List abstract data type and an "array list" implementation

Comp Sci 1575 Data Structures

Data Structures...

Reminder

- Welcome to the first day of Data Structures.
- Now is when it will start to be very helpful to have read the book chapter. The slides, book, and code will parallel each other.

Outline

Introduction
Abstract data types: interfaces
List
Unsorted set
Sorted set
Priority queue
Graphs
Lists are fundamental
Definitions
List operations
List implementation demands
Arrays as lists?
Current position
Example tasks
Iterate
Search
Insert
An actual implementation

Outline

Introduction
Abstract data types: interfaces
List
Unsorted set
Sorted set
Priority queue
Graphs
Lists are fundamental
Definitions
List operations
List implementation demands
Arrays as lists?
Current position
Example tasks
Iterate
Search
Insert
An actual implementation

Interfaces

- List
- Unsorted set (USet)
- Sorted set (SSet)
- Priority queue /
- Graph

List implementations

List implementations			
	get(i)/set(i,x)	add(i,x)/remove(i)	
ArrayStack	$O(1)$	$O(1+n-i)^A$	§ 2.1
ArrayDeque	$O(1)$	$O(1+\min\{i,n-i\})^A$	§ 2.4
DualArrayDeque	$O(1)$	$O(1+\min\{i,n-i\})^A$	§ 2.5
RootishArrayStack	$O(1)$	$O(1+n-i)^A$	§ 2.6
DLList	$O(1+\min\{i,n-i\})$	$O(1+\min\{i,n-i\})$	§ 3.2
SEList	$O(1+\min\{i,n-i\}/b)$	$O(b+\min\{i,n-i\}/b)^A$	§ 3.3
SkiplistList	$O(\log n)^E$	$O(\log n)^E$	§ 4.3

Unsorted set implementations

USet implementations			
	`find(x)`	`add(x)`/`remove(x)`	
`ChainedHashTable`	$O(1)^{E}$	$O(1)^{A,E}$	§ 5.1
`LinearHashTable`	$O(1)^{E}$	$O(1)^{A,E}$	§ 5.2

Sorted set implementations

SSet implementations			
	find(x)	add(x)/remove(x)	
SkiplistSSet	$O(\log n)^{E}$	$O(\log n)^{E}$	§ 4.2
Treap	$O(\log n)^{E}$	$O(\log n)^{E}$	§ 7.2
ScapegoatTree	$O(\log n)$	$O(\log n)^{A}$	§ 8.1
RedBlackTree	$O(\log n)$	$O(\log n)$	§ 9.2
BinaryTrie^{I}	$O(w)$	$O(w)$	§ 13.1
XFastTrie^{I}	$O(\log w)^{A,E}$	$O(w)^{A,E}$	§ 13.2
YFastTrie^{I}	$O(\log w)^{A,E}$	$O(\log w)^{A,E}$	§ 13.3

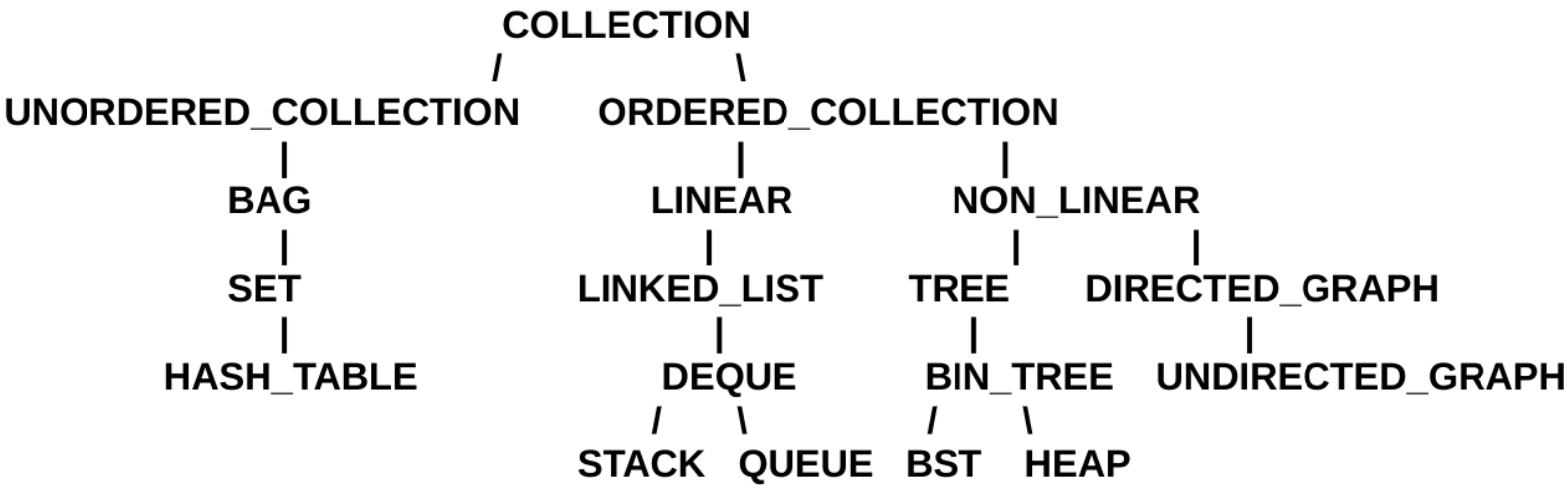
Priority queue implementations

(Priority) Queue implementations			
	`findMin()`	`add(x)`/`remove()`	
`BinaryHeap`	$O(1)$	$O(\log n)^{A}$	§ 10.1
`MeldableHeap`	$O(1)$	$O(\log n)^{E}$	§ 10.2

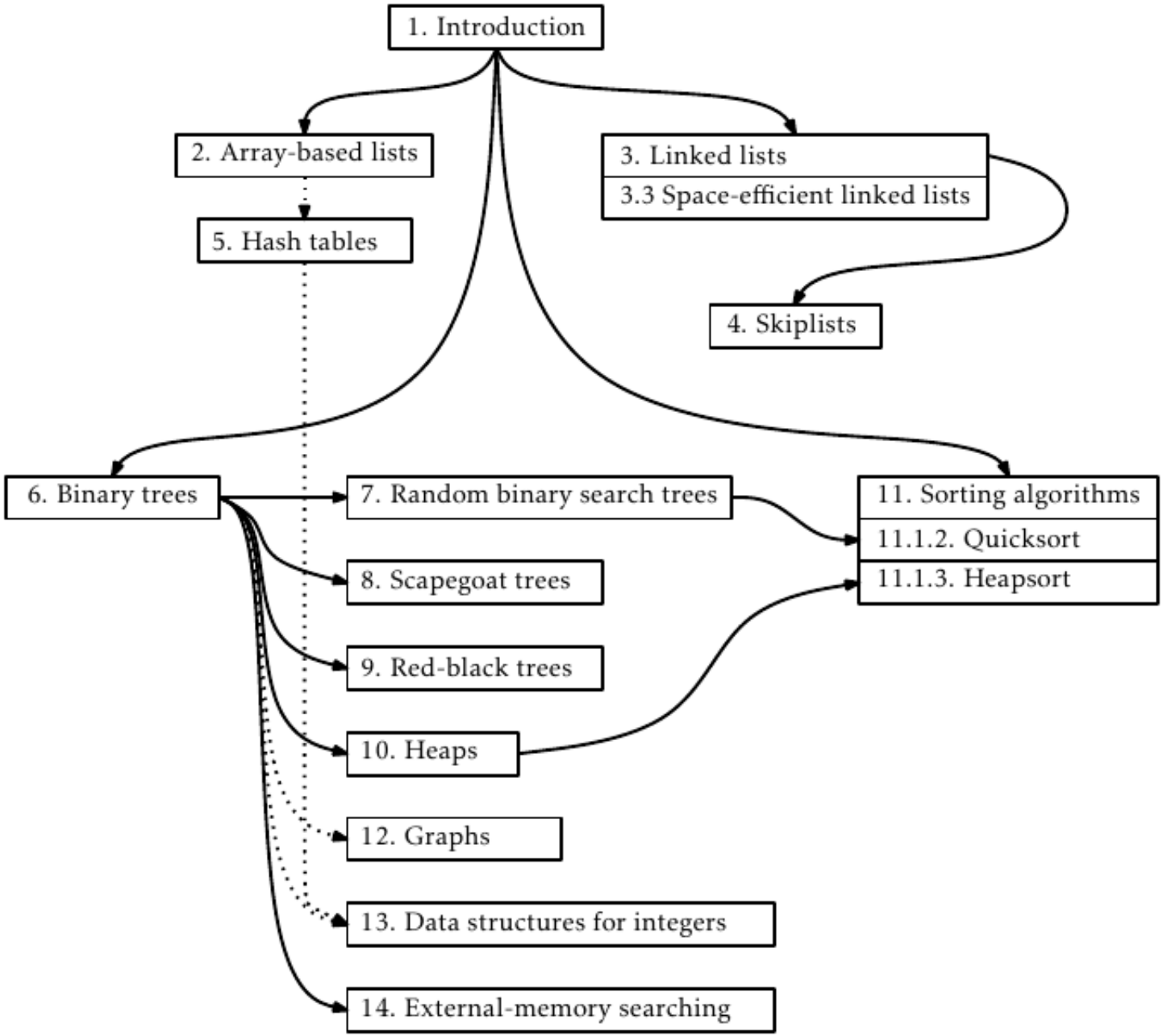
Graph implementations

- Adjacency matrix
- Adjacency list (array and linked)
- Incidence list

Big picture on ADTs



Chapter dependencies in ODS



Outline

Introduction
Abstract data types: interfaces
List
Unsorted set
Sorted set
Priority queue
Graphs
Lists are fundamental
Definitions
List operations
List implementation demands
Arrays as lists?
Current position
Example tasks
Iterate
Search
Insert
An actual implementation

List ADT: Why do we care?

- A list is the most universal user interface for computers: e.g., Bash's $ls, your inbox, your google results, your file browser, etc.
- MANY programs use lists in at least some part of their back-end
- What is the common-language definition of a list?
- What is our formal definition of a list ADT?

Outline

Introduction
Abstract data types: interfaces
List
Unsorted set
Sorted set
Priority queue
Graphs
Lists are fundamental
Definitions
List operations
List implementation demands
Arrays as lists?
Current position
Example tasks
Iterate
Search
Insert
An actual implementation

Definitions: List ADT

- A list is a finite, ordered **sequence** of data items.
- List elements have a **position** or order.
- Ordered is not the same as sorted, which relates order to value; lists can be sorted or unsorted, but are still ordered
- Notation: $< a_0,\ a_1,\ \ldots,\ a_{n-1} >$
- Length/Size of the list is $n - 1$
- Position of element a_i in the list is i
- First element of the list is a_0, the **head**
- Last element is a_{n-1}, the **tail**
- a_i follows (or succeeds) a_{i-1} where $(i < n)$
- a_{i-1} precedes a_i where $(i > 0)$.
- Each list item has a data type
- Operations??

Outline

Introduction
Abstract data types: interfaces
List
Unsorted set
Sorted set
Priority queue
Graphs
Lists are fundamental
Definitions
List operations
List implementation demands
Arrays as lists?
Current position
Example tasks
Iterate
Search
Insert
An actual implementation

Which list ADT operations do we want?

- Should be able to be ordered, and have a position
- Should grow and shrink as we add/insert or remove items
- Should be able to insert and remove elements from anywhere in the list.
- Should be able to gain access to any element's value, either to read it or to change it.
- Should be able to create and clear (or reinitialize) lists.
- Convenient to access next or previous element from "current" one.
- Should be able to locate and/or read items by value or position
- Can have many more arbitrary actions if desired

Check out the pure virtual abstract class, list.h

Outline

Introduction
Abstract data types: interfaces
List
Unsorted set
Sorted set
Priority queue
Graphs
Lists are fundamental
Definitions
List operations
List implementation demands
Arrays as lists?
Current position
Example tasks
Iterate
Search
Insert
An actual implementation

Outline

Introduction
Abstract data types: interfaces
List
Unsorted set
Sorted set
Priority queue
Graphs
Lists are fundamental
Definitions
List operations
List implementation demands
Arrays as lists?
Current position
Example tasks
Iterate
Search
Insert
An actual implementation

How can we accomplish operation implementation?

What about normal arrays? Aren't they already complete lists?

```
int x[4] = {1,5,8,2};
x[0] = 3;
x[1] = 3;
x[2] = 3;
x[3] = 3;
```

- Ordered?
- What about search, insert, remove, grow, shrink, next, previous, length, print whole list, append, pop?

To increase our degree of modularity in operation between functions, which construct is very helpful to keep track of to complete many of the above tasks?

Outline

Introduction
Abstract data types: interfaces
List
Unsorted set
Sorted set
Priority queue
Graphs
Lists are fundamental
Definitions
List operations
List implementation demands
Arrays as lists?
Current position
Example tasks
Iterate
Search
Insert
An actual implementation

Current position construct is helpful to implement

- Most operations can be made to act relative to a current position or a specified position (which could be set to current to complete a task)
- The in-class implementation will support a current position, illustrated here using notation $|$, e.g.,$< 20, 23, |12, 15 >$
- For example, listObject.insert(x) could produce $< 20, 23, x, |12, 15 >$
- This is not true for std:: implementations we'll cover

Outline

Introduction
Abstract data types: interfaces
List
Unsorted set
Sorted set
Priority queue
Graphs
Lists are fundamental
Definitions
List operations
List implementation demands
Arrays as lists?
Current position
Example tasks
Iterate
Search
Insert
An actual implementation

Iterate through a list

One very common task is to iterate through a list:

```
for(L.mvToStart(); L.currPos() < L.length(); L.next())
{
   elem_value = L.getValue();
   doSomething(elem_value);
}
```

Note the use of member functions to move through the lits and check termintation; note that this function does not use private members.

List find function

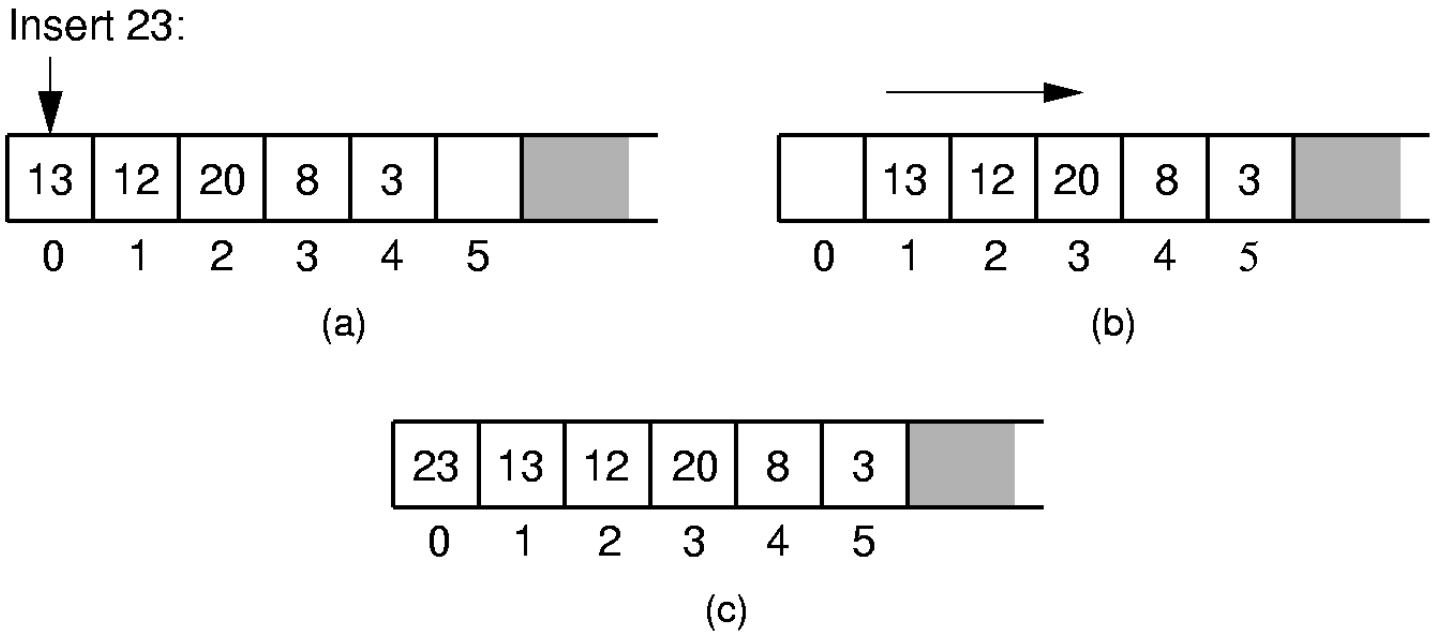

```
// return True if k is in list L, false otherwise

bool find(List<int> &L, int K)
{
   int it;
   for(L.mvToStart(); L.currPos() < L.length(); L.next())
   {
      it = L.getValue();
      if(K == it)
         return true;
   }
   return false;                    // K not found
}
```

What would a better find function return?

Can we provide more information?

Array list insert



- Requires moving elements after insert toward the tail
- How many steps does this take for a list of 4? How about 5? 6? 7?

Outline

Introduction
Abstract data types: interfaces
List
Unsorted set
Sorted set
Priority queue
Graphs
Lists are fundamental
Definitions
List operations
List implementation demands
Arrays as lists?
Current position
Example tasks
Iterate
Search
Insert
An actual implementation

"Array list" implementation

List based on arrays:

- Check out abstract class file, *list.h*
- Go over array list class file, *list_A.h*
- Check out find and print in *ListTest.h*
- Which is all called in *main_AList_easy.cpp*