Heap

#include <stdio.h>

void Insert(int A[], int n)

{

int i = n, temp;

temp = A[i];

while (i > 1 && temp > A[i / 2])

{

A[i] = A[i / 2];

i = i / 2;

}

A[i] = temp;

}

int Delete(int A[], int n)

{

int i, j, x, temp, val;

val = A[1];

x = A[n];

A[1] = A[n];

A[n] = val;

i = 1; j = i \* 2;

while (j < n - 1)

{

if (A[j + 1] > A[j])

j = j + 1;

if (A[i] < A[j])

{

temp = A[i];

A[i] = A[j];

A[j] = temp;

i = j;

j = 2 \* j;

}

else

break;

}

return val;

}

int main()

{

int H[] = { 0,10,20,30,25,5,40,35 };

// 40,25,35,10,5,20,30

int i;

for (i = 2; i <= 7; i++)

Insert(H, i);

for (i = 7; i > 1; i--)

{

Delete(H, i);

}

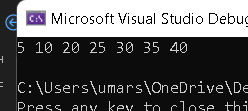
for (i = 1; i <= 7; i++)

printf("%d ", H[i]);

printf("\n");

return 0;

}



Graph

#ifndef Queue\_h

#define Queue\_h

#include <stdlib.h>

struct Node

{

int data;

struct Node\* next;

}\*front = NULL, \* rear = NULL;

void enqueue(int x)

{

struct Node\* t;

t = (struct Node\*)malloc(sizeof(struct Node));

if (t == NULL)

printf("Queue is FUll\n");

else

{

t->data = x;

t->next = NULL;

if (front == NULL)

front = rear = t;

else

{

rear->next = t;

rear = t;

}

}

}

int dequeue()

{

int x = -1;

struct Node\* t;

if (front == NULL)

printf("Queue is Empty\n");

else

{

x = front->data;

t = front;

front = front->next;

free(t);

}

return x;

}

int isEmpty()

{

return front == NULL;

}

#endif /\* Queue\_h \*/

#include <stdio.h>

#include "Queue.h"

void BFS(int G[][7], int start, int n)

{

int i = start, j;

int visited[7] = { 0 };

printf("%d ", i);

visited[i] = 1;

enqueue(i);

while (!isEmpty())

{

i = dequeue();

for (j = 1; j < n; j++)

{

if (G[i][j] == 1 && visited[j] == 0)

{

printf("%d ", j);

visited[j] = 1;

enqueue(j);

}

}

}

}

void DFS(int G[][7], int start, int n)

{

static int visited[7] = { 0 };

int j;

if (visited[start] == 0)

{

printf("%d ", start);

visited[start] = 1;

for (j = 1; j < n; j++)

{

if (G[start][j] == 1 && visited[j] == 0)

DFS(G, j, n);

}

}

}

int main()

{

int G[7][7] = { {0,0,0,0,0,0,0},

{0,0,1,1,0,0,0},

{0,1,0,0,1,0,0},

{0,1,0,0,1,0,0},

{0,0,1,1,0,1,1},

{0,0,0,0,1,0,0},

{0,0,0,0,1,0,0} };

DFS(G, 4, 7);

return 0;

}

Hashing

Linear Probing

#include <stdio.h>

#define SIZE 10

int hash(int key)

{

return key % SIZE;

}

int probe(int H[], int key)

{

int index = hash(key);

int i = 0;

while (H[(index + i) % SIZE] != 0)

i++;

return (index + i) % SIZE;

}

void Insert(int H[], int key)

{

int index = hash(key);

if (H[index] != 0)

index = probe(H, key);

H[index] = key;

}

int Search(int H[], int key)

{

int index = hash(key);

int i = 0;

while (H[(index + i) % SIZE] != key)

i++;

return (index + i) % SIZE;

}

int main()

{

int HT[10] = { 0 };

Insert(HT, 12);

Insert(HT, 25);

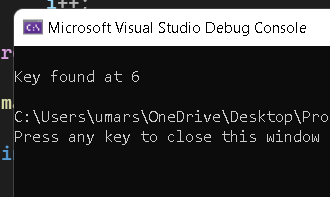
Insert(HT, 35);

Insert(HT, 26);

printf("\nKey found at %d\n", Search(HT, 35));

return 0;

}



Chaining

#ifndef Chains\_h

#define Chains\_h

#include<stdlib.h>

struct Node

{

int data;

struct Node\* next;

};

void SortedInsert(struct Node\*\* H, int x)

{

struct Node\* t, \* q = NULL, \* p = \*H;

t = (struct Node\*)malloc(sizeof(struct Node));

t->data = x;

t->next = NULL;

if (\*H == NULL)

\*H = t;

else

{

while (p && p->data < x)

{

q = p;

p = p->next;

}

if (p == \*H)

{

t->next = \*H;

\*H = t;

}

else

{

t->next = q->next;

q->next = t;

}

}

}

struct Node\* Search(struct Node\* p, int key)

{

while (p != NULL)

{

if (key == p->data)

{

return p;

}

p = p->next;

}

return NULL;

}

#endif /\* Chains\_h \*/

#include <stdio.h>

#include "Chains.h"

int hash(int key)

{

return key % 10;

}

void Insert(struct Node\* H[], int key)

{

int index = hash(key);

SortedInsert(&H[index], key);

}

int main()

{

struct Node\* HT[10];

struct Node\* temp;

int i;

for (i = 0; i < 10; i++)

HT[i] = NULL;

Insert(HT, 12);

Insert(HT, 22);

Insert(HT, 42);

temp = Search(HT[hash(21)], 21);

printf("%d ", temp->data);

return 0;

}