Greedy Approach (GA)

Prof. Young Park

✓Greedy Approach

- An approach/paradigm to designing algorithms!

Greedy Approach

- A **greedy approach** arrives at a solution
  - By making a sequence of choices, each of which simply looks best (locally optimal) at the moment.
  - **Without regard to the choices it has made before or to the choices that it might make in the future.**
  - The choices are irrevocable!
    - ✓**Once a choice is made, it cannot be reconsidered!**

Each Iteration in The Greedy Approach

- A **selection** procedure, chooses the next item to add to the set according to a greedy criterion.
- A **feasibility** check, determines if the new set is feasible by checking whether it is possible to complete this set in such a way as to give a solution to the instance.
- A **solution** check, determines whether the new set constitutes a solution to the instance.

Greedy Approach vs DP

- Like **DP**, the **Greedy Approach** is used to solve **optimization problems**!
  1. **The principle of optimality apply.**
  2. **If locally optimal is globally optimal!**

- The **Greedy Approach** *often* leads to simpler & more efficient solutions compared with **DP**.

Greedy Approach vs DP

- A greedy algorithm **does not always guarantee** an optimal solution.
- Why not?
  - **Locally optimal, but not always globally optimal!**
- **SO, a formal proof is needed** to show that a particular greedy algorithm approach always produces an optimal solution for a problem or not.

Greedy Approach vs DP

- It is usually more difficult to determine whether a greedy approach-based algorithm always produces an optimal solution compared with **DP**.

✓ Greedy Approach-based Algorithms

Greedy Approach-based Algorithms

1. The Min-Coin Change problem via Greedy Approach
2. The MST Problem via Greedy Approach – Prim's algorithm & Kruskal's algorithm
3. The Single-Source Shortest-Paths Problem via Greedy Approach - Dijkstra's algorithm
4. The Scheduling Problem via Greedy Approach
5. The Fractional Knapsack Problem via Greedy Approach

1. The Min-Coin Change problem

- The Min-Coin Change problem
  - **To minimize** the total number of coins returned as change by a sales clerk to a customer.
  - What is the minimum number of coins to make the change?
    - Coins={c_1, c_2, …, c_n}
    - Change
- Idea?

The Min-Coin Change problem via Greedy Approach

- A Greedy Approach:
  - Choose the **largest** remaining coin to minimize!

Example: The Min-Coin Change problem via Greedy Approach

Example:

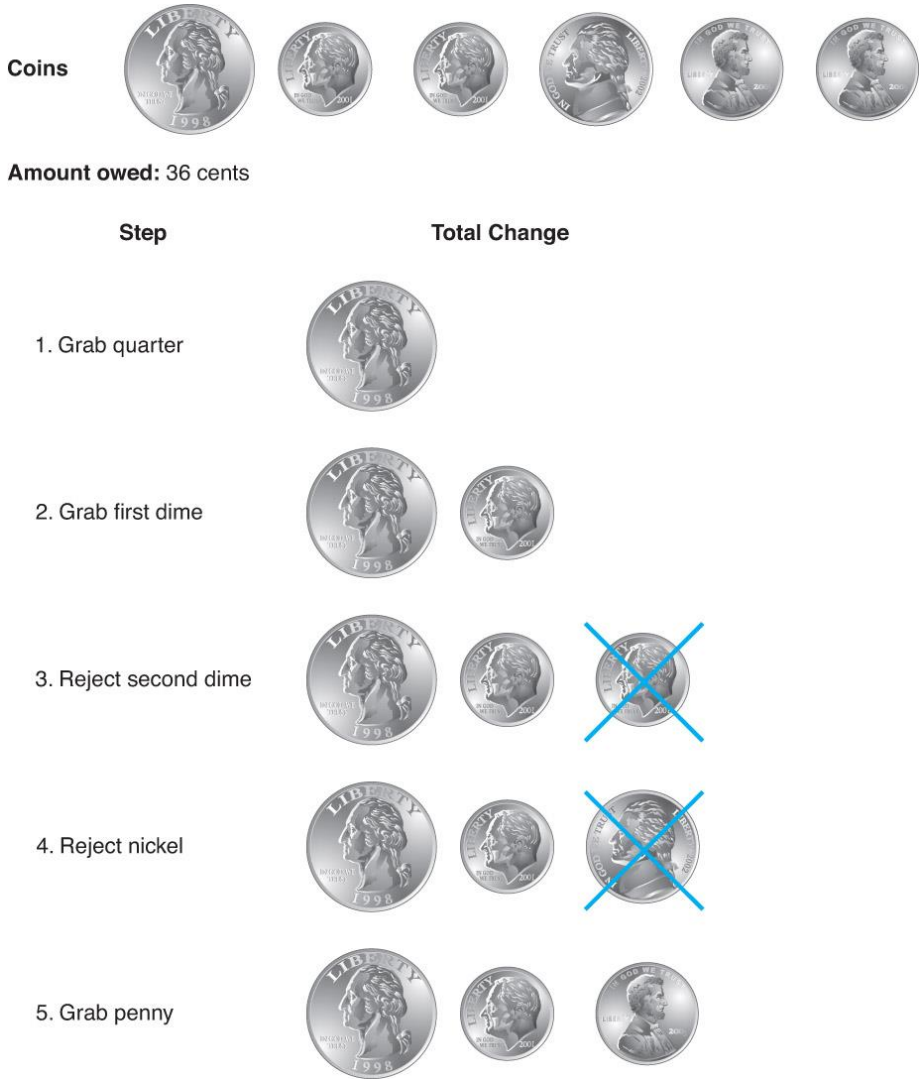


Figure 4.1

The Min-Coin Change problem via Greedy Approach

- This Greedy Approach is optimal!
- O(n log n)

The Min-Coin Change problem via Greedy Approach

- How about **a different version** of the Min-Coin Change problem?
- If there is a **12-cent coin**?
- The greedy approach - Choose the **largest** remaining coin?

Example: The Min-Coin Change problem via Greedy Approach

Example:

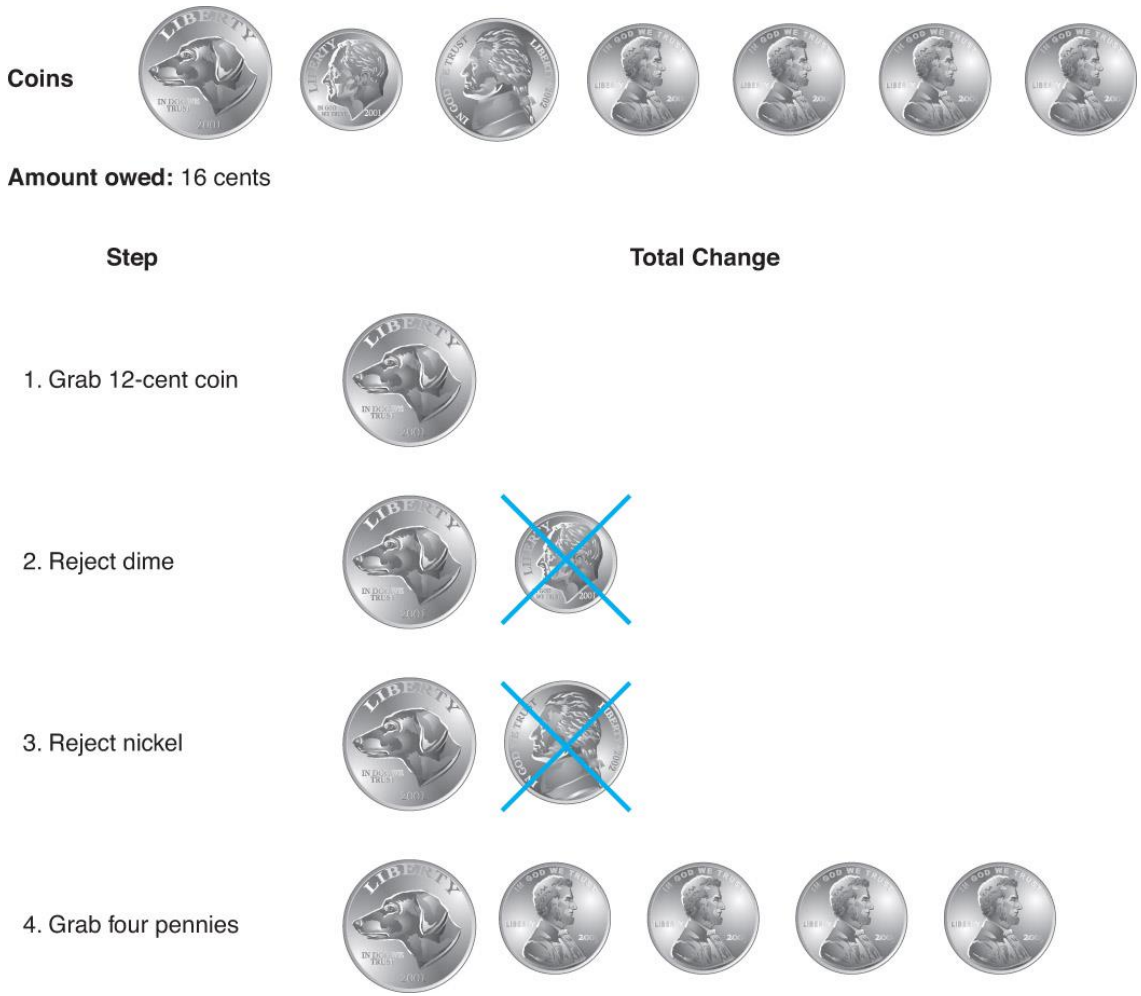


Figure 4.2

The Min-Coin Change problem via Greedy Approach

- This case, the Greedy Approach is not optimal!

- **Greedy Approach does not work this version of the min-coin change problem!**

The Min-Coin Change problem via Greedy Approach

- **Then what?**
  - For general case of fining minimum number of coins?
  - **Use DP!**
  - **The Min-Coin Change problem via DP**

▶QUIZ?

- The **Min-Coin Change** problem? Greedy Approach? DP?
  - Greedy Approach? Yes and No! Depending on coins.
  - DP? Yes!

2. The Minimum-Cost Spanning Tree (MST) Problem

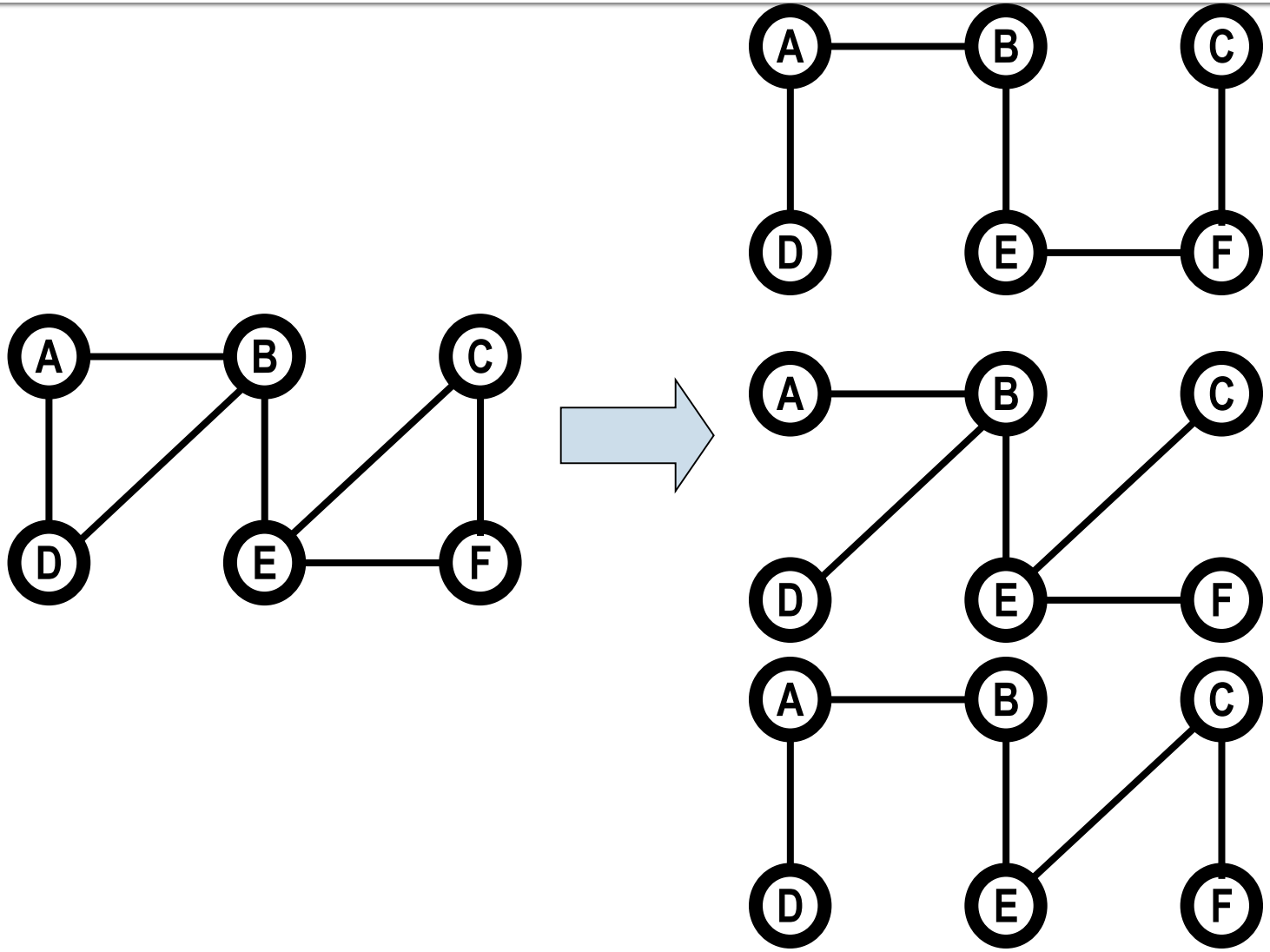
Spanning Tree

- Let G = (V, E) be a **connected, undirected** graph.
- A subgraph G′ = (V′, E′) of a connected, undirected graph G = (V, E) where $V' \subseteq V$ and $E' \subseteq E$ is a spanning tree if G′ has the following properties:
  - V′ = V
  - G′ is connected.
  - G′ is acyclic.

- Observation: **A spanning tree has |V|-1 edges!**

- Every acyclic undirected graph can be viewed as a general, unordered tree!

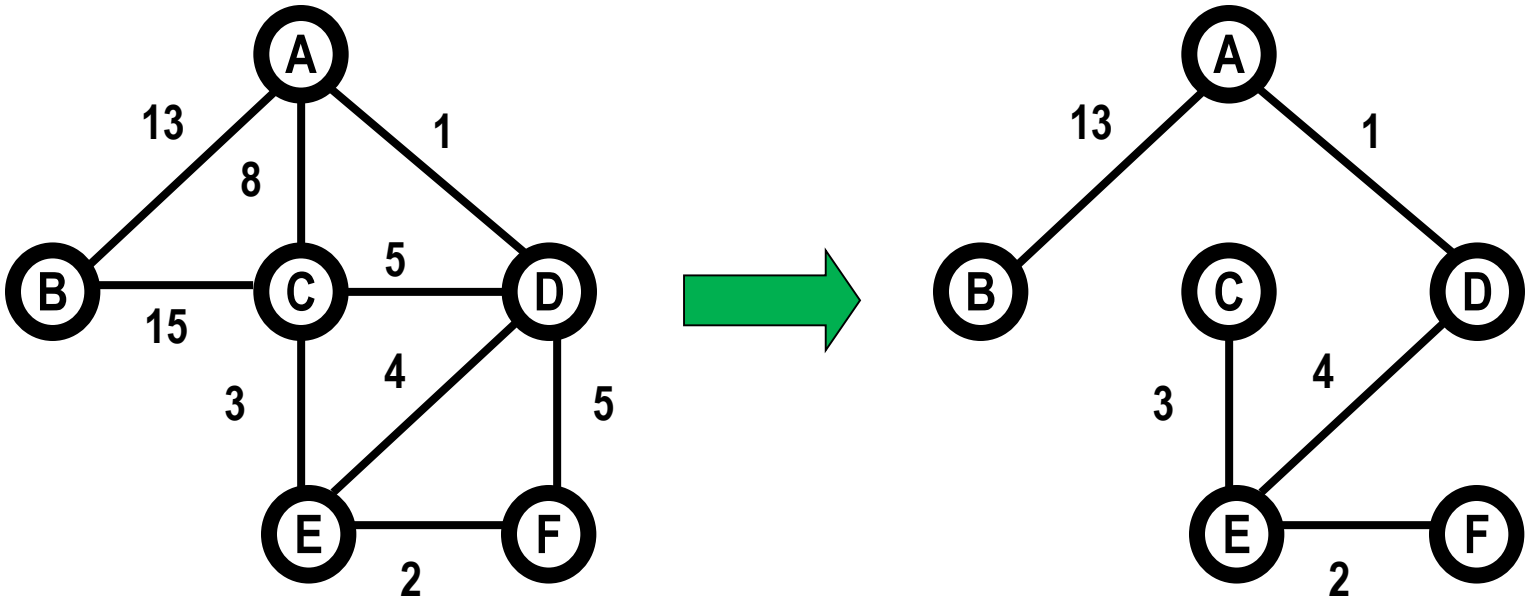
Example: Spanning Tree



Minimum-cost Spanning Tree

- A **minimum-cost** spanning tree of an edge-weighted undirected graph G = (V, E) is
  - A spanning tree of the graph that has the least total cost.
- In general, it is possible for a graph to have several different minimum-cost spanning trees!

Example: Minimum-cost Spanning Tree



The Minimum-Cost Spanning Tree (MST) Problem

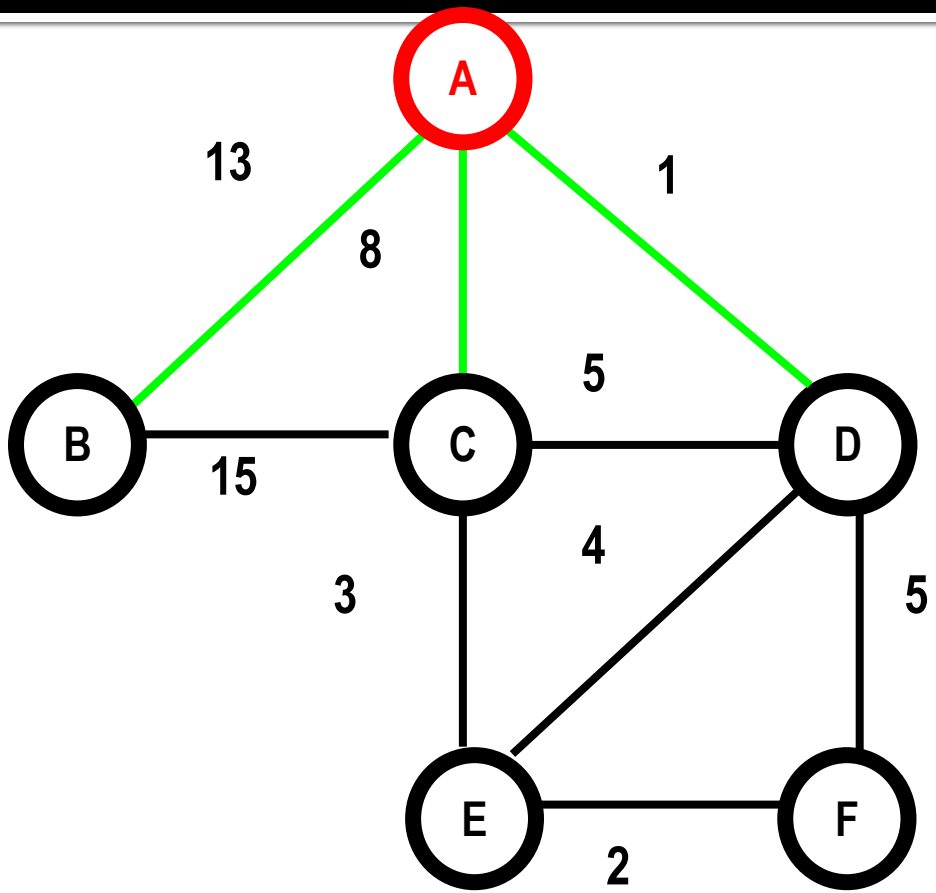
- The Problem:
  - Given an edge-weighted, connected, **undirected** graph G = (V, E), find a minimum-cost spanning tree of G = (V, E).
- This kind of problem has numerous applications.
- Idea?

The MST Problem via Greedy Approach

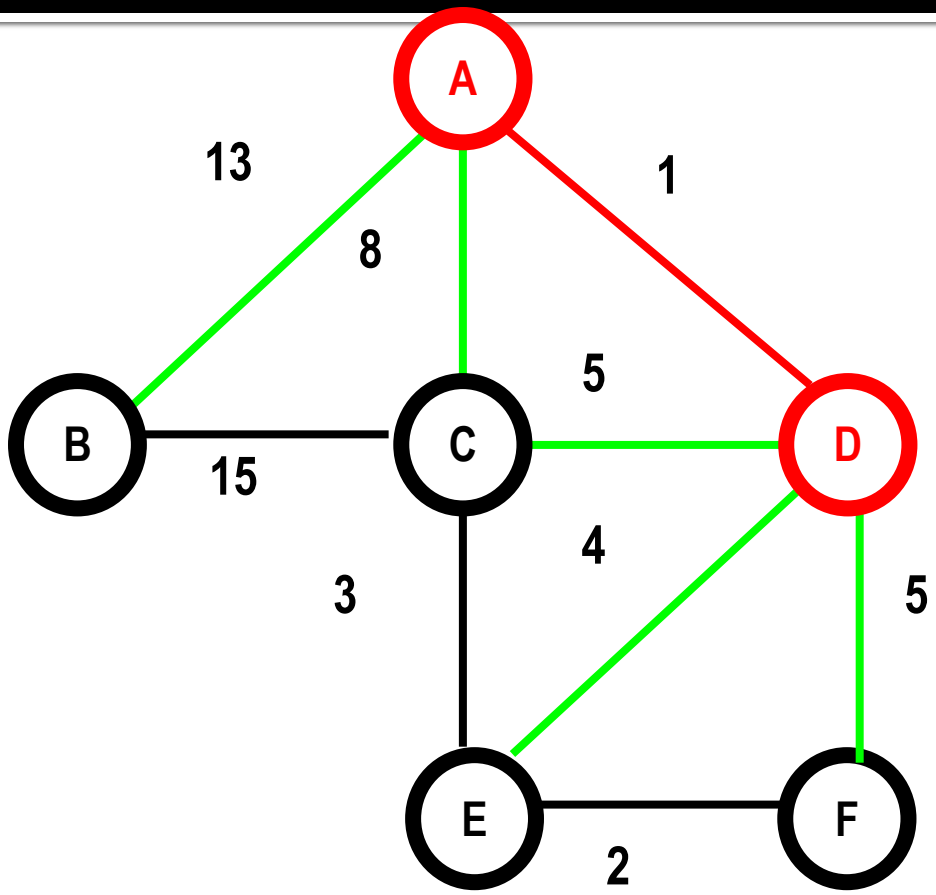
- Two Greedy Algorithms:
  - **Prim's algorithm**
  - **Kruskal's algorithm**

  - Each uses a different locally optimal property.
  - Must prove each algorithm.

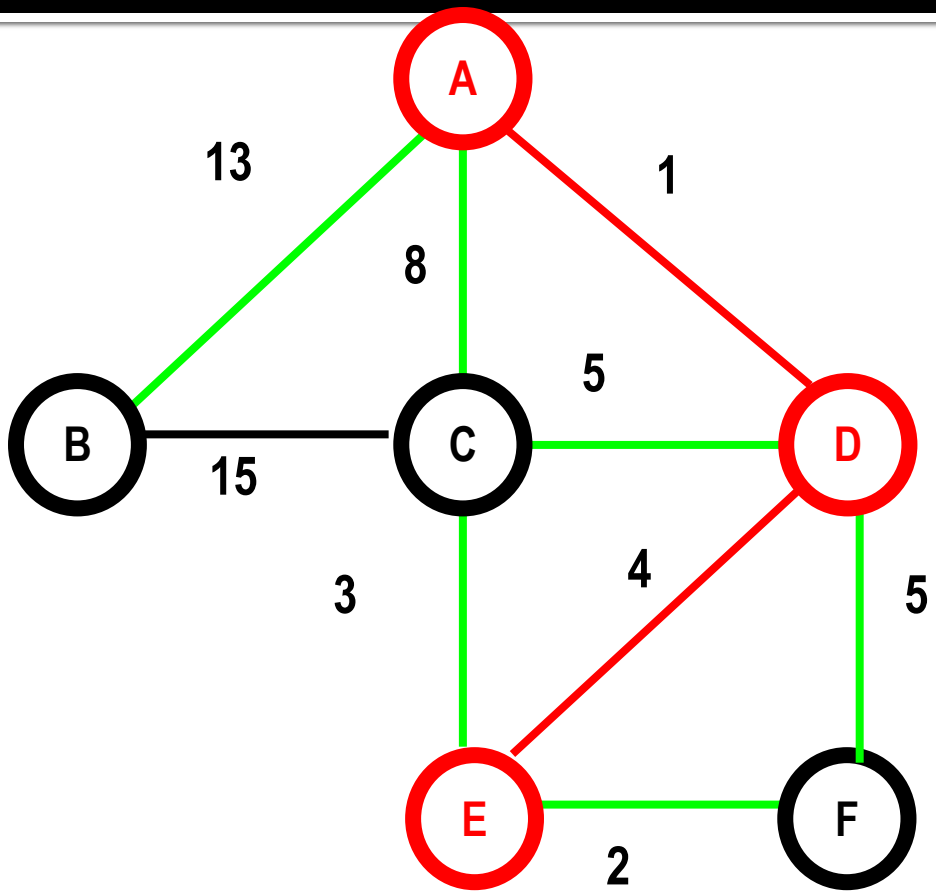
Approach 1: MST



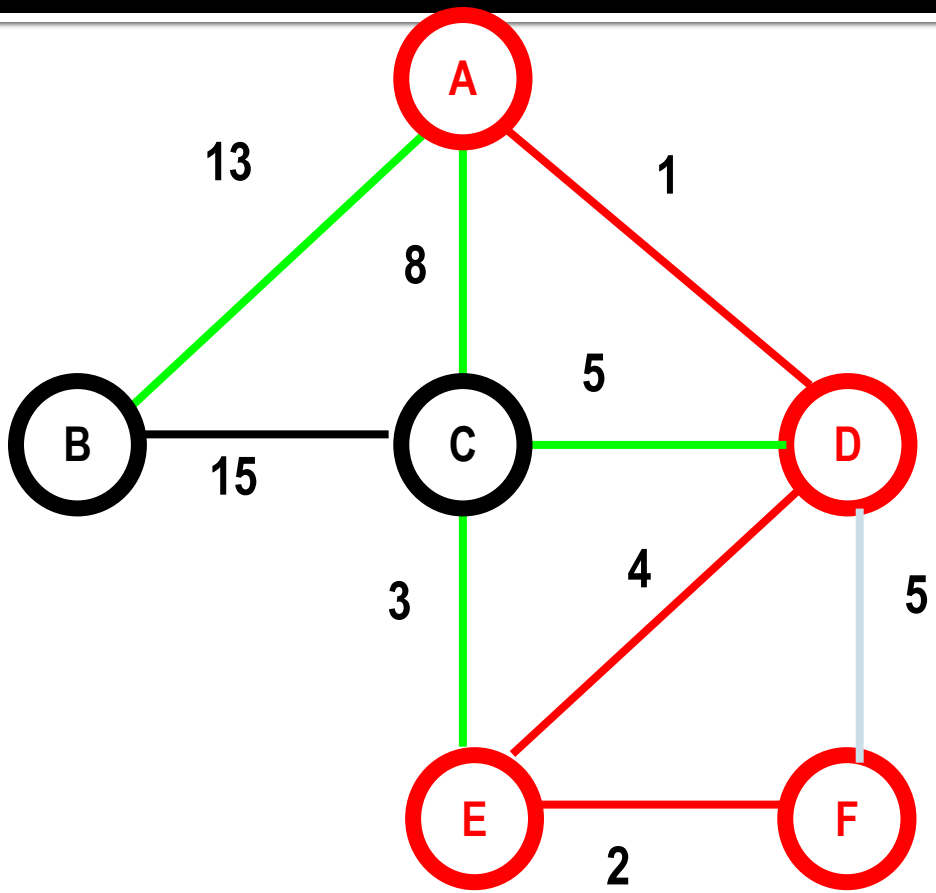
Approach 1: MST



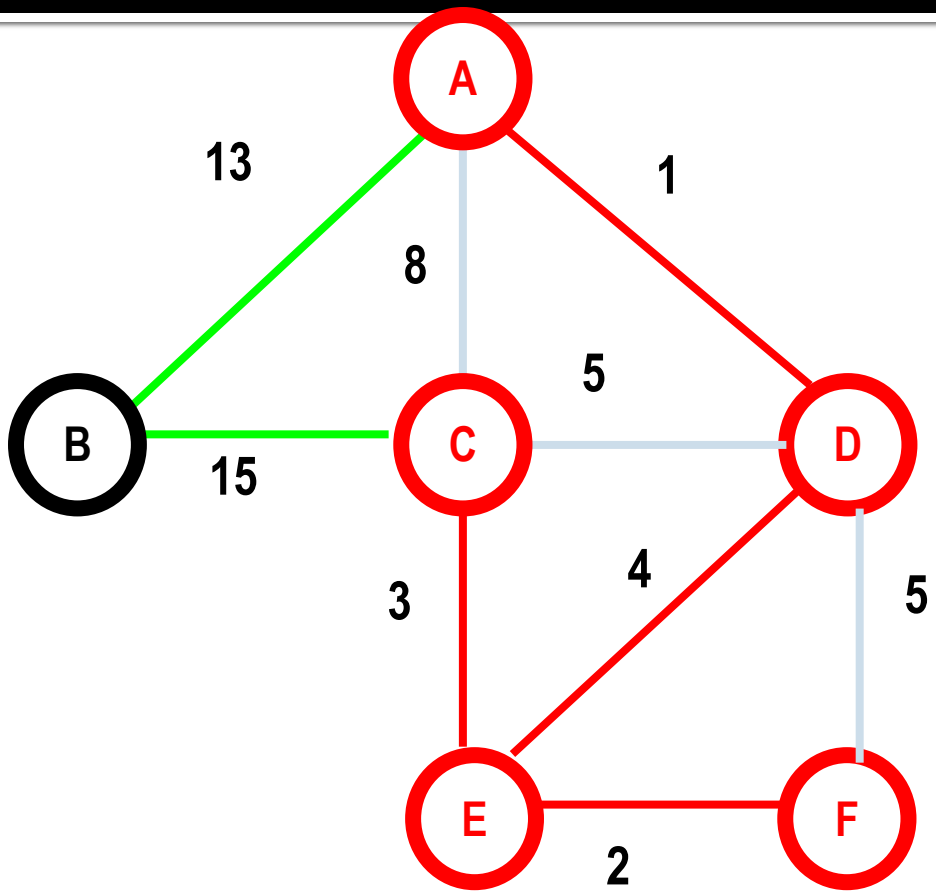
Approach 1: MST



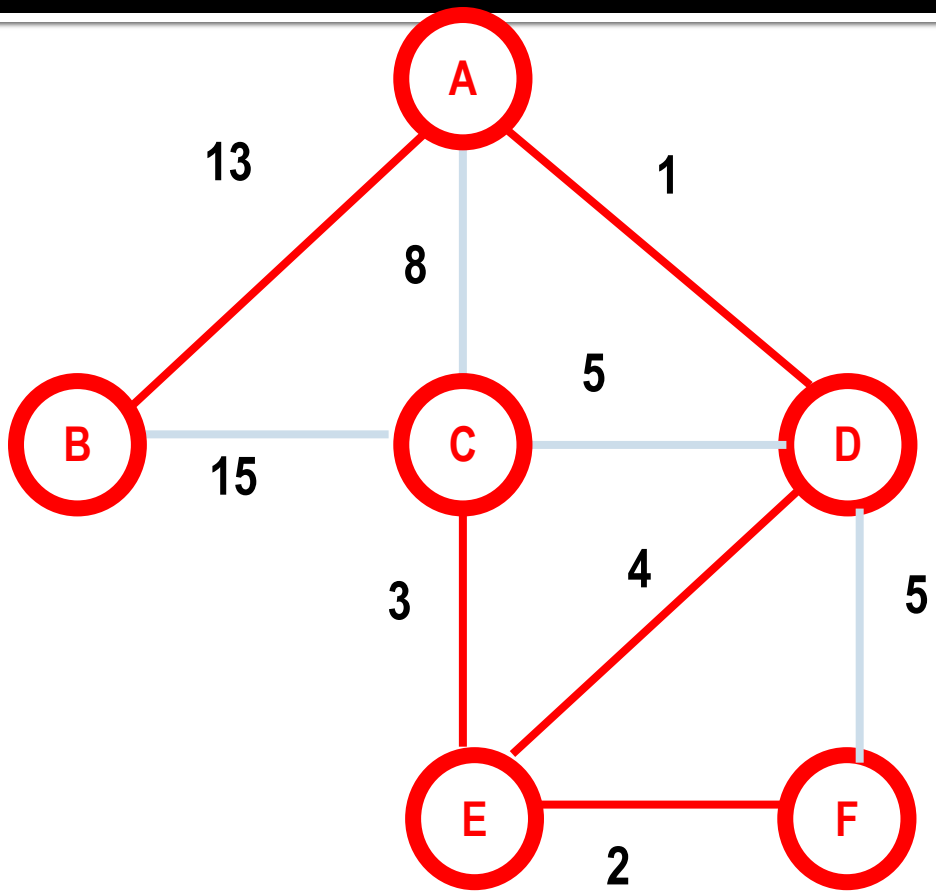
Approach 1: MST



Approach 1: MST



Approach 1: MST



MST(Minimum Spanning Tree)

The MST Problem via Greedy Approach

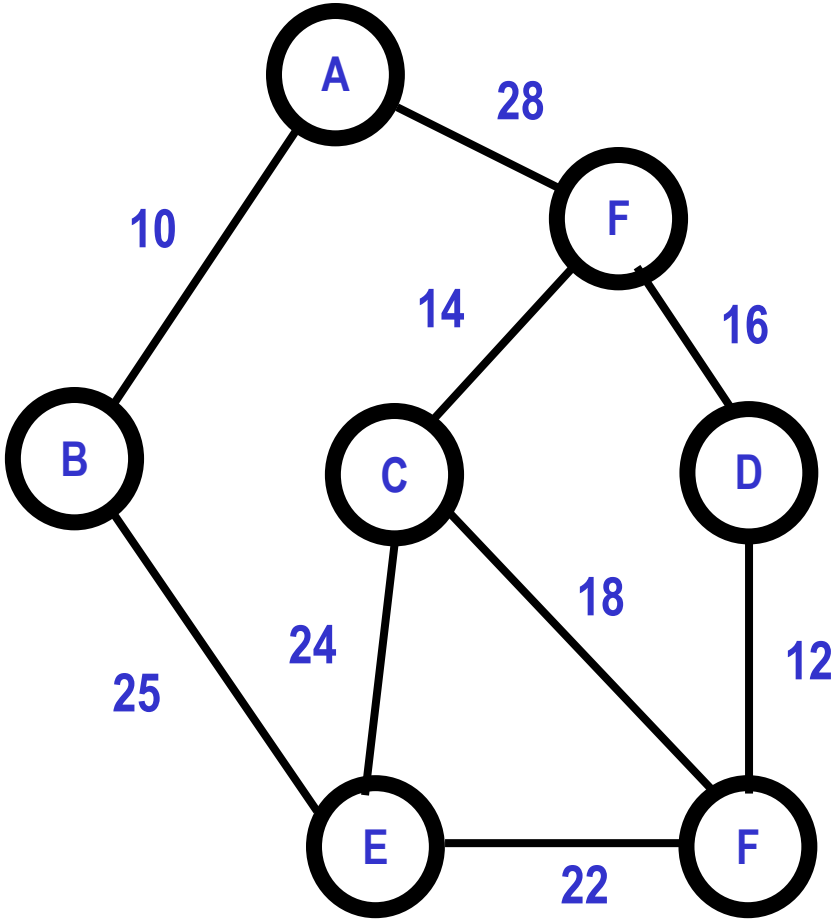
- **Prim's algorithm**
  - A **node-based** greedy algorithm
  - Builds MST by greedily adding nodes

Prim's MST Algorithm via Greedy

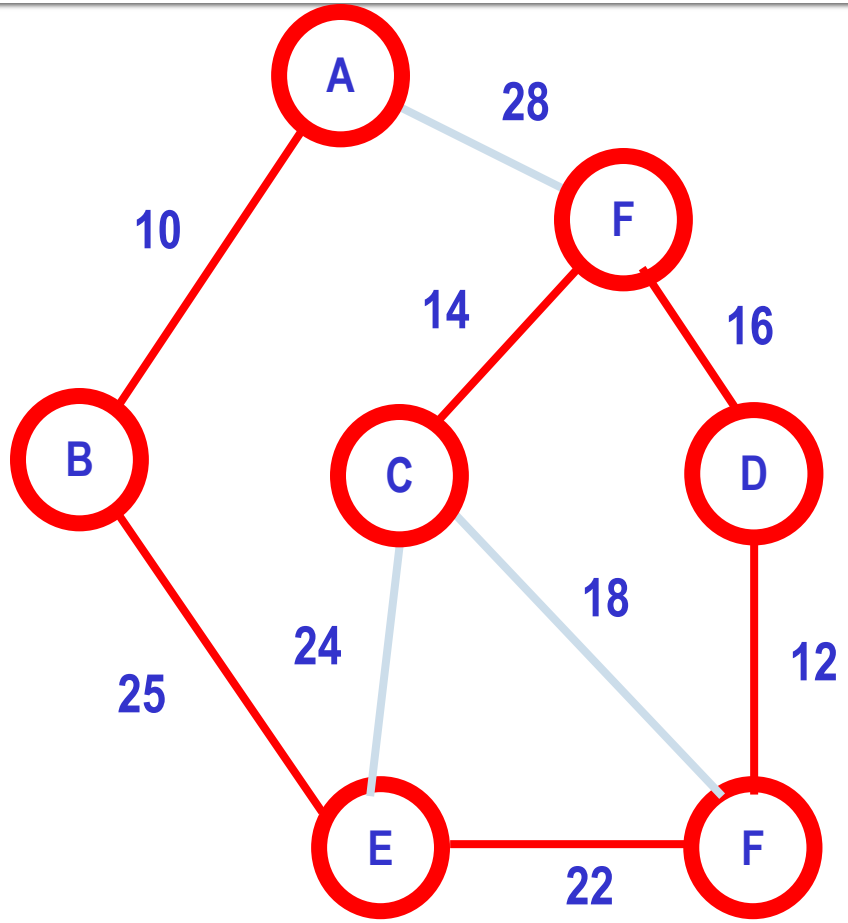
MST_E = ∅
MST_V = {v_1}
while (instance not solved)
{
    //selection procedure and feasibility check
    select a vertex in **V – MST_V nearest** to **MST_V**;
    add the vertex to **MST_V**;
    add the edge (called cut) to **MST_E**;

    //solution check
    if (**MST_V** == V) the instance is solved;
}

▶QUIZ? Prim's MST Algorithm via Greedy



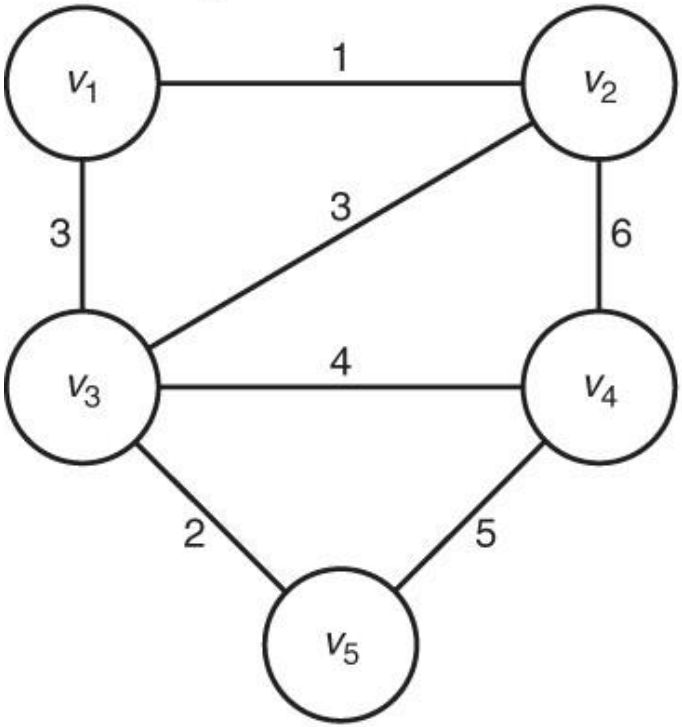
▶QUIZ: Prim's MST Algorithm via Greedy



MST(Minimum Spanning Tree)

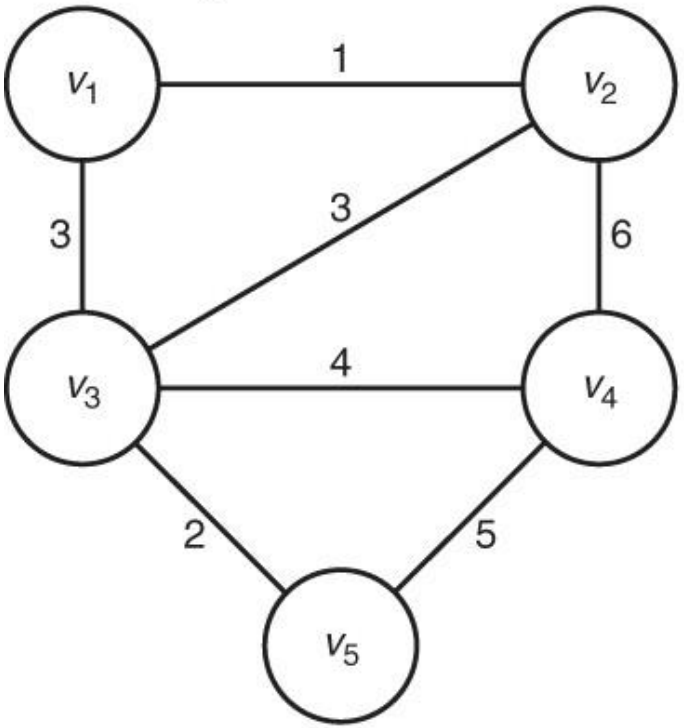
►QUIZ?

Determine a minimum spanning tree.



▶QUIZ: MST by Prim's Algorithm

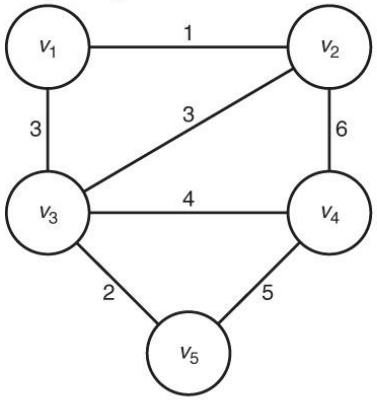
Determine a minimum spanning tree.



MST(Minimum Spanning Tree)

▶QUIZ: MST by Prim's Algorithm

Determine a minimum spanning tree.



1. Vertex v_1 is selected first.

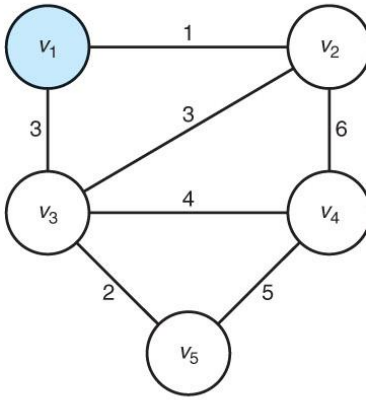
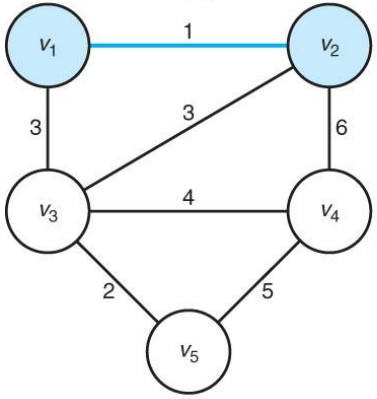
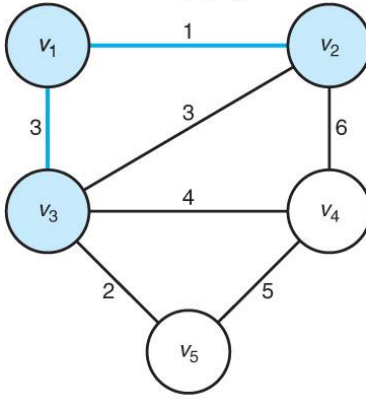


Figure 4.4

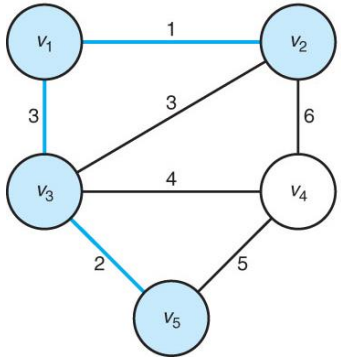
2. Vertex v_2 is selected because it is nearest to $\{v_1\}$.



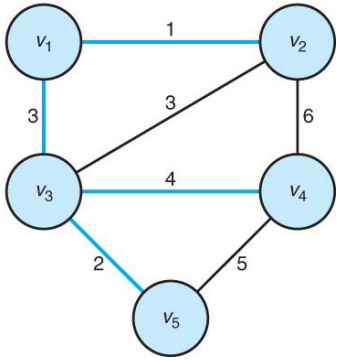
3. Vertex v_3 is selected because it is nearest to $\{v_1, v_2\}$.



4. Vertex v_2 is selected because it is nearest to $\{v_1, v_2, v_3\}$.



5. Vertex v_4 is selected.



Prim's MST Algorithm via Greedy

- **Algorithm 4.1 Prim's Algorithm**
  - Input: W (adjacency matrix)
  - Output: MST_E

Prim's MST Algorithm via Greedy

- Analysis of Algorithm 4.1 Prim's Algorithm
  - $O(|V|^2)$ for adjacency matrix representation

Prim's MST Algorithm via Greedy

- Prim's algorithm produces a spanning tree. But is it minimal?
  - Although greedy algorithms are easier to develop than DP algorithms,
  - It is usually more difficult to determine whether or not the greedy algorithm produces an optimal solution.
  - We need a formal proof for a greedy algorithm.

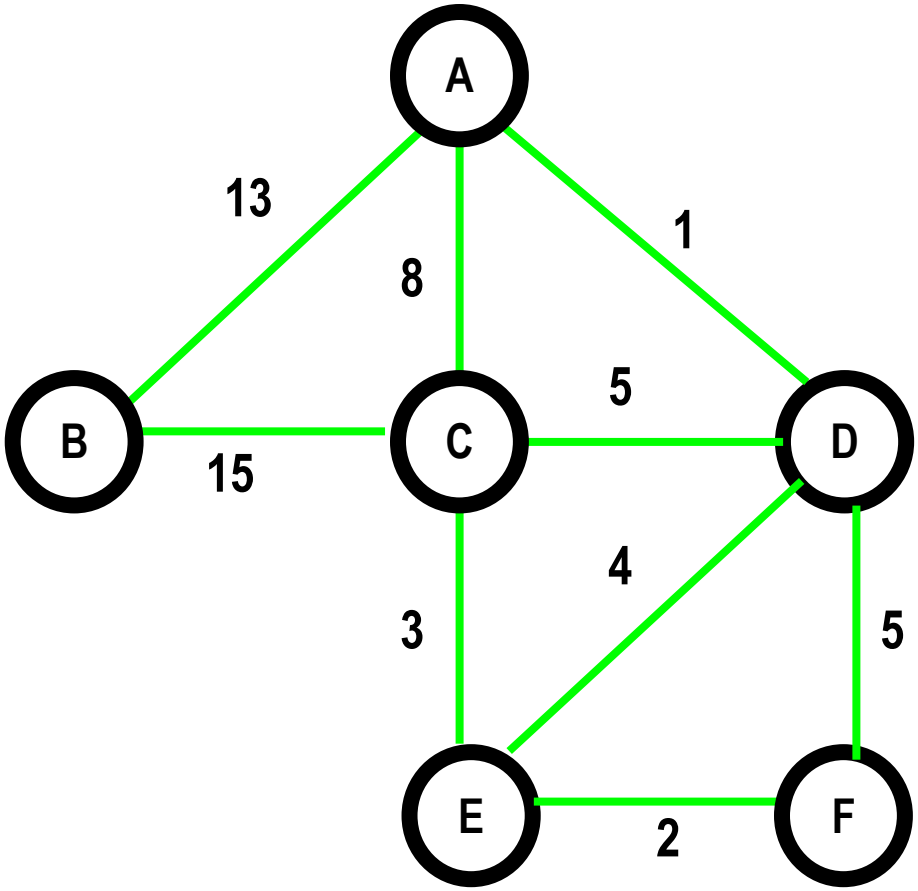
Prim's MST Algorithm via Greedy

- Prim's algorithm always produces a minimum spanning tree?
  - Theorem 4.1

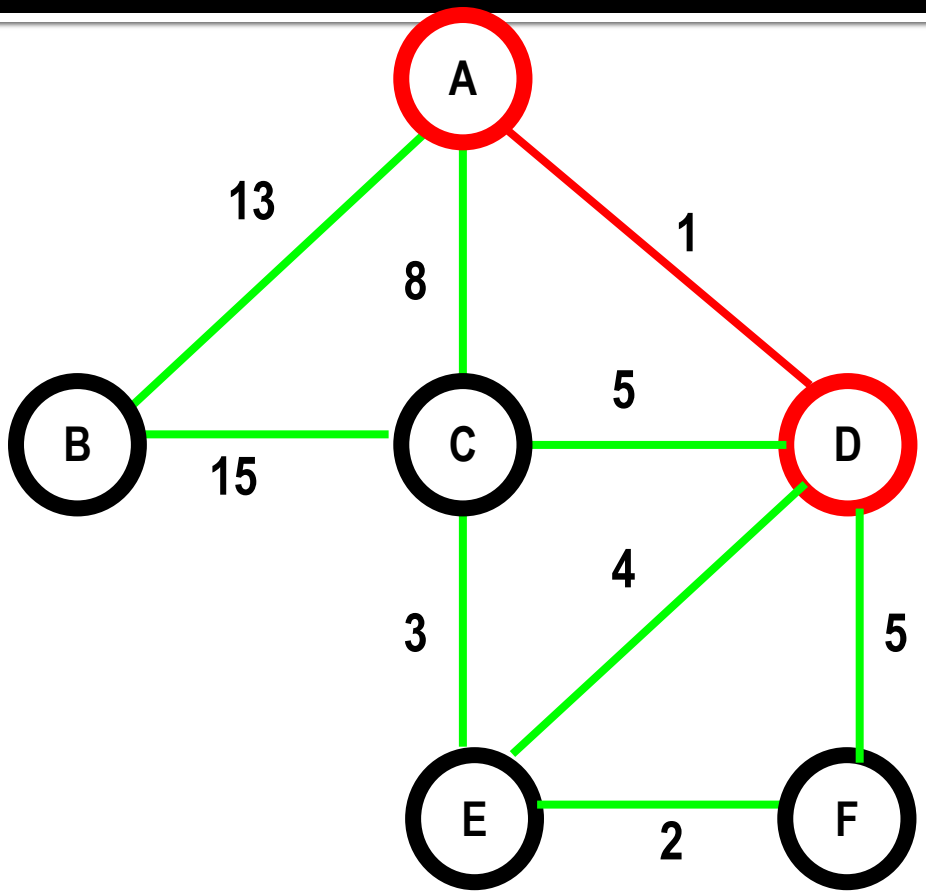
Prim's MST Algorithm via Greedy Visualization

- ***Prim's MST Algorithm via Greedy Visualization***

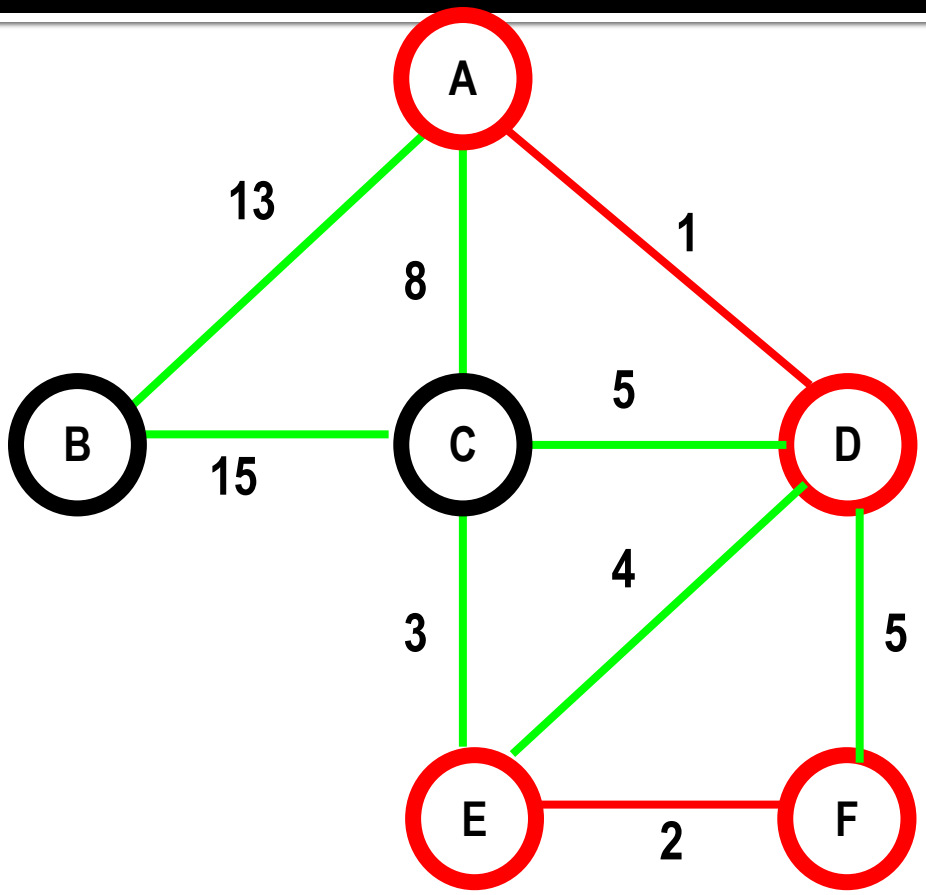
Approach 2: MST



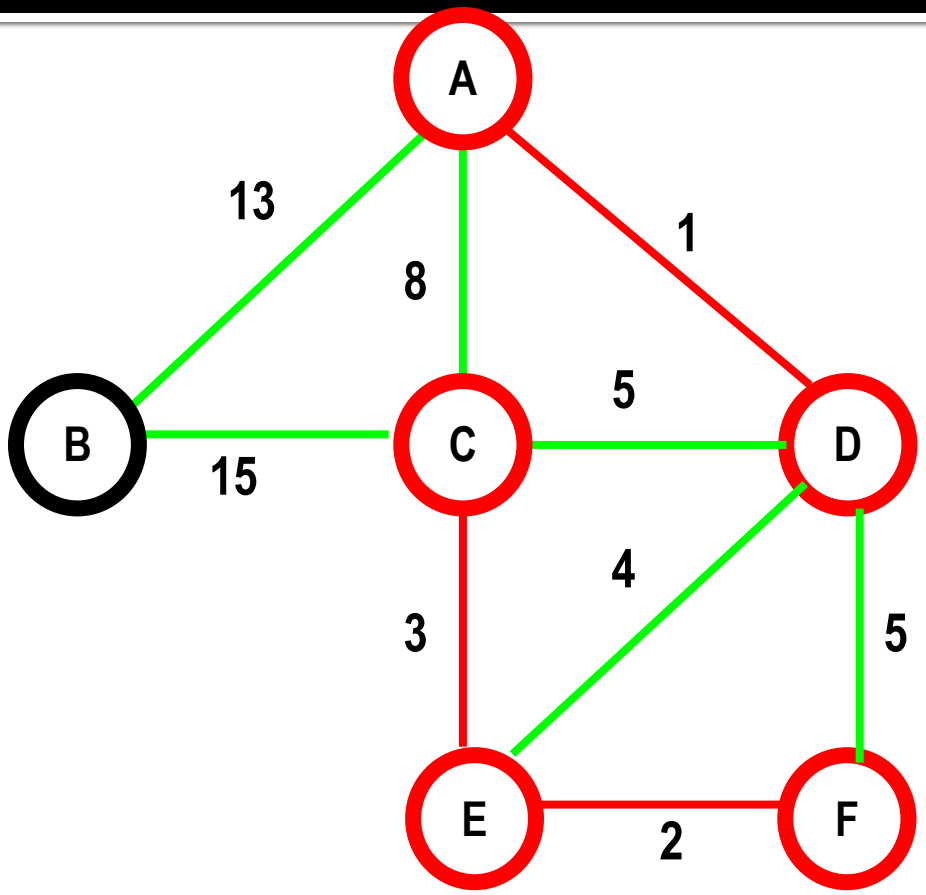
Approach 2: MST



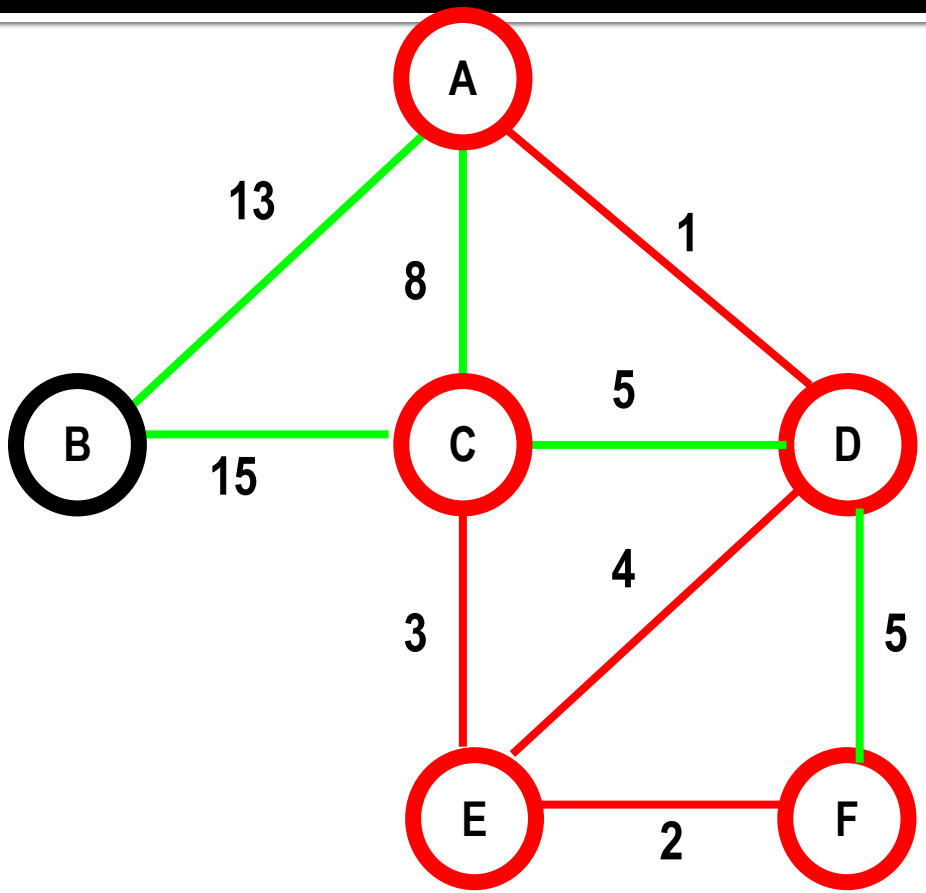
Approach 2: MST



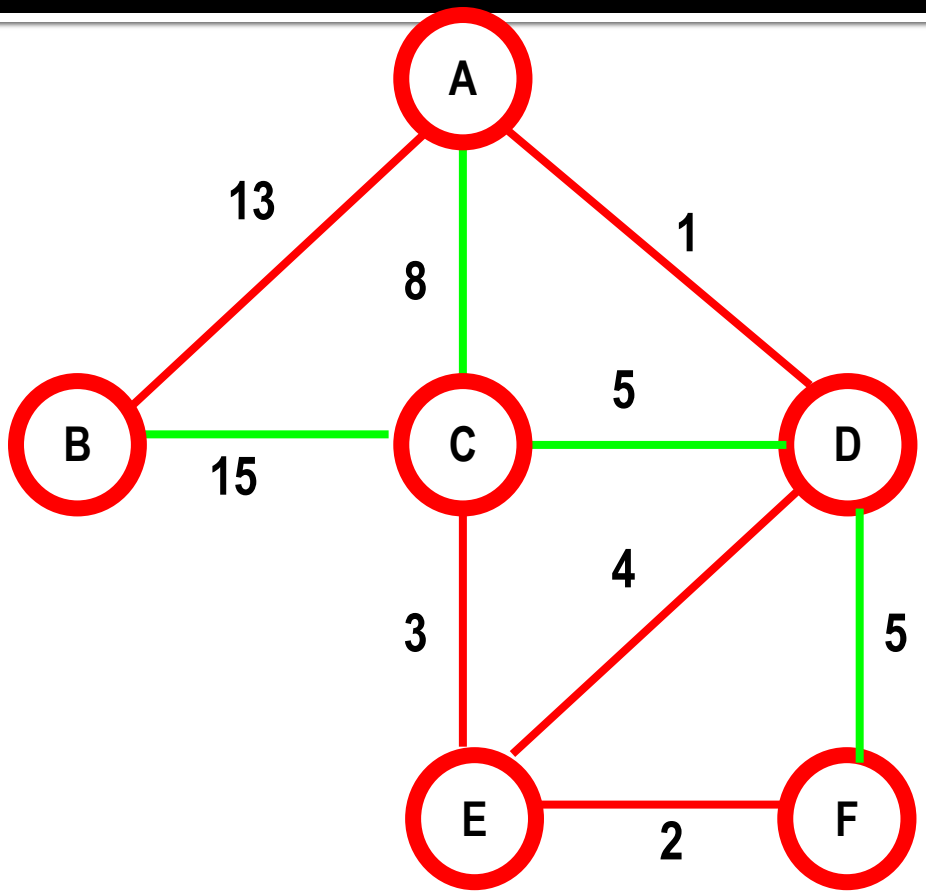
Approach 2: MST



Approach 2: MST



Approach 2: MST



MST(Minimum Spanning Tree)

The MST Problem via Greedy Approach

- **Kruskal's algorithm**
  - An **edge-based** greedy algorithm
  - Builds MST by greedily adding edges

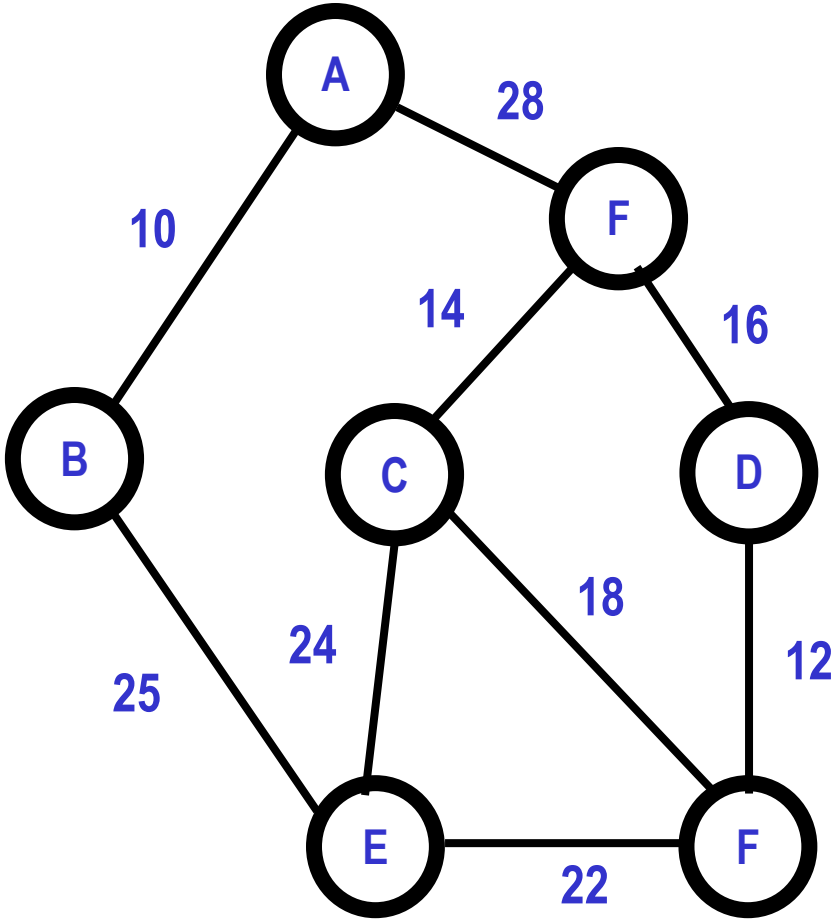
Kruskal's MST Algorithm via Greedy

```
MST_E = ∅;
Create disjoint subsets of V;
Sort the edges in E in non-decreasing order;
while (the instance is not solved)
{
        select next shortest edge; //selection procedure
        // If adding that minimum-weight edge to MST_E does not create a cycle (No cycle!)
        // then add that edge to MST_E;
        if (edge connects 2 vertices in disjoint subsets) //feasibility check
        {
                Merge the subsets;
                Add the edge to MST_E;
        }
        if (all the subsets are merged) //solution check
                instance is solved;
}
```

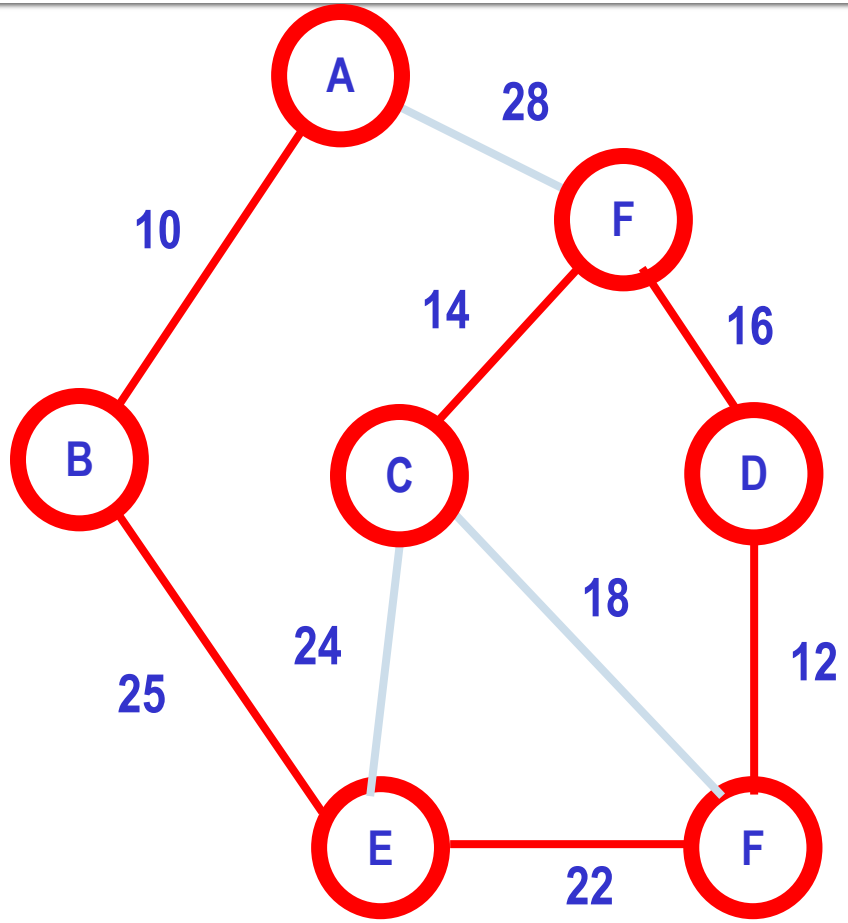
Kruskal's MST Algorithm via Greedy

- Use a **Disjoint-Set ADT to test for cycles.**
  - Union(a,b) unions the disjoint sets named by a and b into a single subset.
  - Find(a) returns the name of the subset containing a.
  - **O(log n)** worst-case time
  - **Appendix C**

▶QUIZ? Kruskal's MST Algorithm via Greedy



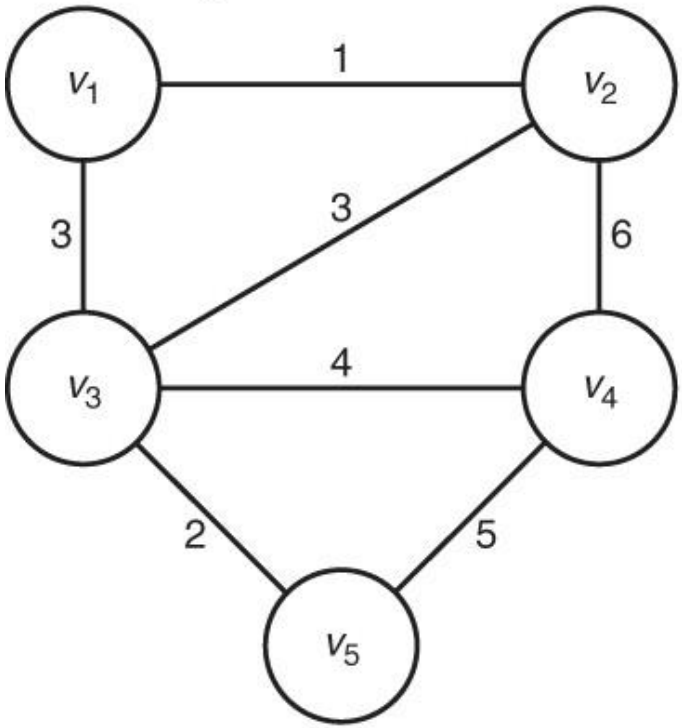
▶QUIZ: Kruskal's MST Algorithm via Greedy



MST(Minimum Spanning Tree)

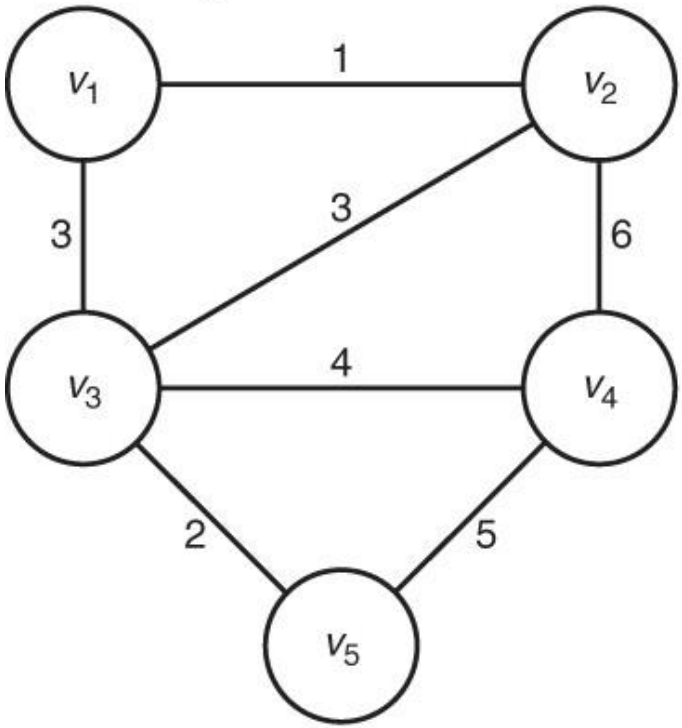
►QUIZ?

Determine a minimum spanning tree.



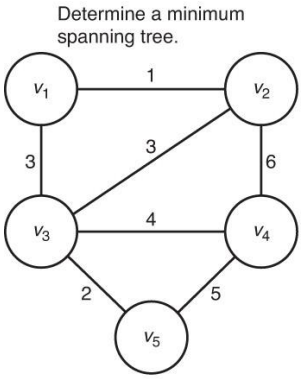
▶QUIZ: MST by Kruskal's Algorithm

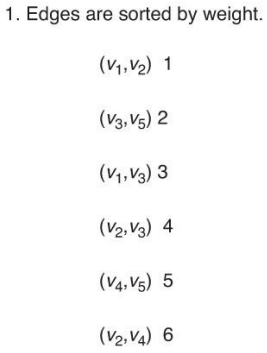
Determine a minimum spanning tree.



MST(Minimum Spanning Tree)

▶QUIZ: MST by Kruskal's Algorithm





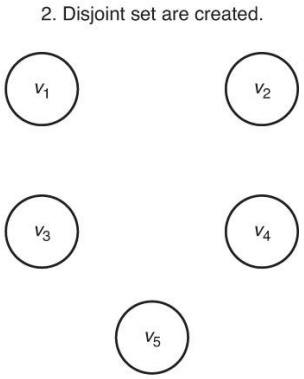
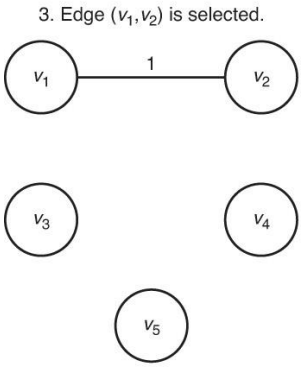
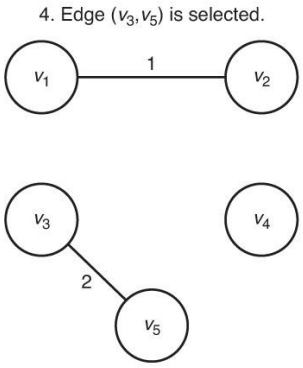
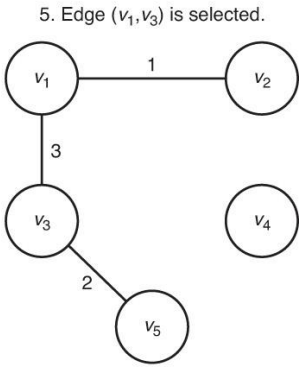
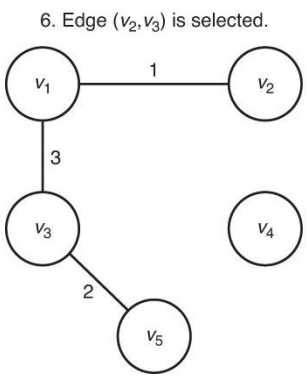


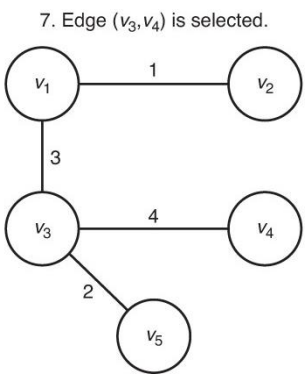
Figure 4.7











Kruskal's MST Algorithm via Greedy

- **Algorithm 4.2 Kruskal's Algorithm**
  - Input: W (adjacency matrix)
  - Output: MST_E

Kruskal's MST Algorithm via Greedy

- Analysis of Algorithm 4.2 Kruskal's Algorithm
  - **O(|E| log |E|)**
    - |E| = O(|V|) for sparse graphs
    - |E| = O($|V|^2$) for dense graphs
  - **O(|V| log |V|) ~ O($|V|^2$ log |V|)**

Kruskal's MST Algorithm via Greedy

- Kruskal's algorithm always produces a minimum spanning tree?
  - Theorem 4.2

Kruskal's MST Algorithm via Greedy Visualization

- ***Kruskal's MST Algorithm via Greedy Visualization***

Prim's MST Algorithm vs. Kruskal's MST Algorithm

- **Prim's vs Kruskal's?**
- For a sparse graph, **Kruskal's** algorithm O(|E| log |E|), i.e., O(|V| log |V|) is faster.
- For a dense graph, **Prim's** algorithm $O(|V|^2)$ is faster.

▶QUIZ?

- Prim's algorithm vs Kruskal's algorithm - Which one faster?
  - Sparse graphs – Kruskal's
  - Dense graphs – Prim's

▶QUIZ?

- Compare Prim's algorithm vs Kruskal's algorithm?

▶QUIZ?

- Prim's algorithm &
  Kruskal's algorithm with **Negative edge weights**?
  - No problem!

▶QUIZ?

- The **Maximum Spanning Tree** problem?
  - No problem!
  - Find **minimum** Spanning Trees with **negative** edge weights.

▶QUIZ?

- The **MST** problem? Greedy Approach? DP?
  - Greedy Approach? Yes!

3. The Single-Source Shortest-Paths (SSSP) Problem

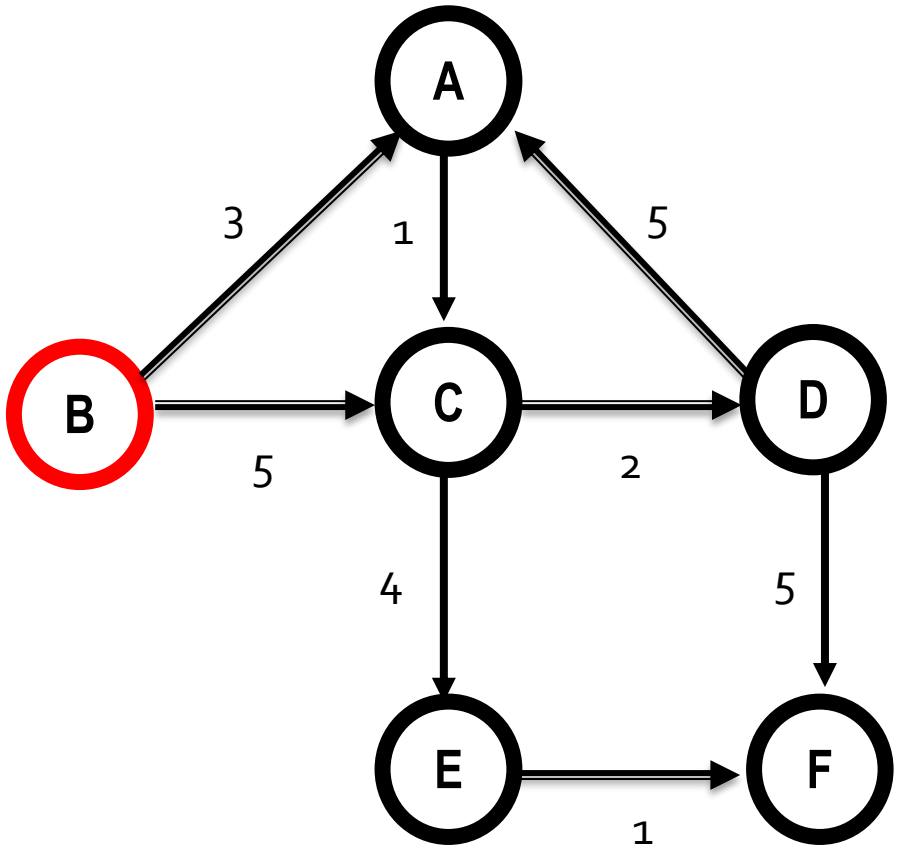
The Single-Source Shortest-Paths Problem

- The SSSP problem is easier than the APSP problem!
  - The SSSP problem solved by **Bellman & Ford's algorithm** via DP!
  - The APSP problem solved by **Floyd & Warshall's algorithm** via DP!
- **How about SSSP** with **no negative edge weights**?

The Single-Source Shortest-Paths Problem via Greedy Approach

- The Problem:
  - Given a **edge-weighted, directed** graph G=(V,E) with weight function W mapping edges to non-negative weights, find a shortest path from a given source vertex s to every vertex v in V.
- Idea?
  - A Greedy-based Algorithm
  - **Dijkstra's algorithm**
    - **Shortest Path Tree (SPT)**
    - Similar to Prim's algorithm!

Example: SSSP by Dijkstra's Algorithm



W

Distance?
Path?
Shortest paths?

Dijkstra's SSSP Algorithm via Greedy

SSSP_E = ∅
SSSP_V = {s} // source vertex
while (instance not solved)
{
    //selection procedure and feasibility check
    Select a vertex v in **V – SSSP_V** that has a **shortest** path from s **using only vertices** in **SSSP_V** **as intermediates**;
    Add the vertex v to **SSSP_V**;
    Add the edge on the shortest path that touches v to **SSSP_E**;

    //solution check
    if (**SSSP_V** == V) the instance is solved;
}

Dijkstra's SSSP Algorithm via Greedy

- For each vertex v, maintains three pieces of information:
  - **SPfound[v]** = The boolean-valued flag.
    - True = The shortest path from the source vertex s to v is known.
    - False = The shortest path from the source vertex s to v is unknown.
  - **Distance[v]** = The length of the shortest known path from the source vertex s to v.
  - **Path[v]** = The predecessor of the vertex v on the shortest path from the source vertex s to v.

Dijkstra's SSSP Algorithm via Greedy

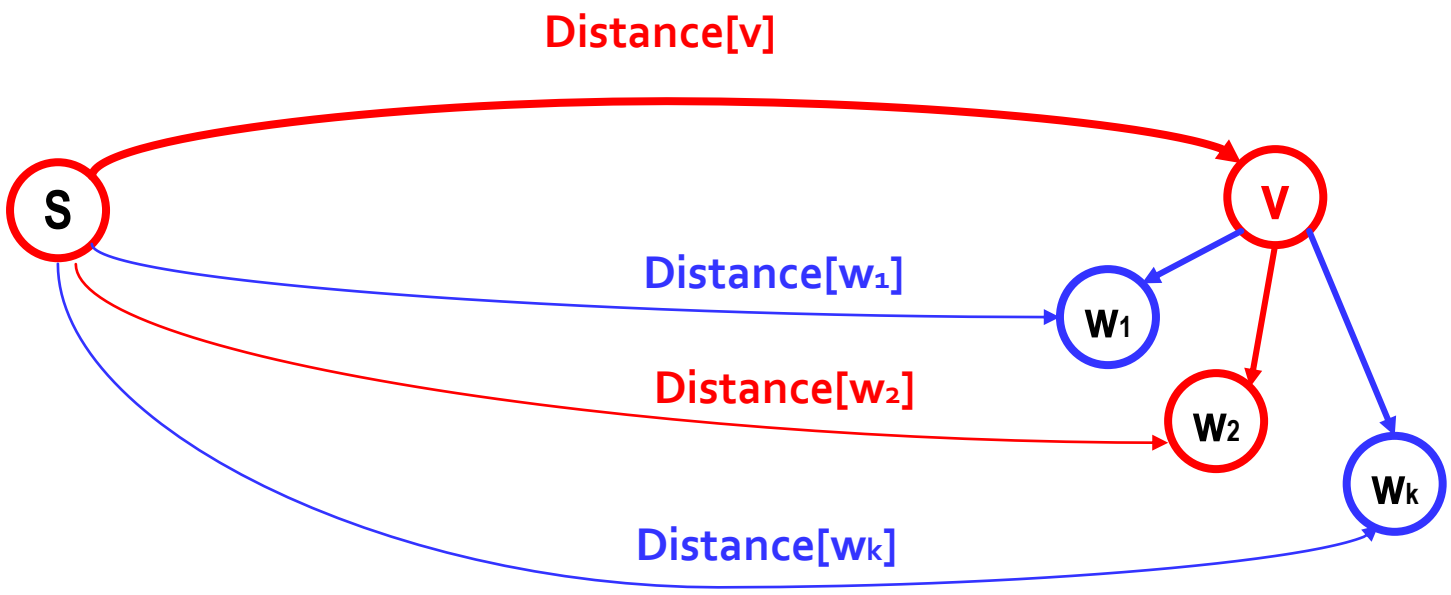
- Initially,
  - SPfound[v] = false for all v in V.
  - Distance[v] =
    - 0 for the source vertex s
    - ∞ for all other v in V.
  - Path[v] = unknown for all v in V.

Dijkstra's SSSP Algorithm via Greedy

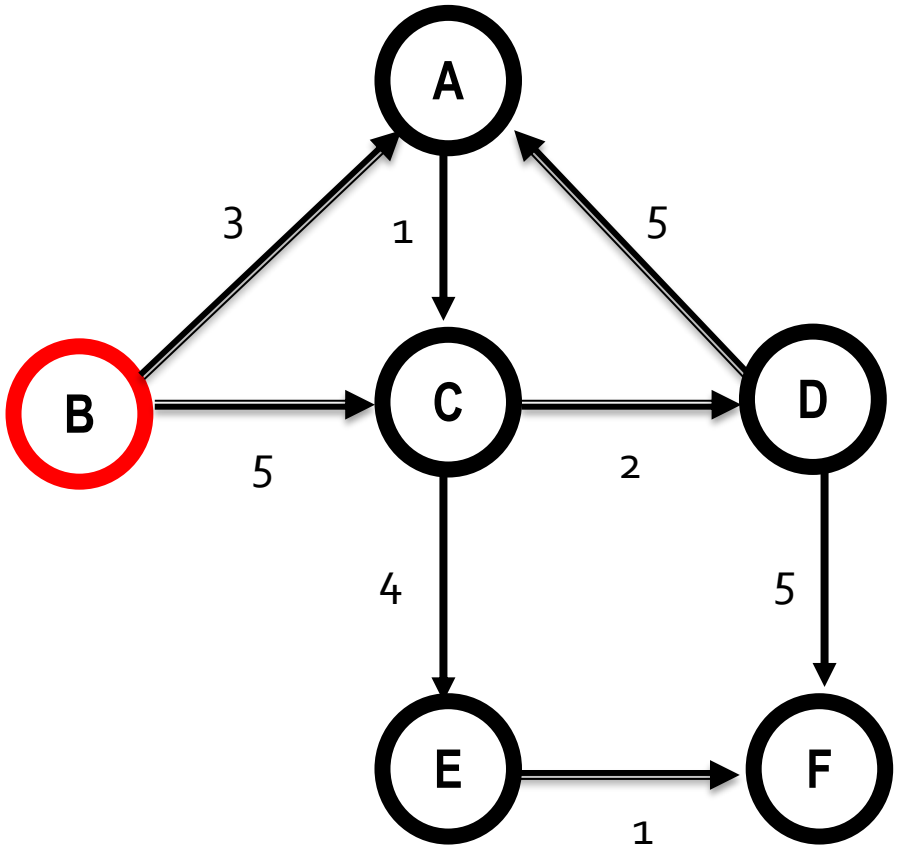
- From the set of vertices **whose SPfound[v] is false**,
  - Select the vertex **v** having the **smallest** distance Distance[v];
  - Set SPfound[**v**] = true;
- For **each vertex w that is adjacent to v (v,w)** & **whose SPfound[w] is false**,
  - Test whether the distance Distance[w] is greater than Distance[**v**] + weight (v,w).
  - If so, then
    - Set Distance[w] = Distance[**v**] + weight (v,w);
    - Set Path[w] = **v**;

Dijkstra's SSSP Algorithm via Greedy

Update **Distance?**

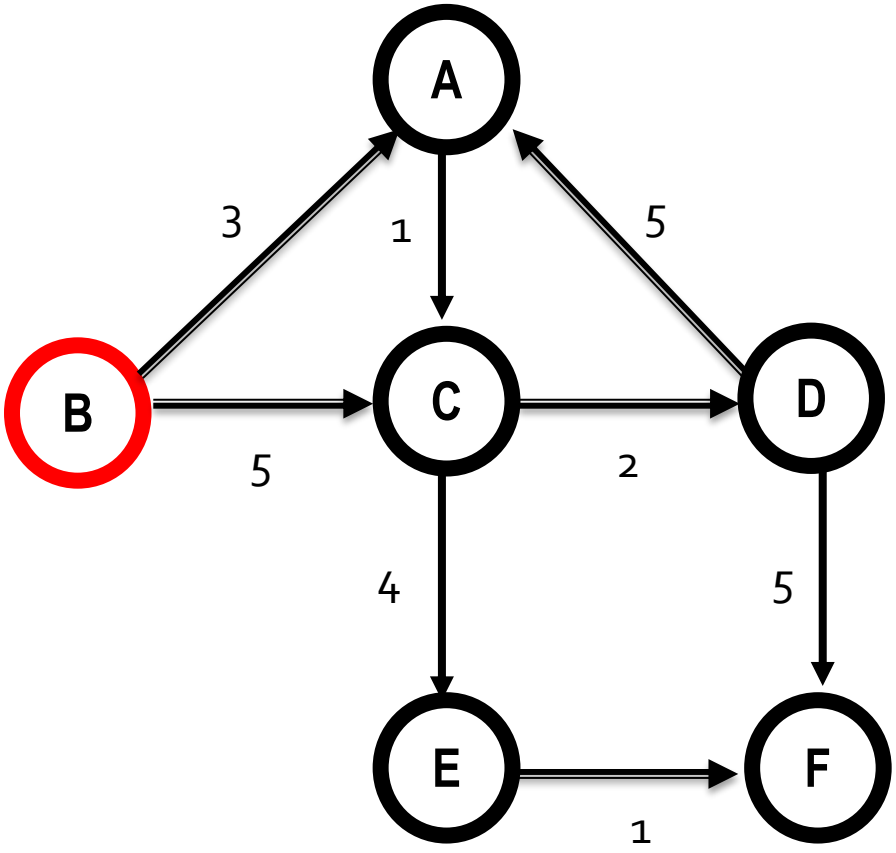


Example: SSSP by Dijkstra's Algorithm



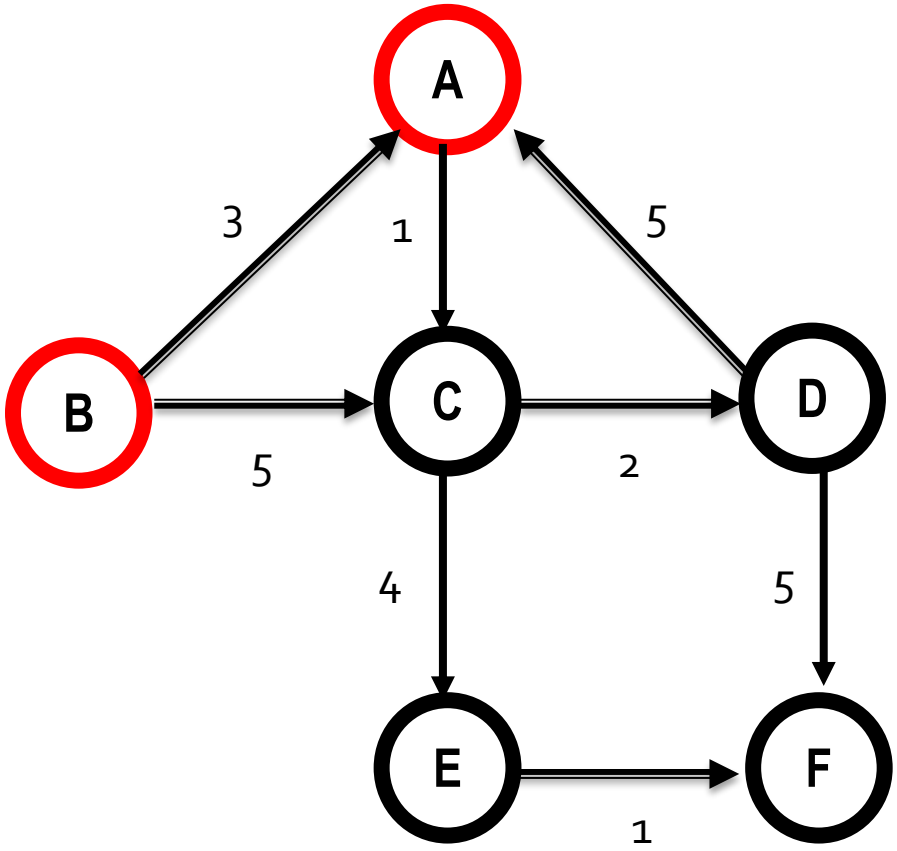
Distance?
Path?
Shortest paths?

Example: The Source Vertex B



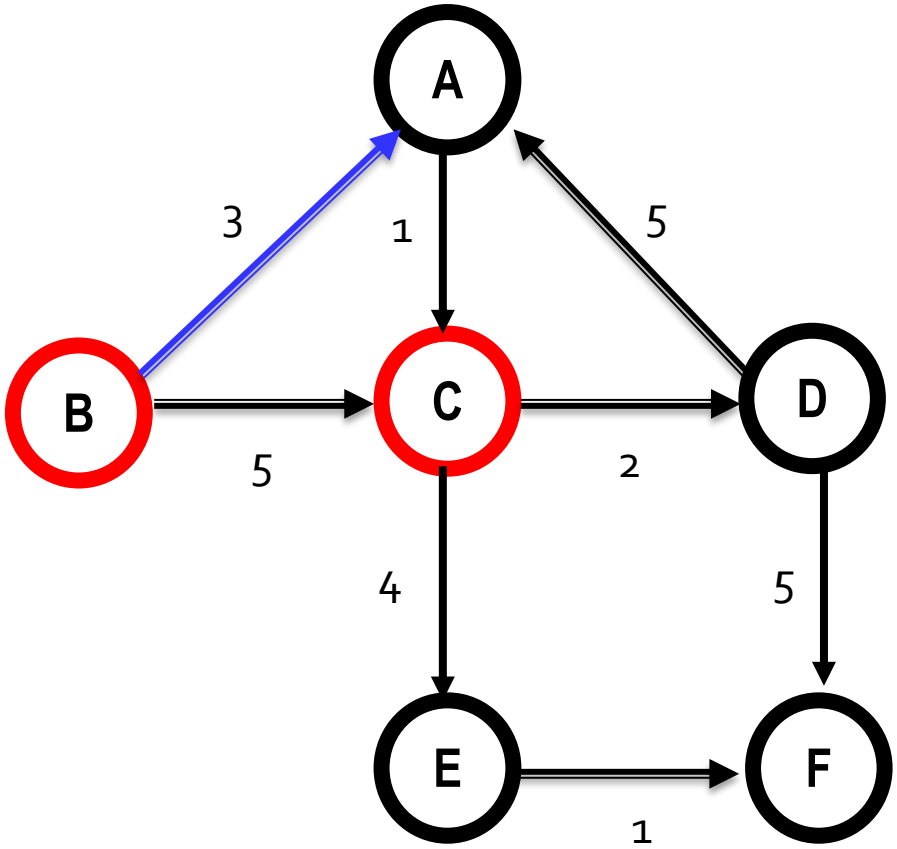
	A	B	C	D	E	F
SPfound	f	f	f	f	f	f
Distance	∞	0	∞	∞	∞	∞
Path	-	-	-	-	-	-

Example: SSSP by Dijkstra's Algorithm



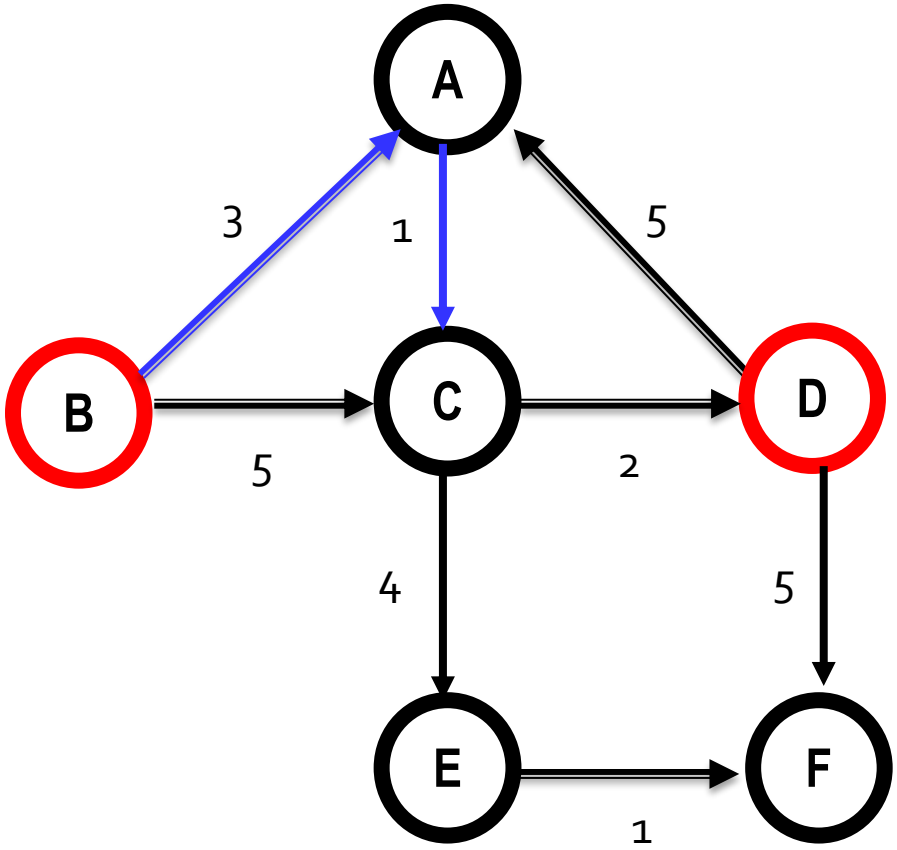
	A	B	C	D	E	F
SPfound	f	t	f	f	f	f
Distance	3	0	5	∞	∞	∞
Path	B	-	B	-	-	-

Example: SSSP by Dijkstra's Algorithm



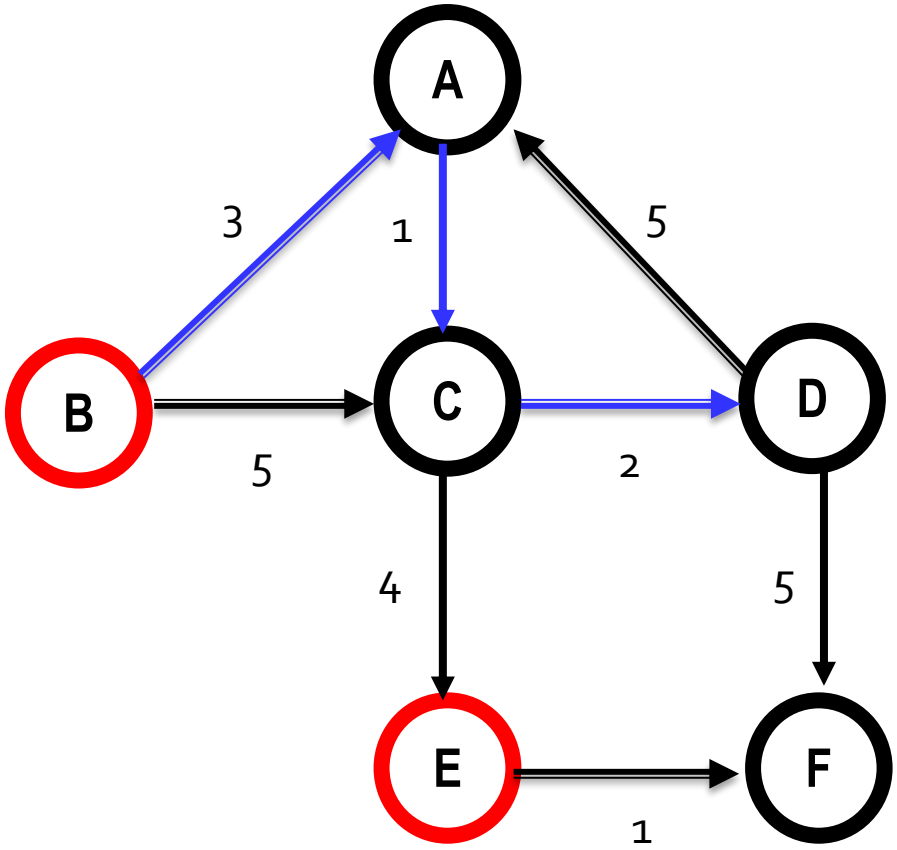
	A	B	C	D	E	F
K	t	t	f	f	f	f
Distance	3	0	4	∞	∞	∞
Path	B	-	A	-	-	-

Example: SSSP by Dijkstra's Algorithm



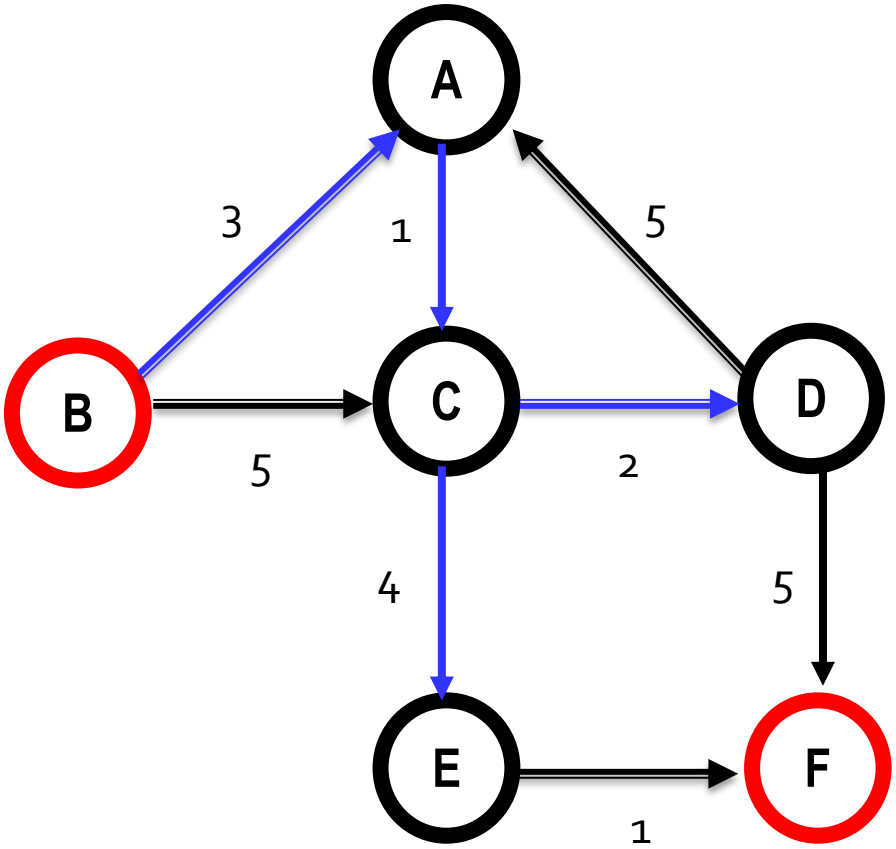
	A	B	C	D	E	F
SPfound	t	t	t	f	f	f
Distance	3	0	4	6	8	∞
Path	B	-	A	C	C	-

Example: SSSP by Dijkstra's Algorithm



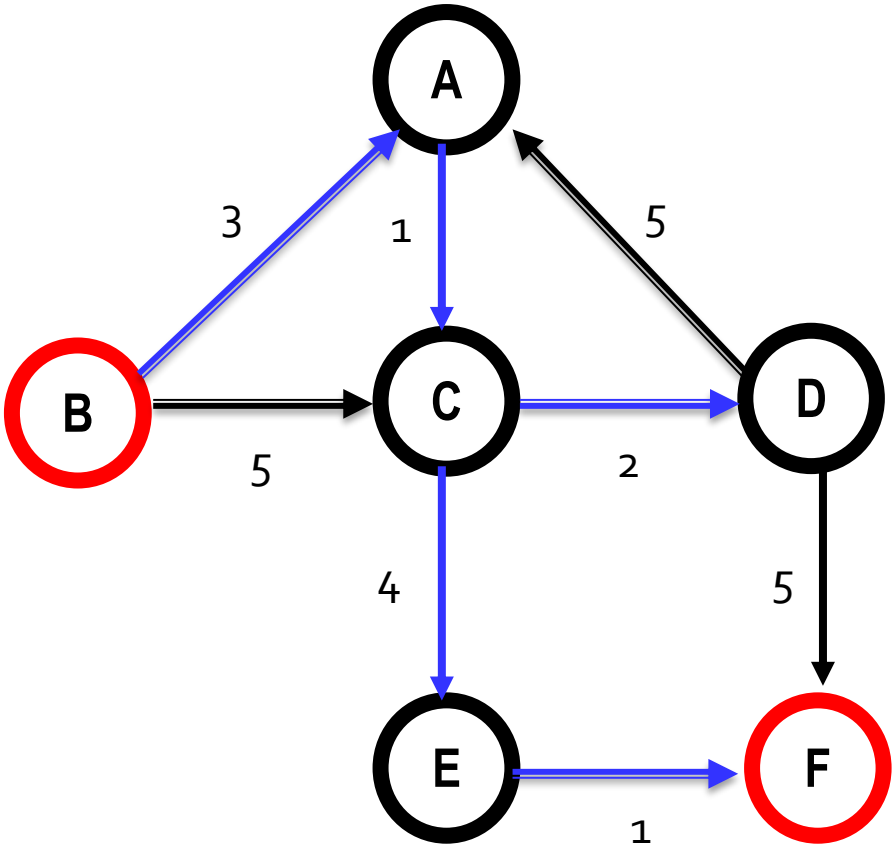
	A	B	C	D	E	F
SPfound	t	t	t	t	f	f
Distance	3	0	4	6	8	11
Path	B	-	A	C	C	D

Example: SSSP by Dijkstra's Algorithm



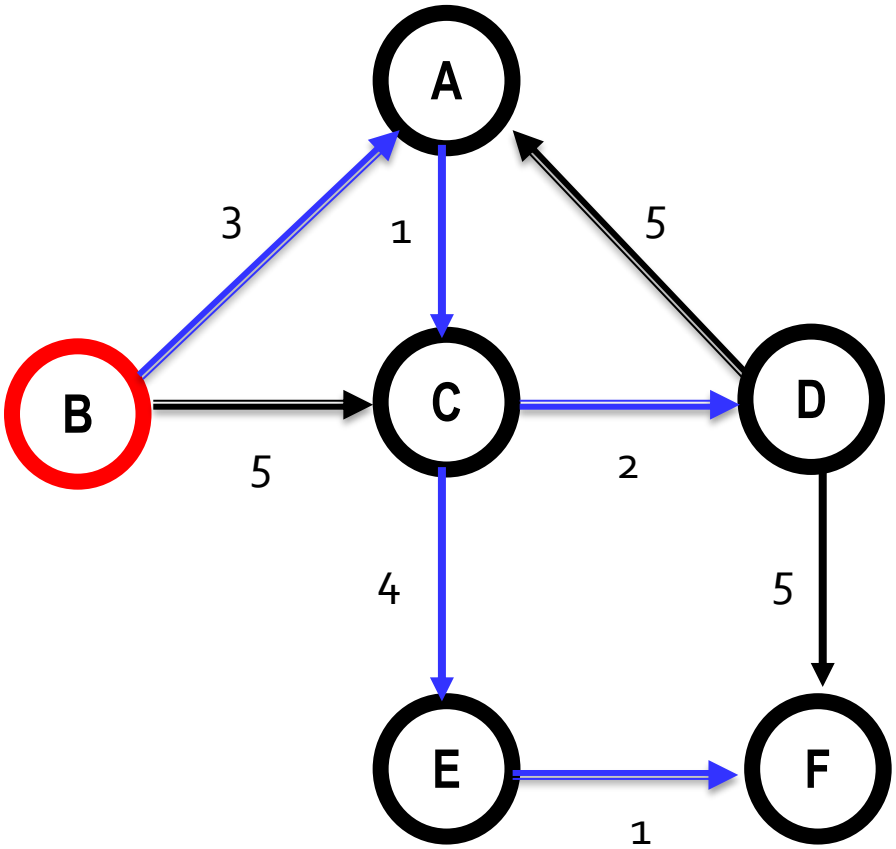
	A	B	C	D	E	F
SPfound	t	t	t	t	t	f
Distance	3	0	4	6	8	9
Path	B	-	A	C	C	E

Example: SSSP by Dijkstra's Algorithm



	A	B	C	D	E	F
SPfound	t	t	t	t	t	f
Distance	3	0	4	6	8	9
Path	B	-	A	C	C	E

Example: SSSP by Dijkstra's Algorithm



SPT(Shortest Path Tree)

	A	B	C	D	E	F
SPfound	t	t	t	t	t	t
Distance	3	0	4	6	8	9

Path	B	-	A	C	C	E

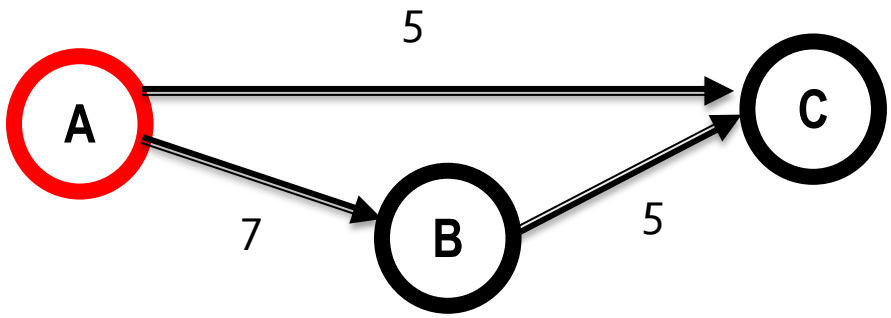
Shortest Paths from B:

B-A (3)
B-A-C (4)
B-A-C-D (6)
B-A-C-E (8)
B-A-C-E-F (9)

✓Dijkstra's SSSP Algorithm via Greedy

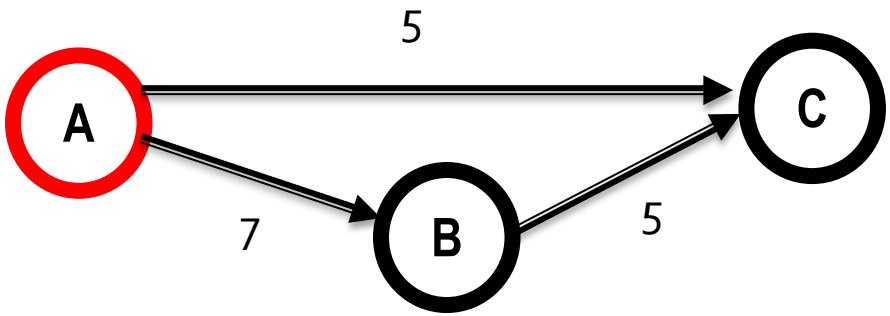
- **Algorithm 4.3 Dijkstra's Algorithm**
  - Input: W (adjacency matrix)
  - Output: Distance (length) & Path (touch)
- Analysis of Algorithm 4.3 Dijkstra's Algorithm
  - **$O(|V|)^2$)** for adjacency matrix representation
- Print Shortest Paths from Path
  - **O(|V|)**
- Dijkstra's algorithm always produces SSSPs.

▶QUIZ? Dijkstra's SSSP Algorithm via Greedy



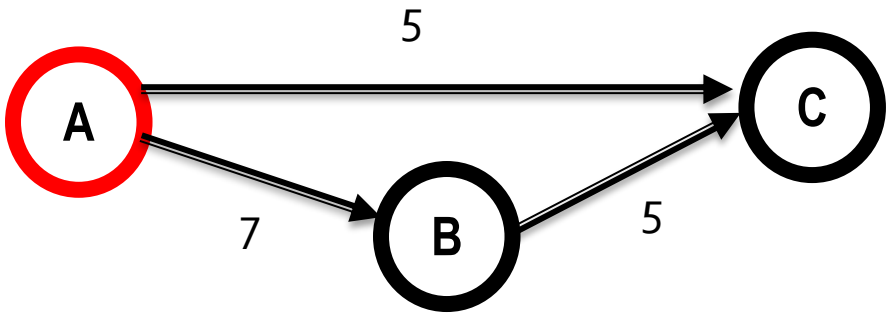
Distance?
Path?
Shortest paths?

▶QUIZ: Dijkstra's SSSP Algorithm via Greedy



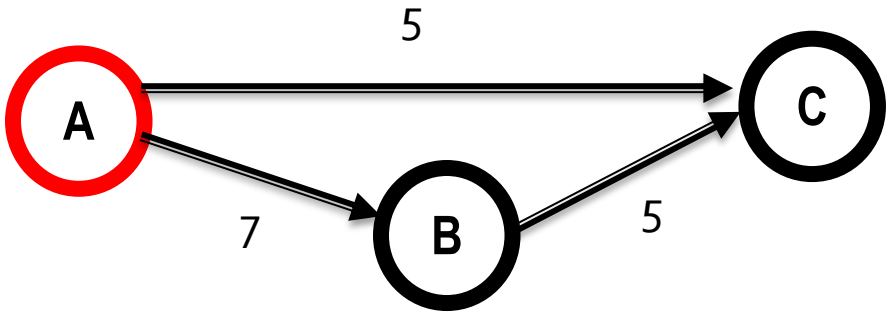
	A	B	C
SPfound	f	f	f
Distance	0	∞	∞
Path	-	-	-

▶QUIZ: Dijkstra's SSSP Algorithm via Greedy



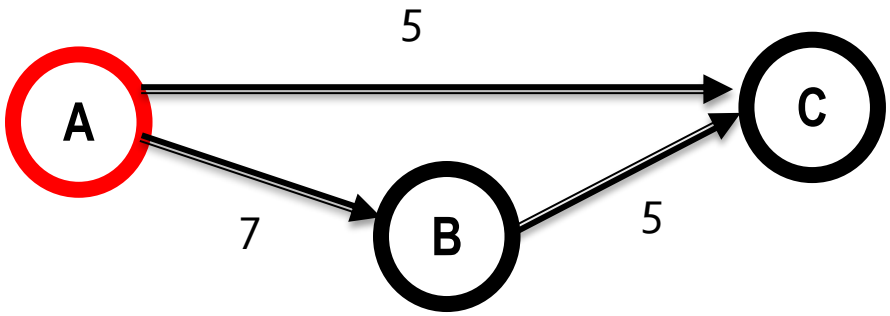
	A	B	C
SPfound	t	f	f
Distance	0	7	5
Path	-	A	A

▶QUIZ: Dijkstra's SSSP Algorithm via Greedy



	A	B	C
SPfound	t	f	t
Distance	0	7	5
Path	-	A	A

▶QUIZ: Dijkstra's SSSP Algorithm via Greedy

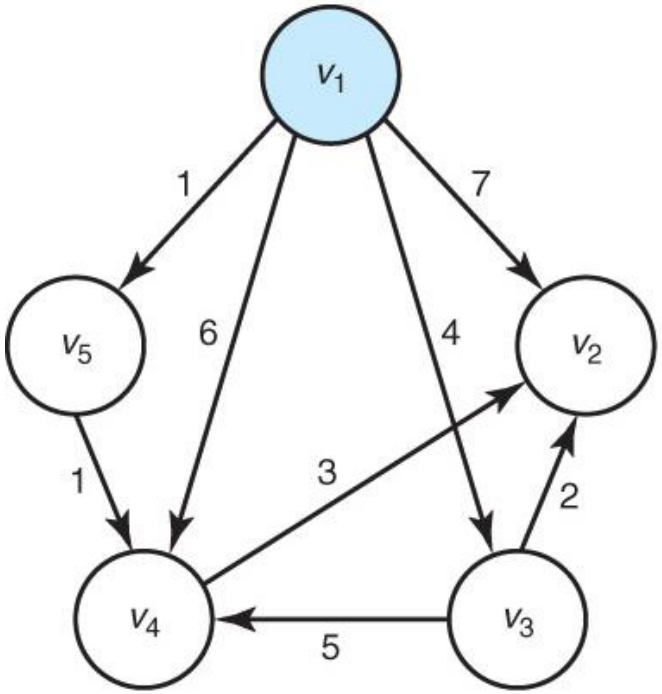


SPT(Shortest Path Tree)

	A	B	C
SPfound	t	t	t
Distance	0	7	5
Path	-	A	A

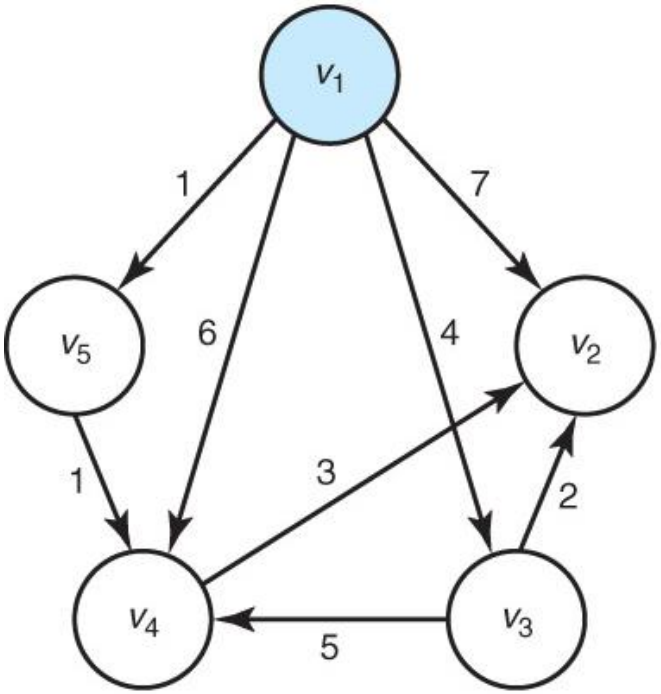
Shortest Paths from A:
A-B (7)
A-C (5)

▶QUIZ? Dijkstra's SSSP Algorithm via Greedy



Distance = ?
Path = ?
Shortest paths?

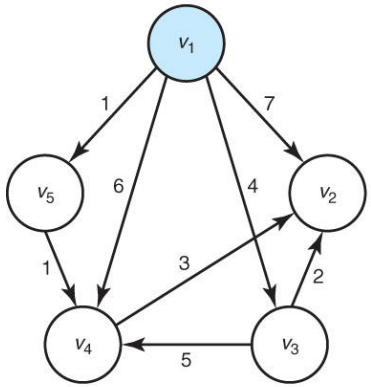
▶QUIZ? Dijkstra's SSSP Algorithm via Greedy



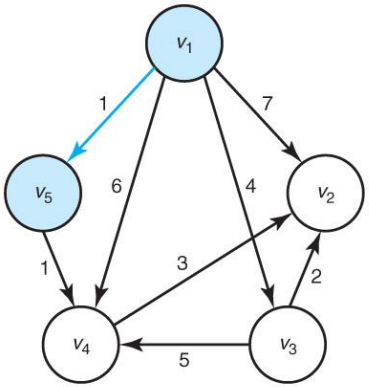
SPT(Shortest Path Tree)

▶QUIZ: Dijkstra's SSSP Algorithm via Greedy

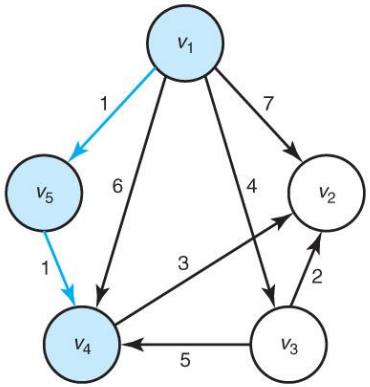
Compute shortest paths from v_1.



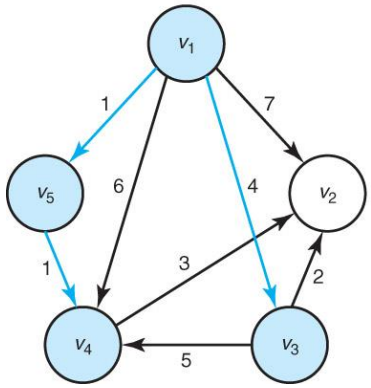
1. Vertex v_5 is selected because it is nearest to v_1.



2. Vertex v_4 is selected because it has the shortest path from v_1 using only vertices in $\{v_5\}$ as intermediates.



3. Vertex v_3 is selected because it has the shortest path from v_1 using only vertices in $\{v_4, v_5\}$ as intermediates.



4. The shortest path from v_1 to v_2 is $[v_1, v_5, v_4, v_2]$.

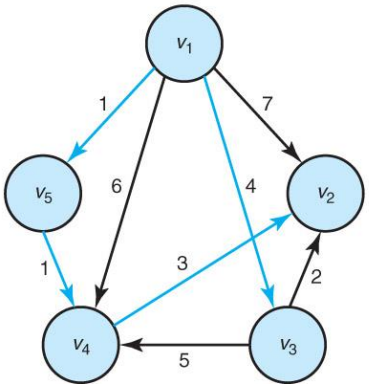
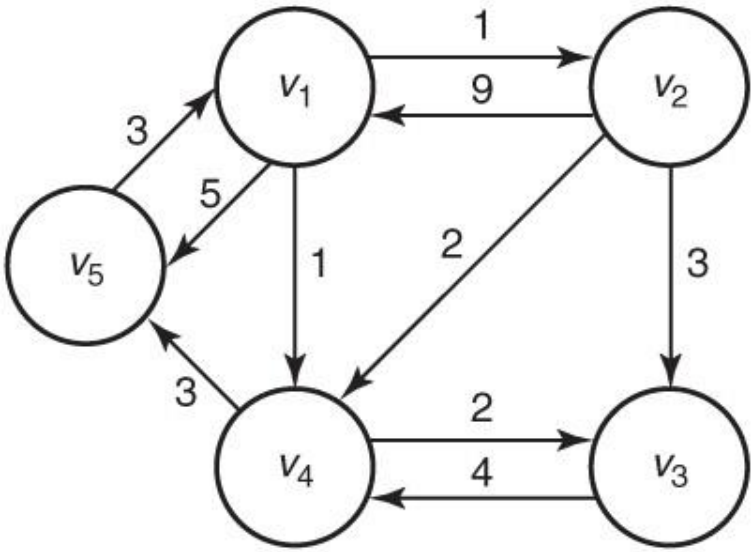


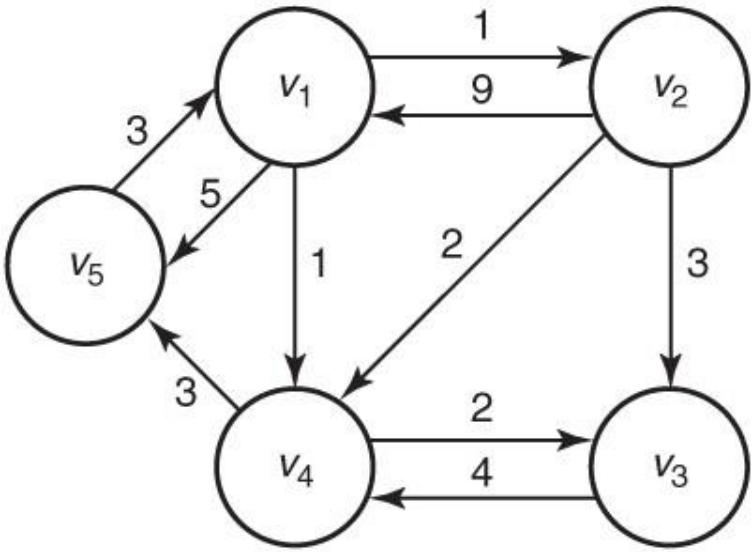
Figure 4.8

▶QUIZ? Dijkstra's SSSP Algorithm via Greedy



Distance = ?
Path = ?
Shortest paths?

▶QUIZ? Dijkstra's SSSP Algorithm via Greedy

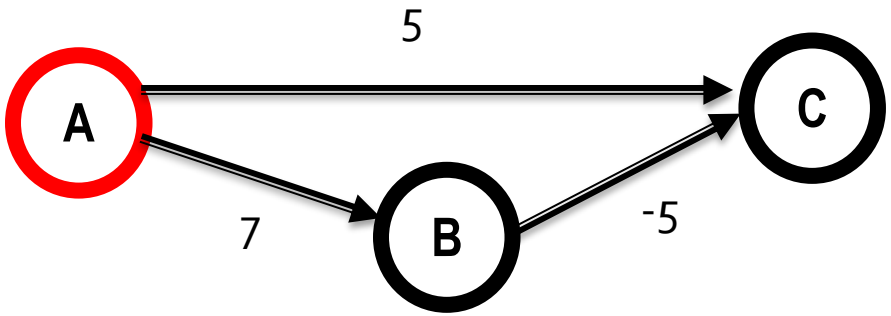


SPT(Shortest Path Tree)

Dijkstra's SSSP Algorithm via Greedy

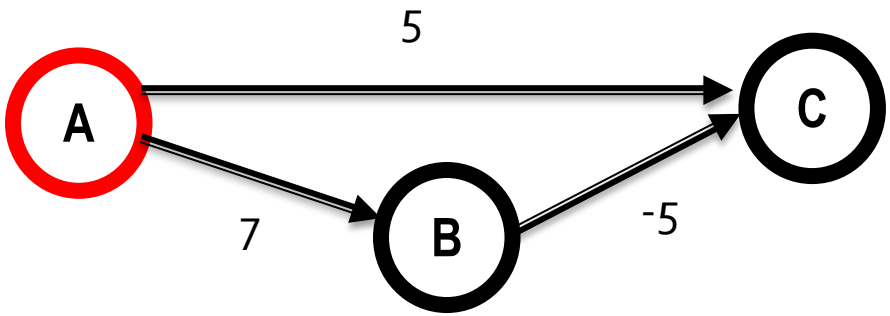
- Negative weights and Dijkstra's algorithm?

▶QUIZ? Dijkstra's SSSP Algorithm via Greedy



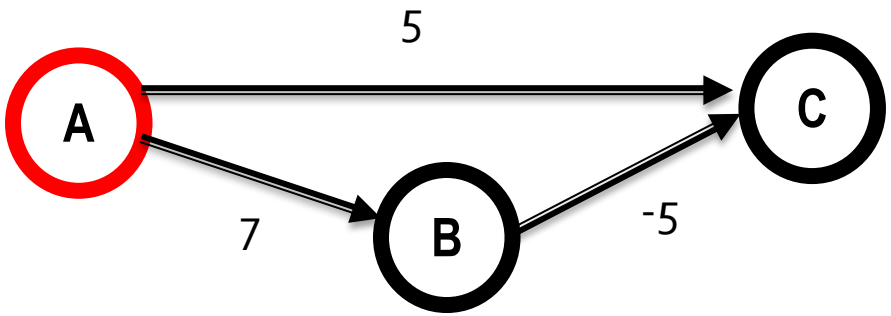
Distance?
Path?
Shortest paths?

▶QUIZ? Dijkstra's SSSP Algorithm via Greedy



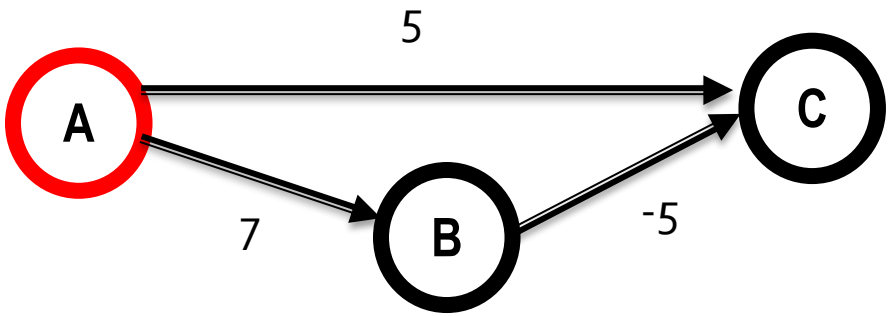
	A	B	C
SPfound	f	f	f
Distance	0	∞	∞
Path	-	-	-

▶QUIZ: Dijkstra's SSSP Algorithm via Greedy



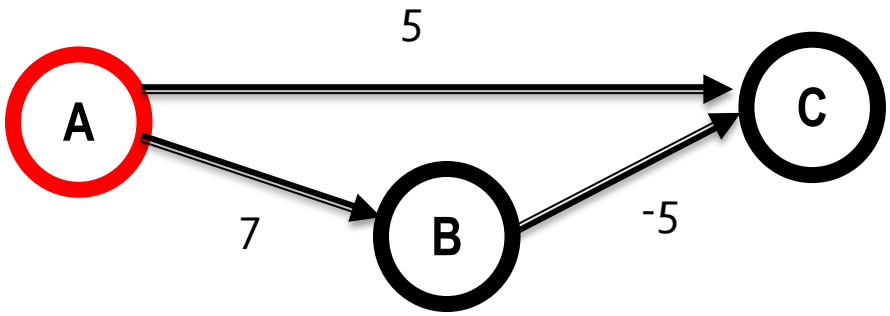
	A	B	C
SPfound	t	f	f
Distance	0	7	5
Path	-	A	A

▶QUIZ: Dijkstra's SSSP Algorithm via Greedy



	A	B	C
SPfound	t	f	t
Distance	0	7	5
Path	-	A	A

▶QUIZ: Dijkstra's SSSP Algorithm via Greedy



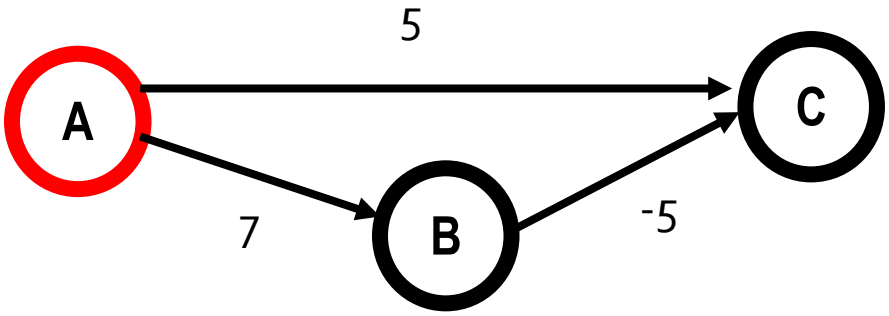
	A	B	C
SPfound	t	t	t
Distance	0	7	5
Path	-	A	A

Shortest Paths from A:
A-B (7)
A-C (5) Wrong!!!!

A-B-C

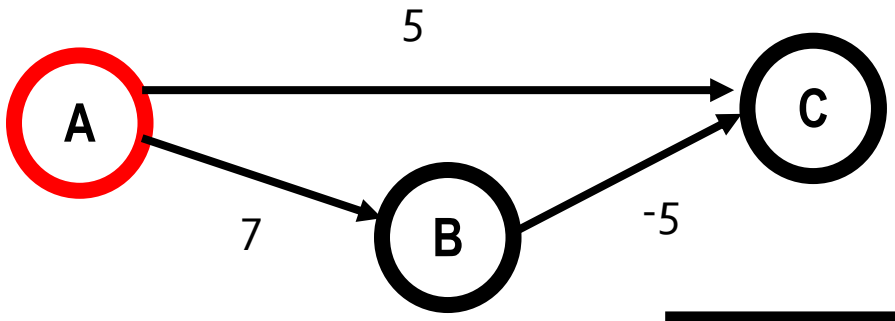
▶QUIZ: Bellman-Ford Algorithm?

Bellman-Ford Algorithm?



Distance?
Path?
Shortest paths?

QUIZ: Bellman & Ford's Algorithm



Shortest Paths from A:
A-B (7)
A-B-C (2)

|V| = 3

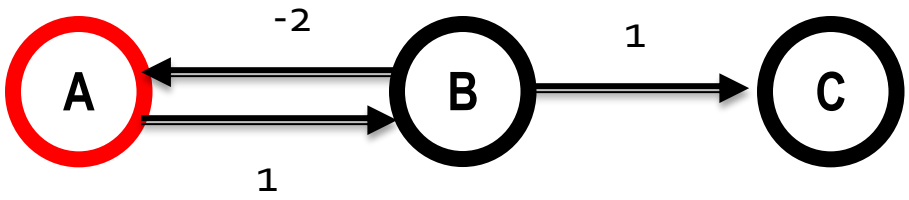
		A	B	C
k=0	Distance	0	∞	∞
k=1	Distance	0	7	5
k=2	Distance	0	7	2

k=0	Path	-	-	-
k=1	Path	-	A	A
k=2	Path	-	A	B

▶QUIZ?

- Dijkstra's algorithm with **negative edge weights**?
  - No!
  - **DP**-based **Bellman-Ford** Algorithm!
  - An example?

▶QUIZ: Negative Cycles? - SSSP From A by Dijkstra's Algorithm



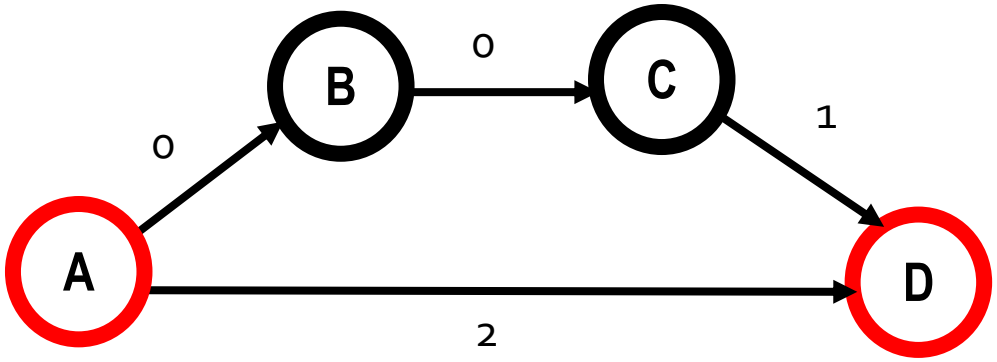
Distance?
Path?
Shortest paths?

▶QUIZ?

- Dijkstra's algorithm if we **double** of the original weight of every edge?
  - No problem!
  - Miles to Kilometers

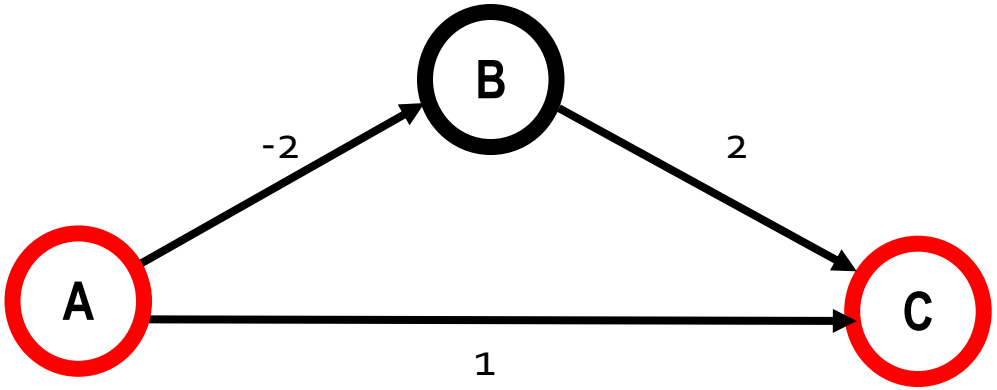
QUIZ?

- Dijkstra's algorithm if we **increase** the weight of every edge **by 1**?
  - No!
  - An example?



QUIZ?

- **Make non-negative** by **adding the largest negative weight** + Dijkstra's algorithm ?
  - No!
  - An example?



▶QUIZ?

- Compare Dijkstra's algorithm vs Bellman-Ford's algorithm?

▶QUIZ?

- For **all nonnegative** edge weights, Dijkstra's algorithm and Bellman-Ford's algorithm produce the **same shortest paths**?
  - Not always the same shortest path!

▶QUIZ?

- Compare Dijkstra's algorithm vs Bellman-Ford's algorithm vs Floyd-Warshall's algorithm?

▶QUIZ?

- Compare Dijkstra's algorithm vs Prim's algorithm?

▶QUIZ?

- The **SSSP** problem? Greedy Approach? DP?
  - Without negative edges - Greedy Approach? Yes! **Dijkstra's**
  - With negative edges – Greedy Approach? No!
  - With negative edges – DP? Yes! **Bellman-Ford's**

Dijkstra's SSSP Algorithm via Greedy Visualization

- *Dijkstra's SSSP Algorithm via Greedy Visualization*

4. The Scheduling Problem

a. Job Scheduling Problem to **minimize** the total time
b. Job Scheduling Problem to **maximize** the total profit

a. The Minimizing Total Time in the System Problem

- A scheduling problem to minimize the total amount of waiting time and service time.
  - **Minimizing** total time in the system

The Minimizing Total Time in the System Problem

- A simple solution:
  - Consider all possible schedules and compute the minimum total time in the system.
- Example:
  - Job1= 5, Job2= 10, Job3= 4
  - **3! = 6 possible schedules**

Schedule	Total time in the system
[1, 2, 3]	39
[1, 3, 2]	33
[2, 1, 3]	44
[2, 3, 1]	43
[3, 1, 2]	32
[3, 2, 1]	37

  - job3->job1->job2 (total time= 32)

The Minimizing Total Time in the System Problem

- If we have *n* jobs, then the number of different schedules is *n*!.
- The algorithm that considers all possible schedules is factorial-time.

- Idea?

The Minimizing Total Time in the System Problem via Greedy Approach

- A Greedy Approach:
  - The only schedule that minimizes the total time in the system is one that **gets the shortest jobs done first**, i.e. schedules jobs in non-decreasing order by **service time**.
- Correctness proof (Theorem 4.3)
- **O(n log n)**

The Minimizing Total Time in the System Problem via Greedy Approach

- Example:
  - Job1= 5, Job2= 10, Job3= 4
  - Job3= 4, Job1= 5, Job2= 10
  - job3->job1->job2 (total time= 32)

Schedule	Total time in the system
[1, 2, 3]	39
[1, 3, 2]	33
[2, 1, 3]	44
[2, 3, 1]	43
[3, 1, 2]	32
[3, 2, 1]	37

b. The Maxmizing Total Profit Scheduling Problem

- Another scheduling problem when each job takes the same amount of time to complete, but each job also has a deadline by which it must start to yield a profit associated with the job.
- The goal is to schedule the jobs so as to **maximize** the total profit.
  - **Scheduling with deadlines**

Job	Deadline	Profit
1	2	30
2	1	35
3	2	25
4	1	40

The Scheduling with Deadlines Problem

- Each job takes one unit of time.
- If job starts before or at its deadline, profit is obtained, otherwise no profit.
- Goal is to schedule jobs to maximize the total profit.
- Not all the jobs have to be scheduled!

The Scheduling with Deadlines Problem

- A sequence is a feasible sequence, if all the jobs in the sequence start by their deadlines.
- A set of jobs is called feasible set if there is at least one feasible sequence for the jobs in the set.
- A feasible sequence with maximum total profit is called optimal sequence.
- The set of jobs in the optimal sequence is called optimal set of jobs.

The Scheduling with Deadlines Problem

- A simple solution:
  - Considers all possible schedules, and it takes factorial time!

- Example:

Job	Deadline	Profit
1	2	30
2	1	35
3	2	25
4	1	40

[4, 1] is a feasible sequence
[1, 4] is not a feasible sequence

Schedule	Total Profit
[1, 3]	30 + 25 = 55
[2, 1]	35 + 30 = 65
[2, 3]	35 + 25 = 60
[3, 1]	25 + 30 = 55
[4, 1]	40 + 30 = 70
[4, 3]	40 + 25 = 65

The Scheduling with Deadlines Problem via Greedy Approach

- A Greedy Approach:
  - A reasonably greedy approach would be to **sort the jobs in non-increasing order by profit**.
    - Select the **highest profit job first**.
    - Add it to the **schedule if possible (feasible)**.

The Scheduling with Deadlines Problem via Greedy Approach

- Example:

Job	Deadline	Profit
1	2	30
2	1	35
3	2	25
4	1	40

Schedule	Total Profit
[1, 3]	30 + 25 = 55
[2, 1]	35 + 30 = 65
[2, 3]	35 + 25 = 60
[3, 1]	25 + 30 = 55
[4, 1]	40 + 30 = 70
[4, 3]	40 + 25 = 65

- Job4 -> Job2 -> Job 1 -> Job 3
- job4->job1 (total profit= 70)

The Scheduling with Deadlines Problem via Greedy Approach

- **Algorithm 4.4** Scheduling with Deadlines Problem via Greedy Approach
- **$O(n^2)$**
- Correctness proof (Theorem 4.4)

5. The Knapsack Problem

- Given a knapsack (rucksack) with maximum weight *W* and a set *S* of *n* items, each with weight *w* and profit (value) *p*, i.e.,
  - $S = \{ (item_1, w_1, p_1), (item_2, w_2, p_2), \ldots, (item_n, w_n, p_n) \}$
- Determine the count of each item to include in the knapsack so that
  - The total weight is less than or equal to W and
  - The total profit (value) is **as large as possible**.

✓The Knapsack Problem

a. **The fractional Knapsack problem**
   - Can take a fractional number of items!
   - Easier than 0-1?

b. **The 0-1 Knapsack problem**
   - For each item, only 1 copy is available & not allowed to break!

a. The Fractional Knapsack

The Fractional Knapsack Problem

- The fractional Knapsack problem
  - Can take a fractional number of items!
  - Easier than 0-1?
- Idea?
  - How about Greedy Approach?

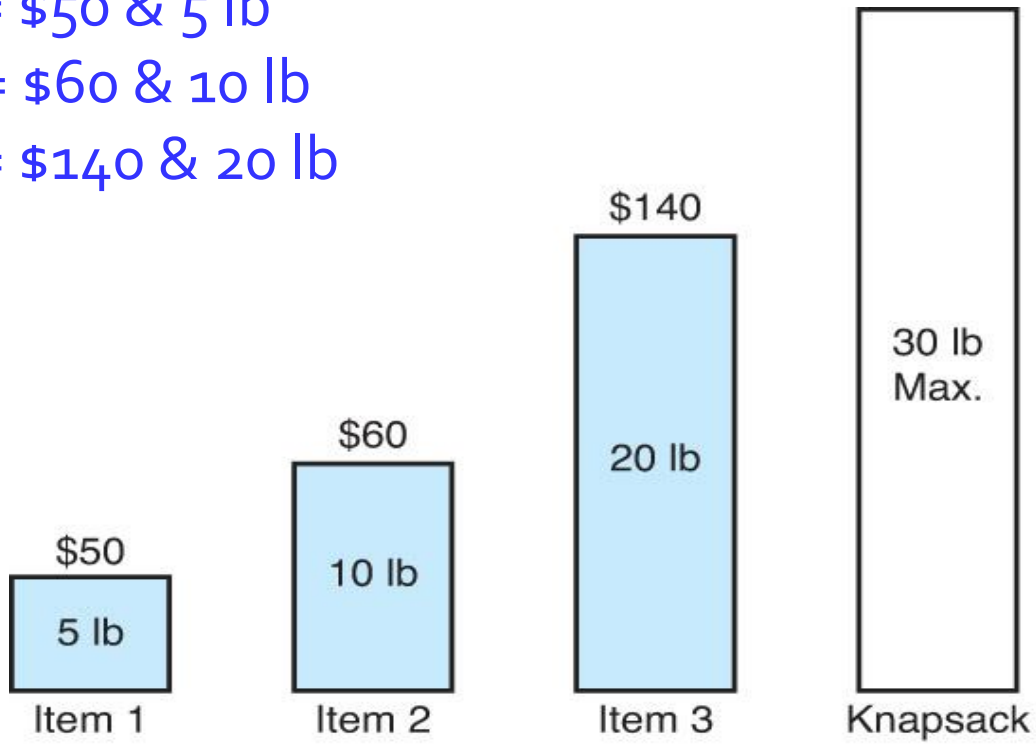
The Fractional Knapsack Problem via Greedy Approach

W = 30 lb

Item1= $50 & 5 lb

Item2= $60 & 10 lb

Item3= $140 & 20 lb



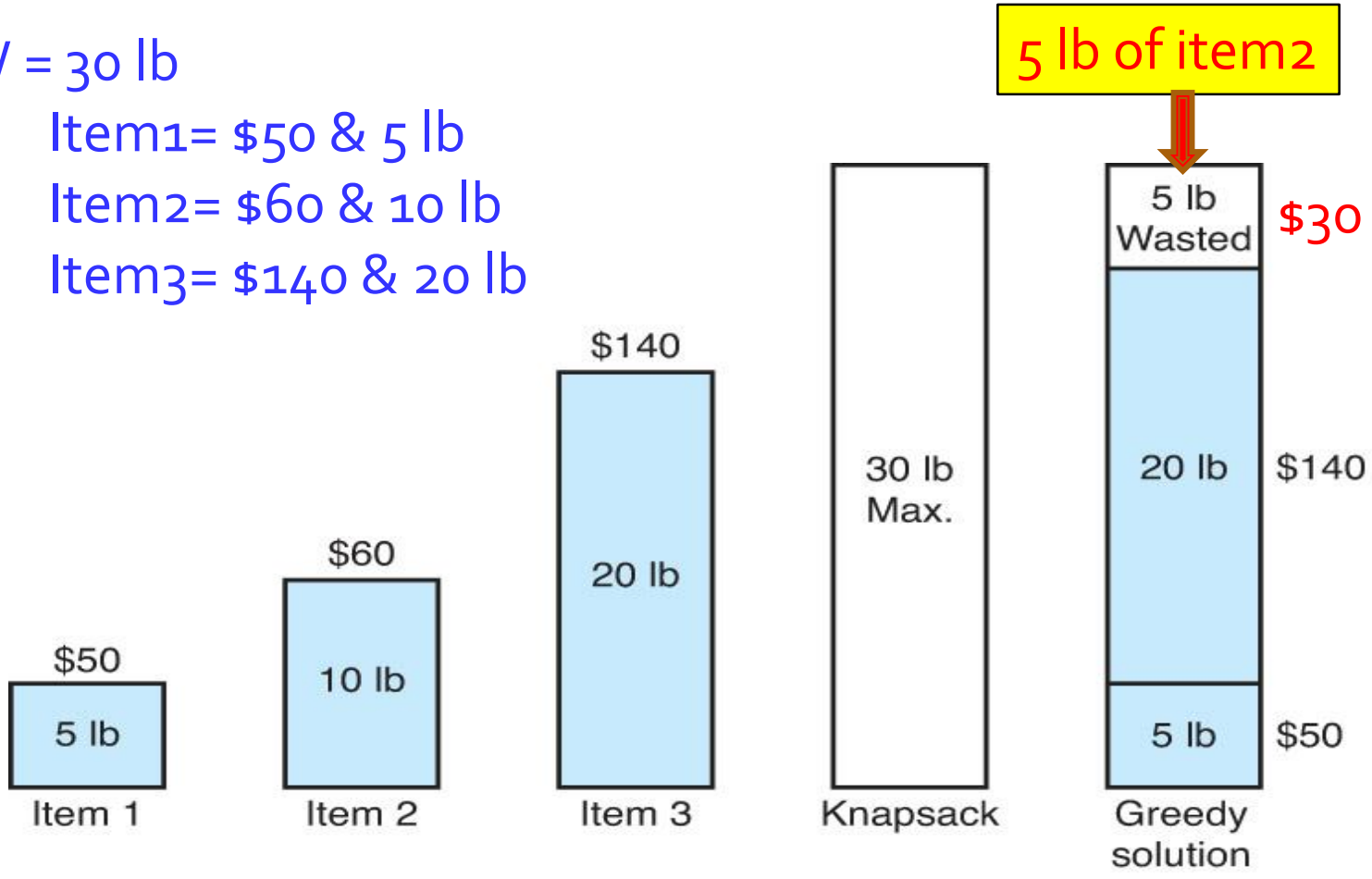
The Fractional Knapsack Problem via Greedy Approach

W = 30 lb

Item1= $50 & 5 lb

Item2= $60 & 10 lb

Item3= $140 & 20 lb



The Fractional Knapsack Problem via Greedy Approach

- Greedy Approach:
  - **Pick the items that have the largest profit per unit weight first.**
- It always leads to an optimal solution.

▶QUIZ?

- **The Fractional Knapsack Problem via Greedy Approach?**
  - W = 16 lb
  - Item1= $40 & 2 lb
  - Item2= $30 & 5 lb
  - Item3= $50 & 10 lb
  - Item4= $10 & 5 lb

  - Item 1, Item 2 & item 3 **(9 lb)** (Max profit= $115)

The Fractional Knapsack Problem via Greedy Approach

- The fractional knapsack problem is **easier than** the 0-1 knapsack problem!
- The **fractional** Knapsack problem **can** be solved with a **greedy approach**!

- **O(nlog n)**

▶QUIZ?

- Greedy approach for the **Fractional** Knapsack Problem?
  - Yes!

b. The 0-1 Knapsack Problem

The 0-1 Knapsack Problem

- The 0-1 Knapsack problem
  - For each item, only 1 copy is available & not allowed to break!
- Idea?
  - How about Greedy Approach?

The 0-1 Knapsack Problem via Greedy Approach

- **Pick the items that have the largest** profit per unit weight first?

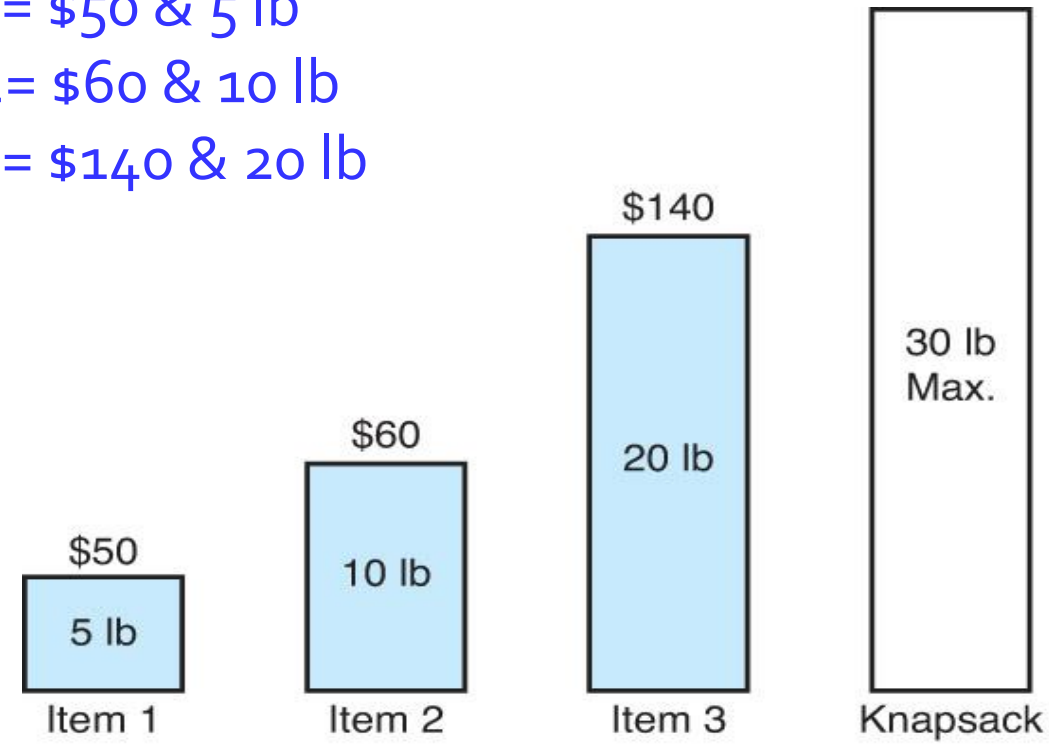
The 0-1 Knapsack Problem via Greedy Approach

W = 30 lb

Item1= $50 & 5 lb

Item2= $60 & 10 lb

Item3= $140 & 20 lb



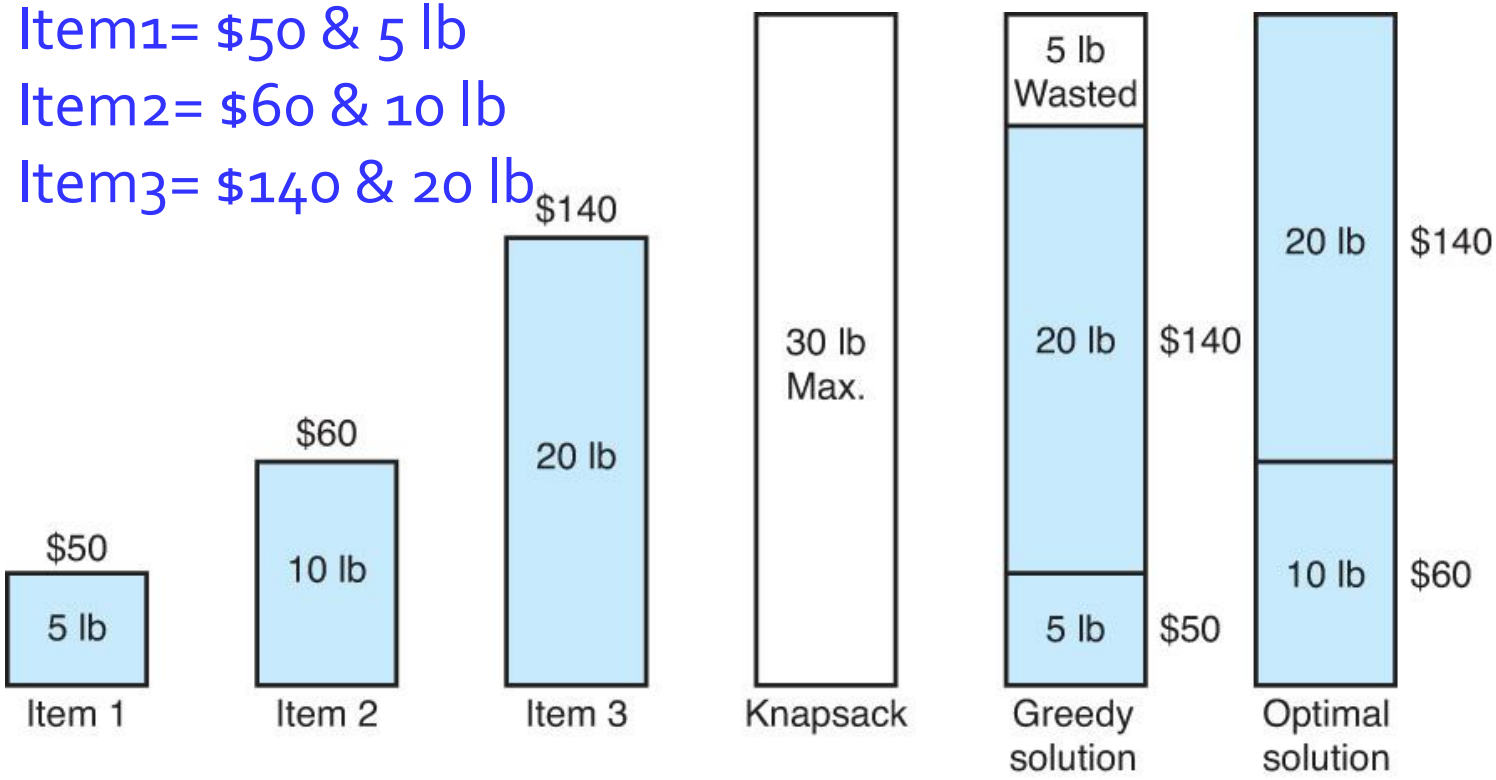
The 0-1 Knapsack Problem via Greedy Approach

W = 30 lb

Item1= $50 & 5 lb

Item2= $60 & 10 lb

Item3= $140 & 20 lb



The 0-1 Knapsack Problem via Greedy Approach

- **Pick the items that have the largest** profit per unit weight first?
- This greedy strategy **does not** solve the 0-1 Knapsack problem!

The 0-1 Knapsack Problem via Greedy Approach

- **Pick the items with the largest profit first?**
- Example:
  - W = 30 lb
  - Item1= $10 & 25 lb
  - Item2= $9 & 10 lb
  - Item3= $9 & 10 lb
- This strategy **does not** work if the most valuable item has a large weight in comparison to its profit.

The 0-1 Knapsack Problem via Greedy Approach

- **Pick the lightest items first?**
- Example:
  - W = 30 lb
  - Item1= $1 & 10 lb
  - Item2= $2 & 20 lb
  - Item3= $10 & 30 lb
- This strategy **does not** work if the most valuable item has a large weight in comparison to its profit.

▶QUIZ?

- **The 0-1 Knapsack Problem via Greedy Approach?**
  - W = 16 lb
  - Item1= $40 & 2 lb
  - Item2= $30 & 5 lb
  - Item3= $50 & 10 lb
  - Item4= $10 & 5 lb

  - Item 1 & item 2 (Max profit= $70) WRONG!

The 0-1 Knapsack Problem via Greedy Approach

- Actually, the **0-1 Knapsack** problem **cannot** be solved with Greedy Approach!

- **Then, WHAT?**

▶QUIZ?

- Greedy approach for the **0-1** Knapsack Problem?
  - No!

✓ The 0-1 Knapsack Problem via DP

The 0-1 Knapsack Problem

- **Brute-force approach?**
  - Consider all subsets of the n items.
  - Discard subsets whose total weight exceeds W.
  - Of the remaining, take the one with maximum profit.
  - **$O(2^n)$**

▶QUIZ?

- **The 0-1 Knapsack Problem via BF Approach?**
  - W = 16 lb
  - Item1= $40 & 2 lb
  - Item2= $30 & 5 lb
  - Item3= $50 & 10 lb
  - Item4= $10 & 5 lb

  - Consider all 2^4= 16 subsets of the 4 items.

  - Item 1 & item 3 (Max profit= $90)

The 0-1 Knapsack Problem

- How about **DP**?
- The principle of optimality applies.
- **So, we can apply DP to the 0-1 Knapsack problem!**

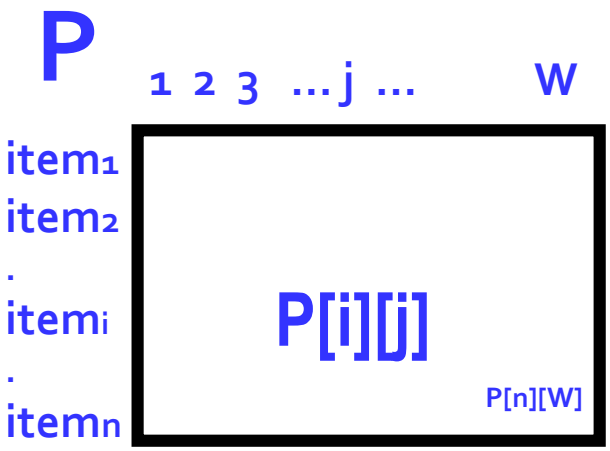
The 0-1 Knapsack Problem via DP

- Input:
  - A set S of items indexed from 1 to n, each of which has profit p_i and weight w_i, and an integer total weight capacity W.
- Output:
  - The profit of the maximum-profited set of items in S whose total weight does not exceed W.

The 0-1 Knapsack Problem via DP

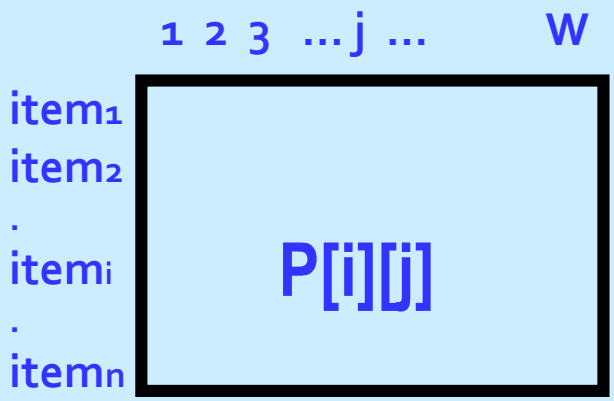
- **P[i][j]** = the max profit obtained when choosing items only from the first **i** items under the restriction that the total weight cannot exceed **j**.

- **P[n][W]**



The 0-1 Knapsack Problem via DP

- A recursive property?



P[i][j] = if w[i] > j

P[i-1][j]

else

max(P[i-1][j], p[i] + P[i-1][j – w[i]])

- P[n][W]

The 0-1 Knapsack Problem via DP

```
int knapsack(int w[1…n], int p[1…n], int W) {
        index i, j;
        int P[0…n][0…W];

        for(i=0; i<=n; i++)
                P[i][0] = 0;
        for(j=0; j<=W; j++)
                P[0][j] = 0;

        for(i=1; i<=n; i++) {
                for(j=1; j<=W; j++) {
                        if(w[i] > j)
                                P[i][j] = P[i-1][j];
                        else
                                P[i][j] = max( P[i-1][j],  p[i] + P[i-1][j – w[i]] );
                }
        }
        return P[n][W];
}
```

▶QUIZ?

- **The 0-1 Knapsack Problem via DP?**
  - W = 16 lb
  - Item1= $40 & 2 lb
  - Item2= $30 & 5 lb
  - Item3= $50 & 10 lb
  - Item4= $10 & 5 lb

  - Item 1 & item 3 (Max profit= $90)

The 0-1 Knapsack Problem via DP

- The **0-1 Knapsack** problem **can** be solved with DP!

The 0-1 Knapsack Problem via DP

- **O(nW)**
  - n = the number of items
  - W=the capacity of knapsack
- **Pseudo-polynomial time**

The 0-1 Knapsack Problem via DP

- A Refinement
- **$O(2^n)$**
  - n = the number of items
- **BUT, still exponential time!**

▶QUIZ?

- DP for the **0-1** Knapsack Problem?
  - **Yes!**
- What is the worst-case time complexity?
  - **O(nW)**
  - **O(2^n)**

Traveling Salesperson Problem via DP

- **Recall!**
- **Algorithm 3.11 DP** Algorithm for the TSP

- **$O(n^2 2^n)$**

Traveling Salesperson Problem via DP

- No one has ever found an algorithm for the Traveling Salesperson problem whose worst-case time complexity is better than exponential!
- Yet, no one has ever proved that the algorithm is not possible!

- The Traveling Salesperson Problem is **hard**.
- NP-complete!

The 0-1 Knapsack Problem via DP

- Like **TSP**
  - No one has ever found an algorithm for **the 0–1 knapsack problem** whose worst-case time complexity is better than exponential!
  - Yet no one has proven that such an algorithm is not possible!
  - The 0–1 knapsack problem is **hard**.
  - **NP-complete!**

▶QUIZ?

- Compare **Fractional Knapsack** problem vs **0-1 Knapsack** problem ?

▶QUIZ?

- The **Knapsack** problem? Greedy Approach? DP?
  - Fractional knapsack – Greedy Approach? Yes!
  - 0-1 Knapsack – Greedy Approach? No!
  - 0-1 Knapsack – DP? Yes!

▶QUIZ?

- Compare **Greedy Approach** vs **Dynamic Programming**?
  - Both solve optimization problems.
  - Greedy algorithms usually simpler, but do not always produce optimal solution – must formally prove.
  - Dynamic Programming – show principle of optimality applies.

✓ Greedy Approach-based Algorithms Summary

1. The Min-Coin Change problem via Greedy Approach
2. The MST Problem via Greedy Approach – Prim's algorithm & Kruskal's algorithm
3. The Single-Source Shortest-Paths Problem via Greedy Approach - Dijkstra's algorithm
4. The Scheduling Problem via Greedy Approach
5. The Fractional Knapsack Problem via Greedy Approach

Homework Assignment

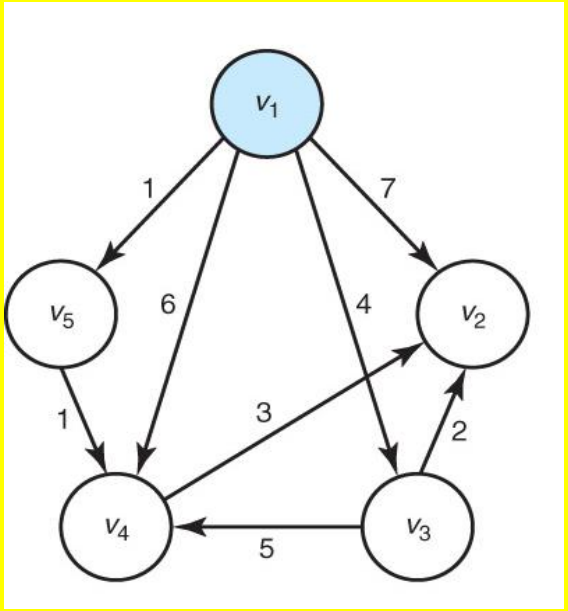
Homework Assignment?

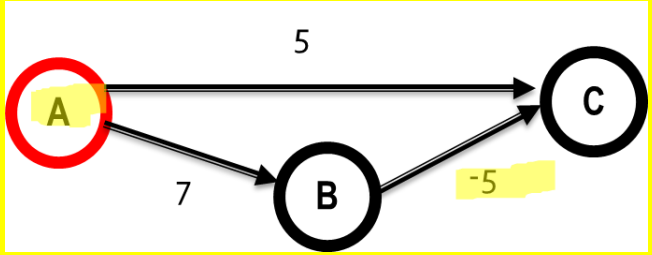
- Chapter 4: Exercise #2 & #7 and #12

Homework Assignment?

- ***Greedy Approach*: Chapter 4: Exercise #14 (Greedy Approach-based Dijkstra Algorithm for the Single Source Shortest Path Problem)**

Test Cases





✓ Textbook Readings

- Chapter 4:
  - 4.1
  - 4.2
  - 4.3
  - 4.5 (4.5.1, 4.5.2 & 4.5.3 only)

the right majors, minors &

ucation?

or students' academic and career success

ging for many students - *Many students change their*
uring college!

e prediction of student success in MMC could
dual students
d their right MMC
chieve their academic goals

END

Prof. Young Park