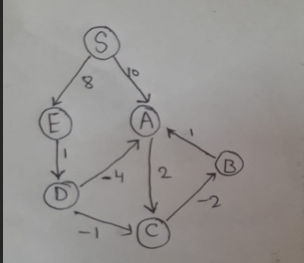
Rushikesh Muraharisetty

Reg.no 21011102067

**DAA - CIA 1**

To find the shortest path from source node to all possible nodes with PRIMS, KRISKALS, DIJKSTRA'S algorithms using C/C++/Python/Java. Give a justification for the choice of the programming language. Give a note of comparison regarding the performance of the algorithms. If you feel that any of the algorithms fail to find any of the required paths, give justification for the same.

The given graph:

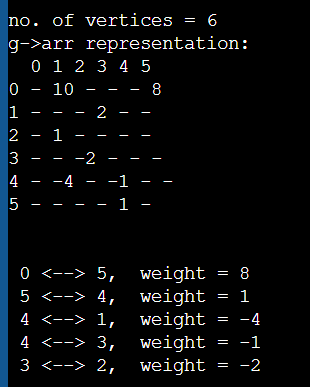


I have chosen C programming language. My justification :  
-> I want to choose a language which supports pointers and is relatively easy to use. Hence ruled out Python and Java  
-> I don’t need object oriented approach to do this analysis, hence I ruled out C++ and sticked with C. My previous semesters’ experience is also a major criteria to choose C.

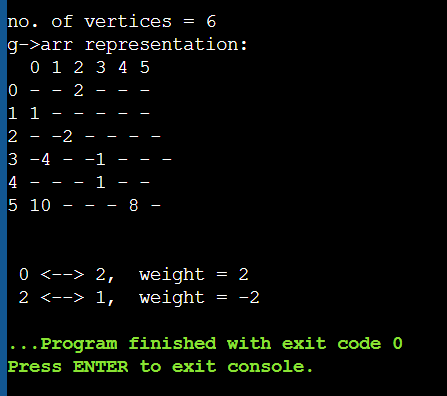
PRIMS analysis:  
  
Prims algorithm works by comparing nearest vertices connected to the visited nodes and comparing the weights to select the lowest weight to produce minimum spanning tree. It generally is applied for undirected graphs.

For the given case, we have taken source as S. We have taken the index of S as 0 which is the source index as said by the question.   
as said, the algorithm will work based on the adjacency symmetric matrix. There is always a path from one node to other.   
The problem with directed graphs is that, there might not be a path from one node to other nodes. In such case, the algorithm fails and the program either terminates with segmentation fault or gives ambiguous/incorrect outputs.

For the given graph, from S, considering the index is 0 for it , will produce right output.



As said, there are cases where path wouldn’t exist, for example for the same graph, if considered A as the source, and run, it will not produce proper output, instead terminated with segmentation fault.

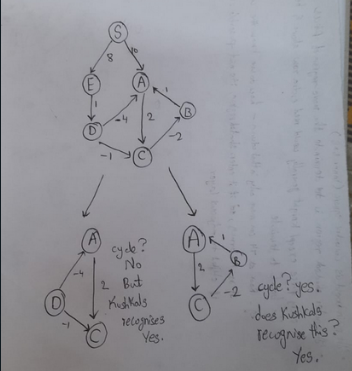


Hence my inferences discussed for prims algorithm.

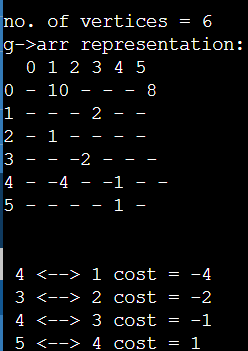
KUSHKALS analysis:

Kushkals algorithm works on the basis of cycle formation. It will take edges increasing order but avoiding cycles is the only thing it checks.

For directed graphs, there might exist subgraphs where there is no cycle according to direction but is a cycle when removed the pointer.



However for this case, the ambiguity is missed. And Output was delivered. However kruskals cant be used for directed graphs.



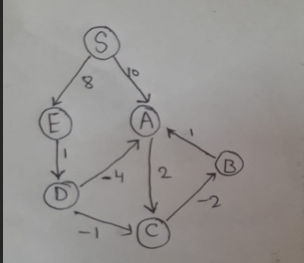
Hence, my inferences are discussed for kruskals algorithm.

Dijkistras analysis:

Dijkistras algorithm is working fine for graphs which are directed because it considers adjacent vertices irrespective of symmetry of the adjacency matrix.

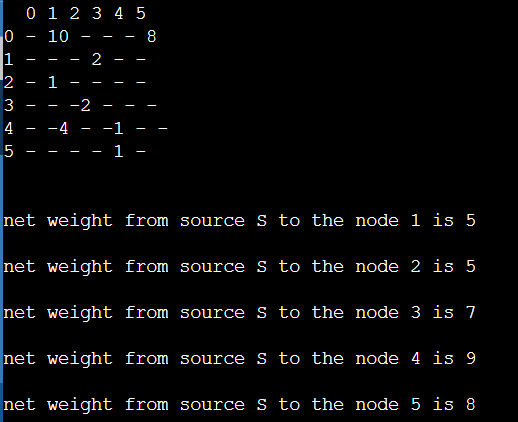
Dijkistras is iterative algorithm. It will go through all nodes one by one and updates the path net weights.

For the given graph,



Dijkistras will not reconsider visited nodes so if there is a negative weight, its possible to miss the ideal path.

Our case is lucky even though negative weights are included because there are no cycles which sum to negative where the algorithm fails to detect.



Hence my inferences discussed for dijkistras algorithm.

**Programs:**

PRIMS:

#include <stdio.h>

#include <stdlib.h>

#include <limits.h>

#define I INT\_MAX

struct graph

{

int\* \*arr ;

int v ;

int\* varr;

};

void display(struct graph\*);

void create(struct graph\*);

void prims(struct graph\* g)

{

int isVisit[g->v];

int choice = 0, weight = 0;

int u = 0, v = 0;

for(int i = 0; i<g->v; i++)

isVisit[i] = 0;

isVisit[0] = 1;

for(int count = 0 ; count < g->v; count++){

weight = I;

for(int i = 0; i<g->v; i++)

{

for(int j = 0; j<g->v; j++)

{

if(g->arr[i][j]

< weight &&

isVisit[i] ){

weight = g->arr[i][j];

u = i; v = j;

}

}

}

if( !isVisit[u] || !isVisit[v] )

{

printf("\n %d <--> %d, weight = %d",u,v,weight);

isVisit[v]=1;

}

g->arr[u][v] = g->arr[v][u] = I;

}

}

int main(){

struct graph g ;

create(&g);

display(&g);

prims(&g);

return 0;

}

void create(struct graph\* g)

{

g->v = 6 ;

g->varr = (int\* )malloc(sizeof(int)\*g->v );

g->arr = (int\*\*)malloc(sizeof(int)\*g->v\*g->v) ;

for(int i = 0 ; i < g->v ; i++)

g->arr[i] = (int\* )malloc(sizeof(int)\*g->v ) ;

for(int i=0; i<g->v ;i++)

{

g->varr[i] = i ;

}

for(int i = 0 ; i < g->v ; i++)

for(int j = 0 ; j < g->v ; j++)

g->arr[i][j] = I ;

g->arr[1][3] = 2 ; //

g->arr[2][1] = 1 ;//

g->arr[3][2] = -2 ;//

g->arr[4][1] = -4 ; //

g->arr[4][3] = -1 ;//

g->arr[5][4] = 1 ;//

g->arr[0][1] = 10; //

g->arr[0][5] = 8 ; //

}

void display(struct graph\* g)

{

printf("\n\nno. of vertices = %d \n",g->v);

printf("g->arr representation: \n");

printf(" ");

for(int i = 0; i < g->v ; i++)

printf("%d ",g->varr[i]);

printf("\n");

for(int i = 0 ; i < g->v ; i++)

{

printf("%d ",g->varr[i]);

for(int j = 0 ; j < g->v ; j++){

if(g->arr[i][j]==I)printf("- ");else

printf("%d ",g->arr[i][j]);}

printf("\n");

}

printf("\n");

}

KRUSKALS:

#include <stdio.h>

#include <stdlib.h>

#include <limits.h>

#define I INT\_MAX

struct graph

{

int\* \*arr ;

int v ;

int\* varr;

};

void display(struct graph\*);

void create(struct graph\*);

int set(int\*, int);

int isCycle(int\* arr, int i, int j)

{

i = set(arr, i);

j = set(arr, j);

if((i != j))

{

arr[j] = i;

return 1;

}

return 0;

}

int set(int\* arr, int tmp)

{

while(1)

{

if(!arr[tmp])

break;

else

tmp = arr[tmp];

}

return tmp;

}

void kushkals(struct graph\* g)

{

int parent[g->v];

int choice = 0, weight = 0, a = 0, b = 0;

for(int i = 0; i<g->v; i++)

parent[i] = 0;

for(int count =1 ; count < g->v; count++)

{

weight = I;

for(int i = 0; i<g->v; i++)

for(int j = 0; j<g->v; j++)

if(g->arr[i][j] < weight)

{

weight = g->arr[i][j];

a = i; b = j;

}

if(isCycle(parent, a, b))

{

printf("\n %d <--> %d cost = %d",a,b,weight) ;

}

g->arr[a][b] = I;

g->arr[b][a] = I;

}

}

int main(){

struct graph g ;

create(&g);

display(&g);

kushkals(&g);

return 0;

}

void create(struct graph\* g)

{

g->v = 6 ;

g->varr = (int\* )malloc(sizeof(int)\*g->v );

g->arr = (int\*\*)malloc(sizeof(int)\*g->v\*g->v) ;

for(int i = 0 ; i < g->v ; i++)

g->arr[i] = (int\* )malloc(sizeof(int)\*g->v ) ;

for(int i=0; i<g->v ;i++)

{

g->varr[i] = i ;

}

for(int i = 0 ; i < g->v ; i++)

for(int j = 0 ; j < g->v ; j++)

g->arr[i][j] = I ;

g->arr[1][3] = 2 ; //

g->arr[2][1] = 1 ;//

g->arr[3][2] = -2 ;//

g->arr[4][1] = -4 ; //

g->arr[4][3] = -1 ;//

g->arr[5][4] = 1 ;//

g->arr[0][1] = 10; //

g->arr[0][5] = 8 ; //

}

void display(struct graph\* g)

{

printf("\n\nno. of vertices = %d \n",g->v);

printf("g->arr representation: \n");

printf(" ");

for(int i = 0; i < g->v ; i++)

printf("%d ",g->varr[i]);

printf("\n");

for(int i = 0 ; i < g->v ; i++)

{

printf("%d ",g->varr[i]);

for(int j = 0 ; j < g->v ; j++){

if(g->arr[i][j]==I)printf("- ");else

printf("%d ",g->arr[i][j]);}

printf("\n");

}

printf("\n");

}

DIJKISTRAS:

#include <stdio.h>

#include <stdlib.h>

#include <limits.h>

#define I INT\_MAX

struct graph

{

int \*\*arr;

int v;

int \*varr;

};

void create (struct graph \*g)

{

g->v = 6;

g->varr = (int \*) malloc (sizeof (int) \* g->v);

g->arr = (int \*\*) malloc (sizeof (int) \* g->v \* g->v);

for (int i = 0; i < g->v; i++)

g->arr[i] = (int \*) malloc (sizeof (int) \* g->v);

for (int i = 0; i < g->v; i++)

{

g->varr[i] = i;

}

for (int i = 0; i < g->v; i++)

for (int j = 0; j < g->v; j++)

g->arr[i][j] = I;

g->arr[1][3] = 2; //

g->arr[2][1] = 1; //

g->arr[3][2] = -2; //

g->arr[4][1] = -4; //

g->arr[4][3] = -1; //

g->arr[5][4] = 1; //

g->arr[0][1] = 10; //

g->arr[0][5] = 8; //

}

void display (struct graph \*g)

{

printf ("\n\nno. of vertices = %d \n", g->v);

printf ("g->arr representation: \n");

printf (" ");

for (int i = 0; i < g->v; i++)

printf ("%d ", g->varr[i]);

printf ("\n");

for (int i = 0; i < g->v; i++)

{

printf ("%d ", g->varr[i]);

for (int j = 0; j < g->v; j++)

{

if (g->arr[i][j] == I)

printf ("- ");

else

printf ("%d ", g->arr[i][j]);

}

printf ("\n");

}

printf ("\n");

}

void djk(struct graph\* g)

{

int isVisit[g->v], path[g->v];

int a = 0, b = 0;

for(int i=0; i<g->v; i++)

{

isVisit[i] = 0;

path[i]=g->arr[0][i];

isVisit[i]=0;

}

isVisit[0] = 1; // the source index is 0

for(int count=1; count<g->v; count++)

{

int weight = I;

for(int i = 0; i<g->v; i++)

if(path[i]<weight && !isVisit[i])

{

weight = path[i];

a = i;

}

isVisit[a] = 1;

for(int i = 0; i < g->v; i++)

if(!isVisit[i])

if(weight+g->arr[a][i] < path[i])

path[i] = weight + g->arr[a][i];

}

for(int i = 1; i<g->v; i++)

printf("\nnet weight from source S to the node %d is %d \n",i,path[i]);

}

int main()

{

struct graph g;

create (&g);

display(&g);

djk(&g);

return 0;

}