**PRACTICAL FILE**

****

**Design and Analysis of Algorithms Lab.**

(**MCA 261)**

**SUBMITTED BY SUBMITTED TO**

Abhishek Tyagi Ms. Sonia Batra

03311104422 Asst. Professor

MCA 3rd Sem

**BANARSIDAS CHANDIWALA INSTITUTE OF INFORMATION TECHNOLOGY *AFFILIATED WITH***

**GURU GOBIND SINGH INDRAPRASTH UNIVERSITY, DELHI**

|  |  |  |  |
| --- | --- | --- | --- |
| S.No | Title | Pg.no | Sign |
| 1 | Write a program to implement randomised quicksort | 3 |  |
| 2 | Write a program to implement randomised merge sort | 6 |  |
| 3 | Write a program to find a substring in a string using naive string matching algorithm | 9 |  |
| 4 | Write a program to find a substring in a string using the Rabin Karp algorithm | 11 |  |
| 5 | Write a program to find a substring in a string using KMP algorithm for string matching | 13 |  |
| 6 | Write a program for the fractional knapsack problem | 16 |  |
| 7 | Write a program for the 0/1 knapsack problem | 20 |  |
| 8 | Write a program to find the minimum cost spanning tree of a given undirected graph using Prim's algorithm | 23 |  |
| 9 | Write a program to implement all pairs shortest path problem using Floyd's algorithm | 25 |  |
| 10 | Write a program to implement some of subsets problem | 28 |  |
| 11 | Write a program to implement n Queens problem using backtracking | 30 |  |

**Q1. Write a program to implement randomised Quicksort.**

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

// Function to swap two elements in an array

void swap(int arr[], int i, int j) {

int temp = arr[i];

arr[i] = arr[j];

arr[j] = temp;

}

// Function to partition the array and return the pivot index

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

int pivot = arr[high];

int i = low - 1;

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

if (arr[j] <= pivot) {

i++;

swap(arr, i, j);

}

}

swap(arr, i + 1, high);

return i + 1;

}

// Function to generate random pivot and call partition function

int randm\_partition(int arr[], int low, int high) {

srand(time(NULL));

int random = low + rand() % (high - low);

swap(arr, random, high);

return partition(arr, low, high);

}

// Function to perform randomized QuickSort

void randm\_quicksort(int arr[], int low, int high) {

if (low < high) {

int pivot = randm\_partition(arr, low, high);

randm\_quicksort(arr, low, pivot - 1);

randm\_quicksort(arr, pivot + 1, high);

}

}

int main() {

int n;

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

scanf("%d", &n);

int arr[n];

// Generating n random elements

srand(time(NULL));

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

arr[i] = rand() ;

}

printf("Generated Numbers : ");

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

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

}

clock\_t start, end;

double time\_used;

start = clock();

randm\_quicksort(arr, 0, n - 1);

end = clock();

time\_used = ((double)(end - start)) / CLOCKS\_PER\_SEC;

printf("\nSorted array: ");

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

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

}

printf("\n");

printf("\nTime taken: %f seconds\n", time\_used);

return 0;

}

**Output**

A close up of numbers

Description automatically generated

**2. Write a program to implement randomised merge sort .**

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

void merge(int arr[], int low, int mid, int high) {

int i, j, k;

int n1 = mid - low + 1;

int n2 = high - mid;

int L[n1], R[n2];

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

L[i] = arr[low + i];

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

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

i = 0;

j = 0;

k = low;

while (i < n1 && j < n2) {

if (L[i] <= R[j]) {

arr[k] = L[i];

i++;

} else {

arr[k] = R[j];

j++;

}

k++;

}

while (i < n1) {

arr[k] = L[i];

i++;

k++;

}

while (j < n2) {

arr[k] = R[j];

j++;

k++;

}

}

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

if (low < high) {

int mid = low + (high - low) / 2;

merge\_sort(arr, low, mid);

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

merge(arr, low, mid, high);

}

}

int main() {

int n, int arr[n];

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

scanf("%d", &n);

srand(time(NULL));

printf("Generated Numbers : ");

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

arr[i] = rand() ;

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

}

clock\_t start, end;

double time\_used;

start = clock();

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

end = clock();

time\_used = ((double)(end - start)) / CLOCKS\_PER\_SEC;

printf("\nSorted array: ");

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

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

}

printf("\nTime taken: %f seconds\n", time\_used);

return 0;

}

**Output**

A close up of numbers

Description automatically generated

**3. Write a program to find a substring in a string using Naive String-Matching Algorithm.**

#include <stdio.h>

#include <string.h>

int search(char \*pat, char \*text) {

int M = strlen(pat);

int N = strlen(text);

int i, j;

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

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

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

break;

}

if (j == M)

return i;

}

return -1;

}

int main() {

char text[100], pat[100];

int index;

printf("\nEnter the text: ");

fgets(text, 100, stdin);

printf("\nEnter the pattern: ");

fgets(pat, 100, stdin);

// Remove newline characters from input

text[strcspn(text, "\n")] = '\0';

pat[strcspn(pat, "\n")] = '\0';

index = search(pat, text);

if (index >= 0) {

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

} else {

printf("Pattern not found\n");

}

return 0 ;

}

**Output**

A screen shot of a computer

Description automatically generated

A close up of text

Description automatically generated

**Q4. Write a program to find a substring in a string using the Rabin Karp Algorithm.**

#include <stdio.h>

#include <string.h>

#define d 256

void search(char \*pat, char \*text, int q) {

int M = strlen(pat);

int N = strlen(text);

int i, j;

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

int p = 0;

int t = 0;

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

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

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

}

if (p == t) {

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

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

break;

}

if (j == M) {

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

}

else{

printf("Pattern not found!");

}

}

}

}

int main() {

char text[100], pat[100];

int q = 101;

printf("Enter the text: ");

fgets(text, 100, stdin);

printf("Enter the pattern: ");

fgets(pat, 100, stdin);

// Remove newline characters from input

text[strcspn(text, "\n")] = '\0';

pat[strcspn(pat, "\n")] = '\0';

search(pat, text, q);

return 0;

}

**Output**

A screenshot of a computer code

Description automatically generated

A close up of text

Description automatically generated

**Q5. Write a program to find a substring in a string using KMP Algorithm for String Matching.**

#include <stdio.h>

#include<string.h>

#include<stdlib.h>

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

{

int len=0;

int i;

lps[0]=0;

i=1;

while(i<M)

{

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

{

len++;

lps[i]=len;

i++;

}

else

{

if(len!=0)

{

len=lps[len-1];

}

else

{

lps[i]=0;

i++;

}

}

}

}

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

{

int M=strlen(pat);

int N=strlen(txt);

int \*lps=(int\*)malloc(sizeof(int)\*M);

int j=0;

computeLPSArray(pat,M,lps);

int i=0;

while(i<N)

{

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

{

j++;

i++;

}

if(j==M)

{

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

j=lps[j-1];

}

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

{

if(j!=0)

{

j=lps[j-1];

}

else

{

i=i+1;

}

}

}

free(lps);

}

int main()

{

char txt[20];

char pat[10];

printf("Enter the text: ");

scanf("%s",&txt);

printf("Enter the pattern: ");

scanf("%s",&pat);

KMPSearch(pat,txt);

return 0;

}

**Output**

A screenshot of a computer

Description automatically generated

**Q6. Write a program for the Fractional Knapsack problem.**

#include<stdio.h>

void main (){

int n, m, w[100], p[100], ratio[100] , i, j, u, temp;

float xr, x[100], total\_profit=0, total\_weight=0;

printf ("Enter the number of items(n): ");

scanf ("%d", &n);

printf ("Enter the capacity of the Knapsack(m): ");

scanf ("%d", &m);

//Initializing remaining capacity of Knapsack (u)

u = m;

//Initializing Solution Array x[]

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

x[i]=0;

}

//Reading the Weights

printf ("Enter the Weights of items: ");

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

printf ("\n\tWeight of item %d = ", i + 1);

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

}

printf ("\nEnter the Profit Values of items: ");

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

printf ("\n\tProfit of item %d = ", i + 1);

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

}

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

ratio[i] = p[i] / w[i];

}

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

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

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

temp = ratio[i];

ratio[i] = ratio[j];

ratio[j] = temp;

temp = w[i];

w[i] = w[j];

w[j] = temp;

temp = p[i];

p[i] = p[j];

p[j] = temp;

}

}

}

printf("\n The Table After Sorting based on the Ratio: \n");

//Printing Item numbers

printf("\nItem:\t\t");

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

printf("%d\t",i+1);

}

printf("\nProfit:\t\t");

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

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

}

printf("\nWeights:\t");

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

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

}

printf ("\nRATIO:\t\t");

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

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

}

//Calculating Solution Array x

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

if(w[i]<=u){

x[i]=1;

u=u-w[i];

}

else if(w[i]>u){

break;

}

}

if(i<=n){

xr = (float)u/w[i];

x[i] = xr;

}

//Printing Solution Array x

printf("\n X = [");

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

printf("%.3f , ",x[i]);

}

printf("]");

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

total\_profit += x[i]\*p[i];

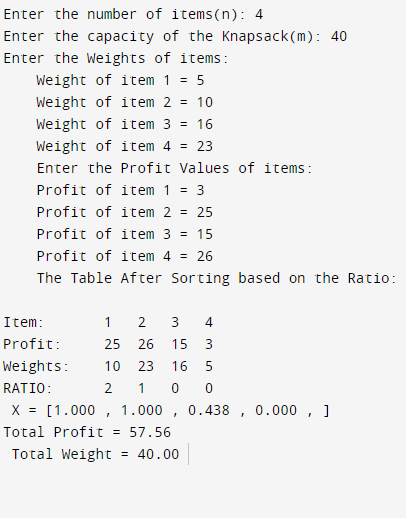
total\_weight += x[i]\*w[i];

}

printf("\nTotal Profit = %.2f \n Total Weight = %.2f ",total\_profit,total\_weight);

}

**Output**



**Q7. Write a program for the 0/1 Knapsack Problem.**

#include<stdio.h>

int w[10], p[10], v[10][10], n, i, j, cap, x[10] = {0};

int max(int i, int j) {

return ((i > j) ? i : j);

}

int knap(int i, int j) {

int value;

if (v[i][j] < 0) {

if (j < w[i])

value = knap(i - 1, j);

else

value = max(knap(i - 1, j), p[i] + knap(i - 1, j - w[i]));

v[i][j] = value;

}

return (v[i][j]);

}

void main() {

int profit, count = 0;

printf("\nEnter the number of elements: ");

scanf("%d", &n);

printf("\nEnter the profit and weights of the elements\n");

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

printf("Item %d : ", i);

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

}

printf("\nEnter the capacity: ");

scanf("%d", &cap);

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

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

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

v[i][j] = 0;

else

v[i][j] = -1;

profit = knap(n, cap);

i = n;

j = cap;

while (j != 0 && i != 0) {

if (v[i][j] != v[i - 1][j]) {

x[i] = 1;

j = j - w[i];

i--;

} else

i--;

}

printf("\nItems included are: \n");

printf("Item\tWeight\tProfit\n");

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

if (x[i])

printf("%d\t%d\t%d\n", ++count, w[i], p[i]);

printf("Total profit = %d\n", profit);

}

**Output**

A screenshot of a computer

Description automatically generated

**Q8. Find Minimum Cost Spanning Tree of a given undirected graph using Prim’s algorithm.**

#include<stdio.h>

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

int visited[10] = {0}, min, mincost = 0, cost[10][10];

void 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\n Minimum cost=%d", mincost);

}

**Output**

A screenshot of a computer

Description automatically generated

**Q9. Implement All-Pairs Shortest Paths Problem using Floyd's algorithm.**

#include <stdio.h>

int min(int, int);

void floyds(int p[10][10], int n);

int min(int a, int b)

{

if (a < b)

return (a);

else

return (b);

}

void floyds(int p[10][10], int n)

{

int i, j, k;

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

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

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

if (i == j)

p[i][j] = 0;

else

p[i][j] = min(p[i][j], p[i][k] + p[k][j]);

}

void main()

{

int p[10][10], w, n, e, u, v, i, j;

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

scanf("%d", &n);

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

scanf("%d", &e);

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

{

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

p[i][j] = 999;

}

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

{

printf("\n Enter the end vertices of edge %d with its weight: ", i);

scanf("%d%d%d", &u, &v, &w);

p[u][v] = w;

}

printf("\n Matrix of input data:\n");

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

{

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

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

printf("\n");

}

floyds(p, n);

printf("\n Transitive closure:\n");

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

{

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

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

printf("\n");

}

printf("\n The shortest paths are:\n");

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

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

{

if (i != j)

printf("\n <%d,%d>=%d", i, j, p[i][j]);

}

}

**Output**

A screenshot of a computer

Description automatically generated

**Q10. Find a subset of a given set S = {s1,s2,.....,sn} of n positive integers whose sum is equal to a given positive integer d. For example, if S= {1, 2, 5, 6, 8} and d = 9 there are two solutions {1,2,6} and {1,8}.A suitable message is to be displayed if the given problem instance doesn't have a solution.**

#include<stdio.h>

int s[10], x[10], d;

void sumofsub(int, int, int);

void main() {

int n, sum = 0;

int i;

printf("\nEnter the size of the set: ");

scanf("%d", &n);

printf("\nEnter the set in increasing order:\n");

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

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

printf("\nEnter the value of d: \n ");

scanf("%d", &d);

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

sum = sum + s[i];

if (sum < d || s[1] > d)

printf("\nNo subset poossible: ");

else

sumofsub(0, 1, sum);

}

void sumofsub(int m, int k, int r) {

int i = 1;

x[k] = 1;

if ((m + s[k]) == d) {

printf("Subset:");

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

if (x[i] == 1)

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

printf("\n");

} else if (m + s[k] + s[k + 1] <= d)

sumofsub(m + s[k], k + 1, r - s[k]);

if ((m + r - s[k] >= d) && (m + s[k + 1] <= d)) {

x[k] = 0;

sumofsub(m, k + 1, r - s[k]);

}

}

**Output**

A screenshot of a computer

Description automatically generated

**Q11. Implement N Queen's problem using Back Tracking.**

#include<stdio.h>

#include<stdlib.h>

#include<math.h>

int a[30], count = 0;

int place(int pos) {

int i;

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

if ((a[i] == a[pos]) || ((abs(a[i] - a[pos]) == abs(i - pos))))

return 0;

}

return 1;

}

void print\_sol(int n) {

int i, j;

count++;

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

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

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

if (a[i] == j)

printf("Q\t");

else

printf("\*\t");

}

printf("\n");

}

}

void queen(int n) {

int k = 1;

a[k] = 0;

while (k != 0) {

a[k] = a[k] + 1;

while ((a[k] <= n) && !place(k))

a[k]++;

if (a[k] <= n) {

if (k == n)

print\_sol(n);

else {

k++;

a[k] = 0;

}

} else

k--;

}

}

void main() {

int i, n;

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

scanf("%d", &n);

queen(n);

printf("\nTotal solutions = %d", count);

}

**Output**

A screenshot of a computer

Description automatically generated