**MANAV RACHNA INTERNATIONAL INSTITUTE OF RESEARCH AND STUDIES**

****

**Design Analysis and Algorithm LAB FILE**

**SUBMITTED TO: SUBMITTED BY:**

**Mohit Chaudhary SAPNA SINHA**

**Assistant Professor 1/18/FET/BCS/079**

**Department Of CSE**

**FET MRIIRS**

**INDEX**

|  |  |  |
| --- | --- | --- |
| S No. | Practical | Date |
| 1 | Sort a set of numbers into ascending/Descending order Using different sorting algorithms and calculate the time complexity by step-count method. | 02/06/20 |
| 2 | WAP to sort a set of numbers using Merge sort and Quick sort using divide and conquer method. | 09/06/20 |
| 3 | Write a program for multiplications of two Matrices using Stassen’s Multiplication Algorithms using Divide and Conquer method. | 16/06/20 |
| 4 | Write a program for finding maximum and minimum in an array using Divide and Conquer method. | 23/06/20 |
| 5 | Write a program for implementation of Convex hull using Divide and Conquer method. | 30/06/20 |
| 6 | Write a program to solve Knapsack problem using Greedy Algorithm. | 07/07/20 |
| 7 | Write a program for implementation of Huffman algorithm. | 14/07/20 |
| 8 | Write a program to solve Sequencing Problems with deadlines using Greedy algorithm. | 04/08/20 |
| 9 | Write a program for implementation of Single source shortest path algorithm for Dijkstra’s algorithm. | 11/08/20 |
| 10 | Implement Graph on two-dimensional array and use Greedy method to obtain minimum-cost spanning tree of the graph using Prim’s and Krushkal’s. | 18/08/20 |
| 11 | Write a program to find a number in an array by binary search method. | 18/08/20 |
| 12 | Write a program of all pairs shortest path using Dynamic programming. | 25/08/20 |
| 13 | Write a program for optimal binary search of an element in an array using Dynamic programming. | 25/08/20 |
| 14 | Write a program for 0/1 Knapsack problem using Dynamic programming. | 01/09/20 |
| 15 | Write a program for Matrix-Chain Multiplication using Dynamic programming. | 01/09/20 |
| 16 | Write a program to find the Largest Common Subsequence’s of two sets using Dynamic programming. | 08/09/20 |
| 17 | Write a program for travelling salesman problem. | 15/09/20 |
| 18 | Write a program for Graph Traversal Algorithms : Depth First Search(DFS). | 15/09/20 |
| 19 | Write a program for Graph Traversal Algorithm: Breadth First Search (BFS). | 22/09/20 |
| 20 | Write a program for solution space for 8 queen problem and solve the problem using Back-Tracking method. | 22/09/20 |
| 21 | Write a program for Sum of Subset problem of a given set using back tracking method. | 29/09/20 |
| 22 | Write a program for string matching by (i) Naive-string matching method and (ii) Rabin-Krap algorithm and compare number of operations done in these methods. | 29/09/20 |
| 23 | Write a program for string matching using Knuth-Morris-Pratt Algorithms. | 06/10/20 |
| 24 | Write a program for string matching using finite Automata method and Knuth-Morris-Pratt Algorithms. | 06/10/20 |

**Program No: 01**

**Aim:** Write a program in C language to sort a set of numbers into ascending/Descending order using different sorting algorithms(Selection, Insertion, Bubble Sort, Quick Sort) and calculate the time complexity by step-count method.

Take the input-set from a table and repeat the operation several times 10,20,30,40, 50 times and plot a graph.

Examine the best case, worst-case and average case by taking suitable input data.

**Source Code:**

#include <stdio.h>

#include <stdlib.h>

int count=0, prev=0;

voidrandomGenerator(int \*arrAdd, int n)

{

inti;

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

{

\*(arrAdd+i) = rand() % 50;

printf(" %d",\*(arrAdd+i));

}

}

void swap(int \*xp, int \*yp)

{

int temp = \*xp;

\*xp = \*yp;

\*yp = temp;

}

voidprintArray(intarr[], int size)

{

inti;

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

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

printf("\n");

}

voidselectionSort(intarr[], int n)

{

inti, j, min\_idx;

printf("\n The sorted array using selection sort: ");

for (i = 0; i< n-1; i++)

{

min\_idx = i;

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

{

if (arr[j] <arr[min\_idx])

min\_idx = j;

count++;

}

swap(&arr[min\_idx], &arr[i]);

count++;

}

}

voidinsertionSort(intarr[], int n)

{

inti, key, j;

printf("\n The sorted array using insertion sort: ");

for (i = 1; i< n; i++) {

key = arr[i];

j = i - 1;

while (j >= 0 &&arr[j] > key) {

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

j = j - 1;

count++;

}

arr[j + 1] = key;

count++;

}

}

voidbubbleSort(intarr[], int n)

{

printf("\n The sorted array using bubble sort: ");

inti, j;

for (i = 0; i< n-1; i++)

for (j = 0; j < n-i-1; j++)

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

swap(&arr[j], &arr[j+1]);

}

int partition (intarr[], int low, int high)

{

int pivot = arr[high];

inti = (low - 1);

for (int j = low; j <= high- 1; j++)

{

if (arr[j] < pivot)

{

i++;

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

}

}

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

return (i + 1);

}

voidquickSort(intarr[], int low, int high)

{

if (low < high)

{

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

quickSort(arr, low, pi - 1);

quickSort(arr, pi + 1, high);

}

}

int main()

{

printf("\n SORTING COMPLEXITIES\n\n");

int arr1[10], arr2[20], arr3[30], arr4[40], arr5[50];

int \*a1=&arr1; int \*a2=&arr2; int \*a3=&arr3; int \*a4=&arr4; int \*a5=&arr5;

//1

int n=10;

printf(" Elements of array 1 are: ");

randomGenerator(a1,n);

selectionSort(arr1,n);

printArray(arr1, n);

printf(" Comparisons: %d\n", count);

prev=count;

printf(" Elements of array 1 are: ");

randomGenerator(a1,n);

insertionSort(arr1, n);

printArray(arr1, n);

printf(" Comparisons: %d\n", count-prev);

prev=count-prev;

printf(" Elements of array 1 are: ");

randomGenerator(a1,n);

bubbleSort(arr1, n);

printArray(arr1, n);

printf(" Comparisons: %d\n", count-prev);

prev=count-prev;

printf(" Elements of array 1 are: ");

randomGenerator(a1,n);

quickSort(arr1, 0, n-1);

printf("\n The sorted array using quick sort: ");

printArray(arr1, n);

printf(" Comparisons: %d\n", count-prev);

prev=count-prev;

//2

n=20;

printf("\n Elements of array 2 are: ");

randomGenerator(a2,n);

selectionSort(arr2,n);

printArray(arr2, n);

printf(" Comparisons: %d\n", count-prev);

prev=count-prev;

printf(" Elements of array 2 are: ");

randomGenerator(a2,n);

insertionSort(arr2, n);

printArray(arr2, n);

printf(" Comparisons: %d\n", count-prev);

prev=count-prev;

printf(" Elements of array 2 are: ");

randomGenerator(a2,n);

bubbleSort(arr2, n);

printArray(arr2, n);

printf(" Comparisons: %d\n", count-prev);

prev=count-prev;

printf(" Elements of array 2 are: ");

randomGenerator(a2,n);

quickSort(arr2, 0, n-1);

printf("\n The sorted array using quick sort: ");

printArray(arr2, n);

printf(" Comparisons: %d\n", count-prev);

prev=count-prev;

//3

n=30;

printf("\n Elements of array 3 are: ");

randomGenerator(a3,n);

selectionSort(arr3,n);

printArray(arr3, n);

printf(" Comparisons: %d\n", count-prev);

prev=count-prev;

printf(" Elements of array 3 are: ");

randomGenerator(a3,n);

insertionSort(arr3, n);

printArray(arr3, n);

printf(" Comparisons: %d\n", count-prev);

prev=count-prev;

printf(" Elements of array 3 are: ");

randomGenerator(a3,n);

bubbleSort(arr3, n);

printArray(arr3, n);

printf(" Comparisons: %d\n", count-prev);

prev=count-prev;

printf(" Elements of array 3 are: ");

randomGenerator(a3,n);

quickSort(arr3, 0, n-1);

printf("\n The sorted array using quick sort: ");

printArray(arr3, n);

printf(" Comparisons: %d\n", count-prev);

prev=count-prev;

//4

n=40;

printf("\n Elements of array 4 are:\n ");

randomGenerator(a4,n);

selectionSort(arr4,n);

printArray(arr4, n);

printf(" Comparisons: %d\n", count-prev);

prev=count-prev;

printf(" Elements of array 4 are:\n ");

randomGenerator(a4,n);

insertionSort(arr4, n);

printArray(arr4, n);

printf(" Comparisons: %d\n", count-prev);

prev=count-prev;

printf(" Elements of array 4 are: ");

randomGenerator(a4,n);

bubbleSort(arr4, n);

printArray(arr4, n);

printf(" Comparisons: %d\n", count-prev);

prev=count-prev;

printf(" Elements of array 4 are: ");

randomGenerator(a4,n);

quickSort(arr4, 0, n-1);

printf("\n The sorted array using quick sort: ");

printArray(arr4, n);

printf(" Comparisons: %d\n", count-prev);

prev=count-prev;

//5

n=50;

printf("\n Elements of array 5 are:\n ");

randomGenerator(a5,n);

selectionSort(arr5,n);

printArray(arr5, n);

printf(" Comparisons: %d\n", count-prev);

prev=count-prev;

printf(" Elements of array 5 are:\n ");

randomGenerator(a5,n);

insertionSort(arr5, n);

printArray(arr5, n);

printf(" Comparisons: %d\n", count-prev);

prev=count-prev;

printf(" Elements of array 5 are: ");

randomGenerator(a5,n);

bubbleSort(arr5, n);

printArray(arr5, n);

printf(" Comparisons: %d\n", count-prev);

prev=count-prev;

printf(" Elements of array 5 are: ");

randomGenerator(a5,n);

quickSort(arr5, 0, n-1);

printf("\n The sorted array using quick sort: ");

printArray(arr5, n);

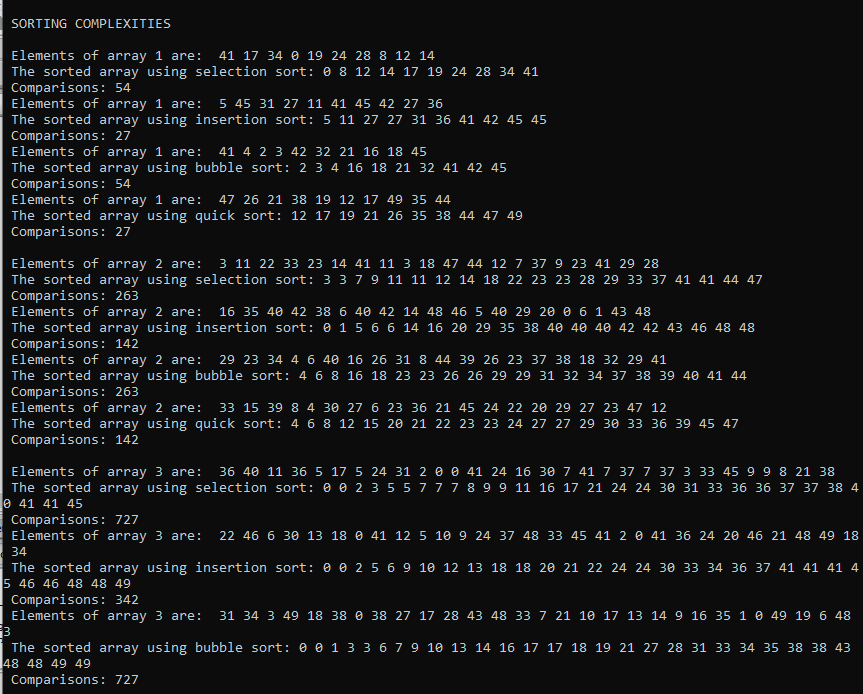
printf(" Comparisons: %d\n", count-prev);

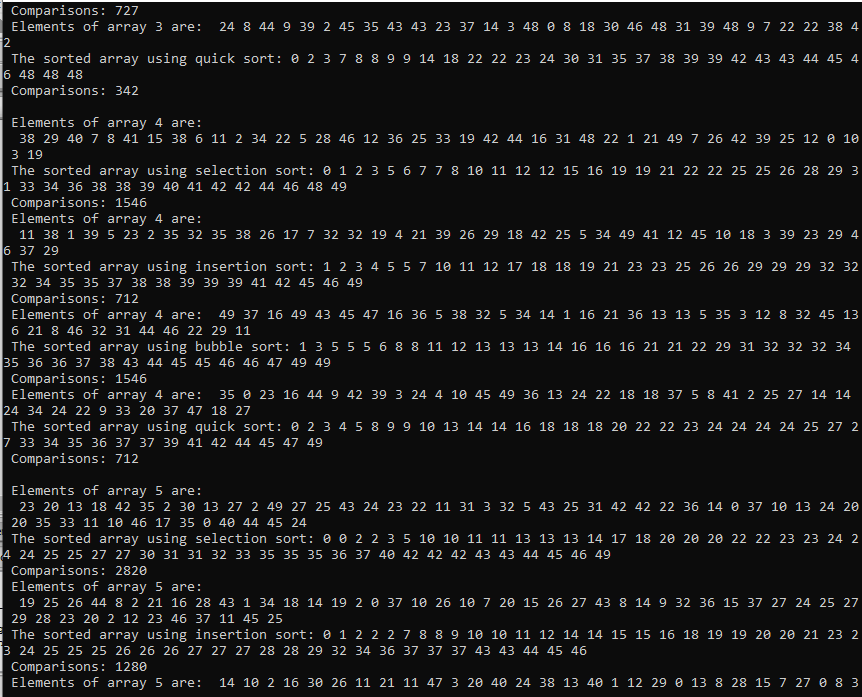
prev=count-prev;

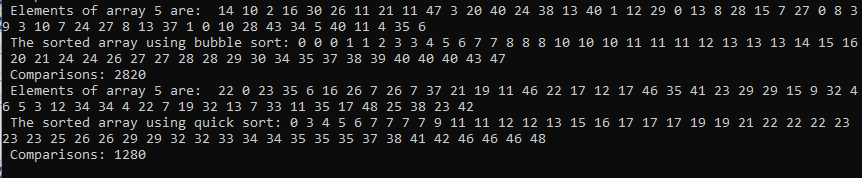
return 0;

}

**Output:**



****

****

**Graph:**

**Program No: 02**

**Aim:** WAP to sort a set of numbers using

**(i) Merge sort and**

**Source Code:**

**Output:**

**(ii) Quick-sort using divide and conquer method.**

**Source Code:**

**Output:**

**PROGRAM NO: 03**

**AIM:** Write a program for multiplications of two Matrices using Stassen’s Multiplication Algorithms using Divide and Conquer method.

**SOURCE CODE:**

#include <stdio.h>

int main()

{

int a[2][2],b[2][2],c[2][2],i,j;

int m1,m2,m3,m4,m5,m6,m7;

printf("Enter the 4 elements of first matrix: ");

for(i=0;i<2;i++)

for(j=0;j<2;j++)

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

printf("Enter the 4 elements of second matrix: ");

for(i=0;i<2;i++)

for(j=0;j<2;j++)

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

printf("\nThe first matrix is\n");

for(i=0;i<2;i++)

{

printf("\n");

for(j=0;j<2;j++)

printf("%d\t",a[i][j]);

}

printf("\nThe second matrix is\n");

for(i=0;i<2;i++)

{

printf("\n");

for(j=0;j<2;j++)

printf("%d\t",b[i][j]);

}

m1= (a[0][0] + a[1][1])\*(b[0][0]+b[1][1]);

m2= (a[1][0]+a[1][1])\*b[0][0];

m3= a[0][0]\*(b[0][1]-b[1][1]);

m4= a[1][1]\*(b[1][0]-b[0][0]);

m5= (a[0][0]+a[0][1])\*b[1][1];

m6= (a[1][0]-a[0][0])\*(b[0][0]+b[0][1]);

m7= (a[0][1]-a[1][1])\*(b[1][0]+b[1][1]);

c[0][0]=m1+m4-m5+m7;

c[0][1]=m3+m5;

c[1][0]=m2+m4;

c[1][1]=m1-m2+m3+m6;

printf("\nAfter multiplication using \n");

for(i=0;i<2;i++)

{

printf("\n");

for(j=0;j<2;j++)

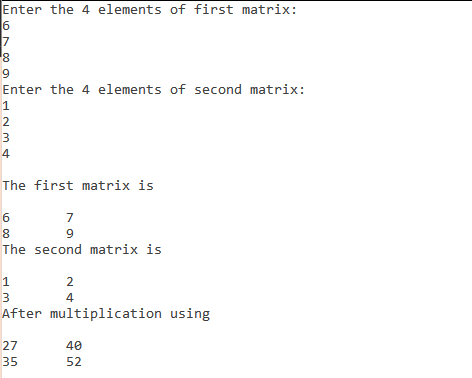
printf("%d\t",c[i][j]);

}

return 0;

}

**OUTPUT:**



**PROGRAM NO: 04**

**AIM:** Write a program for finding maximum and minimum in an array using Divide and Conquer method.

**SOURCE CODE:**

#include<stdio.h>

struct pair

{

  int min;

  int max;

};

struct pair getMinMax(int arr[], int n)

{

  struct pair minmax;

  int i;

  /\*If there is only one element then return it as min and max both\*/

  if (n == 1)

  {

     minmax.max = arr[0];

     minmax.min = arr[0];

     return minmax;

  }

  /\* If there are more than one elements, then initialize min

      and max\*/

  if (arr[0] > arr[1])

  {

      minmax.max = arr[0];

      minmax.min = arr[1];

  }

  else

  {

      minmax.max = arr[1];

      minmax.min = arr[0];

  }

  for (i = 2; i<n; i++)

  {

    if (arr[i] >  minmax.max)

      minmax.max = arr[i];

    else if (arr[i] <  minmax.min)

      minmax.min = arr[i];

  }

  return minmax;

}

/\* Driver program to test above function \*/

int main()

{

  int arr[] = {1000, 11, 445, 1, 330, 3000};

  int arr\_size = 6;

  struct pair minmax = getMinMax (arr, arr\_size);

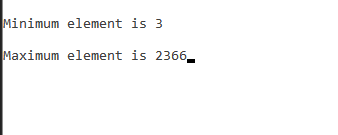
  printf("nMinimum element is %d", minmax.min);

  printf("nMaximum element is %d", minmax.max);

  getchar();

}

**OUTPUT:**



**PROGRAM NO: 05**

**AIM:** Write a program for implementation of Convex hull using Divide and Conquer method.

**SOURCE CODE:**

int compare(const void \*vp1, const void \*vp2)

{

Point \*p1 = (Point \*)vp1;

Point \*p2 = (Point \*)vp2;

int o = orientation(p0, \*p1,

#include <iostream>

#include <stack>

#include <stdlib.h>

using namespace std;

struct Point

{

int x, y;

};

// A global point needed for sorting points with reference

// to the first point Used in compare function of qsort()

Point p0;

// A utility function to find next to top in a stack

Point nextToTop(stack<Point> &S)

{

Point p = S.top();

S.pop();

Point res = S.top();

S.push(p);

return res;

}

// A utility function to swap two points

int swap(Point &p1, Point &p2)

{

Point temp = p1;

p1 = p2;

p2 = temp;

}

// A utility function to return square of distance

// between p1 and p2

int distSq(Point p1, Point p2)

{

return (p1.x - p2.x)\*(p1.x - p2.x) +

(p1.y - p2.y)\*(p1.y - p2.y);

}

int orientation(Point p, Point q, Point r)

{

int val = (q.y - p.y) \* (r.x - q.x) -

(q.x - p.x) \* (r.y - q.y);

if (val == 0) return 0; // colinear

return (val > 0)? 1: 2; // clock or counterclock wise

}

\*p2);

if (o == 0)

return (distSq(p0, \*p2) >= distSq(p0, \*p1))? -1 : 1;

return (o == 2)? -1: 1;

}

void convexHull(Point points[], int n)

{

int ymin = points[0].y, min = 0;

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

{

int y = points[i].y;

if ((y < ymin) || (ymin == y &&

points[i].x < points[min].x))

ymin = points[i].y, min = i;

}

swap(points[0], points[min]);

p0 = points[0];

qsort(&points[1], n-1, sizeof(Point), compare);

int m = 1; // Initialize size of modified array

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

{

while (i < n-1 && orientation(p0, points[i],

points[i+1]) == 0)

i++;

points[m] = points[i];

m++; // Update size of modified array

}

if (m < 3) return;

stack<Point> S;

S.push(points[0]);

S.push(points[1]);

S.push(points[2]);

for (int i = 3; i < m; i++)

{

while (orientation(nextToTop(S), S.top(), points[i]) != 2)

S.pop();

S.push(points[i]);

}

while (!S.empty())

{

Point p = S.top();

cout << "(" << p.x << ", " << p.y <<")" << endl;

S.pop();

}

}

// Driver program to test above functions

int main()

{

Point points[] = {{0, 3}, {1, 1}, {2, 2}, {4, 4},

{0, 0}, {1, 2}, {3, 1}, {3, 3}};

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

convexHull(points, n);

return 0;

}

**OUTPUT**:



**PROGRAM NO: 06**

**AIM:** Write a program to solve Knapsack problem using Greedy Algorithm.

**SOURCE CODE:**

# include<stdio.h>

void knapsack(int n, float weight[], float profit[], float capacity) {

float x[20], tp = 0;

int i, j, u;

u = capacity;

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

x[i] = 0.0;

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

if (weight[i] > u)

break;

else {

x[i] = 1.0;

tp = tp + profit[i];

u = u - weight[i];

}

}

if (i < n)

x[i] = u / weight[i];

tp = tp + (x[i] \* profit[i]);

printf("\nThe result vector is:- ");

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

printf("%f\t", x[i]);

printf("\nMaximum profit is:- %f", tp);

}

int main() {

float weight[20], profit[20], capacity;

int num, i, j;

float ratio[20], temp;

printf("\nEnter the no. of objects:- ");

scanf("%d", &num);

printf("\nEnter the wts and profits of each object:- ");

for (i = 0; i < num; i++) {

scanf("%f %f", &weight[i], &profit[i]);

}

printf("\nEnter the capacityacity of knapsack:- ");

scanf("%f", &capacity);

for (i = 0; i < num; i++) {

ratio[i] = profit[i] / weight[i];

}

for (i = 0; i < num; i++) {

for (j = i + 1; j < num; j++) {

if (ratio[i] < ratio[j]) {

temp = ratio[j];

ratio[j] = ratio[i];

ratio[i] = temp;

temp = weight[j];

weight[j] = weight[i];

weight[i] = temp;

temp = profit[j];

profit[j] = profit[i];

profit[i] = temp;

}

}

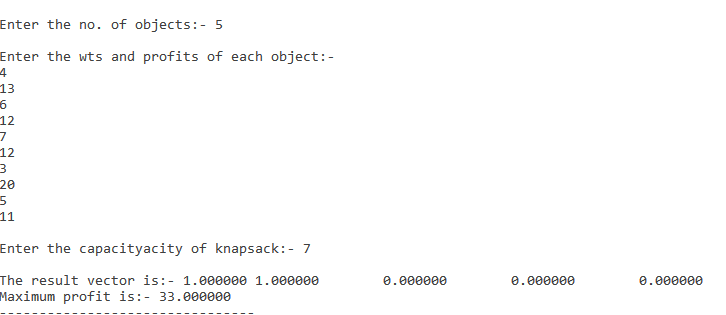
}

knapsack(num, weight, profit, capacity);

return(0);

}

**OUTPUT:**



**PROGRAM NO: 07**

**AIM:** Write a program for implementation of Huffman algorithm.

**Source Code:**

#include<string.h>

#include<stdio.h>

#include<stdlib.h>

typedef struct node

{

char ch;

int freq;

struct node \*left;

struct node \*right;

}node;

node \* heap[100];

int heapSize=0;

void Insert(node \* element)

{

heapSize++;

heap[heapSize] = element;

int now = heapSize;

while(heap[now/2] -> freq > element -> freq)

{

heap[now] = heap[now/2];

now /= 2;

}

heap[now] = element;

}

node \* DeleteMin()

{

node \* minElement,\*lastElement;

int child,now;

minElement = heap[1];

lastElement = heap[heapSize--];

for(now = 1; now\*2 <= heapSize ;now = child)

{

child = now\*2;

if(child != heapSize && heap[child+1]->freq < heap[child] -> freq )

{

child++;

}

if(lastElement -> freq > heap[child] -> freq)

{

heap[now] = heap[child];

}

else

{

break;

}

}

heap[now] = lastElement;

return minElement;

}

void print(node \*temp,char \*code)

{

if(temp->left==NULL && temp->right==NULL)

{

printf("char %c code %s\n",temp->ch,code);

return;

}

int length = strlen(code);

char leftcode[10],rightcode[10];

strcpy(leftcode,code);

strcpy(rightcode,code);

leftcode[length] = '0';

leftcode[length+1] = '\0';

rightcode[length] = '1';

rightcode[length+1] = '\0';

print(temp->left,leftcode);

print(temp->right,rightcode);

}

int main()

{

heap[0] = (node \*)malloc(sizeof(node));

heap[0]->freq = 0;

int n ;

printf("Enter the no of characters: ");

scanf("%d",&n);

printf("Enter the characters and their frequencies: ");

char ch;

int freq,i;

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

{

scanf(" %c",&ch);

scanf("%d",&freq);

node \* temp = (node \*) malloc(sizeof(node));

temp -> ch = ch;

temp -> freq = freq;

temp -> left = temp -> right = NULL;

Insert(temp);

}

if(n==1)

{

printf("char %c code 0\n",ch);

return 0;

}

for(i=0;i<n-1 ;i++)

{

node \* left = DeleteMin();

node \* right = DeleteMin();

node \* temp = (node \*) malloc(sizeof(node));

temp -> ch = 0;

temp -> left = left;

temp -> right = right;

temp -> freq = left->freq + right -> freq;

Insert(temp);

}

node \*tree = DeleteMin();

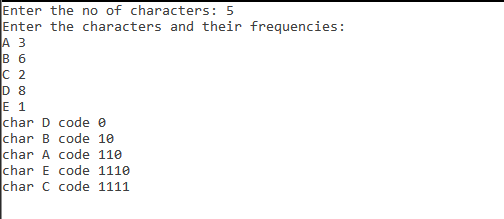
char code[10];

code[0] = '\0';

print(tree,code);

}

**OUTPUT:**



**PROGRAM NO: 08**

**AIM:** Write a program to solve Sequencing Problems with deadlines using Greedy algorithm .

**SOURCE CODE:**

#include <stdio.h>

#define MAX 100

typedef struct Job {

char id[5];

int deadline;

int profit;

} Job;

void jobSequencingWithDeadline(Job jobs[], int n);

int minValue(int x, int y) {

if(x < y) return x;

return y;

}

int main(void) {

//variables

int i, j;

Job jobs[5] = {

{"j1", 2, 60},

{"j2", 1, 100},

{"j3", 3, 20},

{"j4", 2, 40},

{"j5", 1, 20},

};

Job temp;

int n = 5;

for(i = 1; i < n; i++) {

for(j = 0; j < n - i; j++) {

if(jobs[j+1].profit > jobs[j].profit) {

temp = jobs[j+1];

jobs[j+1] = jobs[j];

jobs[j] = temp;

}

}

}

printf("%10s %10s %10s\n", "Job", "Deadline", "Profit");

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

printf("%10s %10i %10i\n", jobs[i].id, jobs[i].deadline, jobs[i].profit);

}

jobSequencingWithDeadline(jobs, n);

return 0;

}

void jobSequencingWithDeadline(Job jobs[], int n) {

int i, j, k, maxprofit;

int timeslot[MAX];

int filledTimeSlot = 0;

int dmax = 0;

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

if(jobs[i].deadline > dmax) {

dmax = jobs[i].deadline;

}

}

for(i = 1; i <= dmax; i++) {

timeslot[i] = -1;

}

printf("dmax: %d\n", dmax);

for(i = 1; i <= n; i++) {

k = minValue(dmax, jobs[i - 1].deadline);

while(k >= 1) {

if(timeslot[k] == -1) {

timeslot[k] = i-1;

filledTimeSlot++;

break;

}

k--;

}

if(filledTimeSlot == dmax) {

break;

}

}

printf("\nRequired Jobs: ");

for(i = 1; i <= dmax; i++) {

printf("%s", jobs[timeslot[i]].id);

if(i < dmax) {

printf(" --> ");

}

}

maxprofit = 0;

for(i = 1; i <= dmax; i++) {

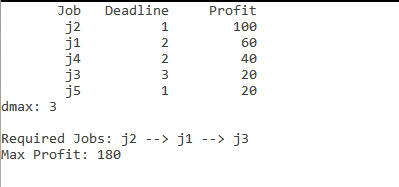
maxprofit += jobs[timeslot[i]].profit;

}

printf("\nMax Profit: %d\n", maxprofit);

}

**OUTPUT:**



**PROGRAM NO: 09**

**AIM:**  Write a program for implementation of Single source shortest path algorithm for Dijkstra’s algorithm.

**SOURCE CODE:**

#include<stdio.h>

#include<conio.h>

#define INFINITY 9999

#define MAX 10

void dijkstra(int G[MAX][MAX],int n,int startnode);

int main()

{

int G[MAX][MAX],i,j,n,u;

printf("Enter no. of vertices:");

scanf("%d",&n);

printf("\nEnter the adjacency matrix:\n");

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

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

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

printf("\nEnter the starting node:");

scanf("%d",&u);

dijkstra(G,n,u);

return 0;

}

void dijkstra(int G[MAX][MAX],int n,int startnode)

{

int cost[MAX][MAX],distance[MAX],pred[MAX];

int visited[MAX],count,mindistance,nextnode,i,j;

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

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

if(G[i][j]==0)

cost[i][j]=INFINITY;

else

cost[i][j]=G[i][j];

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

{

distance[i]=cost[startnode][i];

pred[i]=startnode;

visited[i]=0;

}

distance[startnode]=0;

visited[startnode]=1;

count=1;

while(count<n-1)

{

mindistance=INFINITY;

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

if(distance[i]<mindistance&&!visited[i])

{

mindistance=distance[i];

nextnode=i;

}

visited[nextnode]=1;

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

if(!visited[i])

if(mindistance+cost[nextnode][i]<distance[i])

{

distance[i]=mindistance+cost[nextnode][i];

pred[i]=nextnode;

}

count++;

}

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

if(i!=startnode)

{

printf("\nDistance of node%d=%d",i,distance[i]);

printf("\nPath=%d",i);

j=i;

do

{

j=pred[j];

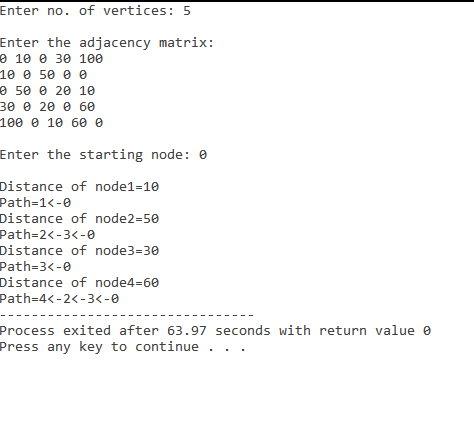
printf("<-%d",j);

}while(j!=startnode);

}

}

**OUTPUT:**



**PROGRAM NO: 10**

**AIM:** Implement Graph on two-dimensional array and use Greedy method to obtain minimum-cost spanning tree of the graph.

**(A) Using Prim’s:**

**SOURCE CODE:**

#include<stdio.h>

#include<conio.h>

int a,b,u,v,n,i,j,ne=1;

int visited[10]= {

0

}

,min,mincost=0,cost[10][10];

int main() {

printf("\n Enter the number of nodes:");

scanf("%d",&n);

printf("\n Enter the adjacency matrix:\n");

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

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

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

if(cost[i][j]==0)

cost[i][j]=999;

}

visited[1]=1;

printf("\n");

while(ne<n) {

for (i=1,min=999;i<=n;i++)

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

if(cost[i][j]<min)

if(visited[i]!=0) {

min=cost[i][j];

a=u=i;

b=v=j;

}

if(visited[u]==0 || visited[v]==0) {

printf("\n Edge %d:(%d %d) cost:%d",ne++,a,b,min);

mincost+=min;

visited[b]=1;

}

cost[a][b]=cost[b][a]=999;

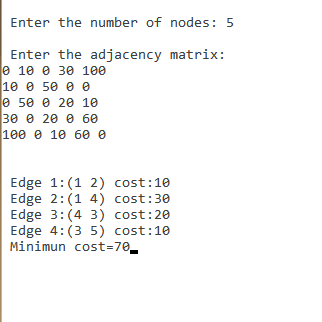
}

printf("\n Minimun cost=%d",mincost);

getch();

}

**OUTPUT:**



**(B) Using krushkals :**

**SOURCE CODE:**

#include<stdio.h>

#include<conio.h>

#include<stdlib.h>

int i,j,k,a,b,u,v,n,ne=1;

int min,mincost=0,cost[9][9],parent[9];

int find(int);

int uni(int,int);

int main()

{

printf("\n\tImplementation of Kruskal's algorithm\n");

printf("\nEnter the no. of vertices:");

scanf("%d",&n);

printf("\nEnter the cost adjacency matrix:\n");

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

{

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

{

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

if(cost[i][j]==0)

cost[i][j]=999;

}

}

printf("The edges of Minimum Cost Spanning Tree are\n");

while(ne < n)

{

for(i=1,min=999;i<=n;i++)

{

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

{

if(cost[i][j] < min)

{

min=cost[i][j];

a=u=i;

b=v=j;

}

}

}

u=find(u);

v=find(v);

if(uni(u,v))

{

printf("%d edge (%d,%d) =%d\n",ne++,a,b,min);

mincost +=min;

}

cost[a][b]=cost[b][a]=999;

}

printf("\n\tMinimum cost = %d\n",mincost);

getch();

}

int find(int i)

{

while(parent[i])

i=parent[i];

return i;

}

int uni(int i,int j)

{

if(i!=j)

{

parent[j]=i;

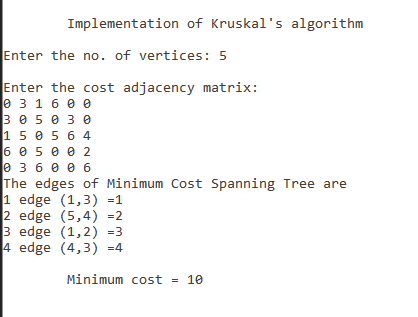
return 1;

}

return 0;

}

**OUTPUT:**



**Program No: 11**

**Aim:** Write a program to find a number in an array by binary search method.

**Source Code:**

#include <stdio.h>

int binarySearch(int arr[], int l, int r, int x)

{

if (r >= l) {

int mid = l + (r - l) / 2;

if (arr[mid] == x)

return mid;

if (arr[mid] > x)

return binarySearch(arr, l, mid - 1, x);

return binarySearch(arr, mid + 1, r, x);

}

return -1;

}

int main(void)

{

int arr[] = { 2, 3, 4, 10, 40 };

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

int x = 10;

int result = binarySearch(arr, 0, n - 1, x);

(result == -1) ? printf("Element is not present in array")

: printf("Element is present at index %d", result);

return 0;

}

**Output:**



**Program No: 12**

**Aim:** Write a program of all pairs shortest path using Dynamic programming.

**Source Code:**

#include<stdio.h>

#define V 4

#define INF 99999

void printSolution(int dist[][V]);

void floydWarshall (int graph[][V])

{

int dist[V][V], i, j, k;

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

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

dist[i][j] = graph[i][j];

for (k = 0; k < V; k++)

{

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

{

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

{

if (dist[i][k] + dist[k][j] < dist[i][j])

dist[i][j] = dist[i][k] + dist[k][j];

}

}

}

printSolution(dist);

}

void printSolution(int dist[][V])

{

printf ("The following matrix shows the shortest distances"

" between every pair of vertices \n");

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

{

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

{

if (dist[i][j] == INF)

printf("%7s", "INF");

else

printf ("%7d", dist[i][j]);

}

printf("\n");

}

}

int main()

{

int graph[V][V] = { {0, 5, INF, 10},

{INF, 0, 3, INF},

{INF, INF, 0, 1},

{INF, INF, INF, 0}

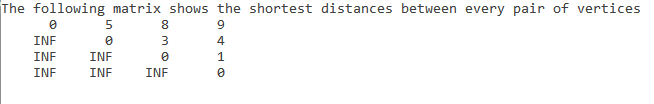
};

floydWarshall(graph);

return 0;

}

**Output:**

****

**Program No: 13**

**Aim:** Write a program for optimal binary search of an element in an array using Dynamic programming.

**Source Code:**

#include <stdio.h>

#include <limits.h>

int sum(int freq[], int i, int j);

int optCost(int freq[], int i, int j)

{

if (j < i)

return 0;

if (j == i)

return freq[i];

int fsum = sum(freq, i, j);

int min = INT\_MAX;

for (int r = i; r <= j; ++r)

{

int cost = optCost(freq, i, r-1) +

optCost(freq, r+1, j);

if (cost < min)

min = cost;

}

return min + fsum;

}

int optimalSearchTree(int keys[], int freq[], int n)

{

return optCost(freq, 0, n-1);

}

int sum(int freq[], int i, int j)

{

int s = 0;

for (int k = i; k <=j; k++)

s += freq[k];

return s;

}

int main()

{

int keys[] = {10, 12, 20};

int freq[] = {34, 8, 50};

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

printf("Cost of Optimal BST is %d ",

optimalSearchTree(keys, freq, n));

return 0;

}

**Output:**

****

**Program No: 14**

**Aim:** Write a program for 0/1 Knapsack problem using Dynamic programming.

**Source Code:**

#include <stdio.h>

int max(int a, int b) { return (a > b) ? a : b; }

int knapSack(int W, int wt[], int val[], int n)

{

if (n == 0 || W == 0)

return 0;

if (wt[n - 1] > W)

return knapSack(W, wt, val, n - 1);

else

return max(

val[n - 1] + knapSack(W - wt[n - 1], wt, val, n - 1),

knapSack(W, wt, val, n - 1));

}

int main()

{

int val[] = { 60, 100, 120 };

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

int W = 50;

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

printf("%d", knapSack(W, wt, val, n));

return 0;

}

**Output:**

****

**Program No: 15**

**Aim:** Write a program for Matrix-Chain Multiplication using Dynamic programming.

**Source Code:**

#include <limits.h>

#include <stdio.h>

int MatrixChainOrder(int p[], int n)

{

int m[n][n];

int i, j, k, L, q;

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

m[i][i] = 0;

for (L = 2; L < n; L++) {

for (i = 1; i < n - L + 1; i++) {

j = i + L - 1;

m[i][j] = INT\_MAX;

for (k = i; k <= j - 1; k++) {

q = m[i][k] + m[k + 1][j] + p[i - 1] \* p[k] \* p[j];

if (q < m[i][j])

m[i][j] = q;

}

}

}

return m[1][n - 1];

}

int main()

{

int arr[] = { 1, 2, 3,4,5,6 };

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

printf("Minimum number of multiplications is %d ",

MatrixChainOrder(arr, size));

getchar();

return 0;

}

**Output:**



**Program No: 16**

**Aim:** Write a program to find the Largest Common Subsequences of two sets using Dynamic programming.

**Source Code:**

#include<bits/stdc++.h>

int max(int a, int b);

int lcs( char \*X, char \*Y, int m, int n )

{

int L[m+1][n+1];

int i, j;

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

{

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

{

if (i == 0 || j == 0)

L[i][j] = 0;

else if (X[i-1] == Y[j-1])

L[i][j] = L[i-1][j-1] + 1;

else

L[i][j] = max(L[i-1][j], L[i][j-1]);

}

}

return L[m][n];

}

int max(int a, int b)

{

return (a > b)? a : b;

}

int main()

{

char X[] = "AGGTAB";

char Y[] = "GXTXAYB";

int m = strlen(X);

int n = strlen(Y);

printf("Length of LCS is %d", lcs( X, Y, m, n ) );

return 0;

}

**Output:**

****

**Program No: 17**

**Aim:** Write a program for travelling salesman problem.

**Source Code:**

#include <stdio.h>

int matrix[25][25], visited\_cities[10], limit, cost = 0;

int tsp(int c)

{

int count, nearest\_city = 999;

int minimum = 999, temp;

for(count = 0; count < limit; count++)

{

if((matrix[c][count] != 0) && (visited\_cities[count] == 0))

{

if(matrix[c][count] < minimum)

{

minimum = matrix[count][0] + matrix[c][count];

}

temp = matrix[c][count];

nearest\_city = count;

}

}

if(minimum != 999)

{

cost = cost + temp;

}

return nearest\_city;

}

void minimum\_cost(int city)

{

int nearest\_city;

visited\_cities[city] = 1;

printf("%d ", city + 1);

nearest\_city = tsp(city);

if(nearest\_city == 999)

{

nearest\_city = 0;

printf("%d", nearest\_city + 1);

cost = cost + matrix[city][nearest\_city];

return;

}

minimum\_cost(nearest\_city);

}

int main()

{

int i, j;

printf("Enter Total Number of Cities:\t");

scanf("%d", &limit);

printf("\nEnter Cost Matrix\n");

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

{

printf("\nEnter %d Elements in Row[%d]\n", limit, i + 1);

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

{

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

}

visited\_cities[i] = 0;

}

printf("\nEntered Cost Matrix\n");

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

{

printf("\n");

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

{

printf("%d ", matrix[i][j]);

}

}

printf("\n\nPath:\t");

minimum\_cost(0);

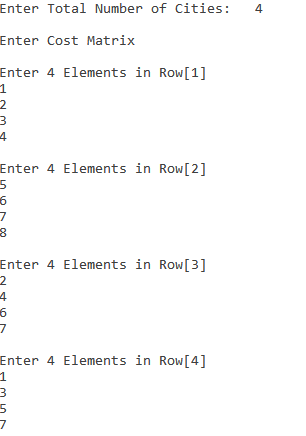
printf("\n\nMinimum Cost: \t");

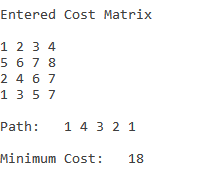
printf("%d\n", cost);

return 0;

}

**Output:**

****

****

**Program No:** 18

**Aim:** Write a program for Graph Traversal Algorithms : Depth First Search(DFS).

**Source Code:**

#include<stdio.h>

#include<stdlib.h>

typedef struct node

{

struct node \*next;

int vertex;

}node;

node \*G[20];

//heads of linked list

int visited[20];

int n;

void read\_graph();

//create adjacency list

void insert(int,int);

//insert an edge (vi,vj) in te adjacency list

void DFS(int);

int main()

{

int i;

read\_graph();

//initialised visited to 0

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

visited[i]=0;

DFS(0);

}

void DFS(int i)

{

node \*p;

printf("\n%d",i);

p=G[i];

visited[i]=1;

while(p!=NULL)

{

i=p->vertex;

if(!visited[i])

DFS(i);

p=p->next;

}

}

void read\_graph()

{

int i,vi,vj,no\_of\_edges;

printf("Enter number of vertices:");

scanf("%d",&n);

//initialise G[] with a null

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

{

G[i]=NULL;

//read edges and insert them in G[]

printf("Enter number of edges:");

scanf("%d",&no\_of\_edges);

for(i=0;i<no\_of\_edges;i++)

{

printf("Enter an edge(u,v):");

scanf("%d%d",&vi,&vj);

insert(vi,vj);

}

}

void insert(int vi,int vj)

{

node \*p,\*q;

//acquire memory for the new node

q=(node\*)malloc(sizeof(node));

q->vertex=vj;

q->next=NULL;

//insert the node in the linked list number vi

if(G[vi]==NULL)

G[vi]=q;

else

{

//go to end of the linked list

p=G[vi]

while(p->next!=NULL)

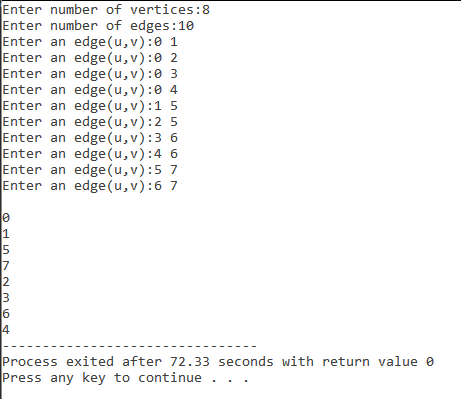
p=p->next;

p->next=q;

}

}

**Output:**



**Program No:** 19

**Aim:** Write a program for Graph Traversal Algorithm: Breadth First Search (BFS).

**Source Code:**

#include<stdio.h>

#include<stdlib.h>

#define MAX 100

#define initial 1

#define waiting 2

#define visited 3

int n;

int adj[MAX][MAX];

int state[MAX];

void create\_graph();

void BF\_Traversal();

void BFS(int v);

int queue[MAX], front = -1,rear = -1;

void insert\_queue(int vertex);

int delete\_queue();

int isEmpty\_queue();

int main()

{

create\_graph();

BF\_Traversal();

return 0;

}

void BF\_Traversal()

{

int v;

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

state[v] = initial;

printf("Enter Start Vertex for BFS: \n");

scanf("%d", &v);

BFS(v);

}

void BFS(int v)

{

int i;

insert\_queue(v);

state[v] = waiting;

while(!isEmpty\_queue())

{

v = delete\_queue( );

printf("%d ",v);

state[v] = visited;

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

{

if(adj[v][i] == 1 && state[i] == initial)

{

insert\_queue(i);

state[i] = waiting;

}

}

}

printf("\n");

}

void insert\_queue(int vertex)

{

if(rear == MAX-1)

printf("Queue Overflow\n");

else

{

if(front == -1)

front = 0;

rear = rear+1;

queue[rear] = vertex ;

}

}

int isEmpty\_queue()

{

if(front == -1 || front > rear)

return 1;

else

return 0;

}

int delete\_queue()

{

int delete\_item;

if(front == -1 || front > rear)

{

printf("Queue Underflow\n");

exit(1);

}

delete\_item = queue[front];

front = front+1;

return delete\_item;

}

void create\_graph()

{

int count,max\_edge,origin,destin;

printf("Enter number of vertices : ");

scanf("%d",&n);

max\_edge = n\*(n-1);

for(count=1; count<=max\_edge; count++)

{

printf("Enter edge %d( -1 -1 to quit ) : ",count);

scanf("%d %d",&origin,&destin);

if((origin == -1) && (destin == -1))

break;

if(origin>=n || destin>=n || origin<0 || destin<0)

{

printf("Invalid edge!\n");

count--;

}

else

{

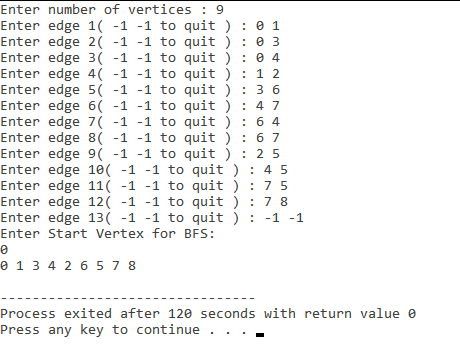
adj[origin][destin] = 1;

}

}

}

**Output:**



**Program No: 20**

**Aim:** Write a program for solution space for 8 queen problem and solve the problem using Back-Tracking method.

**Source Code:**

#include<iostream>

using namespace std;

#define N 8

void printBoard(int board[N][N]) {

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

for (int j = 0; j < N; j++)

cout << board[i][j] << " ";

cout << endl;

}

}

bool isValid(int board[N][N], int row, int col) {

for (int i = 0; i < col; i++) //check whether there is queen in the left or not

if (board[row][i])

return false;

for (int i=row, j=col; i>=0 && j>=0; i--, j--)

if (board[i][j]) //check whether there is queen in the left upper diagonal or not

return false;

for (int i=row, j=col; j>=0 && i<N; i++, j--)

if (board[i][j]) //check whether there is queen in the left lower diagonal or not

return false;

return true;

}

bool solveNQueen(int board[N][N], int col) {

if (col >= N) //when N queens are placed successfully

return true;

for (int i = 0; i < N; i++) { //for each row, check placing of queen is possible or not

if (isValid(board, i, col) ) {

board[i][col] = 1; //if validate, place the queen at place (i, col)

if ( solveNQueen(board, col + 1)) //Go for the other columns recursively

return true;

board[i][col] = 0; //When no place is vacant remove that queen

}

}

return false; //when no possible order is found

}

bool checkSolution() {

int board[N][N];

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

for(int j = 0; j<N; j++)

board[i][j] = 0; //set all elements to 0

if ( solveNQueen(board, 0) == false ) { //starting from 0th column

cout << "Solution does not exist";

return false;

}

printBoard(board);

return true;

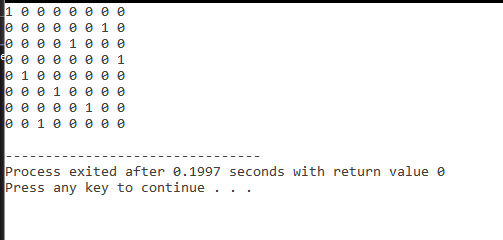
}

int main() {

checkSolution();

}

**Output:**

****

**Program No: 21**

**Aim:** Write a program for Sum of Subset problem of a given set using back tracking method.

**Source Code:**

#include <stdio.h>

#include <stdlib.h>

#define ARRAYSIZE(a) (sizeof(a))/(sizeof(a[0]))

static int total\_nodes;

// prints subset found

void printSubset(int A[], int size)

{

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

{

printf("%\*d", 5, A[i]);

}

printf("n");

}

// inputs

// s - set vector

// t - tuplet vector

// s\_size - set size

// t\_size - tuplet size so far

// sum - sum so far

// ite - nodes count

// target\_sum - sum to be found

void subset\_sum(int s[], int t[],

int s\_size, int t\_size,

int sum, int ite,

int const target\_sum)

{

total\_nodes++;

if( target\_sum == sum )

{

// We found subset

printSubset(t, t\_size);

// Exclude previously added item and consider next candidate

subset\_sum(s, t, s\_size, t\_size-1, sum - s[ite], ite + 1, target\_sum);

return;

}

else

{

// generate nodes along the breadth

for( int i = ite; i < s\_size; i++ )

{

t[t\_size] = s[i];

// consider next level node (along depth)

subset\_sum(s, t, s\_size, t\_size + 1, sum + s[i], i + 1, target\_sum);

}

}

}

// Wrapper to print subsets that sum to target\_sum

// input is weights vector and target\_sum

void generateSubsets(int s[], int size, int target\_sum)

{

int \*tuplet\_vector = (int \*)malloc(size \* sizeof(int));

subset\_sum(s, tuplet\_vector, size, 0, 0, 0, target\_sum);

free(tuplet\_vector);

}

int main()

{

int weights[] = {10, 7, 5, 18, 12, 20, 15};

int size = ARRAYSIZE(weights);

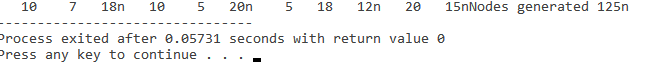
generateSubsets(weights, size, 35);

printf("Nodes generated %dn", total\_nodes);

return 0;

}

**Output:**

****

**Program No: 22**

**Aim:** Write a program for string matching by (i) Naive-string matching method and (ii) Rabin-Krap algorithm and compare number of operations done in these methods.

**Source Code:**

**(i)** Naive-string matching method:-

#include<stdio.h>

#include<string.h>

void search(char \*pat, char \*txt)

{

int M = strlen(pat);

int N = strlen(txt);

/\* A loop to slide pat[] one by one \*/

for (int i = 0; i <= N - M; i++)

{

int j;

/\* For current index i, check for pattern match \*/

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

{

if (txt[i + j] != pat[j])

break;

}

if (j == M) // if pat[0...M-1] = txt[i, i+1, ...i+M-1]

{

printf("Pattern found at index %d \n", i);

}

}

}

/\* Driver program to test above function \*/

int main()

{

char \*txt = "AABAACAADAABAAABAA";

char \*pat = "AABA";

search(pat, txt);

return 0;

}

**(ii)** ) Rabin-Krap algorithm:-

#include <stdio.h>

#include <string.h>

// d is the number of characters in the input alphabet

#define d 256

/\* pat -> pattern

txt -> text

q -> A prime number

\*/

void search(char pat[], char txt[], int q)

{

int M = strlen(pat);

int N = strlen(txt);

int i, j;

int p = 0; // hash value for pattern

int t = 0; // hash value for txt

int h = 1;

// The value of h would be "pow(d, M-1)%q"

for (i = 0; i < M - 1; i++)

h = (h \* d) % q;

// Calculate the hash value of pattern and first

// window of text

for (i = 0; i < M; i++) {

p = (d \* p + pat[i]) % q;

t = (d \* t + txt[i]) % q;

}

// Slide the pattern over text one by one

for (i = 0; i <= N - M; i++) {

// Check the hash values of current window of text

// and pattern. If the hash values match then only

// check for characters on by one

if (p == t) {

/\* Check for characters one by one \*/

for (j = 0; j < M; j++) {

if (txt[i + j] != pat[j])

break;

}

// if p == t and pat[0...M-1] = txt[i, i+1, ...i+M-1]

if (j == M)

printf("Pattern found at index %d \n", i);

}

// Calculate hash value for next window of text: Remove

// leading digit, add trailing digit

if (i < N - M) {

t = (d \* (t - txt[i] \* h) + txt[i + M]) % q;

// We might get negative value of t, converting it

// to positive

if (t < 0)

t = (t + q);

}

}

}

/\* Driver program to test above function \*/

int main()

{

char txt[] = "GEEKS FOR GEEKS";

char pat[] = "GEEK";

int q = 101; // A prime number

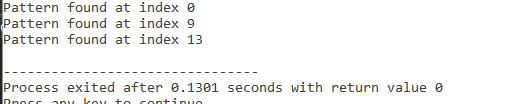
search(pat, txt, q);

return 0;

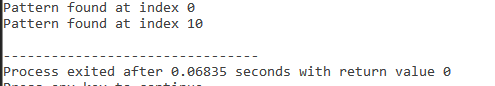
}

**Output:**

Naive-string matching method:-

****

Rabin-Krap algorithm:-



**Program No: 23**

**Aim:** Write a program for string matching using Knuth-Morris-Pratt Algorithms.

**Source Code:**

#include <bits/stdc++.h>

void computeLPSArray(char\* pat, int M, int\* lps);

void KMPSearch(char\* pat, char\* txt)

{

int M = strlen(pat);

int N = strlen(txt);

int lps[M];

computeLPSArray(pat, M, lps);

int i = 0; // index for txt[]

int j = 0; // index for pat[]

while (i < N) {

if (pat[j] == txt[i]) {

j++;

i++;

}

if (j == M) {

printf("Found pattern at index %d ", i - j);

j = lps[j - 1];

}

// mismatch after j matches

else if (i < N && pat[j] != txt[i]) {

// Do not match lps[0..lps[j-1]] characters,

// they will match anyway

if (j != 0)

j = lps[j - 1];

else

i = i + 1;

}

}

}

// Fills lps[] for given patttern pat[0..M-1]

void computeLPSArray(char\* pat, int M, int\* lps)

{

// length of the previous longest prefix suffix

int len = 0;

lps[0] = 0; // lps[0] is always 0

// the loop calculates lps[i] for i = 1 to M-1

int i = 1;

while (i < M) {

if (pat[i] == pat[len]) {

len++;

lps[i] = len;

i++;

}

else // (pat[i] != pat[len])

{

// This is tricky. Consider the example.

// AAACAAAA and i = 7. The idea is similar

// to search step.

if (len != 0) {

len = lps[len - 1];

}

else // if (len == 0)

{

lps[i] = 0;

i++;

}

}

}

}

int main()

{

char txt[] = "ABABDABACDABABCABAB";

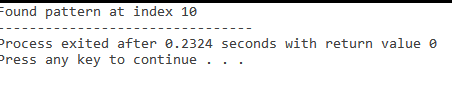
char pat[] = "ABABCABAB";

KMPSearch(pat, txt);

return 0;

}

**Output:**

****

**Program No: 24**

**Aim:** Write a program for string matching using finite Automata method and Knuth-Morris-Pratt Algorithms.

**Source Code:**

**(i)** Finite Automata method

#include<stdio.h>

#include<string.h>

#define NO\_OF\_CHARS 256

int getNextState(char \*pat, int M, int state, int x)

{

// If the character c is same as next character

// in pattern,then simply increment state

if (state < M && x == pat[state])

return state+1;

// ns stores the result which is next state

int ns, i;

// ns finally contains the longest prefix

// which is also suffix in "pat[0..state-1]c"

// Start from the largest possible value

// and stop when you find a prefix which

// is also suffix

for (ns = state; ns > 0; ns--)

{

if (pat[ns-1] == x)

{

for (i = 0; i < ns-1; i++)

if (pat[i] != pat[state-ns+1+i])

break;

if (i == ns-1)

return ns;

}

}

return 0;

}

/\* This function builds the TF table which represents4

Finite Automata for a given pattern \*/

void computeTF(char \*pat, int M, int TF[][NO\_OF\_CHARS])

{

int state, x;

for (state = 0; state <= M; ++state)

for (x = 0; x < NO\_OF\_CHARS; ++x)

TF[state][x] = getNextState(pat, M, state, x);

}

/\* Prints all occurrences of pat in txt \*/

void search(char \*pat, char \*txt)

{

int M = strlen(pat);

int N = strlen(txt);

int TF[M+1][NO\_OF\_CHARS];

computeTF(pat, M, TF);

// Process txt over FA.

int i, state=0;

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

{

state = TF[state][txt[i]];

if (state == M)

printf ("\n Pattern found at index %d",

i-M+1);

}

}

// Driver program to test above function

int main()

{

char \*txt = "AABAACAADAABAAABAA";

char \*pat = "AABA";

search(pat, txt);

return 0;

}

**(ii)** Knuth-Morris-Pratt Algorithms

#include <bits/stdc++.h>

void computeLPSArray(char\* pat, int M, int\* lps);

void KMPSearch(char\* pat, char\* txt)

{

int M = strlen(pat);

int N = strlen(txt);

int lps[M];

computeLPSArray(pat, M, lps);

int i = 0; // index for txt[]

int j = 0; // index for pat[]

while (i < N) {

if (pat[j] == txt[i]) {

j++;

i++;

}

if (j == M) {

printf("Found pattern at index %d ", i - j);

j = lps[j - 1];

}

// mismatch after j matches

else if (i < N && pat[j] != txt[i]) {

// Do not match lps[0..lps[j-1]] characters,

// they will match anyway

if (j != 0)

j = lps[j - 1];

else

i = i + 1;

}

}

}

// Fills lps[] for given patttern pat[0..M-1]

void computeLPSArray(char\* pat, int M, int\* lps)

{

// length of the previous longest prefix suffix

int len = 0;

lps[0] = 0; // lps[0] is always 0

// the loop calculates lps[i] for i = 1 to M-1

int i = 1;

while (i < M) {

if (pat[i] == pat[len]) {

len++;

lps[i] = len;

i++;

}

else // (pat[i] != pat[len])

{

// This is tricky. Consider the example.

// AAACAAAA and i = 7. The idea is similar

// to search step.

if (len != 0) {

len = lps[len - 1];

}

else // if (len == 0)

{

lps[i] = 0;

i++;

}

}

}

}

int main()

{

char txt[] = "ABABDABACDABABCABAB";

char pat[] = "ABABCABAB";

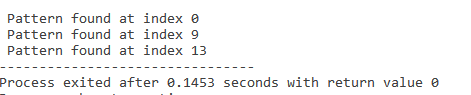
KMPSearch(pat, txt);

return 0;

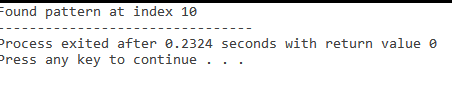
}

**Output:**

Finite Automata method:-

****

Knuth-Morris-Pratt Algorithms:-

****