DSA Mini Textbook

Theo Park

Contents

Preface

Chapter 1

Runtime Analysis

Chapter 2

Intro to Data Structures

Chapter 3

Sorting Algorithms

Chapter 4

Hash Tables

4.1 Division Method

4.2 Multiplication Method

4.3 Collision

4.3.1 Chaining

4.3.2 Open Addressing

Chapter 5

Search Tree

5.1 Binary Search Tree and Its Limit

5.2 2-3 Tree

5.3 Red-Black Tree

5.4 Left-Leaning Red-Black Tree

5.4.1 Deletion in LLRBT

Chapter 6

Undirected Graph

Graph is a set of vertices V and a collection of edges E that connect a pair of vertices. *Undirected graph* is a graph where edges do not have direction. *Degree of a vertex* representing how many edges is this vertex connected to.

- **Handshaking Theorem**: For any undirected graphs, $\sum_{v \text{ in } V} deg(v) = 2 \cdot |E|$
- Maximum degree of a vertex: $deg(v) \leq |V| - 1$, because a vertex cane be connected to all other vertices at most
- Maximum edge count: $|E| \leq \frac{|V|(|V|-1)}{2}$
- ***Complete graph***: A graph is said to be complete when each vertex pair is connected by a unique edge. Id est, a complete graph has the maximum number of edges ($|E| = \frac{|V|(|V|-1)}{2}$) and each vertex has the maximum degree ($deg(v) = |V| - 1$)

6.1 Adjacency Matrix and List

6.2 DFS

```
function DEPTH-FIRST-SEARCH(G, s)        ▷ G is a graph containing |V| vertices, s is the starting node
    S ← new stack
    Push s to S

    while S is not empty do
        u ← popped element from S
        if u is not visited then
            for w adjacent to v do
                if w is not visited then
                    Push w to S
```

6.3 BFS

```
function DEPTH-FIRST-SEARCH(G, s)        ▷ G is a graph containing |V| vertices, s is the starting node
    Q is a new queue
    Enqueue s to Q
```

```
    Mark $v$ as visited
    **while** $Q$ is not empty **do**
        $u \leftarrow$ dequeued item from $Q$
        **for** $w$ adjacent to $u$ **do**
            **if** $w$ is not visited **then**
                Enqueue $w$ to $Q$
                Mark $w$ as visited
```

Chapter 7

Directed Graphs

7.1 Strong Connectivity

7.1.1 Brute-force Strong Connectivity Algorithm

7.1.2 Brute-force using Stack

7.1.3 Strongly Connected Components and Kosaraju's Algorithm

7.2 Directed Acyclic Graphs

7.2.1 Topological Sort

Chapter 8

Weighted Graphs

8.1 Shortest Path

8.1.1 Dijkstra's Algorithm

```
function DIJKSTRA-SHORTEST-PATH(G, s)    ▷ G is a graph containing |V| vertices, s is the starting node
    dist ← array size |V|
    prev ← array size |V|
    Q ← a new min-heap with distance values as keys

    ▷ Initialization step ◁
    for v in Vertices do
        if v = s then dist[v] ← 0
        if v ≠ s then dist[v] ← ∞
        prev[u] ← −1
        Add a tuple (dist[v], v) to Q
        while Q is not empty do
            u ← the value with minimum dist from Q    ▷ O(1)
```

8.1.2 Bellman-Ford Algorithm

Dijkstra should not be used on a graph with negative edge(s).

```
function BELLMAN-FORD-SHORTEST-PATH(G, V)    ▷ G is the graph, V is the vertex list
    dist ← array size |V|
    prev ← array size |V|
    for v in V do
        if v = s then dist[v] ← 0
        if v ≠ s then dist[v] ← ∞
        prev[u] ← −1
    for i = 1 to |V| − 1 do
        for e in E do    ▷ Edge e connects vertex u and v
            if weight[e] + dist[u] < dist[v] then
                dist[v] = dist[u] + weight[e]
                prev[v] = u
    ▷ Run the for loop once again. If the shortest distance is updated, then it means there is a negative weight cycle ◁
```

```
for e in E do                                   ▷ Edge e connects vertex u and v
    if weight[e] + dist[u] < dist[v] then
        output Negative weight edge cycle detected
        return
```

8.2 Articulation Points

An articulation point is a vertex such that removing it from the graph increases the number of connected components.

```
function (G, s, d)          ▷ G is the graph, s is the starting vertex, d is the //TODO
    Mark s as visited
    discovery[s] ← d
    low[s] ← d
```

8.3 Minimum Spanning Tree

- Minimum: $\sum weight$ is minimum
- Spanning: All vertices in the graph are connected
- Tree: No cycle

There are two fundamental properties of MST:

1. *Cycle Property*: For any cycle C in the graph, if the weight of an edge $e \in C$ is higher than any of individual weights of all other edges in C, then its edge cannot belong in the MST.
2. *Cut Property*: For any *cut* (subdivision of graph with disjoint) C in the graph, if the weight of an edge e in the cut-set of C is strictly smaller than the weights of all other edges of the cut-set of C, then this edge belongs to all MST of the graph.

8.3.1 Cycle and Cut Properties

8.3.2 Prim's Algorithm

8.4 Union-Find

8.4.1 Kruskal MST Algorithm

Chapter 9

Strings

9.1 Brute-force String Pattern Matching

9.2 KMP Algorithm

9.3 Trie

9.4 PATRICIA

9.5 Huffman Coding