ICCS313 - Algorithm Analysis, T1 2019/20

Lecture 4: Graph Algotirhms (1)

5 October 2019

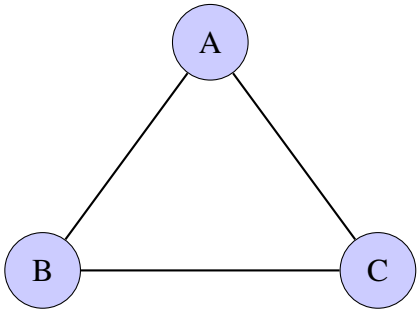
Lecturer: Chaya Hiruncharoenvate *Scribe: -*

- Assignment 2 extended
- Attempt the programming question by Monday

1 Review: Graph

Graph is a very useful representation. We use it model many things in a wide range of applications such as communication, transportation, electronic circuit, social relationship, etc. Basically anything with network-like relationship.

A *graph* (usually denoted by G) consists of two objects: a finite, non-empty set V of vertices, and a set E of edges. The set V can be anything, but the set E must consist of unordered pairs of distinct elements of V. Think of V as a set of points (called vertices), and think of E as specifying which pairs of vertices are joined. For example, suppose G is the following graph.



Here $G = (V, E)$ where $V = \{A, B, C\}$ and $E = \{(A, B), (A, C), (B, C)\}$. Typically, when we talk about the size of graph problems, this concerns the number of vertices $|V|$, often denoted by n and the number of edges $|E|$ denoted by m. Here we have $n = |V| = 3$ and $m = |E| = 3$.

There are many kinds of graphs. In the example you saw, this is a simple undirected graph. Sometimes, edges have direction. If that is the case, the graph is *directed*.

1.1 Graph Representation

There are two popular representations of graphs.

1. Adjacency matrix – an nxn matrix where $A_{u,v} = 1$ if (u, v) is an edge. The space requirement is $\Theta(n^2)$. Checking if (u, v) is an edge is $\Theta(1)$ but listing all edges takes $\Theta(n^2)$.

2. Adjacency list – array of linked-list. The space requirement is $\Theta(n + m)$. Checking if (u, v) is an edge is $O(degree(u))$ while listing all edges takes $\Theta(n + m)$.

1.2 Path and Connectivity

- A **path** in an undirected graph $G = (V, E)$ is a sequence of nodes: $v_1, v_2, \ldots, v_k$ with the property that each consecutive pair (v_{i1}, v_i) is joined by an edge in E.
- A path is **simple** if all nodes are distinct.
- An undirected graph is **connected** if for every pair of nodes u and v, there is a path between u and v.
- A **cycle** is a path $v_1, v_2, \ldots, v_k$ in which $v_1 = v_k$, $k > 2$, and the first $k - 1$ nodes are all distinct.

1.3 Trees

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

Theorem 1.1. *Let G be an undirected graph on n nodes. Any two of the following statements imply the third:*

1) G is connected.

2) G does not contain a cycle.

3) G has $n - 1$ edges.

Proof:

$(1 + 2 \Rightarrow 3)$ Assume G is connected and doesn't contain a cycle. We are going to show this by induction. Assume (3) is true for any graph $< n$ nodes. Consider G with n nodes. Pick any edge e in G. Removing e breaks G into two components: G_1 and G_2. with n_1 and n_2 nodes respectively. Note that $n_1 + n_2 = n$. We know that G_1 and G_2 are connected and don't contain any cycle. By IH, we know that G_1 has $n_1 - 1$ edges and G_2 has $n_2 - 1$ edges. So G has $(n_1 - 1) + (n_2 - 1) + 1 = n - 1$ edges.

$(2 + 3 \Rightarrow 1)$ Assume G has $n - 1$ edges and G does not have a cycle. Again, we can show this by induction. Assume (1) is true for any graph $< n$ nodes. Consider acyclic graph G with $n-1$ edges. Pick a node u in G if you keep follow an edge going out of u, you will eventually reach a node whose degree is 1. Call this node v. Consider G' where we remove v. By IH, we know that G' must be connected. Thus, $G = G' + v$ should be connected as well.

$(1 + 3 \Rightarrow 2)$ Assume G has $n - 1$ edges and G is connected. Suppose G has a cycle. Let (u, v) be an edge in the cycle. Consider $G' = G - (u, v)$. G' is still connected with n nodes, but G' has $n - 2$ edges. This is impossible because a graph needs to have at least $n - 1$ edges to be connected.

■

2 Connectivity Problems

2.1 s-t Connectivity Problem

Given a graph G and two nodes s and t, is there a path between s and t?

How do you solve this problem? You could perform BFS.

Breadth-First search (BFS) algorithm explore nodes outwards from s in all possible directions by adding nodes one "layers" at a time.

```
L0 = {s}
L1 = all neighbors of L0
L2 = all nodes not in L0 and L1, and that have an edge
     to a node in L1.
...
```

Theorem 2.1. *For each i, L_i consists of all nodes at distance exactly i from s. There is a path from s to t $\iff$ t appears in some layer.*

```
hasPath(G,s,t):
    Run BFS on G from s
    return True if t appears any layer or False otherwise
```

If G is represented by an adjacency list, what is the running time of BFS? You could say that it's $O(n^2)$. Well, that's true because in each step, you loop over nodes ($\leq n$), and you repeat this at most n times.

To be more precise, you could argue that, in each step, when you inspect node u, you loop over only $degree(u)$ nodes. So, the total time should be $\sum_{u \in V} degree(u) = 2m$. Thus, the total running time is $O(2m) + O(n) = O(m + n)$

2.2 Connected Component

Given G and a node s, find all nodes connected to s.

This problem has a direct application called *flood fill* or the color bucket in Paint.

Naive Solution: $O(n(m + n))$

```
Let R = {}
for u in V:
  if hasPath(s,u):
    R = R + {u}
return R
```

Better: $O(m + n)$

```
R = {s}
while (there is an edge (u,v) where u in R and v not in R) {
    R = R + {v}
}
return R
```

3 Checking Bipartiteness

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

[Draw a graph that is bipartite and another that is not]

Here we want to determine whether a given undirected graph G is bipartite.

Lemma 3.1. *If G is bipartite, it cannot contain an odd-length cycle.*

Proof: Not possible to 2-color the odd-length cycle, let alone G. ■

Lemma 3.2. *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, and G is bipartite.

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

Proof:

- For (i), suppose no edge joins two nodes in same layer. By BFS property, each edge joins two nodes in adjacent levels. So we could have a bipartition: white = nodes on odd levels, blue = nodes on even levels
- For (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 the level containing z. Consider the cycle contain the edge (x, y), path from x to z and path from y to z. The length of this cycle is $1 + (j - i) + (j - i)$, which is odd.

■

Corollary 3.3. *A graph G is bipartite iff it contains no odd-length cycle.*