Graph Algorithms

Graph \( G = (V, E) \)

- \( V \) - vertices (nodes) \( |V| = n \)
- \( E \subseteq V \times V \) - edges \( |E| = m \)

Edges can be undirected (unordered pairs) or directed (ordered pairs).

Examples

- Undirected

  \[
  V = \{a, b, c, d, e\} \\
  E = \{(a, b), (b, c), (a, c), (c, a)\}
  \]

- Directed

  \[
  V = \{a, b, c\} \\
  E = \{(a, b), (b, c), (a, e)\}
  \]

Basic Notions

- \( u, v \in V \) are adjacent or neighbours if \( (u, v) \in E \)
- \( u \in V \) is incident to \( e \in E \) if \( u \stackrel{e}{\rightarrow} v \)
- \( \deg(v) \) = \# incident edges

- For directed graph, \( \text{indegree}(v), \text{outdegree}(v) \)

  \[
  \text{indeg}.2 \quad \text{outdeg}.3
  \]
- a path is a sequence of vertices 
  \( v_1, v_2, \ldots, v_k \) s.t. \((v_i, v_{i+1}) \in E \) \( i = 1, \ldots, k-1 \)
- a simple path does not repeat vertices.

- a cycle is a path that starts and ends at the same vertex.

- a tree is a connected graph without cycles.

- a graph is connected if every \( u, v \in V \) are joined by a path.

- connected component of a graph
  = maximal connected subgraph


History: Euler, Königsberg bridge problem 1735

Applications - many:
- networks: wireless, transportation, social
- web pages, game configurations, etc
Storing Graphs

- Adjacency matrix: \( A[i,j] = \begin{cases} 1 & \text{if } (i,j) \in E \\ 0 & \text{otherwise} \end{cases} \)

  \( O(n^2) \) space, even if the graph is sparse, \( |E| \ll n^2 \)

  But a query “is \((i,j)\) an edge?” can be answered in \( O(1) \)

- Adjacency lists
  
  For each vertex \(v\), store linked list of \(v\)'s neighbors

  \[
  \begin{array}{ccc}
  \text{a} & \text{b} & \text{c} \\
  \text{b} & \text{c} \\
  \text{c} & \text{a} \\
  \end{array}
  \]

  \( O(n + m) \) space

  A query “is \((i,j)\) an edge?” requires traversing \(i\)'s adjacency list, \( O(n) \) worst case

Some times graphs are stored implicitly, e.g., nodes may represent configurations in a chess game. Generate nodes as you search configuration space.

Can use hash table of adjacency lists to get space \(O(n + m)\) and \(O(1)\) test for edge.
Exploring Graphs — visit all nodes, or all nodes reachable from some "source"

Further — find shortest paths, connected components.

Breadth First / Depth First Search

Cautious search: check everything one edge away, then two...

Order in which vertices are discovered

1, 2, 3, 6, 8, 4, 5, 7

1's neighbours 2's 6's

BFS tree

Use a queue to store vertices that have been discovered but must still be explored

Vertices are marked:

undiscovered → discovered
Explore \((v)\)
- for each neighbour \(u\) of \(v\)
  - if mark\((u)\) = undiscovered
    - mark\((u)\) <= discovered
  - parent\((u)\) <= \(v\)
  - level\((u)\) <= level\((v)\)+1
  - add \(u\) to Queue

BFS

initialize: mark all vertices undiscovered
pick initial vertex \(v_0\) \(\triangleright\) parent\((v_0)\) <= Ø level\((v_0)\) = 0
add \(v_0\) to Queue; mark\((v_0)\) <= discovered
while Queue not empty
  - \(v\) <= remove from Queue
  - Explore\((v)\)

Also useful to store `parent` and `level` (see previous example) See blue additions above.

BFS takes \(O(n + m)\) time. We explore each vertex once and check all incident edges.

Time is \(O(n + \sum_{v} \deg(v)) = O(n + m)\).

Note: \(\sum_{v} \deg(v) = 2m\)

because we count each edge twice.
Properties of BFS

- The parent pointers create a directed tree (because each addition adds a new vertex $u$ with parent $v$ in the tree).

- $u$ is connected to $v_0$ iff BFS from $v_0$ reaches $u$. Stronger:

  **Lemma.** The shortest path from $v_0$ to $u$ has length (# edges) $k$ iff BFS from $v_0$ puts $u$ in level $k$.

  **Proof.** By induction with basis $k=0$:  

  $\Leftarrow$ Suppose $u$ in level $k$. Then $parent(u) = v$ is in level $k-1$. So shortest path $v_0$ to $v$ has length $k-1$ by induction. There is a path $v_0$ to $u$ of length $k$. Is it shortest? Yes, otherwise (by induction) $u$ would be in a level $< k$.

  $\Rightarrow$ Suppose shortest path is $v_0, v_1, \ldots, v_k = u$ then $v_0, v_{k-1}$ is a shortest path of length $k-1$. So $v_{k-1}$ goes in level $k$. Then $u$ (a neighbor of $v_{k-1}$) goes in level $\leq k$. Could $u$ go in level $< k$? No, otherwise (by ind.) there would be a shorter path to $u$. 

Consequences:

1. BFS from \( v_0 \) finds the connected component of \( v_0 \).

   EX: Enhance BFS to find all connected components in time \( O(n+m) \).

2. BFS finds shortest paths (# edges) from \( v_0 \).

   EX: Use BFS to find if a connected graph has a cycle.

   EX: Prove that if \((u,v) \in E\) then \( \text{level}(u), \text{level}(v) \) differ by 0 or 1.

---

BFS to test bipartiteness

\( G \) is bipartite if \( V \) can be partitioned into \( V_1, V_2 \) \((V_1 \cap V_2 = \emptyset)\) s.t. every edge has one end in \( V_1 \) and one end in \( V_2 \).

Note that a bipartite graph cannot have an odd cycle.
Run BFS. \( V_i = \text{odd levels} \quad V_2 = \text{even levels} \).

Test if this works (check edges)
- if YES \( \rightarrow G \) is bipartite
- if NO then there is an edge \((u, v)\) with
  
  \( u, v \) both in \( V_i \) \( (i = 1 \text{ or } 2) \)

By Ex. level \((u)\) and level \((v)\) differ by \( \pm 1 \).
If 1, then one in \( V_i \), one in \( V_2 \).
So \( u, v \) are in same level, say \( k \).

Let \( z = \text{least common ancestor of } u, v \).

Cycle formed by path \((u, z)\) path \((z, v)\) \((u, v)\)
has length \( 2t + 1 \) = odd

Then \( G \) is not bipartite.

This proves:

**Lemma**: \( G \) is bipartite iff it has no odd cycle.

The proof is via an algorithm that finds a bipartition or an odd cycle.