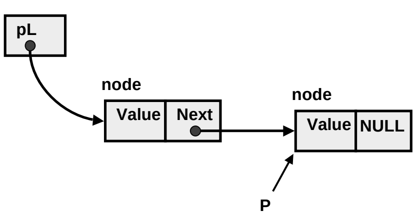
CSE 373 Lecture 5: Lists, Stacks, and Queues

- We will review:
  - More lists and applications
  - Stack ADT and applications
  - Queue ADT and applications
  - Introduction to Trees
- Covered in Chapter 3 of the text

List Operations: Run time analysis

Operation	Array-Based	Pointer-Based
isEmpty	O(1)	O(1)
Insert	O(N)	O(1)
FindPrev	O(1)	O(N)
Delete	O(N)	O(N)
Find	O(N)	O(N)
FindNext	O(1)	O(1)
First	O(1)	O(1)
Kth	O(1)	O(N)
Last	O(1)	O(N)
Length	O(1)	O(N)

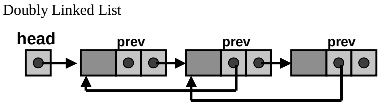
Pointer-Based Linked List



To delete the node pointed to by P, need a pointer to the previous node

Doubly Linked Lists

- FindPrev (and hence Delete) is O(N) because we cannot go to previous node
- Solution: Keep a back-pointer at each node
  - Doubly Linked List



- Advantages: Delete and FindPrev are O(1) operations
- Disadvantages:
  - More space used up (double the number of pointers at each node)
  - More book-keeping for updating the two pointers at each node

Circularly Linked Lists

- Set the pointer of the last node to first node instead of NULL
- Useful when you want to iterate through whole list starting from any node
  - No need to write special code to wrap around at the end
- Circular doubly linked lists speed up both the Delete and Last operations
  - O(1) time for both instead of O(N)

Applications of Linked Lists

- Polynomial ADT: store and manipulate single variable polynomials with non-negative exponents
  - E.g. $10X^3 + 4X^2 + 7 = 10X^3 + 4X^2 + 0X^1 + 7X^0$
  - Data structure: stores coefficients C_i and exponents i
- Array Implementation: $C[i] = C_i$
  - E.g. C[3] = 10, C[2] = 4, C[1] = 0, C[0] = 7
- ADT operations: Input polynomials in arrays A and B
  - Addition: `C[i] = ?`
  - Multiplication: `?`

Applications of Linked Lists

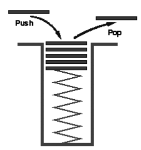
- Polynomial ADT: store and manipulate single variable polynomials with non-negative exponents
  - E.g. $10X^3 + 4X^2 + 7 = 10X^3 + 4X^2 + 0X^1 + 7X^0$
  - Data structure: stores coefficients C_i and exponents i
- Array Implementation: $C[i] = C_i$
  - E.g. C[3] = 10, C[2] = 4, C[1] = 0, C[0] = 7
- ADT operations: Input polynomials in arrays A and B
  - Addition: `C[i] = A[i] + B[i];`
  - Multiplication: `C[i+j] = C[i+j] + A[i]*B[j];`
- Problem with Array implementation: Sparse polynomials
  - E.g. $10X^{3000} + 4X^2 + 7$ → Waste of space and time (C_i are mostly 0s)
  - Use singly linked list, sorted in decreasing order of exponents

Applications of Linked Lists

- Radix Sort: Sorting integers in O(N) time
  - Bucket sort: N integers in the range 0 to B-1
    - Array Count has B elements (“buckets”), initialized to 0
    - Given input integer i, Count[i]++
    - Time: O(B+N) (= O(N) if B is Θ(N))
  - Radix sort = bucket sort on digits of integers
    - Bucket-sort from least significant to most significant digit
    - Use linked list to store numbers that are in same bucket
    - Takes O(P(B+N)) time where P = number of digits
- Multilists: Two (or more) lists combined into one
  - E.g. Students and course registrations
  - Two inter-linked circularly linked lists – one for students in course, other for courses taken by student

Stacks

- Recall: Array implementation of Lists
  - Insert and Delete take O(N) time (need to shift elements)
- What if we avoid shifting by inserting and deleting only at the end of the list?
  - Both operations take O(1) time!
- Stack: Same as list except that Insert/Delete allowed only at the *end of the list* (the top).
- "LIFO" – Last in, First out
- Push: Insert element at top
- Pop: Return and delete top element



Stack ADT

- Operations:
  - void push(Stack S, ElementType E)
  - ElementType pop(Stack S)
  - ElementType top(Stack S)
  - int isEmpty(Stack S)
  - void MakeEmpty(Stack S)
- Implementations:
  - Pointer-based: Linked list with header, S->Next points to top of stack
  - Array-based: Pre-allocate array, top is Stack[TopofStack]
- Run time: All operations are O(1) (except MakeEmpty for pointer implementation which takes Θ(N)).

Applications of Stacks I

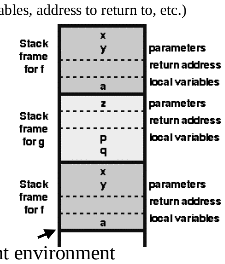
- Compilers and Word Processors: Balancing symbols
  - E.g. (i + 5*(17 – j/(6*k)) is not balanced – ")" is missing
- Balance Checker using Stacks:
  - Make an empty stack and start reading symbols
  - If input is an opening symbol, Push onto stack
  - If input is a closing symbol
    - If stack is empty, report error
    - Else, Pop the stack
      Report error if popped symbol is not corresponding open symbol
  - If EOF and stack is not empty, report error
- Run time: O(N) for N symbols

Applications of Stacks II

- Handling function calls in programming languages
  - Example: Two functions `f` and `g` calling each other: need to store current environment (input parameters, local variables, address to return to, etc.)

```
function f( int x, int y) {
   int a;
   if ( term_cond ) return …;
   a = ….;
   return g( a );
   }

function g( int z ) {
   int p, q;
   p = …. ; q = …. ;
   return f(p,q);
   }
```



Queues

- Consider a list ADT that inserts only at one end and deletes only at other end – this results in a Queue
- Queues are "FIFO" – first in, first out
- Instead of Push and Pop, we have Enqueue and Dequeue
- Why not just use stacks?
  - Items can get buried in stacks and do not appear at the top for a long time – not fair to old items.
  - A queue ensures "fairness" e.g. callers waiting on a customer hotline

Queue ADT

- Operations:
  - void Enqueue(ElementType E, Queue Q)
  - ElementType Dequeue(Queue Q)
  - int IsEmpty(Queue Q)
  - int MakeEmpty(Queue Q)
  - ElementType Front(Queue Q)
- Implementations:
  - Pointer-based is natural – what pointers do you need to keep track of for O(1) implementation of Enqueue and Dequeue?
  - Array-based: can use List operatons Insert and Delete, but O(N) time
  - How can you make array-based Enqueue and Dequeue O(1) time?

Queue ADT

- Operations:
  - void Enqueue(ElementType E, Queue Q)
  - ElementType Dequeue(Queue Q)
  - int IsEmpty(Queue Q)
  - int MakeEmpty(Queue Q)
  - ElementType Front(Queue Q)
- Implementations:
  - Pointer-based is natural – what pointers do you need to keep track of for O(1) implementation of Enqueue and Dequeue?
  - Array-based: can use List operatons Insert and Delete, but O(N) time
  - How can you make array-based Enqueue and Dequeue O(1) time?
    - Use Front and Rear indices: Rear incremented for Enqueue and Front incremented for Dequeue

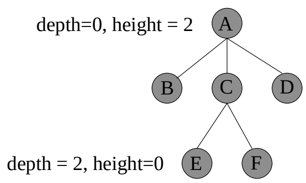
Applications of Queues

- File servers: Users needing access to their files on a shared file server machine are given access on a FIFO basis
- Printer Queue: Jobs submitted to a printer are printed in order of arrival
- Phone calls made to customer service hotlines are usually placed in a queue
- Expected wait-time of real-life queues such as customers on phone lines or ticket counters may be too hard to solve analytically → use queue ADT for simulation

Introduction to Trees

✦ Basic terminology:

- root
- leaves
- parent
- children, siblings
- path
- ancestors
- descendants
- path length
- depth / level
- height
- subtrees



Next class:

Gardening 101: Algorithms for growing, examining, and pruning trees (on your computer)

To do:

Finish Homework no. 1 (due Friday)

Finish reading Chapter 3

Start reading Chapter 4