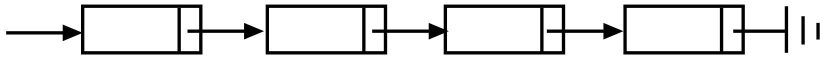
CSE 2320 Notes 8: Linked Lists

CLRS, 10.2-10.3

LINKED LISTS

1. Singly-linked (forward) lists.



Links may be:

Pointers

Subscripts

Disk addresses

Web URLs (a "logical" address vs. a "physical" address in the other three cases)

If the nodes have a key (i.e. a dictionary), should the list be ordered or unordered?

ASSUMPTION: Uniform access probabilities – equal likelihood for accessing each key

expected probes	hit	miss
unordered	$\frac{n+1}{2}$	n
ordered	$\frac{n+1}{2}$	$\frac{n+1}{2}$

Most applications have many more hits than misses.

Many applications, however, need *ordered retrieval* (SUCCESSOR, PREDECESSOR).

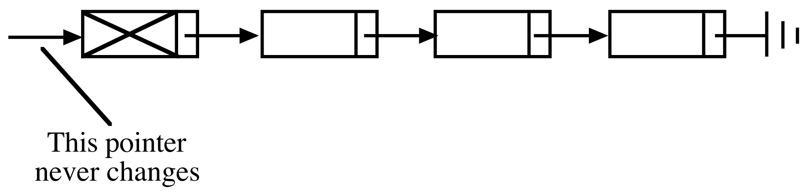
2. Keeping linked list code simple and efficient.

   a. *Header* – dummy node at beginning of list (even if no other nodes).

      Avoids “first node special” cases:

      DELETE

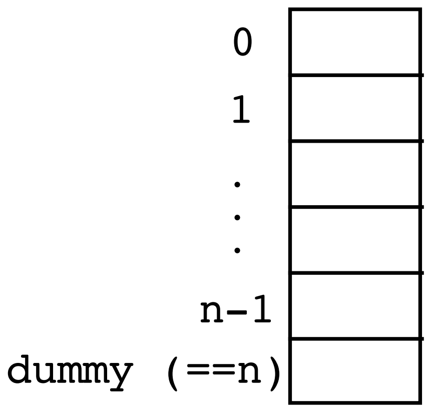
      UNION



      Can be wasteful if an application needs large number of very short lists.

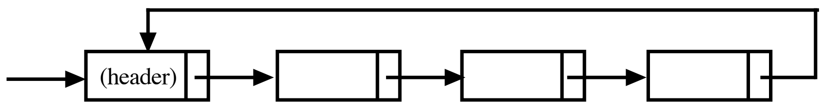
   b. *Sentinel* – dummy element at end of unordered table, unordered list, or tree. (Book uses term “sentinel” for both headers and sentinels.)

      Avoids checking for “end” of data structure.

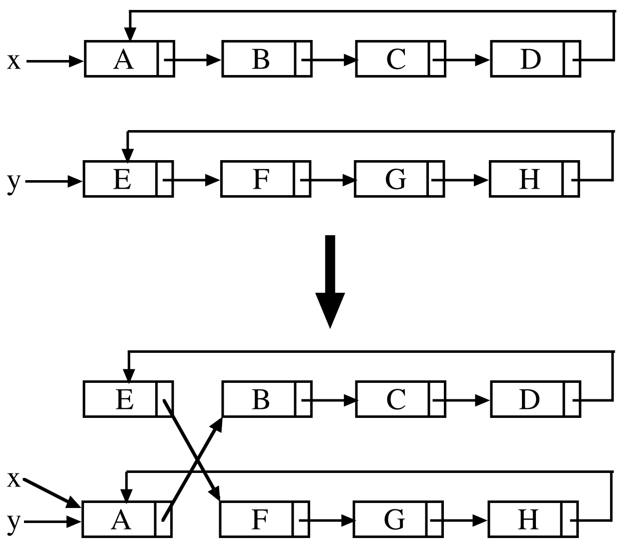


```
A[dummy] = key;
for (i = 0;
     A[i] != key;
     i++)
  ;
if (i == dummy)
  < not actually in table >
else
   < really in table >
```

3. Circular lists – can achieve $\Theta(1)$ time in special cases.



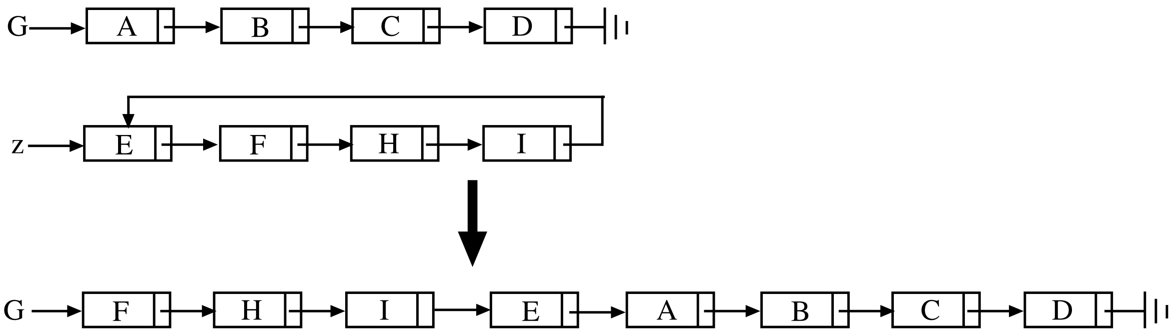
Example 1: Concatenate strings (sequences) stored as linked lists.



```
temp = (*x);
(*x) = (*y);
(*y) = temp;
x = y;
```

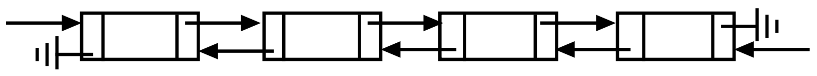
Example 2: Free storage list – avoids `malloc`/`free` overhead

Including unneeded circular list in a garbage list:

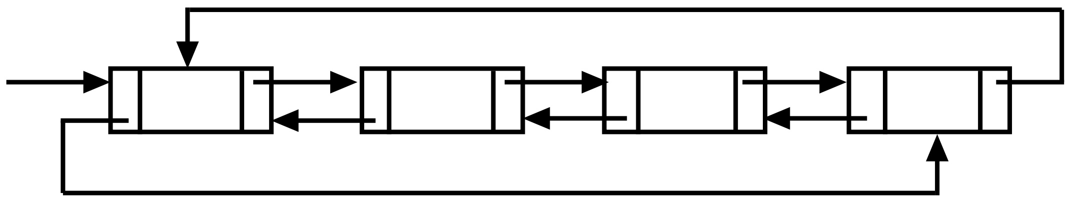


```
work = z->next;
z->next = G;
G = work;
```

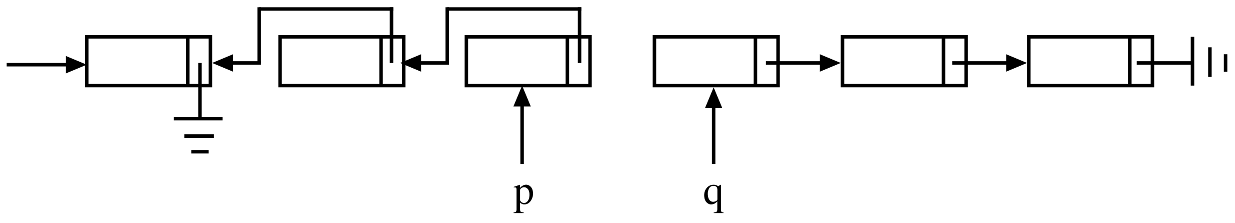
4. Doubly-linked lists.



Can also have circular doubly-linked.



Example 1: Flexibility to go both ways, but can also use the following clever solution if concurrent access is not needed:



Example 2: Student Database

- Each student record is in a number of linked lists: ethnicity, major, place-of-birth, previous colleges, etc. to allow production of reports.

- Regardless of how a record is reached, it may be necessary to remove from one list and insert in another (e.g. change of major). Trade-off:

  - If double linking is used, the predecessor is immediately available but more space is used.

  - Without double linking, the predecessor is found by traversing the list. Suitability depends on length of lists.

- Insert node that `x` points to after node that `p` points to:

```
q=p->next;
x->next=q;
x->prev=p;
p->next=x;
q->prev=x;
```

- Remove node that `x` points to:

```
p=x->prev;
q=x->next;
p->next=q;
q->prev=p;
```

Example 3: Maintain the following abstraction for n elements, 0 . . . n-1:

Specification:

- Initially all elements are *free*, but may become *allocated*.
- A particular free element may be requested and it becomes allocated. (`allocate()`)
- A particular allocated element may be requested and it becomes free. (`freeup()`)
- A request to find and allocate <u>*any*</u> free element may be made. (`allocateAny()`)
- All operations are to be supported in $O(1)$ time.

Implementation:

- An array with n+1 elements is used. Element n acts as a header for a circular, doubly-linked list.
- Element i has a *prev* value that is initially `(n+i-1)%(n+1)` and a *next* value that is initially `(i+1)%(n+1)`.
- `allocate()` is just deletion from a doubly-linked list.
- `freeup()` inserts the freed element after the header.
- `allocateAny()` deletes the successor of the header.
- Possible errors?

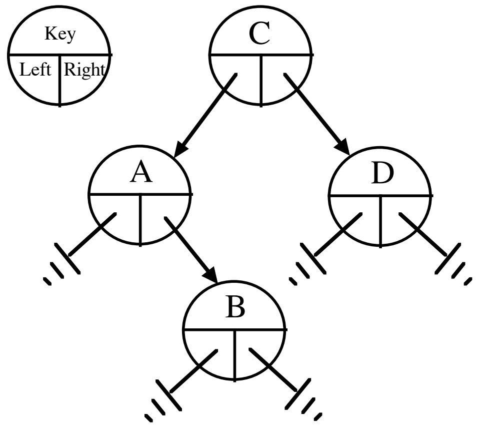
CSE 2320 Notes 9: Rooted Trees

CLRS, 10.4

TREES

Representing Trees

Binary tree



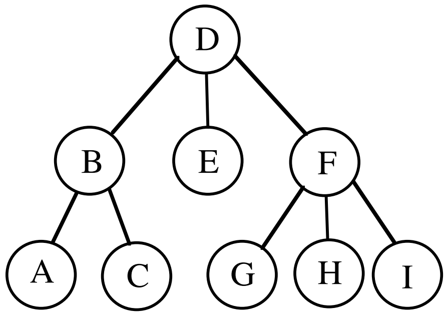
Mandatory

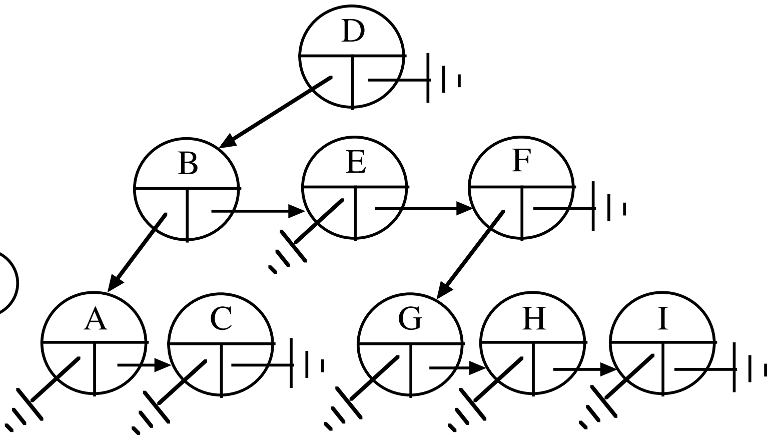
Left
Right

Optional

Parent
Key
Data

Rooted tree with linked siblings





Mandatory

First Child
Right Sibling

Optional

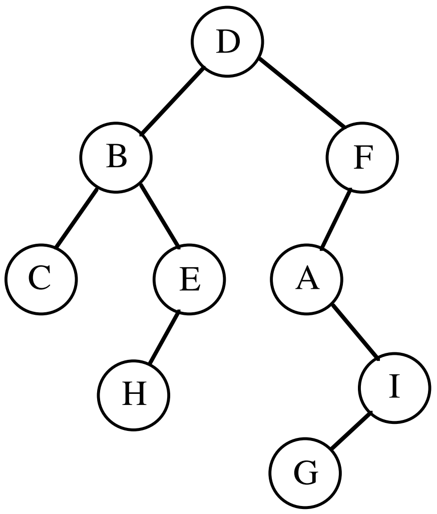
Last Child
Left Sibling
Parent
Key
Data

Binary Tree Traversals (review)

1st Visit – Preorder

2nd Visit – Inorder

3rd Visit – Postorder



Preorder

D B C E H F A I G

Inorder

C B H E D A G I F

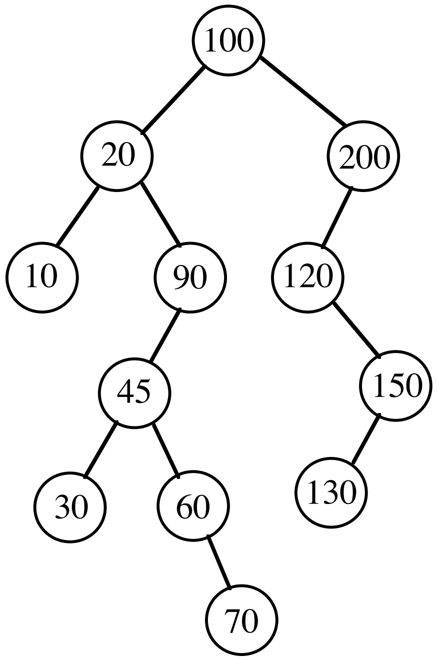
Postorder

C H E B G I A F D

Binary Search Trees

Basic property – Go left for smaller keys. Go right for larger keys.

Which traversal lists the keys in ascending order?



(Use of sentinel)

Operations:

1. Search
2. Minimum in tree
3. Maximum in tree
4. Successor of a node
5. Predecessor of a node
6. Insert
7. Deletion
    a. Leaf
    b. Node with one child
    c. Node with two children
        1. Find node's successor (convention)
        2. Successor has either
            a. Zero children – replace deleted node with successor
            b. One child (right) – point around successor; replace deleted node with successor

    May also use *tombstones* and periodically recycle garbage.

Time for operations?