The material in this presentation belongs to St. Francis Institute of Technology and is solely for educational purposes. Distribution and modifications of the content is prohibited.

Module 6:

Applications of Data Structures

Subject In-charge

Ms.Pratibha Rane

Assistant Professor

Class : III SEM SEIT- B

The material in this presentation belongs to St. Francis Institute of Technology and is solely for educational purposes. Distribution and modifications of the content is prohibited.

Module 6: Applications of Data Structures

.

Applications of Linked Lists: Addition of 2 Polynomials and Multiplication of 2 polynomials.
Applications of Stacks: Reversal of a String, Checking validity of an expression containing nested parenthesis, Function calls, Polish Notation: Introduction to infix, prefix and postfix expressions and their evaluation and conversions.
Application of Queues: Scheduling, Round Robin Scheduling
Applications of Trees: Huffman Tree and Heap Sort. Applications of Graphs: Dijkstra's Algorithm, Minimum Spanning Tree: Prim's Algorithm, Kruskal's Algorithm.

Self-learning Topics: Implementation of applications for Stack, Queues, Linked List, Trees and Graph

Reema Thareja; Data Structures using C; Oxford

The material in this presentation belongs to St. Francis Institute of Technology and is solely for educational purposes. Distribution and modifications of the content is prohibited.

Applications of Linked Lists

1.Polynomial manipulations : polynomial $P(x) = 7x^2 + 15x^3 - 2x^2 + 9$

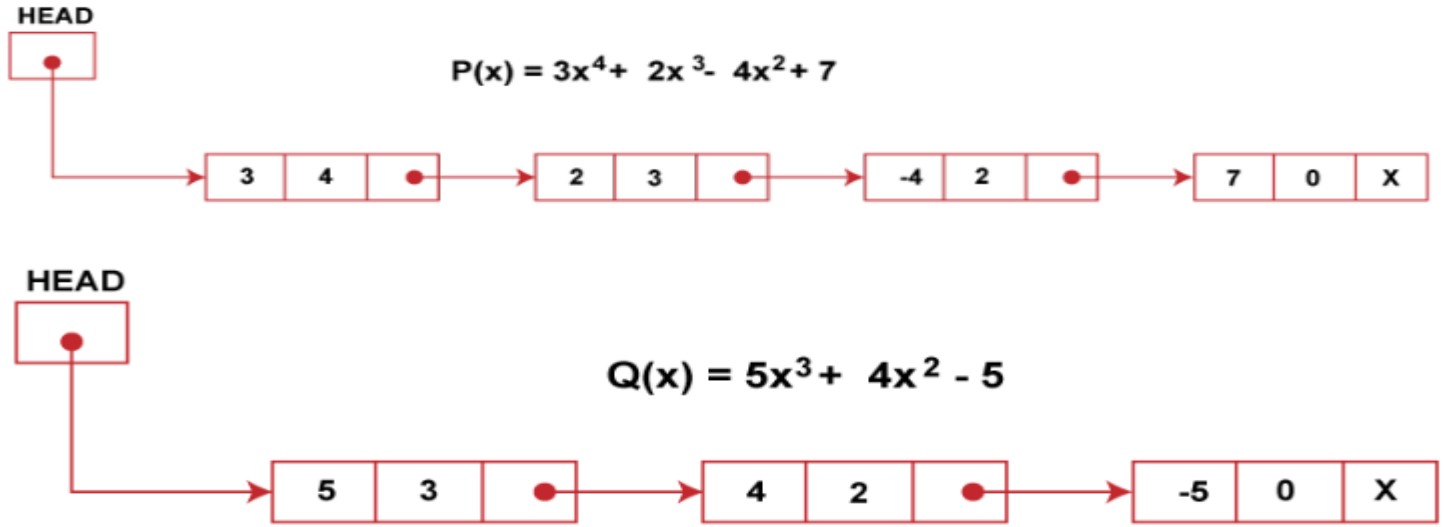
Node representing a term of a polynomial

The material in this presentation belongs to St. Francis Institute of Technology and is solely for educational purposes. Distribution and modifications of the content is prohibited.

Applications of Linked Lists

1.Addition of Polynomials: If the two exponents are equal, the coefficients are added to create a new coefficient.

$P(x) = 3x^4 + 2x^3 - 4x^2 + 7$
$Q(x) = 5x^3 + 4x^2 - 5$



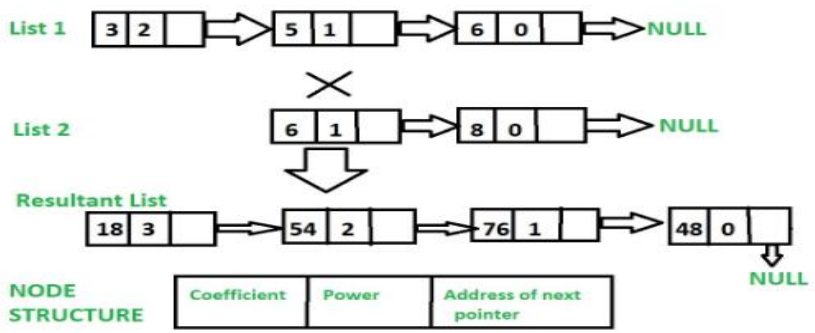
Linked list for resulting polynomial

The material in this presentation belongs to St. Francis Institute of Technology and is solely for educational purposes. Distribution and modifications of the content is prohibited.

Applications of Linked Lists:Multiplication of Polynomials:

- **To multiply two polynomials, we can first multiply each term of one polynomial to the other polynomial.**
- **Suppose the two polynomials have n and m terms. This process will create a polynomial with n*m terms.**
- Poly1: 3x^3 + 6x^1 – 9
  Poly2: 9x^3 – 8x^2 + 7x^1 + 2

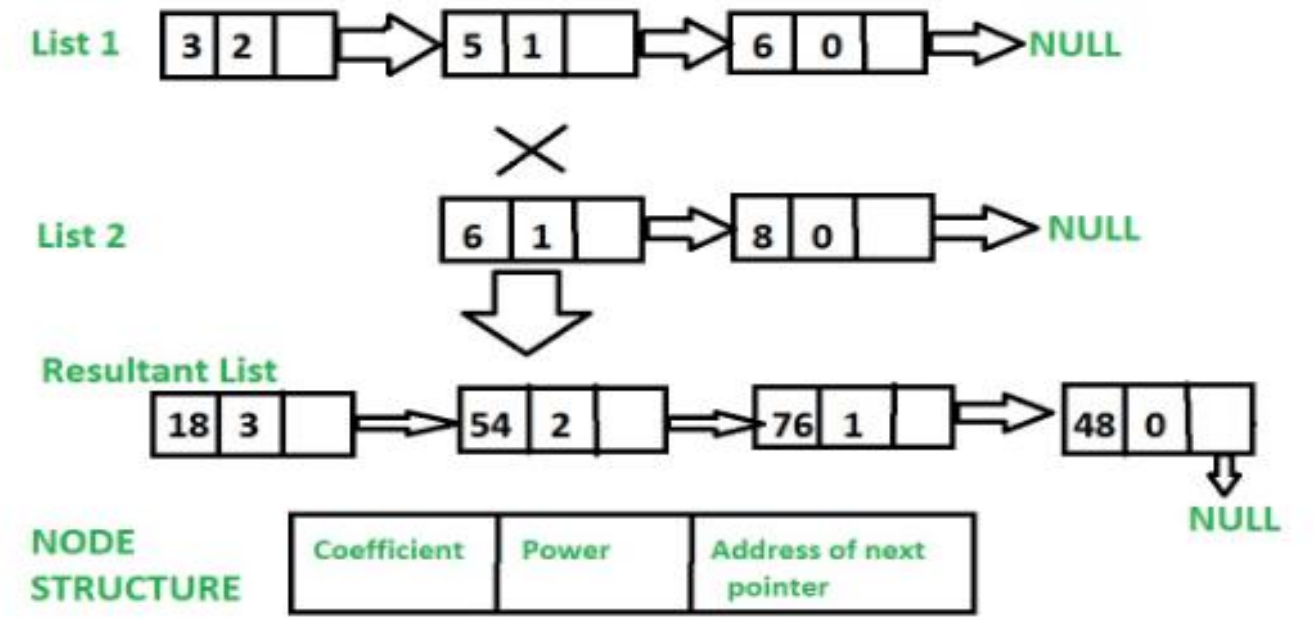
Step1: We have to multiply all the terms of **Poly1** one by one with every term of **Poly2**, so first, we will multiply 3×3 with every other term in **Poly2**.(result: 27×6 – 24×5 + 21×4 + 6×3)
Step2: Now we take 6×1 and multiply it with every other term in **Poly2**.(result: 27×6 – 24×5 + 21×4 + 6×3 + 54×4 – 48×3 + 42×2 + 12×1)
Step3: Now we take -9 and multiply it with every other term in **Poly2**.(result: 27×6 – 24×5 + 21×4 + 6×3 + 54×4 – 48×3 + 42×2 + 12×1 – 81×3 + 72×2 – 63×1 – 18)
We will remove all the duplicates, i.e., add the value of nodes with the same powers.
So the final result: 27×6 – 24×5 + 75×4 – 123×3 114×2 – 51×1 – 18



The material in this presentation belongs to St. Francis Institute of Technology and is solely for educational purposes. Distribution and modifications of the content is prohibited.

Applications of Linked Lists:Multiplication of Polynomials:

- **To multiply two polynomials, we can first multiply each term of one polynomial to the other polynomial.**

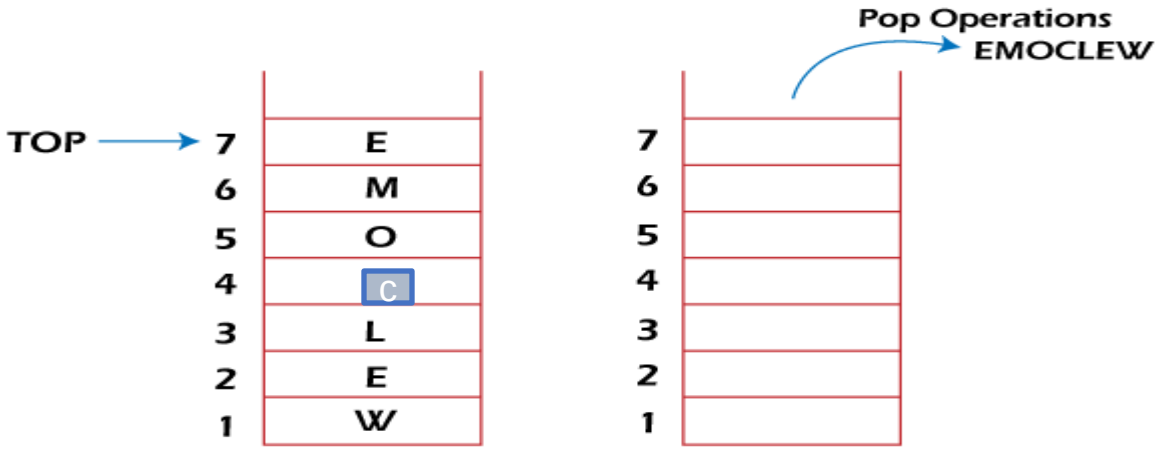


The material in this presentation belongs to St. Francis Institute of Technology and is solely for educational purposes. Distribution and modifications of the content is prohibited.

Application of Stack:

1. Reverse a Data:

- Stack can be used to reverse the characters of a string. T
- his can be achieved by simply pushing one by one each character onto the Stack, which later can be popped from the Stack one by one.
- Because of the **last in first out** property of the Stack, the first character of the Stack is on the bottom of the Stack and the last character of the String is on the Top of the Stack and after performing the pop operation in the Stack, the Stack returns the String in Reverse order.

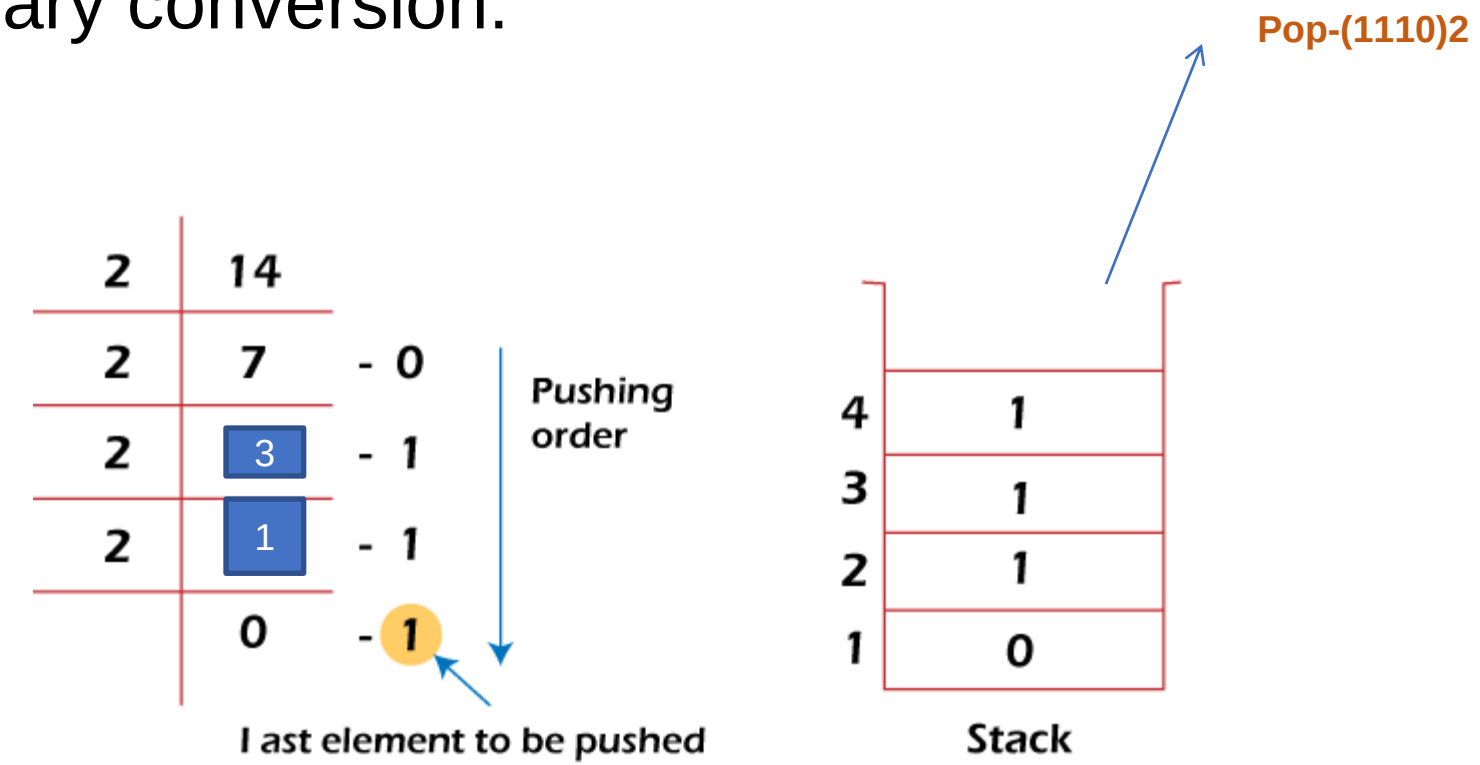


The algorithm contains the following steps:
Create an empty stack.
1.Push all the characters of the string into the stack.
2.One by one, pop each character from the stack until the stack is empty.
3.Place the popped characters back into the string from the zeroth position.

The material in this presentation belongs to St. Francis Institute of Technology and is solely for educational purposes. Distribution and modifications of the content is prohibited.

Application of Stack:

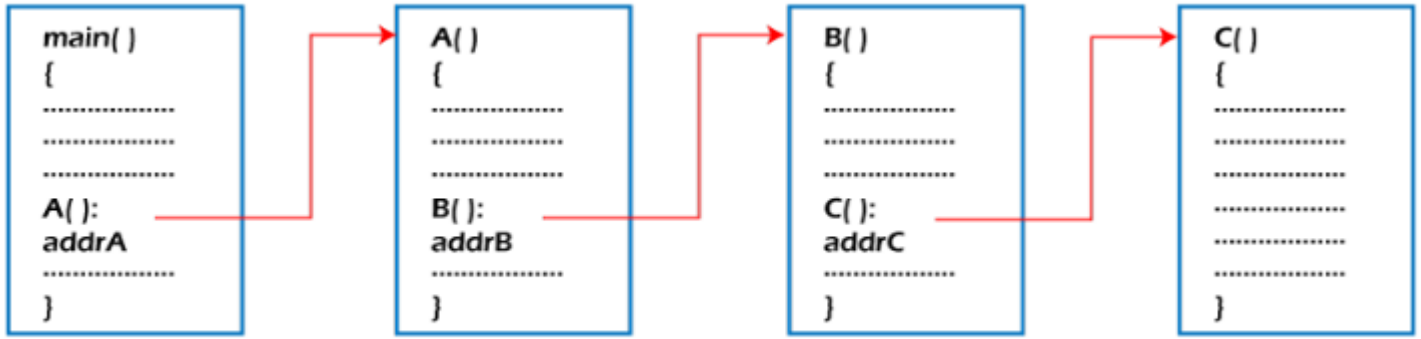
1. Decimal to binary conversion:



The material in this presentation belongs to St. Francis Institute of Technology and is solely for educational purposes. Distribution and modifications of the content is prohibited.

Application of Stack:

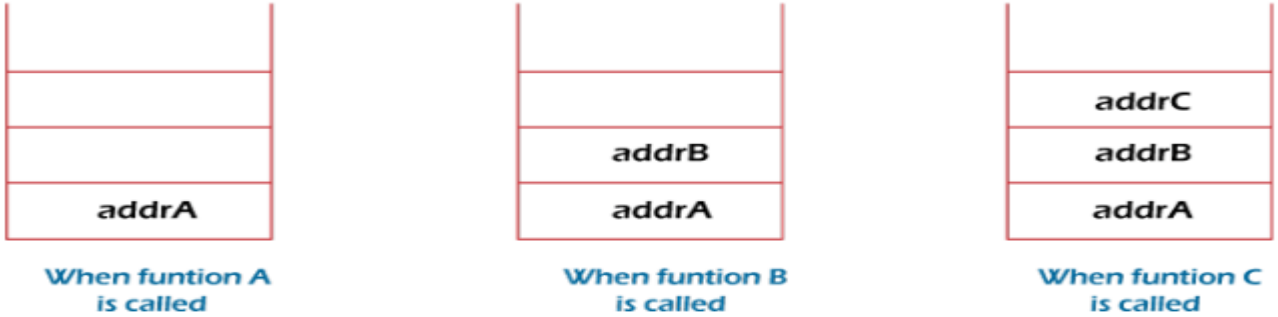
3.Processing Function Calls:



Function call

Stack

Consider addrA, addrB, addrC be the addresses of the statements to which control is returned after completing the function A, B, and C, respectively.



The material in this presentation belongs to St. Francis Institute of Technology and is solely for educational purposes. Distribution and modifications of the content is prohibited.

Application of Stack:

4.Evaluation of Arithmetic Expressions:

Operators	Associativity	Precedence
^ exponentiation	Right to left	Highest followed by *Multiplication and /division
*Multiplication, /division	Left to right	Highest followed by + addition and - subtraction
+ addition, - subtraction	Left to right	lowest

The material in this presentation belongs to St. Francis Institute of Technology and is solely for educational purposes. Distribution and modifications of the content is prohibited.

Application of Stack:

4.Evaluation of Arithmetic Expressions:

There are three notations to represent an arithmetic expression:(Polish Notations)

1.Infix Notation
2.Prefix Notation
3.Postfix Notation

Infix Notation	Prefix Notation	Postfix Notation
A * B	* A B	AB*
(A+B)/C	/+ ABC	AB+C/
(A*B) + (D-C)	+*AB - DC	AB*DC-+

The material in this presentation belongs to St. Francis Institute of Technology and is solely for educational purposes. Distribution and modifications of the content is prohibited.

Application of Stack: .Converting an infix expression into a postfix expression:

```
Step 1: Add ")" to the end of the infix expression
Step 2: Push "(" on to the stack
Step 3: Repeat until each character in the infix notation is scanned
        IF a "(" is encountered, push it on the stack
        IF an operand (whether a digit or a character) is encountered, add it to the
        postfix expression.
        IF a ")" is encountered, then
          a. Repeatedly pop from stack and add it to the postfix expression until a
             "(" is encountered.
          b. Discard the "(". That is, remove the "(" from stack and do not
             add it to the postfix expression
        IF an operator O is encountered, then
          a. Repeatedly pop from stack and add each operator (popped from the stack) to the
             postfix expression which has the same precedence or a higher precedence than O
          b. Push the operator O to the stack
        [END OF IF]
Step 4: Repeatedly pop from the stack and add it to the postfix expression until the stack is empty
Step 5: EXIT
```

Figure 7.22 Algorithm to convert an infix notation to postfix notation

The material in this presentation belongs to St. Francis Institute of Technology and is solely for educational purposes. Distribution and modifications of the content is prohibited.

Application of Stack: Converting an infix expression into a postfix expression

Example: Convert an infix expression into a postfix expression:

A-(B/C + (D%E*F)/G)*H

Homework: **Convert an infix expression into a postfix expression:**(A*(B+(C/D))) Answer:ABCD/+*

Application of Stack:

Converting an infix expression into a postfix expression

Example: Convert an infix expression into a postfix expressior..

A-(B/C + (D%E*F)/G)*H

Sr. No	Infix Character Scanned	Stack	Postfix notation
1	A		A
2	-	-	A
3	(	- (	A
4	B	- (	AB
5	/	- (/	AB
6	C	- (/	ABC
7	+	- (+	ABC /
8	(	-(+(	ABC /
9	D	-(+(	ABC/D
10	%	-(+(%	ABC/D
11	E	-(+(%	ABC/DE
12	*	-(+(*	ABC/DE%
13	F	-(+(*	ABC/DE%F
14	)	-(+	ABC/DE%F*
15	/	-(+/	ABC/DE%F*
16	G	-(+/	ABC/DE%F*G
17	)	-	ABC/DE%F*G/+
18	*	-*	ABC/DE%F*G/+
19	H	-*	ABC/DE%F*G/+H
20	End of Exression		ABC/DE%F*G/+H*-

The material in this presentation belongs to St. Francis Institute of Technology and is solely for educational purposes. Distribution and modifications of the content is prohibited.

Application of Stack:Evaluating Postfix Expression

```
Step 1: Add a ")" at the end of the
        postfix expression
Step 2: Scan every character of the
        postfix expression and repeat
        Steps 3 and 4 until ")"is encountered
Step 3: IF an operand is encountered,
        push it on the stack
        IF an operator O is encountered, then
        a. Pop the top two elements from the
           stack as A and B as A and B
        b. Evaluate B O A, where A is the
           topmost element and B
           is the element below A.
        c. Push the result of evaluation
           on the stack
        [END OF IF]
Step 4: SET RESULT equal to the topmost element
        of the stack
Step 5: EXIT
```

Figure 7.23 Algorithm to evaluate a postfix expression

Infix Expression is:9-((3*4)+8)/4

Postfix Expression :934*8+4/-

Table 7.1 Evaluation of a postfix expression

Character Scanned	Stack
9	9
3	9, 3
4	9, 3, 4
*	9, 12
8	9, 12, 8
+	9, 20
4	9, 20, 4
/	9, 5
-	4

The material in this presentation belongs to St. Francis Institute of Technology and is solely for educational purposes. Distribution and modifications of the content is prohibited.

Application of Stack: Converting an infix expression into prefix expression

Step 1: Reverse the infix string. Note that while reversing the string you must interchange left and right parentheses.
Step 2: Obtain the postfix expression of the infix expression obtained in Step 1.
Step 3: Reverse the postfix expression to get the prefix expression

(A-B/C)*(A/K-L)

↓

(L-K/A)*(C/B-A)

↓

LKA/-CB/A-*

↓

*-A/BC-/AKL

The material in this presentation belongs to St. Francis Institute of Technology and is solely for educational purposes. Distribution and modifications of the content is prohibited.

Application of Stack: Evaluating prefix expression

```
Step 1: Accept the prefix expression
Step 2: Repeat until all the characters
        in the prefix expression have
        been scanned
        (a) Scan the prefix expression
            from right, one character at a
            time.
        (b) If the scanned character is an
            operand, push it on the
            operand stack.
        (c) If the scanned character is an
            operator, then
              (i) Pop two values from the
                  operand stack
             (ii) Apply the operator on
                  the popped operands
            (iii) Push the result on the
                  operand stack
Step 3: END
```

Figure 7.26 Algorithm for evaluation of a prefix expression

prefix expression + - 2 7 * 8 / 4 12

Character scanned	Operand stack
12	12
4	12, 4
/	3
8	3, 8
*	24
7	24, 7
2	24, 7, 2
-	24, 5
+	29

The material in this presentation belongs to St. Francis Institute of Technology and is solely for educational purposes. Distribution and modifications of the content is prohibited.

Application of Stack: 5. Balanced Parentheses checker

- Given an expression containing only ‘(‘, ’)’, ’{‘, ‘}’, ‘[‘, ‘]’ , check whether the expression is balanced or not.
- An expression is balanced if each opening bracket is closed by the same type of closing bracket in the exact same order.
- The idea is to use the LIFO functionality of the stack. As a stack is LIFO data structure, we will remove every opening bracket (‘(‘, ‘{‘, ‘[‘), whenever we encounter the same type of closing bracket (‘)’, ‘}’, ‘]’).
  If for any opening bracket we don’t have a closing bracket, the opening bracket will remain in the stack forever.
  Similarly if we only have a closing bracket without a preceding opening bracket we will just push the closing bracket into the stack and it will remain there forever.
  So, at last after traversing the whole expression, if the stack is empty then it will mean that the given expression is balanced, otherwise if the stack contains elements that would mean the given expression was unbalanced.

The material in this presentation belongs to St. Francis Institute of Technology and is solely for educational purposes. Distribution and modifications of the content is prohibited.

Application of Stack:

Evaluation of Arithmetic Expressions:

Converting an infix expression into a postfix expression:

	Infix Expression	Stack	Postfix Expression
i)	A + B / C + D * (E - F) ^ G)	(	A
ii)	A + B / C + D * (E - F) ^ G)	+ (	A
iii)	A + B / C + D * (E - F) ^ G)	+ (	AB
iv)	A + B / C + D * (E - F) ^ G)	/ + (	ABC
v)	A + B / C + D * (E - F) ^ G)	/ + (	ABC/+
vi)	A + B / C + D * (E - F) ^ G)	+ (	ABC/+D
vii)	A + B / C + D * (E - F) ^ G)	+ (	ABC/+D
viii)	A + B / C + D * (E - F) ^ G)	* + (	ABC/+D
ix)	A + B / C + D * (E - F) ^ G)	* + (	ABC/+D
x)	A + B / C + D * (E - F) ^ G)	(* + (	ABC/+DE
xi)	A + B / C + D * (E - F) ^ G)	- * + (	ABC/+DE
xii)	A + B / C + D * (E - F) ^ G)	- * + (	ABC/+DEF
xiii)	A + B / C + D * (E - F) ^ G)	* + (	ABC/+DEF-
xiv)	A + B / C + D * (E - F) ^ G)	^ * + (	ABC/+DEF-
xv)	A + B / C + D * (E - F) ^ G)	^ * + (	ABC/+DEF-G
xvi)	A + B / C + D * (E - F) ^ G)		ABC/+DEF-G^*+

The material in this presentation belongs to St. Francis Institute of Technology and is solely for educational purposes. Distribution and modifications of the content is prohibited.

Application of Queue:

1. Task Scheduling

A queue is an ideal data structure for task scheduling, which lets you perform all the tasks in order. Initially, all the tasks are pushed to the end of the queue. We can pick the task at the front of the queue and execute it. Once you complete the task, it is removed from the queue, and you select the next task. This process will continue until the queue is empty.

2. Batch Processing

Similar to task scheduling, you can use a queue data structure when you need to perform a large number of tasks in batches. Queue helps to complete all the tasks in order without leaving a task unprocessed.

3. Resource Allocation

The queue also helps to perform resource allocation. If a process requests a resource, it is pushed to the end of the queue. Then the system selects the first request and allocates the requested resource. You can remove this request from the queue once the process finishes the use of the requested resource. After that, the system selects further requests from the queue until the queue is empty.

The material in this presentation belongs to St. Francis Institute of Technology and is solely for educational purposes. Distribution and modifications of the content is prohibited.

Application of Queue:

4. Event Handling

In event handling, a queue data structure stores all the events that have occurred but are yet to be processed by the system. If a new event occurs, it is pushed to the end of the queue, and the system then processes the event from the front of the queue and removes it once handled.

5. Message Buffering

Message buffer stores the messages in the order they arrive and processes them one by one. The queue is an ideal data structure because of its FIFO(First In, First Out) property.

6. Traffic Management

Circular queues are used to manage the traffic lights in a traffic management system for switching the traffic lights one by one, in order.

7.Round robin scheduling Algorithm of OS

First, the processes which are eligible to enter the ready queue enter the ready queue. After entering the first process in Ready Queue is executed for a Time Quantum chunk of time. After execution is complete, the process is removed from the ready queue. Even now the process requires some time to complete its execution, then the process is added to Ready Queue.

The material in this presentation belongs to St. Francis Institute of Technology and is solely for educational purposes. Distribution and modifications of the content is prohibited.

Application of Graph:

-different algorithms to calculate the shortest path between the vertices of a graph G.

These algorithms include:

- Minimum spanning tree
- Dijkstra's algorithm
- Warshall's algorithm

The material in this presentation belongs to St. Francis Institute of Technology and is solely for educational purposes. Distribution and modifications of the content is prohibited.

Application of Graph:

- Minimum spanning tree -
- A spanning tree of a connected, undirected graph G is a sub-graph of G which is a tree that connects all the vertices together. A graph G can have many different spanning trees.
- We can assign *weights* to each edge, and use it to assign a weight to a spanning tree by calculating the sum of the weights of the edges in that spanning tree.

- A *minimum spanning tree* (MST) is defined as a spanning tree with weight less than or equal to the weight of every other spanning tree.

The material in this presentation belongs to St. Francis Institute of Technology and is solely for educational purposes. Distribution and modifications of the content is prohibited.

Application of Graph:Minimum Spanning Tree

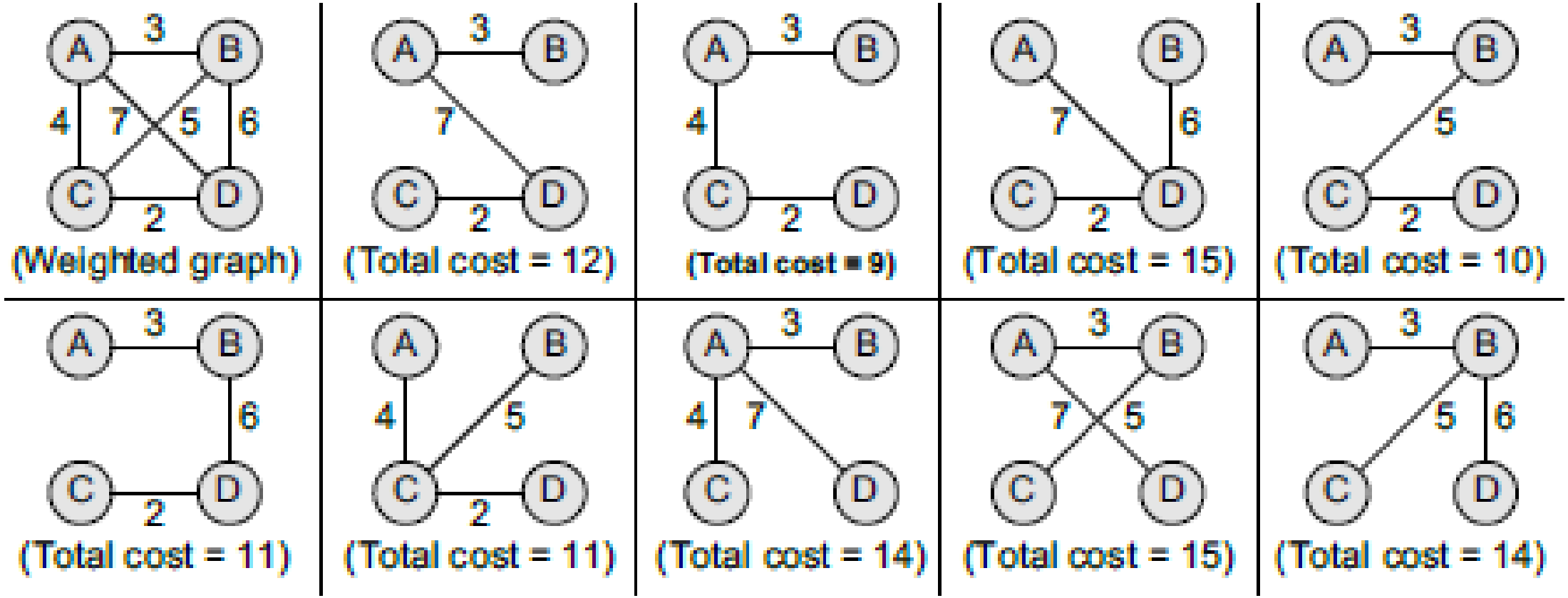


Figure 13.29 Weighted graph and its spanning trees

The material in this presentation belongs to St. Francis Institute of Technology and is solely for educational purposes. Distribution and modifications of the content is prohibited.

A *minimum spanning tree* (MST) is defined as a spanning tree with weight less than or equal to the weight of every other spanning tree.

1.Prim's Algorithm:
2.Kruskal's Algorithm:

The material in this presentation belongs to St. Francis Institute of Technology and is solely for educational purposes. Distribution and modifications of the content is prohibited.

Prim's Algorithm: It builds a tree that includes every vertex and a subset of the edges in such a way that the total weight of all the edges in the tree is minimized. For this, the algorithm maintains three sets of vertices which can be given as below:

1.Tree vertices Vertices that are a part of the minimum spanning tree T.
2.Fringe vertices Vertices that are currently not a part of T, but are adjacent to some tree vertex.
3.***Unseen vertices*** Vertices that are neither tree vertices nor fringe vertices fall under this category.

```
Step 1: Select a starting vertex
Step 2: Repeat Steps 3 and 4 until there are fringe vertices
Step 3:     Select an edge e connecting the tree vertex and
            fringe vertex that has minimum weight
Step 4:     Add the selected edge and the vertex to the
            minimum spanning tree T
        [END OF LOOP]
Step 5: EXIT
```

The material in this presentation belongs to St. Francis Institute of Technology and is solely for educational purposes. Distribution and modifications of the content is prohibited.

Prim's Algorithm(Simplified)

The steps for implementing Prim's algorithm are as follows:

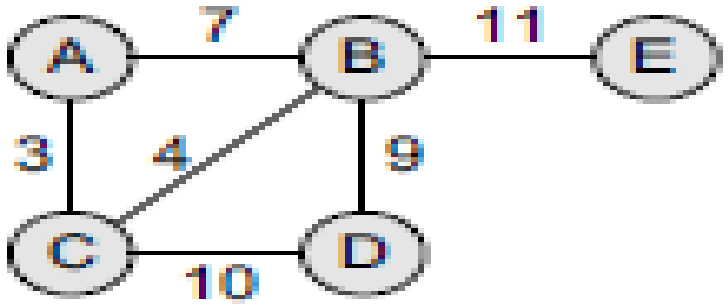
1.Initialize the minimum spanning tree with a vertex chosen at random.

2.Find all the edges that connect the tree to new vertices, find the minimum and add it to the tree

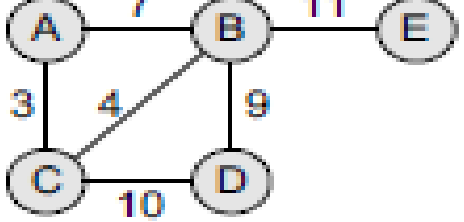
3.Keep repeating step 2 until we get a minimum spanning tree

The material in this presentation belongs to St. Francis Institute of Technology and is solely for educational purposes. Distribution and modifications of the content is prohibited.

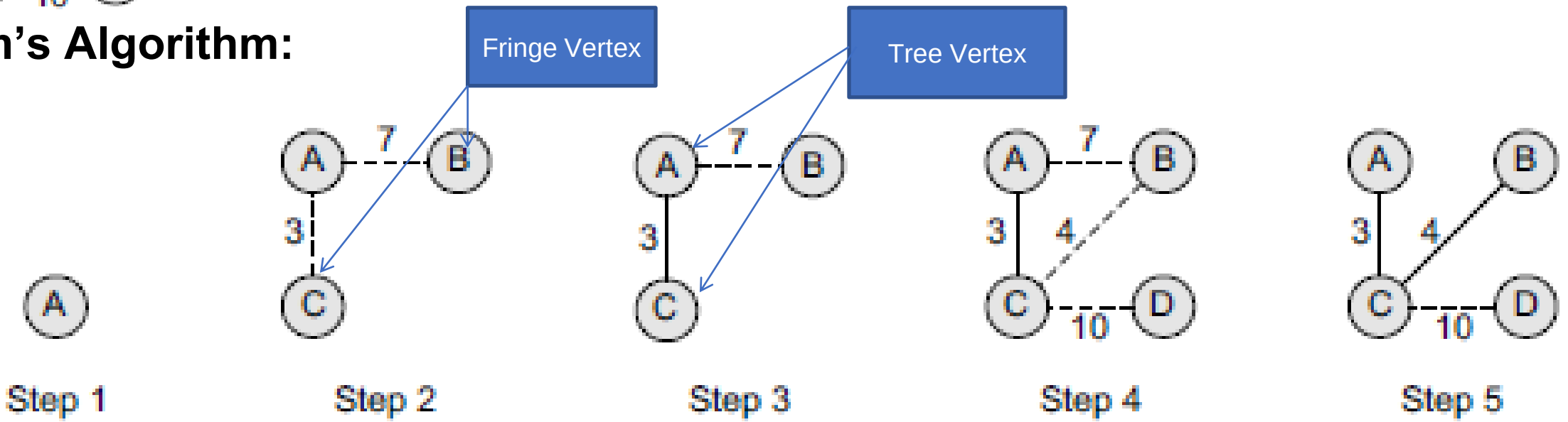
Prim's Algorithm: Example : Consider A as start Vertex and find MST

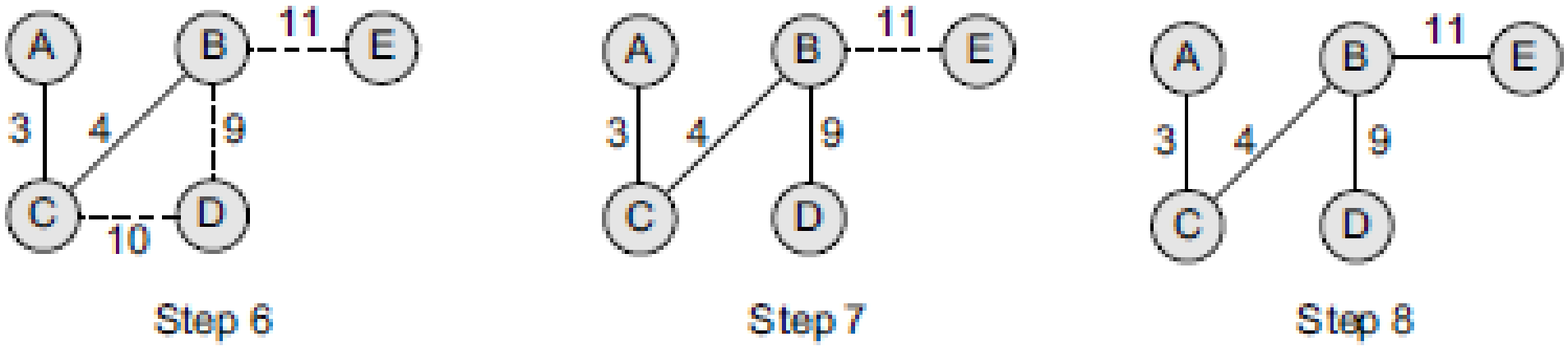


The material in this presentation belongs to St. Francis Institute of Technology and is solely for educational purposes. Distribution and modifications of the content is prohibited.



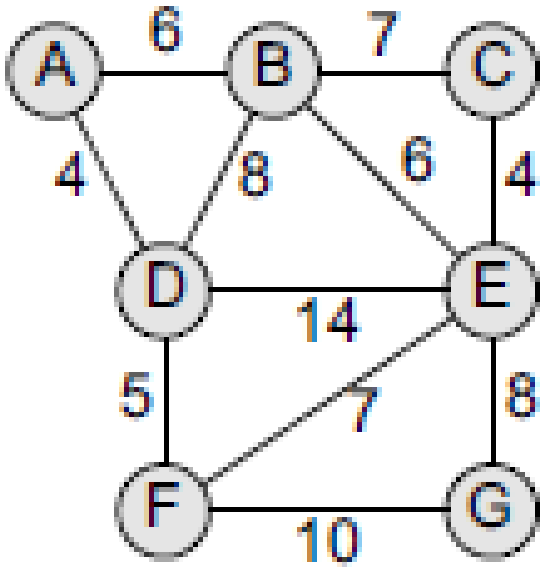
Prim's Algorithm:





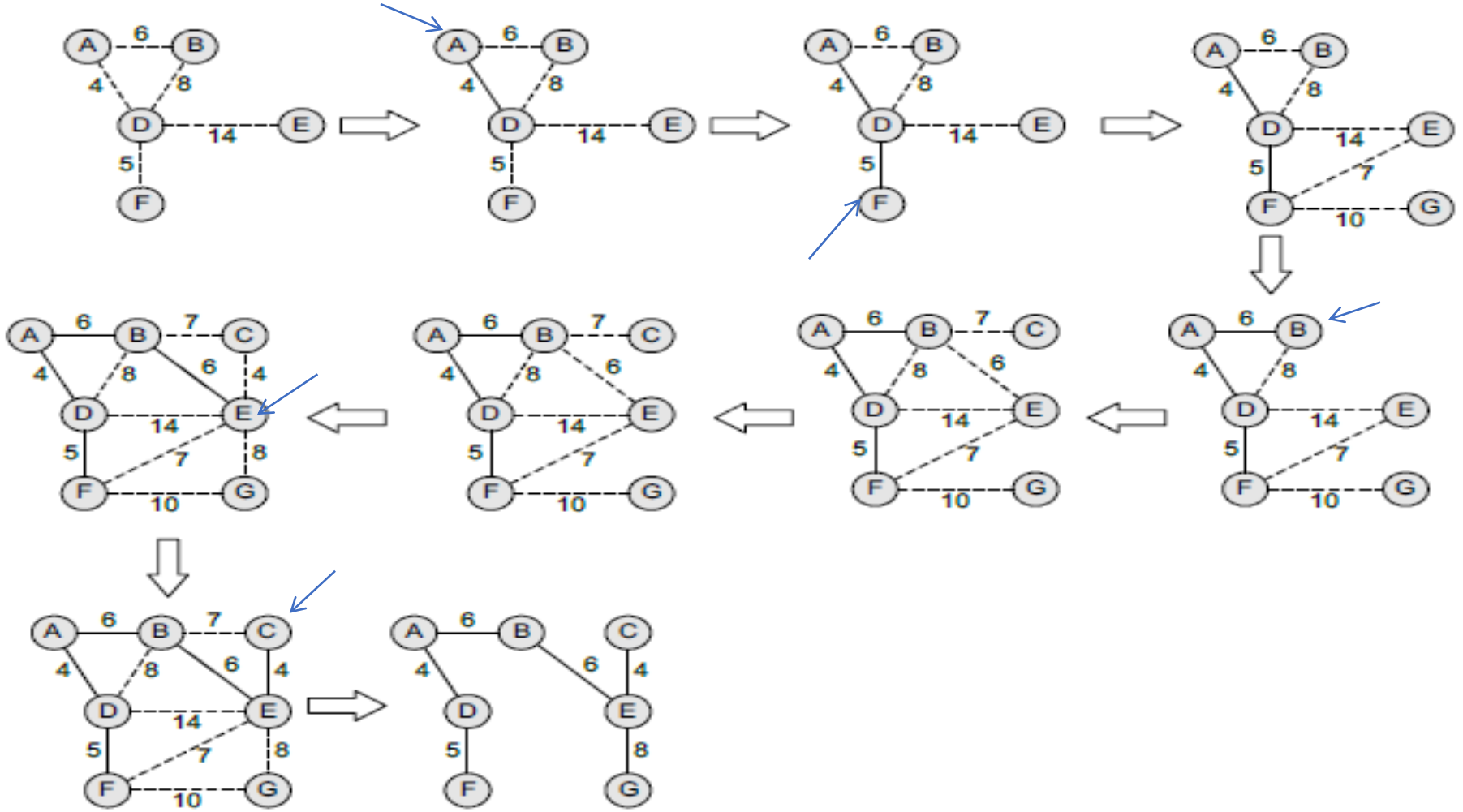
The material in this presentation belongs to St. Francis Institute of Technology and is solely for educational purposes. Distribution and modifications of the content is prohibited.

Prim's Algorithm: Example : Consider D as start Vertex and find MST



The material in this presentation belongs to St. Francis Institute of Technology and is solely for educational purposes. Distribution and modifications of the content is prohibited.

Prim's Algorithm: Example : Consider D as start Vertex and find MST



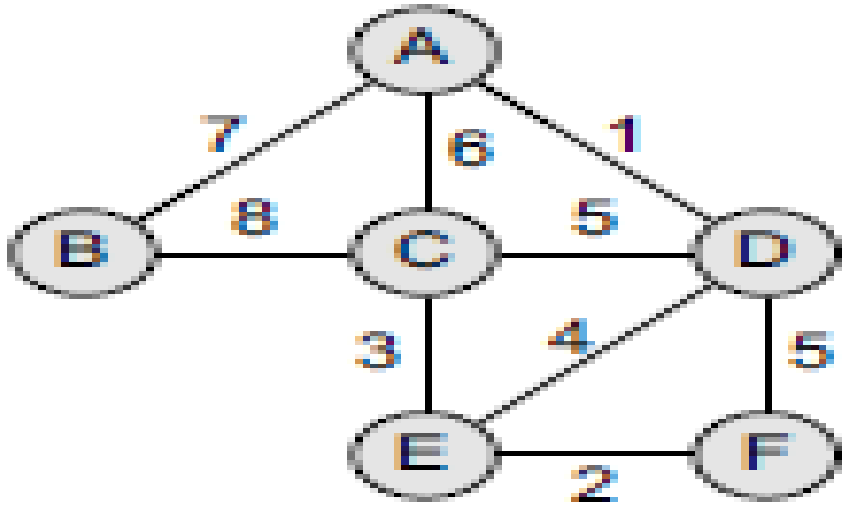
The material in this presentation belongs to St. Francis Institute of Technology and is solely for educational purposes. Distribution and modifications of the content is prohibited.

Kruskal's Algorithm: The algorithm aims to find a subset of the edges that forms a tree that includes every vertex. The total weight of all the edges in the tree is minimized. However, if the graph is not connected, then it finds a *minimum spanning forest*.

```
Step 1: Create a forest in such a way that each graph is a separate
        tree.
Step 2: Create a priority queue Q that contains all the edges of the
        graph.
Step 3: Repeat Steps 4 and 5 while Q is NOT EMPTY
Step 4:      Remove an edge from Q
Step 5: IF the edge obtained in Step 4 connects two different trees,
        then Add it to the forest (for combining two trees into one
        tree).
        ELSE
             Discard the edge
Step 6: END
```

The material in this presentation belongs to St. Francis Institute of Technology and is solely for educational purposes. Distribution and modifications of the content is prohibited.

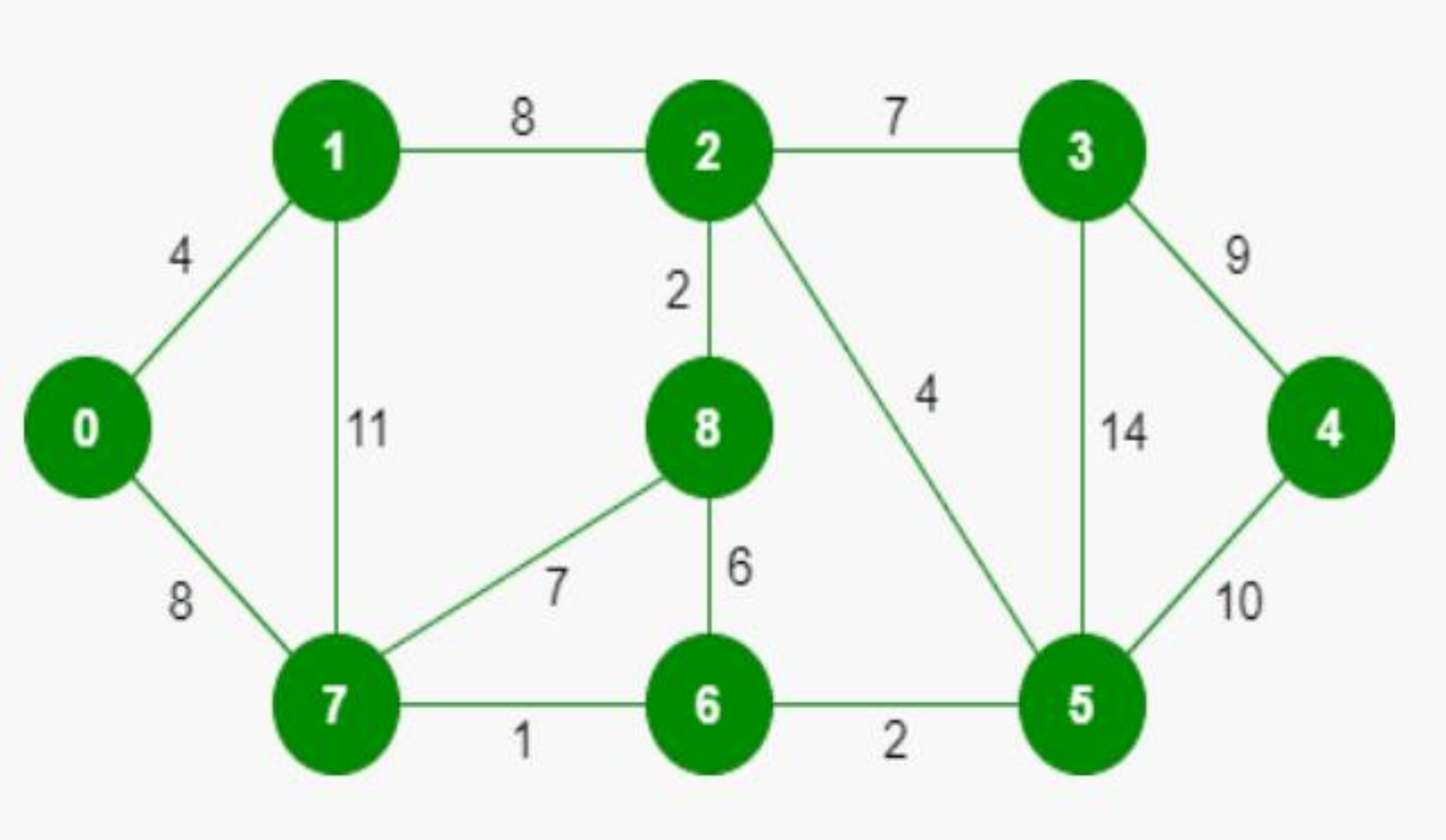
Kruskal's Algorithm: Example1:



Initially ,F={ {A},{B},{C},{D},{E},{F} }
Priority Queue={(A,D),(E,F),(C,E), (E,D), (C,D),(D,F),(A,C),(A,B),(B,C)}
MST={}

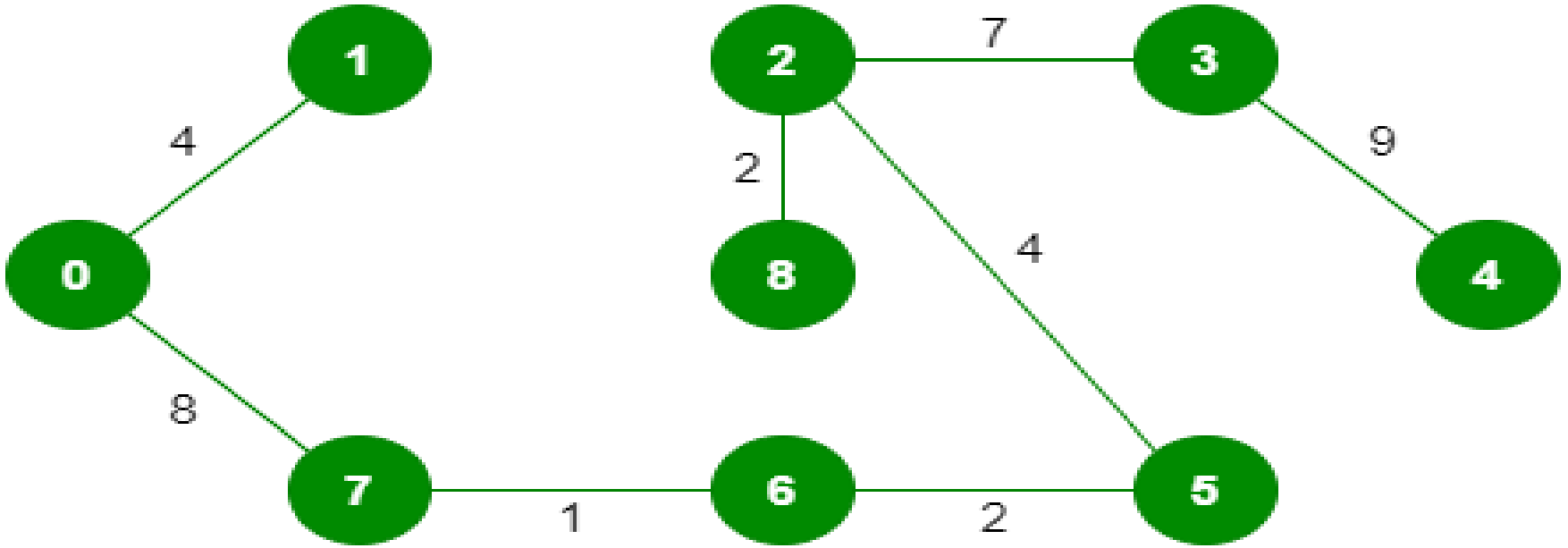
The material in this presentation belongs to St. Francis Institute of Technology and is solely for educational purposes. Distribution and modifications of the content is prohibited.

Solve by Prim's and Kruskal's Algorithm:Example2



The material in this presentation belongs to St. Francis Institute of Technology and is solely for educational purposes. Distribution and modifications of the content is prohibited.

Kruskals ’s Algorithm: Example(MST) with cost 37



The material in this presentation belongs to St. Francis Institute of Technology and is solely for educational purposes. Distribution and modifications of the content is prohibited.

- The advantage of Prim's algorithm is its complexity, which is better than Kruskal's algorithm.
- Therefore, Prim's algorithm is helpful when dealing with dense graphs that have lots of edges.
- However, Prim's algorithm doesn't allow us much control over the chosen edges when multiple edges with the same weight occur

The material in this presentation belongs to St. Francis Institute of Technology and is solely for educational purposes. Distribution and modifications of the content is prohibited.

Dijkstra's Algorithm:

- Helpful in geometry , road networks, shortest path in routing, to study molecules structures in chemistry.

The material in this presentation belongs to St. Francis Institute of Technology and is solely for educational purposes. Distribution and modifications of the content is prohibited.

Dijkstra's Algorithm:

a. Assign to every node tentative dist. Eg.0 for initial node and infinity for others.
b.Set initial node as current . Mark all as unvisited and create set of all unvisited node
c.For current node find its adjacent nodes and calculate dist
d.Compare new distance with current assigned value and assign a smaller one.
Eg a current node marked with dist 6 and edge connecting it neighour B has length 2, then dist to b from a is 6+2 = 8, if B is previously marked with greater dist than 8 then update it to 8.
e.When we consider all neighbours , mark current node as visited and remove from unvisited set
f.If destination node has been marked visited or if smallest tentative dist among the unvisited node set is infinity then stop
g.Otherwise select the unvisited node that is marked with smallest tentative dist, set is as the new current node and go back to step c.

The material in this presentation belongs to St. Francis Institute of Technology and is solely for educational purposes. Distribution and modifications of the content is prohibited.

Example 13.10 Consider the graph G given in Fig. 13.36. Taking D as the initial node, execute the Dijkstra's algorithm on it.

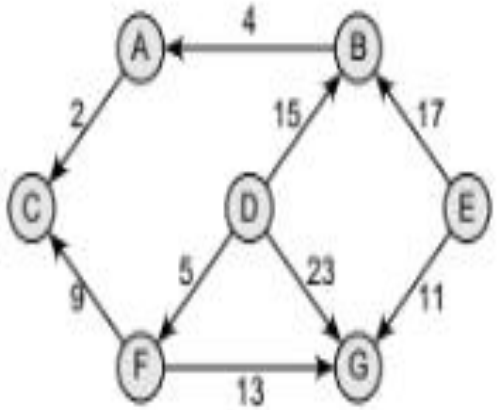


Figure 13.36 Graph G

Step 1: Set the label of D = 0 and N = {D}.

Step 2: Label of D = 0, B = 15, G = 23, and F = 5. Therefore, N = {D, F}.

Step 3: Label of D = 0, B = 15, G has been re-labelled 18 because minimum (5 + 13, 23) = 18, C has been re-labelled 14 (5 + 9). Therefore, N = {D, F, C}.

Step 4: Label of D = 0, B = 15, G = 18. Therefore, N = {D, F, C, B}.

Step 5: Label of D = 0, B = 15, G = 18 and A = 19 (15 + 4). Therefore, N = {D, F, C, B, G}.

Step 6: Label of D = 0 and A = 19. Therefore, N = {D, F, C, B, G, A}

Note that we have no labels for node E; this means that E is not reachable from D. Only the nodes that are in N are reachable from B.

The running time of Dijkstra's algorithm can be given as $O(|V|^2+|E|)=O(|V|^2)$ where V is the set of vertices and E in the graph.

The material in this presentation belongs to St. Francis Institute of Technology and is solely for educational purposes. Distribution and modifications of the content is prohibited.

Dijkstra's Algorithm:

Algorithm:

Function Dijkstra (graph, source)

1. Dist[source] ←0
2. Create vertexset Q
3. For each vertex v in graph:
   a. If v != source
   b. Dist[v] ← infinity
   c. prev [v] ← undefined
   d. Q.add_with_priority(v, dist[v])
      i. While Q is not empty:
      ii. U ← Q.Extract_min()
      iii. For each neighbour v of u:
         1. Alt ← dist[u] + length(u,v)
         2. If alt < dist[v]
            a. Dist[v] ← alt
            b. Prev[v] ← u\
         3. Q.decrease_priority(v,alt)
   e. Return dist, prev

The material in this presentation belongs to St. Francis Institute of Technology and is solely for educational purposes. Distribution and modifications of the content is prohibited.

Dijkstra's algorithm / shortest path algorithm-

Example:Refer PDF

The material in this presentation belongs to St. Francis Institute of Technology and is solely for educational purposes. Distribution and modifications of the content is prohibited.

Warshall's algorithm:

If a graph G is given as G=(V, E), where V is the set of vertices and E is the set of edges, the path matrix of G can be found as, P = A + A2 + A3 + ... + An.

$P_k[i][j]$ → 1 [if there is a path from v_i to v_j. The path should not use any other nodes except $v_1, v_2, \ldots, v_k$]

$P_k[i][j]$ → 0 [otherwise]

The material in this presentation belongs to St. Francis Institute of Technology and is solely for educational purposes. Distribution and modifications of the content is prohibited.

Warshall's algorithm:

1. If $P^1[i][j] = 1$, then there exists an edge from vi to vj that does not use any other vertex except v1.
2. If $P^2[i][j] = 1$, then there exists an edge from vi to vj that does not use any other vertex except v1 and v2.
3. Note that P^0 is equal to the adjacency matrix of G.
4. we can conclude that $P^k[i][j]$ is equal to 1 only when either of the two following cases occur:

- There is a path from vi to vj that does not use any other node except v1, v2, ..., vk–1. Therefore, Pk–1[i][j] = 1.
- There is a path from vi to vk and a path from vk to vj where all the nodes use v1, v2, ..., vk–1. Therefore, Pk–1[i][k] = 1 AND Pk–1[k][j] = 1

Hence, the path matrix P_k can be calculated with the formula given as:

$$P_k[i][j] = P_{k-1}[i][j] \vee (P_{k-1}[i][k] \wedge P_{k-1}[k][j])$$

where $\vee$ indicates logical OR operation and $\wedge$ indicates logical AND operation.

The material in this presentation belongs to St. Francis Institute of Technology and is solely for educational purposes. Distribution and modifications of the content is prohibited.

Warshall's algorithm:

```
Step 1: [INITIALIZE the Path Matrix] Repeat Step 2 for I = 0 to n-1,
        where n is the number of nodes in the graph
Step 2:     Repeat Step 3 for J = 0 to n-1
Step 3:           IF A[I][J] = 0, then SET P[I][J] = 0
                  ELSE P[I][J] = 1
           [END OF LOOP]
      [END OF LOOP]
Step 4: [Calculate the path matrix P] Repeat Step 5 for K = 0 to n-1
Step 5:     Repeat Step 6 for I = 0 to n-1
Step 6:           Repeat Step 7 for J=0 to n-1
Step 7:                 SET P_K[I][J] = P_K-1[I][J] V (P_K-1[I][K]
                        Λ P_K-1[K][J])
Step 8: EXIT
```

https://www.programiz.com/dsa/floyd-warshall-algorithm

The material in this presentation belongs to St. Francis Institute of Technology and is solely for educational purposes. Distribution and modifications of the content is prohibited.

Application of Tree:

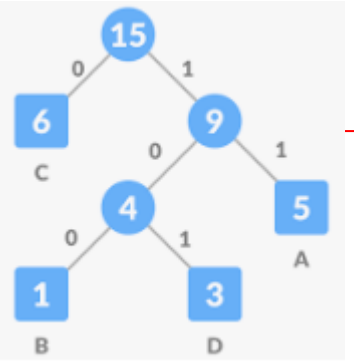
Huffman Tree and Heap Sort.

The material in this presentation belongs to St. Francis Institute of Technology and is solely for educational purposes. Distribution and modifications of the content is prohibited.

Application of Tree: Huffman Tree:

- Huffman coding is an entropy encoding algorithm developed by David A.
- Huffman that is widely used as a lossless data compression technique.
- The Huffman coding algorithm uses a variable-length code table to encode a source character where the variable-length code table is derived on the basis of the estimated probability of occurrence of the source character.

The material in this presentation belongs to St. Francis Institute of Technology and is solely for educational purposes. Distribution and modifications of the content is prohibited.



Application of Tree: Huffman Tree:

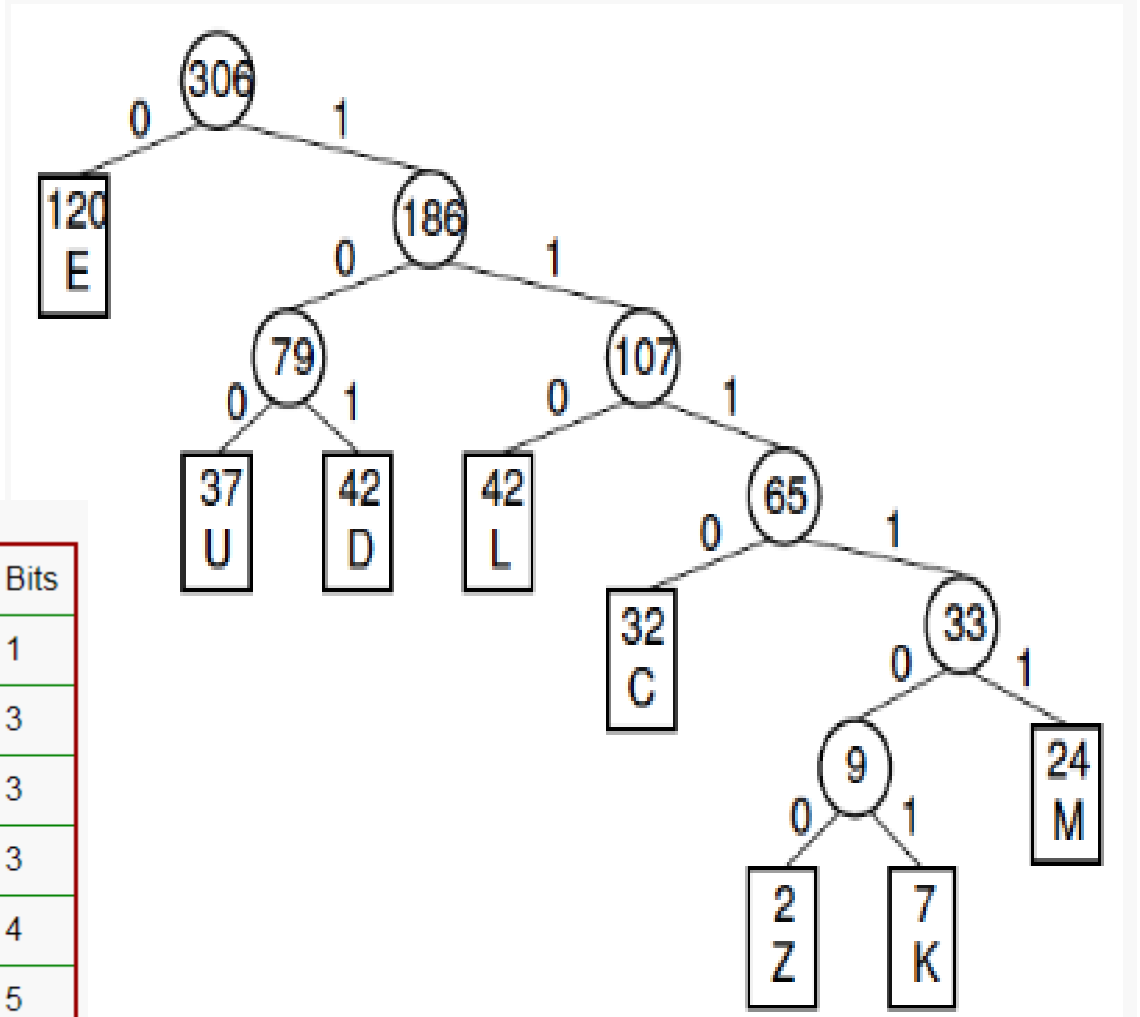
Step 1: Create a leaf node for each character. Add the character and its weight or frequency of occurrence to the priority queue.
Step 2: Repeat Steps 3 to 5 while the total number of nodes in the queue is greater than 1.
Step 3: Remove two nodes that have the lowest weight (or highest priority).
Step 4: Create a new internal node by merging these two nodes as children and with weight equal to the sum of the two nodes' weights.
Step 5: Add the newly created node to the queue.

Figure 9.23 Huffman algorithm

The material in this presentation belongs to St. Francis Institute of Technology and is solely for educational purposes. Distribution and modifications of the content is prohibited.

Huffman Tree:Example1

Letter	Z	K	M	C	U	D	L	E
Frequency	2	7	24	32	37	42	42	120



Huffman code

Letter	Freq	Code	Bits
E	120	0	1
D	42	101	3
L	42	110	3
U	37	100	3
C	32	1110	4
M	24	11111	5
K	7	111101	6
Z	2	111100	6

The material in this presentation belongs to St. Francis Institute of Technology and is solely for educational purposes. Distribution and modifications of the content is prohibited.

Data Coding

When we want to code our data (character) using bits, then we use r bits to code 2^r characters. For example, if r=1, then two characters can be coded. If these two characters are A and B, then A can be coded as 0 and B can be coded as 1 and vice versa.

Table 9.3 Characters with their codes

Character	Code
A	00
E	01
R	11
W	1010
X	1000
Y	1001
Z	1011

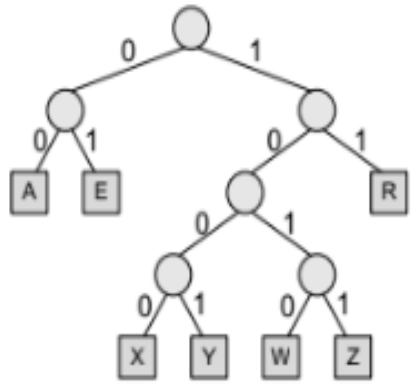


Figure 9.24 Huffman tree

The material in this presentation belongs to St. Francis Institute of Technology and is solely for educational purposes. Distribution and modifications of the content is prohibited.

Huffman Tree:Example2.

A	B	C	D	E	F	G	H	I	J
7	9	11	14	18	21	27	29	35	40

16	C	D	E	F	G	H	I	J
	11	14	18	21	27	29	35	40

A	B
7	9

-Rearrange Priority Queue in ascending order and repeat process

The material in this presentation belongs to St. Francis Institute of Technology and is solely for educational purposes. Distribution and modifications of the content is prohibited.

Huffman Coding Example:"XBXABZZXAD" generate

Huffman code for given String:

(Data, Frequency):(X,3),(B,2)(A,2)(D,1)(z,2)

Priority Queue: (D,1)(A,2)(B,2)(z,2)(X,3)

Priority Queue(B,2)(z,2)(AD,3),(X,3)

Priority Queue(AD,3),(X,3) ,(BZ,4)

Priority Queue(BZ,4),(ADX,6)

Priority Queue(BZADX,10)

Huffman Codes:
X=11
D=100
A=101
Z=01
B=00

The material in this presentation belongs to St. Francis Institute of Technology and is solely for educational purposes. Distribution and modifications of the content is prohibited.

Heapsort:

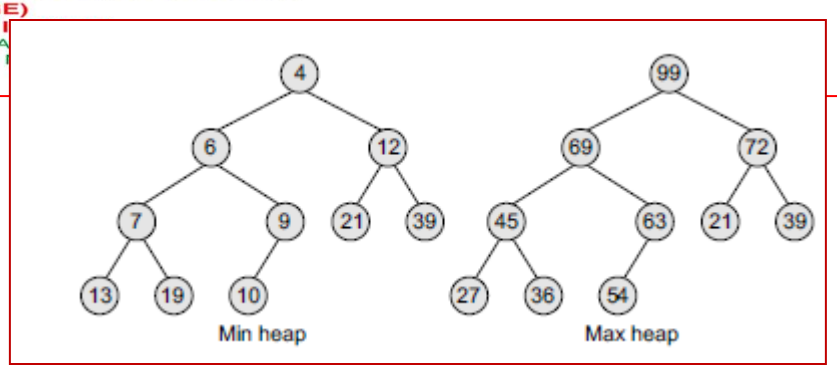
- A binary heap is a complete binary tree in which every node satisfies the heap property which states that:

If B is a child of A, then key(A) ≥ key(B)

- This implies that elements at every node will be either greater than or equal to the element at its left and right child. Thus, the root node has the highest key value in the heap. Such a heap is commonly known as a *max-heap*.
- Alternatively, elements at every node will be either less than or equal to the element at its left and right child. Thus, the root has the lowest key value. Such a heap is called a *min-heap*.

The material in this presentation belongs to St. Francis Institute of Technology and is solely for educational purposes. Distribution and modifications of the content is prohibited.

Heap Sort:



- Being a complete binary tree, all the levels of the tree except the last level are completely filled.
- The height of a binary tree is given as $\log_2 n$, where n is the number of elements.
- Heaps (also known as partially ordered trees) are a very popular data structure for implementing priority queues.
- If the index of any element in the array is i, the element in the index 2i will become the left child and element in 2i+1 index will become the right child. Also, the parent of any element at index i is given by the lower bound of (i)/2.

The material in this presentation belongs to St. Francis Institute of Technology and is solely for educational purposes. Distribution and modifications of the content is prohibited.

Heapsort-

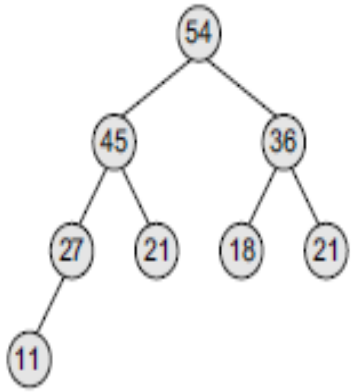


Figure 12.2 Binary heap

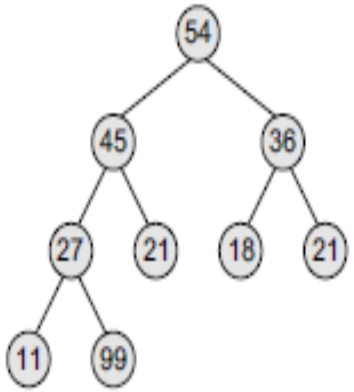


Figure 12.3 Binary heap after insertion of 99

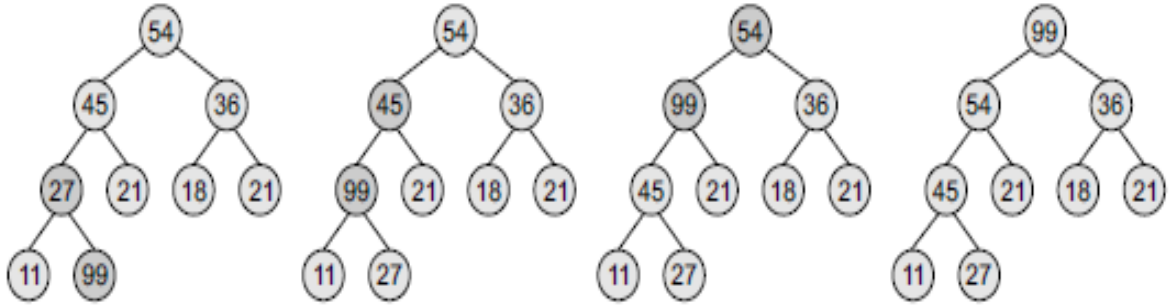
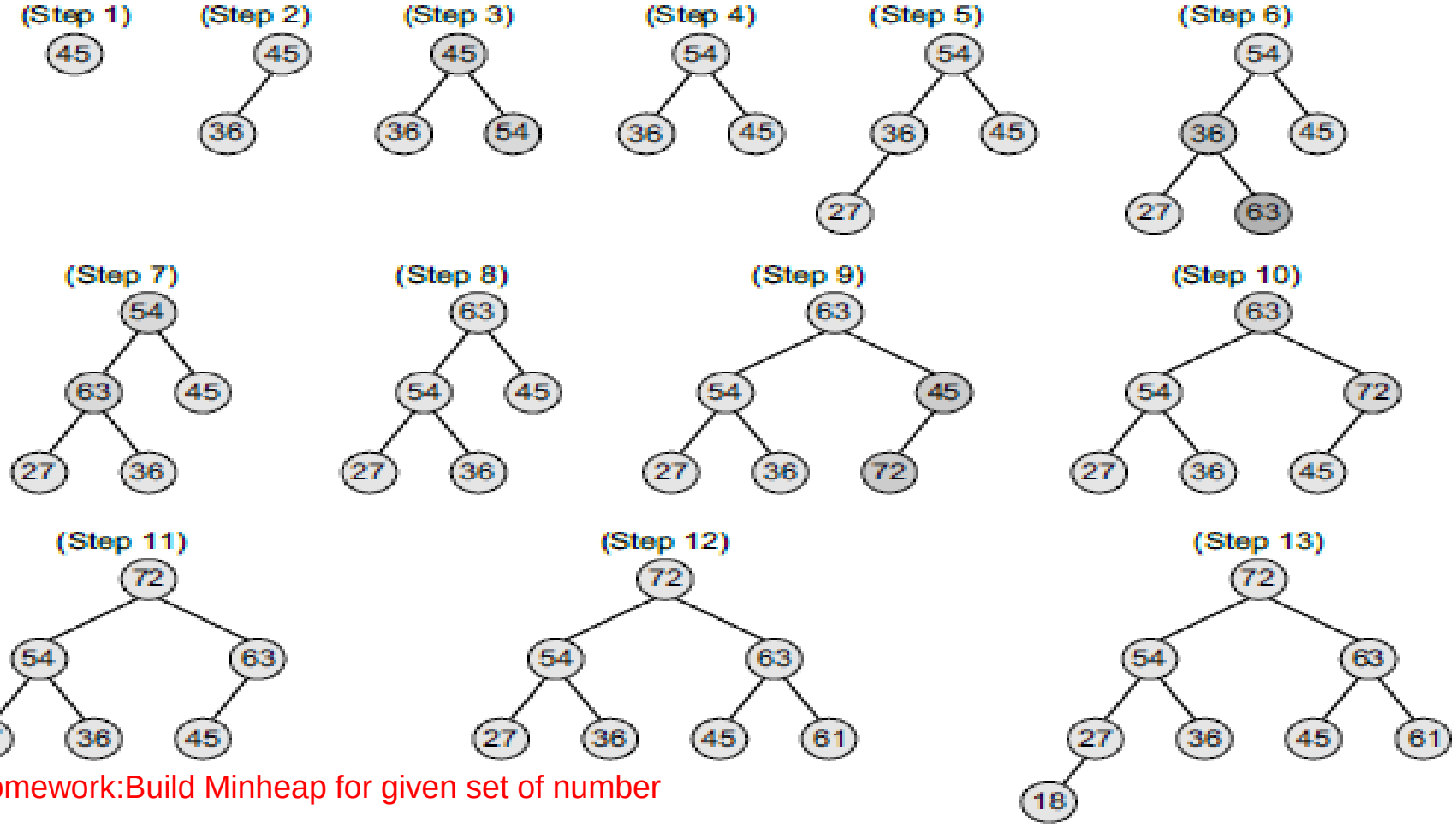


Figure 12.4 Heapify the binary heap

The material in this presentation belongs to St. Francis Institute of Technology and is solely for educational purposes. Distribution and modifications of the content is prohibited.

Heapsort-Example: Build a max heap H from the given set of numbers: 45, 36, 54, 27, 63, 72, 61,and 18. Also draw the memory representation of the heap.



Homework:Build Minheap for given set of number

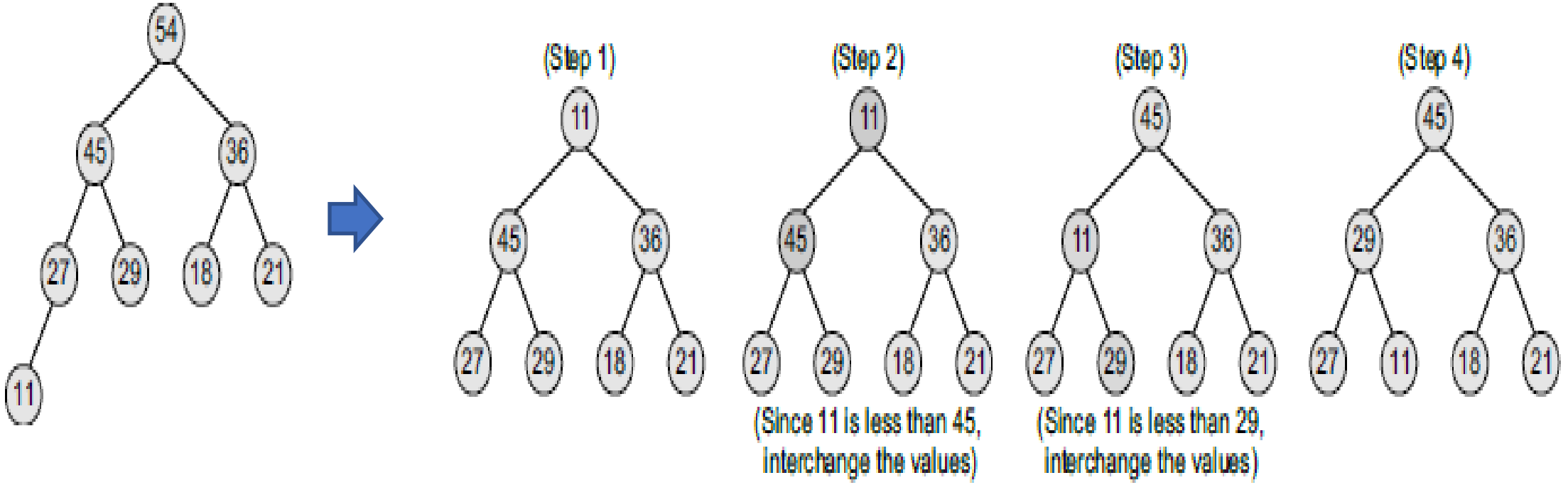
The material in this presentation belongs to St. Francis Institute of Technology and is solely for educational purposes. Distribution and modifications of the content is prohibited.

Heapsort-Deleting an Element from a Binary Heap:

1. Replace the root node's value with the last node's value so that H is still a complete binary tree but not necessarily a heap.
2. Delete the last node.
3. Sink down the new root node's value so that H satisfies the heap property. In this step, interchange the root node's value with its child node's value (whichever is largest among its children).

The material in this presentation belongs to St. Francis Institute of Technology and is solely for educational purposes. Distribution and modifications of the content is prohibited.

Heapsort-Deleting an Element from a Binary Heap:



The material in this presentation belongs to St. Francis Institute of Technology and is solely for educational purposes. Distribution and modifications of the content is prohibited.

Applications of Heapsort-

Applications of Binary Heaps

1. Sorting an array using *heapsort* algorithm.

2. Implementing priority queues.

The material in this presentation belongs to St. Francis Institute of Technology and is solely for educational purposes. Distribution and modifications of the content is prohibited.

Thank You