Graph Algorithms

Dr. John McDonough

Spoken Language Systems
Saarland University

May 7, 2009
Introduction

- This course presents the theory of applying *weighted finite-state transducers* (WFSTs) to several tasks in speech and natural language processing.
- In particular, WFSTs can be applied to the tasks of:
  - speech recognition,
  - parsing of natural languages,
  - morphology,
  - approximation of context free grammars,
  - part-of-speech tagging,
  - machine translation.
- WFSTs are attractive for such applications because of their *efficiency* and *decidability*. 
The computational expense of deciding whether or not a string is accepted by a WFST is linear in the size of the string, \textit{not} in the size of the transducer.

As WFSTs are in essence \textit{directed graphs}, in this lecture we consider algorithms that operate on graphs, including:

- Breadth first search,
- Depth first search,
- Topological sorting.

All of these algorithms will prove useful in applying WFSTs to speech and natural language processing.

Proof by Induction

- In computer science, the most common method of proving a theorem is the *proof by induction*.
- Assume, for example, that we wish to prove

\[ \sum_{n=0}^{N} n^2 = \frac{N(N+1)(2N+1)}{6}. \]  

- The proof by induction proceeds in two steps:
  - Prove that (1) holds for \( N = 0 \) (i.e., the *basis*).
  - Assume that (1) holds for \( N - 1 \), then prove it holds for \( N \) (i.e., the *inductive hypothesis*).

- Clearly (1) holds for \( N = 0 \), so we have established the basis.
In order to prove the inductive hypothesis, note that (1) implies

\[ \sum_{n=0}^{N-1} n^2 = \frac{(N - 1)N(2N - 1)}{6}. \]

This, in turn, implies

\[ \sum_{n=0}^{N} n^2 = \frac{(N - 1)N(2N - 1)}{6} + N^2 = \frac{(N - 1)N(2N - 1) + 6N^2}{6} \]

\[ = \frac{2N^3 - 3N^2 + N + 6N^2}{6} = \frac{2N^3 + 3N^2 + N}{6} \]

\[ = \frac{N(N + 1)(2N + 1)}{6}. \]

With the last equation, we have proven (1).
Sets

- A set is a collection of distinguishable objects known as members or elements.
- That $x$ is a member of the set $S$ is denoted as $x \in S$ and read as “$x$ is in $S$.”
- Two sets $A$ and $B$ are equal, which is denoted as $A = B$, if they contain the same elements. For example, 
  \[ \{1, 2, 3, 1\} = \{1, 3, 2\} = \{3, 2, 1\} \].
- Frequently encountered sets have special notations:
  - $\emptyset$ denotes the empty set.
  - $\mathbb{Z}$ denotes the set of integers, \{\ldots, −2, −1, 0, 1, 2, \ldots\}.
  - $\mathbb{R}$ denotes the set of real numbers.
  - $\mathbb{N}$ denotes the set of natural numbers, \{0, 1, 2, \ldots\}. 
Set Operations

- The *intersection* of sets $A$ and $B$ is the set
  \[ A \cap B = \{ x : x \in A \text{ and } x \in B \}. \]

- The *union* of sets $A$ and $B$ is the set
  \[ A \cup B = \{ x : x \in A \text{ or } x \in B \}. \]

- The *difference* between two sets $A$ and $B$ is the set
  \[ A - B = \{ x : x \in A \text{ and } x \notin B \}. \]
If $x \in A$ implies $x \in B$, then we say $A$ is a subset of $B$ and write $A \subseteq B$.

A set $A$ is a *proper subset* of $B$ when $A \subseteq B$, but $A \neq B$.

For two sets $A$ and $B$, $A = B$ if and only if $A \subseteq B$ and $B \subseteq A$.

The number of elements in a set $A$ is denoted as $|A|$.

A set $A$ has $2^{|A|}$ subsets including $\emptyset$.

The *power set* of $A$, denoted as $2^A$, is the set of all subsets of $A$. 
An ordered pair is denoted as \((a, b)\). The ordered pair \((a, b)\) is not the same as the ordered pair \((b, a)\).

The Cartesian product \(A \times B\) of two sets is the set \(\{(a, b) : a \in A \text{ and } b \in B\}\).

A binary relation \(R\) on two sets \(A\) and \(B\) is a subset of the Cartesian product \(A \times B\).

For \((a, b) \in R\), we typically write \(a R b\).

That \(R\) is binary relation on \(A\) implies \(R\) is a subset of \(A \times A\).

**Example:** “Less than” is a binary relation on the natural numbers given by \(\{(a, b) : a, b \in \mathbb{N} \text{ and } a < b\}\).
A total or linear order $R$ on a set $A$ is a relation whereby for all $a, b \in A$ either $a R b$ or $b R a$.

In other words, every pairing of elements from $A$ can be related by $R$.

For example, “$\leq$” is a linear order on the set of natural numbers.

The function “is a descendant of” is not a linear order on the set of human beings, as there are pairs of individuals neither of whom is descended from the other.
A binary relation $R \subseteq A \times A$ is reflexive if $a R a$ for all $a \in A$. For example, “=” and “≤” are reflexive relations on $\mathbb{N}$, but “<” is not.

A relation $R$ is symmetric if $a R b$ implies $b R a$ for all $a, b \in A$. For example, “=” is symmetric, but “<” and “≤” are not.

A relation is transitive if $a R b$ and $b R c$ imply $a R c$. For example, the relations “<”, “≤”, and “=” are transitive, but the relation $R = \{(a, b) : a, b \in \mathbb{N} \text{ and } a = b - 1\}$ is not.

A relation that is reflexive, symmetric, and transitive is an equivalence relation. For example, “=” is an equivalence relation on $\mathbb{N}$, but “<” is not.
The sets $A$ and $B$ are disjoint if they have no common elements such that $A \cap B = \emptyset$.

The sets of even and odd natural numbers are disjoint.

A set $S = \{S_i\}$ of nonempty subsets forms a partition of a set $S$ if and only if:

- the subsets $S_i$ are pairwise disjoint such that $S_i \cap S_j = \emptyset$ for all $i \neq j$, and
- the union of all subsets is $S$,

$$S = \bigcup_{S_i \in S} S_i.$$ 

The sets of even and odd natural numbers form a partition of $\mathbb{N}$. 

---

**Partition**

- The sets $A$ and $B$ are **disjoint** if they have no common elements such that $A \cap B = \emptyset$.
- The sets of even and odd natural numbers are disjoint.
- A set $S = \{S_i\}$ of nonempty subsets forms a **partition** of a set $S$ if and only if:
  - the subsets $S_i$ are **pairwise disjoint** such that $S_i \cap S_j = \emptyset$ for all $i \neq j$, and
  - the union of all subsets is $S$,

$$S = \bigcup_{S_i \in S} S_i.$$ 

- The sets of even and odd natural numbers form a partition of $\mathbb{N}$. 

If $R$ is an equivalence relation on a set $A$, then for $a \in A$, the *equivalence class* of $a$ is the set $[a] = \{ b \in A : a R b \}$.

In other words, the equivalence class of $a$ is the set of all elements equivalent to $a$.

**Theorem:** The equivalence classes of any equivalence relation $R$ on a set $A$ form a partition of $A$. Any partition of $A$ determines an equivalence relation on $A$ for which the sets in the partition are the equivalence classes.
Graphs

- A graph $G = (V, E)$ is composed of a set $V$ of vertices or edges, and a set $E$ of edges.
- An edge between a vertex $v \in V$ and a vertex $w \in V$ is denoted as $(v, w)$.
- In and undirected graph $(v, w)$ is the same as $(w, v)$.
- In Fig. 1 is shown the graph $V = \{1, 2, 3, 4, 5\}$ and $E = \{(n, m)|n + m = 4 \text{ or } n + m = 7\}$.
A path in a graph is a sequence of vertices $v_1, v_2, \ldots, v_K$ for some $K \geq 1$, such that there is an edge $(v_k, v_{k+1})$ for each $1 \leq k < K$.

The length of the path is $K - 1$.

In Fig. 1, the node sequence 1, 3, 4, is a path, as is only 5.

If $v_1 = v_K$ for some $K > 1$, the path is a cycle.
A directed graph or digraph, which is also denoted as $G = (V, E)$, consists of a finite set of vertices $V$ and a set of ordered pairs of vertices dubbed arcs.

The arc from vertex $v$ to vertex $w$ is denoted as $v \rightarrow w$.

A digraph $(V, E)$ for $V = \{1, 2, 3, 4\}$ and $E = \{i \rightarrow j|i < j\}$ is shown in Fig. 2.

Figure: Simple digraph $(V, E)$ for $V = \{1, 2, 3, 4\}$ and $E = \{i \rightarrow j|i < j\}$.
Paths in Directed Graphs

- A path in a digraph is a sequence of vertices \( v_1, v_2, \ldots, v_K \) for some \( K \geq 1 \) such that \( v_k \rightarrow v_{k+1} \) is an arc for each \( 1 \leq k < K \).
- In this case we speak of a path from \( v_1 \) to \( v_K \).
- So defined, \( 1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \) is a path from 1 to 4 in the digraph shown in Fig. 2.
- For an arc \( v \rightarrow w \), we say \( v \) is the predecessor of \( w \), and \( w \) is the successor of \( v \).
There are two common representations of directed graphs.

In the first representation, for each \( v \in V \) the set of arcs leaving each vertex is stored in an adjacency list \( v\text{.adj} \).

The adjacency list \( v\text{.adj} \) contains all edges of the form \( v \rightarrow w \) for some \( w \in V \).

Such a representation is typically used for sparse graphs for which \(|E|\) is much less than \(|V|^2\).

This representation is also very flexible in that it can be applied to graphs in which the edges contain additional information, such as an input symbol, an output symbol, and a weight (i.e., a weighted finite-state transducer).
Adjacency Matrix Representation

- In the second representation, the fact that an arc $v \rightarrow w$ is stored in a bit matrix $B = [b_{m,n}]$ known as the adjacency matrix.
- If an arc exists between the $m$th and $n$th state, then $b_{m,n} = 1$.
- If no arc exists between the $m$th and $n$th state, then $b_{m,n} = 0$.
- Such a representation is useful for dense matrices, for which $|E|$ is close to $|V|^2$. 
The most basic operation on a graph is to search through it to discover all vertices.

The vertices are assigned a color during the search:
- A node $v$ that has not been previously discovered is white.
- A node $v$ that has been discovered, but whose adjacency list has not been fully explored is gray.
- After the adjacency list of $v$ has been fully explored, it is black.

The distance $v\text{.dist}$ of a node $v$ is the number of edges traversed from the start node $s$ in order to reach $v$.

The predecessor $v\text{.π}$ of a node $v$ is the node from whose adjacency list $v$ was discovered.
Breadth First Search

- Assume we have a directed graph $G = (V, E)$ where every $v \in V$ is initially white, and a first-in-first-out queue $Q$.
- The *breadth first search* (BFS) proceeds according to:

```plaintext
00  s.color ← Gray
01  s.dist ← 0
02  s.π ← NULL
03  push s on Q
04  while |Q| > 0:
    05    pop u from Q
    06    for v ∈ u.adj:
        07        if v.color == White:
            08            v.color ← Gray
            09            v.dist ← u.dist + 1
        10            v.π ← u
    11        push v on Q
12        u.color ← Black
```
Shortest Paths

- For a given source vertex $s \in V$, define the distance from $s$ to some $v \in V$ as the number of arcs traversed going from $s$ to $v$.
- Define the shortest-path distance $\delta(s, v)$ as the smallest possible distance of all paths from $s$ to $v$.
- A path from $s$ to $v$ of length $\delta(s, v)$ is said to be a shortest path.
- A shortest path from $s$ to $v$ is not necessarily unique.
Lemma: Shortest Path

Let $G = (V, E)$ be a directed graph, and let $s \in V$ be an arbitrary vertex. Then given any edge $(v, w) \in E$, it holds

$$\delta(s, w) \leq \delta(s, v) + 1.$$

**Proof:** If $v$ is reachable from $s$, then $w$ must also be reachable from $s$. In this case, the shortest path from $s$ to $w$ cannot be longer than $\delta(s, v)$ plus one for the edge $(v, w)$. 
Lemma: Distances Computed by BFS

Let $G = (V, E)$ be a directed graph. Assume that the BFS is run beginning from the source vertex $s \in V$. Upon termination, the value $v.dist$ computed by the BFS for every $v \in V$ satisfies

$$v.dist \geq \delta(s, v).$$

**Proof:** Make the inductive hypothesis $v.dist \geq \delta(s, v)$.

- Each $v.dist$ is set exactly once and never changed.
- Let $w \in V$ denote a node discovered while exploring $v.adj$.
  - **Basis:** The hypothesis clearly holds for the source vertex $s$ given the assignment in Line 01.
  - **Induction:** Let $w \in V$ denote a vertex that is discovered while expanding the adjacency list of $v \in V$. The inductive hypothesis implies $v.dist \geq \delta(s, v)$. Hence,

$$w.dist = v.dist + 1 \geq \delta(s, v) + 1 \geq \delta(s, w).$$
Theorem: Correctness of BFS

Let $G = (V, E)$ be a directed graph. Assume that the BFS is performed beginning from the source vertex $s \in V$. Upon termination, for every $v \in V$,

$$v.\text{dist} = \delta(s, v).$$

Moreover, one of the shortest paths from $s$ to $v$ is the path from $s$ to $v.\pi$, followed by the edge $v.\pi \rightarrow v$.

**Proof:** Proceeds by induction on sets of the form

$$V_k = \{v \in V : \delta(s, v) = k\}.$$
Recursive Function \texttt{visit}(u)

- Assume we have a directed graph $G = (V, E)$ where every $v \in V$ is initially white, and let \texttt{time} denote a global time stamp.

- Define the recursive function \texttt{visit}(u) for some $u \in V$.

```python
def visit(u):
    u.color ← Gray  # u has been discovered
    u.disc ← time ← time + 1
    for v in u.adj:  # explore edge (u, v)
        if v.color == White:
            v.π ← u
            visit(v)
    u.color ← Black  # u is done, paint it black
    u.fin ← time ← time + 1
```
Depth First Search

Then the complete depth first search over $G$ can be described as

```python
def dfs(G):
    for u in G.V:
        u.color ← White
        u.π ← Null
    time ← 0
    for u in G.V:
        if u.color == White:
            visit(u)
```
Let us define a directed acyclic graph (dag) $G = (V, E)$ as a digraph that contains no cycles.

A topological sort is a linear ordering of all $v \in V$ such that if $u \rightarrow v \in E$, then $u$ appears before $v$ in the ordering.

A topological sort can be performed with the following steps:

- Call DFS($G$) to determine the finishing times $v.\text{finish}$ for each $v \in V$.
- As each $v$ is finished, insert it into the front of a linked list.
- Upon termination, the linked list contains the topologically sorted vertices.
In this lecture, we have provided a basic terminology for describing sets, as well as operations and relations on them.

We have also defined graphs and, more importantly, directed graphs.

Finally, we have discussed searches through graphs, including the breadth first and depth first search.

The breadth first search is useful for determining the shortest path from the source node to a given node.

The depth first search is useful for topological sorting.