Basic Abstract Data Types

CSE 122 ~ Algorithms & ADT

Basic Abstract Data Types

- ADT ~ a mathematical model with a collection of operations defined on that model
  - An interface that defines a data type and operations on values of that data type
  - Interface hides the details of the implementation from the client
    - Interface ~ header files in C
    - Implementation ~ *.c files
- ADTs are independent of implementation

Lists

- ➔ Definition: mathematically a sequence of zero or more elements of a given type
  - a_1, a_2, … , a_n where n ≥ 0 and a_i is of some element type
- ➔ Examples:
  - List of ints, list of floats, list of structures that you defined
  - Ints, floats, points are the elements type

```
struct point{
    int x;
    int y;
}
```

Examples of Lists

- Usually lists are of the same element
- Character strings
  - List of single characters
- Documents
  - A list of lines, which in turn are a list of characters
- n-dimensional points

Notation/Terminology

- ➔ The number of elements *n* is said to be the **length** of the list
  - If n ≥ 1 then a_1 is the first element or head and a_n is the last element or tail
  - If n = 0, the list is empty
- ➔ Sublist
  - Started at some position *i* and taking all the elements up to some position *j*
- ➔ Subsequence
  - Formed by striking out zero or more elements of the list
  - MUST appear in the same order in which they appear in the original list
  - Need not be consecutive

Notation/Terminology

- Let L be the character string “abc”
  - Sublists of L are:
    - Ø,a,b,c,ab,bc,abc
  - Subsequences of L are:
    - All of the above plus ac
- Let L be the character string “abab”
  - Sublists of L are:
  - Subsequences of L are:

Notation/Terminology

- Let L be the character string “abc”
  - Sublists of L are:
    - Ø,a,b,c,ab,bc,abc
  - Subsequences of L are:
    - All of the above plus ac
- Let L be the character string “abab”
  - Sublists of L are:
    - Ø,a,b,ab,ba,aba,bab,abab
  - Subsequences of L are:
    - All of the above plus aa,bb,aab,abb

Operations

- What can you do with a list?
  - end(L)
    - Returns position following n in an n-element list
  - insert(x,p,L)
    - Insert x at position p in List L
  - locate(x,L)
    - Returns the position of x in List L
  - retrieve(p,L)
    - Returns element at position P in List L
  - delete(p,L)
    - Delete element at position p in List L
  - next(p,L)
    - Returns the position immediately after position p in List L
  - previous(p,L)
    - Returns the position proceeding position p in List L
  - makenull(L)
    - Empties List L
  - first(L)
    - Returns the first position on List L
  - printlist(L)
    - Print the elements of L in order of occurrence

Operations

- What can you do with a list cont.
  - lookup(x,L)
    - Return true or false depending on whether element x is in List L
  - concatenation(L_1, L_2)
    - Attach List L_2 to the end of List L_1
  - length(L)
    - return the length of List L
  - isEmpty(L)
    - Return true or false depending on whether List L is empty

Purging Duplicates

- Let us use these operations to purge a list of duplicates
- Assume you have a function *same(x,y)* that returns true if x and y are the same and false if they are not.
    - Sameness depends on the data type

Purging Duplicates

```
//remove duplicates from list
purge(L)
     p, q: position      //p is current postion
                              //q moves ahead to find equal elements
   p = first(L);
     while <> end(L)
          q = next(p,L)
          while q <> end(L)
               if same(retrieve(p,L), retrieve(q,L))
                    delete(q,L)
               else
                    q := next(q,l)
          p := next(p,L)
```

Implementation

- ➔ Array Implementation
  - Traversing is straight forward
  - Easy to append as long as there is space
  - Deletion involves shifting elements
- ➔ Singly Linked Lists
  - Insertions ~ 3 cases at head, middle, end
  - Deletions ~ 3 cases at head, middle, end
- ➔ Doubly Linked Lists
  - Insertions ~ 3 cases at head, middle, end
  - Deletions ~ 3 cases at head, middle, end
- ➔ Circular Lists
  - Singly linked list where the tail points back to the head
  - Insertions ~ 3 cases at head, middle, end
  - Deletions ~ 3 cases at head, middle, end

Advantages and Disadvantages

- Arrays
  - Pros: simple and easy to use, constant access time to elements, arrays are contiguous blocks of memory so neighbors are close to each other ~ cpu caching exploits this
  - Cons: fixed size, position based insertion and deletion, have to move elements
- Dynamic Arrays
  - Pros: can grow as needed, random access
  - Cons: as arrays grow ~ create memory and copy old array into new array - can be expensive, position based insertion and deletion - have to move elements

Advantages and Disadvantages

- Linked Lists
  - Pros: can add one element at a time without the worry of copying and reallocating memory
  - Cons: access time - sequential so O(n), unlike random access O(1)

Fun Things to Do With Lists

- Floyd’s Algorithm
- Josephus problem

Floyd's Algorithm

- ➔ Tortoise and a Hare running on a track - the hare will lap the tortoise

```
contains_loop(Node *head){
     Node *slow = head;
     Node *fast = head;

     while (slow && fast){
          fast = fast->next;
          if(fast == slow) return 1; //a cycle
          if (fast == NULL) return 0; //no cycle
          fast = fast->next;
          if(fast == slow) return 1; //cycle
          slow = slow->next;
     }
     return 0; //cycle not found
}
```

Josephus Problem

- Jewish revolt against Rome [66 CE]
- Josephus and 39 of his comrades were holding out against the Romans in a cave. With defeat imminent, they resolved that, like the rebels at Masada, they would rather die than be slaves to the Romans.
- They decided to arrange themselves in a circle. One man was designated as number one, and they proceeded clockwise, killing every 7th man
- Josephus (an accomplished mathematician according to the story) immediately figured out where to stand in order to be the last to die
- Once he was the last man standing, he simply joined the Romans (:
- Example:
  - Circular list of numbers 1,2,3,4,5,6,7,8,9
  - Kill every 5th man
  - Who is the last man standing?

Josephus Problem

- Jewish revolt against Rome [66 CE]
- Josephus and 39 of his comrades were holding out against the Romans in a cave. With defeat imminent, they resolved that, like the rebels at Masada, they would rather die than be slaves to the Romans.
- They decided to arrange themselves in a circle. One man was designated as number one, and they proceeded clockwise, killing every 7th man
- Josephus (an accomplished mathematician according to the story) immediately figured out where to stand in order to be the last to die
- Once he was the last man standing, he simply joined the Romans (:
- Example:
  - Circular list of numbers 1,2,3,4,5,6,7,8,9
  - Kill every 5th man
  - Who is the last man standing?
    - 5,1,7,4,3,6,9,2 → last man standing is 8