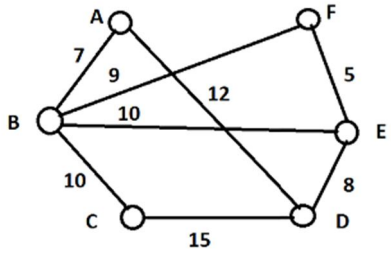
This Lecture is devoted in Analyzing and finding out the Time complexity of Dijkstra's Greedy Choice Algorithm

Let us start with an example graph as following::

Now, we are going to calculate the shortest distance of C to all other vertices in the weighted graph.



The Algorithm includes the following steps ->

1) Look at all the neighbours and try to find out the best candidate.
2) Technique to find the best candidate is traverse through the "cost" column in the matrix table and check each candidate for minimum cost and keep on traversing , if found some cost less than previous than make it minimum and candidate as the best candidate but first you need to check it should not be marked in list of final candidates in the table.

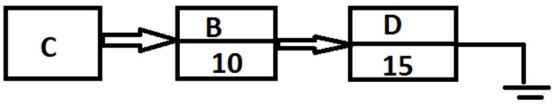
The initial Table is ->

Vertex	**Cost**	**Parent**	**isFinal**
A	inf.		No
B	inf.		No
C	0	C	Yes
D	inf.		No
E	inf.		No
F	inf.		No

This time the best candidate is chosen as C as its cost is less than all other candidates and is then marked as isFinal.

For next iteration with C as parent node

The adjacency list for C is shown as::



For next iteration,

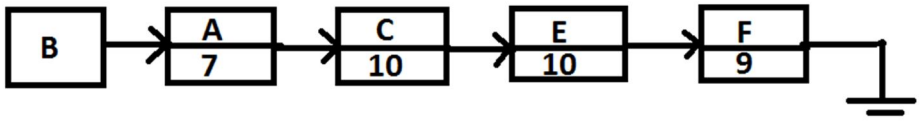
The new Table is ->

Vertex	Cost	Parent	isFinal
A	inf.		No
B	10	C	Yes
C	0	C	Yes
D	15	C	No
E	inf.		No
F	inf.		No

This time the best candidate is chosen as B as its cost is less than all other candidates (skip C cause it is already in isFinal marked) and is then marked as isFinal.

For next iteration with B as parent node

The adjacency list for B is shown as::



For next iteration,

The new Table is ->

Vertex	Cost	Parent	isFinal
A	17	B	No
B	10	C	Yes

C	0	C	Yes
D	15		Yes
E	20	B	No
F	19	B	No

This time the best candidate is chosen as D as its cost is less than all other candidates (skip C & B cause they are already in isFinal marked) and is then marked as isFinal.

*The process is then repeated further.

3) Start again to find and choose the best candidate. Remember every time you pick a best candidate it should be then marked as final in isFinal column of the table.
4) The choice of best candidate is done based on its prior cost irrespective of the fact who is its parent.
5) Keep on repeating the process with above points in mind.

The Time complexity of various steps can be written as::

Creation of table: **O(**linear in terms of degree of vertex to add**) .**

For picking up the minimum , we have to do it for each vertex and have to go through full adjacency list of that vertex .

Time complexity of the Algorithm is:: **O(**$E + V^2$**) .** Since at each step we are required to pick up a Edge with minimum cost and then recalculate the new costs prior to it and then we have to do it for each Vertex hence the given order.

The order can be further reduced as **O(**V^2**).** Since the maximum value of E can be V^2.

The total number of Edges to appear in all adjacency lists is **2E**(since each edge appears twice).

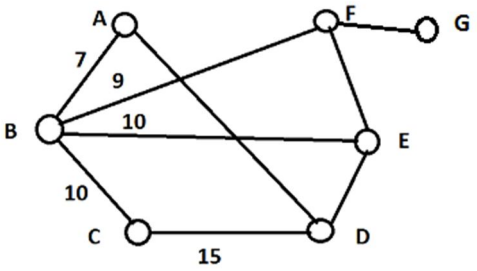
Now, we will try to reduce the asymptotic time complexity with the use of heaps in place of linear algorithm ->

The following points are there to be understood:

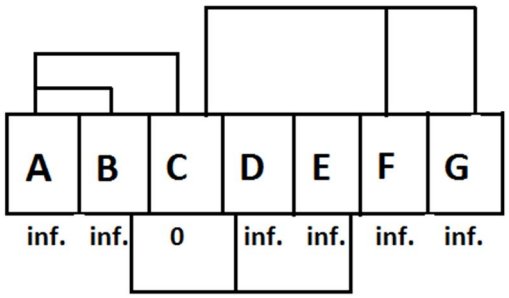
1) The Time Complexity for creating the Heap is **O(n)** which here is **O(v)**.

2) If we are going to pluck out the minimum value we have to go back to the Heap change its parent & weights associated which is in a way changing the heap itself.
3) For performing step(2) we need to do Increment and Decrement weight operations in the Heap.
4) When we are changing the elements in heap we need to add an extra column showing the position of element in Table (its row), but for this too we require an additional pointer from element in heap to its respective row in Table in order to locate its row in table.
5) Here we require index values and pointers for continuous mapping between table rows and Heap's leafs and root.

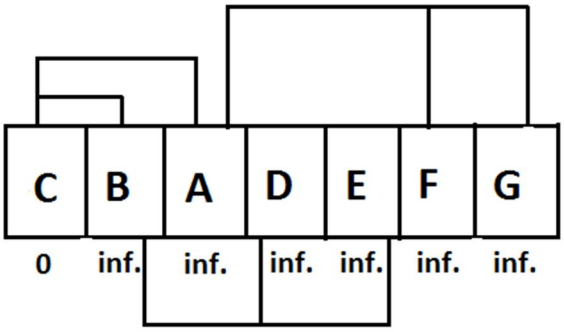
Taking the same graph with an added Edge FG and vertex G:



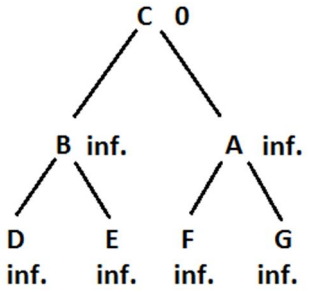
Creating Heap using this graph considering the alphabet coming first as first priority:



The parents and roots are shown here which are to be represented in heap but we are needed to create MIN HEAP for that the root node should be the minimum. Thereby we need to swap A and C and initialize the heap as follows:



The heap will be:



The initial Table is ->

Vertex	Cost	Parent	isFinal	Pointer to heap	Index
A					3
B					2
C					1
D					4
E					5
F					6
G					7

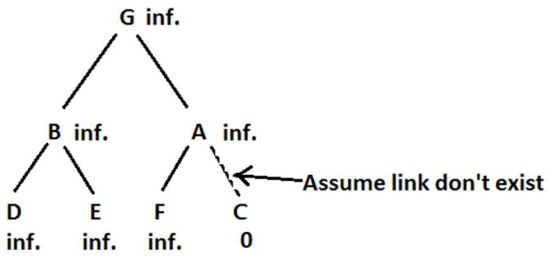
The remaining entries will be as in prev. case at initialization.

Consider the values of index starting from the root incrementing till the bottom and from left to right according to array defining the heap.

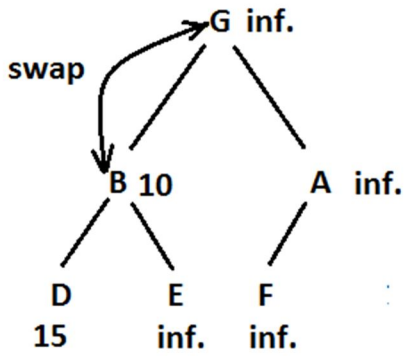
Now performing the operations as discussed earlier but this time on heap.

Removing minimum value from the heap and heapifying ->

Pluck out the minimum value (root of heap) , send it to the bottom assume that link does not exist in reality.



Now go to the adjacency list of C , Do as prev. time change the costs and use Decrement weight operation on heap to Heapify it.



Here we supply the weights and try to create Min HEAP based on value of weights of vertices. Pushing the lowest values to root . Here first swap B with G and then D with G to get min Heap.

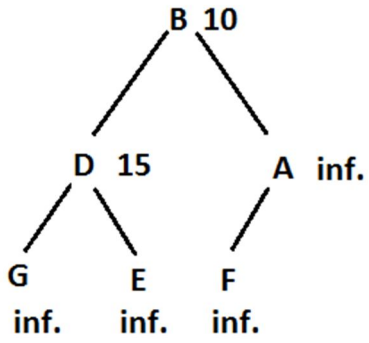
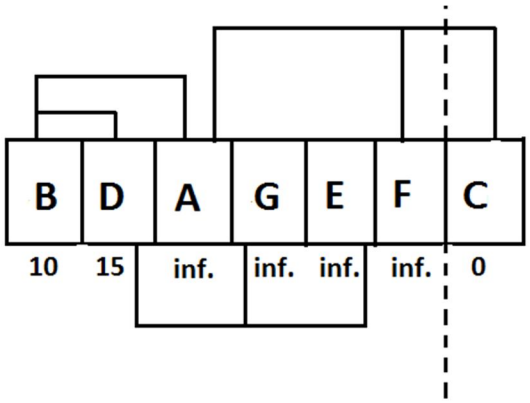


Table represented as:

Vertex	Cost	Parent	isFinal	Pointer to heap	Index
A	inf.		no		3
B	10	C	no		1
C	0	C	yes		7
D	15	C	no		2
E	inf.		no		5
F	inf.		no		6
G	inf.		no		4



Now when done with adjacency list for C in order to get minimum in next step pluck out B from root bring F to root & B to the bottom. And keep on doing this operation further.

Now the Time Complexity measurements are::

1) Every time we pluck out a element from the root of the heap we need to heapify it by swapping nodes of the order of height of heap swaps are made= **O(logV).**
2) Time to build heap is **O(V)**.
3) So the remaining condition for Algorithm is same as previous case and we thereby obtain the **average time** complexity as **O((E + V)*logV)**.

4) For the worst case where **E= V^2** we have time complexity as **O(V^2)** which is same as in prev. case.

Here is a link for getting a smaller version of this text for those who still want more→

www.cs.nott.ac.uk/~nza/G52ADS/dijkstra1.pdf

Created and Submitted by:: Rajat Kulshreshtha (11010846)

Dated on :: 8 March 2013.