**First Semester 2023-24**

**Data Structures and Algorithms Design (Merged-SEZG519/SSZG519)**

**LAB 4 (Dictionaries)**

*Notes: This lab covers the practicals on Dictionaries. We have mentioned programs in C language in the lab sheet. However, students are free to choose any of the programming languages to develop the solution to lab tasks.*

1. Complete the following code to implement binary search tree.

|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  struct node {  int data;  struct node \*right\_child;  struct node \*left\_child;  };  struct node\* new\_node(int x){  struct node \*temp;  temp = malloc(sizeof(struct node));  temp->data = x;  temp->left\_child = NULL;  temp->right\_child = NULL;  return temp;  }  struct node\* search(struct node \* root, int x){  ***//Write your code here…***  }  struct node\* insert(struct node \* root, int x){  ***//Write your code here…***  }  struct node\* find\_minimum(struct node \* root) {  ***//Write your code here…***  }  struct node\* delete(struct node \* root, int x) {  ***//Write your code here…***  }  void inorder(struct node \*root){  ***//Write your code here…***  }  void main() {  struct node \*root;  root = new\_node(20);  insert(root, 5);  insert(root, 1);  insert(root, 15);  insert(root, 9);  insert(root, 7);  insert(root, 12);  insert(root, 30);  insert(root, 25);  insert(root, 40);  insert(root, 45);  insert(root, 42);  inorder(root);  printf("\n");  root = delete(root, 1);  root = delete(root, 40);  root = delete(root, 45);  root = delete(root, 9);  inorder(root);  printf("\n");  } |

1. Understand the following code of hash table. Identify the values which cause collisions.

|  |
| --- |
| #include <stdbool.h>  #include <stdio.h>  #include <stdlib.h>  #define SIZE 20  struct DataItem {  int data;  int key;  };  struct DataItem\* hashArray[SIZE];  struct DataItem\* dummyItem;  struct DataItem\* item;  int hashCode(int key) {  return key % SIZE;  }  struct DataItem \*search(int key) {  //get the hash  int hashIndex = hashCode(key);    //move in array until an empty  while(hashArray[hashIndex] != NULL) {    if(hashArray[hashIndex]->key == key)  return hashArray[hashIndex];    //go to next cell  ++hashIndex;    //wrap around the table  hashIndex %= SIZE;  }  return NULL;  }  void insert(int key,int data) {  struct DataItem \*item = (struct DataItem\*) malloc(sizeof(struct DataItem));  item->data = data;  item->key = key;  //get the hash  int hashIndex = hashCode(key);  //move in array until an empty or deleted cell  while(hashArray[hashIndex] != NULL && hashArray[hashIndex]->key != -1) {  //go to next cell  ++hashIndex;    //wrap around the table  hashIndex %= SIZE;  }  hashArray[hashIndex] = item;  }  struct DataItem\* delete(struct DataItem\* item) {  int key = item->key;  //get the hash  int hashIndex = hashCode(key);  //move in array until an empty  while(hashArray[hashIndex] != NULL) {  if(hashArray[hashIndex]->key == key) {  struct DataItem\* temp = hashArray[hashIndex];  //assign a dummy item at deleted position  hashArray[hashIndex] = dummyItem;  return temp;  }  //go to next cell  ++hashIndex;  //wrap around the table  hashIndex %= SIZE;  }  return NULL;  }  void display() {  int i = 0;  for(i = 0; i<SIZE; i++) {    if(hashArray[i] != NULL)  printf(" (%d,%d)",hashArray[i]->key,hashArray[i]->data);  else  printf(" ~~ ");  }  printf("\n");  }  int main() {  dummyItem = (struct DataItem\*) malloc(sizeof(struct DataItem));  dummyItem->data = -1;  dummyItem->key = -1;  insert(1, 20);  insert(2, 70);  insert(42, 80);  insert(4, 25);  insert(12, 44);  insert(14, 32);  insert(17, 11);  insert(13, 78);  insert(37, 97);  display();  item = search(37);  if(item != NULL) {  printf("Element found: %d\n", item->data);  } else {  printf("Element not found\n");  }  delete(item);  item = search(37);  if(item != NULL) {  printf("Element found: %d\n", item->data);  } else {  printf("Element not found\n");  }  } |