CSE 143

Linked Lists (cont)

[Chapter 4; Chapter 6, pp. 265-271]

Header Nodes

- Insert and Delete at the front of a linked list had to be handled as a special case (no "previous" node)
- One solution: add a header node
  - Empty list represented by header node alone
- List representation in IntSet.h

```
private:
  Node * hdr; // pointer to header node
```


Header Nodes

- Initialization in IntSet.cpp

```
IntSet::Intset( ) {
  // create empty list
  hdr = new Node;
  hdr->next = NULL;
}
```

- Remaining operations much as before

Circular Lists

- Sometimes there's no reason to distinguish a particular node as the "first" node in the list
  - Unordered data (sets)
  - Data is list of items to be processed, but order doesn't really matter (read sensors, update display)
- Idea: **`next`** field in "last" node points back to "first" node.
- Need to keep an external pointer to some node in the list.

Circular Lists

- Example: **`cp`** points to node in *non-empty* circular list of ints. Compute the sum of the node values.

```
Node * cp;  // ptr to one list Node
Node * p;   // ptr to next
            //    unprocessed Node
// process first node
int sum = cp->val;
p = cp->next;

// process remaining nodes
while (p != cp) {
  sum = sum + p->val;
  p = p->next;
}
```


Circular Lists

- Example

Doubly Linked Lists

- It's easy to move forward in a single-linked list. Moving backward is O(n) [i.e., expensive]
- If access to a previous node is needed frequently, it can be done in O(1) time at the cost of an extra pointer in each node.

```
// node for double-linked list of ints
Struct Dnode {
  int val;          // node data
  Dnode * next;     // ptr to next node
  Dnode * prev;     // ptr to previous node
}
Dnode * dlist;      // double-linked list
```


Delete (Double-Linked List)

- If **p** points to node in the middle of a double-linked list, that node can be deleted as follows

```
// delete node *p
p->prev->next = p->next;
p->next->prev = p->prev;
delete p;      // optional--delete if this
               //   node is no longer needed
```


Delete (Double-Linked List)

- If **p** points to node at the beginning or end of a list, the previous code fails (why?). Fixed version:

```
// delete node *p (even if at front or end)
// adjust previous node
if (p->prev == NULL)
  dlist = p->next;
else
  p->prev->next = p->next;
// adjust next node
if (p->next != NULL)
  p->next->prev = p->prev
// delete node (as or if needed)
delete p;
```


Delete (Double-Linked List)

- Example
- How would the code change if the list had a header node?

Insert (Doubly-Linked List)

- If **p** points to a node in the middle of a doubly-linked list, this will insert 42 in the list immediately after node **p**.

```
Dnode * q = new Dnode;
q->val = 42;
q->next = p->next;
q->prev = p;
p->next->prev = q;
p->next = q;
```

- As with delete, additional code is needed if **p** points to the first or last node in the list.

Insert (Doubly-Linked List)

- Example
- Does order of assignment statements matter?

Combinations

- These techniques can be used in various combinations
  - single vs double linked
  - circular vs non-circular
  - header vs no header node
- Choice depends on application, what operations are needed, which ones need to be fast
- New example: Circular, double-linked list with header node

Empty List (circular dll+hdr)

- An empty list is represented by a single header node that points to itself.

```
Dnode * dl = new Dnode;
dl->next = dl;
dl->prev = dl;
```


Insert (circular dll+hdr)

- Insert has no special cases
  - Processing is the same for empty/nonempty list, at front or back, etc.
- Example: insert new node containing 17 after node referred to by pointer p.

```
Dnode * p
Dnode * q = new Dnode;
q->val = 17;
q->next = p->next;
q->prev = p;
p->next->prev = q;
p->next = q;
```


Insert (circular dll+hdr)

- Example

Delete (circular dll+hdr)

- Again, no special cases
- Example: remove node referenced by **p**.

```
p->prev->next = p->next;
p->next->prev = p->prev;
delete p;      // delete if not needed
```

- Works even on an empty list(!)
- Warning: Don't **`delete p`** if **`*p`** is the header node.