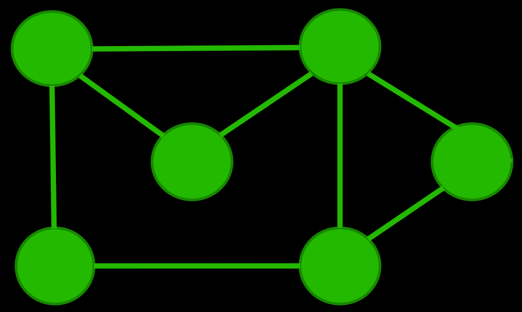
BM40A1500 DATA STRUCTURES AND ALGORITHMS

GRAPHS 2

2024

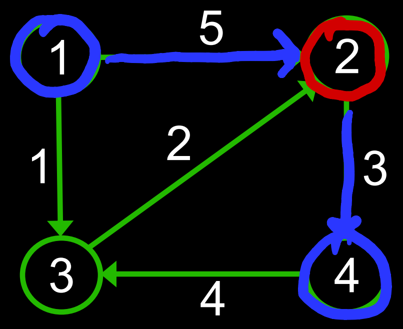
RECAP FROM LAST WEEK

- Why graphs are important?
  - Many problems can be formulated as graphs.
  - Large amount of graph algorithms exist.
- Graph Traversals
  - Depth-First Search
  - Breadth-First Search
- Shortest-paths problem
  - Dijkstra algorithm
  - Single-source shortest paths

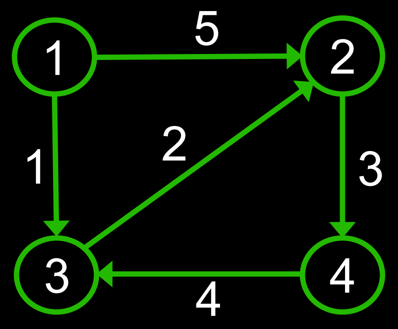


SHORTEST-PATHS PROBLEM: FLOYD’S ALGORITHM

- All-Pairs Shortest Paths
- Floyd’s algorithm:
  - Set distances between vertices to correspond the weight of the edges and set distances without edge as infinity.
  - Go trough all vertices and calculate distances using the vertex *k* (and all the vertices with index smaller than *k*) as intermediate vertices.
  - Update shortest distances when shorter paths are found.



	1	2	3	4
1	0	5	1	~~∞~~ 8
2	∞	0	∞	3
3	∞	2	0	∞
4	∞	∞	4	0



	1	2	3	4
1	0	3	1	6
2	∞	0	7	3
3	∞	2	0	5
4	∞	6	4	0

SHORTEST-PATHS PROBLEM: FLOYD'S ALGORITHM

```
procedure floyd(G)
# initilization
for i = 1 to N
    for j = 1 to N
        if G_{i,j} ≠ 0
            D_{i,j} = G_{i,j}
        else D_{i,j} = ∞

for k = 1 to N
    # test all paths that visit vertex k
    for i = 1 to N
        for j = 1 to N
            if D_{i,k} ≠ ∞ and D_{k,j} ≠ ∞ and D_{i,k} + D_{k,j} < D_{i,j}
                D_{i,j} = D_{i,k} + D_{k,j}
```

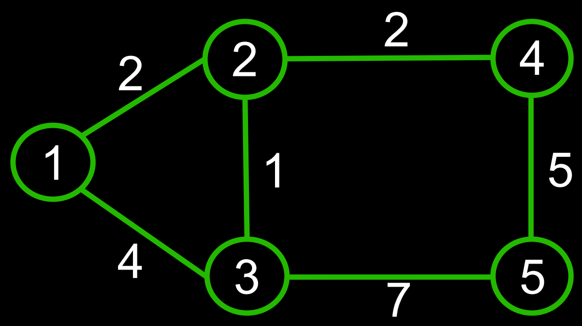
$\Theta(n^2)$

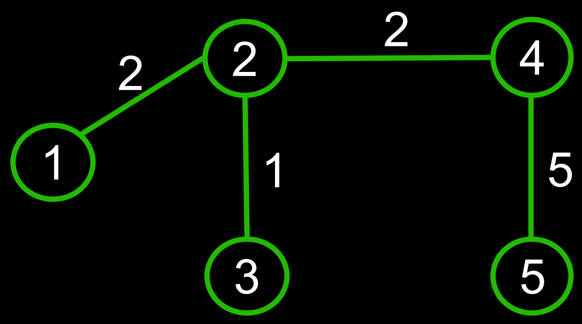
$\Theta(n^3)$

$\Theta(n^3)$

MINIMAL COST SPANNING TREES (MCST)

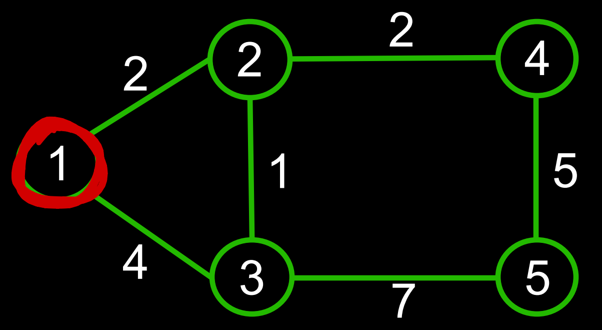
- Subset of edges that
  1. has minimum total cost as measured by summing the values for all the edges in the subset, and
  2. keeps the vertices connected.
- MCST cannot contain cycles → tree
- Applications:
  - Network design (electrical, road, computer, etc.)
  - Clustering
  - Approximation of very complex graph problems (e.g., traveling salesman problem)

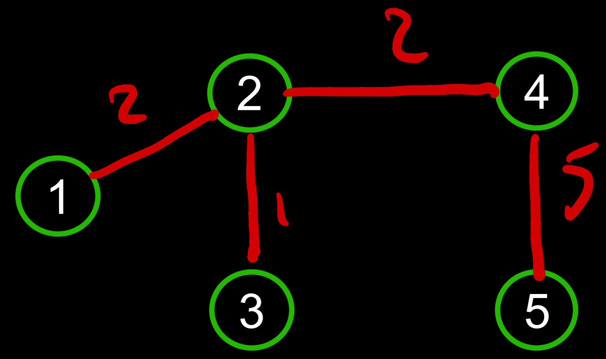




MINIMAL COST SPANNING TREES

- Prim's algorithm:
  - Start with any Vertex *v* in the graph,
  - Pick the least-cost edge connected to *v*. This edge connects *v* to another vertex (*u*)
  - Add Vertex *u* and Edge (*v*,*u*) to the MCST.
  - Next, pick the least-cost edge coming from either *v* or *u* to any other vertex in the graph.
  - Add this edge and the new vertex it reaches to the MCST.
  - Continue until all the vertices have been added.





LUT University

PRIM'S ALGORITHM

```
procedure prim(G, start)
MST = empty
for i = 1 to N  # Initialize
    D_i = ∞
    visited_i = False
D_start = 0

repeat N times
    v,e = FindNearestUnvisitedVertex(G, D, visited)
    visited_v = True
    if v ≠ start
        MST.append(e)
    for u = all neighbors of v
        if D_u > G_v,u
            D_u = G_v,u
```

$\Theta(n)$

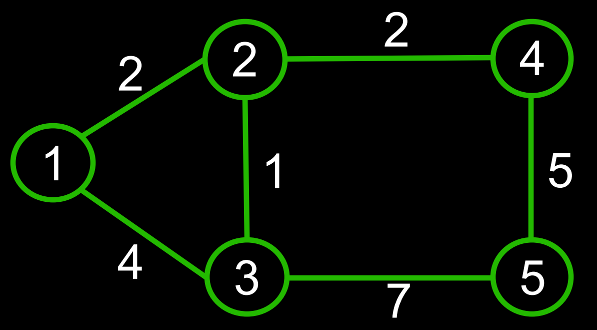
$\Theta(n^2)$

$\Theta(n^2)$

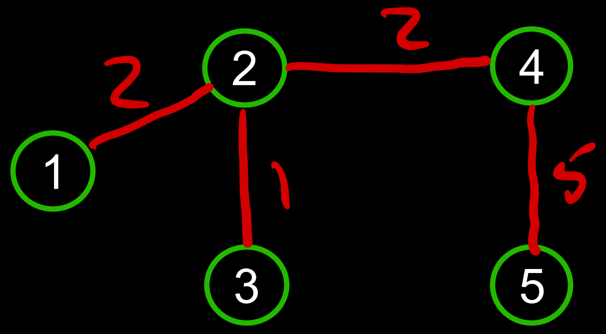
MINIMAL COST SPANNING TREES

- Kruskal's algorithm:
  - First partition the set of vertices into $|\mathbf{V}|$ disjoint sets, each consisting of one vertex.
  - Then process the edges in order of weight.
  - An edge is added to the MCST, and two disjoint sets combined, if the edge connects two vertices in different disjoint sets.
  - This process is repeated until only one disjoint set remains.

  - The principle is very simple, but to implement the algorithm, we need a data structure that allows
    - to check if two vertitces are in different disjoint sets, and
    - to combine two set of vertices.

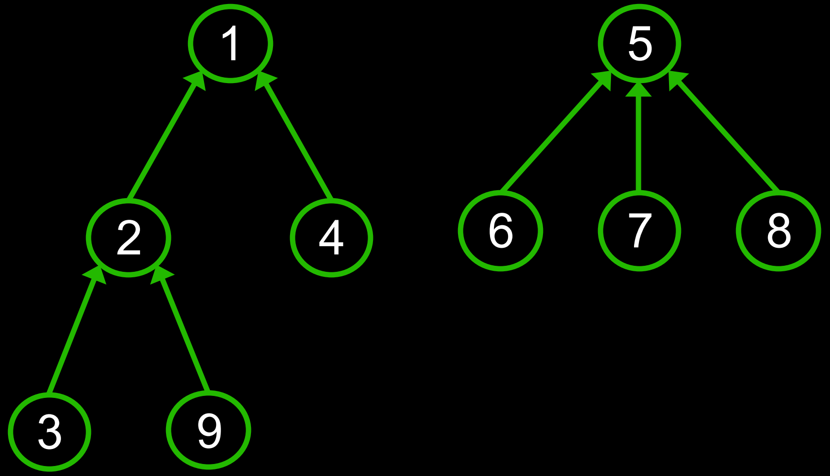


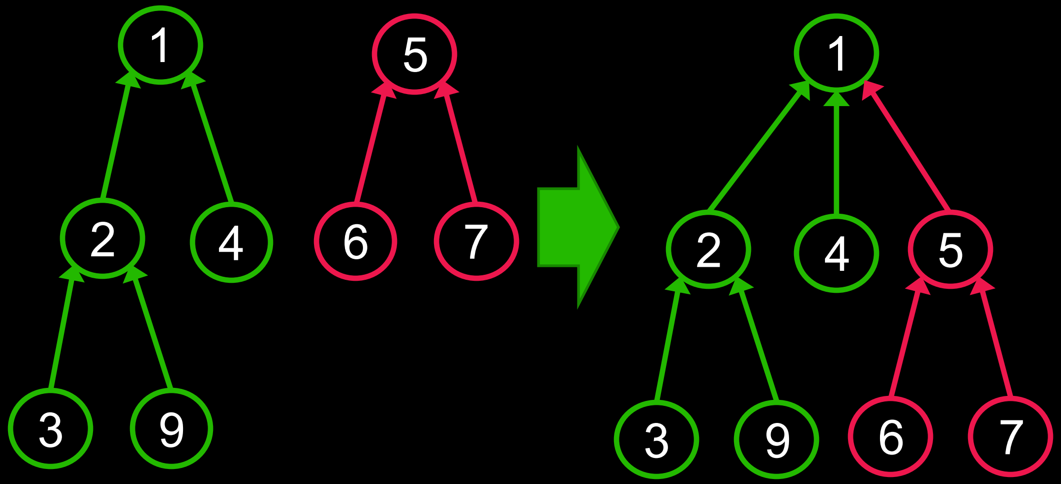
{2,3}: 1
{1,2}: 2
{2,4}: 2
{1,3}: 4
{4,5}: 5
{3,5}: 7



UNION/FIND STRUCTURE

- Data structure that provides efficient operation
  - To check if two elements are in the same set
  - To combine two sets
- The trick is to represent sets as tree structures, where instead of nodes having pointers to the children, nodes have pointers to the parent.
  - Easy to find the root node.
  - If two nodes have the same root node, they are part of the same set.
  - Two sets can be combined by making the root of the smaller tree as a child of the larger tree (union)





KURSKAL'S ALGORITHM

```
procedure Kruskal(G)
    MST = Empty
    for each v in G.V do
        MakeSet(v)    # Make own set for all vertices
    E = sort(G.E)     # Sort edges by weight (smallest to largest)
    for each {u, v} in E
        if FindSet(u) ≠ FindSet(v) then
            MST.append({u, v})
            Union(FindSet(u), FindSet(v))
```

$\Theta(E \log E)$ E = sort(G.E)

$\Theta(E)$

$\Theta(E \log E)$

if CountSets = 1 then return MST

```
MakeSet(v): create a set
FindSet(v): find the set (tree), where Vertex v is stored
Union(a,b): Combines sets a and b
```


LUT
University