Bi directional Search

A graph is a network of nodes connected by arcs or edges.

The two basic graph search algorithms, Breadth-First Search and Depth-First Search aim to find a path between 2 nodes (preferably the shortest) and determine cycles in the graph.

A graph search is done in one direction, either from the source/initial vertex to the goal/target vertex.

To search from both ends simultaneously, a bidirectional search is implemented.

In [1]:

```
example_graph = {0:[1,2], 1:[0,3,4], 3:[1], 4:[1], 2:[0,5,6], 5:[2], 6:[2]}


def bi_directional_search(graph, start, goal):
    # Check if start and goal are equal.
    if start == goal:
        return [start]
    # Get dictionary of currently active vertices with their corresponding paths.
    active_vertices_path_dict = {start: [start], goal: [goal]}
    # Vertices we have already examined.
    inactive_vertices = set()

    while len(active_vertices_path_dict) > 0:

        # Make a copy of active vertices so we can modify the original dictionary as
        active_vertices = list(active_vertices_path_dict.keys())
        for vertex in active_vertices:
            # Get the path to where we are.
            current_path = active_vertices_path_dict[vertex]
            # Record whether we started at start or goal.
            origin = current_path[0]
            # Check for new neighbours.
            current_neighbours = set(graph[vertex]) - inactive_vertices
            # Check if our neighbours hit an active vertex
            if len(current_neighbours.intersection(active_vertices)) > 0:
                for meeting_vertex in current_neighbours.intersection(active_vertice
                    # Check the two paths didn't start at same place. If not, then w
                    if origin != active_vertices_path_dict[meeting_vertex][0]:
                        # Reverse one of the paths.
                        active_vertices_path_dict[meeting_vertex].reverse()
                        # return the combined results
                        return active_vertices_path_dict[vertex] + active_vertices_p

            # No hits, so check for new neighbours to extend our paths.
            if len(set(current_neighbours) - inactive_vertices - set(active_vertices
                # If none, then remove the current path and record the endpoint as i
                active_vertices_path_dict.pop(vertex, None)
                inactive_vertices.add(vertex)
            else:
                # Otherwise extend the paths, remove the previous one and update the
                for neighbour_vertex in current_neighbours - inactive_vertices - set
                    active_vertices_path_dict[neighbour_vertex] = current_path + [ne
                    active_vertices.append(neighbour_vertex)
                active_vertices_path_dict.pop(vertex, None)
                inactive_vertices.add(vertex)

    return None
```

In [3]:

```
bi_directional_search(example_graph, 0, 6)
```

Out[3]:

```
[6, 2, 0]
```

In [ ]:

```
```