Module 9
Graph Algorithms

S. Lakshmivarahan
School of Computer Science
University of Oklahoma
USA-73019
Varahan@ou.edu

Graph Algorithms

- In this first course on Algorithm Analysis, we cover a small subset of graph problems and their algorithm

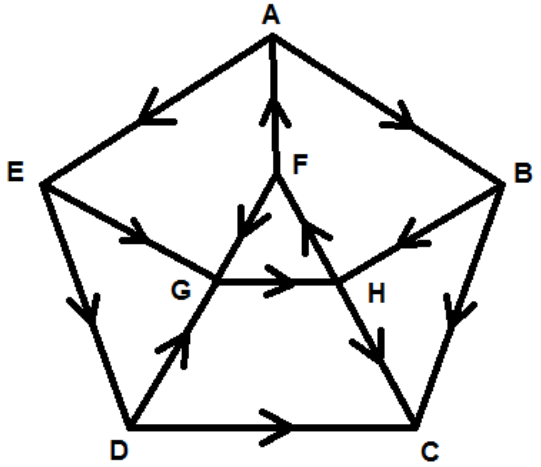
1. Graph Traversal Problem
   - Depth first search
   - Breadth first search
2. Minimal Cost Graph design problem
   - Minimum spanning tree- a greedy approach

Graph Algorithms

3. Path problems
   - Single pair shortest path- Dynamic programming
   - All pair shortest path- Dynamic programming
4. Transitive Closure problem

Graph Traversal Algorithms: DFS

- **Depth-first search (DFS) algorithm**:
- Consider a directed graph G = (V, E) as shown below



- Let N ∈ V be a starting vertex
- Let S be a stack, initially empty
- Let top be the pointer to the top of S

Graph Traversal Algorithms: DFS

- DFS Algorithm

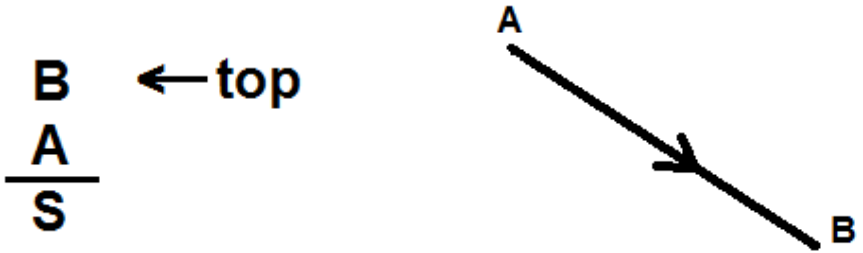
1. Visit, mark, and stack N
2. While S is non-empty, do
3. While there is an unmarked vertex W adjacent to top, do
4. Visit, mark, and stack W
5. Pop s

Graph Traversal Algorithms: DFS

- Setting N ← A, we now illustrate the mechanics of this procedure

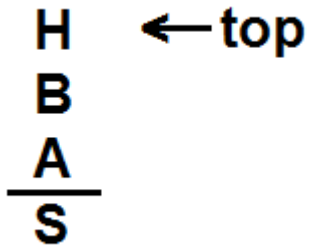
1. Visit, mark, stack A

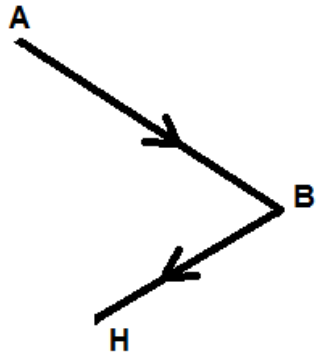
2. Visit, mark, stack B



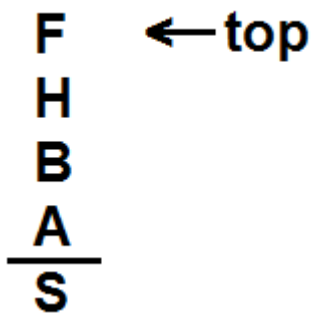
Graph Traversal Algorithms: DFS

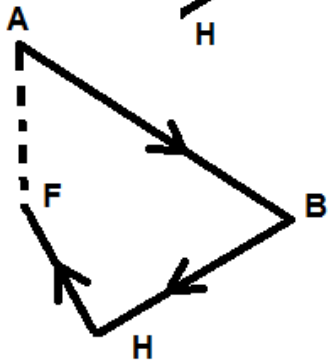
3. Visit, mark, stack H





4. Visit, mark, stack F





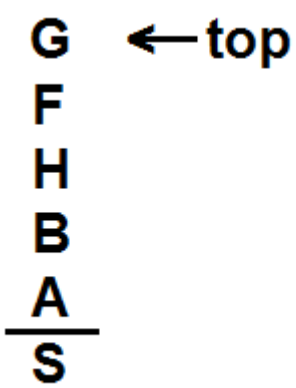
Graph Traversal Algorithms: DFS

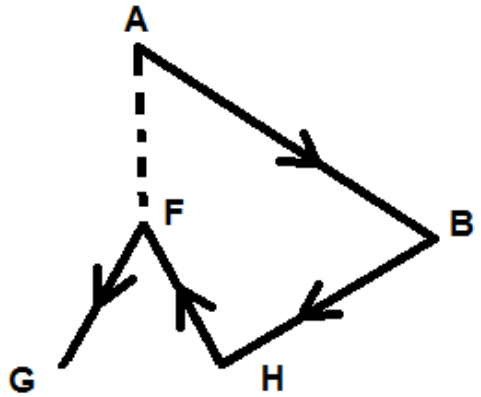
5. Neighbor of F is A which has already been marked. This is a dead end for traversal. Since we can visit G, we now push G onto the stack S and backtrack to H to see if there is a neighbor of H that has not been visited.

- Note: Before proceeding, notice that, if during the traversal we come across a node already visited, then we have discovered a cycle in G. Thus, we have formed a cycle ABHFA and we may register this fact separately.

Graph Traversal Algorithms: DFS

6. Visit, mark, stack G



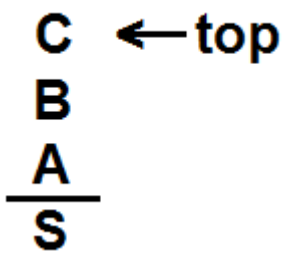


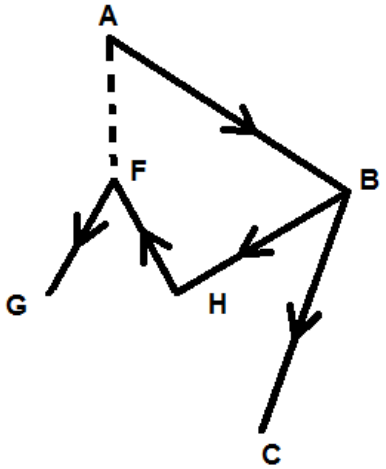
7. Again, we cannot get out of G since the rest of the edges incident on G are directed towards G.

Graph Traversal Algorithms: DFS

8. Pop G and let F be the new top of S. Again, we can go no further from F. Pop F and let H be the top. Since we can go further from H, pop H again, making B the new top.

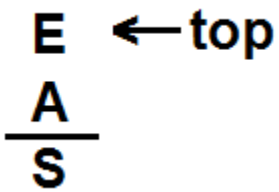
9. Now we can visit, mark, and stack C, which makes it the top

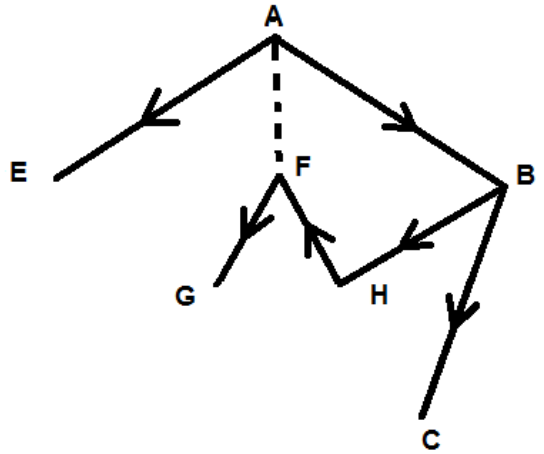




Graph Traversal Algorithms: DFS

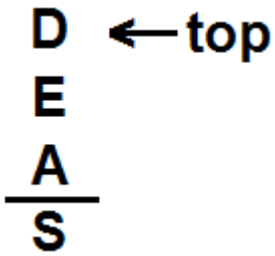
10. Now we can go no further from C. Hence, pop C and B is the top. Since we can go further, pop B and A is the top
11. Now visit E, mark and stack it

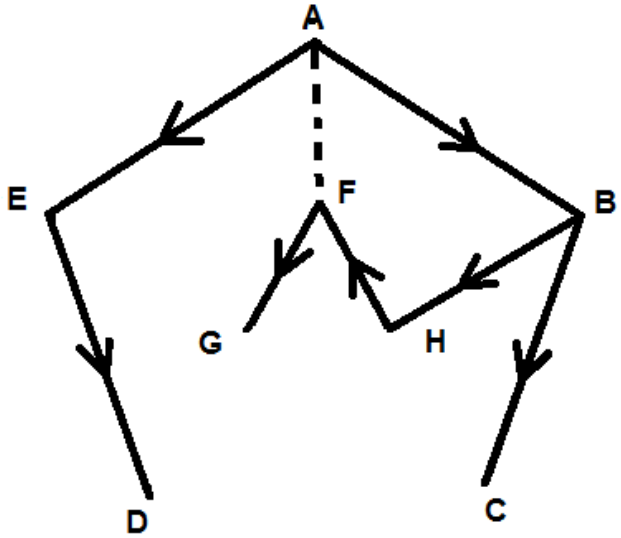




Graph Traversal Algorithms: DFS

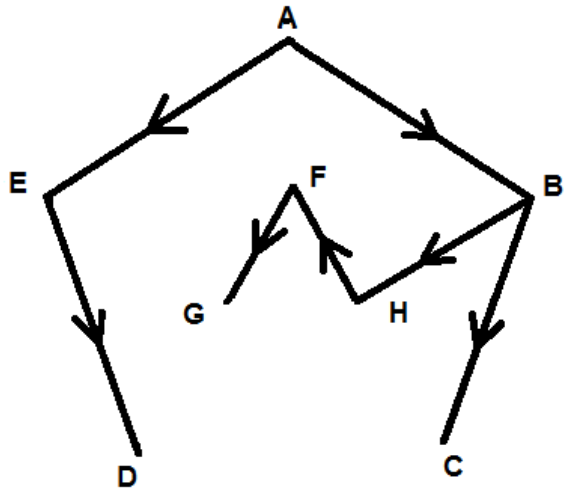
12. From E, we can go to D, so mark, visit, and stack D





Graph Traversal Algorithms: DFS

13. We can go no further, so pop D, pop E, and pop A, and the traversal ends
14. In the end we get a depth first search spanning tree of G given by



Graph Traversal Algorithms: DFS

- Notes: If we start at the node C which is a sink, the DFS gives a spanning forest which is a collection of trees
- The complexity of this process is O(n+m)
- DFS spanning tree is not unique

Homework

1. Compute the complexity
2. Starting from any other node, redo the algorithm

Graph Traversal Algorithms: BFS

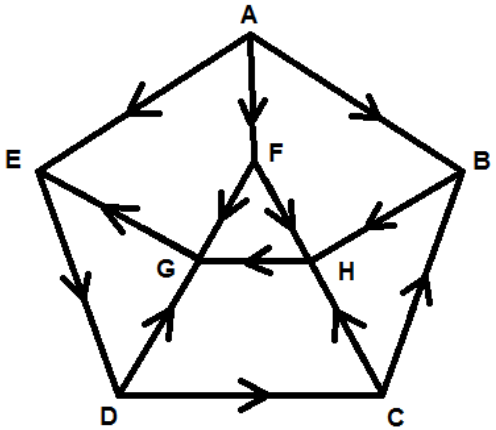
- **Breadth-first search (BFS) algorithm**:
- Let $U \in V$
- Let Q be a queue and $X \leftarrow Q$ denote the removal of the front item in the queue and calling it X

Graph Traversal Algorithms: BFS

1. Visit, mark U, and insert U in Q
2. While Q is nonempty, do
3. X ← Q
4. For each unmarked vertex W adjacent to X, do
5. Visit, mark W
6. Insert W in Q

Homework

3. Apply this algorithm on the graph:



4. Compute the complexity
5. Drop the directions and consider an undirected graph. Do the BFS. Verify that BFS finds the shortest paths from the chosen vertex to all the other nodes

Minimum Spanning Tree

- Suppose your university has n buildings and would like to connect these buildings in a network with the constraint that the cost of connection must be minimum
- That is, we are looking for a cheapest possible network that would connect all the buildings
- Here, the cost is measured by the total length of wire needed

Minimum Spanning Tree

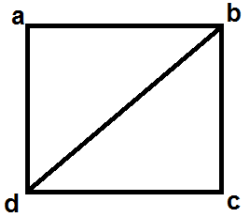
- To start with, the university first employs a survey team to assess how the buildings can be connected and to measure the length of the cable needed for each of the possible ways to interconnect the buildings
- This process gives us a weighted undirected graph G = (V, E, W) where $|V| = n, \quad n - 1 \leq |E| \leq \frac{n(n-1)}{2}$, and W is the collection of the weights of all the edges in E.

Minimum Spanning Tree

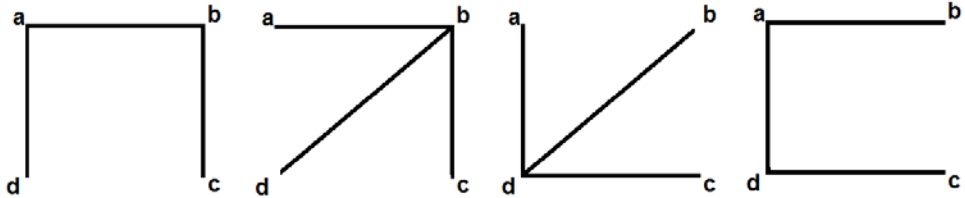
- Our goal is to find a subgraph T = (V, E', W') which is a tree such that |E'| = n-1 and the sum of the weights of the edges in E' is a minimum
- Since this tree T contains all the vertices of G, it is called a spanning tree and since it contains only those edges whose weights sum to a minimum, such a tree is called a minimum spanning tree (MST)

Minimum Spanning Tree

- Given a graph, the spanning tree is not unique
- Let G be the following graph:



- Here are some of the spanning trees of G:



- Homework: Find the set of all spanning trees of the graph above

Minimum Spanning Tree

- Also, note that this is an optimization problem where our goal is to design an optimal (minimal cost) network
- It turns out that this optimal design problem can be solved by using the simple greedy strategy
- Recall that we have used this greedy strategy in the design of Huffman Code for data compression

Minimum Spanning Tree

- There are two well known greedy based algorithms for solving the problem- Kruskel's and Prim's algorithms
- We now describe these two algorithms

Prim's Algorithm

- Let G=(V, E, W) be the given connected weighted graph
- This algorithm starts from an arbitrary vertex and branches out from the part of the spanning tree under development, by adding a new vertex and an associated edge

Prim's Algorithm

- This algorithm divides the nodes into three mutually exclusive subsets

a) Tree Vertices: Those vertices that are part of the growing tree

b) Fringe Vertices: Those vertices that are adjacent to some vertex in the tree

c) Other Vertices: Those vertices that are not in the tree or the fringe

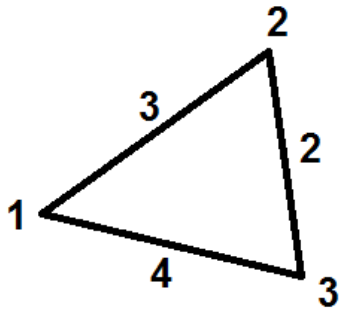
Prim's Algorithm

- Algorithm:

1. Select an arbitrary vertex u ∈ V as the start vertex. u is then in the tree
2. While there are fringe vertices, do
3. Select an edge of minimum weight between the tree and a fringe vertex
4. Add the selected edge and the associated fringe vertex to the tree

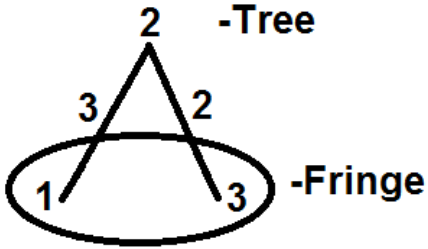
Prim's Algorithm: Example

- Let the graph be



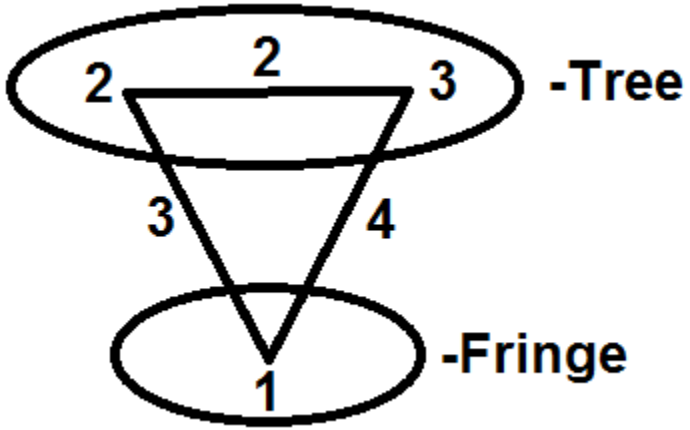
2 – Starting Vertex

- Start at u = 2. Then 1 and 3 are fringe vertices.



Prim's Algorithm: Example

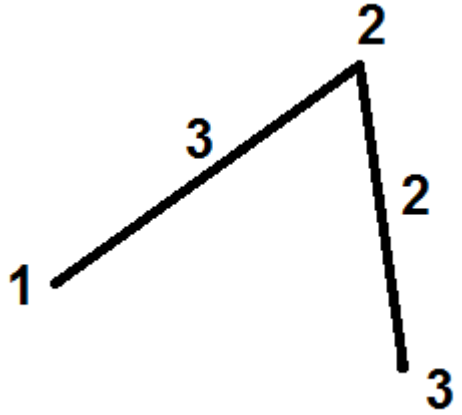
- Of the two edges, the edge (2,3) with weight 2 is the minimum, and so add the edge (2,3) and node 3 to the tree to get:



- Here, {2,3} are in the tree being developed
- {1} is a fringe vertex

Prim's Algorithm: Example

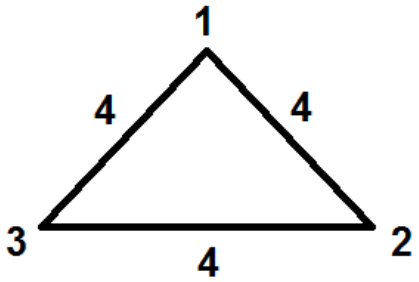
- Since the edge (2,1) has the minimum weight, it becomes part of the tree



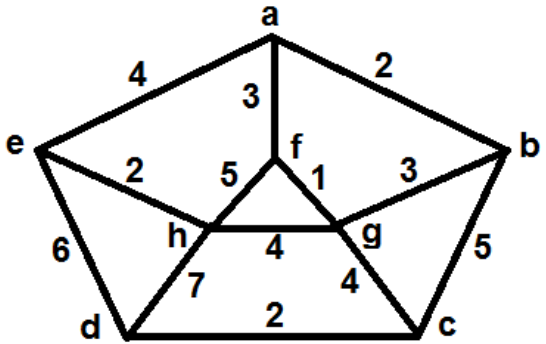
- Here, the MST is unique and the cost is 5

Homework

6. Find the set of all MSTs for the following:



7. Find a MST for the following graph:



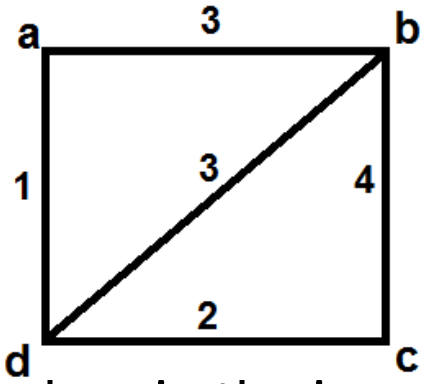
Kruskal's Algorithm

- Let G be a connected, weighted graph, G = (V, E, W)

1. Arrange the edges in increasing order of their weights
2. Start with an empty graph
3. Add edges in the increasing order of weights to evolve the spanning tree, taking care to see that no loop is created

Kruskal's Algorithm: Example

- Consider

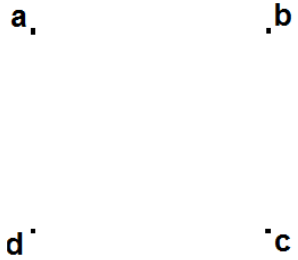


1. Order the edges in the increasing order of weights

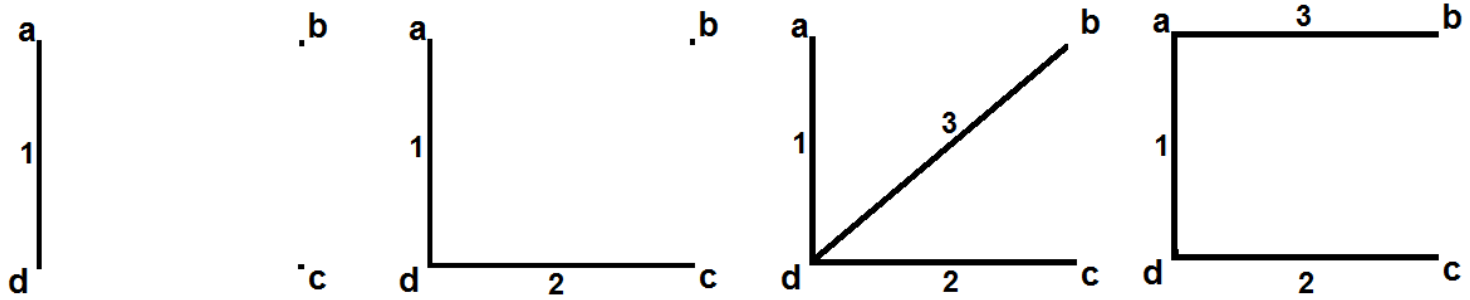
(c,d)	(d,c)	(d,b)	(a,b)	(b,c)
1	2	3	3	4

Kruskal's Algorithm

2. Start with an empty graph:



3. Add edges starting from the minimum weight edge until we obtain a tree:



Homework

8. Compute the complexity of these two algorithms for finding MST

Path Problems

- Let G=(V, E, W) be a weighted graph where all the weights are assumed to be positive
- Our goal is to develop algorithms for two types of path problems

a) Single pair shortest path problem

b) All pairs shortest path problem

Path Problems

a) Single Pair shortest path problems

- Let x and y be two distinct nodes in the graph. Find the shortest path from x to y and its length.

b) All Pairs shortest path problem

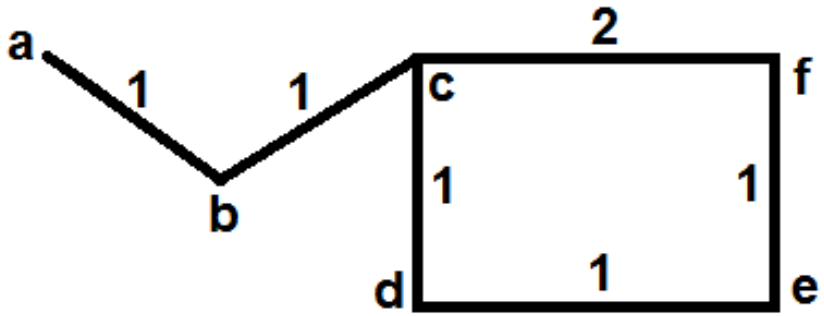
- Find shortest path between every pair (x, y) of distinct nodes in the graph.

Path Problems

- These are discrete optimization problems. It turns out that we cannot solve these by the simple “greedy strategy” as the following counter example illustrates

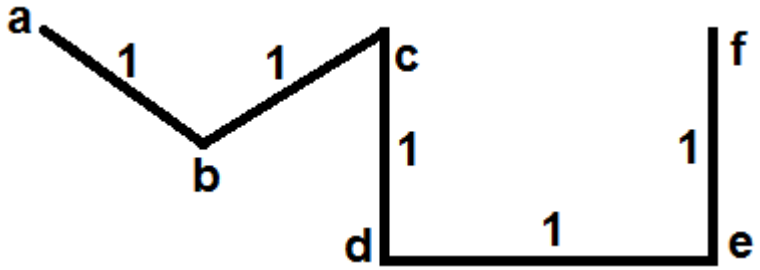
Path Problems: Counter Example

- Consider the following weighted graph:



Path Problems: Counter Example

- By applying the MST algorithm to this graph, we obtain



as its MST. The path from a to f in this MST has length 5, which is not the shortest path, since the direct path (a,b,c,f) in the original graph has length 4.

Path Problems

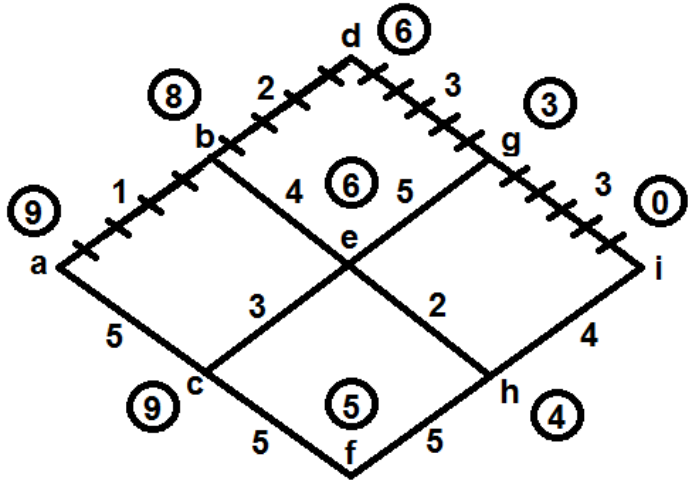
- Stated in other words, we cannot find the shortest paths using the MST which is based on the greedy strategy
- In fact, we need the full power of the Dynamic Programming (DP) to find the shortest paths

Path Problems

- There are essentially two ways of looking at the application of DP. One is to start at the destination y and move towards the source x, or start from the source x and move towards the destination y

DP: Destination to Source

- Consider the following graph:



- Let a be the source and i be the destination

DP: Destination to Source

- We define a weight to each node which denotes the length of the shortest path from that node to the destination node
- Thus, the weight of node i is zero

DP: Destination to Source

- Let us compute the weight of node g. There are two path from g to i given by

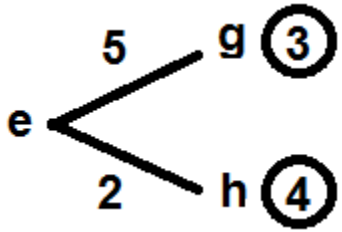
$$g \xrightarrow{3} i \qquad \rightarrow Distance = 3$$

$$g \xrightarrow{5} e \xrightarrow{2} h \xrightarrow{4} i \qquad \rightarrow Distance = 11$$

- Since the direct path has minimum distance 3, the weight of node g = W(g) = 3. This is denoted within a circle besides g.
- Similarly, W(h) = 4

DP: Destination to Source

- Consider the node e:



- The path from e to g + W(g) = 5 + 3 = 8

  The path from e to h + W(h) = 2 + 4 = 6

- Since 6 is the minimum, W(e) = 6
- Continuing this process, we obtain the weights of all nodes

DP: Destination to Source

- Since W(a) = 9, d(a,i) = 9 is the shortest path.
- We need to recover the path of this length
- Recall that, if a and b are two nodes along the shortest path, then

$$w(a) - w(b) = \text{weight of the edge (a, b)}$$

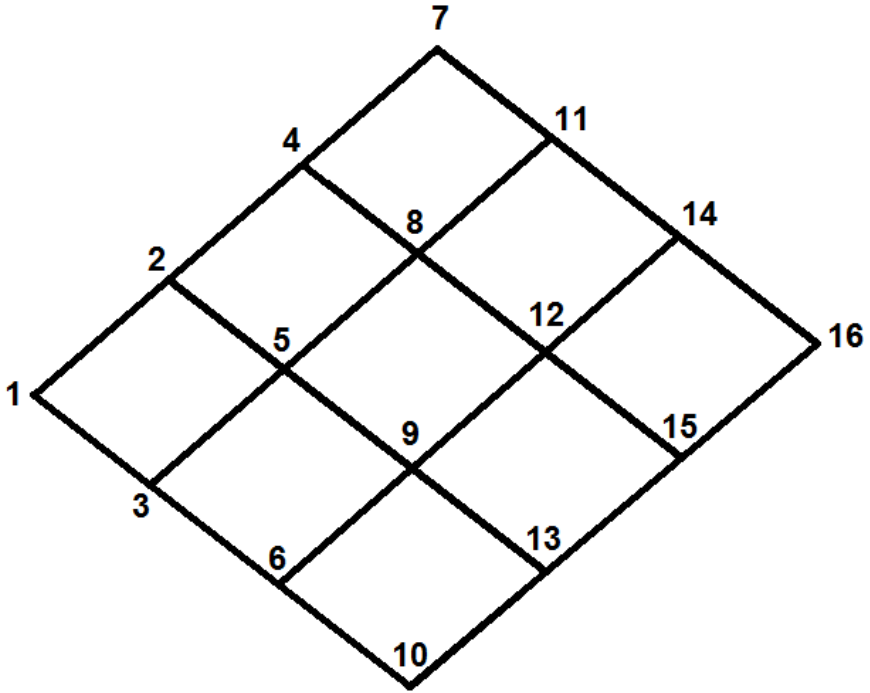
DP: Destination to Source

- Following this idea, the actual shortest path is given by (a,b,d,g,i), shown as the hatched edges from a to i

- Note: In the above algorithm, we end up finding the shortest path from every other node to the destination node.

DP: Destination to Source

- Homework: Consider the following graph. Assign weights to edges at random in the range 1 to 10. Find the shortest path from 1 to 16 and its length



Dijkstra's Algorithm: Source to Destination

- Let G=(V, E, W) be the weighted graph, the vertices are numbered 1, 2, …, n, and 1 is the source and n is the destination
- This algorithm divides the nodes into classes
- Class A nodes are those whose minimum distance from 1 are known. Class B remain to be examined

Dijkstra's Algorithm: Source to Destination

- This algorithm assigns a weight to each node
- The weight of nodes in class A denote the length of the shortest path from node 1 to each node in this class
- The weights of nodes in class B denote the length of the shortest path found thus far
- The algorithm adjusts weights of the nodes in class B. In this process, the weights of class B nodes cannot increase

Dijkstra's Algorithm: Source to Destination

- The node with the least weight from class B migrates from class B to A
- The most recently migrating node is called the pivot node
- The pivot node is used to change the weights of class B nodes
- The algorithm terminates when the destination node n becomes a member of class A

Dijkstra's Algorithm: Source to Destination

- Algorithm:

1. Place the source node 1 in class A and all the others in class B
2. Set the weight of node 1 to zero and that of all others to infinity

   i.e. $w(1) = 0$, $w(i) = \infty$ for $i \neq 1$
3. Do the following until the destination n becomes a member of class A

Dijkstra's Algorithm: Source to Destination

a) The most recent migrant node in class A is called the pivot node

b) Adjust the weights of the class B nodes as follows

i. If a class B node is not connected to the pivot node, its weight does not change

ii. If a class B node is connected to the pivot node, replace its current value by the minimum of (a) its current value and (b) the value of the pivot node and the node in question

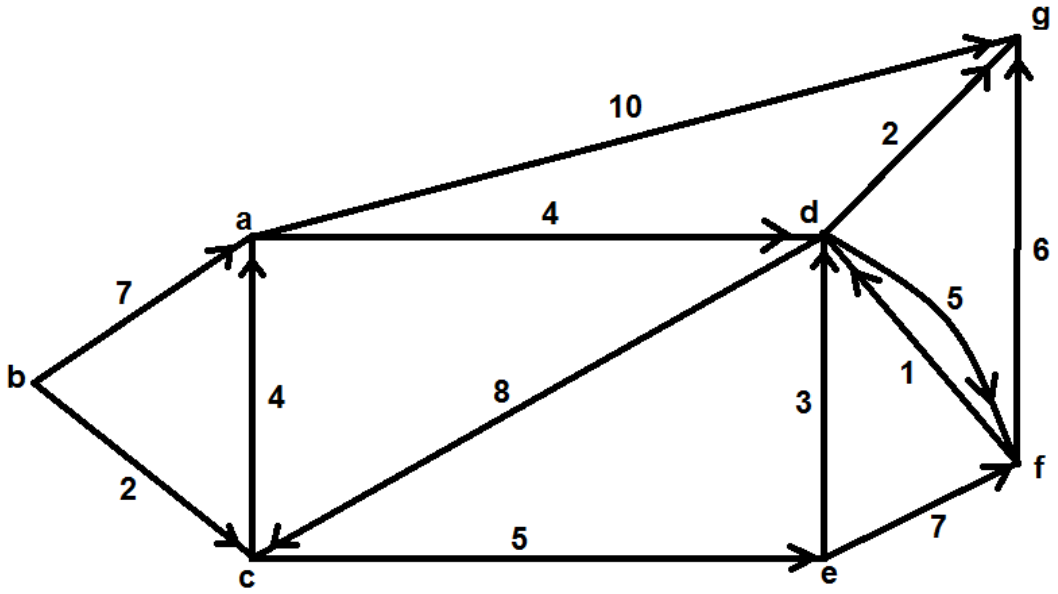
iii. Select a class B node with minimum weight and place it in class A

Homework

9. Verify that this algorithm requires at most $\frac{n(n-1)}{2}$ additions and $\frac{n(n-1)}{2}$ comparisons

Dijkstra's Algorithm: Example

- We now illustrate with an example



Dijkstra's Algorithm: Example

a	b	c	d	e	f	g	
∞	$[0]^{\surd}$	∞	∞	∞	∞	∞	Starting vertex b is labeled 0
7	[0]	2	∞	∞	∞	∞	Successors of b get relabeled
7	[0]	$[2]^{\surd}$	∞	∞	∞	∞	Smallest label (c) becomes final
6	[0]	[2]	∞	7	∞	∞	Successors of c get relabeled
$[6]^{\surd}$	[0]	[2]	∞	7	∞	∞	Smallest label (a) becomes final
[6]	[0]	[2]	10	7	∞	16	Successors of a get relabeled

[x] – Class A

$x^{\surd}$ – Pivot

Dijkstra's Algorithm: Example

a	b	c	d	e	f	g	
[6]	[0]	[2]	10	$[7]^{\surd}$	∞	16	Smallest label (e) becomes final
[6]	[0]	[2]	10	[7]	14	16	Successors of e get relabeled
[6]	[0]	[2]	$[10]^{\surd}$	[7]	14	16	Smallest label (d) becomes final
[6]	[0]	[2]	[10]	[7]	14	12	Successors of d get relabeled
[6]	[0]	[2]	[10]	[7]	14	$[12]^{\surd}$	Vertex g gets its final label and we are done

[x] – Class A

$x^{\surd}$ – Pivot

All pair shortest path problem

- A telephone company must be capable of connecting a call from any phone to any other phone in the shortest possible path
- A trucking company (FedEx, UPS, etc.) must have the capability to deliver a piece of mail from any address to any other address
- These call for knowing shortest paths from every node to every other node

All pair shortest path problem

- In principle, we could apply the Dijkstra's algorithm for every pair of nodes, which takes $O(n^2)$ operations for every pair
- Since there are $\frac{n(n-1)}{2}$ pairs of distinct nodes, this would require potentially $O(n^4)$ operations
- In the following, we describe a Dynamic Programming based algorithm due to Floyd-Warshall that computes the all pair shortest paths in $O(n^3)$ time

Floyd-Warshall Algorithm

- Let G= (V, E, W) be a weighted graph with positive weight
- Let V = {1, 2, … , n} and let {1, 2, … , k} be the subset of the first k vertices
- A pair is specified by a sequence of connected nodes as

$$P = (v_1, v_2, \dots, v_{L-1}, v_L)$$

- In here, v_1 is the start and v_L is the end vertex. The set $\{v_2, \dots, v_{L-1}\}$ are the intermediate edges

Floyd-Warshall Algorithm

- Since the weights are positive, the shortest path is always simple
- Floyd-Warshall algorithm discovers the shortest path from i to j by successively considering the paths that pass through the set {1,2,…,k} as the intermediate nodes as k varies from 1 through n

Floyd-Warshall Algorithm

- Let $d_{ij}^{(k)}$ denote the length of the shrotest path from i to j that pass through the set {1,2,…,k} as intermediate nodes
- Let $d_{ij}^{(k-1)}$ be given for all $1 \leq i,j \leq n$. Then the shortest path from i to j does not pass through the node k. In this case,

$$d_{ij}^{(k)} = d_{ij}^{(k-1)}$$

Floyd-Warshall Algorithm

- If, on the other hand, k is an intermediate node along the shortest path form i to j, then

$$d_{ij}^{(k)} = d_{ik}^{(k-1)} + d_{kj}^{(k-1)}$$

- Consequently, we get the recurrence

$$d_{ij}^{(k)} = \min\{d_{ij}^{(k)},\ d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\}$$

- Clearly, $d_{ij}^{(0)} = w_{ij}$ if $(i, j) \in E$

Floyd-Warshall Algorithm

- Here is the Floyd-Warshall algorithm that takes $O(n^3)$ time

$D^{(0)} = W$

For k = 1 to n

  For i = 1 to n

    For j = 1 to n

$$d_{ij}^{(k)} = \min\{d_{ij}^{(k)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\}$$

Homework

10. Much like other shortest path algorithms, we have now found the lengths of the shortest paths from all i to all j. The problem of recovering the actual path remains. Find the shortest path for each pair (i, j)