29/02/24 Week-10 : Hashing

```
#include<stdio.h>
#include<stdlib.h>
#define MAX_EMPLOYEES 100
#define HASH_TABLE_SIZE 10
struct Employee {
    int key;
};

int hashFunction(int key);
void insertEmployee(struct Employee employees[],
int hashTable[], struct Employee emp);
void displayHashTable(int hashTable[]);

int main(){
    struct Employee employees[MAX_EMPLOYEES];
    int hashTable[HASH_TABLE_SIZE]={0};
    int n, m, i;
    printf("Enter the number of employees: ");
    scanf("%d", &n);
    printf("Enter employee records:\n");
    for(i=0; i<n; i++){
        printf("Employee %d:\n", i+1);
        printf("Enter 4-digit key: ");
        scanf("%d", &employees[i].key);
        insertEmployee(employees, hashTable,
        employees[i]);
    }
printf("\nHash Table:\n");
displayHashTable(hashTable);
return 0;
}
int hashFunction(int key){
    return key % HASH_TABLE_SIZE;
}
```

```
void insertEmployee (struct Employee employees[],
int hashTable[], struct Employee emp) {
  int index = hashFunction (emp.key);
  while (hashTable[index] != 0) {
    index = (index + 1) % HASH_TABLE_SIZE;
  }
  hashTable[index] = emp.key;
}
void displayHashTable (int hashTable[]) {
  int i;
  for (i=0; i<HASH_TABLE_SIZE; ++i) {
    printf ("%d -> ", i);
    if (hashTable[i] == 0) {
      printf ("Empty\n");
    } else {
      printf ("%d\n", hashTable[i]);
    }
  }
}
```

Output:

```
Enter the number of employees : 10
Enter the employee records:
Employee 1 :
Enter 4-digit key 1234
Employee 2
Enter 4-digit key : 3456
Employee 3
Enter 4-digit key : 2675
Employee 4
Enter 4-digit key : 2789
Employee 5
Enter 4 digit key : 1254
```

Employee 6
Enter 4-digit key : 8970
Employee 9
Enter 4-digit key : 4578
Employee 8
Enter 4-digit key : 3241
Employee 9
Enter 4-digit key : 9875
Employee 10
Enter 4-digit key : 2456

Hash table
0 → 8970
1 → 3241
2 → 9875
3 → 2456
4 → 1234
5 → 2675
6 → 3456
7 → 1254
8 → 4578
9 → 2789

8/2/24
29/2/24

Hackerrank

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
  int id;
  int depth;
  struct node *left, *right;
};
void inorder(struct node *tree)
{
  if (tree == NULL)
    return;
  inorder(tree->left);
  printf("%d", tree->id);
  inorder((tree->right));
}
int main(void)
{
  int no_of_nodes, i = 0;
  int l, r, max_depth, k;
  struct node *temp = NULL;
  scanf("%d", &no_of_nodes);
  struct node *tree = (struct node *) calloc
  (no_of_nodes, sizeof(struct node));
  tree[0].depth = 1;
  while (i < no_of_nodes)
  {
    tree[i].id = i+1;
    scanf("%d %d", &l, &r);
    if (l == -1)
      tree[i].left = NULL;
    else
    {
      tree[i].left = &tree[l-1];
```

```
tree[i].right -> depth = tree[i].depth + 1;
max_depth = tree[i].right -> depth + 2;
}
i++;
}
scanf("%d", &i);
while(i--)
{
scanf("%d", &k);
x = k;
while(k <= max_depth)
{
for(k=0; k<no_of_nodes; ++k)
{
if(tree[k].depth == k)
{
temp = tree[k].left;
tree[k].left = tree[k].right;
tree[k].right = temp;
}
}
k = k + x;
}
inorder(tree);
printf("\n");
}
return 0;
}
```

OUTPUT

Input
3
2 3
-1 -1
-1 -1
2
1
1

Output
3 1 2
2 1 3

Input
17
2 3
4 5
6 7
-1 -1
8 9
10 11
12 13
-1 14
-1 -1
15 -
16 17
-1 -
-1 -
-1 -
-1 -
-1 -
-1
2

8th
29/2/24

OUTPUT :

Input
3
2 3
-1 -1
-1 -1
2
1
1

Output
3 1 2
2 1 3

Input
17
2 3
4 5
6 7
-1 -1
8 9
10 11
12 13
-1 14
-1 -1
15 -1
16 12
-1 -1
-1 -1
-1 -1
-1 -1
-1 -1
-1 -1
2

Output
14 8 5 9 2 4 10 7 12 1 3
10 16 6 17 11 15 9 5 14 8 2
13 7 12 4 1 3 17 11 16 6 [0]15

29/2/24