LAB (10) : (Hashing)

(29|2|24)

```
#include <stdio.h>
#define TABLE_SIZE 10

int hashFunction(int key)
{
  return key % TABLE_SIZE;
}
void insertValue(int hashTable[], int key)
{
  int i=0;
  int hkey = hashFunction(key);
  int index;

  do
  {
    index = (hkey + i) % TABLE_SIZE;
    if(hashTable[index] == -1)
    {
      hashTable[index] = key;
      printf("Inserted key %d at index %d\n", key, index);
      return;
    }
    i++;
  }while(i< TABLE_SIZE);
  printf("Unable to insert key %d. Hash table is full.\n", key);
}
```

```
int searchValue (int hashTable[], int key)
{
    int i=0;
    int hkey = hashFunction(key);
    int index;

    do
    {
        index = (hkey+i) % TABLE_SIZE;
        if(hashTable[index] == key)
        {
            printf("key %d found at index %d\n", key, index);
            return index;
        }
        i++;
    }while(i<TABLE_SIZE);
    printf("key %d not found in hash table\n", key);
    return -1;
}

int main()
{
    int hashTable[TABLE_SIZE];
    for(int i=0; i<TABLE_SIZE; i++)
    {
        hashTable[i] = -1;
    }
    insertValue(hashTable, 1234);
    insertValue(hashTable, 5678);
    insertValue(hashTable, 9876);

    searchValue(hashTable, 5678);
    searchValue(hashTable, 1111);

    return 0;
}
```

Output:

```
inserted key 1234 at index 4
inserted key 5678 at index 8
inserted key 9876 at index 6
key 5678 not found in hash table
key 1111 not found in hash table
```

HACKERANK program (Swamp Nodes)

```
#include<assert.h>
#include<stdbool.h>
#include<stdio.h>
#include<stdlib.h>
#include<string.h>

typedef struct Node
{
    int data;
    struct Node *left;
    struct Node *right;
}Node;

Node *createNode(int data)
{
    Node *newNode = (Node*)malloc(sizeof(Node));
    newNode->data = data;
    newNode->left = NULL;
    newNode->right = NULL;
    return newNode;
}

void inorderTraversal(Node* root, int *result, int *index)
{
    if(root == NULL) return;
    inorderTraversal(root->left, result, index);
    result[(*index)++] = root->data;
    inorderTraversal(root->right, result, index);
}

void swapAtLevel(Node *root, int k, int level)
{
    if(root == NULL) return;
    if((level % k == 0)){
        Node *temp = root->left;
        root->left = root->right;
        root->right = temp;
    }
    swapAtLevel(root->left, k, level+1);
    swapAtLevel(root->right, k, level+1);
}

int **swapNodes(int indexes_rows, int index_columns, int **indexes,
int queries_count, int *queries, int *result_rows, int *result_columns)
{
    Node **nodes = (Node**) malloc((indexes_rows+1)*sizeof(Node*));
    for(int i=1; i<=indexes_rows; i++){
        nodes[i] = createNode(i);
    }
```

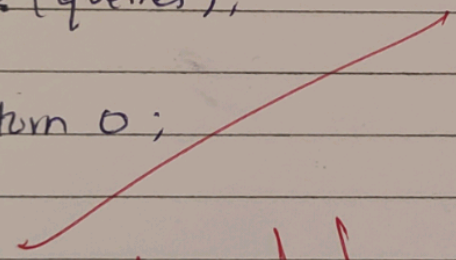
```
    for(int i=0 ; i<indexes_rows ; i++) {
        int leftIndex = indexes[i][0];
        int rightIndex = indexes[i][1];
        if(leftIndex != -1) nodes[i+1]->left = nodes[leftIndex];
        if(rightIndex != -1) nodes[i+1]->right = nodes[rightIndex];
    }
    int **result = (int **) malloc(queries_count * sizeof(int *));
    *result_rows = queries_count;
    *result_columns = indexes_rows;
    for(int i=0; i<queries_count; i++) {
        swapAtLevel(nodes[1], queries[i], 1);
        int *traversalResult = (int *) malloc(indexes_rows * sizeof(int));
        int index = 0;
        inOrderTraversal(nodes[1], traversalResult, &index);
        result[i] = traversalResult;
    }
    free(nodes);
    return result;
}
int main() {
    int n;
    scanf("%d", &n);
    int **indexes = malloc(n * sizeof(int*));
    for(int i=0; i<n; i++) {
        indexes[i] = malloc(2 * sizeof(int));
        scanf("%d %d", &indexes[i][0], &indexes[i][1]);
    }
    int queries_count;
    scanf("%d", &queries_count);
    int *queries = malloc(queries_count * sizeof(int));
    for(int i=0; i<queries_count; i++) {
        scanf("%d", &queries[i]);
    }
    int result_rows;
    int result_columns;
    int **result = swapNodes(n, 2, indexes, queries_count, queries, &result_rows,
                             &result_columns);
    for(int i=0; i<result_rows; i++) {
        for(int j=0; j<result_columns; j++) {
            printf("%d ", result[i][j]);
        }
        printf("\n");
        free(result[i]);
    }
}
```

8th
29/2/24

```
free(result);
for (int i=0; i<n; i++)
{
    free(indexes[i]);
}
free(indexes);
free(queries);

return 0;

}
```



29/12/24