R9 Graphs BFS

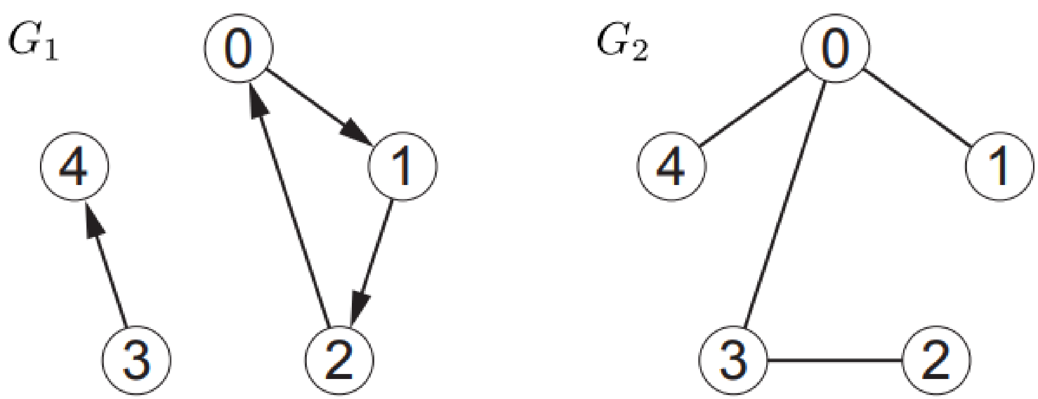
A graph G = (V, E) is a mathematical object comprising a set of vertices V (also called nodes) and a set of edges E.

- *directed* (u,v)
- *undirected* {u,v}
  The *in-degree* and *out-degree* of a vertex v denotes the number of incoming and outgoing edges connected to v respectively.
  When talk about degree, generally mean out-degree.
- *path*
  A path1 in a graph is a sequence of vertices (v0, . . . , vk) such that for every ordered pair of vertices (vi, vi+1), there exists an outgoing edge in the graph from vi to vi+1.
  The length of a path is the number of edges in the path

Graph Representations

adjacency list

The most common way is to store a Set data structure Adj mapping each vertex u to another data structure Adj(u) storing the adjacencies of v, the set of vertices that are accessible from v via a single outgoing edge.



example of *adjacency list* in python of G1 & G2. Using a direct access array for the top-level Set and an array for each adjacency list.

```
A1 = [[1],[2],[0],[4],[]]
A2 = [[1,3,4],[0],[3],[0,2],[0]]
```

example of hash table for outer Adj Set and inner adjacency lists Adj(u), using Python dictionaries

```
S1 = {0:{1},1:{2},2:{0},3:{4}},
S2 = {0:{1,3,4}, 1:{0}, 2:{3}, 3:{0,2}, 4:{0}}
```

Breadth-First Search

A breadth-first search (BFS) from s discovers the level sets of s: level Li is the set of vertices reachable from s via a shortest path of length i (not reachable via a path of shorter length).

- level L_i is the set of vertices reachable from s via a shortest path of length i.
- $L_0 = \{s\}$
- So to compute level Li+1, include every vertex with an incoming edge from a vertex in Li, t*hat has not already been assigned a level.*, means not repeated vertices.

```
def bfs(Adj, s):   # Adj: adjacency list, s: staring vertex
    parent = [Noen for v in Adj]  # O(V)  add a length|Adj| None array.
    parent[s] = s   # root is string vertex
    level = [[s]]   # level 0 is [s]
    while 0 < len(leve[-1]):    # while level has vertices
        level.append([])        # add a new level
        for u in level[-2]:     # loop over last full level
            for v in Adj[u]:    # loop over neighbors
                if parent[v] is None:    # only v is not be visited.
                    parent[v] = u
                    level[-1].append(v)
    return parent, level
```

- runtime
  so the runtime is O(|E| + |V|), see in note of LEC9