[CS203] **Design and Analysis of Algorithms**

Course Instructor: Dr. Dibyendu Roy — Autumn 2023-24

Scribed by: Prakhar Jain(202251099) — Lecture (Week 10)

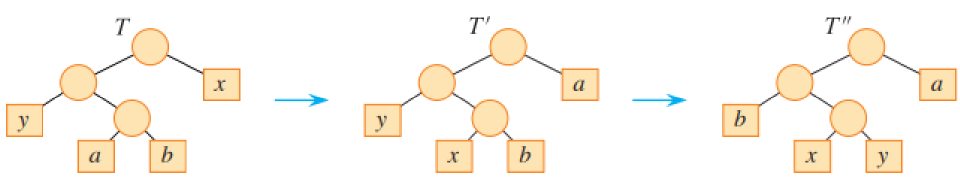
Lemma 1: Order of Characters in the Optimal Tree

Statement:

- In an optimal tree, the character with the lowest frequency will be at the leaf position farthest from the root.
- Characters x and y must differ in position by exactly one bit.

Proof:

1. Let T be an optimal tree and a be the character with the lowest frequency.
2. Assume, without loss of generality, that x and y are characters in T and $x.\text{freq} \leq y.\text{freq}$.
3. Consider T', the tree obtained by deleting x and y from T.
4. The new tree T' will be optimal, as characters with the lowest frequency are removed.
5. Since a has the lowest frequency, it will be placed at the least position in T'.
6. Now, introduce x and y back into T' with their frequencies unchanged.
7. T' with x and y follows the same encoding order as T due to the optimal structure.
8. Thus, x and y must differ in position by exactly one bit.



Lemma 2: Adjusting Frequencies for Optimality

Statement:

- If we set $z.\text{freq} = x.\text{freq} + y.\text{freq}$, the resulting tree is also optimal.

Proof:

1. Consider an optimal tree T with characters x and y.
2. Let T' be the tree obtained by setting $z.\text{freq} = x.\text{freq} + y.\text{freq}$.

3. Express the cost relation:

$$B(T) = B(T') - (x.\text{freq} + y.\text{freq})$$

4. Since x.freq and y.freq are non-negative, the additional term is non-negative.
5. Therefore, $B(T')$ is less than $B(T)$.
6. Thus, T' is an optimal tree.

Graph Representation

You can represent a graph $G = (V, E)$ either as an adjacency list or as an adjacency matrix.

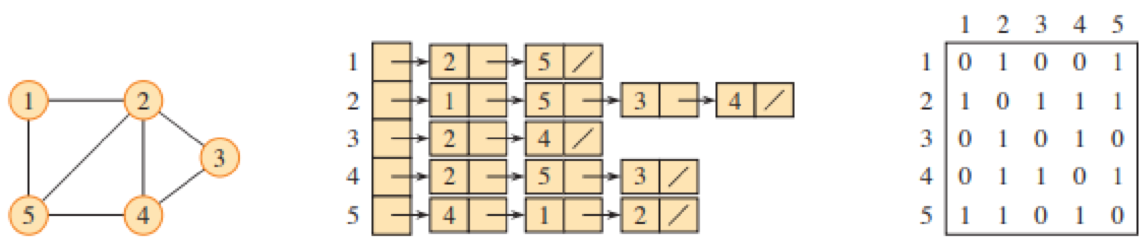


Figure 1: Two representations of an undirected graph. (a) An undirected graph G with 5 vertices and 7 edges. (b) An adjacency-list representation of G. (c) The adjacency-matrix representation of G.

Graph Illustrations

Undirected Graph:

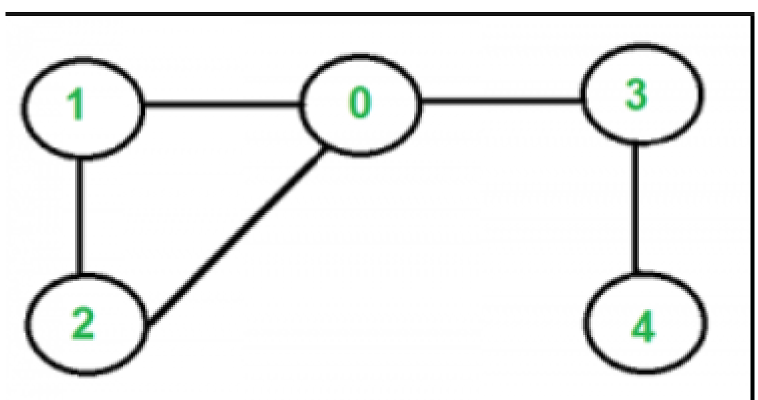


Figure 2: Undirected Graph

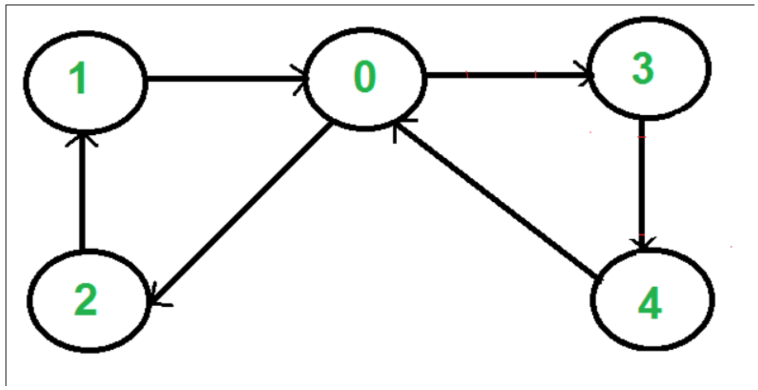


Figure 3: Directed Graph

Directed Graph:

Graph Theory

You can choose between two standard ways to represent a graph $G = (V, E)$: as a collection of adjacency lists or as an adjacency matrix. Either way applies to both directed and undirected graphs. Because the adjacency-list representation provides a compact way to represent sparse graphs—those for which $|E|$ is much less than $|V|$—it is usually the method of choice. Most of the graph algorithms presented in this book assume that an input graph is represented in adjacency-list form. You might prefer an adjacency-matrix representation, however, when the graph is dense—$|E|$ is close to $|V|^2$—or when you need to be able to tell quickly whether there is an edge connecting two given vertices. For example, two of the 550 Chapter 20 Elementary Graph Algorithms all-pairs shortest-paths algorithms presented in Chapter 23 assume that their input graphs are represented by adjacency matrices.

The adjacency-matrix representation of a graph $G = (V, E)$ assumes that the vertices are numbered $1, 2, \ldots, |V|$ in some arbitrary manner. The adjacency-matrix representation of graph G consists of a $|V| \times |V|$ matrix $A = (a_{ij})$ such that

$$a_{ij} = \begin{cases} 1 & \text{if } (i, j) \in E, \\ 0 & \text{otherwise.} \end{cases}$$

Weighted Graph:

Weighted edges we want to consider shortest path

If there is no edge then we put infinite there

Graph Properties

In the context of a graph, several properties characterize the behavior and relationships of vertices. Here, we discuss the source vertex, vertex color, and distance.

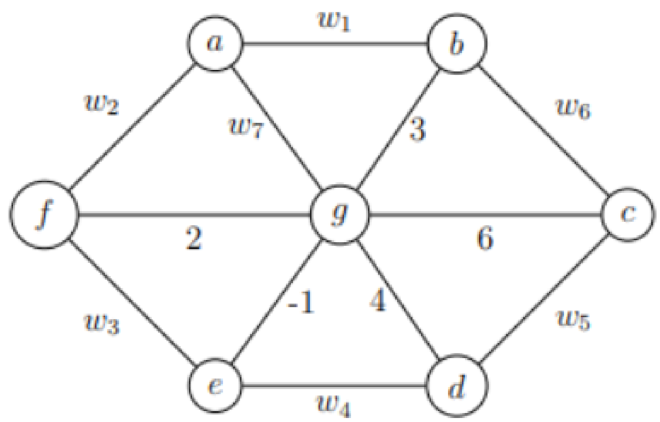
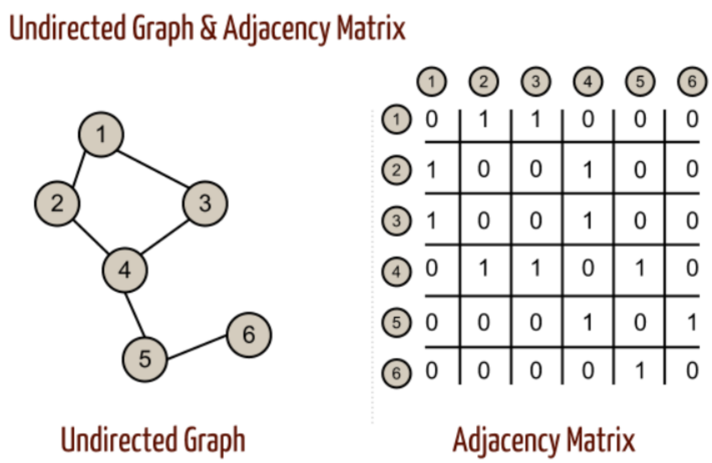


Figure 4: weighted graph



Source Vertex:

The source vertex, often denoted as s, is a central element in graph analysis. It serves as the starting point for various algorithms and measurements.

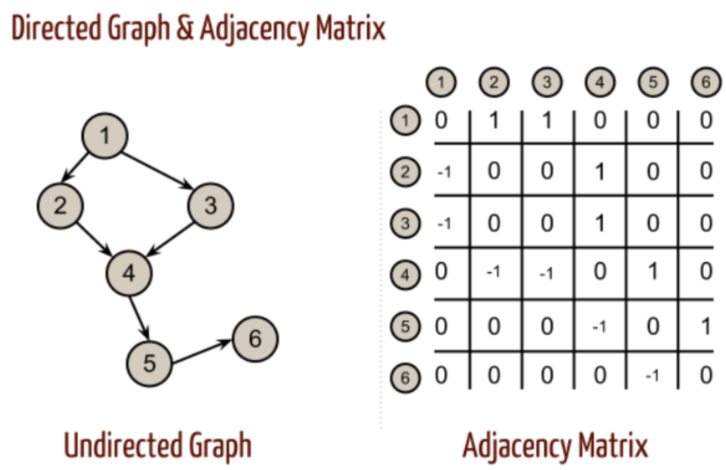
Vertex Color:

Vertices in a graph can be categorized into different colors, typically represented as white, grey, and black. These colors signify different states during the execution of graph algorithms.

- **White:** Vertices that have not been discovered or explored.
- **Grey:** Vertices that are discovered but not yet fully explored.
- **Black:** Vertices that are fully explored.

Distance from Source:

The distance from the source vertex s to another vertex v, denoted as $d(s, v)$, is defined as the minimum number of edges in the shortest path from s to v.



Predecessor and Parent:

In the context of traversing a graph, a predecessor (or parent) of a vertex u is the vertex from which u is reached in the traversal. For vertex v, if v has a predecessor, it is denoted as $p(v)$.

Vertex Colors and Distances

During the exploration of a graph, vertices are assigned different colors to indicate their exploration status. Additionally, distances between vertices play a crucial role in graph traversal algorithms. Here, we discuss the color and distance aspects, and touch upon the concept of minimum distances.

Vertex Colors:

- **White:** Vertices that have not been explored or touched.
- **Grey:** Vertices that are currently being explored or touched.
- **Black:** Vertices that have been fully explored or searched.

Distances:

The distance between two vertices, denoted as $d(u, v)$, is the length of the shortest path from vertex u to vertex v in the graph.

Minimum Distance:

For a source vertex s and a destination vertex v, the minimum distance from s to v, denoted as $d(s, v)$, is the smallest among all possible distances.

Example:

If $d(S, V) = 6$, it means there exists a path from vertex S to vertex V with a minimum distance of 6. For example, $d(5, 4) = 11$ indicates the minimum distance from vertex 5 to vertex 4 is 11.

Algorithm 1 Breadth-First Search

```
procedure BFS(G = (V, E), s)
    for each vertex u ∈ G : V do
        u.color ← WHITE
        u.d ← ∞
        u.π ← NIL
    end for
    s.color ← GRAY
    s.d ← 0
    s.π ← NIL
    Q ← empty queue
    Enqueue(Q, s)
    while Q is not empty do
        u ← Dequeue(Q)
        for each vertex v ∈ G.Adj(u) do
            if v.color = WHITE then
                v.color ← GRAY
                v.d ← u.d + 1
                v.π ← u
                Enqueue(Q, v)
            end if
        end for
        u.color ← BLACK
    end while
end procedure
```

Depth-First Search (DFS)

DFS is another graph traversal algorithm that explores as far as possible along each branch before backtracking. While DFS does not explicitly use distances, it plays a vital role in discovering the structure of the graph.