1. Chapter 3 graph

Undirected graph. G = (V, E)

1. parameters

   V = set of nodes

   E = set of edges between pairs of nodes.

   Graph size parameters: n = $|V|$, m = $|E|$.

2. graph representation:

	Adjeceny matrix	Adjeceny list
description	A[u][v] = 1 if (u, v) is an edge	all the nodes can reach u linked in L[u]
Space	O(n^2)	O(m + n)
checking (u, v)	O(1)	O(degree(u))
Identifying all edges	O(n^2)	O(n + m)

3. paths and connectivity

   1. $path$

      A $path$ in an undirected graph $G = (V, E)$ is a sequence P of nodes $v_1, v_2, \ldots, v_{k-1}, v_k$ with the property that each consecutive pair v_i, v_{i+1} is joined by an edge in E.

   2. $simple\ path$

      A path is $simple$ if all nodes are **distinct**.

   3. $connected\ undirected\ graph$

      An undirected graph is $connected$ if for every pair of nodes u and v, there is a path between u and v.

   4. $cycles$

      A cycle is a path $v_1, v_2, \ldots, v_{k-1}, v_k$ in which $v_1 = v_k,\ k > 2$, and the first $k-1$ nodes are all distinct.

   5. $trees$

      An undirected graph is a **tree** if it is **connected** and does not contain a **cycle**.

   6. $rooted\ trees$

      Given a tree T, choose a root node r and orient each edge away from r. r si the root.

      $Theorem:$

Let G be an undirected graph on n nodes. Any **two** of the following statements imply the **third**.

G is **connected**.

G does not contain a **cycle**.

G has $n-1$ edges

Breadth First Search

1. $BFS\ algorithm$

   $L_0 = \{s\}$

   $L_{i+1} = all\ nodes\ that\ do\ not\ belong\ to\ an\ earlierlayer,$
   $and\ that\ have\ an\ edge\ to\ a\ node\ in\ L_i$

2. $property$

   Let T be a $BFS\ tree$ of $G=(V,E)$, and let (x,y) be an edge of G. Then the level of x and y differ by at most 1

3. $Theorem\ 1:$

   For each i, L_i consists of all nodes at distance exactly i from s.

   There is a path from s to t if t appears in some layer.

   $Theorem\ 2:$

   The above implementation of BFS runs in $O(m+n)$ time if the graph is given by its adjacency representation.

   $proof\ of\ Theorem\ 2:$

   runs in $O(m+n)$ time:

   – when we consider node u, there are $deg(u)$ incident edges (u,v)

   – total time processing edges is $\sum_{u \in V} deg(u) = 2m$

4. Connect component: find all the reachable from s

   solution:

   BFS from s = explore in order of distance from s

   DFS from s = explore in a different way

Testing Bipartiteness

1. Def:

   An undirected graph $G=(V,E)$ is **bipartite** if the nodes can be colored red or blue such that every edge has one **red** and one **blue** end.

2. $Why\ testing\ bipartiteness$

   Many graph problems become:

   - easier if the underlying graph is bipartite (matching)
   - tractable(poly-time) if the underlying graph is bipartite (independent set)

Before attempting to design an algorithm, we need to understand structure of bipartite graphs.

3. $Lemma\ 1$

   If a graph G is bipartite, it cannot contain an odd length circle.

   $Proof\ 1$:

   Not possible to 2-color the odd cycle, let alone G

   $Lemma\ 2$. Let G be a connected graph, and let $L_0, \ldots, L_k$ be the layers produced by BFS starting at node s. Exactly one of the following holds.

   (i) No edge of G joins two nodes of the same layer, then G is **bipartite**.

   (ii) An edge of G joins two nodes of the same layer, then G contains an odd-length cycle, hence is not **bipartite**.

   $Proof\ of\ 2(i)$

   - Suppose no edge joins two nodes on same layer.
   - By previous property on page 19, this implies every **edge** join two nodes in **adjacent** layers(beacause level differ by at most **1**).
   - Color nodes on odd levels with **red**, nodes on even levels with **blue** -> Bipartition.

   $Proof\ of\ 2(ii)$

   - Suppose (x, y) is an edge with x, y in same level L_j.
   - Let $z = lca(x, y) = lowest\ common\ ancestor$.
   - Let L_i be level containing z.
   - Consider cycle that takes edge from x to y, then path from y to z, then path from z to x.
   - Its length is $1 + (j - i) + (j - i)$, which is odd.

   $Corollary$

   A graph G is bipartite **if and only if** it contain no odd length cycle

Connectivity in Directed Graphs

1. Def

   Directed graph: $G = (V, E)$

   - Edge (u, v) goes from node u to node v

2. $Mutually\ reachable$

   - Node u and v are **mutually reachable** if there is a path from u to v and also a path from v to u.

3. $Strong\ Connectivity$

   - A graph is strongly connected if **every pair of nodes** is **mutually reachable**.

   $Lemma$

   - G is strongly connected **if and only if** $\exists s\ in\ G\ s.t.$ every node is reachable from s, and s is reachable from every node

$Proof$

- "if": $\forall u,\ v\ in\ G$

  Path from u to v: $u - s$ then $s - v$.

  Path from v to u: $v - s$ then $s - u$.

- "only if"

  Follows from definition.

$Theorem$: Can determine if G is strongly connected in $O(m+n)$

$Proof$

- Pick any node s.
- Run BFS from s in G, we got set of reachable nodes S_1.
- Run BFS from s in G^{rev}, we got set of reachable nodes S_2. ($s \Rightarrow v\ in\ G^{rev}$ become $v \Rightarrow\ s\ in\ G$)
- Return true **if and only if** all nodes reached in both BFS executions, that is, $S_1 = S_2 = |V|\ of\ G$
- Find the most strong connective subgraph of G: it's the subgraph construct by the nodes set: $S_1 \cap S_2$
- Correctness follows immediately from previous $lemma$.

DAGs and Topological Ordering

1. $Def\ of\ DAG(Directed\ Acyclic\ Graph)$

   - An DAG is a directed graph that contains no **directed** cycle

   $Def\ of\ Topological\ Order$

   - A topological order of a directed graph $G = (V, E)$ **is an ordering** of its nodes as $[v_1, \ldots, v_n]$ such that for every **directed** edge (v_i, v_j) we have $i < j$

2. $Precedence\ constraints$

   Edge (v_i, v_j) means task v_i must occur before v_j

3. $Lemma1: topological\ order \Rightarrow DAG$

   If G has a topological order, then G is a DAG

   $Proof(by\ contradiction)$

   - Suppose that G has a topological order $v_1, \ldots, v_n$ and that G also has a directed cycle C. Let's see what happens.
   - Let v_i be the **lowest-indexed node** in C, and let v_j be the node just **before** v_i; thus (v_j, v_i) is an edge.
   - By our choice of i, we have $i < j$.
   - On the other hand, since (v_j, v_i) is an edge and $v_1, \ldots, v_n$ is a topological order, we must have $j < i$, a contradiction.

   $Lemma2: DAG \Rightarrow a\ node\ with\ 0\ in-degree$

   If G is a DAG, then G has **a node** with no **incoming edges**.

$Proof$

- Suppose that G is a DAG and every node has at least one **incoming edge**. Let's see what happens.
- Pick any node v, and begin following edges backward from v. Since v has at least one incoming edge (u, v) we can walk backward to u.
- Then, since u has at least one incoming edge (x, u), we can walk backward to x.
- Repeat until we visit a node, say w, twice.
- Let C denote the sequence of nodes encountered between successive visits to w. C is a cycle.

$Lemma 3 : DAG \Rightarrow topological\ order$

If G is a DAG, then G has a topological order.

$Proof(by\ induction)$

- Base case: true if $n = 1$.
- Given DAG on $n > 1$ nodes, find a node v with no incoming edges.
- $G - \{v\}$ is a DAG, since **deleting v cannot create cycles**.
- By inductive hypothesis, $G - \{v\}$ has a topological ordering.
- Place v first in topological ordering; then append nodes of $G - \{v\}$ in topological order. This is valid since v has no incoming edges.

4. $Running\ time$

$Theorem$

- Algorithm finds a topological order in $O(m + n)$ time.

$Proof$

- Initialization:$O(m + n)$
- Maintain a field, $count$, denoting the count of incoming edges of a node(Can be recorded during the initialization)
- Find all nodes with no incoming edge and put them in a queue q and a list $List$: $O(n)$
- continue to dequeue the q and get node $n1$, find the delete the outer edge of $n1$, and update the $count$ fields of nodes that $n1$ can directly go to, filter the nodes that become $0 - in - degree$, put them in the q and $List$: $O(m + n)$