More linked lists

September 24, 2003

What we'll cover today

- Double linked lists
- Circular lists
- Applications of linked lists

Double linked lists

- Each node contains a reference to its successor
  and its predecessor
- Advantage: can access neighbors on both sides
  - avoids the removeFromTail() problem in Monday's
    example
- Disadvantage: processing list entries is a bit more
  complicated now

NameNode2 class

```java
public class NameNode2 {
    public NameObject nameObj;
    public NameNode2 prev; // points to previous node
    public NameNode2 next; // points to next node

    public NameNode2(NameObject name) {
        this(name, null, null); // create first item in list
    }

    public NameNode2(NameObject name, NameNode n1, NameNode n2) {
        nameObj = name;
        prev = n1;
        next = n2;
    }
}
```
Adding a node to the tail

- Create and initialize a new node, \( n \)
- Set \( n.next = null; \)
- Set \( n.prev = tail; \)
- Set \( tail = n; \)
- Set \( n.prev.next = tail; \)
- (watch for case of empty linked list!)

Deleting a node from the tail

- Retrieve data object from the present tail
- Set \( tail = tail.prev; \)
- Set \( tail.next = null; \)
- (unless only one node in the list)
- Now, deleting from tail executes in constant time too (O(1))

Circular linked list

- Can be single- or double-linked list
- The last node points to the first node
  - instead of “head” and “tail” references, we keep a reference to the “tail” of the list only
- Applications: “resource-sharing” queues in networks, computer systems, OS, ...

Operations on circular lists

- Adding to tail is similar to previous examples
- Removing from tail:
  - single-linked circular list: still have to loop through the entire list to find the predecessor to the last node --> O(n) time
  - double-linked circular list: this takes O(1) time
Skip lists

- Application of a linked list
- Facilitates searching
- Different “levels” of links

Self-organizing list

- List in which entries are dynamically reordered to facilitate searching
- Alternate to the skip list idea

Example: web cache

- Stores references to web pages that a person or group of people have recently visited
- Many different algorithms:
  - Most popular: cached pages are ordered in terms of number of times accessed
  - Most recent: cached pages are ordered from most recently accessed to least recently accessed
  - Count: cached pages are ordered on the basis of how many times they've been accessed so far
  - Arbitrary: some arbitrary criteria (i.e., by site, by day) is used to order the cached pages

Web cache example

- Q: Which algorithm is the most efficient?
Which method is best for organizing a list?

- “Most popular” algorithm is the “benchmark”
  - book: *optimal static ordering*
- “Count” and “Most recent” algorithms take, at most, twice as many operations for searching as “Most popular”
  - “Arbitrary order” can be even worse than this
- For very large data sets, linked lists will be slow

Application: Sparse tables

- Sometimes, only a small amount of a dataset is needed for a particular problem
- Examples:
  - Students enrolled in a particular course is a small subset of the students enrolled in a college
  - People who live on a particular block in Manhattan are a small (tiny!) subset of the people who live in NYC
  - “Sparse matrices”: large matrices with mostly 0's

Sparse tables

- We could use an array to store data like this
  - but, we'll have many, many blank spots
  - wastes space!!
- Solution: use a linked list
  - use only enough memory to store the entries on the list
  - links between nodes --- skip over empty slots in table

Java: class LinkedList

- Double-linked list with head and tail references
- in java.util
- Contains functions to add, remove, find, etc. items in a linked list
- In this class, you are required to write your own linked lists, unless otherwise indicated
Why are linked lists important?

- Several other key data structures are implemented using linked lists
  - stacks
  - queues
  - trees
- Tradeoff: arrays are typically “faster”, but linked lists allow for more flexibility when adding, deleting, and searching for nodes.