[CS203] Design and Analysis of Algorithms

Course Instructor: Dr. Dibyendu Roy
Scribed by: Prakhar Jain (202251099)

Autumn 2023-24
Lecture (Weak 11)

1 Breadth-First Search (BFS) Traversal

BFS is an algorithm used to explore a graph systematically, starting from a designated source vertex. The key idea is to visit all vertices at the current level before moving on to the next level. This ensures that the algorithm explores the graph in layers, resembling the structure of a tree.

1.1 Initialization

- Begin with a source vertex and initialize a queue, often denoted as Q.
- Mark the source vertex as visited.

1.2 Exploration

Repeat the following steps until the queue is empty:

while Q *is not empty* **do**

Dequeue a vertex from Q.

Visit all unvisited neighbors of the dequeued vertex.

Enqueue each unvisited neighbor and mark it as visited.

=0

1.3 Queue Representation

The queue Q reflects the order in which vertices are discovered.

- Initially, Q contains only the source vertex.
- As the algorithm progresses, vertices are enqueued and dequeued, mirroring the exploration process.

1.4 BFS Tree

The result of the BFS traversal is often represented as a tree, called the BFS tree.

- Each level of the tree corresponds to a layer of vertices discovered in the graph.
- The tree is rooted at the source vertex, and the edges represent the exploration order.

1.5 Termination

BFS terminates when every vertex is visited, ensuring that all vertices are reachable from the source.

- Optionally, termination can occur when the queue is empty or when a specific condition is met.

1.6 Diagram

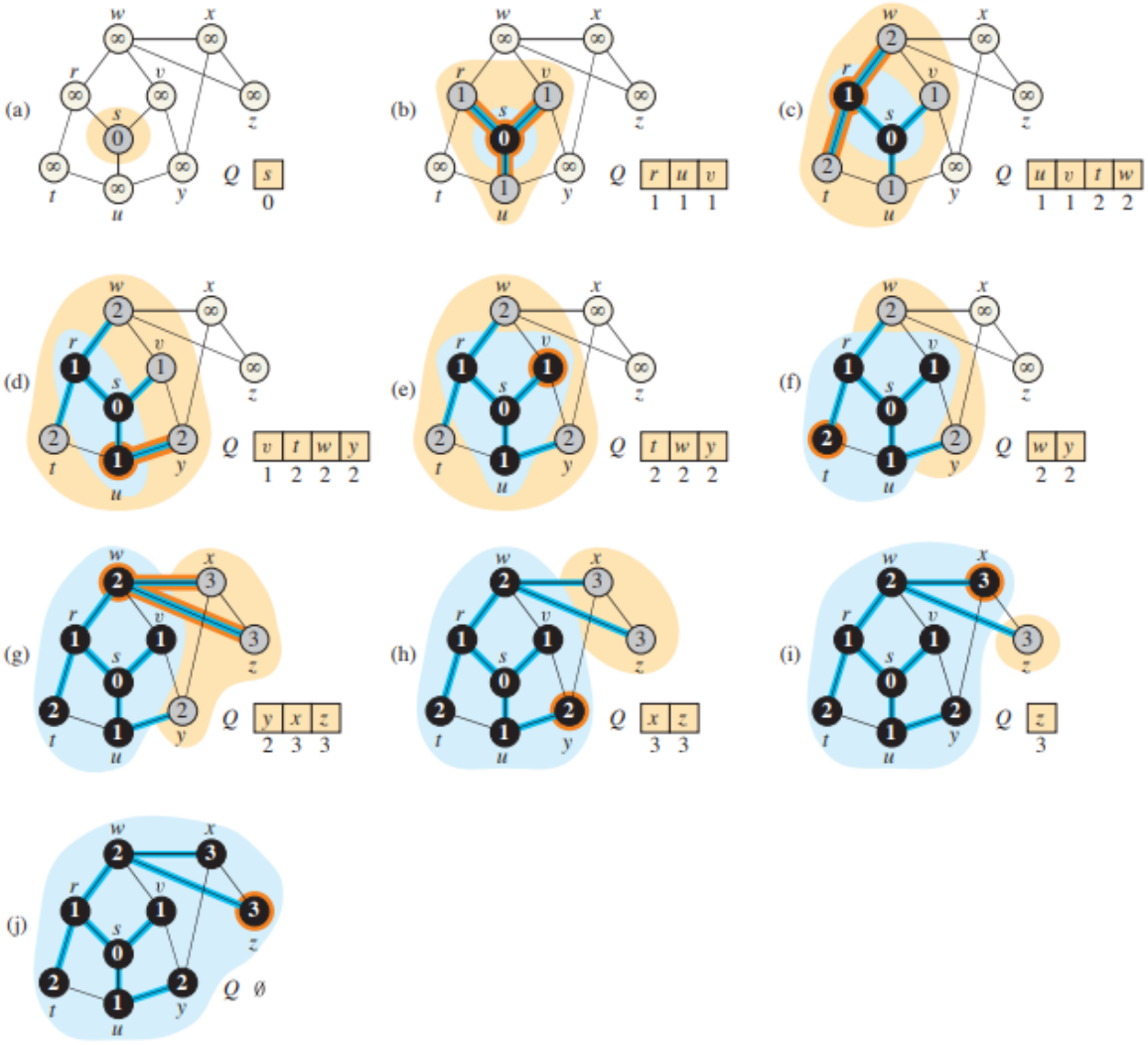

Figure 1: The operation of BFS on an undirected graph.

2 Shortest Paths

In graph theory, determining the shortest path between two vertices is a fundamental problem with various applications, such as network routing and optimization. The shortest-path distance from a source vertex s to a destination vertex v is defined as the minimum number of edges in any path from s to v. If there is no direct path from s to v, the shortest-path distance is conventionally set to ∞.

A path with the minimum number of edges from s to v is known as a shortest path. The concept of shortest paths is essential in understanding and optimizing network structures.

2.1 Bounding Shortest-Path Distances from Above

- **Statement:** For any directed or undirected graph $G = (V, E)$ and an arbitrary vertex $s \in V$, the shortest-path distance from s to any vertex v is bounded from above by the shortest-path distance from s to any other vertex u plus one, for any edge $(u, v) \in E$.
- **Proof:**
  - If u is reachable from s, then so is v. The shortest path from s to v cannot be longer than the shortest path from s to u followed by the edge (u, v).
  - If u is not reachable from s, then $d(s, u) = \infty$, and again, the inequality holds.

2.2 Proof of Lemma: $v.d \leq d(s, v)$

Let $G = (V, E)$ be a directed or undirected graph, and suppose that BFS is run on G from a given source vertex $s \in V$. Then, for each vertex $v \in V$, the value $v.d$ computed by BFS satisfies $v.d \leq d(s, v)$ at all times, including at termination.

Proof: The lemma is true intuitively because any finite value assigned to $v.d$ equals the number of edges on some path from s to v. The formal proof is by induction on the number of ENQUEUE operations. The inductive hypothesis is that $v.d \leq d(s, v)$ for all $v \in V$.

Base Case: The inductive hypothesis holds here because $s.d = 0 = d(s, s)$, and $v.d = 1 \leq d(s, v)$ for all $v \in V \setminus \{s\}$.

Inductive Step: Consider a white vertex v that is discovered during the search from a vertex u. The inductive hypothesis implies that $u.d \leq d(s, u)$

$$v.d = u.d + 1 \leq d(s, u) + 1 \leq d(s, v).$$

Vertex v is then enqueued, and it is never enqueued again because it is also grayed.Thus, the value of $v.d$ never changes again, and the inductive hypothesis is maintained.

To prove that $v.d = d(s, v)$, we first show more precisely how the queue Q operates during the course of BFS. The next lemma shows that at all times, the d values of vertices in the queue either are all the same or form a sequence $\langle k, k, \ldots, k, k + 1, k + 1, \ldots, k + 1\rangle$ for some integer $k \geq 0$.

2.3 Proof of Lemma: $v_r.d \leq v_1.d + 1$ and $v_i.d \leq v_{i+1}.d$ for $i = 1, 2, \ldots, r - 1$

Lemma: Suppose that during the execution of BFS on a graph $G = (V, E)$, the queue Q contains the vertices $v_1, v_2, \ldots, v_r$, where v_1 is the head of Q and v_r is the tail. Then, $v_r.d \leq v_1.d + 1$ and $v_i.d \leq v_{i+1}.d$ for $i = 1, 2, \ldots, r - 1$.

Proof: The proof is by induction on the number of queue operations.

Base Case: Initially, when the queue contains only s, the lemma trivially holds.

Inductive Step: We must prove that the lemma holds after both dequeuing and enqueuing a vertex.

1. Dequeuing: Let v_1 be dequeued, and v_2 becomes the new head. By the inductive hypothesis, $v_1.d \leq v_2.d$. Thus, $v_r.d \leq v_1.d + 1$ and $v_i.d \leq v_{i+1}.d$ for $i = 1, 2, \ldots, r-1$.

2. Enqueuing: When line 17 of BFS enqueues a vertex v onto a queue containing vertices $v_1, v_2, \ldots, v_r$, the enqueued vertex becomes v_{r+1}.

- If the queue was empty before v was enqueued, then after enqueuing v, we have $r = 1$, and the lemma trivially holds.
- Now suppose that the queue was nonempty when v was enqueued. At that time, the procedure most recently removed vertex u from the queue. Before removal, $u = v_1$ and the inductive hypothesis held, so that $u.d \leq v_2.d$ and $v_r.d \leq u.d + 1$.
- After u is removed from the queue, the vertex that had been v_2 becomes the new head v_1 of the queue, so that now $u.d \leq v_1.d$.
- Thus, $v_{r+1}.d = v.d = u.d + 1 \leq v_1.d + 1$, and $v_i.d \leq v_{i+1}.d$ for $i = 1, 2, \ldots, r$.

Therefore, the lemma follows when v is enqueued.

Conclusion: By induction, the lemma holds for all queue operations during the execution of BFS.

2.4 Theorem 20.5: Correctness of Breadth-First Search

Theorem: Let $G = (V, E)$ be a directed or undirected graph, and suppose that BFS is run on G from a given source vertex $s \in V$. During its execution, BFS discovers every vertex $v \in V$ that is reachable from the source s, and upon termination, $v.d = d(s, v)$ for all $v \in V$. Moreover, for any vertex $v \neq s$ that is reachable from s, one of the shortest paths from s to v is a shortest path from s to $v.\pi$ followed by the edge $(v.\pi, v)$.

Proof: Assume for contradiction that some vertex receives a d value not equal to its shortest-path distance. Let v be a vertex with the minimum $d(s, v)$. By Lemma 20.2, $v.d \neq d(s, v)$. Since $v \neq s$, v must be reachable from s, and let u be the vertex immediately preceding v on some shortest path from s to v. This leads to a contradiction as follows:

$$v.d > d(s, v) = d(s, u) + 1 = u.d + 1.$$

Consider the time when BFS dequeues vertex u from the queue in line 11. Whether v is white, gray, or black leads to a contradiction of the inequality $v.d > u.d + 1$. Therefore, $v.d = d(s, v)$ for all $v \in V$. All reachable vertices are discovered, and the theorem is proved.

3 Depth-First Search (DFS) Algorithm

Algorithm 2: DFS-VISIT Algorithm

```
   Input  : Graph G = (V, E)
   Output: Depth-First Search of G
 1 for each vertex u ∈ G.V do
 2 |   u.color ← WHITE;
 3 |   u.π ← NIL;
 4 end
 5 time ← 0;
 6 for each vertex u ∈ G.V do
 7 |   if u.color == WHITE then
 8 |   |   DFS-VISIT(G, u);
 9 |   end
10 end
11 DFS-VISIT G, u
12 |   time ← time + 1;
13 |   u.d ← time;
14 |   u.color ← GRAY;
15 |   for each vertex v ∈ G.Adj[u] do
16 |   |   if v.color == WHITE then
17 |   |   |   v.π ← u;
18 |   |   |   DFS-VISIT G, v
19 |   |   |   |   ;
20 |   |   end
21 |   end
22 |   time ← time + 1;
23 |   u.f ← time;
24 |   u.color ← BLACK;
```

4 Topological Sorting

A *topological sort* of a directed acyclic graph (DAG) $G = (V, E)$ is a linear ordering of all its vertices such that if G contains an edge (u, v), then u appears before v in the ordering.

4.1 Diagram

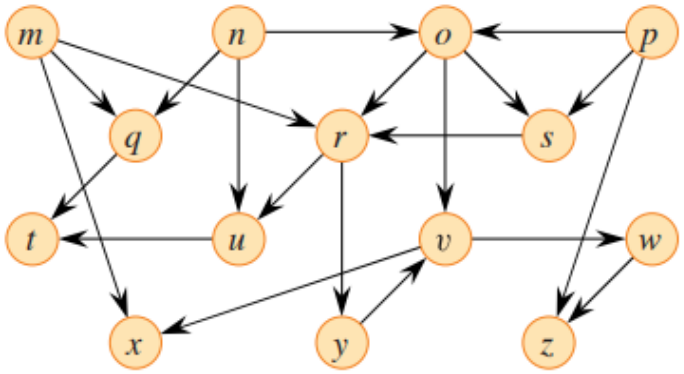

Figure 2: A dag for topological sorting.

5 Minimum Spanning Tree (MST)

A *Minimum Spanning Tree (MST)* of a connected graph $G = (V, E)$ with edge weights $w : E \to \mathbb{R}$ is a tree $T = (V_T, E_T)$ such that:

- V_T is a subset of V, and E_T is a subset of E.
- T is connected.
- T is acyclic.
- V_T spans V, i.e., V_T includes all vertices of V.
- The sum of the weights of the edges in E_T is minimized.

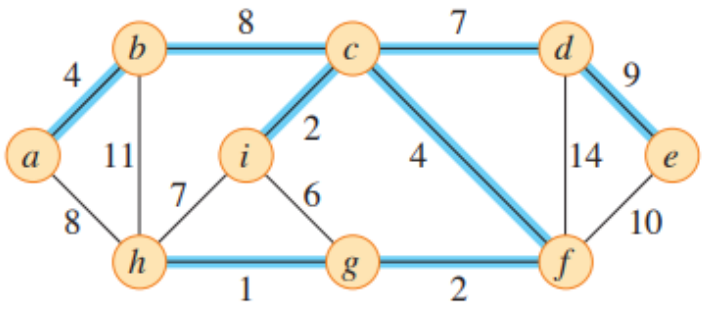

Figure 3: A minimum spanning tree for a connected graph.

5.1 Algorithm: Kruskal's Algorithm

1. For a graph $G = (V, E, w)$, where w represents edge weights.
2. Sort all the edges in non-decreasing order of their weights.

3. Initialize an empty graph $T = (V, \emptyset)$.

4. Iterate through the sorted edges. Add an edge to T if adding it does not form a cycle.

5. Continue until T forms a spanning tree.

5.2 Safe Edge

An edge is considered *safe* for a set of vertices S if it is the minimum-weight edge that connects a vertex in S to a vertex outside S.

5.3 Cut Property

A *cut* of a graph $G = (V, E)$ is a partition of V into two disjoint sets S and $V - S$. An edge $e = (u, v)$ *crosses* the cut $(S, V - S)$ if u is in S and v is in $V - S$.

A spanning tree T *respects* a cut $(S, V - S)$ if no edge in T crosses the cut.