CS 375 – Analysis of Algorithms

Professor Eric Aaron

Lecture – M W 1:00pm

Lecture Meeting Location: Davis 117

Business

• HW5 in already

• HW6 out soon, due Monday, March 2

• HW3 back today
Map-Reduce, pt. 1: \textit{map}

- Two other useful functions on lists: \textit{map} and \textit{reduce}
  - The map-reduce paradigm is one of the keys to Google’s success!
    (and that of other large-scale platforms)

- \textit{map} is a function on lists that works by applying a function \( f \) to each element of a list \( L = [i_0, i_1, \ldots, i_n] \) individually, and returning the list of those results

- Example: Use map to…
  - \textit{Add 3}: Given a list of numbers \( L \), add 3 to every element on the list

First, create function \textit{add3}:
\[
def \text{add3}(n):
\text{return } n+3
\]

Then use it with \textit{map}:
\[
\text{map}(\text{add3}, L)
\]

Example: \text{map}(\text{add3}, [42, 375]) returns [45, 378]

Map-Reduce, pt. 2: \textit{reduce}

- Two other useful functions on lists: \textit{map} and \textit{reduce}
  - The map-reduce paradigm is one of the keys to Google’s success!
    (and that of other large-scale platforms)

- \textit{reduce} is a function on lists that works by applying a two-argument function \( f \) to elements of a list \( L = [i_0, i_1, \ldots, i_n] \) successively

For example, imagine you have a function \textit{add}(\( x, y \)) that returns \( x + y \)

Then, \textit{reduce}(\text{add}, [i_0, i_1, \ldots, i_n]) would return \( i_0 + i_1 + \ldots + i_n \)

- … but this doesn’t say how \textit{reduce} is implemented
Map-Reduce, pt. 3: map-reduce

We’ll come back to the question of how reduce is implemented—and, for that matter, how map is implemented, which also wasn’t specified!

First, though, it’s worth seeing how map and reduce work together

- map and reduce functions work well together!
- Example: Recall list of lists \( L=[L_0, L_1, \ldots, L_n] \) from before
  - How could we use map and reduce to find the length of the longest list?

Something similar could be done to process files, data streams, etc.

Note: With map\((f, [L_0, L_1, \ldots, L_n])\), the computations \( f(L_0), f(L_1), \ldots \) etc. can be done in parallel. Using map to parallelize computation can make map-reduce computations very efficient in multi-processor environments!

Bottleneck: full map computation must complete before reduce is called

Implementing map

- The map and reduce functions come from the paradigm of functional programming
- How might one implement these functions?
  - (Recursive implementations are a good option! Do you see why?)
- For example, how might we efficiently implement \( \text{map}(f, L) \) for our LList data structure?
- How might that differ from an efficient implementation of \( \text{map}(f, L) \) for Java/Python-like lists?

\[
\text{map}(f, L) \\
\text{Input: function } f, \text{ list } [i_0, i_1, \ldots, i_n] \\
\text{Output: List } [f(i_0), f(i_1), \ldots, f(i_n)]
\]
Implementations of \textit{map}

\begin{itemize}
  \item \textbf{Specification:}
  \begin{itemize}
    \item Input: function \(f\), list \(L = [i_0, \ldots, i_n]\)
    \item Output: list \([f(i_0), \ldots, f(i_n)]\)
  \end{itemize}

  \item \textbf{For LLLists:}
  \begin{verbatim}
  def LLmap(f,L):
    if L == []:
      return []
    else:
      return cons(f(first(L)), LLmap(f,rest(L)))
  \end{verbatim}

  \item \textbf{For Java/Python-style lists:}
  \begin{verbatim}
  def mapJP(f,L):
    if L==[]:
      return []
    else:
      return mapJP(f,L[0:len(L)-1]).add(f(L[len(L)-1]))
  \end{verbatim}
\end{itemize}

Implementing \textit{reduce}

\begin{itemize}
  \item Recall, \textit{reduce} is a function on lists that works by applying a two-argument function \(f\) to elements of a list \(L = [i_0, i_1, \ldots, i_n]\) \textit{successively}
    \begin{itemize}
      \item So, \textit{reduce} reduces a list of values to a single value
    \end{itemize}
  \item Let’s write recursive algorithms for \textit{reduce} to pair with our map functions
  \item Any questions before we start…?
    \begin{itemize}
      \item How would we start?
    \end{itemize}
\end{itemize}
Implementing \textit{reduce}

- Recall, \textit{reduce} is a function on lists that works by applying a two-argument function \( f \) to elements of a list \( L = [i_0, i_1, \ldots, i_n] \) successively
  - So, \textit{reduce} reduces a list of values to a single value
- Let’s write recursive algorithms for \textit{reduce} to pair with our map functions
- Some \textit{reduce} implementations take three arguments
  - \( f \): the 2-argument function to be applied
  - \( L \): the list of elements to be reduced to one value by \( f \)
  - \( base \): a value to return in the case where \( L \) is empty

If \( L \) were a list of numbers and \( f \) is function \texttt{add(x,y)} (returning \( x+y \)), what would be a good option for \textit{base}?

Implementing \textit{reduce}

- Recall, \textit{reduce} is a function on lists that works by applying a two-argument function \( f \) to elements of a list \( L = [i_0, i_1, \ldots, i_n] \) successively
  - So, \textit{reduce} reduces a list of values to a single value
- Let’s write recursive algorithms for \textit{reduce}:
  - \( f \): the 2-argument function to be applied
  - \( L \): the list of elements to be reduced to one value by \( f \)
  - \texttt{base}: a value to return in the case where \( L \) is empty
- How might we efficiently implement \textit{reduce}\( (f,L,\text{base}) \) for our LList data structure?
- How might that differ from an efficient implementation of \textit{reduce}\( (f,L,\text{base}) \) for Java/Python-like lists?
Implementations of \textit{reduce} 

**LLLists**

Remember the principle—let your foundations tell you what to do!

The underlying list definitions suggest different implementations for each list type.

For LLLists:

\begin{verbatim}
def LLreduce(f,L,base):
    if L == []
        return base
    else
        if rest(L) == []
            return first(L)
        else
            return f(first(L),LLreduce(f,rest(L),base))
\end{verbatim}

Implementations of \textit{reduce} 

**Java/Python-style Lists**

Remember the principle—let your foundations tell you what to do!

The underlying list definitions suggest different implementations for each list type.

For Java/Python-style lists:

\begin{verbatim}
def reduceJP(f,L,base):
    if L==[]
        return base
    else
        if len(L) == 1
            return L[0]
        else
            return f(reduceJP(f,L[0:len(L)-1],base),L[len(L)-1])
\end{verbatim}
Implementations of \textit{reduce}

For LLists:
\begin{verbatim}
def LLreduce(f,L,base):
    if L == []
        return base
    else
        if rest(L) == []
            return first(L)
        else
            return f(first(L),LLreduce(f,rest(L),base))
\end{verbatim}

For Java/Python-style lists:
\begin{verbatim}
def reduceJP(f,L,base):
    if L==[]
        return base
    else
        if len(L) == 1
            return L[0]
        else
            return f(reduceJP(f,L[0:len(L)-1],base),L[len(L)-1])
\end{verbatim}

How do these implementations reflect their different foundations?
Do they compute different results from each other?

Hint: Yes, they do.

What restrictions might be put on their usage, because of those differences?

For Your Consideration

- The map-reduce paradigm is broadly useful—it’s good to be familiar with it
  - (Origin: the recursion-heavy paradigm of functional programming!)
  - … but that’s not the only takehome message!

- In general, foundations affect algorithms, implementations, and applications
  - This is a good illustrative example, but it happens in other areas, too
  - These are the kinds of things that you should keep in mind when you design your own programming language!
## Design Paradigm Analysis

<table>
<thead>
<tr>
<th>Design Paradigm</th>
<th>Analysis</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Complexity (Efficiency)</td>
</tr>
<tr>
<td>Iterative</td>
<td>Counting</td>
</tr>
<tr>
<td></td>
<td>(Exact count of operations / space used)</td>
</tr>
<tr>
<td>Recursive</td>
<td>Solving <em>recurrences</em></td>
</tr>
</tbody>
</table>
And now, for something almost completely different

“It’s…”

The Sorting Problem; Insertion Sort

- There are many algorithms that solve the sorting problem

<table>
<thead>
<tr>
<th>Input: A sequence L of n numbers &lt;a₁, ..., aₙ&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output: A permutation (reordering) &lt;b₁, ..., bₙ&gt; of the input sequence (perhaps leaving them unchanged) such that b₁ ≤ b₂ ≤ ... ≤ bₙ</td>
</tr>
</tbody>
</table>

- What algorithms are you familiar with?
- Sorting algorithms can be expressed iteratively or recursively

What’s the fastest sorting algorithm?
... and how would we even know?
... and what does “fastest” mean in context, anyway?
The Sorting Problem; Insertion Sort

• There are many algorithms that solve the sorting problem

<table>
<thead>
<tr>
<th>Input: A sequence L of n numbers &lt;a₁, ..., aₙ&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output: A permutation (reordering) &lt;b₁, ..., bₙ&gt; of the input sequence (perhaps leaving them unchanged) such that b₁ ≤ b₂ ≤ ... ≤ bₙ</td>
</tr>
</tbody>
</table>

- What algorithms are you familiar with?
- Sorting algorithms can be expressed iteratively or recursively

• One classic, efficient algorithm: Insertion sort
  - Based on its name, any ideas about how it might work on an array?

  How could we write this algorithm iteratively (in pseudocode)?

Insertion Sort: An Example

• Example of insertion sort working on an array (taken from CLRS, section 2.1)
  - Any questions on what the insert part of the algorithm does, and how it’s used to sort an array?

![Diagram of Insertion Sort Example](image)