Manju Muralidharan Priya Fall 2024

CS210 Data Structures & Algorithms
Mock Exam - Extra Credit Assignment

Hash Tables

We have a hash table of size 10 (indexes 0 through 9). We will use two different hash functions on the same set of keys. Show the steps clearly for each key when using both hash functions, and then write down the final state of the hash table after all insertions are done.

Keys to Insert:
AI, cat, ZzZ, mY, Cube, Hello, ZZ, aaa, DoG

Hash Function 1 (Binary-ASCII Summation with Double Hashing)

1. **For each character in the key:**
    - Find its ASCII decimal value.
    - Sum all these ASCII values.
2. **Compute the hash index as:**

index = (sum of ASCII values) mod 10

3. **Collision Resolution (Double Hashing):**

If a collision occurs at the initial index h1, compute a secondary hash value:
h2 = 7 - (sum of ASCII values mod 7)
Then for each collision attempt i (starting from 1), try the next slot as:
(h1 + i * h2) mod 10
Continue until you find an empty slot.
Example Using Hash Function 1 with the key "Be":

- 'B' ASCII = 66, 'e' ASCII = 101
- Sum = 66 + 101 = 167
- Initial Index = 167 mod 10 = 7

If index 7 is free, place "Be" there. If not, use double hashing as described.
You will follow these steps for each of the ten keys and insert them into a hash table (initially empty).
Show the final table.

Hash Function 2 (Alphabetical Position Summation with Linear Probing)

1. **Assign each character a position based on the alphabet (case-insensitive):**

A/a = 1, B/b = 2, C/c = 3, …, Z/z = 26.

2. **Sum all positions of the letters in the key.**
3. **Compute the hash index as:**

index = (sum of all letter positions) mod 10

4. **Collision Resolution (Linear Probing):**

If the computed index is occupied, try (index + 1) mod 10, then (index + 2) mod 10, and so on, until you find an empty slot.

Example Using Hash Function 2 with the key "Be":

- 'B' = 2, 'e' = 5
- Sum = 2 + 5 = 7
- Index = 7 mod 10 = 7

If index 7 is free, place "Be" there; if not, try index 8, then 9, and loop back to 0 if needed.
Again, insert the same ten keys using this second hash function starting from an empty table. Show the steps and the final state of the hash table.

Your Task:

1. Insert all ten keys into the hash table using Hash Function 1 (with double hashing) and show the final table.
2. Then, starting from a fresh, empty hash table, insert all ten keys using Hash Function 2 (with linear probing) and show the final table.

Make sure to show the calculations for at least one or two keys in detail (like the given examples) and outline the steps for all other keys.

HASH FUNCTION 1

Index	Key,Value
0	Hello
1	aaa
2	cat
3	mY
4	ZzZ
5	Cube
6	ZZ
7	
8	AI
9	DoG

HASH FUNCTION 2

Index	Key,Value
0	AI
1	Cube
2	Hello
3	ZZ
4	cat
5	aaa
6	DoG
7	
8	ZzZ
9	mY

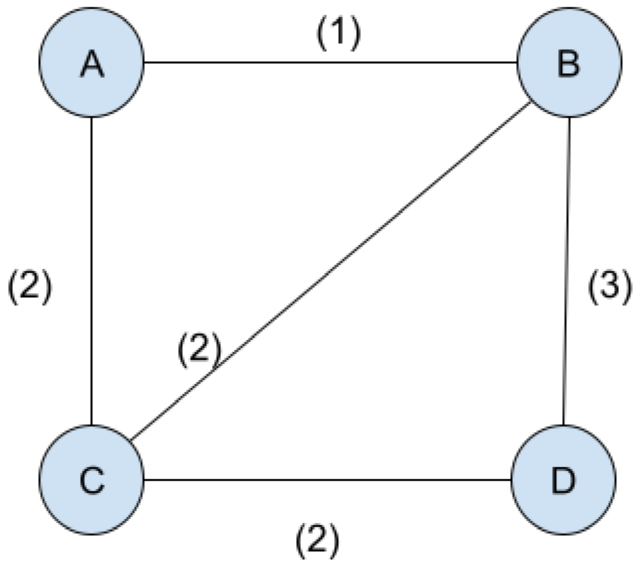
Minimum Spanning Trees

Consider the following weighted graph with four vertices: A, B, C, and D.
Vertices: A, B, C, D
Edges with Weights:

- A–B = 1
- A–C = 2
- B–C = 2
- C–D = 2
- B–D = 3

Important: There is **no** A–D edge.

Graph Structure



Your Tasks:

1. Construct the MST using Prim's Algorithm:
   - Start from vertex A.
   - Show the order in which edges are chosen and list the final MST.
2. Construct the MST using Kruskal's Algorithm:
   - Show the order in which edges are chosen and list the final MST.

After completing both methods, compare the MSTs you found. Do they contain the same edges, or are they different sets of edges with the same total weight?

Using Prim's Algorithm:

1. Start at A.
2. Add edge A–B (weight = 1).
3. Add edge A–C (weight = 2).
4. Add edge C–D (weight = 2).

Final MST (Prim's): Edges: {A–B, A–C, C–D}
Total Weight: 5

Using Kruskal's Algorithm:

1. Sort edges: A–B (1), A–C (2), B–C (2), C–D (2), B–D (3).
2. Add edge A–B (weight = 1).
3. Add edge A–C (weight = 2).
4. Add edge C–D (weight = 2).

Final MST (Kruskal's): Edges: {A–B, A–C, C–D}
Total Weight: 5

Comparison: Both algorithms yield the same MST with the same total weight.

BFS & DFS

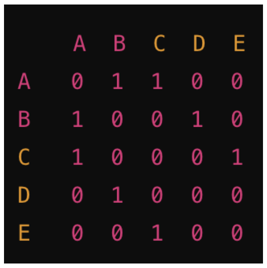
Given the undirected graph below, apply both Breadth-First Search (BFS) and Depth-First Search (DFS) starting from vertex A. For each algorithm, list the order in which the vertices are visited.
Graph:

- Vertices: A, B, C, D, E
- Edges: (A–B), (A–C), (B–D), (C–E)

Your Tasks:

1. Draw the graph's Adjacency matrix
2. Write down the order of vertices visited by BFS starting at A.
3. Write down the order of vertices visited by DFS starting at A.

Make sure to show the steps and the final order of visitation for each method.



	A	B	C	D	E
A	0	1	1	0	0
B	1	0	0	1	0
C	1	0	0	0	1
D	0	1	0	0	0
E	0	0	1	0	0

BFS Order: A → B → C → D → E
DFS Order: A → B → D → C → E

Dijkstra's

Consider the following directed, weighted graph with vertices A, B, C, D, and E:
Edges:

- A → B (weight 2)
- A → C (weight 5)
- B → C (weight 1)
- B → D (weight 3)
- C → D (weight 2)

Vertex E is isolated.

Your Tasks:

1. Draws the adjacency list of this graph.
2. Using Dijkstra's algorithm, find the shortest path distances from the start vertex A to all other vertices. Fill the following table and make sure to show all intermediate values

Adjacency List :

A: [(B, 2), (C, 5)]

B: [(C, 1), (D, 3)]

C: [(D, 2)]

D: []

E: []

Vertex	Distance	Predecessor
A	INF or 0	NIL
B	2	A
C	~~5~~ 3	~~A~~ B
D	5	B
E	INF	NIL