Graphs and Its Representations

Dr. Shrutilipi Bhattacharjee, Assistant Professor, Dept. of IT, NIT Karnataka, India

Tree

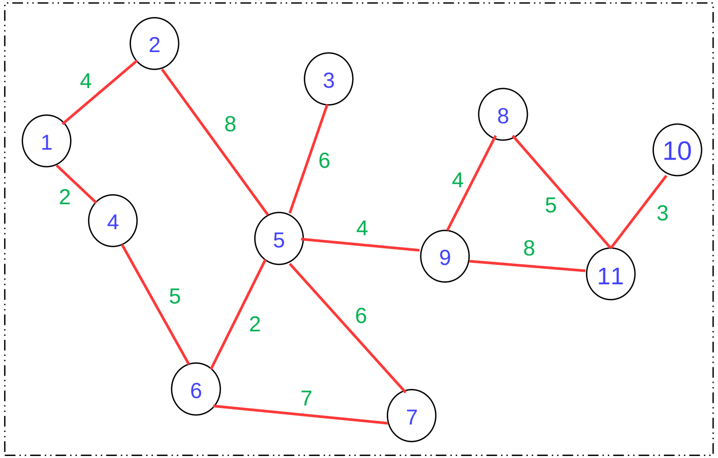
- Connected graph that has no cycles
- ***N*** vertex connected graph with ***N – 1*** edges
- In graph terminology, the term rooted tree is used to denote what we were earlier calling a ***tree***

Spanning Tree

- Subgraph that includes all vertices of the original graph
- Subgraph is a tree
- If original graph has ***N*** vertices, the spanning tree has ***N*** vertices and ***N – 1*** edges

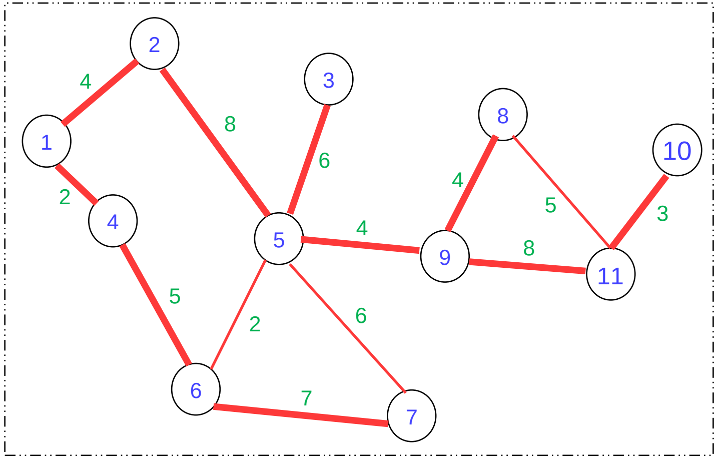
Minimum Cost Spanning Tree

- Tree cost is sum of edge weights/costs



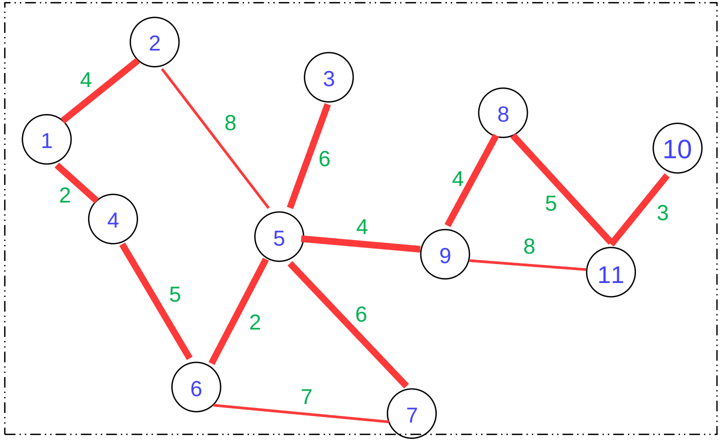
A Spanning Tree

- Spanning tree cost = 51

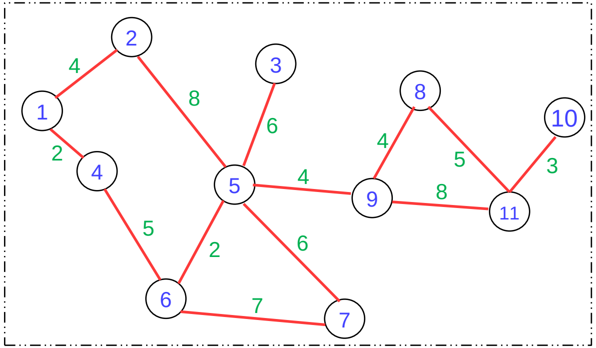


Minimum Cost Spanning Tree

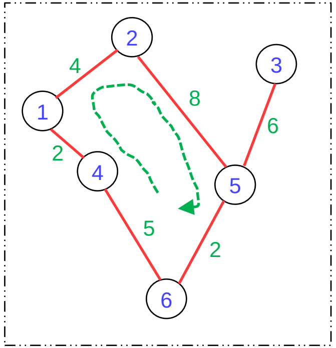
- Spanning tree cost = 41
- In the communication networks area, we are interested in finding minimum cost spanning trees



Tree vs. Subgraph vs. Spanning Tree

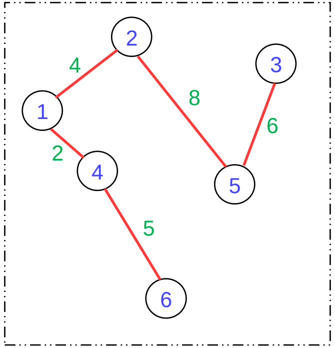


Graph ***G***



A subgraph of ***G***

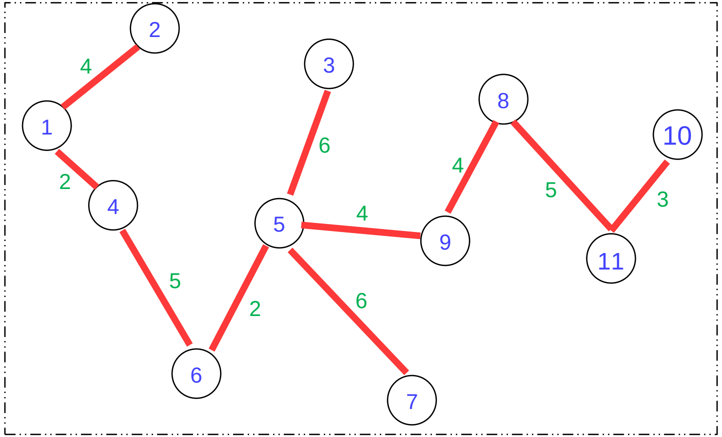
Not a tree



A tree of subgraph of ***G***

Not a spanning-tree

(may not include all the vertices)



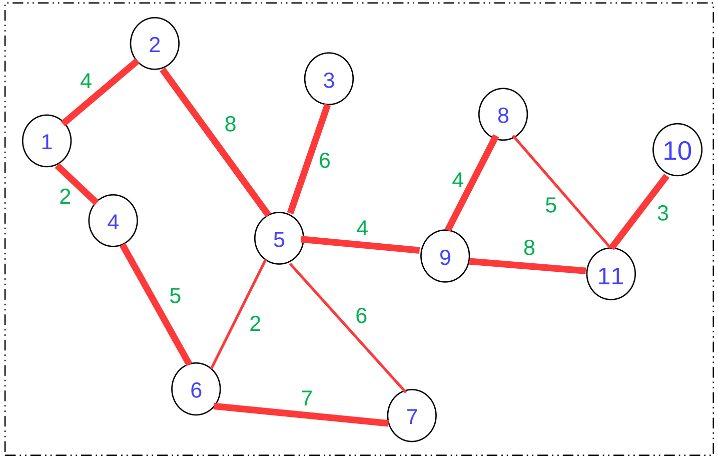
A spanning-tree of subgraph ***G***

Minimum Cost Spanning Tree

- Source = 1, weights = needed power
- Cost = 4 + 8 + 5 + 6 + 7 + 8 + 3 = 41

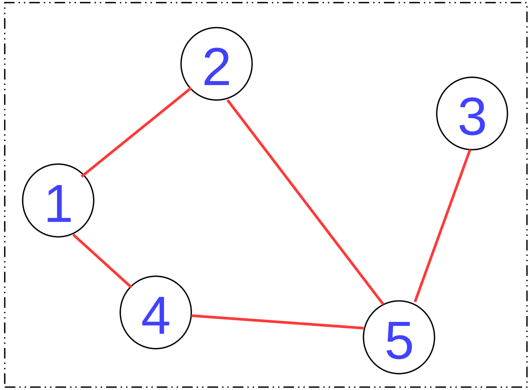
A Wireless Broadcast Tree

- Edge cost is power needed to reach a node
- If vertex 1 broadcasts with power 2, only vertex 4 is reached
- If it broadcasts with power 4, both 2 and 4 are reached
- Min-broadcast rooted spanning tree is NP-hard
- Cost of tree of previous slide becomes 26



Graph Representation

- Adjacency matrix
- Adjacency lists
  - Linked adjacency lists
  - Array adjacency lists



Adjacency matrix

- 0/1 ***N x N*** matrix, where ***N*** = number of vertices
- **A(i, j)** = 1 iff ***(i, j)*** is an edge

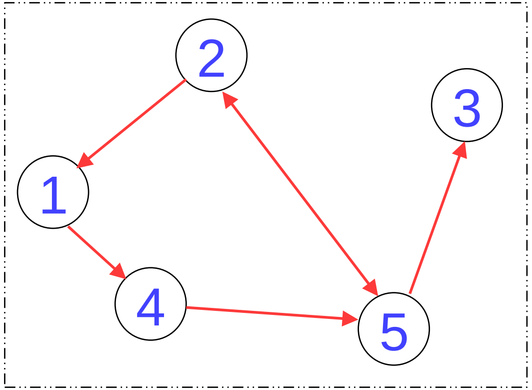
	1	**2**	**3**	**4**	**5**
1	0	1	0	1	0
2	1	0	0	0	1
3	0	0	0	0	1
4	1	0	0	0	1
5	0	1	1	1	0

Adjacency matrix properties

- Diagonal entries are zero
- Adjacency matrix of an undirected graph is symmetric
  - ***A(i, j)*** = ***A(j, i)*** for all ***i*** and ***j***

Adjacency Matrix (Digraph)

- Diagonal entries are zero
- Adjacency matrix of a digraph need not be symmetric



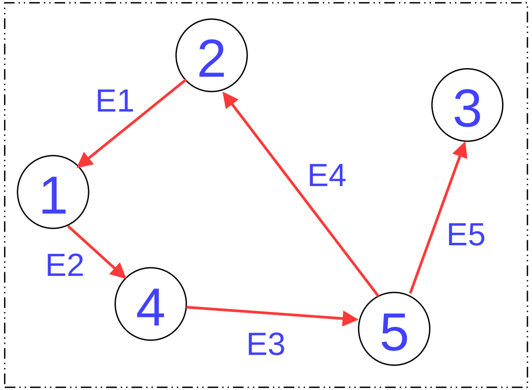
Adjacency matrix properties

- ***N^2*** bits of space
- For an undirected graph, may store only lower or upper triangle (exclude diagonal): ***$(N – 1)N/2$*** bits
- ***$O(N)$*** time to find vertex degree and/or vertices adjacent to a given vertex
- The disadvantage of adjacency matrices is their space demand of $\Theta(N^2)$
- Graphs are often sparse, with far fewer edges than $\Theta(N^2)$

	1	**2**	**3**	**4**	**5**
1	0	0	0	1	0
2	1	0	0	0	1
3	0	0	0	0	0
4	0	0	0	0	1
5	0	1	1	0	0

Incidence Matrix

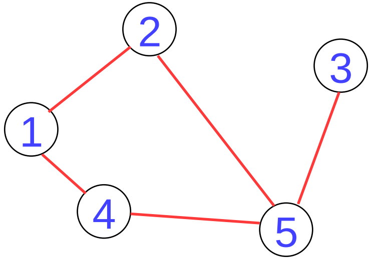
- In Incidence matrix representation, graph can be represented using a matrix of size: (Total number of vertices × total number of edges)
- In this matrix, columns represent edges and rows represent vertices
- This matrix is filled with either 0 or 1 or -1, where,
  - 0 is used to represent row edge which is not connected to column vertex
  - 1 is used to represent row edge which is connected as outgoing edge to column vertex
  - -1 is used to represent row edge which is connected as incoming edge to column vertex



	E1	E2	E3	E4	E5
1	-1	1	0	0	0
2	1	0	0	-1	0
3	0	0	0	0	-1
4	0	-1	1	0	0
5	0	0	-1	1	1

Adjacency Lists

- Adjacency list for vertex ***i*** is a linear list of vertices adjacent from vertex ***i***
- An array of ***N*** adjacency lists



A-List[1] = (2, 4)

A-List[2] = (1, 5)

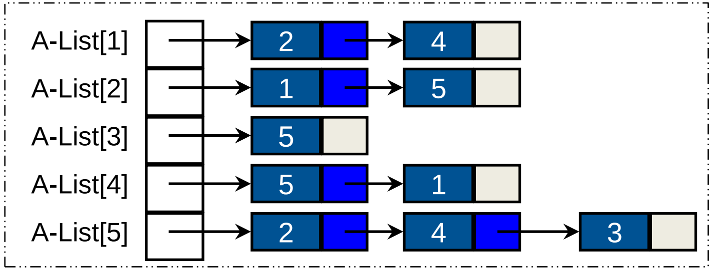
A-List[3] = (5)

A-List[4] = (5, 1)

A-List[5] = (2, 4, 3)

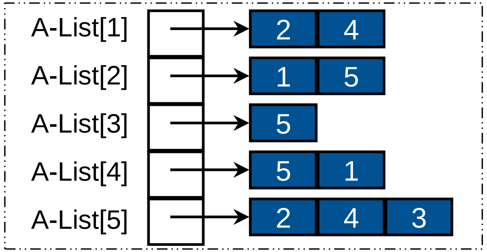
Linked Adjacency Lists

- Each adjacency list is a chain
- Array length = ***N***
- Number of chain nodes = 2E (undirected graph)
- Number of chain nodes = E (digraph)



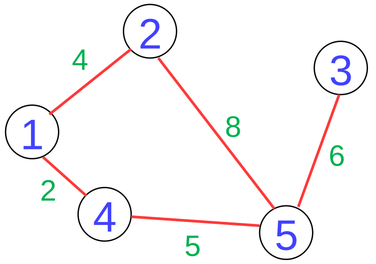
Array Adjacency Lists

- Each adjacency list is an array list
- Array Length = ***N***
- Number of list elements = 2E (undirected graph)
- Number of list elements = E (digraph)



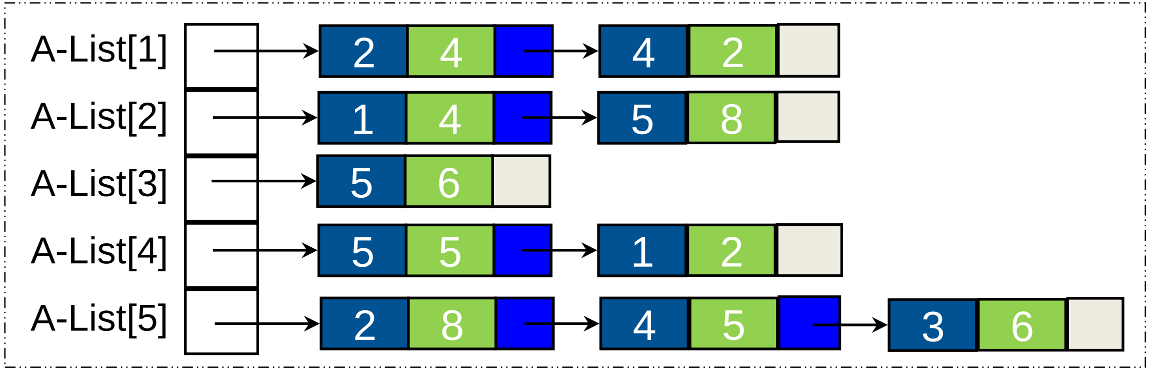
Weighted Graphs

- Cost adjacency matrix
- ***C(i, j)*** = cost of edge ***(i, j)***
- Adjacency lists → each list element is a pair (adjacent vertex, edge weight)



Abstract Methods of Graph

- int vertices();
- int edges();
- boolean existsEdge(int i, int j);
- putEdge(Object theEdge);
- void removeEdge(int i, int j);
- int degree(int i);
- int inDegree(int i);
- int outDegree(int i);



Graph Representation: Adjacency List

A structure to represent an adjacency list node

```
struct AdjListNode
{
    int dest;
    struct AdjListNode* next;
};
```

A structure to represent an adjacency list

```
struct AdjList
{
    struct AdjListNode *head;
};
```

A structure to represent a graph

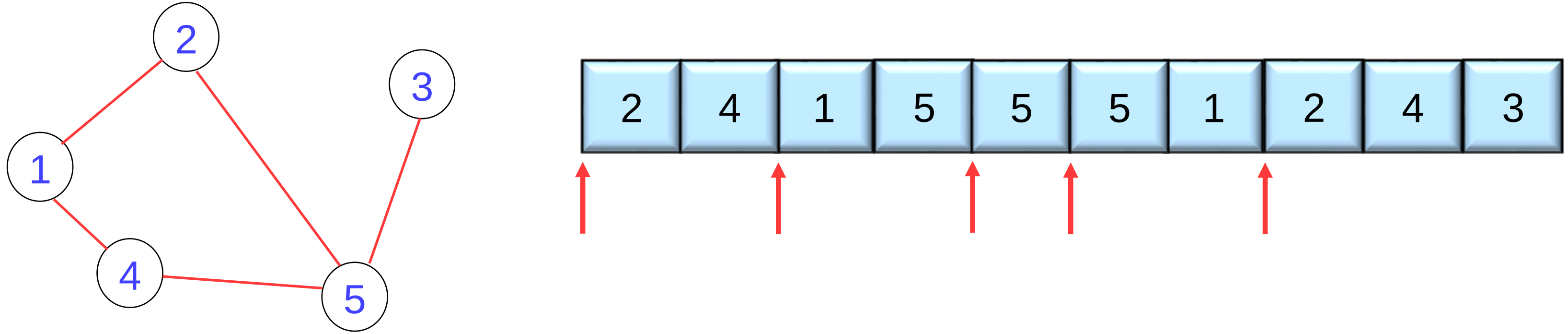
```
struct Graph
{
    int V;
    struct AdjList* array;
};
```

- Adjacency lists are not well suited for parallelism
- Since the lists require that we traverse the neighbors of a vertex sequentially

Adjacency Array

- Similar to an adjacency list, an adjacency array keeps the neighbors of all vertices, one after another, in an array

- It separately, keeps an array of indices that tell us where in the adj array to look for the neighbors of each vertex



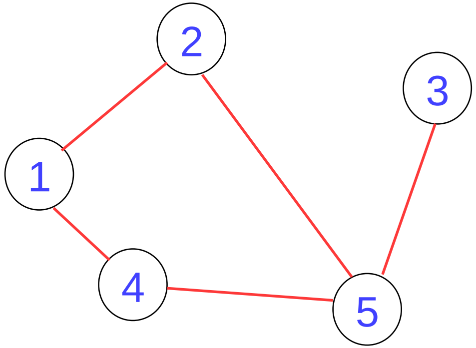
- The disadvantage of this approach is that it is not easy to insert new edges

Edge List

- Edge list: An unordered list of all edges in the graph

- Advantages
  - easy to loop/iterate over all edges

- Disadvantages
  - Hard to tell if an edge exists from A to B
  - Hard to tell how many edges A vertex touches (its degree)
  - New edges



1	1	2	3	4
2	4	5	5	5

Runtime Table

N vertices, ***E*** edges No parallel edges No self-loops	**Edge list**	**Adjacency list**	**Adjacency matrix**
Space	***N + E***	***N + E***	$\boldsymbol{N^2}$
Finding all adjacent vertices to ***v***	***E***	deg(***v***)	***N***
Determining if ***v*** is adjacent to ***w***	***E***	deg(***v***)	***1***
Inserting a vertex	***1***	***1***	$\boldsymbol{N^2}$
Inserting an edge	***1***	***1***	***1***
Removing vertex ***v***	***E***	***1***	$\boldsymbol{N^2}$
Removing an edge	***E***	deg(***v***)	***1***

Optimal Representation

- Choosing the optimal data structure to represent a given graph G is actually dependent on the density of edges within G
- This can be roughly summarized as follows:
  - If $|E| \approx |V|$ i.e., there are about as many edges as there are vertices then G is considered Sparse and an adjacency list is preferred
  - If $|E| \approx (|V|^2)$ i.e., is close to the maximum number of edges in G then it is considered Dense and the adjacency matrix is preferred
- Edge Lists are rarely used due to their poor adjacency complexity

Next Lecture

Graph Algorithms: Breadth-First Search (BFS)

Thank you for your attention...

Any question?

Contact:
Department of Information Technology, NITK Surathkal, India
6^{th} Floor, Room: 13
Phone: +91-9477678768
E-mail: shrutilipi@nitk.edu.in