Data Structures and Algorithms

Lecture 22

Aniket Basu Roy

2026-03-20 Fri

Contents

1 Agenda

1.1 Graphs

- k-Clique problem for proximity graphs

- Graph Traversals
- Breadth-First Search (BFS)
- Depth-First Search (DFS)

2 k-Clique Problem

- Input: A simple undirected graph $G = (V, E)$
- Question: Does there exist a clique of size k in G?

Q. What is the time complexity of the k-Clique problem for

- unit disk graphs
- unit interval graphs

Hint: A linear scan is useful for the 1D problem. For the $2n$ critical points (the two end points per interval) maintain a depth vector that denotes the number of intervals that contain critical point i.

- Takeaway: Given the problem we want to solve (e.g., k-Clique problem) and special class of input graph (e.g., unit interval graphs), it may be more useful to avoid the generic data structures (e.g., adjacency list/matrix) to represent the input graph, and use something more specialized (e.g., depth vector).

3 Graph Traversals

- Systematically visit every vertex and edge of a graph
- Two fundamental strategies: BFS and DFS
- Both run in $\Theta(V + E)$ time (on adjacency list representation)
- Basis for almost all graph algorithms

4 Breadth-First Search (BFS)

4.1 The Idea

Given a source vertex s, explore vertices in order of their **distance** from s.

- First visit all vertices at distance 1 from s
- Then all vertices at distance 2
- And so on — like ripples spreading outward

4.2 Data Structure: Queue (FIFO)

- Vertices to be explored are kept in a queue
- ENQUEUE: add to back; DEQUEUE: remove from front
- Ensures vertices are processed in non-decreasing order of distance from s

5 Depth-First Search (DFS)

5.1 The Idea

Explore as **deep** as possible before backtracking.

- From a vertex u, recursively visit an unvisited neighbor v
- When no unvisited neighbors remain, backtrack to the parent
- Continue until all vertices are visited

5.2 Data Structure: Stack (LIFO)

- Recursive implementation uses the **call stack** implicitly
- Iterative version uses an explicit stack
- Unlike BFS: does NOT necessarily find shortest paths

5.3 Timestamps

DFS records two timestamps per vertex (CLRS):

- $u.d$: **discovery time** — when u is first seen (grayed)
- $u.f$: **finish time** — when u's adjacency list is fully explored (blackened)
- $1 \leq u.d < u.f \leq 2|V|$

6 Applications of BFS and DFS

6.1 Shortest Paths in Unweighted Graphs

BFS from s computes $v.d = \delta(s, v)$ for all vertices v reachable from s.

- Works because BFS explores vertices in non-decreasing order of distance.
- The BFS tree gives the actual shortest paths (trace back via $v.\pi$).
- **Limitation**: only works for unweighted (or unit-weight) graphs.

6.2 Connected Components

To find all connected components of an undirected graph:

```
CONNECTED-COMPONENTS(G)
1.  for each vertex u in G.V
2.      u.component = NIL
3.  comp = 0
4.  for each vertex u in G.V
5.      if u.component == NIL
6.          comp = comp + 1
7.          BFS(G, u)        // or DFS
8.          // mark all visited vertices with comp
```

Time: $\Theta(V + E)$.

6.3 Bipartiteness Testing

A graph G is **bipartite** iff it contains no odd-length cycle.

```
IS-BIPARTITE(G)
1.  for each vertex u in G.V
2.      u.color = WHITE
3.  for each vertex s in G.V
4.      if s.color == WHITE
5.          s.color = RED
6.          ENQUEUE(Q, s)
7.          while Q not empty
8.              u = DEQUEUE(Q)
9.              for each v in G.Adj[u]
10.                 if v.color == WHITE
11.                     v.color = opposite(u.color)
12.                     ENQUEUE(Q, v)
13.                 elseif v.color == u.color
14.                     return FALSE    // odd cycle found
15. return TRUE
```

BFS 2-colors the graph (RED/BLUE). If a conflict arises, an odd cycle exists.

6.4 Cycle Detection

A directed graph G has a cycle iff DFS finds a **back edge**.

Proof sketch:

- ($\Rightarrow$) If G has a cycle $v_0 \to v_1 \to \cdots \to v_k \to v_0$, then DFS will discover v_0 before v_k finishes, so (v_k, v_0) is a back edge.
- ($\Leftarrow$) A back edge (u, v) with v an ancestor of u directly gives a cycle.

For undirected graphs: a back edge also reveals a cycle (any non-tree edge is a back edge in undirected DFS).