Introduction to Advanced Data Structures and Algorithms

Jon Turner
Computer Science & Engineering
Washington University

www.arl.wustl.edu/~jst
Analysis of Algorithms

- Why analyze algorithms?
  - evaluate algorithm performance
  - compare different algorithms

- Analyze what about them?
  - running time, memory usage, solution quality
  - worst-case and “typical” case

- Computational complexity
  - understanding intrinsic difficulty
    - classifying problems according to difficulty
  - algorithms provide upper bound
  - to show problem is hard, must show that any algorithm to solve it requires at least a given amount of resources
    - transform problems to establish “equivalent” difficulty
Computational Problems

- Informally, a *computational problem* can be described in terms of
  - form of input provided to an algorithm for the problem
  - form of the output such an algorithm should produce
  - the relationship between input and output

**SORTING**

INPUT: A list of integers $A = (a_1, \ldots, a_n)$
OUTPUT: A list of integers $B = (b_1, \ldots, b_n)$ such that $B$
is a permutation of $A$ and $b_1 \leq b_2 \leq \cdots \leq b_n$

**MATCHING**

INPUT: A graph $G = (V, E)$ and an integer $k$
OUTPUT: A set $M \subseteq E$ such that $|M| = k$ and such that no
vertex in $V$ is incident to more than one edge of $M$
Random Access Machine

- Abstract computational model with
  - a fixed and finite program
  - an unbounded memory
  - a read-only input file
  - a write-only output file

- Each memory register can hold an arbitrary integer
- Each tape cell can hold a single symbol from a finite alphabet $\Sigma$

- Instruction set:
  - $x \leftarrow y$, $x \leftarrow y \{+,-,*,...,\} z$
  - goto label
  - if $y \{<, =, ...\} z$ goto label
  - $x \leftarrow$ input, output $\leftarrow y$

- Addressing modes:
  - $x$ may be direct or indirect reference
  - $y$ and $z$ may be constants, direct or indirect references
Asymptotic Analysis

- Focus on growth rate of running times
  » simplifies analysis
  » yields results that are largely independent of details of computational model

- Let $f$, $g$ be functions from the non-negative integers to the positive reals
  » say “$f$ is $O(g)$” if there are positive constants $c$, $n_0$ such that $0 \leq f(n) \leq cg(n)$ for all $n > n_0$
  » say “$f$ is $\Omega(g)$” if there are positive constants $c$, $n_0$ such that $0 \leq cg(n) \leq f(n)$ for all $n > n_0$
  » for example: $\log n$ is $O(n)$ and $n^2$ is $\Omega(n^2-n)$
Dose of Reality

- Classical analysis neglects some important factors
- Memory-latency gap
  - in real processors, access to main memory takes ≈100 ns
    - time enough for processor to execute hundreds of instructions
  - caches save recently-used results on-chip to avoid main memory accesses
    - relies on locality of reference observed in typical programs
    - programs using large linked data structures can exhibit poor locality of reference leading to poor cache performance
- Newer multi-threaded/multi-core processors require algorithms that can exploit parallelism
  - quad core and eight core processors with multiple threads per core now common, many more coming soon
Algorithmic Notation

- **Intervals.** An interval $[j..k]$ denotes sequence $j,...,k$  
  $[j,k..m]$ denotes the sequence $j,k,j+2(k-j),...,m$  
  **example:** $[1,3..7]$ denotes the sequence $1,3,5,7$

- **Lists.** A list $q=[x_1,...,x_n]$ is a sequence of elements; $x_1$  
  is the head, $x_n$ is the tail. Basic list operations:  
  **Access:** If $q=[x_1,...,x_n]$, $q(i)=x_i$  
  **Sublist:** $q[i..j]=[x_i,...,x_j]$  
  **Concatenation:** If $r=[y_1,...,y_m]$, $q \& r=[x_1,...,x_n,y_1,...,y_m]$

- **Sets.** A set $s=\{x_1,...,x_n\}$ is unordered collection of  
  distinct items; basic operations are union $\cup$,  
  intersection $\cap$ and difference $-$

- **Maps.** A map $f=\{[x_1,y_1],...,[x_n,y_n]\}$ is set of ordered  
  pairs, no two having same first coordinate  
  **domain($f$)$=\{x_1,...,x_n\}$ and range($f$)$=\{y_1,...,y_n\}$  
  **assignment** $f(x):=y$ adds the pair $[x,y]$ to $f$
- **Assignment**
  \[ x_1, \ldots, x_n := \text{expression} \]
  \[ x_1, \ldots, x_n := \exp_1, \ldots, \exp_n \]
  \[ x \leftrightarrow y \]

- **If statement**
  \[
  \begin{align*}
  \text{if} & \quad \text{condition}_1 \Rightarrow \text{statement list}_1 \\
  & \quad \ldots \\
  & \quad \text{condition}_n \Rightarrow \text{statement list}_n \\
  \text{fi}
  \end{align*}
  \]

- **Do statement**
  \[
  \begin{align*}
  \text{do} & \quad \text{condition}_1 \Rightarrow \text{statement list}_1 \\
  & \quad \ldots \\
  & \quad \text{condition}_n \Rightarrow \text{statement list}_n \\
  \text{od}
  \end{align*}
  \]
For statement
  for iterator ⇒ statement list rof

Subroutines
  procedure name(parameter list); statement list end
  type function name(parameter list); statement list end
  predicate name(param list); statement list end
  return or
  return expression

Example. Binary search

integer function search(list s, integer x, lo, hi);
  integer mid;
  if lo > hi ⇒ return 0 fi;
  mid := [(lo + hi)/2];
  if s(mid) = x ⇒ return mid
  | s(mid) < x ⇒ return search(s,x,mid+1,hi)
  | s(mid) > x ⇒ return search(s,x,lo,mid-1)
  fi
end;
Index-Based Data Structures

- Consider list in which list items are subset of \([1..n]\)
  - such a list can be implemented as an array of \textit{next} values
    - for item \(i\) not on list, let \(\text{next}[i]=-1\) for fast membership test
  - \([7,5,3,8,2]\)

\[
\begin{array}{cccccccccccc}
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\
\text{next} & -1 & 0 & 8 & -1 & 3 & -1 & 5 & 2 & -1 & -1
\end{array}
\]

- \textit{ListSet} defined on list items in \([1..n]\)
  - each item belongs to exactly one list (possibly singleton)
  - implement as circular lists in shared arrays \textit{next} and \textit{prev}

\[
\begin{array}{cccccccccccc}
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 \\
\text{next} & 6 & 7 & 1 & 4 & 12 & 3 & 2 & 11 & 8 & 5 & 9 & 10 \\
\text{prev} & 3 & 7 & 6 & 4 & 10 & 1 & 2 & 11 & 8 & 12 & 9 & 5
\end{array}
\]

- Index values can be used as common “handle” in multiple data structures
Algorithmic Representations of Graphs

- Basic graph provides methods for traversing adjacency lists, adding edges and IO
  - adjacency lists based on endpoints 2e, 2e+1 for edge e
- Variants add additional data and methods

C++ class hierarchy for graph types
- Graph
- WGraph
- Digraph
- Wdigraph
- Fiograph
- Wfiograph
Exercises

1. For each of the following problems, give a precise statement of the problem in the style used on page 3.

   Testing if a given string is a palindrome (reads the same way forwards and backwards).

   Input: Character string \( s = a_1 a_2 \ldots a_n \)
   Output: True if \( a_i = a_{n+1-i} \), for \( 1 \leq i \leq n \), else False.

   Find a Hamiltonian cycle (a simple cycle that includes every vertex) in an undirected graph.

   Input: An undirected graph \( G = (V, E) \) with \( n \) vertices.
   Output: A list of vertices \( u_1 u_2 \ldots u_n \) where \( u_i \in V \) for all \( i \), \( u_i \neq u_j \) for all \( i \neq j \) and \( (u_i, u_{i+1}) \in E \) for \( 1 \leq i \leq n \) and \( (u_n, u_1) \in E \).

2. For the directed tree below, list the vertices in the order they would be visited by a pre-order traversal and a post-order traversal.

   Preorder: a g d c b h f e
   Postorder: d c g b f e h a

3. In the directed graph below, list the vertices in the order they would be visited by a depth-first search and by a breadth-first search, starting from vertex \( a \). Assume that the adjacency lists are sorted by their "far" endpoints.

   Depth-first: a b c g d f e h
   Breadth-first: a b g c f d e h
4. Complete the missing entries in the following data structure representing an undirected graph.

```
<table>
<thead>
<tr>
<th>edges</th>
<th>left</th>
<th>right</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>a</td>
<td>b</td>
</tr>
<tr>
<td>2</td>
<td>c</td>
<td>e</td>
</tr>
<tr>
<td>3</td>
<td>e</td>
<td>d</td>
</tr>
<tr>
<td>4</td>
<td>c</td>
<td>b</td>
</tr>
<tr>
<td>5</td>
<td>d</td>
<td>c</td>
</tr>
<tr>
<td>6</td>
<td>a</td>
<td>d</td>
</tr>
</tbody>
</table>

adjLists

<table>
<thead>
<tr>
<th>next</th>
<th>prev</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td></td>
</tr>
</tbody>
</table>

first_endpoint

<table>
<thead>
<tr>
<th>first_endp</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

Draw a picture of the graph.