VISVESVARAYA TECHNOLOGICAL UNIVERSITY

“JnanaSangama”, Belgaum -590014, Karnataka.



LAB REPORT

on

Analysis and Design of Algorithms

Submitted by

Aishwarya Vijay (1BM22CS025)

in partial fulfillment for the award of the degree of

BACHELOR OF ENGINEERING

in

COMPUTER SCIENCE AND ENGINEERING



B.M.S. COLLEGE OF ENGINEERING

(Autonomous Institution under VTU)

BENGALURU-560019

April-2024 to August-2024

B. M. S. College of Engineering,

Bull Temple Road, Bangalore 560019

(Affiliated to Visvesvaraya Technological University, Belgaum)

Department of Computer Science and Engineering



CERTIFICATE

This is to certify that the Lab work entitled “Analysis and Design of Algorithms” carried out by ARYA HIMANSHU RESHAMDALAL (1BM22CS055), who is bonafide student of B.M.S. College of Engineering. It is in partial fulfillment for the award of Bachelor of Engineering in Computer Science and Engineering of the Visvesvaraya Technological University, Belgaum during the academic semester April-2024 to August-2024. The Lab report has been approved as it satisfies the academic requirements in respect of an Analysis and Design of Algorithms (23CS4PCADA) work prescribed for the said degree.

Shravya A R Dr. Jyothi S Nayak

Assistant Professor Professor and Head

Department of CSE Department of CSE

BMSCE, Bengaluru BMSCE, Bengaluru

Index Sheet

|  |  |  |
| --- | --- | --- |
| Lab  Program No. | Program Details | Page No. |
| 1 | Leetcode exercises on Stacks, Queues, Circular Queues, Priority Queues. | 1 |
| 2 | Leetcode exercises on DFS, BFS. | 2 |
| 3 | Leetcode exercises on Trees and Graphs. | 3 |
| 4 | * Write program to obtain the Topological ordering of vertices in a given digraph. * Leet Code. | 4 |
| 5 | Implement Johnson Trotter algorithm to generate permutations. | 6 |
| 6 | Sort a given set of N integer elements using Merge Sort technique and compute its time taken. Run the program for different values of N and record the time taken to sort. | 7 |
| 7 | Sort a given set of N integer elements using Quick Sort technique and compute its time taken. | 9 |
| 8 | Sort a given set of N integer elements using Heap Sort technique and compute its time taken. | 13 |
| 9 | * Implement 0/1 Knapsack problem using dynamic programming. * Leet Code. | 16 |
| 10 | * Implement All Pair Shortest paths problem using Floyd’s algorithm. * Leet Code. | 18 |
| 11 | * Find Minimum Cost Spanning Tree of a given undirected graph using Prim’s algorithm. * Find Minimum Cost Spanning Tree of a given undirected graph using Kruskal’s algorithm. | 20 |
| 12 | Implement Fractional Knapsack using Greedy technique. | 22 |
| 13 | From a given vertex in a weighted connected graph, find shortest paths to other vertices using Dijkstra’s algorithm. | 23 |
| 14 | Implement “N-Queens Problem” using Backtracking. | 25 |

Course Outcome

|  |  |
| --- | --- |
| CO1 | Analyze time complexity of Recursive and Non-recursive algorithms using asymptotic notations. |
| CO2 | Apply various design techniques for the given problem. |
| CO3 | Apply the knowledge of complexity classes P, NP, and NP-Complete and prove certain problems are NP-Complete |
| CO4 | Design efficient algorithms and conduct practical experiments to solve problems. |

**1. Write program to obtain the Topological ordering of vertices in a given digraph.**

//C program to implement topological sort using DFS

#include <stdio.h>

int n, a[10][10], res[10], s[10], top = 0;

void dfs(int, int, int[][10]); void dfs\_top(int, int[][10]);

int main()

{

printf("Enter the no. of nodes"); scanf("%d", &n);

int i, j; for (i = 0; i < n; i++) { for (j = 0; j < n; j++) { scanf("%d", &a[i][j]);

}

}

dfs\_top(n, a); printf("Solution: "); for (i = n - 1; i >= 0; i--) { printf("%d ", res[i]);

} return 0;

}

void dfs\_top(int n, int a[][10]) { int i; for (i = 0; i < n; i++) { s[i] = 0;

} for (i = 0; i < n; i++) { if (s[i] == 0) { dfs(i, n, a);

}

}

}

void dfs(int j, int n, int a[][10]) { s[j] = 1;

int i; for (i = 0; i < n; i++) { if (a[j][i] == 1 && s[i] == 0) { dfs(i, n, a);

}

} res[top++] = j;

}

**OUTPUT:**

Enter the no. of nodes6

0 0 1 1 0 0

# 0 0 1 1 0

[0 0 0 1 0 1](#_Toc31842)

[0 0 0 0 0 1](#_Toc31843)

[0 0 0 0 00 0 0 0 0 0 Solution: 1 4 0 2 3 5 1](#_Toc31844)

//C program to implement topological sort using source removal method

#include<stdio.h> int a[10][10],n,t[10],indegree[10]; int stack[10],top=-1;

void computeIndegree(int,int [][10]);

void tps\_SourceRemoval(int,int [][10]);

int main(){

printf("Enter the no. of nodes: "); scanf("%d",&n);

int i,j;

for(i=0;i<n;i++){ for(j=0;j<n;j++){ scanf("%d",&a[i][j]);

}

}

computeIndegree(n,a); tps\_SourceRemoval(n,a); printf("Solution:"); for(i=0;i<n;i++){ printf("%d ",t[i]);

} return 0;

}

void computeIndegree(int n,int a[][10]){ int i,j,sum=0; for(i=0;i<n;i++){ sum=0; for(j=0;j<n;j++){ sum=sum+a[j][i];

}

indegree[i]=sum;

}

}

void tps\_SourceRemoval(int n,int a[][10]){

int i,j,v;

for(i=0;i<n;i++){ if(indegree[i]==0){ stack[++top]=i;

} } int k=0; while(top!=-1){ v=stack[top--]; t[k++]=v; for(i=0;i<n;i++){ if(a[v][i]!=0){ indegree[i]=indegree[i]-1; if(indegree[i]==0){ stack[++top]=i;

}

}

}

}

}

**OUTPUT:**

Enter the no. of nodes: 5

1. 0 1 0 0
2. 0 0 1 0

0 0 0 0 1

0 0 1 0 1

0 0 0 0 0

Solution:1 3 0 2 4

**2. Implement Johnson Trotter algorithm to generate permutations.**

#include <stdio.h>

#include <stdlib.h>

void swap(int\* a, int\* b) { int temp = \*a; \*a = \*b;

\*b = temp;

}

void generatePermutations(int arr[], int start, int end) { if (start == end) { for (int i = 0; i <= end; i++) { printf("%d ", arr[i]);

} printf("\n"); } else { for (int i = start; i <= end; i++) { swap(&arr[start], &arr[i]); generatePermutations(arr, start + 1, end); swap(&arr[start], &arr[i]); // backtrack

}

}

}

int main() {

int n;

printf("Enter the number of elements: "); scanf("%d", &n);

int\* arr = (int\*)malloc(n \* sizeof(int)); printf("Enter the elements: "); for (int i = 0; i < n; i++) { scanf("%d", &arr[i]);

}

generatePermutations(arr, 0, n - 1);

free(arr);

return 0;

}

**OUTPUT:**

Enter the number of elements: 4

Enter the elements: 1 2 3 4

1 2 3 4

1 2 4 3

1 3 2 4

1 3 4 2

1 4 3 2

1. 4 2 3
2. 1 3 4

2 1 4 3

2 3 1 4

2 3 4 1

2 4 3 1

1. 4 1 3
2. 2 1 4

3 2 4 1

3 1 2 4

3 1 4 2

3 4 1 2

1. 4 2 1
2. 2 3 1

4 2 1 3

4 3 2 1

4 3 1 2

4 1 3 2

4 1 2 3

**3. Sort a given set of N integer elements using Merge Sort technique and compute its time taken. Run the program for different values of N and record the time taken to sort.**

//C program to implement merge sort

#include <stdio.h> #include<time.h> int a[20],n;

void simple\_sort(int [],int,int,int); void merge\_sort(int[],int,int); int main()

{

int i; clock\_t start, end; double time\_taken;

printf("Enter the no. of elements:"); scanf("%d", &n); printf("Enter the array elements:"); for (i = 0; i < n; i++) { scanf("%d", &a[i]);

}

start = clock(); merge\_sort(a, 0, n - 1); end = clock();

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

printf("Sorted array:"); for (i = 0; i < n; i++) { printf("%d ", a[i]);

}

printf("\n");

printf("Time taken to sort: %f seconds\n", time\_taken);

return 0;

}

void merge\_sort(int a[],int low, int high){ if(low<high){ int mid=(low+high)/2; merge\_sort(a,low,mid); merge\_sort(a,mid+1,high); simple\_sort(a,low,mid,high);

}

}

void simple\_sort(int a[],int low, int mid, int high){ int i=low,j=mid+1,k=low;

int c[n];

while(i<=mid && j<=high){

if(a[i]<a[j]){ c[k++]=a[i]; i++; }else{ c[k++]=a[j]; j++;

}

}

while(i<=mid){ c[k++]=a[i]; i++;

}

while(j<=high){ c[k++]=a[j]; j++;

}

for(i=low;i<=high;i++){

a[i]=c[i];

}

}

**OUTPUT:**

Enter the no. of elements:10

Enter the array elements:8 96 32 75 62 78 63 48 56 100

Sorted array:8 32 48 56 62 63 75 78 96 100

Time taken to sort: 0.000002 seconds

**4. Sort a given set of N integer elements using Quick Sort technique and compute its time taken.**

//C program to implement quick sort

#include <stdio.h> #include<time.h> int a[20],n;

int partition(int [],int, int); void quick\_sort(int [],int,int); void swap(int\*,int\*); int main()

{

int i; clock\_t start, end;

double time\_taken;

printf("Enter the no. of elements:"); scanf("%d", &n); printf("Enter the array elements:"); for (i = 0; i < n; i++) { scanf("%d", &a[i]);

}

start = clock(); quick\_sort(a, 0, n - 1); end = clock();

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

printf("Sorted array:"); for (i = 0; i < n; i++) { printf("%d ", a[i]);

}

printf("\n");

printf("Time taken to sort: %f seconds\n", time\_taken);

return 0;

}

void swap(int \*a,int \*b){ int temp=\*a;

\*a=\*b;

\*b=temp;

}

void quick\_sort(int a[],int low,int high){ if(low<high){ int mid=partition(a,low,high); quick\_sort(a,low,mid-1); quick\_sort(a,mid+1,high);

}

}

int partition(int a[],int low,int high){ int pivot=a[low]; int i=low; int j=high+1;

while(i<=j){ do{ i=i+1;

}while(a[i]<pivot && i<=high);

do{ j=j-1;

}while(a[j]>pivot && j>=low);

if(i<j){ swap(&a[i],&a[j]);

}

}

swap(&a[j],&a[low]); return j;

}

**OUTPUT:**

Enter the no. of elements:10

Enter the array elements:96 53 26 78 12 63 85 12 06 95

Sorted array:6 12 12 26 53 63 78 85 95 96

Time taken to sort: 0.000002 seconds

**5. Sort a given set of N integer elements using Heap Sort technique and compute its time taken.**

//C program to implement heapify

#include<stdio.h> int a[10],n;

void heapify(int[],int);

int main(){

printf("Enter the number of array elements:"); scanf("%d",&n);

int i;

printf("Enter array elements:"); for(i=0;i<n;i++){ scanf("%d",&a[i]);

} heapify(a,n); printf("Array elements:"); for(i=0;i<n;i++){ printf(" %d",a[i]);

} return 0;

}

void heapify(int a[],int n){

int k;

for(k=1;k<n;k++){ int key=a[k]; int c=k; int p=(c-1)/2; while(c>0 && key>a[p]){

a[c]=a[p]; c=p; p=(c-1)/2;

} a[c]=key;

}

}

**OUTPUT:**

Enter the number of array elements:7

Enter array elements:50 25 30 75 100 45 80

Array elements: 100 75 80 25 50 30 45

**6. Implement 0/1 Knapsack problem using dynamic programming.**

//C program to implement knapsack problem in dynamic programming

#include <stdio.h> int n,m,w[10],p[10],v[10][10]; void knapsack(int,int,int[],int[]); int max(int,int); int main()

{

int i,j;

printf("Enter the no. of items:"); scanf("%d",&n); printf("Enter the capacity of knapsack:"); scanf("%d",&m); printf("Enter weights:"); for(i=0;i<n;i++){ scanf("%d",&w[i]);

} printf("Enter profits:"); for(i=0;i<n;i++){ scanf("%d",&p[i]);

}

knapsack(n,m,w,p); printf("Optimal Solution:\n"); for(i=0;i<n;i++){ for(j=0;j<n;j++){ printf("%d ",v[i][j]);

} printf("\n");

} return 0;

}

void knapsack(int n, int m, int w[],int p[]){ int i,j;

for(i=0;i<n;i++){ for(j=0;j<m;j++){ if(i==0 || j==0){ v[i][j]=0; }else if(w[i]>j){ v[i][j]=v[i-1][j];

}else{ v[i][j]=max(v[i-1][j],((v[i-1][j-w[i]])+p[i]));

}

}

}

}

int max(int a,int b){ if(a>b){ return a; }else{ return b;

}

}

**OUTPUT:**

Enter the no. of items:4

Enter the capacity of knapsack:5

Enter weights:2 1 3 2

Enter profits:12 10 20 15 Optimal Solution:

0 0 0 0

0 10 10 10

0 10 10 20

0 10 15 25

**7. Implement All Pair Shortest paths problem using Floyd’s algorithm.**

//C program to implement floyd's algorithm

#include <stdio.h> int a[10][10],D[10][10],n; void floyd(int [][10],int); int min(int,int); int main() {

printf("Enter the no. of vertices:"); scanf("%d",&n); printf("Enter the cost adjacency matrix:\n"); int i,j;

for(i=0;i<n;i++){ for(j=0;j<n;j++){ scanf("%d",&a[i][j]);

} } floyd(a,n); printf("Distance Matrix:\n"); for(i=0;i<n;i++){ for(j=0;j<n;j++){ printf("%d ",D[i][j]);

} printf("\n");

} return 0;

}

void floyd(int a[][10],int n){

int i,j,k;

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

D[i][j]=a[i][j];

}

}

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

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

}

}

}

}

int min(int a,int b){ if(a<b){ return a; }else{ return b;

}

}

**OUTPUT:**

Enter the no. of vertices:4 Enter the cost adjacency matrix:

0 99 3 99

2 0 99 99

99 6 0 1

7 99 99 0 Distance Matrix:

0 9 3 4

2 0 5 6

8 6 0 1

7 16 10 0

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

//C program to implement prim's algorithm

#include <stdio.h>

int cost[10][10], n, t[10][2], sum;

void prims(int cost[10][10], int n);

int main() { int i, j;

printf("Enter the number of vertices: "); scanf("%d", &n);

printf("Enter the cost adjacency matrix:\n"); for (i = 0; i < n; i++) { for (j = 0; j < n; j++) { scanf("%d", &cost[i][j]);

}

}

prims(cost, n);

printf("Edges of the minimal spanning tree:\n"); for (i = 0; i < n - 1; i++) { printf("(%d, %d) ", t[i][0], t[i][1]);

}

printf("\nSum of minimal spanning tree: %d\n", sum);

return 0;

}

void prims(int cost[10][10], int n) {

int i, j, u, v; int min, source; int p[10], d[10], s[10];

min = 999; source = 0;

// Initialize arrays for (i = 0; i < n; i++) { d[i] = cost[source][i]; s[i] = 0; p[i] = source;

}

s[source] = 1; sum = 0; int k = 0;

// Find MST for (i = 0; i < n - 1; i++) { min = 999; u = -1;

// Find the vertex with minimum distance to the MST for (j = 0; j < n; j++) { if (s[j] == 0 && d[j] < min) { min = d[j]; u = j;

}

}

if (u != -1) {

// Add edge to MST t[k][0] = u; t[k][1] = p[u]; k++; sum += cost[u][p[u]]; s[u] = 1;

// Update distances for (v = 0; v < n; v++) { if (s[v] == 0 && cost[u][v] < d[v]) { d[v] = cost[u][v]; p[v] = u;

}

}

}

}

}

**OUTPUT:**

Enter the number of vertices: 4

Enter the cost adjacency matrix:

1. 1 5 2
2. 0 99 99

5 99 0 3

2 99 3 0

Edges of the minimal spanning tree:

(1, 0) (3, 0) (2, 3)

Sum of minimal spanning tree: 6

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

//C program to implement Kruskal’s algorithm

#include <stdio.h>

int cost[10][10], n, t[10][2], sum;

void kruskal(int cost[10][10], int n); int find(int parent[10], int i);

int main() { int i, j;

printf("Enter the number of vertices: "); scanf("%d", &n);

printf("Enter the cost adjacency matrix:\n"); for (i = 0; i < n; i++) { for (j = 0; j < n; j++) { scanf("%d", &cost[i][j]);

}

}

kruskal(cost, n);

printf("Edges of the minimal spanning tree:\n"); for (i = 0; i < n - 1; i++) { printf("(%d, %d) ", t[i][0], t[i][1]);

}

printf("\nSum of minimal spanning tree: %d\n", sum);

return 0;

}

void kruskal(int cost[10][10], int n) {

int min, u, v, count, k; int parent[10];

k = 0; sum = 0;

// Initialize parent array for Union-Find for (int i = 0; i < n; i++) { parent[i] = i;

}

count = 0; while (count < n - 1) { min = 999; u = -1; v = -1;

// Find the minimum edge for (int i = 0; i < n; i++) { for (int j = 0; j < n; j++) { if (find(parent, i) != find(parent, j) && cost[i][j] < min) { min = cost[i][j]; u = i; v = j;

}

}

}

// Perform Union operation int root\_u = find(parent, u); int root\_v = find(parent, v);

if (root\_u != root\_v) { parent[root\_u] = root\_v; t[k][0] = u; t[k][1] = v; sum += min; k++; count++;

}

}

}

int find(int parent[10], int i) {

while (parent[i] != i) { i = parent[i]; } return i;

}

**OUTPUT:**

Enter the number of vertices: 4 Enter the cost adjacency matrix:

1. 1 5 2
2. 0 99 99

5 99 0 3

2 99 3 0

Edges of the minimal spanning tree:

(1, 0) (3, 0) (2, 3)

Sum of minimal spanning tree: 6

**9. Implement fractional Knapsack problem using Greedy technique.**

#include <stdio.h>

void knapsack(int n, int p[], int w[], int W) { int used[n]; for (int i = 0; i < n; ++i) used[i] = 0; int cur\_w = W; float tot\_v = 0.0; int i, maxi; 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; if (w[maxi] <= cur\_w) { cur\_w -= w[maxi]; tot\_v += p[maxi];

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

} else { int taken = cur\_w; cur\_w = 0; tot\_v += (float)taken/p[maxi] \* p[maxi]; printf("Added %d%% (%d, %d) of object %d in the bag.\n", (int)((float)taken/w[maxi] \* 100), w[maxi], p[maxi], maxi + 1);

}

}

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

int main() { int n, W; printf("Enter the number of objects: "); scanf("%d", &n); int p[n], w[n]; printf("Enter the profits of the objects: "); for(int i = 0; i < n; i++){ scanf("%d", &p[i]);

}

printf("Enter the weights of the objects: "); for(int i = 0; i < n; i++){

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

}

printf("Enter the maximum weight of the bag: "); scanf("%d", &W);

knapsack(n, p, w, W);

return 0;

}

**OUTPUT:**

Enter the number of objects: 7

Enter the profits of the objects: 5 10 15 7 8 9 4

Enter the weights of the objects: 1 3 5 4 1 3 2

Enter the maximum weight of the bag: 15

Added object 4 (4, 7) completely in the bag. Space left: 11. Added object 7 (2, 4) completely in the bag. Space left: 9.

Added object 3 (5, 15) completely in the bag. Space left: 4.

Added object 6 (3, 9) completely in the bag. Space left: 1.

Added 33% (3, 10) of object 2 in the bag. Filled the bag with objects worth 36.00.

**10. From a given vertex in a weighted connected graph, find shortest paths to other vertices using Dijkstra’s algorithm.**

// C program to implement Dijkstra's algorithm

#include <stdio.h>

int cost[10][10], n, result[10][2], weight[10];

void dijkstras(int [][10], int );

int main()

{

int i, j, s;

printf("Enter the number of vertices: "); scanf("%d", &n);

printf("Enter the cost adjacency matrix:\n"); for (i = 0; i < n; i++) { for (j = 0; j < n; j++) { scanf("%d", &cost[i][j]);

}

}

printf("Enter the source vertex: "); scanf("%d", &s); dijkstras(cost, s);

printf("Path:\n"); for (i = 1; i < n; i++) { printf("(%d, %d) with weight %d ", result[i][0], result[i][1], weight[result[i][1]]);

} return 0;

}

void dijkstras(int cost[][10], int s){ int d[10], p[10], visited[10]; int i, j, min, u, v, k; for(i = 0; i < 10; i++){ d[i] = 999; visited[i] = 0; p[i] = s;

} d[s] = 0; visited[s] = 1; for(i = 0; i < n; i++){ min = 999; u = 0; for(j = 0; j < n; j++){ if(visited[j] == 0){ if(d[j] < min){ min = d[j]; u = j;

}

} }

visited[u] = 1;

for(v = 0; v < n; v++){ if(visited[v] == 0 && (d[u] + cost[u][v] < d[v])){ d[v] = d[u] + cost[u][v]; p[v] = u;

}

} } for(i = 0; i < n; i++){ result[i][0] = p[i]; result[i][1] = i; weight[i] = d[i];

}

}

**OUTPUT:**

Enter the number of vertices: 4 Enter the cost adjacency matrix:

1. 1 5 2
2. 0 99 99

5 99 0 3

2 99 3 0

Enter the source vertex: 0 Path:

(0, 1) with weight 1 (0, 2) with weight 5 (0, 3) with weight 2

**11. Implement “N-Queens Problem” using Backtracking.**

#include <stdio.h> #include <stdbool.h>

bool place(int[], int); void printSolution(int[], int); void nQueens(int); int main() { int n;

printf("Enter the number of queens: "); scanf("%d",&n); nQueens(n); return 0;

}

void nQueens(int n){

int x[10]; int count=0; int k=1; while(k!=0){ x[k]=x[k]+1; while(x[k]<=n && !place(x,k)){ x[k]=x[k]+1;

}

if(x[k]<=n){ if(k==n){ printSolution(x, n); printf("Solution found\n"); count++; }else{ k++; x[k]=0;

} }else{ k--;

}

}

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

}

bool place(int x[10], int k){

int i;

for(i=1;i<k;i++){ if((x[i]==x[k])||(i-x[i]==k-x[k])||(i+x[i]==k+x[k])){ return false;

} } return true;

}

void printSolution(int x[10], int n){

int i;

for(i=1;i<=n;i++){ printf("%d ", x[i]);

} printf("\n");

}

**OUTPUT:**

Enter the number of queens: 4

2 4 1 3

Solution found 3 1 4 2

Solution found

Total solutions: 2