**Algorithmic Aspects of Telecommunication Networks**

**CS 6385.001- Fall 2017**

**PROJECT 2**

**Nagamochi-Ibaraki Algorithm**

**Submitted by:**

**Ponmalar Silambaram Chandrabose (pxs162030)**

**CONTENTS:**

1. Minimum cut in undirected graph
2. Objective
3. The Overview
4. Nagamochi- Ibaraki Algorithm
5. Analysis and Charts
6. Graphs
7. Appendix
8. References

Minimum Cut in Undirected Graph:

Objective:

The aim of this project is to

* Implement the Nagamochi-Ibaraki Algorithm for finding the minimum cut in undirected graphs.
* Implement the code to find the edge connectivity and critical edges for randomly generated values of m and n.
* To express the edge connectivity and the critical edges as average degree of the graph.

Overview:

Steps in developing algorithm:

* A node is picked and list of all the nodes connected to it are obtained.
* Then the Maximum Adjacency ordering is performed.
* Last two nodes of the result are picked.
* Connectivity (Gxy) is calculated.
* Overall connectivity is calculated and the average values are displayed.

Nagamochi – Ibaraki Algorithm:

Nagamochi and Ibaraki developed the algorithm that is used to calculate the minimum cut for undirected graph without using flows, paths and Menger’s theorem.

Considerations:

* Let G = (V, E) be an undirected graph with n nodes and m edges in which parallel and looped edges are not allowed.
* Let c(x, y) = c(x, y; G) denote the minimum cardinality of a cut separating x and y.
* ƛ(G) is the minimum of c(x, y) values over all pairs of distinct nodes.
* Let Vi denote the set of first i elements for an ordering of v1,…, vn.

For two disjoint subsets X, Y of nodes d(X, Y) denotes the number of edges connecting X and Y.

We use d(X) = d(X, V-X)

An ordering is legal if

d(V(i-1), vi) >= d(V(i-1), vj) , for every pair i,j (2 < i < j < n).

The algorithm of Nagamochi and Ibaraki is based on the following observation. Let r and gr be two nodes for which d(r) : c(n,A).

If there is a minimum cut of G (i.e. a cut of À(G) elements) separating r and g, then À(G) : d(r) and the star of e is such a cut.

If no minimum cut of G separates r and g, then shrinking c and g into one node does not destroy any minimum cut. In this case it suffices to compute the edge-connectivity of the shrunken graph.

Minimum Cut Algorithm Pseudocode:

Input: A graph G = (V; E).

Output: A minimum cut X in G.

Step 1:Let G1:=G and i:=1.

Step 2: while i<n do

Compute the local edge-connectivity ƛGi(ui, vi)=dG(vi) for the last two vertices ui, vi ∈ Gi, in an MA ordering of Gi (where vi is assumed to be the last vertex), and contract vertices ui, vi into a single vertex, denoting the resulting graph by Gi+1. Let i:=i + 1.

end /\* while \*/

Step 3: Find i = i ∗

that minimizes ƛGi(ui; vi) among all i = 1,2,…,n − 1. Then output the set of vertices X contracted to vi∗ before obtaining Gi∗ .

MA Ordering:

S ← {v1}

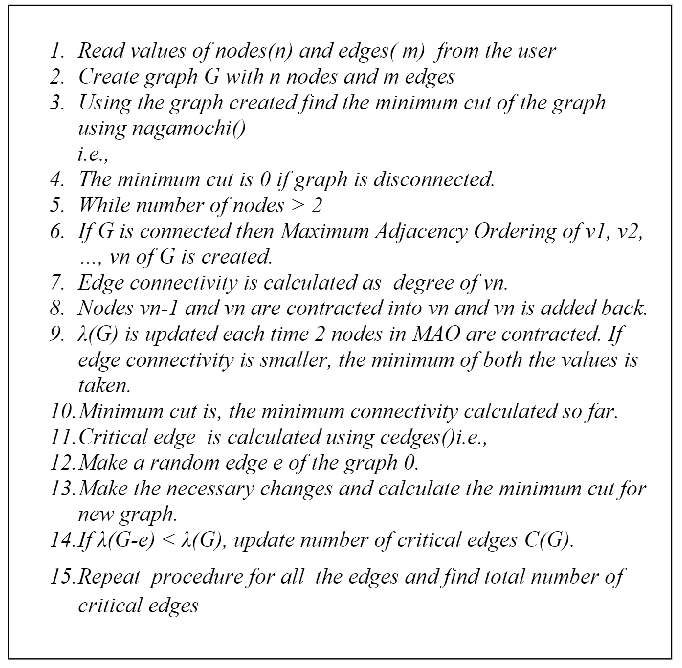
For i ← 2 to n

Choose vi to maximize u(δ(S, {v})) ∀v ∈ V − S

S ← S ∪ {vi}

In each iteration, the algorithm looks at all vertices not in the set S and picks the one which maximizes the capacity of arcs connecting it to nodes in S.

Pseudocode:



Appendix:

#include<stdio.h>

#include <iostream.h>

#include<math.h>

#include<conio.h>

#include<stdlib.h>

using namespace std;

void graphgen(int,int);

void graphgenerator(int, int);

int nagamochi (int);

int cedges(int , int);

int adj[100][100];

int nagibamatrix[100][100];

int cpy[100][100];

int edgeexist[100][100];

int d[100];

//Draw a graph

void graphgen(int i, int j)

{

FILE\* graph; graph=fopen("graph.dot","w");

fprintf(graph,"digraph G {\n");

fprintf(graph,"%d -> %d [dir=none];\n",i,j);

fprintf(graph,"}");

fclose(graph);

}

//Algorithm

int nagamochi(int tot)

{

int max = 0,n=0,l,u,t,lamda=0,start, choosenode, prev1, prev2;

int madj[100];

while (tot > 2 )

{

start = 0;

for (l=0;l l<tot;l++)

madj[l]=0;

choosenode = rand()%tot; //choosing a random node

madj[start] = choosenode;

for (l=1;l<tot;l++)

{

max = 0;

for (u=0;u<tot; u++)

{

if (adj[choosenode][u] > max)

{

max = adj[choosenode][u];

n= u;

}

}

madj[++start]= n;

for (u=0;u<tot;u++)

{

adj[choosenode][u] += adj[n][u];

adj[u][choosenode] += adj[u][n];

adj[choosenode][choosenode] = 0;

}

for (u=0;u<tot;u++)

{

adj[n][u]=adj[u][n]=0;

}

}

if ((!lamda) || (lamda >= d[madj[tot-1]]))

{

lamda = d[madj[tot-1]];

}

prev1 = madj[tot-1];

prev2 = madj[tot-2];

for (l=0; l < tot; l++)

{

nagibamatrix[prev2][l] += nagibamatrix[prev1][l];

nagibamatrix[l][prev2] += nagibamatrix[l][prev1];

nagibamatrix[prev2][prev2] = 0;

nagibamatrix[prev1][l] = nagibamatrix[tot-1][l];

nagibamatrix[l][prev1] = nagibamatrix[l][tot-1];

}

tot = tot-1;

for (l=0; l<tot; l++)

for (t=0; t <tot; t++)

adj[l][t]=nagibamatrix[l][t];

}

return (lamda);

}

//\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_FUNCTION TO CREATE GRAPH\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

void graphgenerator(int n, int maxedges)

{

int i,j,p,q,x;

int cnt = maxedges;

while(cnt>0)

{

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

{

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

{

if ((i == j))

{

adj[i][j] = adj[j][i] = 0;

edgeexist[i][j] = edgeexist[j][i] = 1;

}

else

{

if ((edgeexist[i][j] == 0) && (edgeexist[j][i] == 0))

{

if ((adj[i][j] == 0) && (adj[j][i] == 0))

{

x = (rand()%1);

if (x = 1)

{

adj[i][j] = adj[j][i] = 1;

cnt--;

d[i]=d[i]+1;

d[j]=d[j]+1;

edgeexist[i][j] = edgeexist[j][i] = 1;

graphgen(i,j);

if (cnt <= 0)

break;

}

else

{ adj[i][j] = adj[j][i] = 0;

edgeexist[i][j] = edgeexist[j][i] = 1;

}

}

}

}

}

if (cnt <= 0)

break;

}

if (cnt)

{

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

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

edgeexist[p][q]= 0;

}

}

}

//\_\_\_\_\_FUNCTION TO COMPUTE CRITICAL EDGES\_\_\_\_\_\_\_

int cedges(int n, int l){

int cnt = 0,i,j;

int criticalarray[100];

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

{

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

{

if(adj[i][j]==1)

{adj[i][j]=0;

adj[j][i]=0;

criticalarray[i]=d[i];

criticalarray[j]=d[j];

adj[i][j]=1;

adj[j][i]=1;

}

}

}

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

{

if(nagamochi(d[i])>l)

{

cnt++;

}

}

return cnt;

}

int main()

{

int di,q,lr,h,l,criticaledges,n,p,m,b;

cout<<"\n"<<"\t"<<"\t"<<"\*NAGAMOCHI-IBARAKI ALGORITHM\*"<<"\n";

cout<<"\n"<<"\*\*\*\*\*\*ENTER THE NUMBER OF NODES\*\*\*\*\*\*\*:"<<"\n";

cin>>n;

cout<<"\n"<<"\*\*\*\*\*\*ENTER THE NUMBER OF EDGES\*\*\*\*\*\*\*\*:"<<"\n";

cin>>m;

l = 0;

di = 2\*m/n;

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

{

d[lr]=0;

}

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

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

edgeexist[p][q]= 0;

graphgenerator(n,m);

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

{

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

{

nagibamatrix[b][h]=adj[b][h];

cpy[b][h]=adj[b][h];

}

}

l = nagamochi(n);

criticaledges = cedges(n,l);

cout<<"\n "<<"No. OF EDGES"<<"\t"<<" DEGREE"<<" \t "<<"LAMBDA "<<"\t"<<" CRITICAL EDGES"<<" \n";

cout<<"\n "<<m<<" \t\t "<<di<<"\t\t "<<l<<"\t\t "<<criticaledges;

getch();

return 0;

}

References:

1. <http://web.cs.elte.hu/~frank/cikkek/FrankR2.pdf>
2. <https://people.orie.cornell.edu/dpw/orie633/LectureNotes/lecture7.pdf>