**DAA PRACTICAL**

**SINJINI SAMANTA**

ADMISSION NO: 23MS0142

COURSE: M.Sc.

MATHEMATICS AND COMPUTING

**ASSIGNMENT 1**

1. Write a program that takes as input a natural number n and outputs the sum of the first n natural numbers.

#include <stdio.h>

int findSum(int n)

{

int sum = 0;

for (int x = 1; x <= n; x++)

sum = sum + x;

return sum;

}

int main()

{

int n;

printf("Enter n: ");

scanf("%d",&n);

printf("%d", findSum(n));

return 0;

}

**OUTPUT**

Enter n: 5

15

1. Write a program that takes as input a natural number n and gives n! as output

#include<stdio.h>

long int factorial(int n);

int main() {

int n;

printf("Enter a positive integer: ");

scanf("%d",&n);

printf("Factorial of %d = %ld", n, factorial(n));

return 0;

}

long int factorial(int n) {

if (n>=1)

return n\*factorial(n-1);

else

return 1;

}

**OUTPUT**

Enter a positive integer: 6

Factorial of 6 = 720

1. Write a program that takes as input a natural number n and prints the first n numbers of the Fibonacci sequence.

#include <stdio.h>

int main() {

int i, n;

int t1 = 0, t2 = 1;

int nextTerm = t1 + t2;

printf("Enter the number of terms: ");

scanf("%d", &n);

printf("Fibonacci Series: %d, %d, ", t1, t2);

for (i = 3; i <= n; ++i) {

printf("%d, ", nextTerm);

t1 = t2;

t2 = nextTerm;

nextTerm = t1 + t2;

}

return 0;

}

**OUTPUT**

Enter the number of terms: 10

Fibonacci Series: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34

**ASSIGNMENT 2 Date:7/8/24**

1. Write a program that takes as input two sorted arrays and merge them into a single sorted array.

#include <stdio.h>

int main() {

int n1, n2, n3;

printf("Enter the size of the 1st array: ");

scanf("%d",&n1);

printf("Enter the size of the 2nd array: ");

scanf("%d",&n2);

n3= n1+ n2;

int arr1[n1], arr2[n2], arr3[n3];

printf("\nEnter The elements of the 1st array: ");

for(int i=0; i<n1; i++){

scanf("%d",&arr1[i]);

arr3[i] = arr1[i];

}

int k = n1;

printf("\nEnter The elements of the 2nd array: ");

for(int i=0; i<n2; i++){

scanf("%d",&arr2[i]);

arr3[k] = arr2[i];

k++;

}

printf("\nThe merged array before sorting: ");

for(int i=0;i<n3;i++){

printf("%d ",arr3[i]);

}

printf("\nSorted array is : ");

for(int i=0; i<n3; i++){

int temp;

for(int j=i+1; j<n3; j++){

if(arr3[i]>arr3[j]){

temp = arr3[i];

arr3[i] = arr3[j];

arr3[j] = temp;

}

}

}

printf("\nThe merged array after sorting: ");

for(int i=0;i<n3;i++){

printf("%d ",arr3[i]);

}

return 0;

}

**OUTPUT**

Enter the size of the 1st array: 5

Enter the size of the 2nd array: 4

Enter The elements of the 1st array: 5 8 10 19 21

Enter The elements of the 2nd array: 2 5 7 15

The merged array before sorting: 5 8 10 19 21 2 5 7 15

Sorted array is :

The merged array after sorting: 2 5 5 7 8 10 15 19 21

1. Write a program using recursion to implement Merge Sort.

#include<stdio.h>

int arr[20]; // array to be sorted

int main()

{

int n,i;

printf("Enter the size of array\n");

scanf("%d",&n);

printf("Enter the elements:");

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

scanf("%d",&arr[i]);

merge\_sort(arr,0,n-1);

printf("Sorted array:");

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

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

return 0;

}

int merge\_sort(int arr[],int low,int high)

{

int mid;

if(low<high)

{

mid=(low+high)/2;

merge\_sort(arr,low,mid);

merge\_sort(arr,mid+1,high);

merge(arr,low,mid,high);

}

}

int merge(int arr[],int l,int m,int h)

{

int arr1[10],arr2[10];

int n1,n2,i,j,k;

n1=m-l+1;

n2=h-m;

for(i=0;i<n1;i++)

arr1[i]=arr[l+i];

for(j=0;j<n2;j++)

arr2[j]=arr[m+j+1];

arr1[i]=9999; // To mark the end of each temporary array

arr2[j]=9999;

i=0;j=0;

for(k=l;k<=h;k++)

{

if(arr1[i]<=arr2[j])

arr[k]=arr1[i++];

else

arr[k]=arr2[j++];

}

}

**OUTPUT**

Enter the size of array

7

Enter the elements:6 2 10 5 4 11 15

Sorted array:2 4 5 6 10 11 15

1. Write a program using recursion such that given an integer n, it returns True if it is a power of four. Otherwise, returns False.

#include <stdio.h>

#include<stdbool.h>

int power( int n){

if( n==0)

return false;

if( n == 1){

return true;

}

return power(n/4);

}

int main() {

int n;

printf("Enter a number: ");

scanf("%d", &n);

if (power(n)) {

printf("True");

} else {

printf("False");

}

}

**OUTPUT**

Enter a number: 16

True

1. Write a program using recursion to calculate the summation of the digits in the given number.

#include <stdio.h>

int sum\_of\_digit(int n)

{

    if (n == 0)

       return 0;

    return (n % 10 + sum\_of\_digit(n / 10));

}

int main()

{

    int num;

    printf("Enter the digit: ");

    scanf("%d",&num);

    int result = sum\_of\_digit(num);

    printf("Sum of digits in %d is %d\n", num, result);

    return 0;

}

OUTPUT

Enter the digit: 164

Sum of digits in 164 is 11

**ASSIGNMENT 3 Date:14/8/24**

1. Write a program to using recursion convert a decimal number to binary.

#include <stdio.h>

int binary(int n){

    if(n ==0)

        return 0;

    else

        return (n % 2 + 10\* binary(n/2));

}

int main() {

    int n;

    printf("Enter a number to convert it into binary: ");

    scanf("%d",&n);

    printf("Binary form is: %d",binary(n));

    return 0;

}

**OUTPUT**

Enter a number to convert it into binary: 23

Binary form is: 10111

1. Write a program to implement Insertion Sort.

#include <stdio.h>

#include <math.h>

void insertionSort( int arr[], int n)

{

    int i,j,k;

    for(i=1; i<n;i++)

    {

        k = arr[i];

        j = i-1;

        while(j >=0 && arr[j]>k)

        {

            arr[j+1] = arr[j];

            j = j-1;

        }

        arr[j+1] = k;

    }

}

void printArray(int arr[], int n)

{

    for(int i=0;i<n;i++){

        printf("%d ",arr[i]);

    }

    printf("\n");

}

int main() {

    int i,n, arr[10];

    printf("Enter the number of element in the array: ");

    scanf("%d",&n);

    printf("Enter the array: ");

    //n = sizeof(arr) / sizeof(arr[0]);

    for(i=0;i<n;i++){

        scanf("%d",&arr[i]);

    }

    insertionSort(arr, n);

    printArray(arr, n);

    return 0;

}

**OUTPUT**

Enter the number of element in the array: 6

Enter the array: 6 2 10 5 4 11 15

2 4 5 6 10 11

1. Write a program using recursion to reverse a number.

#include <stdio.h>

int revNum = 0, r;

int reverse(int n)

{

    if(n) {

        r = n%10;

        revNum = revNum\*10 + r;

        reverse(n/10);

    } else

        return revNum;

    return revNum;

}

int main() {

    int n,rev;

    printf("Enter a number: ");

    scanf("%d",&n);

    rev = reverse(n);

    printf("The reversed number is: %d",rev);

    return 0;

}

**OUTPUT**

Enter a number: 1690

0961

1. Write a program using recursion to calculate the power of any number

#include <stdio.h>

int power(int a, int b)

{

    if(b!=0)

        return (a \* power(a, b-1));

    else

        return 1;

}

int main() {

    int a,b,num;

    printf("Enter a & b to get a^b: ");

    scanf("%d%d",&a,&b);

    num = power(a,b);

    printf("The ans is: %d",num);

    return 0;

}

**OUTPUT**

Enter a & b to get a^b: 5 3

The ans is: 125

1. Write a program using recursion to find the product of two numbers.

#include <stdio.h>

int product(int a, int b)

{

    if(a==0 | b==0)

        return 0;

    else

        return a+product(a,b-1);

}

int main() {

    int a,b,num;

    printf("Enter a & b to get a\*b: ");

    scanf("%d%d",&a,&b);

    num = product(a,b);

    printf("The ans is: %d",num);

    return 0;

}

**OUTPUT**

Enter a & b to get a\*b: 5 3

The ans is: 15

**ASSIGNMENT 4 Date:21/8/24**

1. Write a program that takes as input an array of numbers along an partition element from the array and returns a partition array corresponding to the element.

#include <stdio.h>

void partition(int arr[], int n, int p, int result[]) {

    int i;

    int l = 0;

    int g = n - 1;

   for (i = 0; i < n; i++) {

        if (arr[i] < p) {

            result[l++] = arr[i];

        } else if (arr[i] > p) {

            result[g--] = arr[i];

        }

    }

    int index = l;

    result[index] = p;

    printf("Partitioned Array: ");

    for (i = 0; i < n; i++) {

        printf("%d ", result[i]);

    }

    printf("\n");

}

int main() {

    int arr[] = {6, 2, 10, 5, 4, 11, 15, 5};

    int size = sizeof(arr) / sizeof(arr[0]);

    int partitionElement = 6;

    int result[size];

    partition(arr, size, partitionElement, result);

    return 0;

}

**OUTPUT**

Partitioned Array: 2 5 4 5 6 15 11 10

1. Write a program using recursion to implement QuickSort

#include <stdio.h>

void swap(int \*a, int \*b){

    int temp = \*a;

    \*a = \*b;

    \*b = temp;

}

int partition(int arr[], int l, int h)

{

    int pivot = arr[l];

    int i = l;

    int j = h;

    while(i<j){

        while(arr[i]<=pivot && i<= h-1){

            i++;

        }

        while (arr[j]>pivot && j>=l+1){

            j--;

        }

        if(i<j){

            swap(&arr[i], & arr[j]);

        }

    }

    swap(&arr[l], &arr[j]);

    return j;

}

void quickSort(int arr[], int l, int h){

    if(l<h){

        int p = partition(arr, l, h);

        quickSort(arr, l, p-1);

        quickSort(arr, p+1, h);

    }

}

void PrintArray(int arr[], int n){

    for(int i=0;i<n;i++)

        printf("%d ",arr[i]);

    printf("\n");

}

int main() {

    //int n,i;

    int arr[] = {6, 2, 10, 5, 4, 11, 15, 5};

    int n = sizeof(arr) / sizeof(arr[0]);

    quickSort(arr, 0, n-1);

    printf("\nThe sorted array is: ");

    PrintArray(arr,n);

    return 0;

}

**OUTPUT**

The sorted array is: 2 4 5 5 6 10 11 15

1. Write a program to verify if Quicksort is stable.

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

typedef struct {

int value;

int originalIndex;

} Element;

void swap(Element \*a, Element \*b) {

Element temp = \*a;

\*a = \*b;

\*b = temp;

}

int partition(Element arr[], int low, int high) {

int pivot = arr[high].value;

int i = low - 1;

for (int j = low; j < high; j++) {

if (arr[j].value < pivot) {

i++;

swap(&arr[i], &arr[j]);

}

}

swap(&arr[i + 1], &arr[high]);

return i + 1;

}

void quickSort(Element arr[], int low, int high) {

if (low < high) {

int pi = partition(arr, low, high);

quickSort(arr, low, pi - 1);

quickSort(arr, pi + 1, high);

}

}

void printArray(Element arr[], int size) {

for (int i = 0; i < size; i++) {

printf("Value: %d, Original Index: %d\n", arr[i].value, arr[i].originalIndex);

}

}

void checkStability(Element arr[], int size) {

int stable = 1;

for (int i = 1; i < size; i++) {

if (arr[i].value == arr[i - 1].value && arr[i].originalIndex < arr[i - 1].originalIndex) {

stable = 0;

break;

}

}

if (stable) {

printf("QuickSort is stable.\n");

} else {

printf("QuickSort is not stable.\n");

}

}

int main()

{

int size,i;

printf("Enter the size of the array:\n");

scanf("%d",&size);

Element arr[size];

printf("Enter the array element with their original index:\n");

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

{

printf("Element:");

scanf("%d",&arr[i].value);

printf("Original Index:");

scanf("%d",&arr[i].originalIndex);

}

quickSort(arr, 0, size - 1);

printf("\nSorted array:\n");

printArray(arr, size);

checkStability(arr, size);

return 0;

}

**OUTPUT**

Enter the size of the array:

5

Enter the array element with their original index:

Element:4

Original Index:0

Element:3

Original Index:1

Element:2

Original Index:2

Element:3

Original Index:3

Element:1

Original Index:4

Sorted array:

Value: 1, Original Index: 4

Value: 2, Original Index: 2

Value: 3, Original Index: 3

Value: 3, Original Index: 1

Value: 4, Original Index: 0

QuickSort is not stable

**ASSIGNMENT 5 Date:04/09/24**

1. Apply Bubble Sort to the algorithm

#include<stdio.h>  
//#include<conio.h>  
int main()  
{  
    int arr[50], i,j, n, temp;  
    int count=0;  
    printf("Enter the number of elements in the array: ");  
    scanf("%d",&n);  
    printf("ENter elements: ");  
    for(i=0;i<n;i++)  
        scanf("%d",&arr[i]);  
    for(i=0;i<(n-1);i++){  
        for(j=0;j<(n-i-1);j++)  
        {  
            count++;  
            if(arr[j]>arr[j+1]){  
                temp = arr[j];  
                arr[j]=arr[j+1];  
                arr[j+1] = temp;  
            }  
        }  
    }  
    printf("\nArray after applying Bubble sort: ");  
    for(i=0;i<n;i++)  
        printf("%d, ", arr[i]);  
    printf("\nThe number of comparison is: %d",count);  
}

**OUTPUT**

Enter the number of elements in the array: 7

Enter elements: 78 2 5 23 76 12 15

Array after applying Bubble sort: 2, 5, 12, 15, 23, 76, 78,

The number of comparison is: 21

**Optimised Bubble Sort**

#include<stdio.h>  
void bubbleSort(int arr[], int n, int \*count)  
{  
    int i,j, swapped,temp;  
    \*count =0;  
   for(i=0;i<(n-1);i++){  
       swapped =0;  
        for(j=0;j<(n-i-1);j++)  
        {  
            (\*count)++;  
            if(arr[j]>arr[j+1]){  
                temp = arr[j];  
                arr[j]=arr[j+1];  
                arr[j+1] = temp;  
                swapped = 1;  
            }  
        }  
        if(swapped == 0)  
            break;  
    }  
}  
void printArray(int arr[], int n){  
    for(int i=0;i<n;i++)  
        printf("%d, ", arr[i]);  
    printf("\n");  
}  
int main()  
{  
    int arr[50], i,j, n, temp;  
    int count=0;  
    printf("Enter the number of elements in the array: ");  
    scanf("%d",&n);  
    printf("ENter elements: ");  
    for(i=0;i<n;i++)  
        scanf("%d",&arr[i]);  
     
    bubbleSort(arr, n, &count);  
    printf("\nArray after applying Bubble sort: ");  
    printArray(arr,n);  
    printf("\nThe number of comparison is: %d",count);  
}

**OUTPUT**

Enter the number of elements in the array: 7

Enter elements: 78 2 5 23 76 12 17

Array after applying Bubble sort: 2, 5, 12, 17, 23, 76, 78,

The number of comparison is: 18

1. **Construct a binary search tree from a random array of size 15. Also, write a recursive function for in-order traversal to print the elements.**

#include <stdio.h>

#include <stdlib.h>

struct Node{

int data;

struct Node\* left;

struct Node\* right;};

struct Node\* newNode(int data)

{

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

node->data = data;

node->left = NULL;

node->right = NULL;

return node;

}

struct Node\* insert(struct Node\* node,int data)

{

if(node==NULL)

return newNode(data);

if(data < node->data)

node->left = insert(node->left,data);

else if(data > node->data)

node->right = insert(node->right,data);

return node;

}

void inorderTraversal(struct Node\* root)

{

if(root!=NULL)

{

inorderTraversal(root->left);

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

inorderTraversal(root->right);

}

}

int main()

{

int n,i;

struct Node\* root = NULL;

printf("enter the size of array:");

scanf("%d",&n);

if(n<=0)

{

printf("the size of the array must be positive\n");

return 1;

}

int \*values=(int \*)malloc(n\*sizeof(int));

if(values==NULL)

{

printf("memory allocation failed");

return 1;

}

printf("enter the array element:\n");

for(int i=0;i<n;i++)

scanf("%d",&values[i]);

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

root=insert(root,values[i]);

printf("the inorder traversal of the array elements is:\n");

inorderTraversal(root);

printf("\n");

}

**OUTPUT**

enter the size of array:15

enter the array element:

41 12 51 26 14 19 35 78 50 13 92 17 28 64 85 27

the inorder traversal of the array elements is:

12 13 14 17 19 26 28 35 41 50 51 64 78 85 92

**ASSIGNMENT 6** Date: 11/09/24

1. **Implement the heap sort algorithms to sort a given array in both ascending and descending order. (Use both Min and Max heap)**

#include <stdio.h>  
void swap(int \*a, int \*b){  
    int temp =\*a;  
    \*a = \*b;  
    \*b = temp;  
}  
void heapify(int arr[], int n, int i)  
{  
    int largest =i;  
    int left = 2\*i+1;  
    int right = 2\*i+2;  
     
    if(left < n && arr[left]>arr[largest])  
        largest = left;  
    if(right < n && arr[right]>arr[largest])  
        largest = right;  
    if(largest != i){  
        swap(&arr[i], &arr[largest]);  
    heapify(arr, n, largest);  
    }  
}  
  
void heapsort(int arr[], int n)  
{  
    for(int i=n/2-1; i>=0; i--)  
        heapify(arr,n,i);  
    for(int i= n-1;i>=0;i--){  
        swap(&arr[0], &arr[i]);  
        heapify(arr,i,0);  
    }  
}  
void printArray(int arr[], int n)  
{  
    for(int i=0;i<n;i++)  
        printf("%d ",arr[i]);  
    printf("\n");  
}  
int main() {  
    int n,arr[20];;  
    printf("Enter the number of element in the array: ");  
    scanf("%d",&n);  
    printf("Enter the elements: \n");  
    for(int i=0;i<n;i++)  
        scanf("%d",&arr[i]);  
         
    heapsort(arr,n);  
    printf("Sorted array is: \n");  
    printArray(arr,n);  
    return 0;  
}

**OUTPUT**

Enter the number of element in the array: 7  
Enter the elements:  
78 2 5 23 76 12 17  
Sorted array is:  
2 5 12 17 23 76 78

**Implement the Fibonacci Program using both Divide and Conquer, and Dynamic Programming**

#include <stdio.h>  
#include <time.h>  
  
int fibDC(int n) {  
    if (n <= 1)  
        return n;  
    return fibDC(n - 1) + fibDC(n - 2);  
}  
  
int fibDP(int n) {  
    if (n <= 1)  
        return n;  
     
    int fib[n + 1];  
    fib[0] = 0;  
    fib[1] = 1;  
     
    for (int i = 2; i <= n; i++)  
        fib[i] = fib[i - 1] + fib[i - 2];  
     
    return fib[n];  
}  
int main() {  
    int n;  
    printf("Enter the value of n for Fibonacci calculation: ");  
    scanf("%d", &n);  
     
    clock\_t start, end;  
    double cpu\_time\_used;  
    start = clock();  
    end = clock();  
    cpu\_time\_used = ((double) (end - start)) / CLOCKS\_PER\_SEC;  
    printf("Fibonacci series is: ");  
    for(int i=0;i<=n;i++)  
        printf("%d ",fibDC(i));  
    printf("\nTime taken by Divide and Conquer: %f seconds\n", cpu\_time\_used);  
     
    start = clock();  
    end = clock();  
    cpu\_time\_used = ((double) (end - start)) / CLOCKS\_PER\_SEC;  
    printf("Fibonacci series is: ");  
    for(int i=0;i<=n;i++)  
        printf("%d ",fibDP(i));  
    printf("\nTime taken by Dynamic Programming: %f seconds\n", cpu\_time\_used);  
     
    return 0;  
}

**OUTPUT**

Enter the value of n for Fibonacci calculation: 5  
Fibonacci series is: 0 1 1 2 3 5  
Time taken by Divide and Conquer: 0.000002 seconds  
Fibonacci series is: 0 1 1 2 3 5

Time taken by Dynamic Programming: 0.000001 seconds

**ASSIGNMENT 7** Date:25/09/24

1. **Implement the fractional knapsack problem**.

#include <stdio.h>

int n=3;

int p[10] = {10, 20, 30};

int w[10] = {60, 100, 120};

int W= 50;

int main() {

    int cur\_w;

    float tot\_v;

    int i, maxi;

    int used[10];

    for(i=0;i<n;++i)

        used[i] = 0;

    cur\_w = W;

    while(cur\_w >0)

    {

        maxi = -1;

        for(i=0;i<n; ++i){

            if((used[i] == 0) && ((maxi == -1) || ((float)w[i]/p[i] > (float)w[maxi]/p[maxi])))

                maxi = i;

        }

        used[maxi] = 1;

        cur\_w -= p[maxi];

        tot\_v += w[maxi];

        if(cur\_w >= 0)

            printf("Added object %d (%d, %d) completely in the bag. Space left: %d.\n", maxi+1, w[maxi], p[maxi], cur\_w);

        else{

            printf("Added %d (%d, %d) of object %d in the bag.\n", (int)((1 + (float)cur\_w/p[maxi])\*100), w[maxi], p[maxi], maxi+1);

            tot\_v -= w[maxi];

            tot\_v += (1+ (float)cur\_w/p[maxi])\*w[maxi];

        }

    }

    printf("Filled the bag of objects worth %.2f. \n", tot\_v);

return 0;

}

**OUTPUT**

Added object 1 (60, 10) completely in the bag. Space left: 40.

Added object 2 (100, 20) completely in the bag. Space left: 20.

Added 66 (120, 30) of object 3 in the bag.

Filled the bag of objects worth 240.00.

**Assignment 8** Date: 28/09/24

**Implement the Huffman codes algorithm**.

#include <stdio.h>

#include <stdlib.h>

#define MAX\_TREE\_HT 50

struct MinHNode {

  char item;

  unsigned freq;

  struct MinHNode \*left, \*right;

};

struct MinHeap {

  unsigned size;

  unsigned capacity;

  struct MinHNode \*\*array;

};

// Function declaration

void printArray(int arr[], int n);

// Create nodes

struct MinHNode \*newNode(char item, unsigned freq) {

  struct MinHNode \*temp = (struct MinHNode \*)malloc(sizeof(struct MinHNode));

  temp->left = temp->right = NULL;

  temp->item = item;

  temp->freq = freq;

return temp;

}

// Create min heap

struct MinHeap \*createMinH(unsigned capacity) {

  struct MinHeap \*minHeap = (struct MinHeap \*)malloc(sizeof(struct MinHeap));

minHeap->size = 0;

  minHeap->capacity = capacity;

  minHeap->array = (struct MinHNode \*\*)malloc(minHeap->capacity \* sizeof(struct MinHNode \*));

  return minHeap;

}

// Function to swap

void swapMinHNode(struct MinHNode \*\*a, struct MinHNode \*\*b) {

  struct MinHNode \*t = \*a;

  \*a = \*b;

  \*b = t;

}

// Heapify

void minHeapify(struct MinHeap \*minHeap, int idx) {

  int smallest = idx;

  int left = 2 \* idx + 1;

  int right = 2 \* idx + 2;

if (left < minHeap->size && minHeap->array[left]->freq < minHeap->array[smallest]->freq)

    smallest = left;

if (right < minHeap->size && minHeap->array[right]->freq < minHeap->array[smallest]->freq)

    smallest = right;

if (smallest != idx) {

    swapMinHNode(&minHeap->array[smallest], &minHeap->array[idx]);

    minHeapify(minHeap, smallest);

  }

}

// Check if size if 1

int checkSizeOne(struct MinHeap \*minHeap) {

  return (minHeap->size == 1);

}

// Extract min

struct MinHNode \*extractMin(struct MinHeap \*minHeap) {

  struct MinHNode \*temp = minHeap->array[0];

  minHeap->array[0] = minHeap->array[minHeap->size - 1];

--minHeap->size;

  minHeapify(minHeap, 0);

return temp;

}

// Insertion function

void insertMinHeap(struct MinHeap \*minHeap, struct MinHNode \*minHeapNode) {

  ++minHeap->size;

  int i = minHeap->size - 1;

  while (i && minHeapNode->freq < minHeap->array[(i - 1) / 2]->freq) {

    minHeap->array[i] = minHeap->array[(i - 1) / 2];

    i = (i - 1) / 2;

  }

  minHeap->array[i] = minHeapNode;

}

void buildMinHeap(struct MinHeap \*minHeap) {

  int n = minHeap->size - 1;

  int i;

for (i = (n - 1) / 2; i >= 0; --i)

    minHeapify(minHeap, i);

}

int isLeaf(struct MinHNode \*root) {

  return !(root->left) && !(root->right);

}

struct MinHeap \*createAndBuildMinHeap(char item[], int freq[], int size) {

  struct MinHeap \*minHeap = createMinH(size);

  for (int i = 0; i < size; ++i)

    minHeap->array[i] = newNode(item[i], freq[i]);

  minHeap->size = size;

  buildMinHeap(minHeap);

return minHeap;

}

struct MinHNode \*buildHuffmanTree(char item[], int freq[], int size) {

  struct MinHNode \*left, \*right, \*top;

  struct MinHeap \*minHeap = createAndBuildMinHeap(item, freq, size);

  while (!checkSizeOne(minHeap)) {

    left = extractMin(minHeap);

    right = extractMin(minHeap);

    top = newNode('$', left->freq + right->freq);

    top->left = left;

    top->right = right;

    insertMinHeap(minHeap, top);

  }

  return extractMin(minHeap);

}

void printHCodes(struct MinHNode \*root, int arr[], int top) {

  if (root->left) {

    arr[top] = 0;

    printHCodes(root->left, arr, top + 1);

  }

  if (root->right) {

    arr[top] = 1;

    printHCodes(root->right, arr, top + 1);

  }

  if (isLeaf(root)) {

    printf(" %c | ", root->item);

    printArray(arr, top);

  }

}

// Wrapper function

void HuffmanCodes(char item[], int freq[], int size) {

  struct MinHNode \*root = buildHuffmanTree(item, freq, size);

  int arr[MAX\_TREE\_HT], top = 0;

  printHCodes(root, arr, top);

}

void printArray(int arr[], int n) {

  int i;

  for (i = 0; i < n; ++i)

    printf("%d", arr[i]);

  printf("\n");

}

int main() {

  char arr[] = {'A', 'B', 'C', 'D'};

  int freq[] = {5, 1, 6, 3};

  int size = sizeof(arr) / sizeof(arr[0]);

  printf(" Char | Huffman code ");

  printf("\n--------------------\n");

  HuffmanCodes(arr, freq, size);

}

**OUTPUT**

Char | Huffman code

--------------------

C | 0

B | 100

D | 101

A | 11

**ASSIGNMENT 9** Date: 16/10/24

1. **Write a program to solve the Job Sequencing Problem with deadlines using a greedy algorithm. The program should find the maximum profit that can be earned by scheduling jobs within their deadlines.**

#include <stdio.h>

#include <stdlib.h>

struct Job {

int id; // Job ID

int deadline; // Job deadline

int profit; // Profit for the job

};

int compare(const void \*a, const void \*b) {

struct Job \*jobA = (struct Job \*)a;

struct Job \*jobB = (struct Job \*)b;

return jobB->profit - jobA->profit;

}

int jobSequencing(struct Job jobs[], int n) {

qsort(jobs, n, sizeof(struct Job), compare);

int result[n]; // Result array to keep track of free time slots

int jobSequence[n]; // Array to store the sequence of job IDs

int maxProfit = 0;

for (int i = 0; i < n; i++) {

result[i] = -1;

}

for (int i = 0; i < n; i++) {

for (int j = jobs[i].deadline - 1; j >= 0; j--) {

if (j < n && result[j] == -1) {

result[j] = i; // Assign job to this slot

jobSequence[j] = jobs[i].id; // Store the job ID in sequence

maxProfit += jobs[i].profit; // Update maximum profit

break;

}

}

}

printf("Scheduled Jobs (in order):\n");

for (int i = 0; i < n; i++) {

if (result[i] != -1) {

printf("Job ID: %d, Profit: %d, Deadline: %d\n",

jobs[result[i]].id, jobs[result[i]].profit, jobs[result[i]].deadline);

}

}

return maxProfit;

}

int main() {

struct Job jobs[] = {

{1, 2, 6},

{2, 1, 8},

{3, 1, 5},

{4, 2, 10}

};

int n = sizeof(jobs) / sizeof(jobs[0]);

int maxProfit = jobSequencing(jobs, n);

printf("Maximum Profit: %d\n", maxProfit);

return 0;

}

**OUTPUT**

Scheduled Jobs (in order):

Job ID: 2, Profit: 8, Deadline: 1

Job ID: 4, Profit: 10, Deadline: 2

Maximum Profit: 18

1. **Implement Prim’s Algorithm using a Min Heap to find the minimum cost MST of any random graph**.

#include <limits.h>

#include <stdbool.h>

#include <stdio.h>

#define V 5

int minKey(int key[], bool mstSet[])

{

int min = INT\_MAX, min\_index;

for (int v = 0; v < V; v++)

if (mstSet[v] == false && key[v] < min)

min = key[v], min\_index = v;

return min\_index;

}

int printMST(int parent[], int graph[V][V])

{

printf("Edge \tWeight\n");

for (int i = 1; i < V; i++)

printf("%d - %d \t%d \n", parent[i], i,

graph[i][parent[i]]);

}

void primMST(int graph[V][V])

{

int parent[V], key[V];

bool mstSet[V];

for (int i = 0; i < V; i++)

key[i] = INT\_MAX, mstSet[i] = false;

key[0] = 0;

parent[0] = -1;

for (int count = 0; count < V - 1; count++) {

int u = minKey(key, mstSet);

mstSet[u] = true;

for (int v = 0; v < V; v++)

if (graph[u][v] && mstSet[v] == false

&& graph[u][v] < key[v])

parent[v] = u, key[v] = graph[u][v];

}

printMST(parent, graph);

}

int main()

{

int graph[V][V] = { { 0, 2, 0, 6, 0 },

{ 2, 0, 3, 8, 5 },

{ 0, 3, 0, 0, 7 },

{ 6, 8, 0, 0, 9 },

{ 0, 5, 7, 9, 0 } };

primMST(graph);

return 0;

}

**OUTPUT**

Edge Weight

0 - 1 2

1 - 2 3

0 - 3 6

1 - 4 5

**Assignment 10**  Date: 23/10/24

**1. Implement the BFS algorithm.**

#include <stdio.h>  
   
int n, i, j, visited[10], queue[10], front = -1, rear = -1;  
int adj[10][10];  
   
void bfs(int v)  
{  
    for (i = 1; i <= n; i++)  
        if (adj[v][i] && !visited[i])  
            queue[++rear] = i;  
    if (front <= rear)  
    {  
        visited[queue[front]] = 1;  
        bfs(queue[front++]);  
    }  
}  
   
void main()  
{  
    int v;  
    printf("Enter the number of vertices: ");  
    scanf("%d", &n);  
    for (i = 1; i <= n; i++)  
    {  
        queue[i] = 0;  
        visited[i] = 0;  
    }  
    printf("Enter graph data in matrix form:    \n");  
    for (i = 1; i <= n; i++)  
        for (j = 1; j <= n; j++)  
            scanf("%d", &adj[i][j]);  
    printf("Enter the starting vertex: ");  
    scanf("%d", &v);  
    bfs(v);  
    printf("The node which are reachable are:    \n");  
    for (i = 1; i <= n; i++)  
        if (visited[i])  
            printf("%d\t", i);  
        else  
            printf("BFS is not possible. Not all nodes are reachable");  
    return 0;  
}

**OUTPUT**

Enter the number of vertices: 5

Enter graph data in matrix form:

0 1 0 1 0

1 0 1 1 0

0 1 0 1 1

1 1 1 0 1

0 0 1 1 0

Enter the starting vertex: 1

The node which are reachable are:

1 2 3 4 5

2**. Modify your BFS algorithm to detect cycles in a given graph**.

#include <bits/stdc++.h>  
using namespace std;  
  
void addEdge(vector<int> adj[], int u, int v)  
{  
    adj[u].push\_back(v);  
    adj[v].push\_back(u);  
}  
  
bool isCyclicConnected(vector<int> adj[], int s, int V, vector<bool>& visited){  
    // Create a queue for BFS  
    queue<int> q;  
    // Enqueue the current node  
    q.push(s);  
  
    while (!q.empty()) {  
        // Dequeue a vertex from queue and print it  
        int v = q.front();  
        q.pop();  
  
        if (visited[v] == 1) {  
            return true; // Cycle detected  
        }  
  
        visited[v] = 1; // Mark as visited  
  
        // Visit adjacent nodes  
        for (auto it : adj[v]) {  
            if (visited[it] == 0) {  
                q.push(it);  
            }  
        }  
    }  
    return false;  
}  
  
bool isCyclicDisconnected(vector<int> adj[], int V){  
    // Mark all the vertices as not visited  
    vector<bool> visited(V, false);  
  
    for (int i = 0; i < V; i++) {  
        if (!visited[i]  
            && isCyclicConnected(adj, i, V, visited))  
            return true;  
    }  
    return false;  
}  
int main(){  
    int V = 4;  
    vector<int> adj[V];  
    addEdge(adj, 0, 1);  
    addEdge(adj, 1, 2);  
    addEdge(adj, 2, 0);  
    addEdge(adj, 2, 3);  
  
    if (isCyclicDisconnected(adj, V))  
        cout << "Yes";  
    else  
        cout << "No";  
  
    return 0;  
}

**OUTPUT**

Yes

**Assignment 11** Date:30/10/24

1. **Implement Dijkstra’s algorithm to find out the Single-Source Shortest Paths.**

#include <stdio.h>

#include <stdlib.h>

#include <limits.h>

#define V 5

int minDistance(int dist[], int sptSet[]) {

    int min = INT\_MAX, min\_index;

    for (int v = 0; v < V; v++) {

        if (sptSet[v] == 0 && dist[v] <= min) {

            min = dist[v];

            min\_index = v;

        }

    }

    return min\_index;

}

void dijkstra(int graph[V][V], int src) {

    int dist[V];

    int sptSet[V];

    for (int i = 0; i < V; i++) {

        dist[i] = INT\_MAX;

        sptSet[i] = 0;

    }

    dist[src] = 0;

    for (int count = 0; count < V - 1; count++) {

        int u = minDistance(dist, sptSet);

        sptSet[u] = 1;

        for (int v = 0; v < V; v++) {

            if (!sptSet[v] && graph[u][v] && dist[u] != INT\_MAX && dist[u] + graph[u][v] < dist[v]) {

                dist[v] = dist[u] + graph[u][v];

            }

        }

    }

    printf("Vertex\tDistance from Source\n");

    for (int i = 0; i < V; i++)

        printf("%d\t%d\n", i, dist[i]);

}

int main() {

    int graph[V][V] = {{0, 10, 0, 30, 100},{10, 0, 50, 0, 0},{0, 50, 0, 20, 10}, {30, 0, 20, 0, 60},{100, 0, 10, 60, 0}};

    dijkstra(graph, 0);

    return 0;

}

**OUTPUT**

Vertex Distance from Source

0 0

1 10

2 50

3 30

4 60

**Assignment 12**  Date: 6/11/24

1. Implement the Bellman-Ford algorithm.

#include <iostream>

#include <vector>

using namespace std;

vector<int> bellmanFord(int V, vector<vector<int>>& edges, int src) {

vector<int> dist(V, 1e8);

dist[src] = 0;

for (int i = 0; i < V; i++) {

for (vector<int> edge : edges) {

int u = edge[0];

int v = edge[1];

int wt = edge[2];

if (dist[u] != 1e8 && dist[u] + wt < dist[v]) {

if(i == V - 1)

return {-1};

dist[v] = dist[u] + wt;

}

}

}

return dist;

}

int main() {

int V = 5;

vector<vector<int>> edges = {{1, 3, 2}, {4, 3, -1}, {2, 4, 1},

{1, 2, 1}, {0, 1, 5}};

int src = 0;

vector<int> ans = bellmanFord(V, edges, src);

for (int dist : ans)

cout << dist << " ";

return 0;

}

**OUTPUT**

0 5 6 6 7

2. **Implement the 0/1 Knapsack problem using dynamic programming with the following approaches:**

**• Top-down approach**

#include <stdio.h>  
int memo[100][100];  
void initializeMemo(int n, int W) {  
    for (int i = 0; i <= n; i++) {  
        for (int w = 0; w <= W; w++) {  
            memo[i][w] = -1; // Initialize all entries to -1 (indicating not yet computed)  
        }  
    }  
}  
int knapsack(int W, int weights[], int values[], int n) {  
      if (n == 0 || W == 0) {  
        return 0;  
    }  
if (memo[n][W] != -1) {  
        return memo[n][W];  
    }  
if (weights[n-1] > W) {  
        memo[n][W] = knapsack(W, weights, values, n-1);  
    } else {  
        int include\_item = values[n-1] + knapsack(W - weights[n-1], weights, values, n-1);  
        int exclude\_item = knapsack(W, weights, values, n-1);  
        memo[n][W] = (include\_item > exclude\_item) ? include\_item : exclude\_item;  
    }  
    return memo[n][W];  
}  
  
int main() {  
    int values[] = {60, 100, 120};   // Values (profits) of the items  
    int weights[] = {10, 20, 30};    // Weights of the items  
    int W = 50;                      // Capacity of the knapsack  
    int n = sizeof(values) / sizeof(values[0]);  
    initializeMemo(n, W);  
   int max\_value = knapsack(W, weights, values, n);  
    printf("Maximum value that can be obtained: %d\n", max\_value);  
  
    return 0;  
}

**• Bottom-up approach**

#include <stdio.h>

int knapsack(int W, int weights[], int values[], int n) {

int dp[n+1][W+1];

for (int i = 0; i <= n; i++) {

for (int w = 0; w <= W; w++) {

// Base case: If no items or knapsack capacity is 0

if (i == 0 || w == 0) {

dp[i][w] = 0;

} else if (weights[i-1] <= w) {

// If the current item can be included

dp[i][w] = (values[i-1] + dp[i-1][w-weights[i-1]] > dp[i-1][w])

? (values[i-1] + dp[i-1][w-weights[i-1]])

: dp[i-1][w];

} else {

// If the current item cannot be included

dp[i][w] = dp[i-1][w];

}

}

}

return dp[n][W];

}

int main() {

int values[] = {60, 100, 120}; // Values (profits) of the items

int weights[] = {10, 20, 30}; // Weights of the items

int W = 50; // Capacity of the knapsack

int n = sizeof(values) / sizeof(values[0]);

int max\_value = knapsack(W, weights, values, n);

printf("Maximum value that can be obtained: %d\n", max\_value);

return 0;

}

**OUTPUT**

Maximum value that can be obtained: 220.