DA-ALG1000

Minimum Searching Tree using Prim´s Algorithm

By:

Amer Sisic & John-Ivar Hauge

Summary:

This report is showing how Prim´s algorithm can be implemented in java-language. It describes the process needed to get a successful algorithm using matrices.

Contents

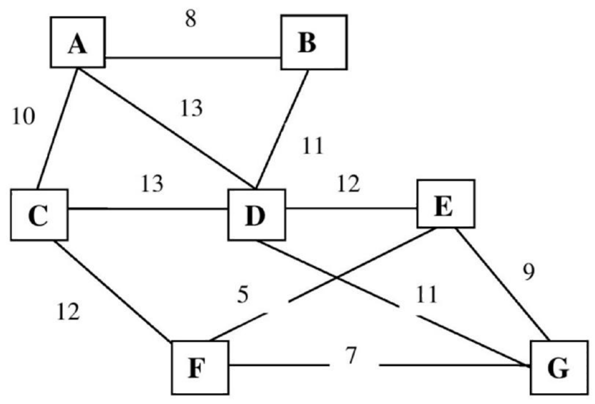
Task given:

Obligatorisk oppgave nr. 2 i DA-ALG1000 vår 2015

I følgende figur tenker vi oss at nodene A, B, C, D, E, F og G indikerer hus. Kantene mellom husene indikerer at det kan trekkes en kabel for telefon mellom 2 hus til en kostnad av det tallet som er oppgitt ved den tilhørende kanten. Prisen er oppgitt i antall 1000 kr. Programmér en løsning slik at alle hus er tilknyttet og at total kostnad er minst mulig. Man skal kunne finne samme løsning uansett hvilken node/hus man starter i. Programmet **skal** skrive ut:

> **total kostnad**
> **hvilke hus som parvis velges underveis med tilhørende kostnad**

Ønsker ikke å se en løsning som kan spores til å ha blitt funnet på Internett. Oppgaven leveres som én fil (gjerne .zip) og rapport skal være med (i samme fila!). Oppgaven er tenkt løst vha. en tabell/matrise-representasjon av grafen, men forsøk dere gjerne på en annen metode. Lykke til!

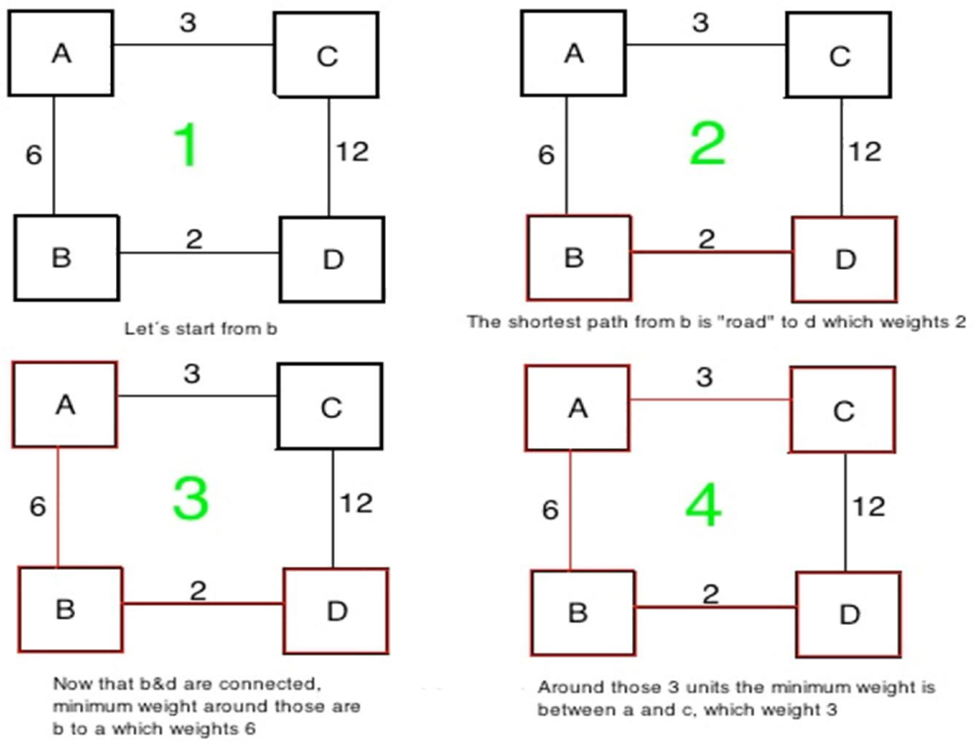


What is Prim´s algorithm, and how does it work?

Assume you have network of n-houses which you want to connect together with a road, roads can have different cost between houses and you want to find the minimum cost where all of the houses are connected together from the first house to the nth house.

Prim´s algorithm tells us that you can start at any given house and find the road that is "cheapest/minimum weight" which is connected to the other house. Now those two houses form a unit, next time we look to connect a road, you have to find the minimum cost of the road to the next house then those 3 houses form a unit, and so on..

In the end you will end up with n-1 roads that are connected to between houses, and the minimum cost is the sum of the each road-cost added together. Figure 1 shows how an example of Prim´s algorithm


A
3
C
6
1
12
B
2
D
Let´s start from b
A
3
C
6
2
12
B
2
D
The shortest path from b is "road" to d which weights 2
A
3
C
6
3
12
B
2
D
Now that b&d are connected, minimum weight around those are b to a which weights 6
A
3
C
6
4
12
B
2
D
Around those 3 units the minimum weight is between a and c, which weight 3

Figur 1Prim´s algorithm visualization

As figure 1 shows we started at b and found the shortest path was to d, and around those units we have shortest path from b to a, and shortest path between those three is a to c.

Last we have c to d, but because b to d is already connected together, and b to a, there is a connection between c and d, so we don´t need to connect them. In the end we end up with 3 “roads” where the total-weight is 2+6+3 = 11.

Implementation:

This java implementation of Prim´s algorithm uses matrices to solve the problem. The way it works is that user inputs a double array which contains weight of the edges that are connected from node to node, nodes that are not connected are inputted as 0 or less.

Once the double array is created, then that array goes into a method called prepareArray which prepares the array by setting anything that is 0 or less to Integer.MAX_VALUE. The reasoning for this is because it will be easier to find minimum value of each column later.

```
public static void prepareArray(int a[][])
	{
		for(int i = 0; i <a.length ; i++){
			for(int j = 0; j < a.length; j++)
				if(a[i][j]<=0)//Anything less or equal 0 is changed to MAX_VALUE
					a[i][j] = Integer.MAX_VALUE;
		}

	}
```

After this a new method is called, void minimumWeight which uses two inputs, one is the prepared array and second is index to start with. Here is where all of the calculation is happening. Inside we have several variables and arrays. When user enters the index to start with, the visited array starts with that index. Visited array contains elements of boolean type true-false, initially all the elements are false.

```
public static void minimumWeight(int a[][],int n){//n is the index user want to start in
```

```
boolean visited[] = new boolean[a.length];//Visited array to check if elements in a
have been visited
```

After the selected index is set as true, then program enters the first loop, this loop goes n-1 times (amount of edges)

```
for(int edge= 0; edge< a.length-1 ; edge++){//edge = 1 less than the array length
```

Each time this loop enters new state a min variable is defined as Integer.MAX_VALUE, this way we can check later for minimum values inside the matrix.

Next two loops are for rows and columns; first one checks the row and second column in that row.

Each time row-loop enters a new state, it checks if that row is visited, if that value is true then we go over to the last loop.

```
for(int i = 0; i < a.length; i++){//Rows
				if(visited[i]==true){//if this row has been visited then enter column for-
loop
					for(int k = 0; k < a.length; k++){//Columns
```

In the column loop, we check if that column has been visited and if not (visited is set as false), then a check is performed to find the lowest value in that column.

```
if(visited[k]==false){//donÂ´t visit allready visited elements
                if(min > a[i][k]){
                        min = a[i][k];//Set the new minimum
                        iCopy=i;
                        kCopy=k;
```

The reasoning for iCopy and kCopy is to know what edge the code is in, and later stored in array.

Once the checks are done, visited is set as true.

```
visited[kCopy]=true;//Set as visited

totalWeight += min;//increase weight with minimum values
stored[edge]=""+nodes[iCopy]+"--->"+nodes[kCopy]+"="+min+" ";//Store the cost between edges
```

This continues to all edges are visited, and visited array elements contains only true variables. When that is over, we get the minimum value and the value between of the each edge printed out.

```
System.out.print("Total weight of the edges: "+ totalWeight+"\n");//Print the total weight
System.out.print(Arrays.toString(stored));//Print the cost between edges
```

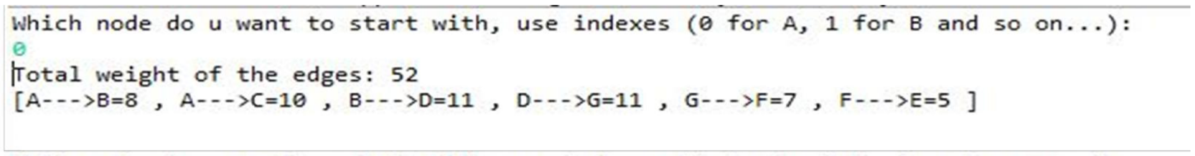
As we can see from the code Big O notation of the program is O(n^3).

Testing:

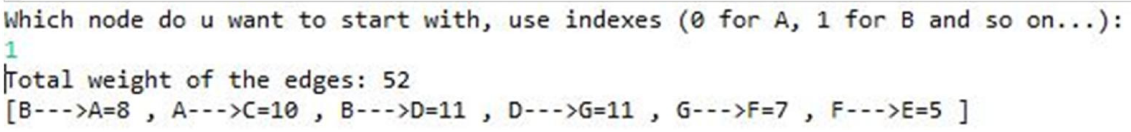
The following tests were done:

- Check if the result is same with different start index
- What happens if start index is out of bounds

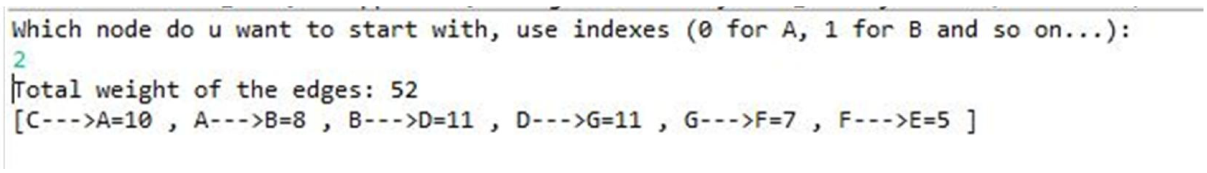
Checking if the result is same through all of start indexes:



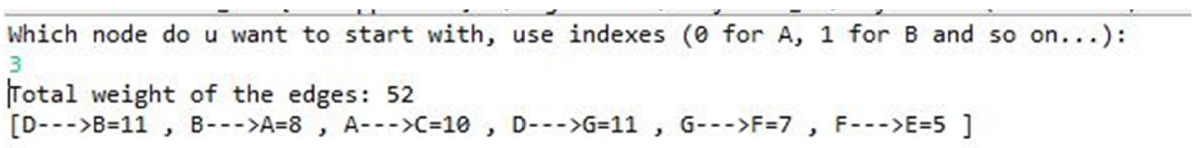
```
Which node do u want to start with, use indexes (0 for A, 1 for B and so on...):
0
Total weight of the edges: 52
[A--->B=8 , A--->C=10 , B--->D=11 , D--->G=11 , G--->F=7 , F--->E=5 ]
```



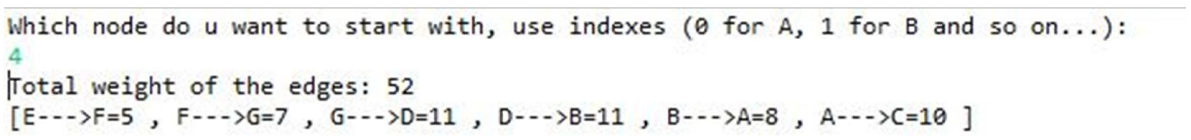
```
Which node do u want to start with, use indexes (0 for A, 1 for B and so on...):
1
Total weight of the edges: 52
[B--->A=8 , A--->C=10 , B--->D=11 , D--->G=11 , G--->F=7 , F--->E=5 ]
```



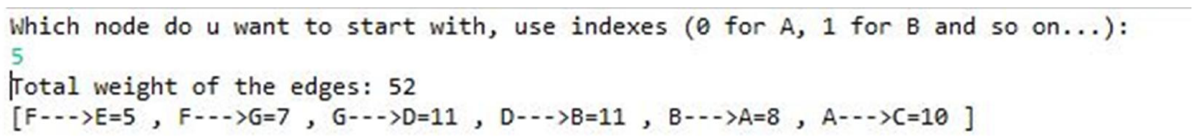
```
Which node do u want to start with, use indexes (0 for A, 1 for B and so on...):
2
Total weight of the edges: 52
[C--->A=10 , A--->B=8 , B--->D=11 , D--->G=11 , G--->F=7 , F--->E=5 ]
```



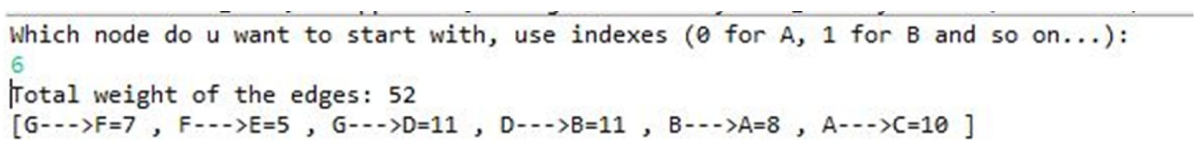
```
Which node do u want to start with, use indexes (0 for A, 1 for B and so on...):
3
Total weight of the edges: 52
[D--->B=11 , B--->A=8 , A--->C=10 , D--->G=11 , G--->F=7 , F--->E=5 ]
```



```
Which node do u want to start with, use indexes (0 for A, 1 for B and so on...):
4
Total weight of the edges: 52
[E--->F=5 , F--->G=7 , G--->D=11 , D--->B=11 , B--->A=8 , A--->C=10 ]
```

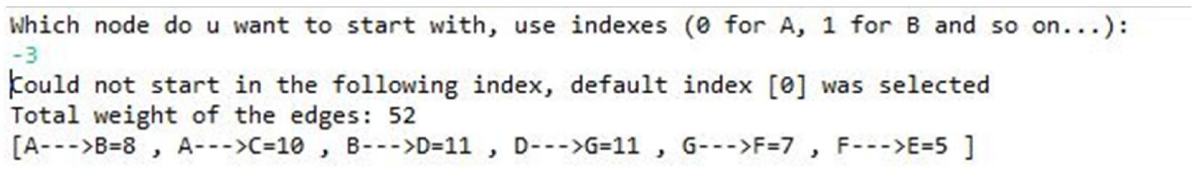


```
Which node do u want to start with, use indexes (0 for A, 1 for B and so on...):
5
Total weight of the edges: 52
[F--->E=5 , F--->G=7 , G--->D=11 , D--->B=11 , B--->A=8 , A--->C=10 ]
```

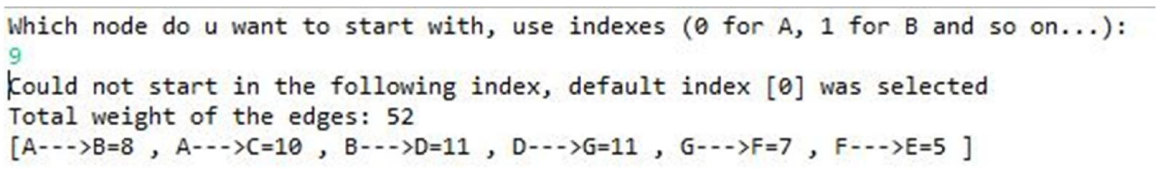


```
Which node do u want to start with, use indexes (0 for A, 1 for B and so on...):
6
Total weight of the edges: 52
[G--->F=7 , F--->E=5 , G--->D=11 , D--->B=11 , B--->A=8 , A--->C=10 ]
```

If start index is larger than array.length or less than 0:



```
Which node do u want to start with, use indexes (0 for A, 1 for B and so on...):
-3
Could not start in the following index, default index [0] was selected
Total weight of the edges: 52
[A--->B=8 , A--->C=10 , B--->D=11 , D--->G=11 , G--->F=7 , F--->E=5 ]
```



```
Which node do u want to start with, use indexes (0 for A, 1 for B and so on...):
9
Could not start in the following index, default index [0] was selected
Total weight of the edges: 52
[A--->B=8 , A--->C=10 , B--->D=11 , D--->G=11 , G--->F=7 , F--->E=5 ]
```

Conclusion:

The algorithm works fine; user can start from any point and still get same results on total-weight of edges. Biggest concerns is the O(n^3) which can be a problem for large network of object that are connected together. Also the needing of go through the arrays to exchange 0 to max values is probably not the best solution, and could have been implemented better.