Graphs & Graph Algorithms 3

CMSC 132

Department of Computer Science

University of Maryland, College Park

Overview

- Graph implementation
  - Adjacency list / matrix / set
- Spanning trees
- Minimum spanning tree
  - Prim's algorithm
  - Kruskal's algorithm

Graph Implementation

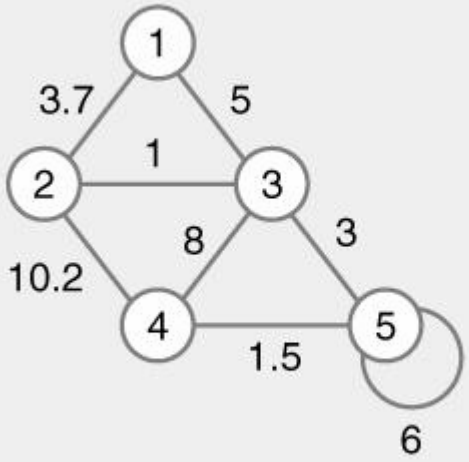
- How do we represent edges?
  - Adjacency matrix
    - 2D array of neighbors
  - Adjacency list
    - list of neighbors
  - Adjacency set/map
- Important for very large graphs
  - Affects efficiency / storage

Adjacency Matrix

- **Representation**
  - 2D array
  - Position j, k $\Rightarrow$ edge between nodes n_j, n_k
  - Unweighted graph
    - Matrix elements $\Rightarrow$ boolean
  - Weighted graph
    - Matrix elements $\Rightarrow$ weight

Adjacency Matrix

- **Example**



	1	2	3	4	5
1	∘				
2	3.7	∘			
3	5.0	1.0	∘		
4	∘	10.2	8.0	∘	
5	∘	∘	3.0	1.5	6.0

Adjacency Matrix

- **Properties**
  - Single array for entire graph
  - Only upper / lower triangle matrix needed for undirected graph
    - Since n_j, n_k implies n_k, n_j

Adjacency List

- **Representation**
  - **Linked or array list for each node of neighbors/successors**
    - **for directed graph, may need predecessors as well**
  - **Unweighted graph**
    - **store neighbor**
  - **Weighted graph**
    - **store neighbor, weight**

Adjacency List

- **Example**
  - **Unweighted graph**

Node	*Neighbor List*
1	2 → 3
2	1 → 3 → 4
3	1 → 2 → 4 → 5
4	2 → 3 → 5
5	3 → 4 → 5

  - **Weighted graph**

Node	*Neighbor List*
1	(2, 3.7) → (3, 5.0)
2	(1, 3.7) → (3, 1.0) → (4, 10.2)
3	(1, 5.0) → (2, 1.0) → (4, 8.0) → (5, 3.0)
4	(2, 10.2) → (3, 8.0) → (5, 1.5)
5	(3, 3.0) → (4, 1.5) → (5, 6.0)

Adjacency Set/Map

- For each node, store a Set or Map of neighbors/successors
  - for directed graphs, may need separate Map for predecessors
- For unweighted graphs, use a Set
- For weighted graphs, use a Map from nodes to weights

Graph Space Requirements

- **Adjacency matrix**
  - ½ N^2 entries (for graph with N nodes, E edges)
  - Many empty entries for large graphs
- **Adjacency list**
  - E entries
- **Adjacency Set/Map**
  - E entries
  - Space overhead per entry higher than for adjacency list

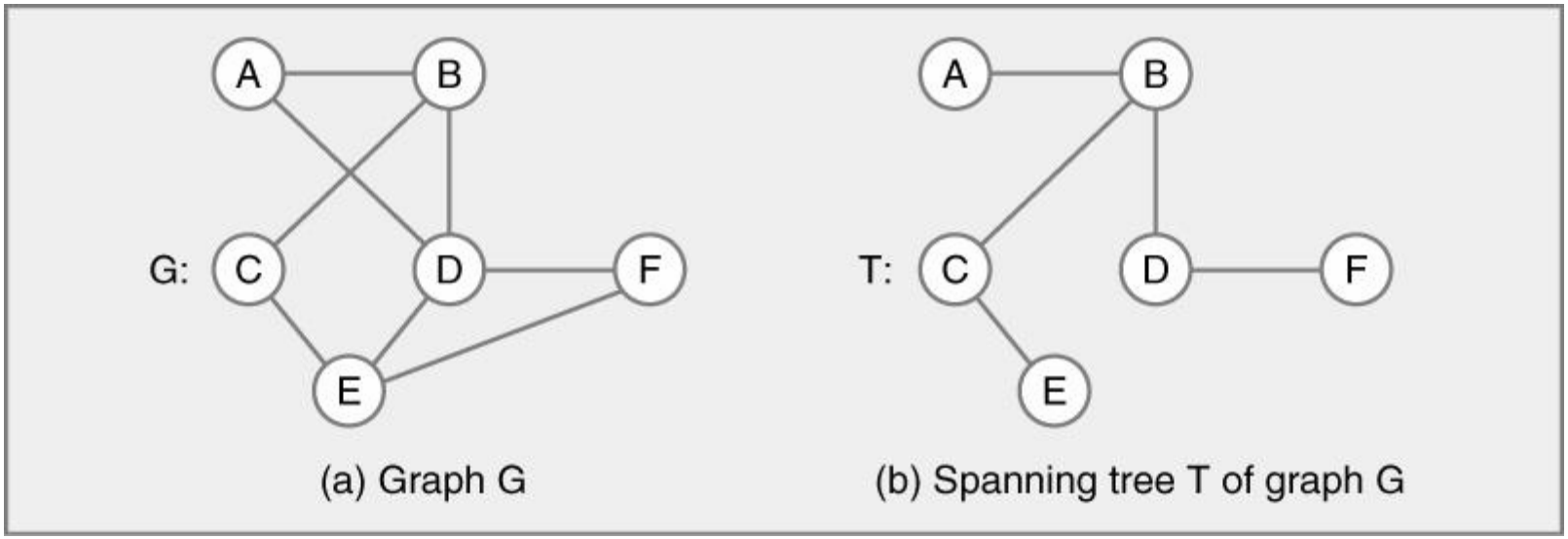
Graph Time Requirements

- **Average Complexity of operations**
  - **For graph with N nodes, E edges**

Operation	Adj Matrix	Adj List	Adj Set/Map
Find edge	O(1)	O(E/N)	O(1)
Insert edge	O(1)	O(E/N)	O(1)
Delete edge	O(1)	O(E/N)	O(1)
Enumerate edges	O(N)	O(E/N)	O(E/N)

Spanning Tree

- Set of edges connecting all nodes in graph
  - need N-1 edges for N nodes
  - no cycles, can be thought of as a tree
- Can build tree during traversal



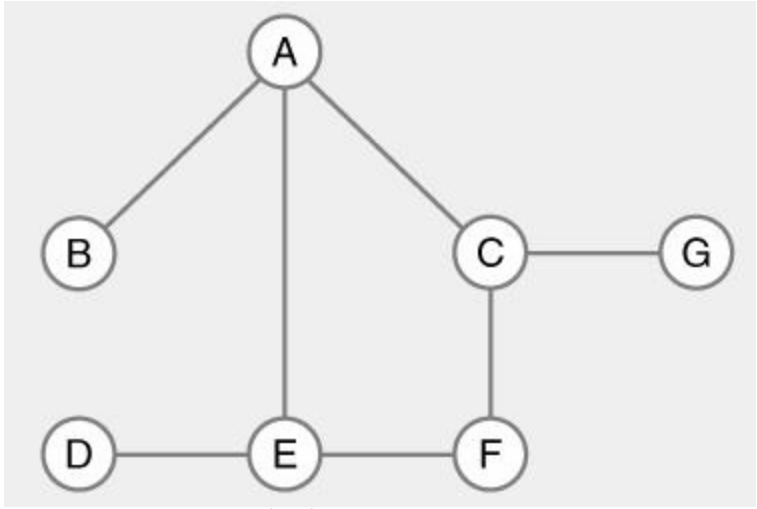
Recursive Spanning Tree Construction

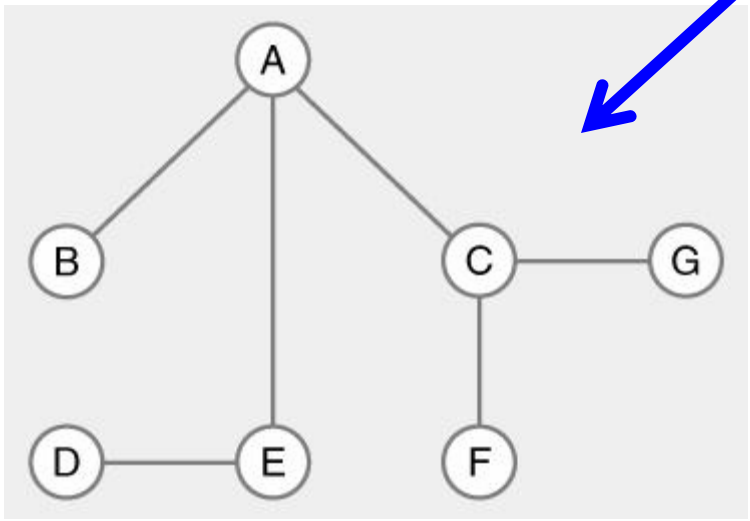
```
Known = { start }
explore ( start );

void explore (Node X) {
    for each successor Y of X
        if (Y is not in Known)
            Parent[Y] = X
            Add Y to Known
            explore(Y)
```

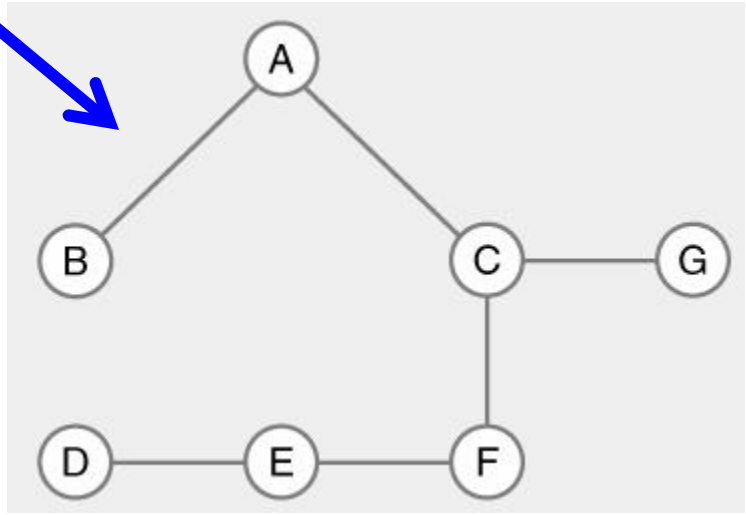
Spanning Tree Construction

```
Known = { start }
Discovered = { start }
while ( Discovered ≠ ∅ )
    take node X out of Discovered
    for each successor Y of X
            if (Y is not in Known)
                Parent[Y] = X
                Add Y to Discovered
                Add Y to Known
```

Breadth & Depth First Spanning Trees



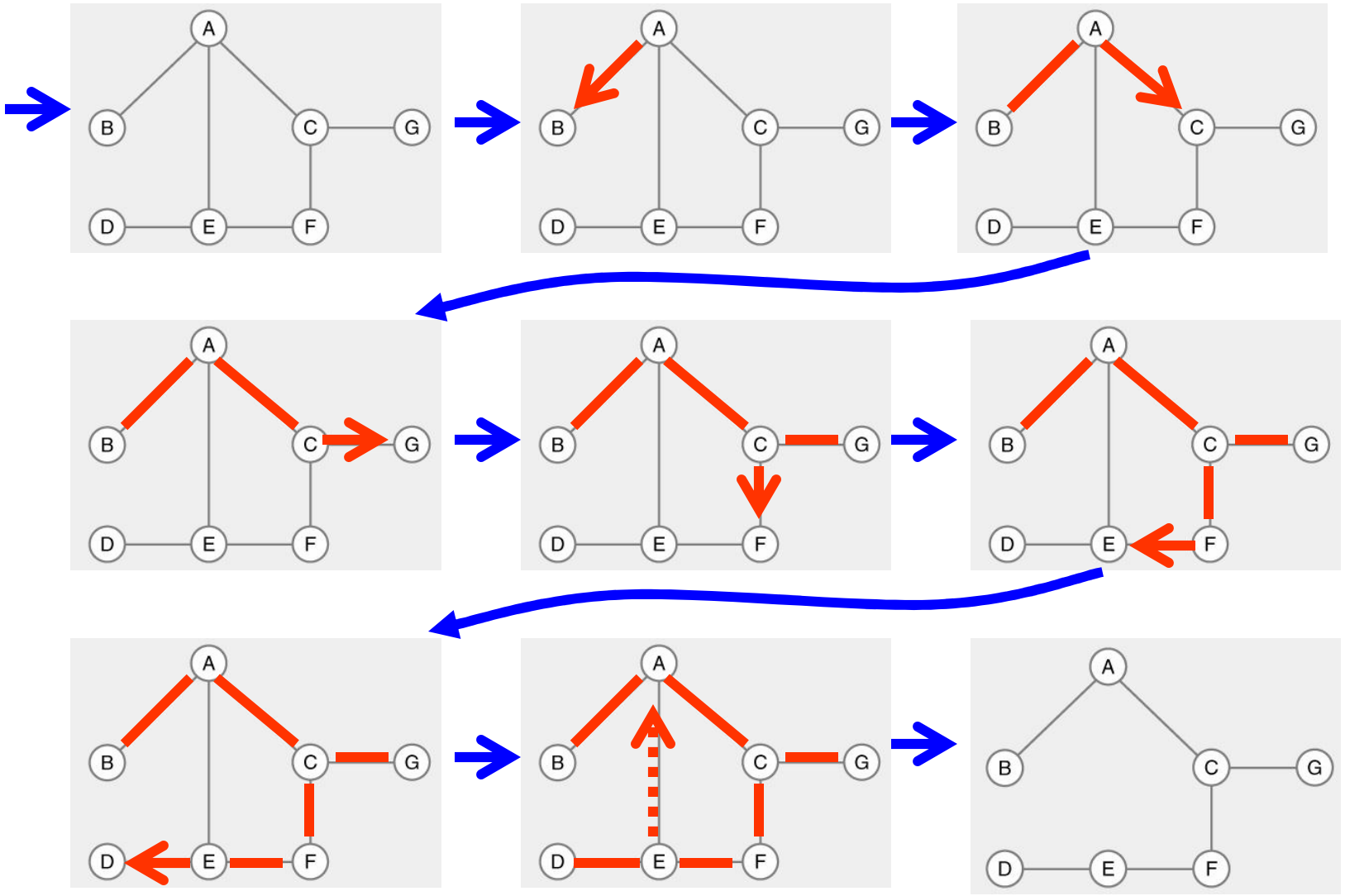




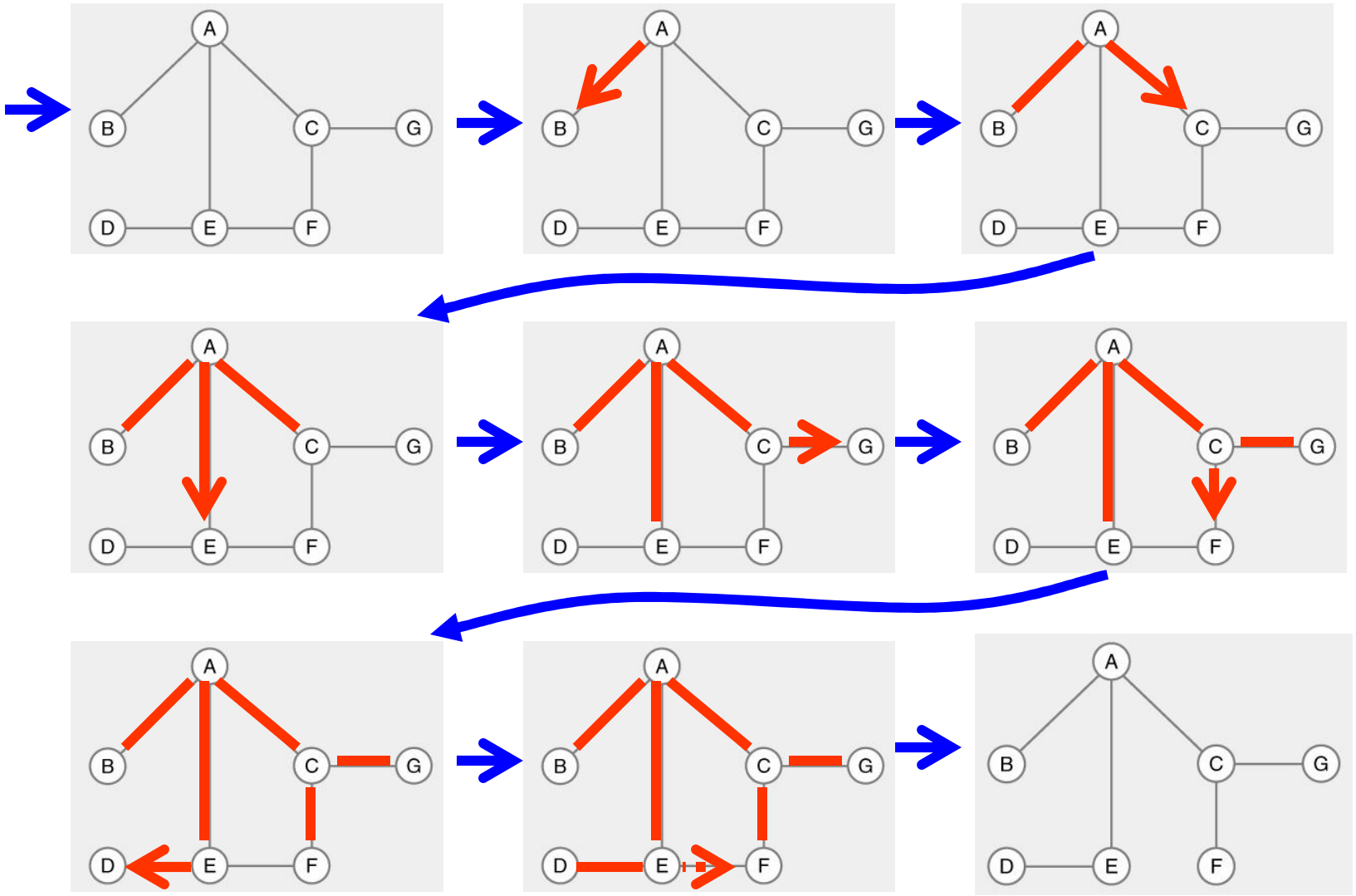
Breadth-first

Depth-first

Depth-First Spanning Tree Example



Breadth-First Spanning Tree Example

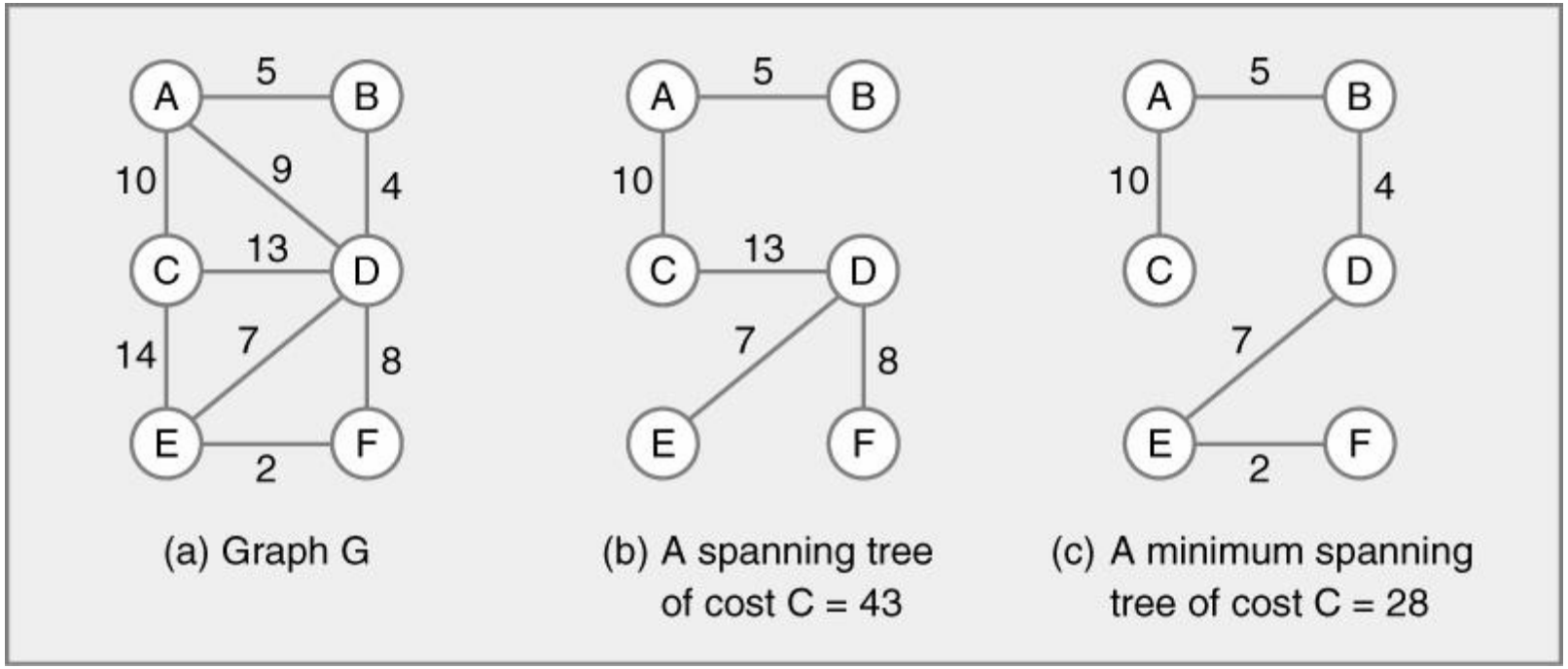


Spanning Tree Construction

- **Multiple spanning trees possible**
  - **Different breadth-first traversals**
    - **Nodes same distance visited in different order**
  - **Different depth-first traversals**
    - **Neighbors of node visited in different order**
  - **Different traversals yield different spanning trees**

Minimum Spanning Tree (MST)

- Spanning tree with minimum total edge weight
- Multiple MSTs possible (with same weight)



Algorithms for MST

- Two well known algorithms for minimum spanning tree
  - developed independently
- Prim's algorithm
  - described in book
- Kruskal's algorithm
  - Not Clyde Kruskal (prof in our department, but his uncle)

Shortest Path – Djikstra's Algorithm

```
S = {}, P[ ] = none for all nodes
C[start] = 0, C[ ] = ∞ for all other nodes
while ( not all nodes in S )
    find node K not in S with smallest C[K]
    add K to S
    for each node J not in S  adjacent to K
        if ( C[K] + cost of (K,J) < C[J] )
            C[J] = C[K] + cost of (K,J)
            P[J] = K
```

Optimal solution computed with greedy algorithm

MST – Prim's Algorithm

```
S = {}, P[ ] = none for all nodes
C[start] = 0, C[ ] = ∞ for all other nodes
while ( not all nodes in S )
    find node K not in S with smallest C[K]
    add K to S
    for each node J not in S  adjacent to K
        if ( /* C[K] + */ cost of (K,J) < C[J] )
            C[J] = /* C[K] + */ cost of (K,J)
            P[J] = K
```

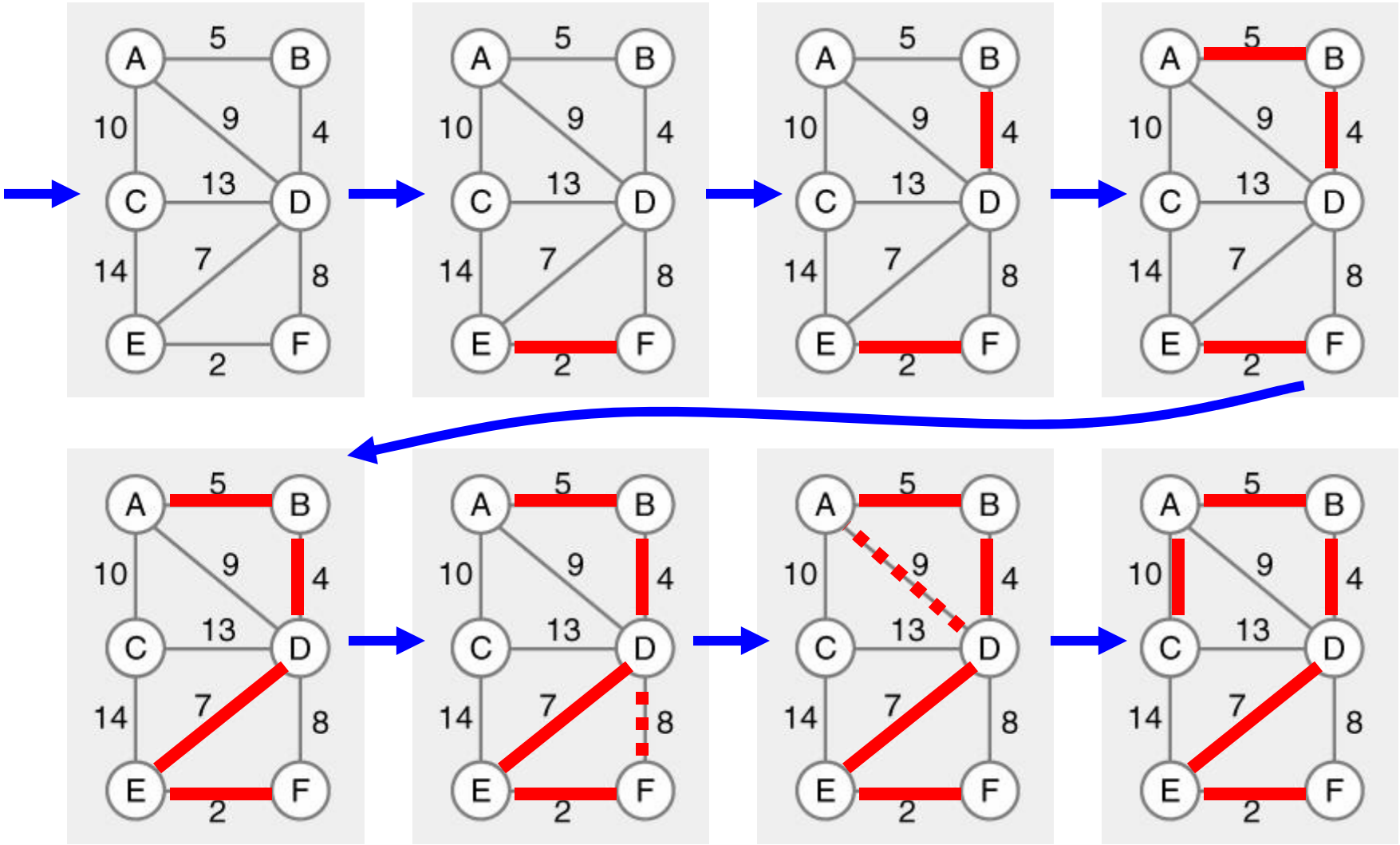
***Optimal* solution computed with *greedy* algorithm**

MST – Kruskal's Algorithm

sort edges by weight (from least to most)
tree = ∅
for each edge (X,Y) in order
  if it does not create a cycle
    add (X,Y) to tree
    stop when tree has N–1 edges

***Optimal* solution computed with *greedy* algorithm**

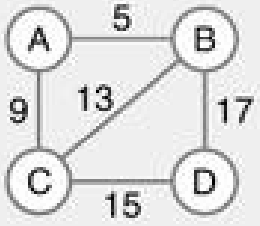
MST – Kruskal's Algorithm Example



MST – Kruskal's Algorithm

- When does adding (X,Y) to tree create cycle?
- Traversal approach
  - Traverse tree starting at X
  - If we can reach Y, adding (X,Y) would create cycle
- Connected subgraph approach
  - Maintain set of nodes for each connected subgraph
  - Initialize one connected subgraph for each node
  - If X, Y in same set, adding (X,Y) would create cycle
  - Otherwise
    - We can add edge (X,Y) to spanning tree
    - Merge sets containing X, Y (single subgraph)

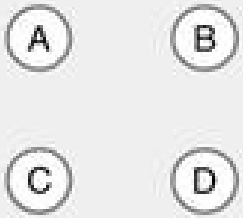
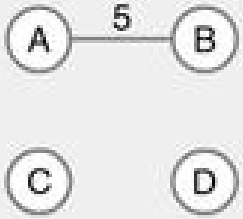
MST – Connected Subgraph Example



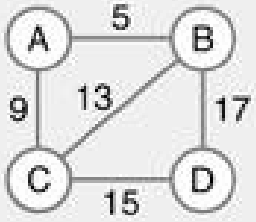
Original graph

Ordered set of edges

<A, B> 5
<A, C> 9
<B, C> 13
<C, D> 15
<B, D> 17

	MST	Sets	Edge being considered for addition	
1.		{A} {B} {C} {D}	<A, B>	Include, since it connects two nodes in distinct sets
2.		{A, B} {C} {D}	<A, C>	Include, since it connects two nodes in distinct sets

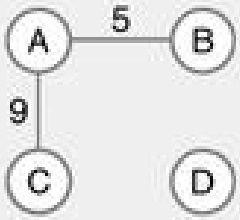
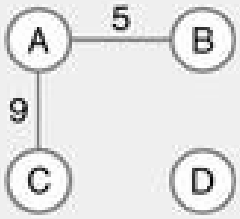
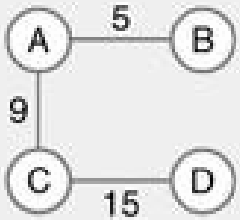
MST – Connected Subgraph Example



Original graph

Ordered set of edges

<A, B>	5
<A, C>	9
<B, C>	13
<C, D>	15
<B, D>	17

	MST	Sets	Edge being considered for addition	
3.		{A, B, C} {D}	<B, C>	Reject, since it connects nodes in the same set and would create a cycle
4.		{A, B, C} {D}	<C, D>	Include, since it connects two nodes in distinct sets
		{A, B, C, D}	Finished	

Union find algorithm/data structure

- Algorithm and data structure that allows you to ask this question.
- Start with n nodes, each in different subgraphs
- Two operations:
  - Are nodes x and y in the same subgraph?
  - Merge the subgraphs containing x and y

How fast is it?

- Ackermann's function
- int A(x,y) {
  if (x == 0) return y+1;
  if (y == 0) return A(x-1, 1);
  return A(x-1, A(x, y-1));
- A(2,2) = 7
- A(3,3) = 61
- A(4,2) = $2^{65536} - 3$
- A(4,3) = $2^{2^{65536}} - 3$
- A(4,4) = $2^{2^{2^{65536}}} - 3$

Inverse Ackermann's function

- $\alpha(n)$ is the inverse Ackermann's function
- $\alpha(n)$ = the smallest k s.t. A(k,k) >= n
- α(number of atoms in universe) = 4

- A sequence of n operations on a union find data structure requires $O(n\ \alpha(n))$ time