Graph Algorithms

Graphs are important data structures. Graphs can express arbitrary relationships between objects.

Simple Graphs

- Vertices
- Edges (—, —→)

Labelled Graphs

- Vertices and vertex labels
- Edges and edge labels

For now, simple graphs.
A graph $G$ consists of a set of vertices $V$ and a set of edges $E$ such that $(u,v) \in E \rightarrow u, v \in V$ and $u$ is connected to $v$ with an edge.
Representation

1. Adjacency lists

2. Adjacency matrix

   1. Adjacency list
      Array Adj of |V| lists, O(E)
      Adj[u] is a pointer to a list of vertices v such that (u,v) \in Edges
      Memory ________, Lookup ____________

   

Directed edge: u \rightarrow v
Undirected edge: u \rightarrow v
2. Adjacency matrix

Matrix $A$ $|V| \times |V|$, where

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

Memory $\Theta(V^2)$, Lookup $O(1)$

Both allow added satellite data easily

Adjacency list better for sparse graphs

---

Traversing Graphs

Search for paths satisfying various constraints (e.g., shortest path)

- Visit some set of vertices (e.g., tours)
- Search for subgraphs (e.g., graph matching (isomorphisms))

Techniques:

1. Breadth-First Search (BFS)
2. Depth-First Search (DFS)
Breadth-First Search (BFS)

Breadth-first search produces breadth-first tree. Path from s to x in BF tree is the shortest path in terms of the number of edges.

**BFS**: Given graph \( G = (V,E) \) and source s

- Visit every vertex reachable from s in one edge that has not already been visited
- Visit every vertex reachable from s in two edges that has not already been visited
- ...

**BFS Data Structures**

A node in a BF tree represents a vertex
Use a queue to remember frontier of search

Note: Cormen et al.’s BFS algorithm uses color instead of visited.

- Unvisited vertex: white
- Discovered vertex: gray
- Visited vertex: black

---

**Pseudocode**

BFS(G, s)

1. foreach v in (V - {s}) ; initialize
2. visited(v) = False
3. pred(v) = NIL
4. distance(v) = ∞
5. visited(s) = True ; visit start vertex
6. distance(s) = 0
7. pred(s) = NIL
8. Enqueue(Q, s)
9. while not QueueEmpty(Q)
10. u = DeQueue(Q)
11. foreach v in Adj[u]
12 if not visited(v) 
13 then visited(v) = True 
14 distance(v) = distance(u) + 1 
15 pred(v) = u 
16 Enqueue(Q, v) 

Examples 

Analysis 

_____ Enqueue / Dequeue operations 
    each vertex is processed only once 
_____ total time scanning adjacency list 
_____ 
_______ BFS running time worst case 

Properties 

δ(s, v) = shortest-path distance from s to v 

Theorem 23.4 
G = (V, E) directed or undirected 
BFS(G, s), s in V 
Upon termination of BFS, every vertex v in V reachable from s has distance(v) = δ(s, v) 
For vertex v ≠ s reachable from s, one shortest path from s to v is the shortest path from s to pred(v) followed by edge (pred(v), v)
Proof: by induction on distance from s

Print-Path(G, s, v) ; O(V)
  if v = s
  then print s
  else if pred(v) = NIL
    then "no path"
    else Print-Path(G, s, pred(v))
      print v

Predecessor Subgraph

\( G_{pred} = (V_{pred}, E_{pred}) \) is a **predecessor subgraph** of G if
\[
V_{pred} = \{ v \in V \mid \text{pred}(v) \neq \text{NIL} \} \cup \{s\}
\]
\[
E_{pred} = \{ (\text{pred}(v), v) \in E \mid v \in V_{pred} - \{s\} \}
\]

**Lemma 23.5** The **pred** tree generated by BFS results in a predecessor subgraph \( G_{pred} \) which defines a BF tree.
Depth-First Search

Vertex in DFF of T

<table>
<thead>
<tr>
<th>pred</th>
<th>visited</th>
<th>discover</th>
<th>finish</th>
</tr>
</thead>
</table>

Note again, Cormen et al. use color instead of visited. 

Discover is the time when vertex first visited. 

Finish is the time when all vertices reachable from this vertex have been visited.

DFS

1. Given G
2. Pick an unvisited vertex v, remember the rest
3. Recurse on vertices adjacent to v

DFS(G)
   foreach v in V
visited(v) = False
pred(v) = NIL
time = 0
foreach u in V
    if not visited(u)
        then Visit(u)

Visit(u)
    visited(u) = True
time = time + 1
discover(u) = time
foreach v in Adj[u] ; $\Theta(E)$
    if not visited(v)
        then pred(v) = u
        Visit(v)
time = time + 1
finished(u) = time

The total running time for DFS is $\Theta(V + E)$
Topological Sort

**Application:** Scheduling

Let u represent event 1, and let v represent event 2

u must occur before v occurs

**Problem:** Find a schedule for executing the events that preserves the “before” relation.

**Solution:** Represent events as vertices and before(u,v) as a directed edge (u,v).

Let G = (V, E) be a directed, acyclic graph (DAG). If (u,v) ∈ E, then u appears before v in the ordering of events in the schedule.
The vertices and edges in G are referred to as the topology of G.
We want to sort this topology based on some key such that \( u \to v \) implies \( \text{key}(u) < \text{key}(v) \) (or \( \text{key}(v) < \text{key}(u) \) reverse sorted).
The finish times assigned by DFS satisfy this constraint.

---

**Pseudocode**

TopologicalSort(G) \( ; \Theta(V + E) \)
- DFS(G), as each vertex finishes, insert it on the front of the linked list
- return linked list of vertices

---

**Example: Professor Bumstead gets dressed**

![Diagram showing the order of dressing]
DFS considers vertices in alphabetical order by label.

Note that a different order for DFS yields a different schedule.

**Strongly Connected Components**

Many graph applications look for a minimal way to connect each vertex to every other vertex.

Examples: bridging gaps, identifying bottlenecks

A graph $G = (V, E)$ is ________________ if for every pair of vertices $<u,v>$, $u,v \in V$, there is a path ($\sim$) from $u$ to $v$ ($u \sim v$) and from $v$ to $u$ ($v \sim u$).

A __________________________ (SCC) of a graph $G = (V, E)$ is a maximal set $U \subseteq V$ such that for every pair $<u,v> \in U$, $u \sim v$ and $v \sim u$.

**Define:** The ______________ of graph $G = (V, E)$ is the graph $G^T = (V, E^T)$, where $E^T = \{(u,v) \mid (v,u) \in E\}$.

Time to create $G^T = O(V+E)$
Pseudocode

SCC(G)
DFS(G) to compute finishing times
compute $G^T$
DFS($G^T$) considering vertices in main loop in
decreasing order by finish time
output each tree in DFF of $T$ as a SCC

Example

DFS(G)

DFS($G^T$)

SCCs: $\{c, d\}$
$\{s, b, a\}$
Applications