Open Addressing

↙ Probing.

↘ Double Hashing

Integer [] buckets; ← Use Linear Probing

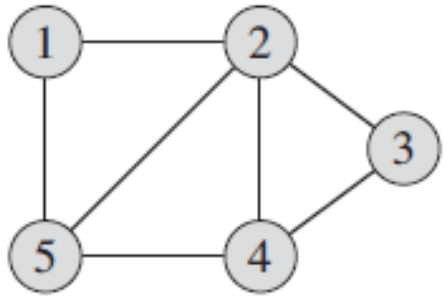
Store (Key, value)

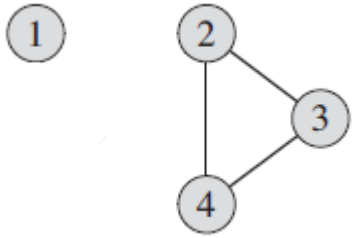
Chaining BST [] buckets

Graphs

What is a Graph?

- A non-linear data structure, like a Tree.
  (Linear data structure example: Linked List)
- Used to model pairwise relations between objects.
- Mathematically, a Graph G is defined as G = {V, E}.
  V = a nonempty set of Vertices
  E = possibly nonempty set of Edges
- Vertices (also referred as Nodes) represent the “objects”
- Edges are the links/lines that “connect” vertices.

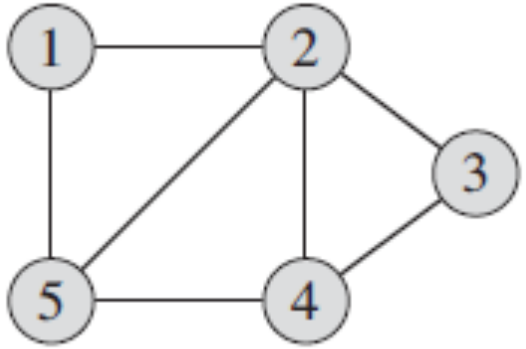




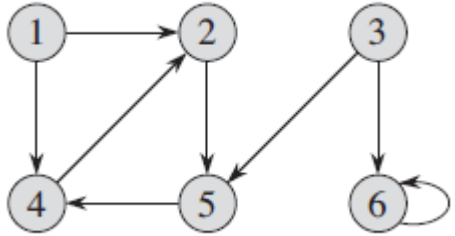
Types of Graphs

- Undirected Graphs

Edges do not have direction



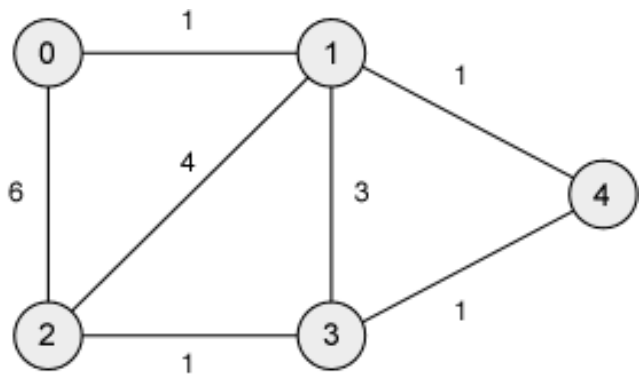
- Directed Graphs



Each edge has a direction

- Weighted Graphs

Each edge got a weight

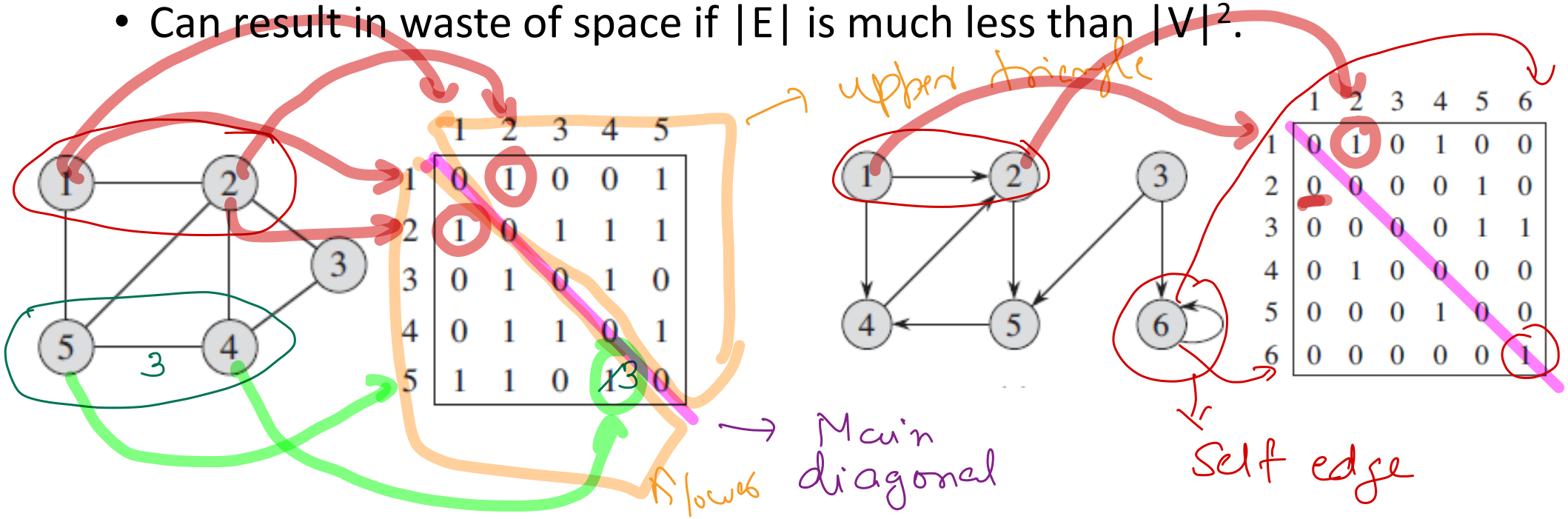


Application of Graphs

- Graphs are used to represent “flow”.
- To represent a map where roads are edges and the intersection of roads is a vertex. Example of directed weighted graph. Navigation systems can use shortest path algorithm to find shortest path between two points.
- Facebook users are considered as vertex and an edge exists between two users if they are “friends”. Example of undirected graph.
- In Operating System, used for Job Scheduling and Resource Allocation to detect deadlock.

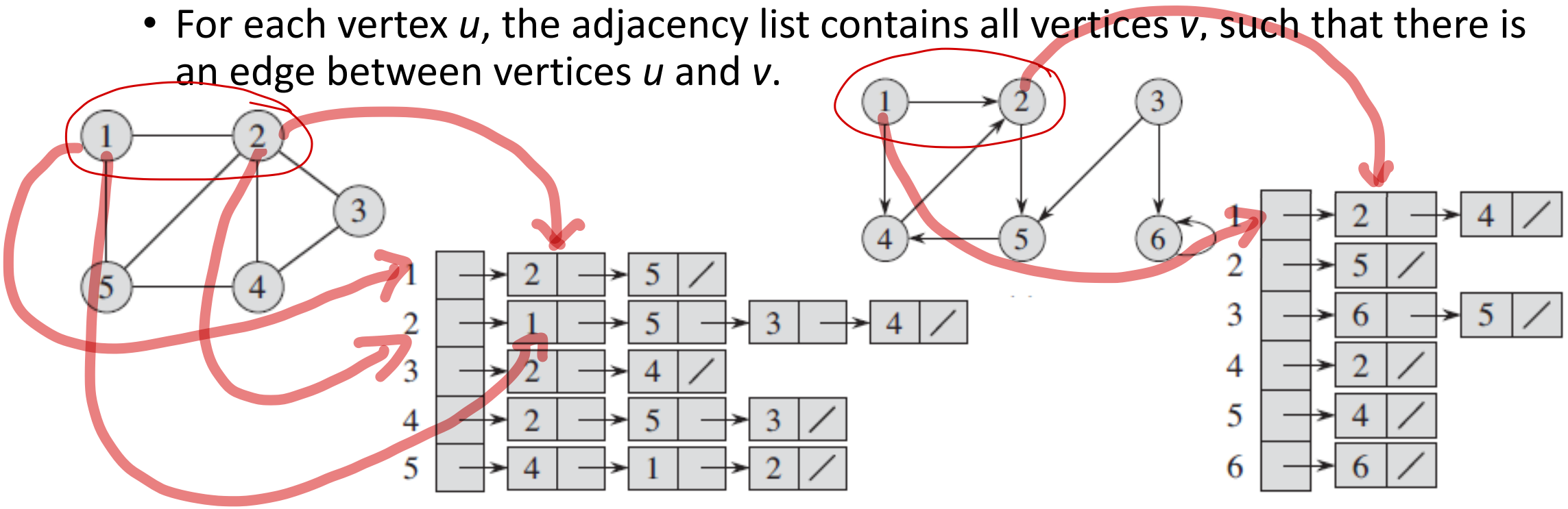
Graph Representations

- Adjacency Matrix
  - It is a 2D array of size |V| x |V|
  - Every cell a_{ij} is 1 if there exists an edge between vertex v_i and v_j and 0, otherwise.
  - Can result in waste of space if |E| is much less than $|V|^2$.



Graph Representations

- Adjacency List
  - It is a 1D array of |V| lists. One list for each vertex.
  - For each vertex *u*, the adjacency list contains all vertices *v*, such that there is an edge between vertices *u* and *v*.



Adj Matrix

```
EdgeInfo[][] adjMat;
```

Adj list

```
list < list < EdgeInfo >>
    adjlist
```

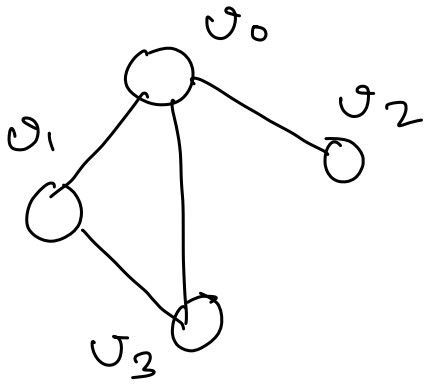
```
EdgeInfo {
    adjVertex
    weight
}
```

Graph Traversal

- Depth First (DFS)
- Breadth first (BFS)

Tree is a special type of graph

- Tree do not have cycle
- All nodes of tree are connected.



BFS

F T	F T	F T	F T
v_0	v_1	v_2	v_3

isVisited

queue

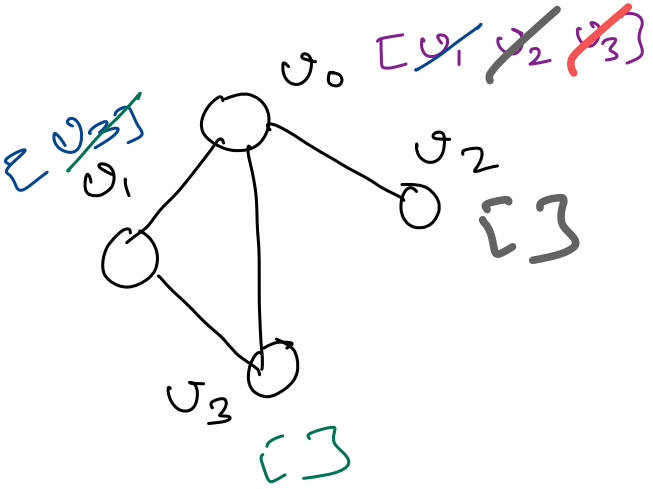
v_0	v_1	v_2	v_3	v_3

o/p: v_0 v_1 v_2 v_3

current → v_0 v_1 v_2 v_3 v_3

BFS()

- Create isVisited array of size vertexCount.
- Set all elements in isVisited to FALSE.
- Add startVertex to the queue. // We use 0 as startVertex
- while (queue is not empty) do
  - Remove vertex, vi, from queue.
  - if (vi is not visited) then
    - Mark vi as visited and process it.
    - For every adjacent vertex, vj, to vi that is not visited
      - Add vj to queue
- Stop



DFS

is Visited

v_0	v_1	v_2	v_3
T	T	T	T

current → v_0 v_1 v_3 v_2

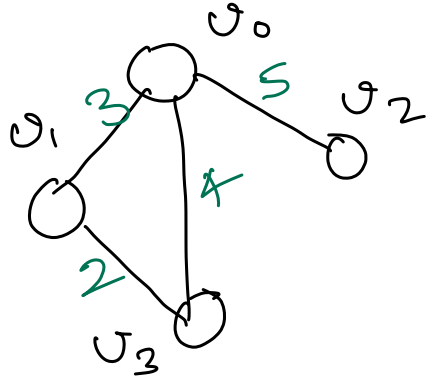
O/P: v_0 v_1 v_3 v_2

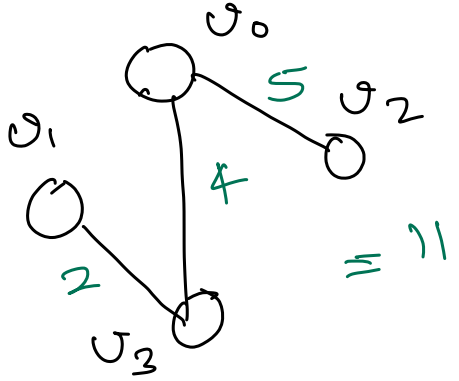
DFS()
- Create isVisited array of size vertexCount.
- Set all elements in isVisited to FALSE.
- DFSHelper(1, isVisited)
- Stop

DFSHelper(startVertex, isVisited)
- if (startVertex is visited) then
  - Stop
- Mark startVertex as visited
- Process startVertex
- For every adjacent vertex, vj, to startVertex that is not visited
  - DFSHelper(vj, isVisited)
- Stop

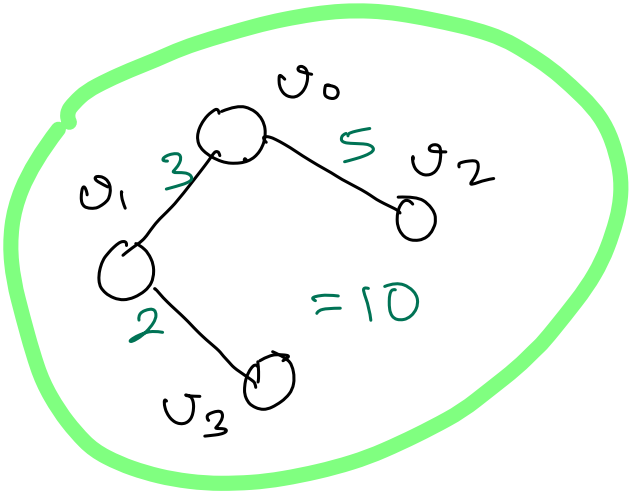
Spanning Tree

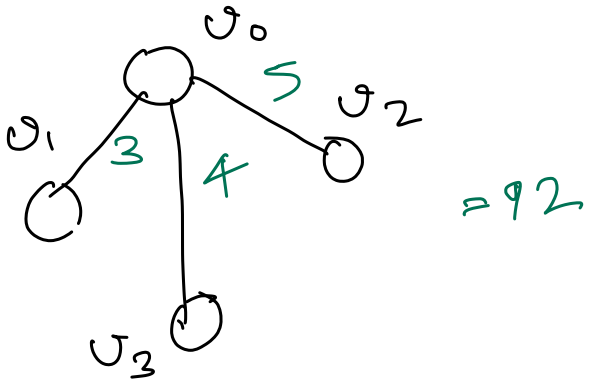
A graph with all vertices connected without forming a cycle





Brute Force → Enumerate all possible solutions.





Minimum Spanning Tree → Spanning Tree in which sum of edge weights is smallest.

. Spanning Tree
A tree that included all vertices of graph, with minimum (|V| - 1) possible edges, such that the graph is connected.

Minimum Spanning Tree
Spanning tree where sum of the weights of edges is minimum.

Connected Graph
From any vertex, we can reach any other vertex in the graph.

Greedy Algorithm ⟶ knapsack

At each step we locally optimize the solution.

Kruskal and Prim
Picks edge with minimum weight.

Dijkstra
Instead of picking the edge with the smallest weight, pick the vertex with the smallest distance.

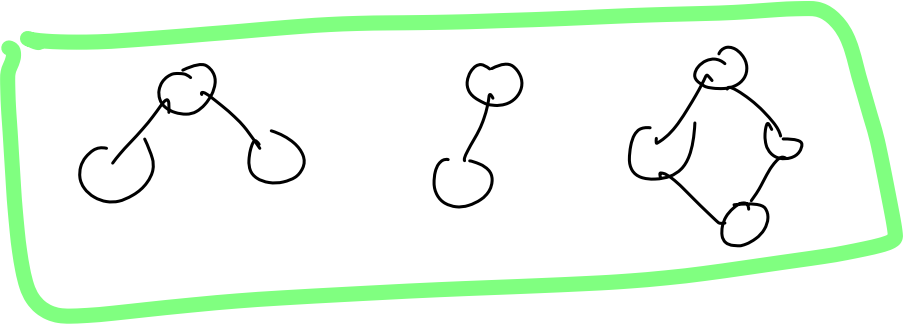
Spanning Tree
A tree that included all vertices of graph, with minimum (|V| - 1) possible edges, such that the graph is connected.

Minimum Spanning Tree
Spanning tree where sum of the weights of edges is minimum.

Connected Graph
From any vertex, we can reach any other vertex in the graph.

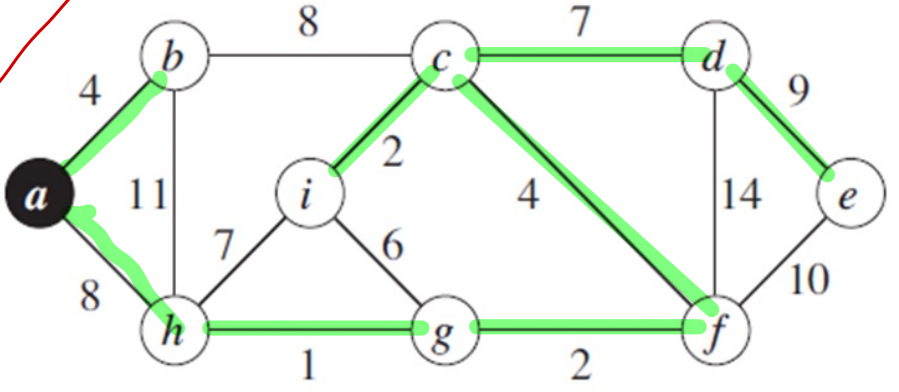
Exercise: How to check if a graph is connected?

Exercise: If a graph is not connected then how many disconnected parts are there?



Weight	U	V
1	G	H
2	C	I
2	F	G
4	A	B
4	C	F
6	G	I
7	C	D
7	I	J
8	A	H
8	B	C
9	D	E
10	E	F
11	B	H
14	D	F

→ Form cycle

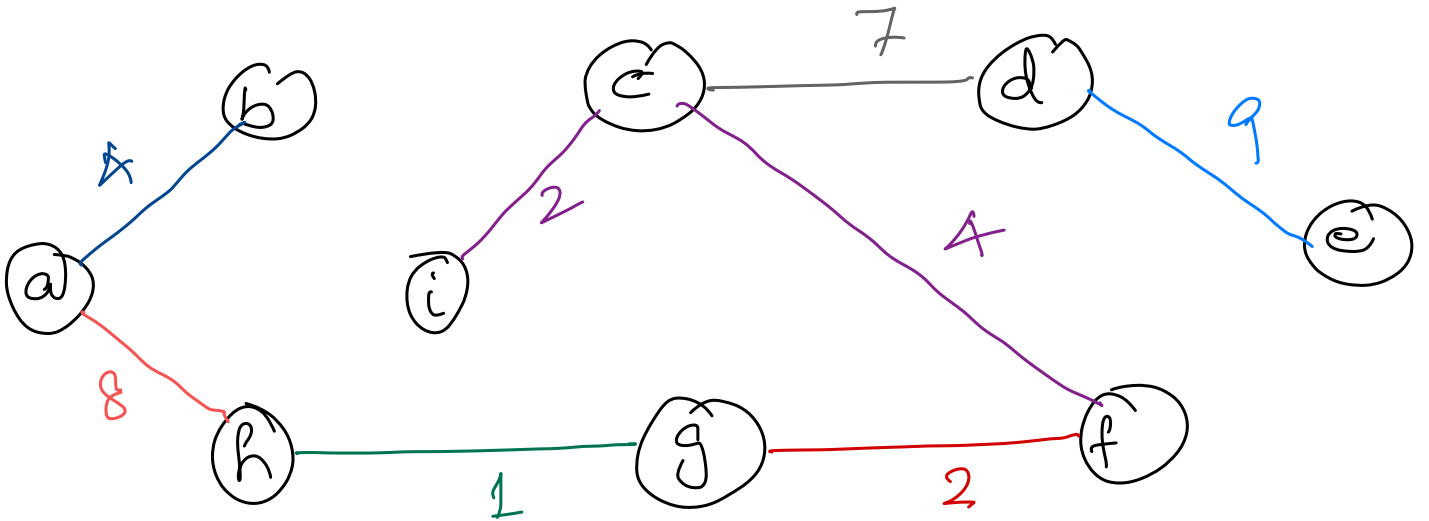


Kruskal's Minimum Spanning Tree Algorithm

- Initialise min spanning tree to empty (only vertices).
- Sort all edges in ascending order of their weight.
- while (vertexCount - 1) edges are not added to tree do
  - Pick the edge with min weight.
  - Add an edge to the tree if it does not form a cycle.
- Stop.

Traversal

Union Find

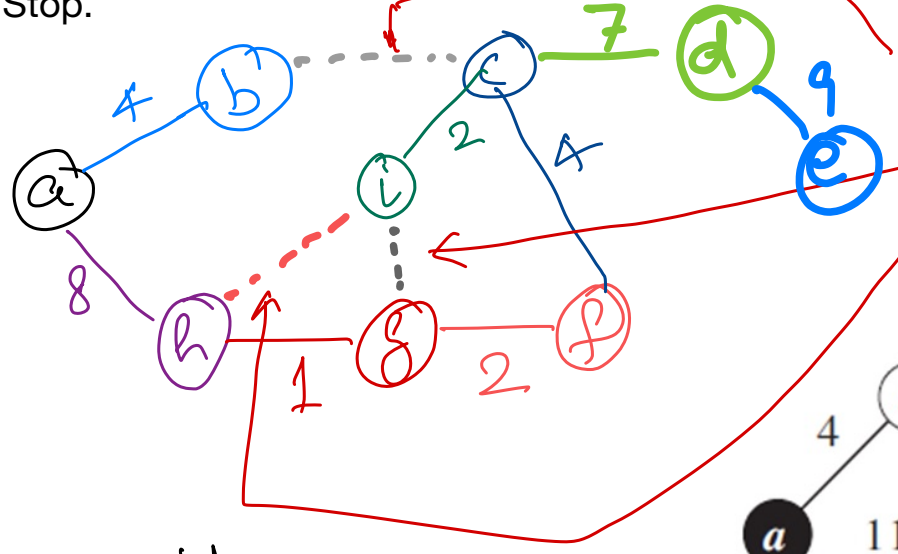


Min Spanning Tree => 4 + 8 + 2 + 1 + 2 + 4 + 7 + 9

= 37

Prim's Minimum Spanning Tree Algorithm

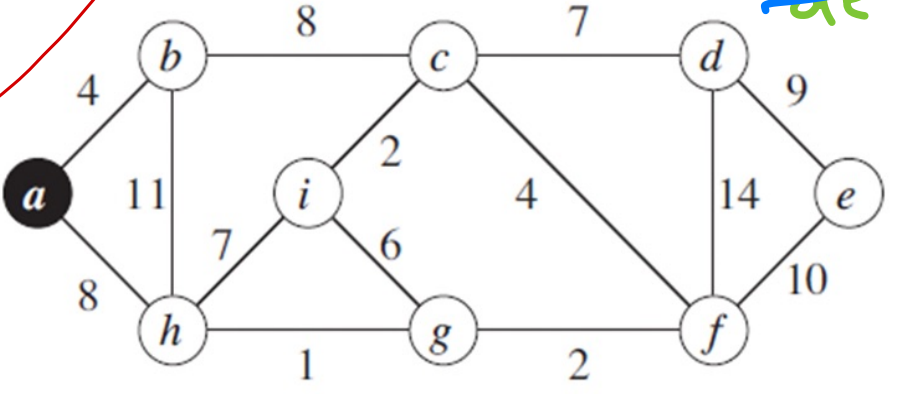
- Initialise min spanning tree with any vertex from graph.
- while min spanning tree do not have vertexCount number of vertex
  - Get all edges in the graph that connect the tree 2 newly added
  - Add the edge with min weight to the tree, if no cycle is formed.
- Stop.



forms cycle

ab 4 gi 6
ah 8 gf 2
bc 8 fc 4
bh 11 fd 14
ic 7 fe 10
hg 1 cd 7
ci 2
de 9

weight = 37



Kruskal

⇓

Sparse graph

⇓

with fewer edges

Prim

⇓

Dense graph

⇓

graph with lots of edges.