSP) problem

OUTLINE (OF PART A)

- Review of the MST greedy algorithm
- Proof of optimality of the greedy solution of the MST problem
- Review of the SSSP greedy algorithm
- Proof of optimality of the greedy solution of the SSSP problem

KRUSKAL'S GREEDY MST ALGORITHM

- **Procedure** ComputeMST(**in**:$W[1{:}n,1{:}n]$;**out**:T)   // non-edges (i,j): $W[i,j] = \infty$
- **begin**
  - Put in T all the n nodes and no edges;
  - **while** T has less than n-1 edges **do**
    - Select a min-weight edge e out of the remaining edges e;
    - Delete e from the graph;
    - **if** (e does not create a cycle in T) **then**
      - Add e to T;
    - **endif**
-     **endwhile**
- **end** ComputeMST

ILLUSTRATION OF THE GREEDY MST ALGORITHM

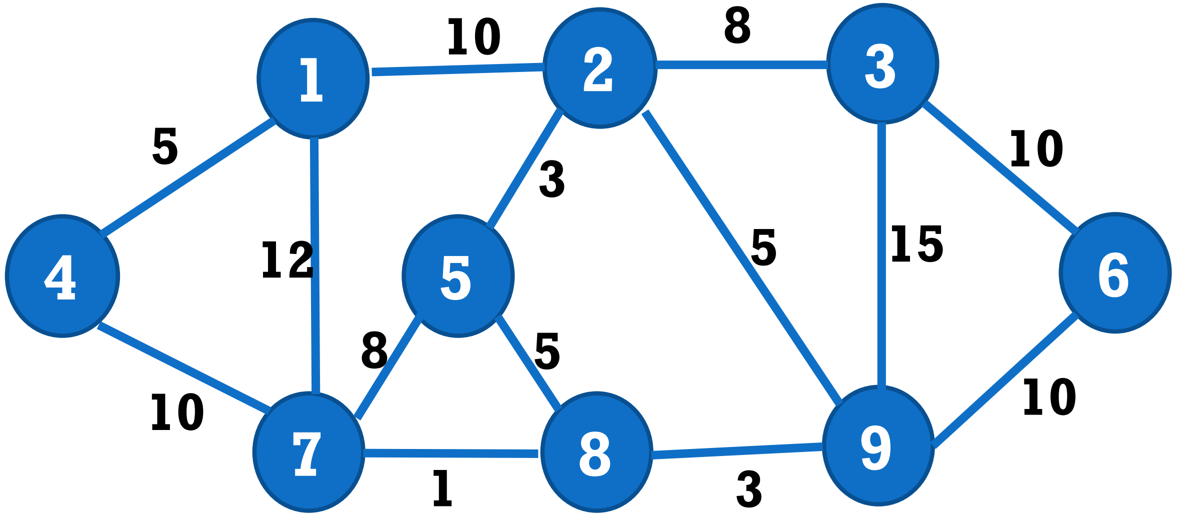
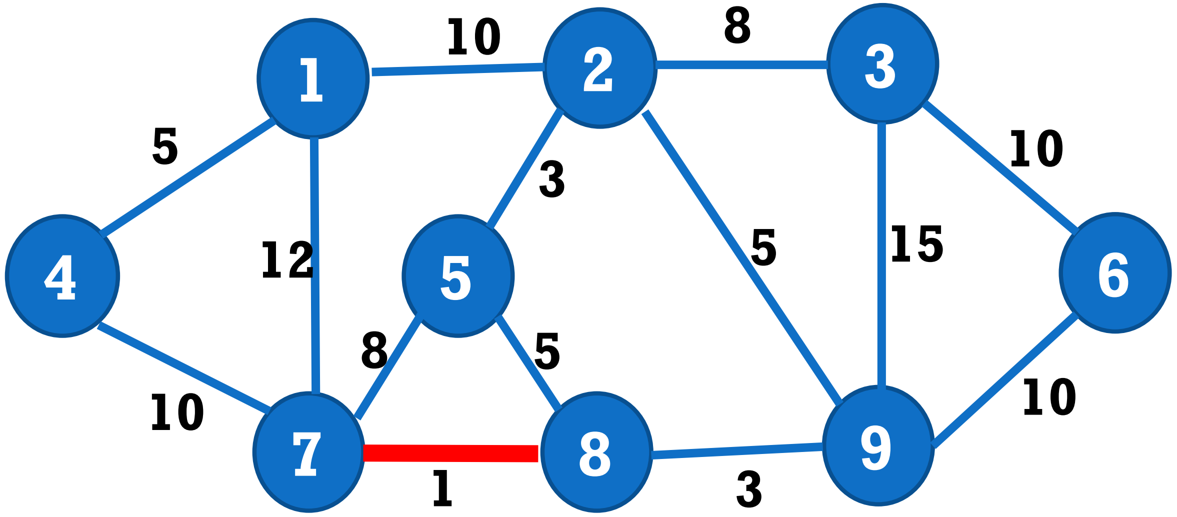
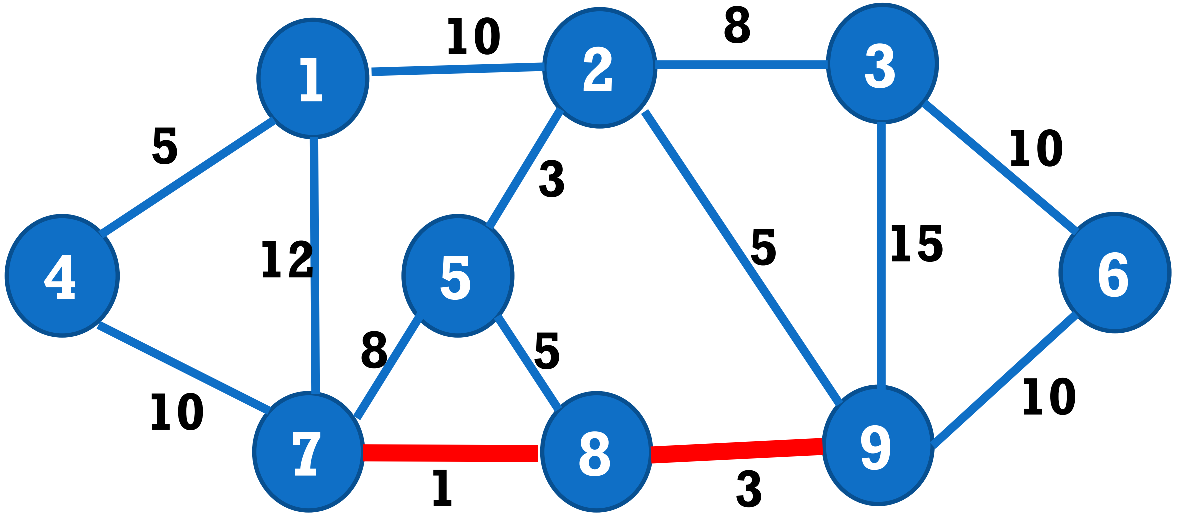


ILLUSTRATION OF THE GREEDY MST ALGORITHM



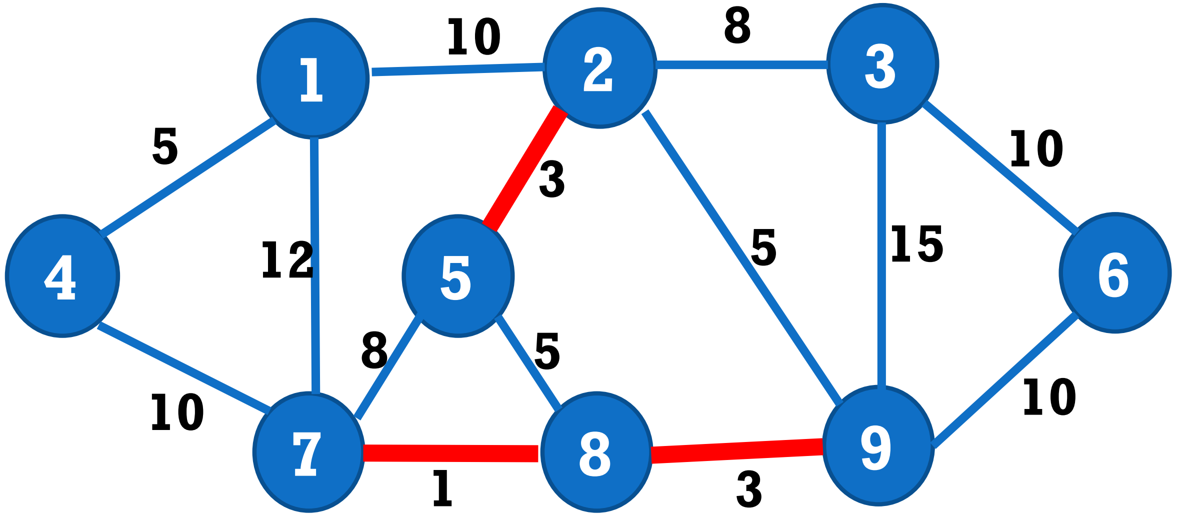
- Min edge: (7,8). No cycle => OK to add

ILLUSTRATION OF THE GREEDY MST ALGORITHM



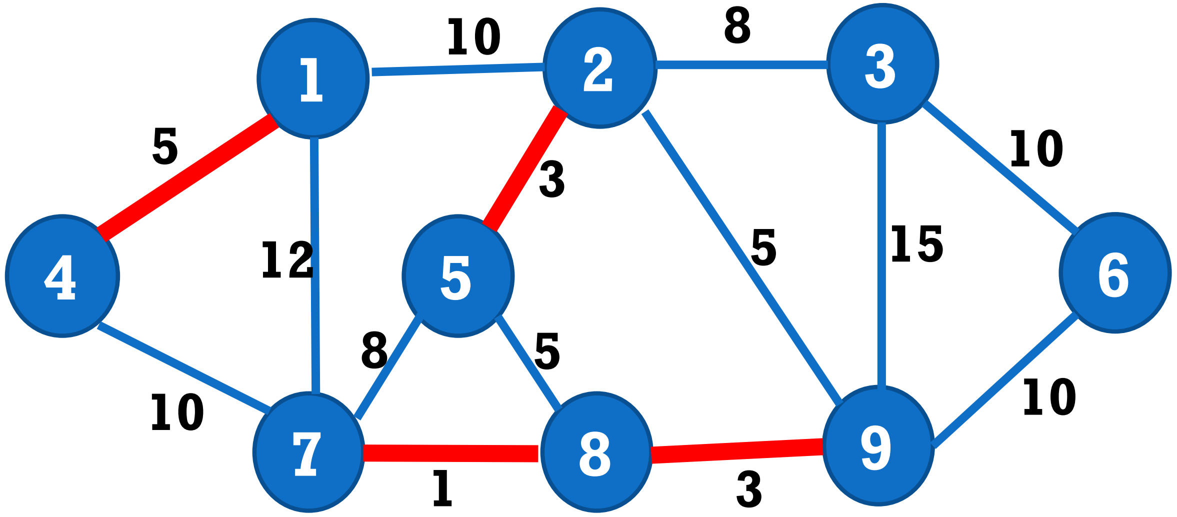
- Min edge: (8,9), (2,5). Pick (8,9). No cycle => OK to add

ILLUSTRATION OF THE GREEDY MST ALGORITHM



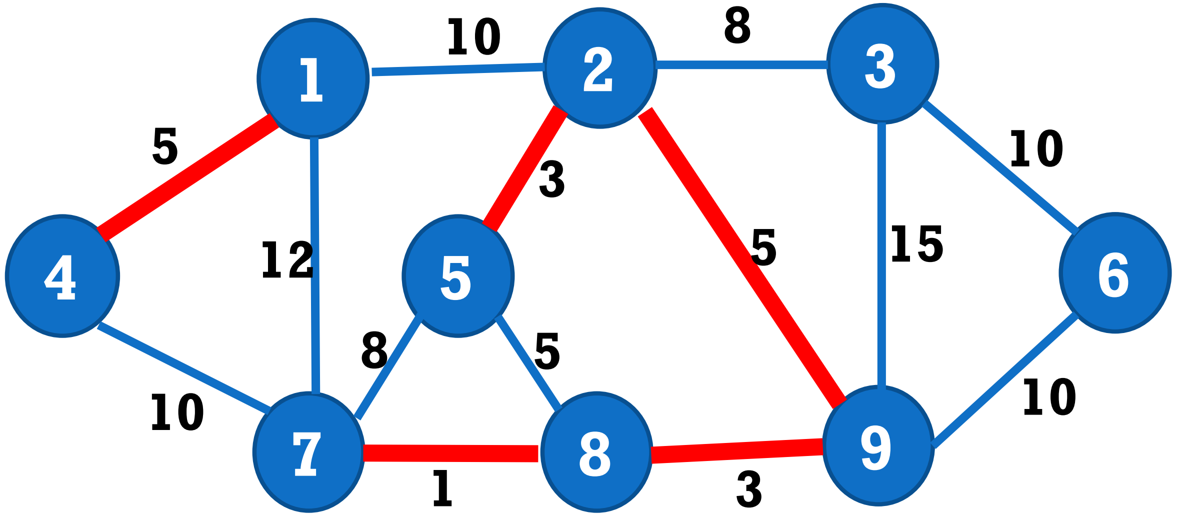
- Min edge: (2,5). No cycle => OK to add

ILLUSTRATION OF THE GREEDY MST ALGORITHM



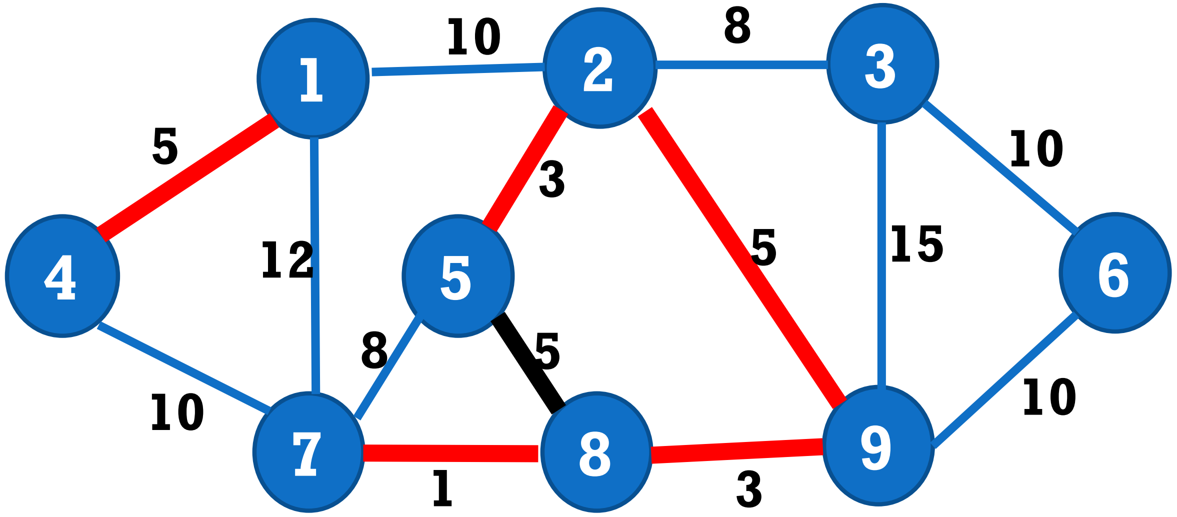
- Min edge: (1,4), (2,5), (5,8). Pick (1,4): No cycle => OK to add

ILLUSTRATION OF THE GREEDY MST ALGORITHM



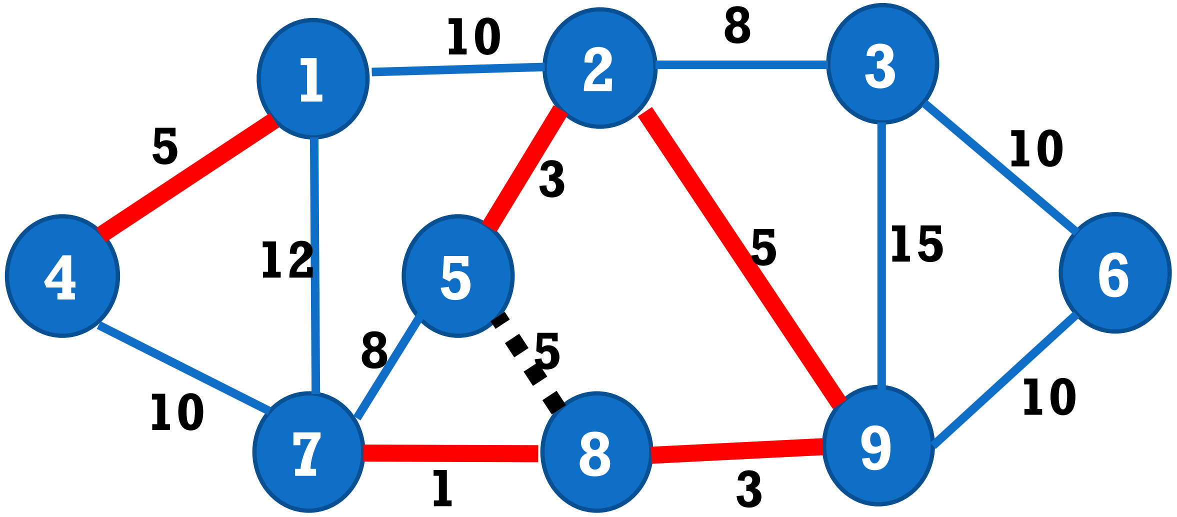
- Min edge: (2,5), (5,8). Pick (2,5). No cycle => OK to add

ILLUSTRATION OF THE GREEDY MST ALGORITHM



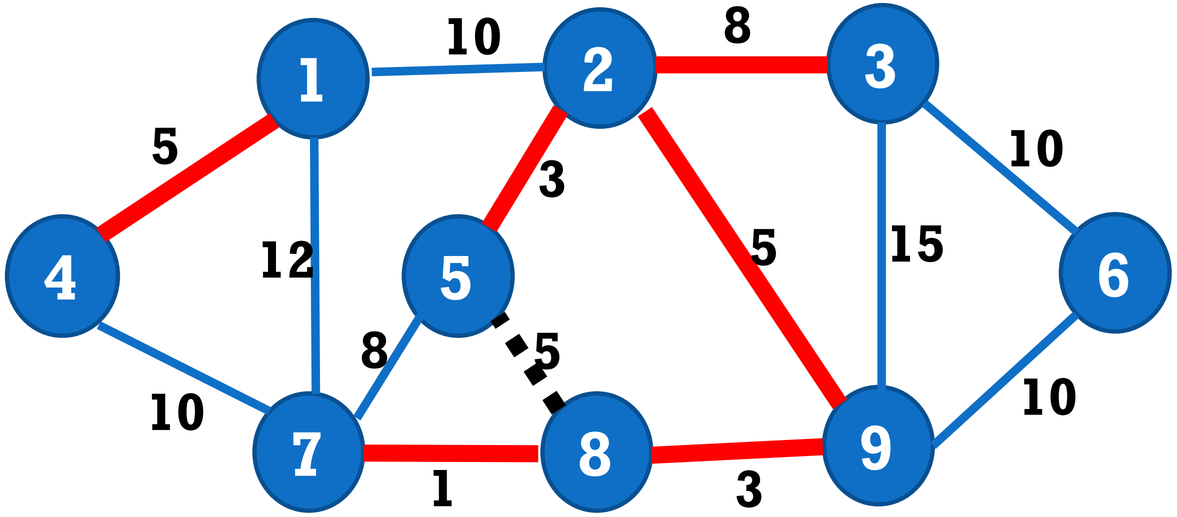
- Min edge: (5,8). Creates cycle

ILLUSTRATION OF THE GREEDY MST ALGORITHM



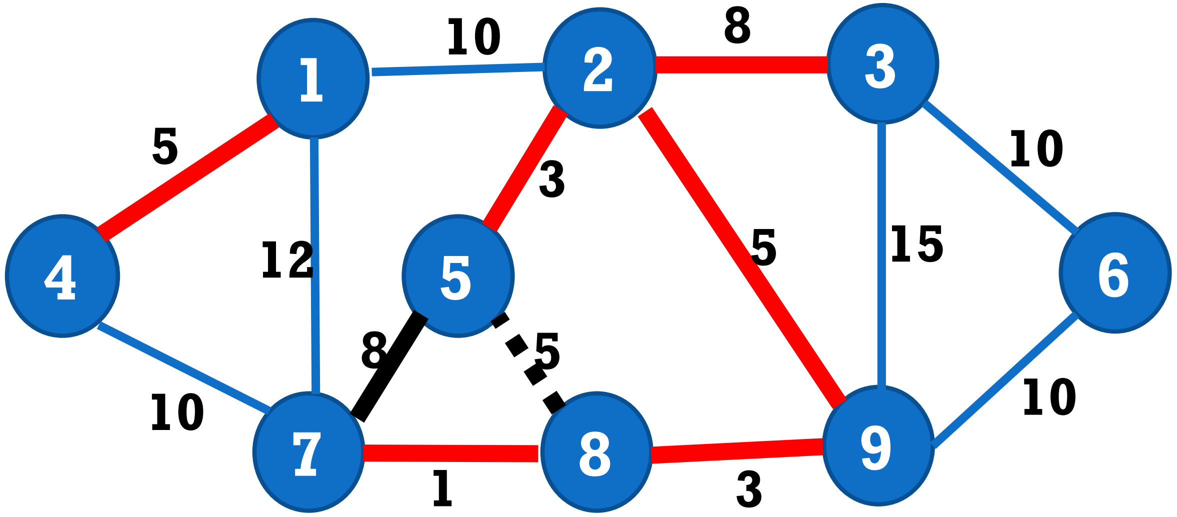
- Min edge: (5,8). Creates cycle => throw it

ILLUSTRATION OF THE GREEDY MST ALGORITHM



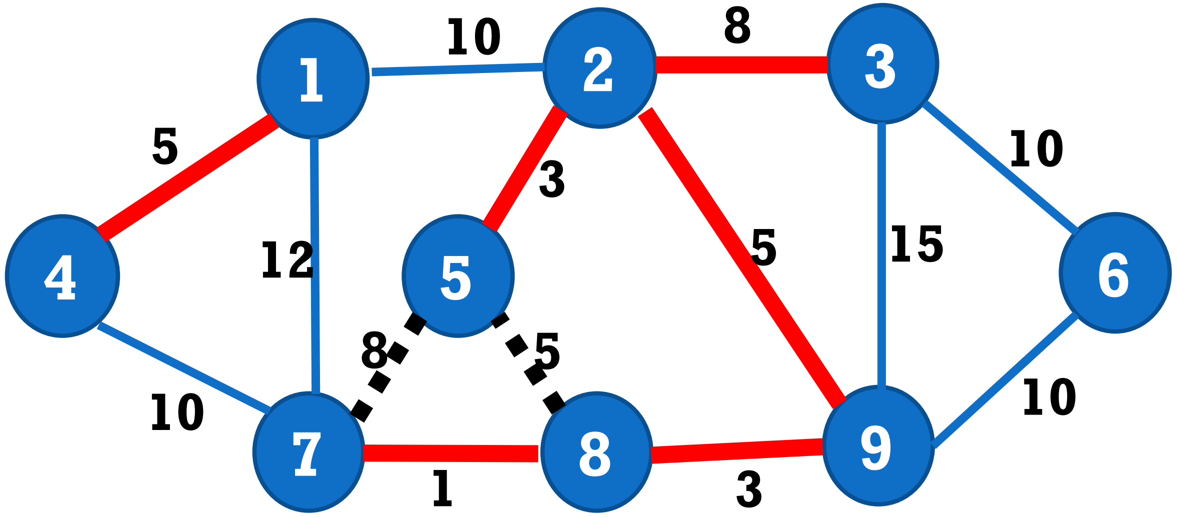
- Min edge: (2,3) and (5,7). Pick (2,3): No cycle => OK to add

ILLUSTRATION OF THE GREEDY MST ALGORITHM



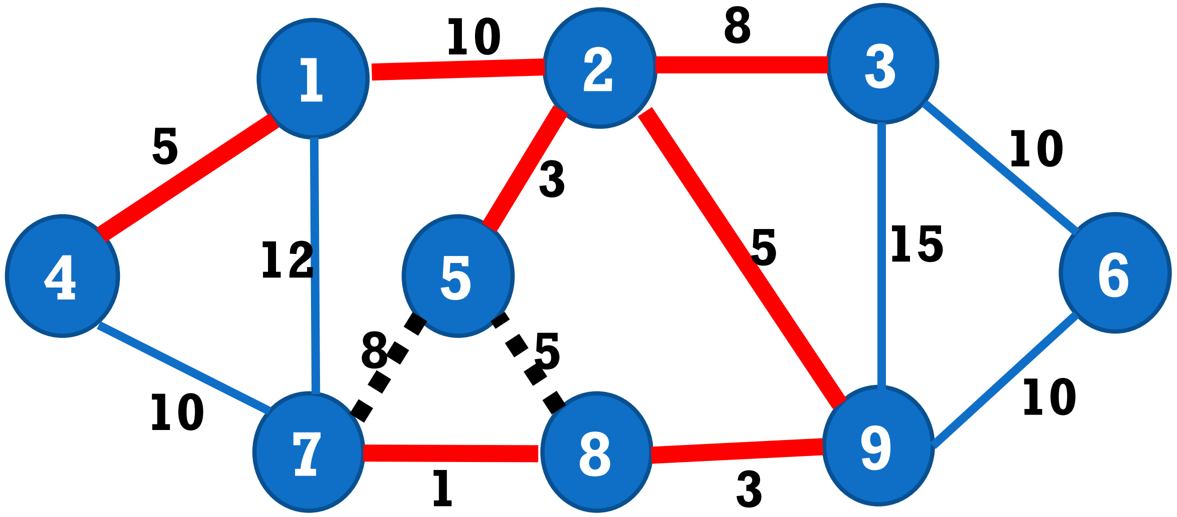
- Min edge: (5,7). Creates cycle

ILLUSTRATION OF THE GREEDY MST ALGORITHM



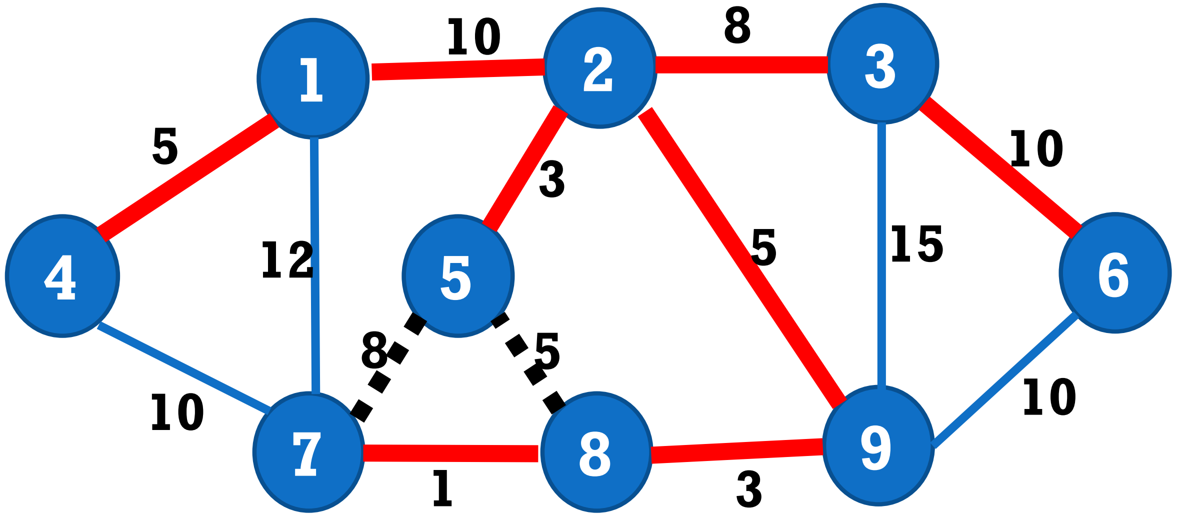
- Min edge: (5,7). Creates cycle => throw it

ILLUSTRATION OF THE GREEDY MST ALGORITHM



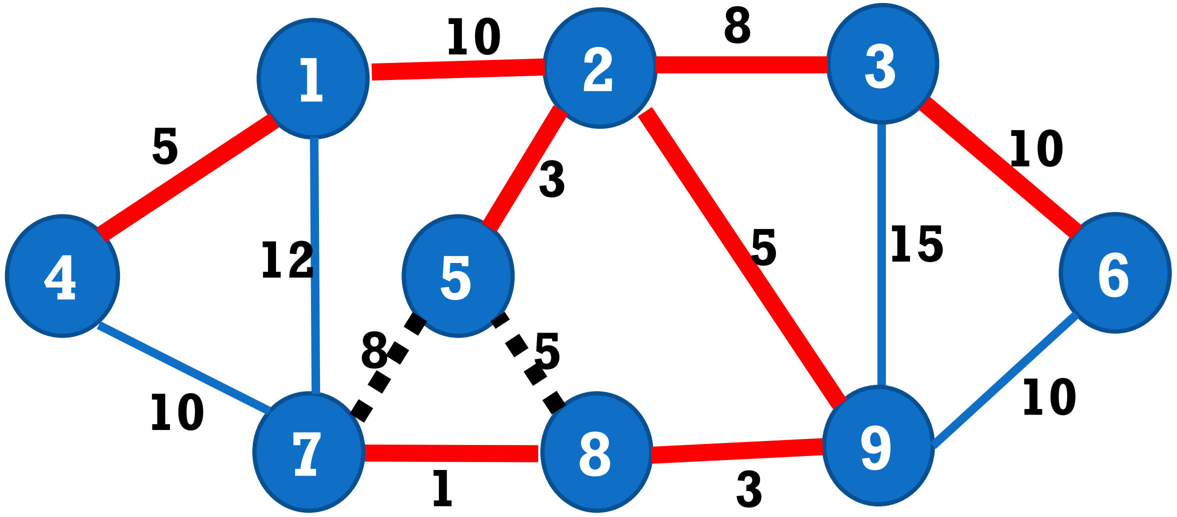
- Min edge: (1,2), (3,10), (4,7), 6,9). Pick (1,2): No cycle => OK to add

ILLUSTRATION OF THE GREEDY MST ALGORITHM



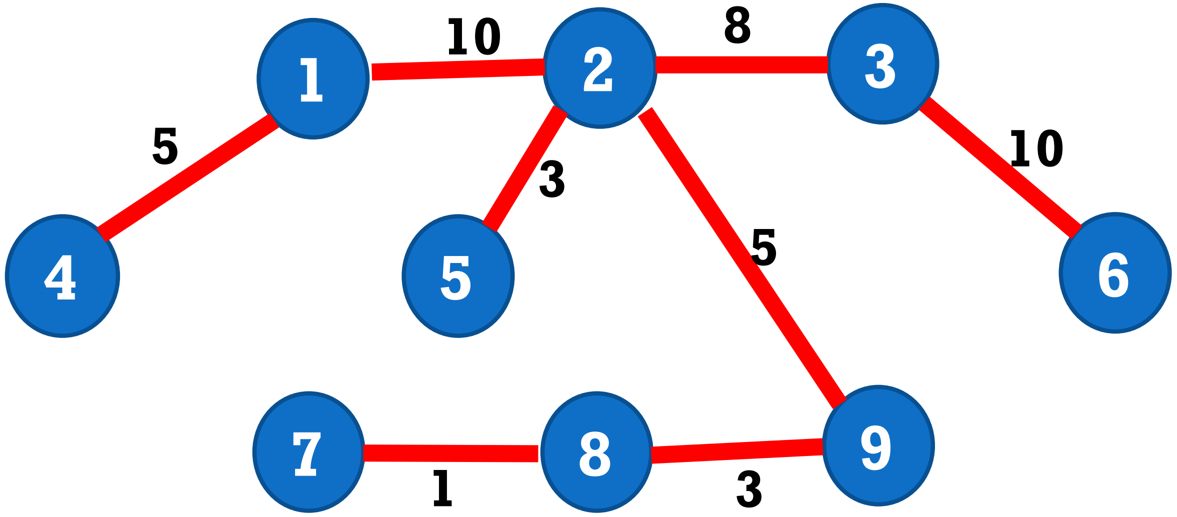
- Min edge: (3,10), (4,7), (6,9). Pick (3,10): No cycle => OK to add

ILLUSTRATION OF THE GREEDY MST ALGORITHM



- Tree completed (got 8 edges)

ILLUSTRATION OF THE GREEDY MST ALGORITHM



- This is the spanning tree produced by the greedy algorithm

PROOF OF OPTIMALITY (1/6)

Theorem: The greedy ComputeMST algorithm computes a mininum spanning tree.

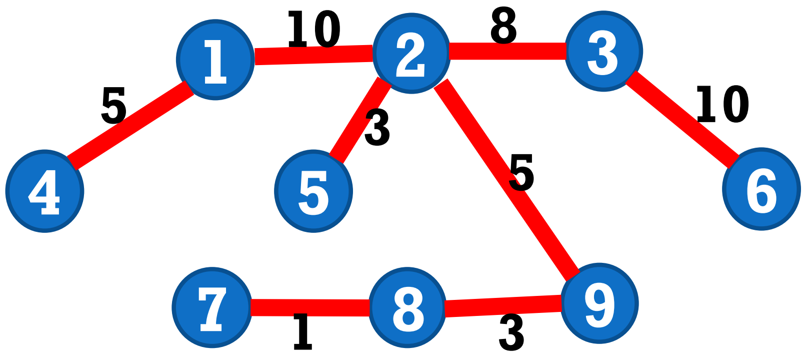
Proof:

- Let T be the tree generated by the algorithm
- Let T' be a minimum spanning tree
- We need to prove that T is a minimum spanning tree, i.e., $W\ (T) = W(T')$
- If $T = T'$, done. So assume that $T \neq T'$.
- **Strategy**: T' will be transformed to T without weight change:
    - Substitute a carefully selected edge $e \in T - T'$ (i.e., in T but not in T') for an edge $e' \in T' - T$, without changing the weight of T'.
    - This makes T' resemble T more
    - This transform is repeated several times until T' becomes identical to T without weight change.
    - This will show that $W(\text{initial } T') = W(\text{final transformed } T') = W(T)$, i.e. $W(\text{initial } T') = W(T)$
    - Which implies that T is a minimum spanning tree

PROOF OF OPTIMALITY (2/6)

Proof (Continued):

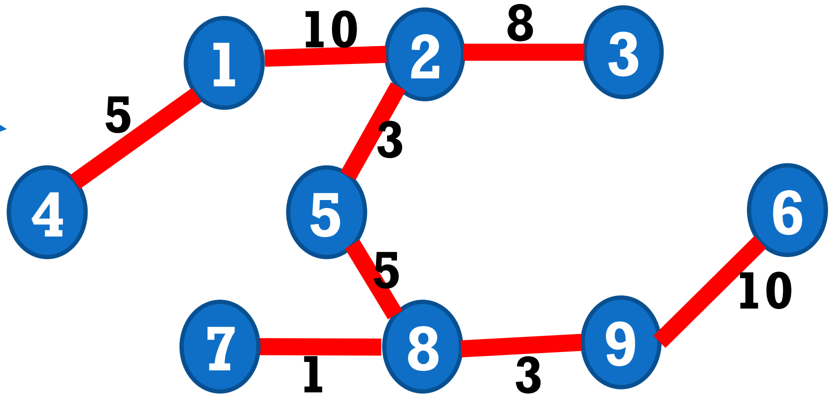
T



Example (just to illustrate the steps of the proof for understandability):

- $T - T' = \{(3,6), (2,9)\}$
- $T' - T = \{(6,9), (5,8)\}$
- $T \cap T' = \{(1,4), (1,2), (2,3), (2,5), (7,8), (8,9)\}$

T'



PROOF OF OPTIMALITY (3/6)

1. Let $e = \text{min\_weight\_edge\_of}(T - T')$
2. Add e to T' (temporarily).
3. This creates a cycle $e_1 (= e), e_2, \ldots, e_k$; **$e_2, \ldots, e_k$ are in T'**
4. This cycle must have an edge $e_j \notin T$ b/c T has no cycles
5. e_j can't be e_1 because $e_1 = e$, which is in T
6. Thus, e_j must be one of $e_2, \ldots, e_k$, and so is in T'
7. Take $e' = e_j \in T' - T$
8. Transform T': $T'' = T' \cup \{e\} - \{e'\}$

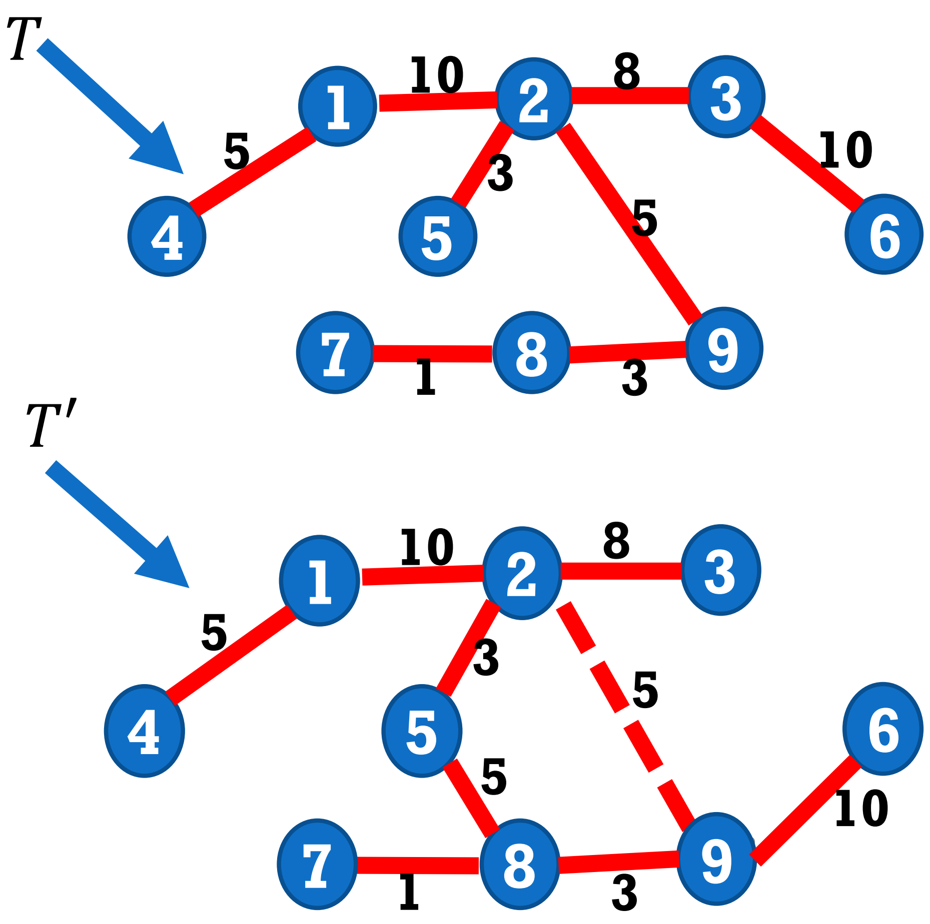
All edges in T that are < W(e) are also in T'

Ex: $e = (2,9)$. edges in T that are < W(e) are $\{(7,8), (8,9), (2,5)\}$, which are also in T'

Blue inserts are for the example

- $e = \text{min\_weigh\_edge\_of}(T - T') = \min(\{(3,6), (2,9)\}) = (2,9)$
- Add e to T' temporarily. This creates a cycle: (2,9) (9,8) (8,5) (5,2)
- Edge (8,5) in that cycle is in T' but not in T
- So, we can take $e' = (8,5)$
- Note: W(e)=W(e') Coincidence?

- $T - T' = \{(3,6), (2,9)\}$
- $T' - T = \{(6,9), (5,8)\}$
- $T \cap T' = \{(1,4), (1,2), (2,3), (2,5), (7,8), (8,9)\}$
- $T \cap T'' = \{(1,4), (1,2), (2,3), (2,5), (7,8), (8,9), (8,5)\}$



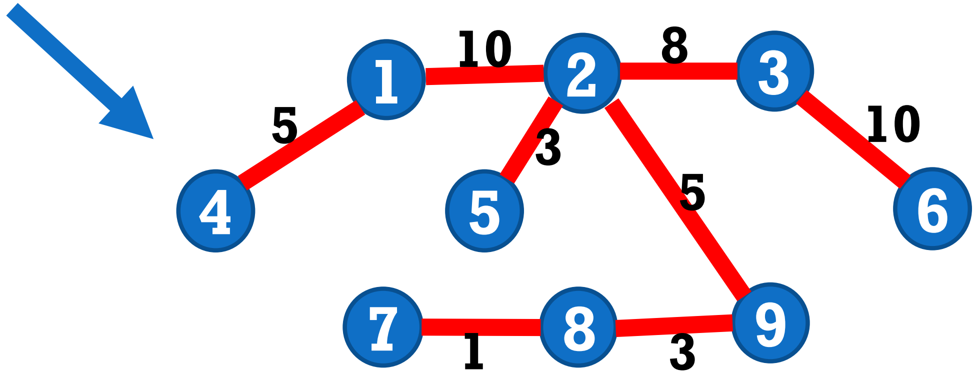
PROOF OF OPTIMALITY (4/6)

Proof (Continued)

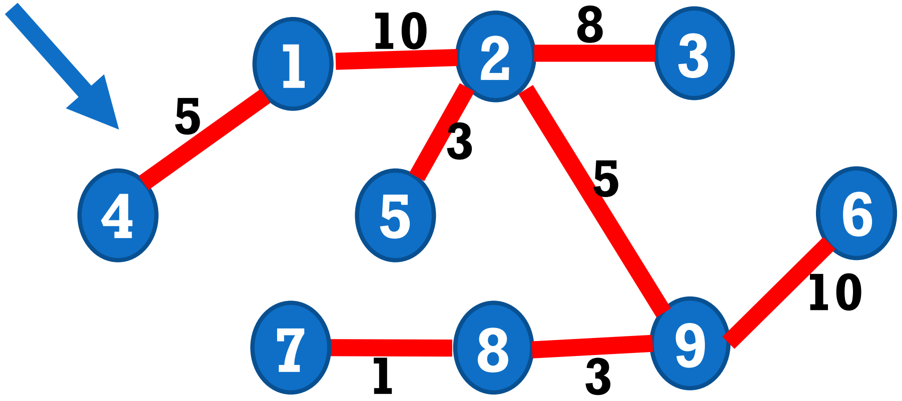
Example:

- $e = (2,9)$, $e' = (8,5)$
- $T'' = T' \cup \{e\} - \{e'\}$

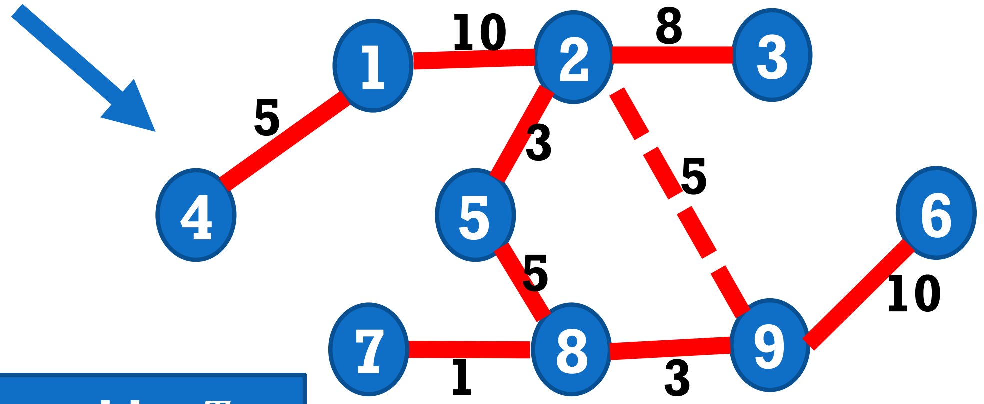
T



T''



T'



Notice how T'' resembles T more than T' resembles T

PROOF OF OPTIMALITY (5/6)

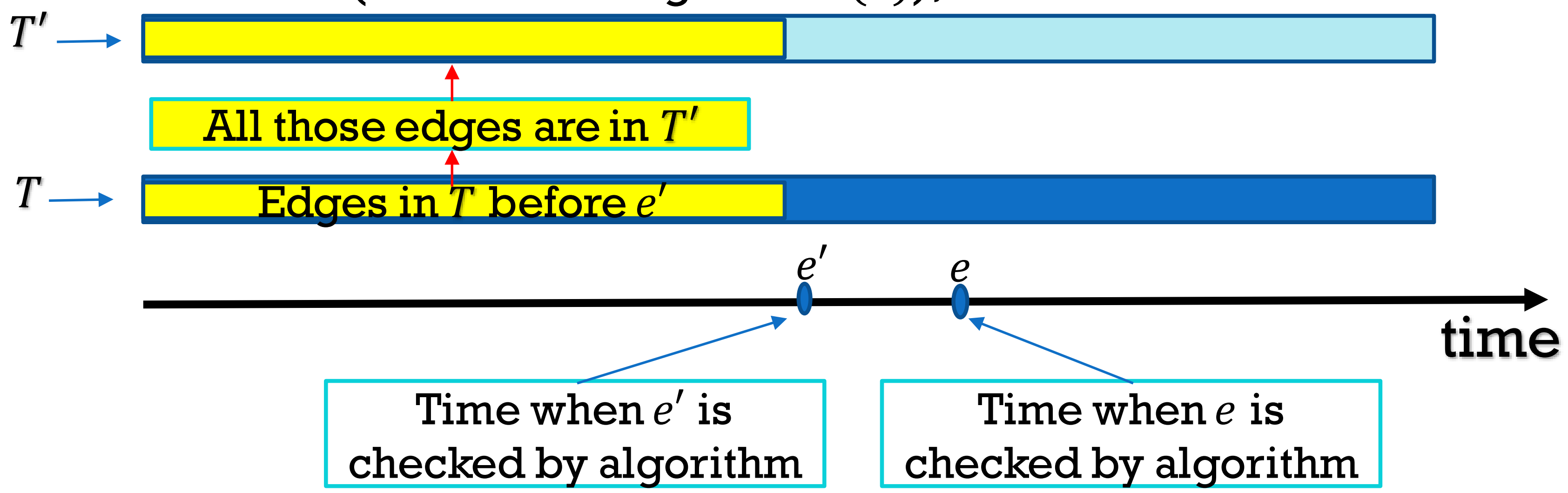
Proof (Continued):

Remember that → All edges in T that are < W($\boldsymbol{e}$) are also in T'

9. Claim: $W(e') \geq W(e)$

- We prove the claim by contradiction. Assume $W(e') < W(e)$
- $W(e') < W(e) \Rightarrow$ the greedy algorithm would process e' before e
- All the edges that are processed before e' (and so of weight < $W(e)$), and which were entered into T, are also T'

When algorithm checks e', it finds that adding e' to T would not create a cycle b/c at that time, $e'$and all prior edges in T are also in T', and T' has no cycles



- Thus, e' would have to be added by the algorithm to T, contradicting the fact that e' is not in T.
- Therefore, the claim $W(e') \geq W(e)$ must be true

PROOF OF OPTIMALITY (6/6)

Proof (Continued):

10. Since $T'' = T' \cup \{e\} - \{e'\}$,

    we have $W(T'') = W(T') + W(e) - W(e') \leq W(T')$

11. $W(T'') \leq W(T')$ and T' is minimum $\Rightarrow$ T'' is minimum and $W(T'') = W(T')$

12. This shows that the transform (from T' to T'') does not change the weight.

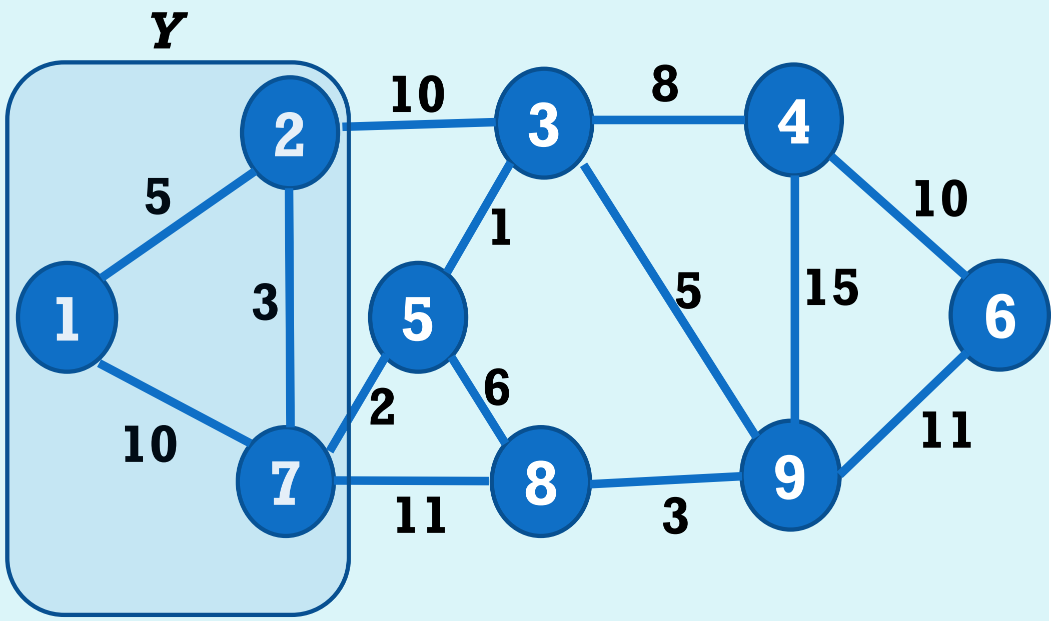
13. By our strategy laid out earlier, this transform can be repeated, yielding in the end a tree identical to T, and implying that T has the same weight as initial T', which is minimum

14. Therefore, T is a minimum spanning tree. Q.E.D.

GREEDY SINGLE-SOURCE SHORTEST PATHS
-- DISTANCE APPROXIMATIONS: SPECIAL PATHS --

- Let Y be a set := {s} initially
- **Definition**: A path from s to a node x outside Y is called ***special path*** if each intermediary node on the path belongs to Y.
- Let DIST[1:n] be:
  - DIST[i] = the length of the shortest special path from s to i
- **Greedy selection policy**: choose from outside Y the node of minimum DIST value, and add it to Y
- **Claim** (proved later) :

  $\forall i \in Y, DIST[i] = distance[i]$, that is, when a node i joins Y, its DIST is equal to its distance from s.



- Special paths:
  - 1, 2, 3 because 1 is source, and 1 and 2 are inside Y
  - 1, 2, 7, 5 b/c 1 is source and 1, 2, and 7 are inside Y
  - 1,5 (missing edge is an edge of weight ∞)
- Not Special paths: 1, 2, 3, 4 (b/c 3 is not in Y); 1,7,5,8 (Why?)

GREEDY SINGLE-SOURCE SHORTEST PATHS ALGORITHM

```
Procedure SSSP( in: W[1:n,1:n], s; out: DIST[1:n]);
begin
    for i =1 to n do:  DIST[i]  := W[s,i]; endfor
    // implement Y as Boolean array Y[1:n] :Y[i]= 1 if i ∈Y, 0 otherwise
    Boolean Y[1:n];        // initialized to 0
    Y[s] := 1;             // add s to set Y
    for num =2 to n do
        Select a node u from out of  Y (i.e., Y[u]==0) such that
          DIST[u]  = min {DIST[i]  | Y[i]  = 0};
        Y[u] := 1;         // Add u to Y
        // update the DIST values of the other nodes
        for all node v  where Y[v] = 0 do
            DIST[v]= min (DIST[v], DIST[u]+W[u,v]);
        endfor
    endfor
End SSSP
```

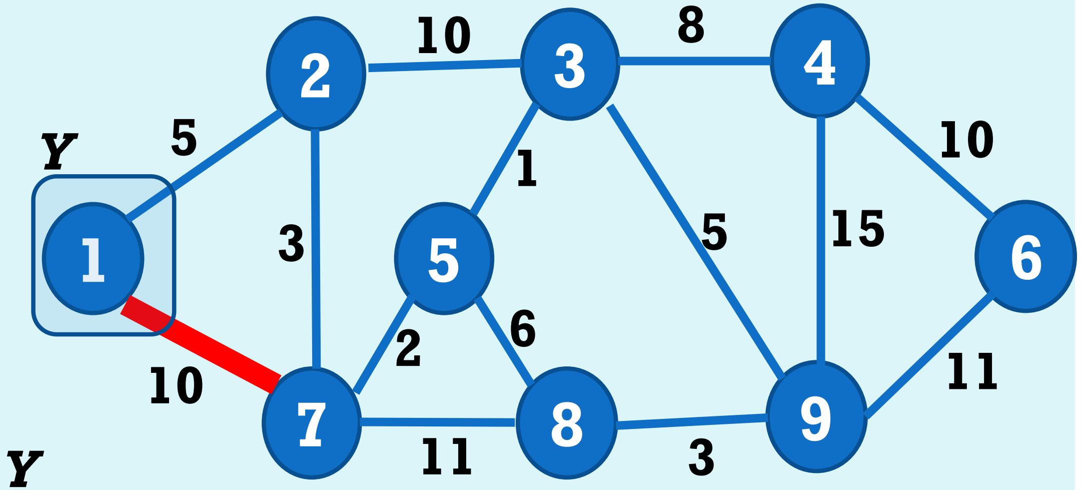
SPECIAL PATHS AND DIST
-- UPDATES EXAMPLE --

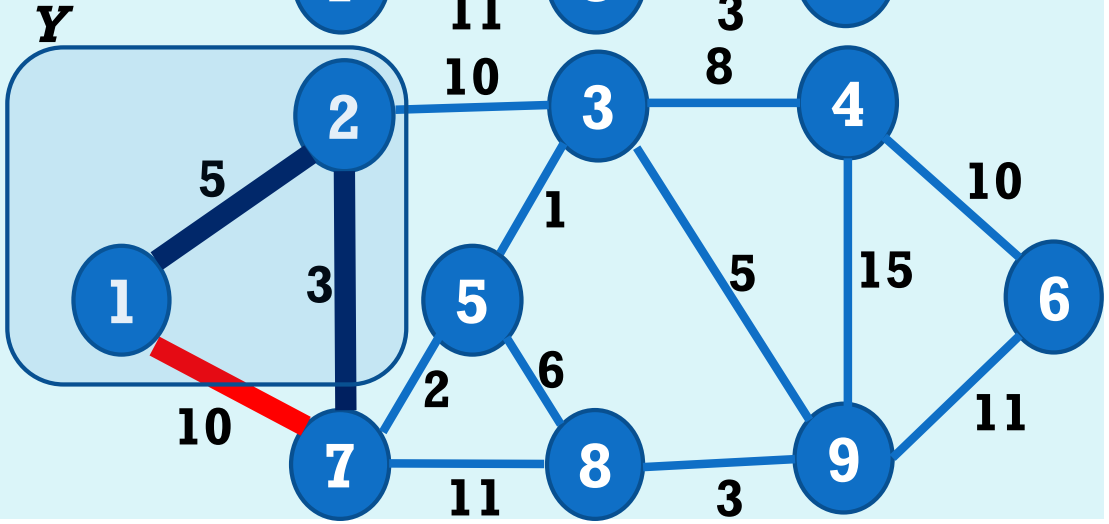
- $DIST[v] = \min(DIST[v],\ DIST[u] + W[u,v])$

i:	1	2	3	4	5	6	7	8	9
DIST[i]	0	5	∞	∞	∞	∞	10	∞	∞

- **Before update: DIST[7]=10**
- After update: DIST[7]= min(10, DIST[2]+W[2,7])= min(10, 5+3)=8.
- DIST[3]=min(∞, DIST[2]+W[2,3])=15

i:	1	2	3	4	5	6	7	8	9
DIST[i]	0	5	15	∞	∞	∞	8	∞	∞





OPTIMALITY OF THE GREEDY SSSP (1/3)

Theorem: When a node u enters Y, we have DIST[u] = distance(s,u).

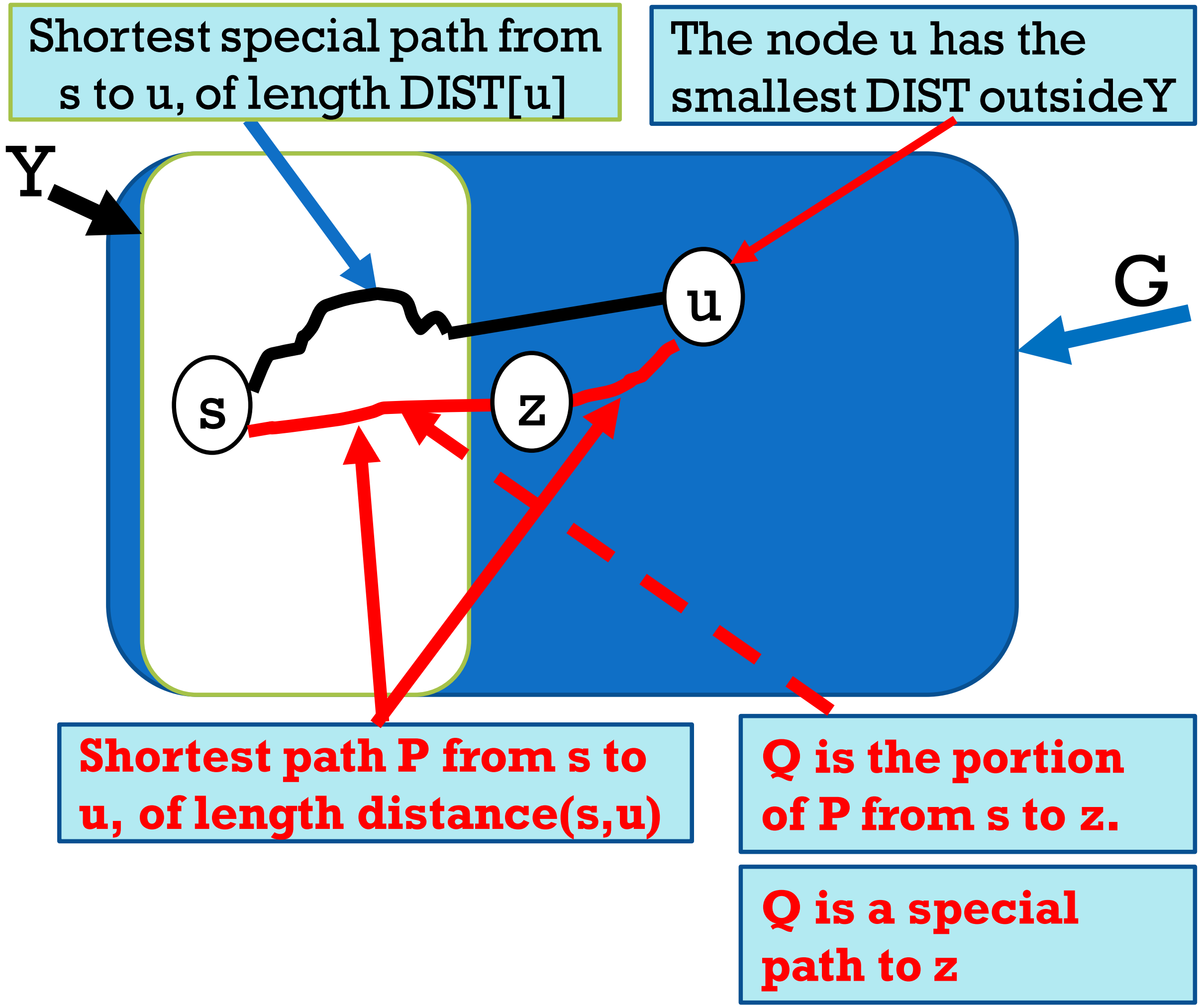
Proof:

- The proof is by induction on the number k of elements in Y.
- Basis: k=1. That is, Y has only node s. Well, DIST[s]=W[s,s]=0; also, distance(s,s)=0. Thus, DIST[s]=distance(s,s).
- Induction step:
  - Assume the theorem holds for every node v that had entered Y before u. That is, assume that for every node v that entered into Y before u, v satisfies DIST[v] = distance(s,v).
  - Prove that the theorem holds for u (which is selected by the algorithm to be the next node to enter Y). That is, prove that DIST[u] = distance(s,u). We do so by contradiction.

OPTIMALITY OF THE GREEDY SSSP (2/3)

Proof (continued): **DIST[u] = distance(s,u) ??**

- Assume DIST[u] ≠ distance(s,u). That is, distance(s,u) < DIST[u]
- This means the shortest path from s to u (call that path P) is not a special path.
- This implies that at some point, P exits Y going through some node/nodes before reaching u.
- Let z be 1st such node, Q the portion of P from s to z.
- Q is a special path to z => $\text{DIST}[z] \leq \text{length}(Q)$
- $\text{DIST}[z] \leq \text{length}(Q) \leq \text{length}(P) = \text{distance}(s,u) < \text{DIST}[u]$
- $\therefore\ \text{DIST}[z] < \text{DIST}[u]$
  - contradicting the choice of u as having the smallest DIST outside Y.



OPTIMALITY OF THE GREEDY SSSP (3/3)

Proof (continued):

- The contradiction means that the assumption that

$$\text{DIST}[u] \neq \text{distance}(s,u)$$

  must be false

- Hence, DIST[u] = distance(s,u). Q.E.D.

LESSONS LEARNED SO FAR

- The same greedy policy on the same problem can be implemented in different ways
- Some implementations can be much faster (e.g., min-heap for greedy sorting)
- Pre-processing the input can be very helpful (e.g., sorting P/W)
- The greedy method does not always guarantee optimality
- To prove non-optimality, use counter-examples
- For the same problem, one can formulate different greedy policies, some non-optimal and some optimal
- Greedy selections may have to be discarded sometimes (like in MST)
- Sometime problems may have to be reformulated to make the greedy formulatable (as in SSSP)
- Proving optimality of greedy solutions can be quite involved, but not too hard

OTHER WELL-KNOWN GREEDY ALGORITHMS

- **The Huffman Coding** (for lossless compression): Optimal
- **Activity selection problem**: Optimal
- Many other scheduling problems, graph problems, etc., where the greedy solution may not be optimal but is often sub-optimal (i.e., better than random solution but is not necessarily optimal)
- In fact, when algorithms for finding an optimal solution are too slow, designers resort to (fast) greedy algorithms and are contented with the sub-optimal greedy solutions