Adam Gluck CSE 2321 Final

Graph Data Structures

Adjacency Matric

- Space: $\Theta(|V|^2)$
- Lookup for edge takes constant time
- Useful Matrix Operations

Adjacency List

- Space: $\Theta(|V| + |E|)$
- Fast lookup for neighbors of vertex

Vocabulary

Directed Graph

- Tree Edge: (u,v) is a tree edge if $v.\pi = u$
- Back Edge: (u,v) is a back edge if v is an ancestor of u
  - Needed for graph to have cycles
- Forward Edge: (u,v) is a forward edge if u is an ancestor of and (u, v) is not a tree edge
- Cross Edge: All other edges are cross Edges

Undirected Graph

- Tree Edge: (u,v) is a tree edge if $v.\pi = u$ or $u.\pi = v$
- Back Edge: (u,v) is a back edge if v is an ancestor of u or if u is an ancestor of v

Breadth First Search

Uses

1. Distance between two vertices.
2. Distance between a source and all vertices.
3. Distance between all pairs of vertices.
4. Determine if a graph is bipartite.
5. Determine the number of connected components
6. $\Theta(|V| + |E|)$ traversal

Pseudocode

```
BFS(G,root):
    frontier = new Queue();
    root.distance = 0
    frontier.push(root);
    for (w != root) w.distance = ∞;

    while (frontier not empty):
        v = frontier.dequeue();
        for (w successor of v):
            if (w.distance == ∞):
                frontier.enqueue(w);
                w.distance = v.distance + 1;
```

Depth First Search

Uses

1. Detecting cycle in a graph
2. Path Finding
3. Topological Sorting
4. Determine if a graph is bipartite.
5. Finding Strongly Connected Components of a graph
6. $\Theta(|V| + |E|)$ traversal

Pseudocode

```
DFS(G):
    for each u ∈ V
        u.color = white
        u.π = nil
    time = 0
    for each u ∈ V
        if u.color = white
            Visit(G, u)
// Recursive
Visit(G, u):
    time = time + 1
    u.d = time
    u.color = gray
    for each v ∈ G.Adj[u]
        if v.color = white
            v.π = u
            Visit(G, v)
    u.color = black
    time = time + 1
    u.f = time

// Stack
Visit(G, u)
    let s be a stack
    s.push(u)
    while s is not empty
        w = s.pop()
        if w.color = white
            time = time + 1
            w.d = time
            w.color = gray
            s.push(w)
            for each v ∈ G.Adj[w]
                if v.color = white
                    v.π = w
                    s.push(v)
        if w.color = gray
            w.color = black
            time = time + 1
            w.f = time
```

Parameters

- v.π: Parent of v, used to recover order DFS visited vertices
- v.d: Discovery time of v, time when vertex is first found by algorithm
- v.f: Finishing time of v, time when vertex is processed by algorithm
- v.color
  - White: node is undiscovered
  - Gray: node is discovered and being processed

- Black: node has been visited and is finished processing

Topological Sort

- Requires directed graph is directed and has no nontrivial cycles (Acrylic graph)
- Returns an ordered list of nodes where no edges point backwards
- Runtime is $\Theta(|V| + |E|)$

Algorithm

1. Call DFS(G) to compute finishing times for each vertex
   - Check for cycles can be added by checking if w is gray in Visit function
2. Return list of nodes sorted by finishing time in reverse order

Strongly Connected Components

- A directed graph is strongly connected if for any $x, y \in V$ there is a path from x to y and a path from y to x
- Runs in $\Theta(|V| + |E|)$

Algorithm

1. Pick a vertex v ∈ V and use DFS to check if every vertex can be reached from v. If not, return false.
2. Compute G^T, the transpose graph of G.
3. Do DFS in G^T to check if every vertex can be reached from v. If not, return false. Otherwise, return true

Planar Graphs

- A graph is planar if it has a planar embedding
- A planar embedding is a way to draw a graph so no edges cross
- Planar graphs divide the plane they are drawn on into regions called faces

Euler's polyhedral formula

1. let v = |V|, e = |E|, and f = the number of faces
2. v – e + f = 2 for all connected planar graphs

Two Color Algorithm

- Colors graph nodes with one of two colors and returns whether the graph is bipartite
- Runs is $\Theta(|V| + |E|)$

```
Bipartite(G)
    bipartite = true
    for each v in V
        v.color = None
    for each v in V
        if v.color = None
            bipartite = bipartite and
two_color(G,v)
    return bipartite

two_color(G,s)
    s.color = 0
    q.enqueue(s)
    while q.length != 0
        v = q.dequeue()
        for each w s.t. {v,w} in E
            if w.color = v.color
                return false
            else if w.color = None
                w.color = (v.color + 1) mod 2
                q.enqueue(w)
    return true
```

Hamiltonian Cycle Check

1. Iterate over all n! possible orderings of vertices. For each ordering check to see if it is a Hamiltonian cycle by checking if the vertices are connected by an edge

2. Modify DFS to explore all possible paths. For example, instead of coloring vertices black they are colored white so that vertices can be "re-explored" after DFS backtracks

Universal Sink Check

- Start in the top left of the matrix. If it contains a 0, move to the right, if it contains a 1, move down. Continue this process until there are no more rows or columns to move to. The row and column you end in need to be checked to see if the corresponding vertex is a universal sink. This process works because if A[i][j] = 1, then i cannot be a universal sink because it has an outgoing edge. If A[i][j] = 0, then j cannot be a universal sink because there is no edge from i to j
- Runs in $\Theta(|V|)$

Compute diameter of graph

- Do BFS from each vertex, this gives you the shortest distance for all pairs. Then find the max distance over all pairs.
- Runs in $\Theta(|V|^2 + |V||E|)$

Transpose Directed Graph

- It is enough to iterate through each row/list of the data struc- ture and add the revered edges to a new data structure. This would result in time O(|V |2) for the adjacency matrix and time O(|V | + |E|) for the adjacency list