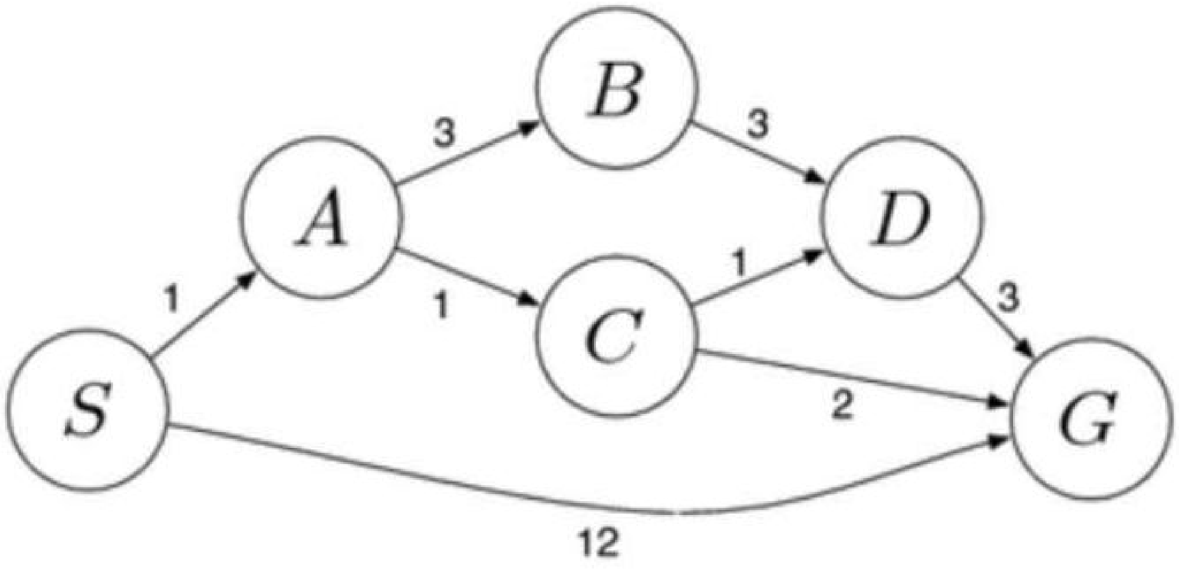
UCS implementation using priority queue

Algorithm

```
Insert the root into the queue
While the queue is not empty
        Dequeue the maximum priority element from the queue
        (If priorities are same, alphabetically smaller path is
        chosen)
        If the path is ending in the goal state, print the path and exit
        Else
        Insert all the children of the dequeued element, with the
        Cumulative (total) costs as priority.
```

Example

Now let us apply the algorithm on the graph below to find out least cost path between **S** and **G**. Here example graph is directed one but **UCS** can be apply if it is bidirectional or undirected graph also. We will go through each iteration and look at the final output. Each element of the priority queue is written as [path , cumulative cost/total cost].



/*put root node (S) into queue with its total cost (0)*/
Init : { **[S , 0]** }.

/*Dequeue **[S , 0]** check whether goal reached, if not, queue all the children (**S-->A , S-->G**) of **S** with total cost as a priority.*/
Step 1 : { **[S-->A , 1]** , **[S-->G , 12]** }.

/*Dequeue **[S-->A , 1],** check whether goal reached, if not, queue all the children (**S-->A-->B , S-->A-->C**) of **A** with total cost as a priority.*/
Step 2 : { **[S-->A-->C , 2]** , **[S-->A-->B , 4] , [S-->G , 12]** }.

/*Dequeue **[S-->A-->C , 2],** check whether goal reached, if not, queue all the children (**S-->A-->C-->D , S-->A-->C-->G**) of **C** with total cost as a priority. */
Step 3 : { **[S-->A-->C-->D , 3]** , **[S-->A-->B , 4] , [S-->A-->C-->G , 4] , [S-->G , 12]** }.

/*Dequeue **[S-->A-->C-->D , 3],** check whether goal reached, if not, queue all the children (**S-->A-->C-->D->G**) of **D** with total cost as a priority. */
Step 4 : { **[S-->A-->B , 4] , [S-->A-->C-->G , 4] , [S-->A-->C-->D-->G , 6] , [S-->G , 12]** }.

/*Dequeue **[S-->A-->B , 4],** check whether goal reached, if not, queue all the children (**S-->A-->B-->D**) of **B** with total cost as a priority. */
Step 5 : { **[S-->A-->C-->G , 4] , [S-->A-->C-->D-->G , 6] , [S-->A-->B-->D , 7] , [S-->G , 12]** }.

/*Dequeue **[S-->A-->C-->G , 4],** check whether goal reached, yes we reached the goal **G***/
Step 5 : final path **[S-->A-->C-->G , 4]**

Some Findings

Things worth mentioning:

The algorithm returns the first path encountered. It does not search for all paths.

The algorithm returns a path which is optimal in terms of cost.

At any given point in the execution, the algorithm never expands a node which has a cost greater than the cost of the shortest path in the graph. The elements in the priority queue have almost the same costs at a given time, and thus the name Uniform Cost Search. It may seem as if the elements don't have almost the same costs, from the above example. But when applied on a much larger graph it is certainly so.