Saurabh singh yadav
19095087
CSE-102
Electronics Engineering

1(a).
We can use Hashing to implement python Dictionary in C.
We can use ***Hashing with Direct Chaining.***

In hashing, large keys are converted into small keys by using ***hash functions***. The values are then stored in a data structure called **hash table**. The idea of hashing is to distribute entries (key/value pairs) uniformly across an array. Each element is assigned a key . By using that key you can access the element in O(1) time. Using the key, the hash function computes an index that suggests where an entry can be found or inserted.

Pseudo-Code
```
void hashing(key,value){
 arr[16]  //create array of required size let say
index=hash_function(key)   //→ will return an index less than 16
insert(index)        //→ will insert the value at that index one index can store more than one value

}

get_Value(key){
index=hash_function(key)   //→ will return an index less than 16
get(index)  //→ return value
}
```

1(b).
I think the book index present at the end book is a like Python Dictionary which can implement using hashing in C.
So according to me ***Hashing with direct chaining*** is the best way to do so, pseudo_code for which will be same as in 1(a).

2(a).
Dijkstra’s algorithm is an algorithm for finding the shortest paths between nodes in a graph.
We will need queue data structure to implement Dijkstra’s algorithm in C.

We overestimate the distance of each vertex from the starting vertex. Then we visit each node and its neighbors to find the shortest subpath to those neighbors.
The algorithm uses a greedy approach in the sense that we find the next best solution hoping that the end result is the best solution for the whole problem.

Pseudo-Code

```
 Dijkstra(G, S)  //here G is graph and S is Source
```

```
for each vertex V in G
    distance[V] = infinite
    previous[V] =NULL       //previous is maintained to get path
    If V != S, add V to Priority Queue Q
distance[S] =0

while Q IS ! EMPTY
    U       // Extract MIN from Q
    for each unvisited neighbour  V of U
        temp  // distance[U] + edge_weight(U, V)
        if temp < distance[V]
            distance[V] =temp
            previous[V] = U
return distance[], previous[]
```

2(b).
Yes, We can apply queue data structure in a search algorithm known as ***Breadth First Search***.
This algorithm is used for traversing graph
It start at some arbitrary node of a graph and explores the neighbours nodes (present at current level) first , before moving to the next level neighbours .
The data structure used in a Breadth-first algorithm is a queue. The algorithm makes sure that every node is visited at least once, and not twice. It makes sure that all the neighboring nodes for a particular node are visited, not more than once.

Pseudo-Code

```
BFS(G)
while all the matrices are not explores,do
enqueue(any vertex)
while Q is not empty
 p=Dequeue()
 if p is unvisited
print p and mark p as visited
 enqueue(all adjacent visited vertices of p)
```