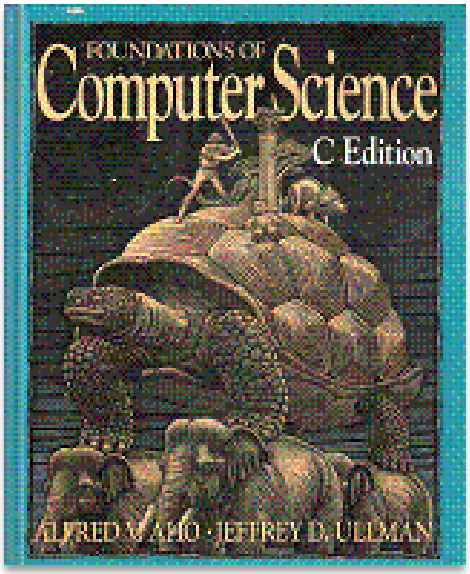
Data Structure

List

Shin Hong

Mar 17, 2020



http://infolab.stanford.edu/~ullman/focs.html

Chapter 6. The List Data Model

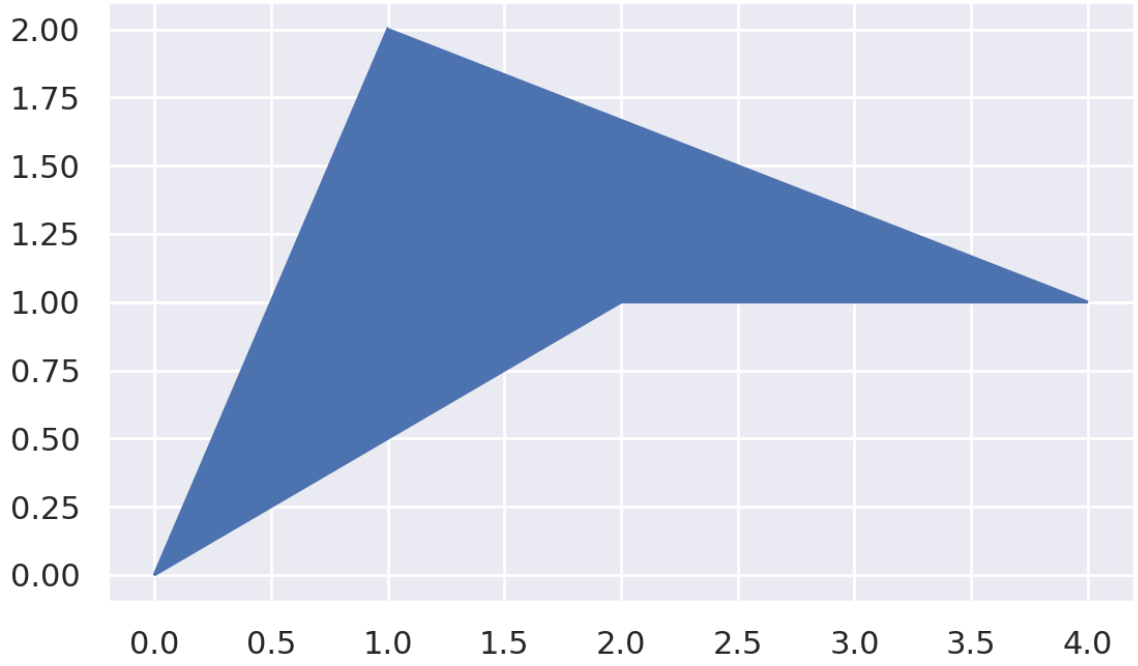
List

- A list is a finite sequence of zero or more elements
  - a list is a list of a type *T* if all its elements belong to *T*
  - a list is written with its elements separated by commas and enclosed in parentheses: $(a_1, a_2, \ldots, a_n)$
    - we say that element a_i occurs at position i

- Examples
  - (2, 3, 5, 7, 11, 13, 17, 19)
  - (helium, neon, argon, krypton, xenon, radon)
  - (31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31)
  - A text document is a list of strings, and a string is a list of characters

Examle: 2D Polygon

- A list of points where the first and the last are the same
- A point is a list of two real numbers (or a pair of two real numbers)
- Ex. ((0,0), (2, 1), (4, 1), (1, 2), (0,0))



Attributes of List

- The length of a list of the number of occurrences of elements on the list
  - the empty list a list of length 0
  - the length counts positions, not distinct symbols

- A non-empty list has a head and a tail
  - head: first element
  - tail: the remainder list excluding the first element
  - ex. (helium, neon, argon, krypton, xenon, radon)
    - head: helium
    - tail: (neon, argon, krypton, xenon, radon)

Sublist and Subsequence

- A sublist of a list $L = (a_1, a_2, \ldots\ a_n)$ is a list formed by starting at a position i and taking all the elements up to a later position j $(a_i, a_{i+1}, \ldots\ a_j)$ for $1 \leq i \leq j \leq n$, or ϵ
  - a sublist is sometime called as substring
  - prefixes and suffixes are sublists
- A subsequence is a list $L = (a_1, a_2, \ldots\ a_n)$ formed by selecting some elements while keeping the same order, $(a_{k_1}, a_{k_2}, \ldots\ a_{k_m})$ where $1 \leq m \leq n$ and $k_j < k_{j+1}$ for $1 \leq j < m$ or ϵ
- E.g., Given list (a, b, c)
  (a,b) is a sublist, but (a,c) is not a sublist;
  (a,c) is a subsequence where m=2 and $a_{k_1} = 1$ and $a_{k_2} = 1$

Operations on List

- insertion
- deletion
- lookup
- concatenation
- sorting
- merging

Insertion, Deletion and Concatenation

- Inset an element x onto a list L
  - add x after the last element
  - add one more occurrence of x
- Delete an occurrence of x from L
  - need to specify which occurrence to delete
    - e.g., delete first occurrence, delete all occurrences, etc.
- Concatenate two lists L and M by forming the list that begins with the elements of L and continues with the elements of M

Data Structure for List

- Data structure
  - Data type
  - A set of operations
- List data structure
  - Array list
  - Linked list
- List operations
  - add, remove, delete, retrieve, concatenate, etc.

Array-based List

- An array-based list consists of
  - an array to hold elements
  - a variable to represent the number of the elements currently held
  - a variable to represent the number of elements possible to hold (i.e., capacity)

- Operations
  - insert a new integer to the list
  - look up from the list to check whether a specific integer is stored in the list
  - remove a specific element from the list
  - merge two lists into one list

Linked List Data Structure (1/4)

- A list is either an empty list, or a pair of an integer (head) and a following list (tail)
- An integer is contained in a list if the integer is the head of the list, or if the integer belongs to the tail
  - an empty list does not contain any integer

Linked List Data Structure (2/4)

- A linked list can be represented as a pointer to first node
  - an empty list is represented as a null
  - node is a structure to represent a pair of an integer and a list, which is a building block of a linked list
  - a node consists of an integer and a pointer to a list

Linked List Data Structure (3/4)

- Operations
  - insert a new integer to the list
  - look up from the list to check whether a specific integer is stored in the list
  - remove a specific element from the list
  - merge two lists into one list

- See `linkedlist/version1`

Linked List Data Structure (4/4)

- A linked list can be represented as a pointer to first node and a pointer to last node
  - a node consists of an integer and a pointer to next node
  - Ex. see `linkedlist/ver2`

- Array-based list vs. Linked list

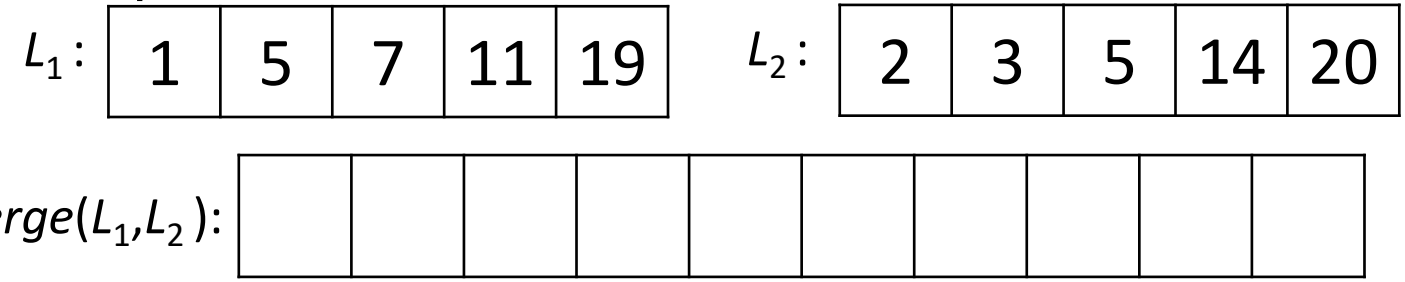
Sorted List

- A sorted list arranges elements in ascending/descending order
  - A sorted list without duplicate elements works as a set
  - Ex. `sortedlist/arraylist`

- Operations
  - insert a new integer to the list
  - look up from the list to check whether a specific integer is stored in the list
  - remove a specific element from the list
  - merge two lists into one list

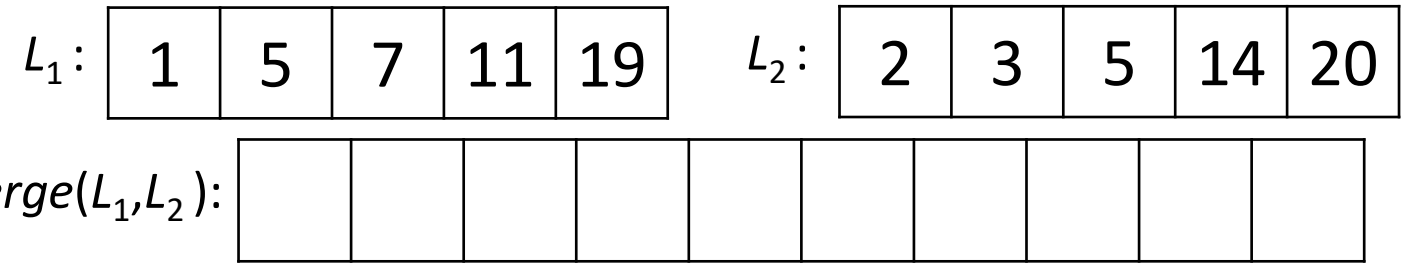
Merge: 1st Approach

- For given two lists L_1 and L_2,
  1. create a new list by concatenating the two lists,
  2. relocate each element e of L_2 by repeatedly swapping e with the one in the earlier position until the earlier one is less than e
- Example:



Merge: 2nd Approach

- For given two lists L_1 and L_2,
  1. For each element e in L_2,
     1. Find the last position p in L_1 such that all elements earlier than p are less than e,
     2. Move L_1 elements upto p to the new list
     3. Insert e into the new list
  2. If there exist remaining elements in L_1, insert them to the new list
- Example:



Merge: 3rd Approach

- For given two lists L_1 and L_2,
  take a greater one from the two first elements of L_1 and L_2,
  insert it to the new list until nothing remains in L_1 and L_2.
- Example: