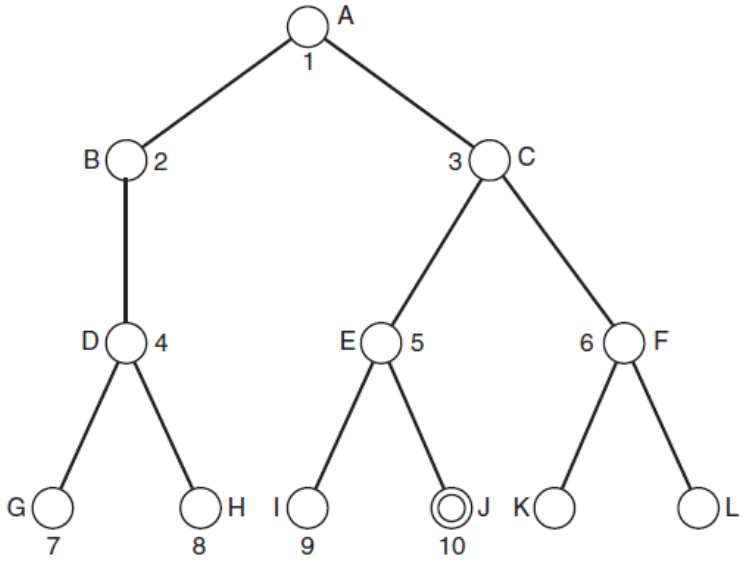
7.1 Breadth First Search:

A pseudocode implementation of breadth-first search is given below. BFS is a simple strategy in which the root node is expanded first, then all the successors of the root node are expanded next, then *their* successors, and so on. In general, all the nodes are expanded at a given depth in the search tree before any nodes at the next level are expanded. In Breadth-first search the *shallowest* unexpanded node is chosen for expansion. This is achieved very simply by using a FIFO queue for the frontier. Thus, new nodes (which are always deeper than their parents) go to the back of the queue, and old nodes, which are shallower than the new nodes, get expanded first. There is one slight tweak on the general graph-search algorithm, which is that the goal test is applied to each node when it is *generated* rather than when it is selected for expansion.

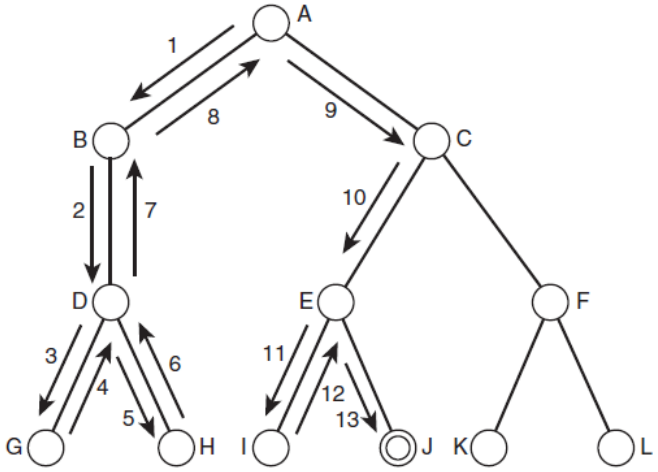



```
visited = [] # List for visited nodes.
queue = []     #Initialize a queue

def bfs(visited, graph, node): #function for BFS
  visited.append(node)
  queue.append(node)

  while queue:          # Creating loop to visit each node
    m = queue.pop(0)
    print (m, end = " ")

    for neighbour in graph[m]:
      if neighbour not in visited:
        visited.append(neighbour)
        queue.append(neighbour)
```

Depth-First Search:

Depth-first search always expands the *deepest* node in the current frontier of the search tree. The search proceeds immediately to the deepest level of the search tree, where the nodes have no successors. As those nodes are expanded, they are dropped from the frontier, so then the search

"backs up" to the next deepest node that still has unexplored successors. Depth-first search uses a LIFO queue. A LIFO queue means that the most recently generated node is chosen for expansion. This must be the deepest unexpanded node because it is one deeper than its parent—which, in turn, was the deepest unexpanded node when it was selected.




```
visited = []# Set to keep track of visited nodes of graph.

def dfs(visited, graph, node):  #function for dfs
    if node not in visited:
        print (node)
        visited.append(node)
        for neighbour in graph[node]:
            dfs(visited, graph, neighbour)
```

Uniform Cost Search:

Instead of expanding the shallowest node as in BFS, **uniform-cost search** expands the node n with the *lowest path cost* g(n). This is done by storing the frontier as a priority queue ordered by g. The algorithm is shown below. At each stage, the path that has the lowest cost so far is extended. In this way, the path that is generated is likely to be the path with the lowest overall cost.

```python
def search(graph, start, end):
    if start not in graph:
        raise TypeError(str(start) + ' not found in graph !')
        return
    if end not in graph:
        raise TypeError(str(end) + ' not found in graph !')
        return

    queue = Q.PriorityQueue()
    queue.put((0, [start]))

    while not queue.empty():
        node = queue.get()
        current = node[1][len(node[1]) - 1]

        if end in node[1]:
            print("Path found: " + str(node[1]) + ", Cost = " + str(node[0]))
            break

        cost = node[0]
        for neighbor in graph[current]:
            temp = node[1][:]
            temp.append(neighbor)
            queue.put((cost + graph[current][neighbor], temp))
```

Greedy Best First Search

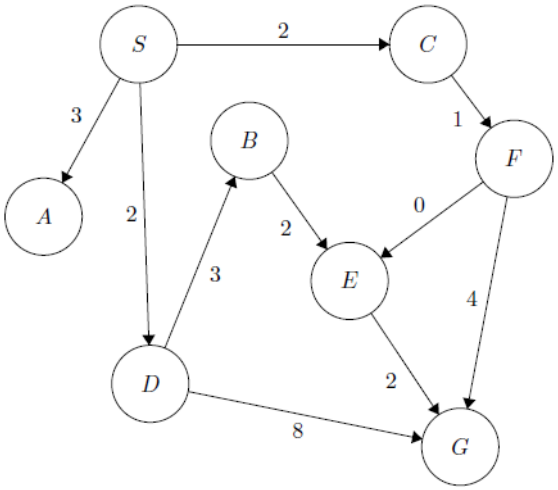
In Best-first search algorithm a node is selected for expansion based on an **evaluation function**, $f(n)$. The evaluation function is construed as a cost estimate, so the node with the *lowest* evaluation is expanded first. **Greedy best-first search** tries to expand the node that is closest to the goal, on the grounds that this is likely to lead to a solution quickly. Thus, it evaluates nodes by using just the heuristic function; that is, $f(n) = h(n)$.

```
Function best ()
{
     queue = [];     // initialize an empty queue
     state = root_node;   // initialize the start state
     while (true)
     {
          if is_goal (state)
               then return SUCCESS
          else
          {
               add_to_front_of_queue (successors (state));
               sort (queue);
          }
          if queue == []
               then report FAILURE;
          state = queue [0]; // state = first item in queue
          remove_first_item_from (queue);
     }
}
```

Lab Tasks

Exercise 7.1.

Using Breadth First Search (BFS) algorithm, Uniform cost search, DFS and greedy BFS, write out the order in which nodes are added to the explored set, with **start state S** and **goal state G**. Break ties in alphabetical order. Additionally, what is the path returned by each algorithm? What is the total cost of each path?



Graph02: