Lab 9 15 Oct

Q1 (35%)

Hashing with chaining

In this assignment we implement a simple hashing method and the resolution of collision via chaining.
Define a HashTable class with the following attributes:

`Table: an array or vector along with necessary pointers to linked lists (to implement chaining)`
`Collisions`: number of collisions
Methods:
`Insert(key) : Insert a key into the table. In case of collision, the key should be inserted into the corresponding linked list.`

`Search(key): Searches for key and returns True if the key is found, false otherwise.`

`Delete(key): Searches for the key and deletes if found.`

Key is assumed to be a nonnegative integer.

Hash function to be used: Let p be the smallest prime greater than 2000. Let a = 503
And b = 1087. Then h(x) = (ax + b) mod p.

Input Format
The first line of the input contains a single integer n, the number of operations. The following n lines,
each of which is one of the following:

- `Insert key`
- `Search key`
- `Delete key`

Output Format
For every Search operation, print TRUE if the key is found in the table, FALSE otherwise.
For Insert and Delete operations, do not print anything.
After processing all operations:

Print only those hash table slots that contain at least one key.
The slot indices should be printed in ascending order.
The keys within each slot should be printed in descending order, separated by a space.
slot_index key1 key2 key3 ...

Constraints:

- Number of operations, n ≤ 2000.
- 0 ≤ Keys ≤ 100000.

Bonus (2 marks, capped to 5%): Extend your implementation to handle strings as keys. You may choose your favorite encoding scheme for hashing (please mention the function as a comment in your code)

Sample Testcase:

Input
9
Insert 5
Insert 25
Insert 2008
Insert 10009
Search 25
Search 50
Delete 25
Insert 4020
Search 10009

Output
TRUE
FALSE
TRUE
72 10009
117 4020
1599 2008 5

Q2 (35%)

In this assignment we implement resolution of collision via different probing methods.
You can build on the HashTable class defined for Q1. Use the same hash function as in Q1

Methods:

`Insert(key) : Insert a key into the table.In case of a collision, the next location should be obtained based on the selected probing method`

`Search(key): Searches for key and returns True if the key is found, false otherwise.`

`NextLocationQUad(key,loc): Obtains the next location based on the quadratic probing method.`

`NextLocationDouble(key,loc): Obtains the next location based on the double hashing probing method where the function is h_1(x) = x mod 61 and h_2(x) = (x^2 +1) mod 61. (Double hashing as defined in the class: Suppose h_1(k) = i and A[i] is occupied. Then iteratively look for locations (i + f (j)) mod p where f (j) = j · h_2(k), for j = 1, 2, . . .`

`Delete(key): Searches for the key and deletes if found.`

Key is assumed to be a nonnegative integer.

Hash function to be used **for quadratic probing:** Let p be the smallest prime greater than 2000. Let a = 503 And b = 1087. Then h(x) = (ax + b) mod p.
For double hashing, using h_1 and h_2 as defined above.

Input Format

The first line of the input contains a single integer n, the number of operations and the second line contains the open addressing scheme to be used (`Quad` or `Double`) The following n lines,
each of which is one of the following:

- `Insert key`
- `Search key`
- `Delete key`

Output Format

For every Search operation, print TRUE if the key is found in the table, FALSE otherwise.
For Insert and Delete operations, do not print anything.
After processing all operations:

Print only those hash table slots that contain at least one key.
The slot indices should be printed in ascending order.
slot_index key

Constraints:

- Number of operations, n ≤ 2000.
- 0 ≤ Keys ≤ 100000.

Sample Testcase
Input:
7
Quad
Insert 1
Insert 2004
Search 2004
Delete 1
Search 2004
Insert 4007
Search 1

Output:
TRUE
TRUE
FALSE
1590 4007
1591 2004

Q3 (30%)

Implementation of Graphs. Adj list and Adj matrix. BFS.

You are given a set of cities along with the road network and a list of pairs of cities that have a direct railway line connecting between them. Given the number of cities and road connections among them, you are expected to find the following:

Given a city A , decide whether every other city can be visited from A using the given road network. You must build a graph data structure and the BFS method for this purpose.

Note: You may use the queue

Input Format

First line of input contains non negative integer n the number of cities and second line contains non negative integer m the number of direct roads between the cities. The third line contains city number A (number between 0 to n-1). Each of the next m lines denotes direct roads between pairs of cities.

Constraints

1<=n<=10^4
1<=m<=10^6

Output Format

output either prints “YES” if every other city can be visited starting from city A, else prints “NO”.

Method: You are expected to define a suitable graph class and implement the Breadth first search method. Using any other method for search is not acceptable.

Sample Testcase

Input:

5
4
0
0 1
0 2
0 3
2 4

Output

YES