

LABORATORY MANUAL

# DESIGN AND ANALYSIS OF ALGORITHMS B18CS4010

**SCHOOL OF COMPUTING AND INFORMATION TECHNOLOGY**

|  |  |
| --- | --- |
| **Name** | ASHOK K PATIL |
| **School** | SCHOOL OF COMPUTING AND INFORMATION TECHNOLOGY |
| **Semester** | IV |
| **Section** |  |
| **Academic Year** | 2019-20 |

|  |
| --- |
| **LAB REQUIREMENTS** |
| Following are the required hardware and software for this Course, which are available in the laboratory.   * **Hardware:** Desktop system or Virtual machine in a cloud with Operating System installed. The Configuration of the systems are Pentium IV Processor with 1 GB RAM and 250 GB Hard Disk   .Desktop systems support dual booting with Windows / Linux Operating System.   * **Software:** C-compiler.[*gcc* compiler in Linux Environment (fedora 8 to fedora 20), and Turbo C- IDE on Windows Environment]. |

This manual is designed for amateur programmers. It is intended that the learning procedure is made interactive and interesting. The programs presented here are to be compiled, modified and enhanced for better performance and efficiency. Every design paradigm is to be clearly understood and implemented for effective learning.

The Linux and windows environments are used as explained below.

**Linux environment:** All the programs have been composed using ‘*vi’* editor and compiled/executed with the *gcc* compiler in Linux environment. The execution steps are as follows:

1. Open vi editor using the shell command

### $vi <filename.c>

1. Type in the C-code according to the program logic. Save the program and exit to get shell command prompt **(:wq)**
2. Compile the program using gcc compiler using the shell command

### $gcc -o exeFileName <filename.c>

1. Execute the program using the following shell command

### $./exeFileName

**Windows environment:** Programs can be edited, compiled and executed using Turbo C/C++ IDE. All operations like opening a file, saving a file, compiling the program and executing the program are listed under different menus inside the IDE. Keyboard shortcuts are also available for the abovementioned operations.

**Note:** Every lab in the manual includes problem statement, learning outcomes, theoretical description, algorithm, program, program description, expected results, implementation phase, simulation of syntax and logical errors, final program with results, assignment given, and viva question.

**Recommendation**: Linux environment is preferred for executing the programs composed in this manual.

# CONTENTS

|  |  |  |
| --- | --- | --- |
| **Sl. No** | **PROBLEM STATEMENT** | **Page No** |
| **1** | **Search for a given pattern in a text string using Brute Force String Matching.** | **5**  **7**  **333**  **4** |
| **2** | **Sort a set of elements in ascending order using Quick Sort algorithm.** |
| **3 a** | **Find Minimum Cost Spanning Tree of a given undirected graph using Kruskal's** | **9** |
| **3 b** | **Find Minimum Cost Spanning Tree of a given undirected graph using Prim’s algorithms. Differentiate the methods.** | **11**  **13**  **9** |
| **4** | **From a given vertex in a weighted connected graph, find shortest paths to other vertices using Dijikstra’s algorithm** |
| **5** | **Design and Implement 0/1 Knapsack problem using Dynamic Programming.** | **15** |
| **6** | **Implement All-Pairs Shortest Paths Problem using Floyd's algorithm** | **16**  **18** |
| **7** | **Obtain the DFS ordering of vertices in a given digraph.** |
| **8** | **Implement Horspool’s algorithm for String Matching and find the number of key comparisons in successful search and unsuccessful search** | **20** |
| **9** | **Sort a given set of elements in ascending order which has duplicate entries. Use the sorting by counting algorithm** | **22** |

`

|  |  |  |
| --- | --- | --- |
| **10** | **Implement N Queen's problem using Back Tracking.** | **24**  **26**  **22222** |
| **11** | **Write a program to sort all transactions of Big Mall by quantity of sales.** |
| **12** | **Write a program to find network of people of same location in Linkedin social network.** | **29** |

**Program 1**

**//Search for a given pattern in a text string using Brute Force String Matching.**

**#include <stdio.h>**

**#include <string.h>**

**void main()**

**{**

**char a[100], b[100];**

**int i,n,m,j;**

**clrscr();**

**printf("Enter some text\n");**

**gets(a);**

**printf("Enter a string to find\n");**

**gets(b);**

**m=strlen(b);**

**n=strlen(a);**

**for(i=0;i<=n-m;i++)**

**{**

**j=0;**

**while(j<m && b[j]==a[i+j])**

**{**

**j=j+1;**

**}**

**if(j==m)**

**{**

**printf("SUBSTRING FOUND AT LOCATION %d\n",i+1);**

**getch();**

**exit();**

**}**

**}**

**printf("SUBSTRING NOT FOUND\n");**

**getch();**

**}**

**Output:**

****

**Program 2**

**Sort a set of elements in ascending order using Quick Sort algorithm.**

**#include<stdio.h>**

**void quicksort(int number[25],int first,int last){**

**int i, j, pivot, temp;**

**if(first<last){**

**pivot=first;**

**i=first;**

**j=last;**

**while(i<j){**

**while(number[i]<=number[pivot]&&i<last)**

**i++;**

**while(number[j]>number[pivot])**

**j--;**

**if(i<j){**

**temp=number[i];**

**number[i]=number[j];**

**number[j]=temp;**

**}**

**}**

**temp=number[pivot];**

**number[pivot]=number[j];**

**number[j]=temp;**

**quicksort(number,first,j-1);**

**quicksort(number,j+1,last);**

**}**

**}**

**void main()**

**{**

**int i, count, number[25];**

**clrscr();**

**printf("How many elements are u going to enter?: ");**

**scanf("%d",&count);**

**printf("Enter %d elements: ", count);**

**for(i=0;i<count;i++)**

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

**quicksort(number,0,count-1);**

**printf("Order of Sorted elements: ");**

**for(i=0;i<count;i++)**

**printf(" %d",number[i]);**

**getch();**

**}**

**OUTPUT:**

**Enter the number of elements: 5**

**Enter the elements**

**45 8 21 3 4**

**Sorted elements are**

**3 4 8 21 45**

**Program 3a**

**Find minimum cost spanning tree of a given undirected graph using Kruskal’s algorithm.**

**#include<stdio.h>**

**#include<conio.h>**

**#define INFINITY 999**

**#define MAX 100**

**int parent[MAX],cost[MAX][MAX],t[MAX][2];**

**int find(int v)**

**{**

**while(parent[v])**

**{**

**v=parent[v];**

**}**

**return v;**

**}**

**void union1(int i,int j)**

**{**

**parent[j]=i;**

**}**

**void kruskal(int n)**

**{**

**int i,j,k,u,v,mincost,res1,res2,sum=0;**

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

**{**

**mincost=INFINITY;**

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

**{**

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

**{**

**if(i==j) continue;**

**if(cost[i][j]<mincost)**

**{**

**u=find(i);**

**v=find(j);**

**if(u!=v)**

**{**

**res1=i;**

**res2=j;**

**mincost=cost[i][j];**

**}**

**}**

**}**

**}**

**union1(res1,find(res2));**

**t[k][1]=res1;**

**t[k][2]=res2;**

**sum=sum+mincost;**

**}**

**printf("\nCost of spanning tree is %d\n",sum);**

**printf("\nEdges of spanning tree are\n");**

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

**printf("%d->%d\n",t[i][1],t[i][2]);**

**}**

**void main()**

**{**

**int i,j,n;**

**clrscr();**

**printf("\nEnter the number of vertices : ");**

**scanf("%d",&n);**

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

**parent[i]=0;**

**printf("\nEnter the cost adjacency matrix 0-for self edge and 999-if no edge\n");**

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

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

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

**kruskal(n);**

**getch();**

**}**

**OUTPUT:**

**Enter the number of vertices: 4**

**Enter the cost matrix 0-for self edge and 999-if no edge**

**0 20 2 999**

**20 0 15 5**

**2 15 0 25**

**999 5 25 0**

**Cost of the spanning tree**

**22**

**Edges of the spanning tree:**

**1->3**

**2->4**

**2->3**

**Program 3b**

**Find minimum cost spanning tree of a given undirected graph using prims algorithm**

**#include<stdio.h>**

**#include<conio.h>**

**#define INFINITY 999**

**int prim(int cost[10][10],int source,int n)**

**{**

**int i,j,sum=0,visited[10];**

**int distance[10],vertex[10];**

**int min,u,v;**

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

**{**

**vertex[i]=source;**

**visited[i]=0;**

**distance[i]=cost[source][i];**

**}**

**visited[source]=1;**

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

**{**

**min=INFINITY;**

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

**{**

**if(!visited[j]&&distance[j]<min)**

**{**

**min=distance[j];**

**u=j;**

**}**

**}**

**visited[u]=1;**

**sum=sum+distance[u];**

**printf("\n%d->%d",vertex[u],u);**

**for(v=1;v<=n;v++)**

**{**

**if(!visited[v]&&cost[u][v]<distance[v])**

**{**

**distance[v]=cost[u][v];**

**vertex[v]=u;**

**}**

**}**

**}**

**return sum;**

**}**

**void main()**

**{**

**int a[10][10],n,i,j,m,source;**

**clrscr();**

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

**scanf("%d",&n);**

**printf("\nenter the cost matrix:\n 0-self loop and 999-no edge:\n");**

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

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

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

**printf("\n enter the source:\n");**

**scanf("%d",&source);**

**m=prim(a,source,n);**

**printf("\n the cost of spanning tree=%d",m);**

**getch();**

**}**

**OUTPUT:**

**Enter the number of vertices: 5**

**Enter the cost matrix 0-for self edge and 999-if no edge**

**0 3 4 999 5**

**3 0 999 6 1**

**4 999 0 9 7**

**999 6 9 0 2**

**5 1 7 2 0**

**Enter the source**

**2**

**2->5**

**5->4**

**2->1**

**1->3**

**Cost = 10**

**Program 4**

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

**#include<stdio.h>**

**#include<conio.h>**

**#define INFINITY 999**

**void dijkstra(int cost[10][10],int n,int source,int distance[10])**

**{**

**int visited[10],min,u;**

**int i,j;**

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

**{**

**distance[i]=cost[source][i];**

**visited[i]=0;**

**}**

**visited[source]=1;**

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

**{**

**min=INFINITY;**

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

**if(visited[j]==0 && distance[j]<min)**

**{**

**min=distance[j];**

**u=j;**

**}**

**visited[u]=1;**

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

**if(visited[j]==0 && (distance[u]+cost[u][j])<distance[j])**

**{**

**distance[j]=distance[u]+cost[u][j];**

**}**

**}**

**}**

void main()

**{**

**int n,cost[10][10],distance[10];**

**int i,j,source,sum;**

**clrscr();**

**printf("\nEnter how many nodes : ");**

**scanf("%d",&n);**

**printf("\nCost Matrix\nEnter 999 for no edge\n");**

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

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

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

**printf("Enter the source node\n");**

**scanf("%d",&source);**

**dijkstra(cost,n,source,distance);**

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

**printf("\n\nShortest Distance from %d to %d is %d",source,i,distance[i]);**

**getch();**

**}**

**OUTPUT:**

**Enter how many nodes:4**

**Cost Matrix**

**Enter 999 for no edge**

**999 999 3 999**

**999 999 4 7**

**999 4 999 15**

**999 7 15 999**

**Enter the source node**

**1**

**Shortest Distance for 1 to 1 is 999**

**Shortest Distance for 1 to 2 is 7**

**Shortest Distance for 1 to 3 is 3**

**Shortest Distance for 1 to 4 is 14**

**Program 5**

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

**#include<stdio.h>**

**#include<conio.h>**

**int w[10],p[10],n;**

**int max(int a,int b)**

**{**

**return a>b?a:b;**

**}**

**int knap(int i,int m)**

**{**

**if(i==n) return w[i]>m?0:p[i];**

**if (w[i]>m) return knap(i+1,m);**

**return max(knap(i+1,m),knap(i+1,m-w[i])+p[i]);**

**}**

**void main()**

**{**

**int m,i,max\_profit;**

**clrscr();**

**printf("\nEnter the number of objects: ");**

**scanf("%d",&n);**

**printf("\nEnter the knapsack capacity: ");**

**scanf("%d",&m);**

**printf("\nEnter profit followed by weight: ");**

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

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

**max\_profit=knap(1,m);**

**printf("\nMax profit = %d",max\_profit);**

**getch();**

**}**

**OUTPUT:**

**Enter the number of objects: 3**

**Enter the knapsack capacity:116**

**Enter the profit followed by weight:**

**100 12**

**12 15**

**20 30**

**Max profit=132**

**Program 6**

**Implement All-Pairs Shortest Paths Problem using Floyd's algorithm. Parallelize this algorithm, implement it using OpenMP and determine the speed-up achieved.**

**#include<stdio.h>**

**#include<omp.h>**

**#define INFINITY 999**

**int min(int i,int j)**

**{**

**if(i<j)**

**return i;**

**else**

**return j;**

**}**

**void floyd(int n,int p[10][10])**

**{**

**int i,j,k;**

**#pragma omp parallel for private(i,j,k) shared(p)**

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

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

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

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

**}**

**int main()**

**{**

**int i,j,n,a[10][10],d[10][10],source;**

**double starttime,end time;**

**printf("Enter the no.of nodes: ");**

**scanf("%d",&n);**

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

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

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

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

**starttime=omp\_get\_wtime();**

**floyd(n,a);**

**endtime=omp\_get\_wtime();**

**printf("\n\nThe distance matrix is \n");**

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

**{**

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

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

**printf("\n");**

**}**

**printf("\n\nThe time taken is %l0.9f\n",(double)(starttime-endtime));**

**return 0;**

**}**

**OUTPUT:**

**Enter the no of nodes: 5**

**Enter the adjancy matrix**

**0 15 8 10 999**

**15 0 4 999 999**

**8 4 0 999 12**

**10 999 999 0 7**

**999 999 12 7 0**

**The distance matrix is**

**0 12 8 10 17**

**12 0 4 22 16**

**8 4 0 18 12**

**10 22 18 0 7**

**17 16 12 7 0**

**The time taken is: 0.001107683**

**Program 7**

**Obtain the DFS ordering of vertices in a given digraph.**

**#include<stdio.h>**

**int i,visit[20],n,adj[20][20],s,topo\_order[10];**

**void dfs(int v)**

**{**

**int w;**

**visit[v]=1;**

**for(w=1;w<=n;w++)**

**if((adj[v][w]==1) && (visit[w]==0))**

**dfs(w);**

**topo\_order[i--]=v;**

**}**

**void main()**

**{**

**int v,w;**

**clrscr();**

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

**scanf("%d",&n);**

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

**for(v=1;v<=n;v++)**

**for(w=1;w<=n;w++)**

**scanf("%d",&adj[v][w]);**

**for(v=1;v<=n;v++)**

**visit[v]=0;**

**i=n;**

**for(v=1;v<=n;v++)**

**{**

**if(visit[v]==0)**

**dfs(v);**

**}**

**printf("\nTopological sorting is:");**

**for(v=1;v<=n;v++)**

**printf("v%d ",topo\_order[v]);**

**getch();**

**}**

**OUTPUT 1 :**

**Enter the number of vertices:**

**5**

**Enter the adjacency matrix**

**0 0 1 0 0**

**0 0 1 0 0**

**0 0 0 1 1**

**0 0 0 0 1**

**0 0 0 0 0**

**Topological ordering is v1 v2 v3 v4 v5**

**OUTPUT 2 :**

**enter the number of vertices:**

**3**

**Enter the adjacency matrix**

**0 1 0**

**0 0 1**

**1 0 0**

**Topological ordering is v1 v2 v3**

**Program 8:**

**Implement Horspool’s algorithm for String Matching and find the number of key**

**comparisons in successful search and unsuccessful search**

**#include<stdio.h>**

**#include<conio.h>**

**void main()**

**{**

**int table[126];**

**char t[100],p[25];**

**int n,i,k,j,m,flag=0;**

**clrscr();**

**printf(“Enter the text : “);**

**gets(t);**

**n=strlen(t);**

**printf(“Enter the pattern : “);**

**gets(p);**

**m=strlen(p);**

**for(i=0;i<126;i++)**

**table[i]=m;**

**for(j=0;j<m-2;j++)**

**table[p[j]]=m-1-j;**

**i=m-1;**

**while(i<=n-1)**

**{**

**k=0;**

**while(k<=m-1 && p[m-1-k]==t[i-k])**

**k++;**

**if(k==m)**

**{**

**printf(“The position of the pattern is %dn”,i-m+2);**

**flag=1;**

**break;**

**}**

**else**

**i=i+table[t[i]];**

**}**

**if(!flag)**

**printf(“Pattern is not found in the given text “);**

**getch();**

**}**

**OUTPUT:**

**Enter the text in which pattern is to be searched:**

**god is great**

**Enter the pattern to be searched:**

**great**

**Length of text=12 Length of pattern=5**

**The desired pattern was found starting from position 8**

**Enter the text in which pattern is to be searched:**

**god is great**

**Enter the pattern to be searched:**

**king**

**Length of text=12 Length of pattern=4**

**The pattern was not found in the given text**

**Program 9.**

**Sort a given set of elements in ascending order which has duplicate entries. Use the sorting by counting algorithm**

**#include <stdio.h>**

**/\* Counting sort function \*/**

**void counting\_sort(int a[], int k, int n)**

**{**

**int i, j;**

**int b[15], c[100];**

**for (i = 0; i <= k; i++)**

**c[i] = 0;**

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

**c[a[j]] = c[a[j]] + 1;**

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

**c[i] = c[i] + c[i-1];**

**for (j = n; j >= 1; j--)**

**{**

**b[c[a[j]]] = a[j];**

**c[a[j]] = c[a[j]] - 1;**

**}**

**printf("The Sorted array is : ");**

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

**printf("%d,", b[i]);**

**}**

**getch();**

**void main()**

**{**

**int n, k = 0, a[15], i;**

**printf("Input number of elements: ");**

**scanf("%d", &n);**

**printf("Input the array elements one by one: \n");**

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

**{**

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

**if (a[i] > k) {**

**k = a[i];**

**}**

**}**

**counting\_sort(a, k, n);**

**printf("\n");**

**getch();**

**}**

**OUTPUT:**

**Input number of elemets**

**5**

**Input the array elements one by one:**

**15**

**12**

**01**

**13**

**11**

**Output:**

**Input number of elements: Input the array elements one by one:**

**The Sorted array is :**

**15,**

**12**

**01**

**13**

**11**

**Program 10**

**Implement N Queen’s problem using back tracking.**

**#include<stdio.h>**

**#include<stdlib.h>**

**#include<conio.h>**

**#define MAX 50**

**int can\_place(int c[],int r)**

**{**

**int i;**

**for(i=0;i<r;i++)**

**if(c[i]==c[r] || (abs(c[i]-c[r])==abs(i-r)))**

**return 0;**

**return 1;**

**}**

**void display(int c[],int n)**

**{**

**int i,j;**

**char cb[10][10];**

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

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

**cb[i][j]='-';**

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

**cb[i][c[i]]='Q';**

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

**{**

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

**printf("%c",cb[i][j]);**

**printf("\n");**

**}**

**}**

**void n\_queens(int n)**

**{**

**int r;**

**int c[MAX];**

**c[0]= -1;**

**r=0;**

**while(r>=0)**

**{**

**c[r]++;**

**while(c[r]<n && !can\_place(c,r))**

**c[r]++;**

**if(c[r]<n)**

**{**

**if(r==n-1)**

**{**

**display(c,n);**

**printf("\n");**

**}**

**else**

**{**

**r++;**

**c[r]=-1;**

**}**

**}**

**else**

**r--;**

**}**

**}**

**void main()**

**{**

**int n;**

**clrscr();**

**printf("\nEnter the number of queens : ");**

**scanf("%d",&n);**

**n\_queens(n);**

**getch();**

**}**

**OUTPUT:**

**Enter the number of queens;**

**4**

**- Q - -**

**- - - Q**

**Q - - -**

**- - Q -**

**- - Q -**

**Q - - -**

**- - - Q**

**- Q - -**

**Program 11**

**Write a program to sort all transactions of Big Mall by quantity of sales.**

**#include<stdio.h>**

**#include<conio.h>**

**struct BIG\_MALL**

**{**

**int item\_no,quantity;**

**float price;**

**char item\_name[20];**

**}s[100],t;**

**void main()**

**{**

**int i,j,n;**

**float temp=3.2f;**

**clrscr();**

**printf("Enter the number of items to purchase n=");**

**scanf("%d",&n);**

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

**{**

**printf("Enter the %d Item number: \n",i+1);**

**scanf("%d",&s[i].item\_no);**

**printf("Enter the Item name without spaces:");**

**scanf("%s",s[i].item\_name);**

**printf("Enter the quantity of items:");**

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

**printf("Enter the PRICE of item purchased");**

**fflush(stdin);**

**scanf("%f",&temp);**

**s[i].price=temp;**

**}**

**printf("\n PURSHACE details are\n");**

**printf("\nItem no\tItem Name\t\Quantity\tPrice\n");**

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

**printf("%d\t%s\t\t%d\t\t%3.2f\n", s[i].item\_no,s[i].item\_name,s[i].quantity,s[i].price);**

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

**{**

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

**{**

**if(s[j].quantity>s[j+1].quantity)**

**{**

**t=s[j];**

**s[j]=s[j+1];**

**s[j+1]=t;**

**}**

**}**

**}**

**printf("\n PURSHACE details after sorting by quantity are\n");**

**printf("\nItem no\tItem Name\t\Quantity\tPrice\n");**

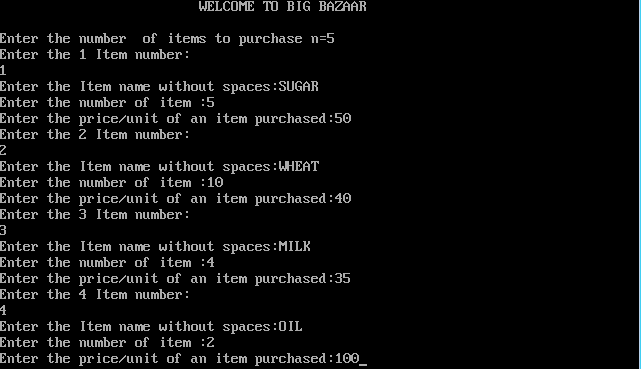
**for(i=0;i<n;i++)**

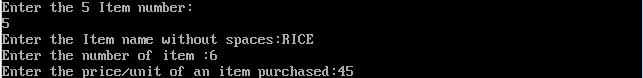
**printf("%d\t%s\t\t%d\t\t%3.2f\n", s[i].item\_no,s[i].item\_name,s[i].quantity,s[i].price);**

**getch();**

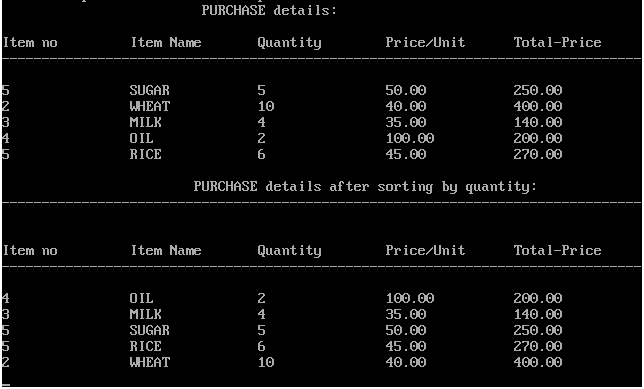
**}**

**INPUT:**

****

****

**OUTPUT:**



**Program 12**

**Write a program to find network of people of same location in Linkedin social network.**