Linked Lists

COMP2402
Carleton University
Winter 2018
LinkedLists

• Recursive Data structure
• Divides elements into individual parts called nodes
• Efficient add/remove, expensive access

• Reading:
  – ODS Chapter 3
Singly-Linked List

Theorem:

An **SLList** implements the **Stack** and **(FIFO) Queue** interfaces. An SLList supports the operations:

- **push(x)** and **pop()** in \( O(1) \) time per operation
- **add(x)** and **remove()** in \( O(1) \) time per operation.
Doubly-Linked List

• Linked list with forward and backward pointers in each node

• Use of a dummy node simplifies edge cases
Theorem:

An **DLList** implements the **List** interface. A DLLList supports the operations:

- **get(i)** and **set(i,x)** in $O(1+\min\{i,n-i\})$ time per operation
- **add(i,x)** and **remove(i)** in $O(1+\min\{i,n-i\})$ time per operation.

– Note: Ignoring the cost of **getNode(i)** all operations take $O(1)$ time per operation given a pointer to the location of interest
Theorem:

An \texttt{SELList} implements the \texttt{List} interface. Ignoring the cost of expand() and shrink(), an SELList with block size \(b\) supports the operations:

- \(\text{get}(i)/\text{set}(i,x)\) in \(O(1+\min\{i,n-i\}/b)\) time per operation,
- \(\text{add}(i,x)/\text{remove}(i)\) in \(O(b+\min\{i,n-i\}/b)\) time per operation

In addition, starting with an empty SELList, any sequence of \(m \geq 1\) add/remove operations results in \(O(bm)\) time spent on calls to \texttt{expand()} and \texttt{shrink()}.

The \textbf{space used} by an SELList that stores \(n\) elements is: \(n + O(b+n/b)\)