Algorithms

Grad Refresher Course 2011
University of British Columbia

Ron Maharik
maharik@cs.ubc.ca
About this talk

- For those incoming grad students who
  - Do not have a CS background, or
  - Have a CS background from a long time ago
- Discuss some fundamental concepts from algorithms and CS theory
- Ease the transition into any grad-level CS course
- Based on the 2009 version by Brad Bingham
- Some slides used from MIT OpenCourseWare
UBC CS Theory Courses (UGrad)

- **CPSC 320: Intermediate Algorithm Design and Analysis**
  - Required for CS undergrads
  - Offered in term 1 (Belleville) and term 2 (Meyer)

- **CPSC 421: Intro to Theory of Computing**
  - Offered in term 1 (Friedman)

- **CSPC 420: Advanced Alg. Design & Analysis**
  - Offered in term 2 (Kirkpatrick)
Outline

- Asymptotic Notation and Analysis
- Graphs and algorithms
- NP-Completeness & undecidability
- Resources to Learn More
Pseudocode

- How do we analyze algorithms? Start with a pseudocode description!
- Specifies an algorithm mathematically
- Independent of hardware details, programming languages, etc.
- Reason about **scalability** in a mathematical way
Insertion sort

```
INSERTION-SORT (A, n)    ▷ A[1..n]
for j ← 2 to n
    do key ← A[j]
        i ← j - 1
        while i > 0 and A[i] > key
            do A[i+1] ← A[i]
                i ← i - 1
        A[i+1] = key
```

```
A:
    1   i   j   n

sorted

key
```
Example of insertion sort

8 2 4 9 3 6

sorted  unsorted
Example of insertion sort

8 2 4 9 3 6

sorted  unsorted
Example of insertion sort

8 2
2 8

4 9 3 6

sorted  unsorted
Example of insertion sort

sorted

unsorted
Example of insertion sort

2 8 4
2 8 4
2 4 8

9 3 6
9 3 6
9 3 6

sorted
unsorted
Example of insertion sort
Example of insertion sort

8 2 4 9
2 8 4 9
2 4 8 9
2 4 8 9

sorted  unsorted

3 6
3 6
3 6
3 6
Example of insertion sort

```plaintext
8  2  4  9  |  3  6
 2  8  4  9  |  3  6
 2  4  8  9  |  3  6
 2  4  8  9  |  3  6
```

sorted  unsorted
Example of insertion sort

2  8  2  4  9  3  6
2  8  4  9  3  6
2  4  8  9  3  6
2  4  8  9  3  6
2  3  4  8  9  6
2  3  4  8  9  6
sorted  unsorted
Example of insertion sort

\[
\begin{array}{cccccccc}
8 & 2 & 4 & 9 & 3 & 6 \\
2 & 8 & 4 & 9 & 3 & 6 \\
2 & 4 & 8 & 9 & 3 & 6 \\
2 & 4 & 8 & 9 & 3 & 6 \\
2 & 3 & 4 & 8 & 9 & 6 \\
\end{array}
\]

sorted \hspace{1cm} \text{unsorted}
Example of insertion sort

8 2 4 9 3 6
2 8 4 9 3 6
2 4 8 9 3 6
2 4 8 9 3 6
2 3 4 8 9 6
2 3 4 6 8 9
2 3 4 6 8 9
done
sorted (left)
unsorted (right)
How to Analyze?

- Count operations in the Random Access Machine (RAM) model:
  - Single processor
  - Infinite memory, constant time reads/writes
  - “Reasonable” instruction set

- Asymptotically (Big-O)
- Scenarios: worst case, average case
- What to analyze: time complexity, space complexity
**Big-O Notation**

\( O(f(n)) \) is a set of functions

\[ g(n) \in O(f(n)) \iff \text{there exist constants } c, n_0 \text{ such that} \]

\[ g(n) \leq c \cdot f(n) \text{ for all } n \geq n_0 \]

- In words:
  
  “for sufficiently large inputs, the function \( g(n) \) is dominated by a scaled \( f(n) \)”

- An upper bound, in an asymptotic sense

- Usually, \( g(n) \) is a complicated expression and \( f(n) \) is simple
Big-O: Illustration
**Big-O Example**

Show that \( \frac{1}{2}n^2 + 3n \) is \( O(n^2) \)

i.e., find constants \( c \) and \( n_0 \) where

\[
\frac{1}{2}n^2 + 3n \leq c \cdot n^2, \forall n \geq n_0
\]

Solution: choose \( c=1 \), solve for \( n \):

\[
\frac{1}{2}n^2 + 3n \leq n^2 \Rightarrow 6 \leq n \Rightarrow n_0 = 6
\]

In general, ignore constants and drop lower order terms. For example:

\( 2n^4 + 6n^3 + 100n - 27 \) is \( O(n^4) \)
Big-Omega, Big-Theta

\( g(n) \in \Omega(f(n)) \iff \text{there exist constants } c, n_0 \text{ such that} \)

\[ g(n) \geq c \cdot f(n) \text{ for all } n \geq n_0 \]

• In words:
  “for sufficiently large inputs, the function \( g(n) \) dominates a scaled \( f(n) \)”

• A lower bound, in an asymptotic sense

\( g(n) \in \Theta(f(n)) \text{ if } g(n) \in O(f(n)) \text{ and } g(n) \in \Omega(f(n)) \)

• A “tight” asymptotic bound
# Complexity “Food Chain”

<table>
<thead>
<tr>
<th>Name</th>
<th>Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>Constant</td>
<td>O(1)</td>
</tr>
<tr>
<td>Logarithmic</td>
<td>O(log(n))</td>
</tr>
<tr>
<td>Linear</td>
<td>O(n)</td>
</tr>
<tr>
<td>Linearithmic</td>
<td>O(n \log(n))</td>
</tr>
<tr>
<td>Quadratic</td>
<td>O(n^2)</td>
</tr>
<tr>
<td>Polynomial</td>
<td>O(n^p)</td>
</tr>
<tr>
<td>Exponential</td>
<td>O(2^n)</td>
</tr>
</tbody>
</table>
Insertion sort

Count the number of times these lines execute

for-loop runs for n-1 iterations

while-loop runs at most j-1 iterations (on worst-case input)

- Runtime is \[ \sum_{j=2}^{n} (j-1) = n(n-1)/2 - 1 \in \Theta(n^2) \]
- In general, all sorting algorithms* are in \( \Omega(n \log(n)) \)

*(in the comparison model)
Graphs

Definition. A directed graph (digraph) \( G = (V, E) \) is an ordered pair consisting of
- a set \( V \) of vertices (singular: vertex),
- a set \( E \subseteq V \times V \) of edges.

In an undirected graph \( G = (V, E) \), the edge set \( E \) consists of unordered pairs of vertices.
Adjacency-matrix representation

The *adjacency matrix* of a graph $G = (V, E)$, where $V = \{1, 2, \ldots, n\}$, is the matrix $A[1 \ldots n, 1 \ldots n]$ given by

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

$$A = \begin{array}{cccccc} & 1 & 2 & 3 & 4 \\
1 & 0 & 1 & 1 & 0 \\
2 & 0 & 0 & 1 & 0 \\
3 & 0 & 0 & 0 & 0 \\
4 & 0 & 0 & 1 & 0 \end{array}$$

$\Theta(V^2)$ storage $\Rightarrow$ *dense representation*. 

November 9, 2005  Copyright © 2001-5 by Erik D. Demaine and Charles E. Leiserson
Single-source shortest paths

**Problem.** From a given source vertex \( s \in V \), find the shortest-path weights \( \delta(s, v) \) for all \( v \in V \).

If all edge weights \( w(u, v) \) are nonnegative, all shortest-path weights must exist.
Single-source shortest paths

**Problem.** From a given source vertex $s \in V$, find the shortest-path weights $\delta(s, v)$ for all $v \in V$.

If all edge weights $w(u, v)$ are *nonnegative*, all shortest-path weights must exist.

**Idea:** Greedy.

1. Maintain a set $S$ of vertices whose shortest-path distances from $s$ are known.
2. At each step add to $S$ the vertex $v \in V - S = Q$ whose distance estimate from $s$ is minimal.
3. Update the distance estimates of vertices adjacent to $v$. 
Example of Dijkstra’s algorithm

Graph with nonnegative edge weights:
Example of Dijkstra’s algorithm

Initialize:

\[ Q: \begin{array}{cccccc}
A & B & C & D & E \\
0 & \infty & \infty & \infty & \infty
\end{array} \]

\[ S: \{ \} \]
Example of Dijkstra’s algorithm

“\(A\)” ← \textbf{Extract-Min}(\(Q\)):

\[
\begin{array}{c}
Q: & A & B & C & D & E \\
0 & \infty & \infty & \infty & \infty & \infty \\
\end{array}
\]

\[
\begin{array}{c}
S: \{ A \}
\end{array}
\]
Example of Dijkstra’s algorithm

Relax all edges leaving $A$:

$$Q: \begin{array}{cccccc} A & B & C & D & E \\ 0 & \infty & \infty & \infty & \infty \\ 10 & 3 & \infty & \infty & \infty \end{array}$$

$$S: \{ A \}$$
Example of Dijkstra’s algorithm

“C” ← Extract-Min(Q):

\[ Q: \begin{array}{cccccc}
A & B & C & D & E \\
\hline
0 & \infty & \infty & \infty & \infty \\
10 & 3 & \infty & \infty & \infty \\
\end{array} \]

\[ S: \{ A, C \} \]
Example of Dijkstra's algorithm

Relax all edges leaving C:

\[ Q: \begin{array}{cccccc}
A & B & C & D & E \\
0 & \infty & \infty & \infty & \infty \\
10 & 3 & \infty & \infty & \infty \\
7 & & 11 & 5 \\
\end{array} \]

\[ S: \{ A, C \} \]
Example of Dijkstra’s algorithm

“E” ← Extract-Min(Q):

Q: | A | B | C | D | E |
---|---|---|---|---|---|
 0 | 10| ∞ | ∞ | ∞ | ∞ |
 7 | 3 | 11| 5 | 5 |

S: { A, C, E }

Diagram:

- A (0) connected to B (10), C (3), D (∞), E (7)
- B (10) connected to C (3), D (2)
- C (3) connected to D (8), E (2)
- E (5) connected to D (9)
Example of Dijkstra’s algorithm

Relax all edges leaving $E$:

$Q$: $\begin{array}{cccccc}
A & B & C & D & E \\
0 & \infty & \infty & \infty & \infty \\
10 & 3 & \infty & \infty & \infty \\
7 & 11 & 5 & & \\
7 & 11 & & & \\
\end{array}$

$S$: $\{A, C, E\}$
Example of Dijkstra’s algorithm

“B” ← \textbf{Extract-Min}(Q):

\begin{align*}
Q: & \quad A & B & C & D & E \\
& 0 & \infty & \infty & \infty & \infty \\
& 10 & 3 & \infty & \infty & \infty \\
& 7 & & 11 & 5 & \\
& 7 & & & & \\
\end{align*}

S: \{ A, C, E, B \}
Example of Dijkstra’s algorithm

Relax all edges leaving $B$:

$Q$: $A$ $B$ $C$ $D$ $E$

<table>
<thead>
<tr>
<th></th>
<th>$A$</th>
<th>$B$</th>
<th>$C$</th>
<th>$D$</th>
<th>$E$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S$</td>
<td>0</td>
<td>$\infty$</td>
<td>$\infty$</td>
<td>$\infty$</td>
<td>$\infty$</td>
</tr>
<tr>
<td>$S$</td>
<td>10</td>
<td>3</td>
<td>$\infty$</td>
<td>$\infty$</td>
<td>8</td>
</tr>
<tr>
<td>$S$</td>
<td>7</td>
<td>3</td>
<td>11</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>$S$</td>
<td>7</td>
<td>7</td>
<td>11</td>
<td>9</td>
<td>9</td>
</tr>
</tbody>
</table>

$S$: { $A$, $C$, $E$, $B$ }
Example of Dijkstra’s algorithm

“D” ← Extract-Min(Q):

Q: | A | B | C | D | E |
---|---|---|---|---|---|
   | 0 | ∞ | ∞ | ∞ | ∞ |
   | 10| 3 | ∞ | ∞ | ∞ |
   | 7 | 7 | 11| 5 |   |

S: { A, C, E, B, D }
Traveling Salesman Problem (TSP)

- The run time of Dijkstra's algorithm in $O(n^2)$ with naïve data structures. This is polynomial, so we say the shortest-path problem can be solved in “polynomial time”.

- What about the following (similar) problem?
- Given a directed graph with edge weights, find a path that
  1) Visits all vertices, and
  2) Minimizes the path weight (sum of edges)

- There is no known algorithm for solving this in polynomial time. Why? **TSP is NP-complete**.
Knapsack Problem
**NP-Completeness**

- NP-complete problems are a class of problems for which there is no known algorithm that run in $O(n^k)$ time, for *any* constant $k$
- Equivalently, all known algorithms for solving NP-complete problems are likely to be unacceptably slow
- If such an algorithm is found, or proven to not to exist, this solves the famous “P=NP?” question (such a proof is worth $1$ Million USD)
- NP-complete problems are *verifiable* in polynomial time
- NP-hard: problems that are “at least as hard as NP-complete”
NP-Completeness

• How do I prove that problem X is NP-complete?
  • Show that candidate solutions for X can be checked in polynomial time;
• Show that there exists an NP-complete problem Y such that an algorithm that solves X can also solve Y. This is called a reduction.
• A reduction establishes that a fast solution for X would also give a fast solution for Y.
• The first established NP-complete problem was boolean satisfiability, or SAT. This is called Cook's Theorem (1971).
• If your problem is NP-complete, you can safely give up looking for an efficient, exact algorithm.
Graph Coloring

- Given an undirected graph, color each vertex such that no two vertices of the same color share an edge. What is the fewest number of colors that can be used?
- Checking “Is a graph colorable using 2 colors?” is easy, and solvable in polynomial time.
- Checking “Is a graph colorable using 3 colors?” is NP-complete.
Example: 2-colorable Graph
Example: 2-colorable Graph
Example: 3-colorable Graph
Example: 3-colorable Graph
Undecidability

• Even worse than NP-complete, undecidable problems are those for which no algorithm can exist that is guaranteed to always solve it correctly.
• Example 1: the halting problem: “Will my problem ever stop executing?”
• Example 2: Kolmogorov complexity: “What is the simplest program that can generate a given string?”
• Problems can be shown to be undecidable by using similar reductions as for NP-completeness proofs.
Graph Traversal – BFS and DFS

http://eecourses.technion.ac.il/044268/
To learn more...

Books:

  - accessible and easy to read
  - The comprehensive algorithms “bible”
  - Often abbreviated as CLR or CLRS
To learn more...

Courses: CPSC 320, 421, 500, 506

People: BETA Lab

These slides: