Discussion #33

Adjacency Matrices
Topics

• Adjacency matrix for a directed graph
• Reachability
• Algorithmic Complexity and Correctness
  – Big Oh
  – Proofs of correctness for algorithms
    • Loop invariants
    • Induction
Adjacency Matrix

- Definition: Let $G = (V, E)$ be a simple digraph. Let $V = \{v_1, v_2, \ldots v_n\}$ be the vertices (nodes). Order the vertices from $v_1$ to $v_n$. The $n \times n$ matrix $A$ whose elements are given by

$$a_{ij} = \begin{cases} 1, & \text{if } (v_i, v_j) \in E \\ 0, & \text{otherwise} \end{cases}$$

is the adjacency matrix of the graph $G$.

- Example:
Space: Adjacency Lists vs. Matricies

- Space \((n\) vertices and \(m\) edges)
  - matrix: \(n^2 + n \times \) (vertex-name size)
    - = matrix size + header size
    - matrix can be bits, but bits are not directly addressable
  - list: \(n \times \) (header-node size) + \(m \times \) (list-node size)

- Sparse: few edges — 0 in the extreme case
  - Matrix — fixed size: so no size benefit
  - List — variable size: as little as \(n \times \) (vertex-node size)

- Dense: many edges — \(n^2\) in the extreme case
  - Matrix — fixed size: so no size loss
  - List — variable size: as much as \(n \times \) (header-node size) + \(n^2 \times \) (list-node size)
Operations: Adjacency Lists vs. Matricies

- Operations depend on sparse/dense and what’s being done.
- Examples \((n\) nodes and \(m\) edges)
  - Is there an arc from \(x\) to \(y\)?
    - Matrix: \(O(1)\) — check value at \((x, y)\)
    - List: \(O(n)\) — index to \(x\), traverse list to \(y\) or end
  - Get successor nodes of a node.
    - Matrix: \(O(n)\) — scan a row
    - List: \(O(n)\) — traverse a linked list
  - Get predecessor nodes of a node.
    - Matrix: \(O(n)\) — scan a column
    - List: \(O(n+m)\) — traverse all linked lists, which could be as bad as \(O(n+n^2) = O(n^2)\).
Powers of Adjacency Matrices

- Powers of adjacency matrices: $A^2, A^3, \ldots$
- Can compute powers
  - Using ordinary arithmetic:
    \[
    a_{ij} = \sum_{k=1}^{n} a_{ik} \cdot a_{kj}
    \]
  - Using Boolean arithmetic:
    \[
    a_{ij} = \bigvee_{k=1}^{n} a_{ik} \land a_{kj}
    \]
Powers Using Ordinary Arithmetic

\[
A = \begin{bmatrix}
1 & 0 & 1 & 0 & 0 \\
2 & 0 & 0 & 1 & 1 \\
3 & 0 & 0 & 0 & 1 \\
4 & 0 & 1 & 0 & 1 \\
\end{bmatrix}
\]

\[
A^2 = \begin{bmatrix}
1 & 0 & 0 & 1 & 1 \\
2 & 0 & 1 & 0 & 2 \\
3 & 0 & 1 & 0 & 1 \\
4 & 0 & 1 & 1 & 2 \\
\end{bmatrix}
\]

\[
A^3 = \begin{bmatrix}
1 & 0 & 1 & 0 & 2 \\
2 & 0 & 2 & 1 & 3 \\
3 & 0 & 1 & 1 & 2 \\
4 & 0 & 2 & 1 & 4 \\
\end{bmatrix}
\]

\[
<1,2,4> \\
<2,3,4> <2,4,4> \\
<3,4,4> \\
<4,2,4> <4,4,4>
\]

\[
<1,2,3,4> <1,2,4,4> \\
<2,4,2,4> <2,4,4,4> <2,3,4,4> \\
<3,4,2,4> <3,4,4,4> \\
<4,2,3,4> <4,2,4,4> <4,4,2,4> <4,4,4,4>
\]
The element in the $i$th row and the $j$th column of $A^n$ is equal to the number of paths of length $n$ from the $i$th node to the $j$th node.
Powers Using Ordinary Arithmetic
(continued…)

\[
A = \begin{bmatrix}
1 & 2 & 3 & 4 \\
1 & 0 & 1 & 0 \\
2 & 0 & 0 & 1 \\
3 & 0 & 0 & 0 \\
4 & 0 & 1 & 0 \\
\end{bmatrix}
\]

\[
A^1 + A^2 = \# \text{ of ways in 2 or fewer}
\]

\[
A^1 + A^2 + A^3 = \# \text{ of ways in 3 or fewer}
\]

\[
A^1 + A^2 + A^3 + A^4 = \# \text{ of ways in 4 or fewer}
\]

\[
A^2 = \begin{bmatrix}
1 & 2 & 3 & 4 \\
1 & 0 & 0 & 1 \\
2 & 0 & 1 & 0 \\
3 & 0 & 1 & 0 \\
4 & 0 & 1 & 0 \\
\end{bmatrix}
\]

\[
A^3 = \begin{bmatrix}
1 & 2 & 3 & 4 \\
1 & 0 & 1 & 0 \\
2 & 0 & 2 & 1 \\
3 & 0 & 1 & 1 \\
4 & 0 & 2 & 1 \\
\end{bmatrix}
\]

Discussion #31

9/12
Powers Using Boolean Arithmetic

- Using Boolean arithmetic, we compute reachability.
- We can compute the paths of length 2 by $A^2$, 3 by $A^3$… and of length $n$ by $A^n$. Since the longest possible path we care about for reachability is $n$, we obtain all paths by $A \lor A^2 \lor A^3 \lor \ldots \lor A^n$.
- The reachability matrix is the transitive closure of $A$.
- Algorithm:

\[
R = A;
\]
\[
\text{for } (i = 2 \text{ to } n) \\
\{ \text{ compute } A^i; \\
R = R \lor A^i; \\
\}
\]

\[
\begin{align*}
\text{for } (j = 1 \text{ to } n) \\
\{ \text{ for } (k = 1 \text{ to } n) \\
\{ a_{jk}^i = 0; \\
\text{ for } (m = 1 \text{ to } n) \\
\{ a_{jk}^i = a_{jk}^i \lor (a_{jm}^{i-1} \land a_{mk}^1); \\
\}
\}
\}
\]
Reachability

\[
A = \begin{bmatrix}
1 & 2 & 3 & 4 \\
1 & 0 & 1 & 0 \\
2 & 0 & 0 & 1 \\
3 & 0 & 0 & 0 \\
4 & 0 & 1 & 0 
\end{bmatrix}
\]

\[
A \lor A^2 = \begin{bmatrix}
1 & 2 & 3 & 4 \\
1 & 0 & 1 & 1 \\
2 & 0 & 1 & 1 \\
3 & 0 & 1 & 0 \\
4 & 0 & 1 & 1 
\end{bmatrix}
\]

\[
A^2 \lor A^3 = \begin{bmatrix}
1 & 2 & 3 & 4 \\
0 & 1 & 1 & 1 \\
0 & 1 & 1 & 1 \\
0 & 1 & 0 & 1 \\
0 & 1 & 1 & 1 
\end{bmatrix}
\]

\[
A^3 = \begin{bmatrix}
1 & 2 & 3 & 4 \\
0 & 1 & 1 & 1 \\
0 & 1 & 1 & 1 \\
0 & 1 & 1 & 1 \\
0 & 1 & 1 & 1 
\end{bmatrix}
\]

1\textsuperscript{st} iteration 2\textsuperscript{nd} iteration 3\textsuperscript{rd} iteration
Timeout … for an Important CS Theme

• Programmatic problem solving
  – Understand — gather information, analyze, specify
  – Solve
    • Choose “proper” data structures and algorithms
    • Implement
    • “Prove” correct
      Test ← Exhaustively? Mathematical proof? → Prove
    Reason & Test
  – Optimize
    • Tweak? (Often not much here — compilers optimize)
    • Can I reduce the Big Oh?

• Example
  – Can we prove that the reachability algorithm works?
  – Can we find a faster algorithm to compute reachability?
The algorithm terminates.

- Loops terminate: each has a descending seq. on a well-founded poset.
- Example:

```plaintext
R = A;
for (i = 2 to n) {
    compute A_i;
    R = R \lor A_i;
}
```

The sequence \( n - i \) for \( i = 2 \) to \( n \) is a descending sequence on the natural numbers. (For our example it is \(<2, 1, 0>\).)

The algorithm produces the correct result.

- No loops: Reason that steps lead to the desired conclusion.
- Loops:
  - Find an appropriate loop invariant — a T/F condition that stays the same as the loop executes, is based on the number of iterations, and leads to the conclusion we need.
  - Prove by induction that the loop invariant holds as the loop executes.
**Loop Invariant Example:** \[ \sum_{i=1}^{n} A[i] \]

\[ \text{sum} = 0 \]
\[ \text{for } i = 1 \text{ to } n \]
\[ \text{sum} = \text{sum} + A[i] \]


**Induction proof:**

**Basis:** \( k = 0 \) (initialization, before the loop begins), \( \text{sum} = 0 \), which is the sum after 0 iterations. (Or, we could start with \( k=1 \): After 1 iteration, \( \text{sum} = A[1] \).)

Loop Invariant Example: Selection Sort

-- A is an array of n distinct integers
for i = 1 to n
find location j of the smallest integer in A[i] … A[n]
    swap A[i] and A[j]

Loop invariant:


Induction proof:

Basis: k = 0 (initialization, before the loop begins), A is an array of n distinct integers which may or may not be sorted. (Or, we could start with k=1: After 1 iteration A[1] is sorted \& A[1] < all of A[1+1], …, A[n].)

Proof of Correctness for Reachability

\[ R = A; \]
\[ \text{for (i = 2 to n)} \]
\[ \{ \text{compute } A^i; \]
\[ R = R \lor A^i; \]
\[ \} \]

- Loop invariant: After \( z \) outer loops, \( a_{xy} = 1 \) iff \( y \) is reachable from \( x \) along a path of length \( z+1 \) or less from \( x \) to \( y \).
- Note: the loop terminates after \( n-1 \) outer loops since \( i \) runs from 2 to \( n \). Thus, when the algorithm terminates, if the loop invariant holds, \( a_{xy} = 1 \) iff \( y \) is reachable from \( x \) along a path of length \( (n-1)+1 = n \) or less from \( x \) to \( y \), and thus is reachable since \( n \) is the longest a path can be (that starts at a node \( N \), visits all other nodes once, and comes back to \( N \)).
- Inductive proof
  - Basis: \( z=0 \) (i.e. before entering the outer loop), \( R = A \) and \( a_{xy} \) is 1 iff there is a path from \( x \) to \( y \); thus \( a_{xy} = 1 \) iff \( y \) is reachable from \( x \) along a path of length \( 0+1 = 1 \) or less from \( x \) to \( y \).
  - Induction: By the induction hypothesis we have that after \( z \) outer loops, \( a_{xy} = 1 \) iff \( y \) is reachable from \( x \) along a path of length \( z+1 \) or less from \( x \) to \( y \). In the \( z+1 \) iteration we compute \( A^{z+1} \), in which \( a_{xy} = 1 \) iff there is a path of length \( z+1 \) from \( x \) to some node \( q \) and an edge from \( q \) to \( y \), i.e. a path of length \( z+2 \) from \( x \) to \( y \), and we add these reachable possibilities for paths of length \( z+2 \) to the result \( R \). Hence, after \( z+1 \) outer loops, \( a_{xy} = 1 \) iff \( y \) is reachable from \( x \) along a path of length \( (z+1)+1 = z+2 \) or less from \( x \) to \( y \).