**Social Network Simulation**

Implement a social network using the friendship model (no one way connection between users). Following tables describe the relevant data schemas:

|  |
| --- |
| **User** |
| username (string) |
| first\_name (string) |
| last\_name (string) |

|  |
| --- |
| **Friendship** |
| id (int) |
| requester (username, string) |
| requestee (username, string) |
| accepted (boolean) |

1. Write and implement the needed interfaces to perform basic operations such as creating users and creating friendships.

2. Create functionality that accepts a set of usernames, and reports whether the set forms a fully connected network (all pairs of nodes are connected).

Solution of 1 & 2: Explanation

This Problem is related to Graph Theory. Here Each Username behave like a node of the Graph. To solve this problem I used the R language interface that accept set of the usernames.

For this Netsim Packages have to be installed. A set of the empty process is created that acts as set of the usernames. After that we are defining how many processes is to be created. If we enter 5 username as the process then by default network so created are R matrix. Also since the Network that network is created with parameters directed = TRUE{Firendship}, reflexive = FALSE{NOT friendship}. The matrix has “reflexive" values on the diagonal which are automatically set to 0.

[,1] [,2] [,3] [,4] [,5]

[1,] 0 1 1 1 1

[2,] 1 0 1 1 1

[3,] 1 1 0 1 1

[4,] 1 1 1 0 1

[5,] 1 1 1 1 0

This matrix shows that set of the user {1,2,3,4,5} fully connected to each other. As all pairs of the node are created, so they are in the friendship to each other.

**CODE: R language R version 3.2.1**

install.packages(c("NetSim", "Rcpp"))

library("NetSim") // Loading required package: Rcpp

processState <- create\_process\_state() // empty process state created

nActors <- 10

network <- create\_network(matrix(1, nActors, nActors)) // complete network with5 nodes added to the process

processState <- add\_network(processState, network, name = "friendship") //directed=TRUE{FRIENDSHIP}

get\_network\_index(processState, name = "friendship")

as.matrix(network) // matrix representation

Results:

[,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]

[1,] 0 1 1 1 1 1 1 1 1 1

[2,] 1 0 1 1 1 1 1 1 1 1

[3,] 1 1 0 1 1 1 1 1 1 1

[4,] 1 1 1 0 1 1 1 1 1 1

[5,] 1 1 1 1 0 1 1 1 1 1

[6,] 1 1 1 1 1 0 1 1 1 1

[7,] 1 1 1 1 1 1 0 1 1 1

[8,] 1 1 1 1 1 1 1 0 1 1

[9,] 1 1 1 1 1 1 1 1 0 1

[10,] 1 1 1 1 1 1 1 1 1 0

3. Create functionality that reports the degree of connection between two users.

Solution: Degree of the connection is determined by no. of Edge incidents on the particular vertices.

So to count the no. of the edges on the particular vertices.

C: code to find no. In/Out - Degree of Vertices of a Graph

#include<stdio.h>

#include<conio.h>

typedef struct node

{

struct node \*next;

int vertex;

}node;

node \*g[20];

int n,visited[20];

int indegree(int i);

int outdegree(int i);

void dfs(int i);

void insert(int vi,int vj)

{

node \*p,\*q;

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

q->vertex=vj;

q->next=NULL;

if(g[vi]==NULL)

g[vi]=q;

else

{

p=g[vi];

while(p->next!=NULL)

p=p->next;

p->next=q;

}

}

void readgraph()

{

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

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

g[i]=NULL;

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

scanf("%d",&n);

printf("\nEnter the no of Edges::");

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

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

{

printf("\nEnter the Edge(u,v)::");

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

insert(vi,vj);

}

}

void main()

{

int i,j,k;

clrscr();

readgraph();

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

visited[i]=0;

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

printf("\nNode\tIndegree\tOutdegree");

printf("\n=====================================================");

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

{

j=indegree(i);

k=outdegree(i);

printf("\n%2d\t%4d\t\t%5d",i,j,k);

}

printf("\n-----------------------------------------------------");

\*/

dfs(0);

getch();

}

int outdegree(int i)

{

int j=0;

node \*p;

p=g[i];

while(p!=NULL)

{

p=p->next;

j++;

}

return(j);

}

int indegree(int v)

{

int i,j=0,k;

node \*p;

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

{

p=g[i];

while(p!=NULL)

{

if(p->vertex==v)

j++;

p=p->next;

}

}

return(j);

}

void dfs(int i)

{

node \*p;

p=g[i];

visited[i]=1;

printf("\nVisit->%d",i);

while(p!=NULL)

{

i=p->vertex;

if(!visited[i])

dfs(i);

p=p->next;

}

}

**Note:**

1. Feel free to create any other data types or interfaces that you might need.

2. You may use any programming language that uses a freely available compiler/interpreter that runs on Linux.

3. Include any files needed to build the program in case of compiled languages (e.g. makefile for C)

4. Upload your solution to github and email us a link to the solution repository within two days of us sending you this problem statement.

**Evaluation criteria:**

All submissions will be evaluated on the following parameters:

1. Functionality/usability of solution
2. Software design
3. Algorithm quality
4. Programming skills