CS 375 – Analysis of Algorithms

Professor Eric Aaron

Lecture – M W 1:00pm

Lecture Meeting Location: Davis 117

Business

• HW3 in already
  – (How did it go?)
• HW4 out soon, due Monday, Feb. 24
• HW1 back today
• Weekly TA Hours (at least on weeks when HWs are due)
  – When: Sunday, 7:30-8:30pm
  – Where: Davis 117
List Algorithms

- We’ve seen how the definition of a binary tree can guide the design of algorithms on binary trees...
- Many common algorithms are written on lists
  - How does the definition of a list guide the designs of those algorithms?

What are some differences between arrays and lists, in this context?
- Among the significant differences between an array and a list:
  - Arrays are stored as contiguous blocks of memory; lists, when not simply extensions of arrays, are node-based (linked lists)
  - Arrays have direct, constant-time indexed access to any element; lists require traversing a list to reach an element, which is not constant-time
  - Because lists are node-based, it can be constant-time to access the sub-list of all but the first element as a distinct object... as LLists do

List Algorithms:
Remove (first occurrence of an element)

- Consider the problem of removing the first occurrence of an element from a sequence, specified here for a list

Input: item \( i \) and list \( L = [x_0, \ldots, x_n] \)
Output: If \( i = x_k \) and \( k \) is the smallest value for which \( i = x_k \),
  return \( [x_0, \ldots, x_{k-1}, x_{k+1}, \ldots x_n] \)
  Otherwise—i.e., when there is no \( k \) such that \( i = x_k \)—return \( L \)

- How would you design an algorithm to solve this problem...
  - Iteratively?
  - Recursively?
  - How would the complexity of this be different on a list (i.e., a linked list) than on an array?
Definition of our \textit{LList} Data Structure

• Throughout CS375, we will sometimes refer to an \textit{LList} data structure, representing a list of elements
• In English, we’d say an LList is:
  – Either empty,
  – Or
    • an element, called \textit{first}
    • and an LList, called \textit{rest}, representing all the elements after \textit{first}

Is this a good definition? Consider Principle 1: Keep your foundations simple....

Is this consistent with your understanding of list structures—that is, \textit{linked list} structures (which are typically node-based in implementation)?

NOTE: This definition may show up on HW, too!

Definition of our \textit{LList} Data Structure

• In English, we’d say an LList is:
  – Either the empty list,
  – Or
    • an element, called \textit{first}
    • and an LList, called \textit{rest}, representing all the elements after \textit{first}

• To be unambiguous about how we work with LLists, these will be the primitive functions defined on LLists:
  – \texttt{first(L)}: returns the \textit{first} element of an LList \texttt{L}
  – \texttt{rest(L)}: returns the \textit{rest} sublist of an LList \texttt{L}
  – \texttt{cons(v,L)}: a \textit{constructor} function that takes an item \texttt{v} and an LList \texttt{L} and returns a new LList \texttt{L’} such that
    • \texttt{v} is the element \textit{first} of \texttt{L’}
    • \texttt{L} is the sublist \textit{rest} of \texttt{L’}

Sometimes, the empty list is written as \texttt{[]}.

We could display all LLists in \texttt{[brackets]}, as is usual for lists.

We’ll treat all three of these as constant-time \texttt{fn}s.

For example, how would you write this LList as a list in \texttt{[brackets]}?

\texttt{cons(1,(cons(2,cons(3,[]))))}

NOTE: This definition may show up on HW, too!
Definition of our LList Data Structure

- To be unambiguous about how we work with LLists, these will be the primitive functions defined on Llists:
  - first(L): returns the first element of an LList L
  - rest(L): returns the rest sublist of an LList L
  - cons(v,L): a constructor function that takes an item v and an LList L and returns a new LList L’ such that
    • v is the element first of L’
    • L is the sublist rest of L’

Important note: first(L), rest(L), and cons(v,L) are functions that return values; they are not fields of an object. Because of this, we cannot assign values to them—e.g., first(L) = 3 or rest(L) = [3] is not permitted.

What could be done instead, with this syntax, to change the first element of some LList L to 3?

[Ans: We could do L = cons(3,rest(L)) ]
LList Example:

Remove (first occurrence of an element)

- Consider the problem of removing the first occurrence of an element from a sequence, specified here for a list

  **Input:** item \( i \) and LList \( L = [x_0, \ldots, x_n] \)

  **Output:** If \( i = x_k \) and \( k \) is the smallest value for which \( i = x_k \)
  
  return LList \([x_0, \ldots, x_{k-1}, x_{k+1}, \ldots x_n]\)

  Otherwise—i.e., when there is no \( k \) such that \( i = x_k \)—return \( L \)

- How would you design an algorithm to solve this problem?

  It’s going to be recursive, because the definition of LList is recursive. Follow the definition!

  How would we break the problem down into 1 or more smaller subproblems, and then use the result(s) in a solution for the original problem?

LList Example:

Remove (first occurrence of an element)

- Consider the problem of removing the first occurrence of an element from a sequence, specified here for a list

  **Input:** item \( i \) and LList \( L = [x_0, \ldots, x_n] \)

  **Output:** If \( i = x_k \) and \( k \) is the smallest value for which \( i = x_k \)
  
  return LList \([x_0, \ldots, x_{k-1}, x_{k+1}, \ldots x_n]\)

  Otherwise—i.e., when there is no \( k \) such that \( i = x_k \)—return \( L \)

- How would you design an algorithm to solve this problem?

  **Algorithm:** LLRemove\((i, T)\)
  
  // see specification immediately above
  
  if \( L = [\] \)
  
  return \( L \)

  else
  
  if \( i = \text{first}(L) \)
  
  return \( \text{rest}(L) \)

  else:
  
  return cons(\text{first}(L), LLRemove\((i, \text{rest}(L))\))
Break It Down Again

• In general, different ways of breaking down a problem into subproblems can lead to different algorithms e.g., Mergesort vs. ... any other sort, basically

• Different data structures, by their definitions, suggest different natural ways to break problems into subproblems
  – How would an IntBinTree suggest breaking a problem into subproblems?
  – How would a list (node-based, e.g., LList) would suggest breaking a problem into subproblems?
  – How about for an array?

This isn’t to say that, for any given data structure, some approach is always applied!
This is just looking for common approaches, and what makes them natural in context.

Break It Down Again

• In general, different ways of breaking down a problem into subproblems can lead to different algorithms e.g., Mergesort vs. ... any other sort, basically

• Different data structures, by their definitions, suggest different natural ways to break problems into subproblems
  – How would an IntBinTree suggest breaking a problem into subproblems? [subproblems on sub-trees—kinda one half at a time]
  – How would a list (node-based, e.g., LList) would suggest breaking a problem into subproblems? [subproblems on lists one element shorter]
  – How about for an array? [subproblems involve changing indices and iterating over indexed ranges—index access is central to arrays!]

In all of these cases, the foundations—the definitions of the underlying structure—suggest that approach to breaking into subproblems
Break It Down Again

• In general, different ways of breaking down a problem into subproblems can lead to different algorithms
• Looking back to the search problem…

  Input: item i and collection L
  Output: True if i is an element of L, False otherwise

  – We’ve written an algorithm for this on IBT’s
  – What’s a straightforward way to solve this on arrays?

Break It Down Again

• In general, different ways of breaking down a problem into subproblems can lead to different algorithms
• Looking back to the search problem…

  Input: item i and collection L
  Output: True if i is an element of L, False otherwise

  This is the same specification as for lists, but extended to apply to other collections, too

  – We’ve written an algorithm for this on IBT’s
  – What’s a straightforward way to solve this on arrays? [sequential search]

• What’s a more efficient, perhaps less straightforward solution on arrays?
Binary Search

- *Binary search*: Divide-and-conquer search algorithm on arrays
  - Designed for *sorted* arrays—uses fact that array is sorted for more efficient algorithm
- Are you familiar with this algorithm?
  - How could search be made more efficient on a sorted array?
  - What’s a recursive algorithm for binary search?

**Important Note:** The divide-and-conquer paradigm is broadly applicable!

Even though arrays don’t naturally suggest recursive strategies the way IntBinTrees and LLists do, it can still be a useful paradigm on arrays.

Binary search is a classic algorithm!

---

**Algorithm:** `BinSrch(A,v,low,high)`
- If `low > high`
  - Return `False`
- Else
  - `mid = (low+high)/2` # int division
  - If `v == A[mid]`
    - Return `True`
  - Else if `v > A[mid]`
    - Return `BinSrch(A,v,mid+1,high)`
  - Else # must be `v < A[mid]`
    - Return `BinSrch(A,v,low,mid-1)`

**Problem:**
- Input: sorted array `A`, value `v` for which to search, integers `low` and `high` to specify range of `A` in which to search
- Output: True if `v` is an element of `A[low..high]`, False otherwise
Binary Search

- **Binary search**: Divide-and-conquer search algorithm on arrays
  - Designed for *sorted* arrays—uses fact that array is sorted for more efficient algorithm

- Are you familiar with this algorithm?
  - How could search be made more efficient on a sorted array?

`Algorithm: BinSrch(A,v,low,high)`

if low > high
  return False
else
  mid = (low+high)/2 # int division
  if v == A[mid]
    return True
  elif v > A[mid]
    return BinSrch(A,v,mid+1,high)
  else # must be v < A[mid]
    return BinSrch(A,v,low,mid-1)

**Note**: It’s the same sequence A each time. Copying or altering A (with, e.g., list slicing) would take extra time

**Important**: The recursive cases bring the sub-problems closer to the base case where low > high

**What would the initial call to this function be, to find v in all of A?**