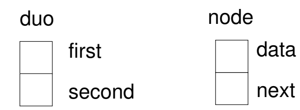
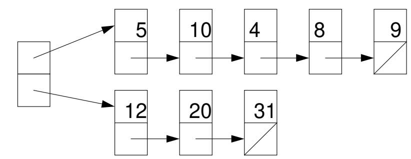
CSE 326
Lecture 2
Spring Quarter 1997

Splitting a List

- split(a,x) - returns two lists the first list has elements $\leq$ a and the second has elements > a.
- Example a = 10 x = 5, 12, 20, 10, 4, 8, 9, 31

  result 5, 10, 4, 8, 9 and 12, 20, 31





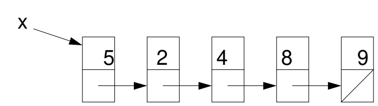
Destructive Splitting

- split(a,x) {

  local duo z;

  if x = null
     {z := newduo; z.first := null, z.second := null}
  else
     {z := split(a,x.next);
      if x.data $\leq$ a then
         {x.next := z.first; z.first := x}
      else
         {x.next := z.second; z.second := x}
  return(z)
  }

Split Example



split(6,x)

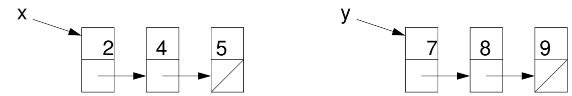
Concatenate Lists

- Example: concatenate $a_1, a_2, \ldots, a_n$ with $b_1, b_2, \ldots, b_m$ yields $a_1, a_2, \ldots, a_n, b_1, b_2, \ldots, b_m$
- concatenate(x,y){

```
local pointer to a node z;

if x = null then
   return(y)
else
   {z := concatenate(x.next,y);
    x.next := z;
    return(x)}
}
```

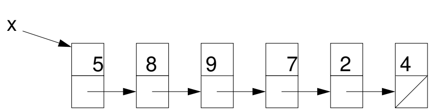

Concatenate Example



List Quicksort

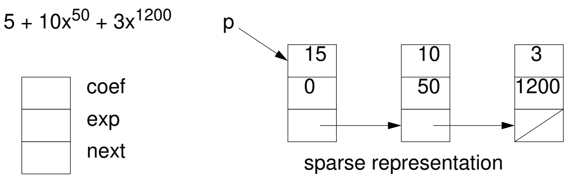
- quicksort(x){

```
local duo z;
if x = null return(null)
else {
z := split(x.data,x.next);
z.first := quicksort(z.first);
z.second := quicksort(z.second);
x.next := null;
return(concatenate(z.first, concatenate(x,z.second)))}
}
```



Sparse Polynomials

- $15 + 10x^{50} + 3x^{1200}$

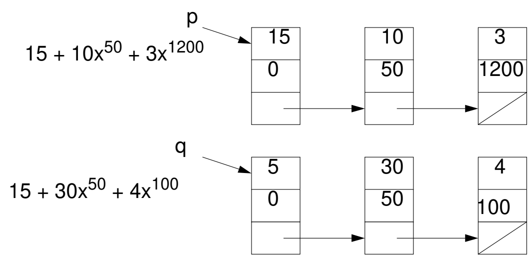


sparse representation

- A dense representation would not have the "exp" field or would be an array of coefficients.
- Sparse represention is very similar to an ordered list and is generally better than a dense represention, because less storage needed for data and less time for operations.
- In the sparse representation the 0 polynomial is represented by null.

Addition of two polynomials

- Similar to merging two sorted lists.



Recursive Addition

- Add(p,q) {

```
local pointer r;
case {
p = null:  r := q;
q = null:  r := p;
p.exp = q.exp: {r := p;
                    r.coef = p.coef + q.coef;
                    r.next := Add(p.next,q.next);
                    free(q)};
p.exp < q.exp: {r := p;
                    r.next := Add(p.next,q)};
q.exp < p.exp: {r := q;
                    r.next := Add(p,q.next)}}:
return(r)}
```


Doubly Linked List

- Uses

  - list needs to be traversed in two directions
  - deletion in the middle of a list needs to be efficient

- Example ADT - String Editing

  text string - $a_1a_2a_3...a_n$ with two positions "cursor" and "mark"

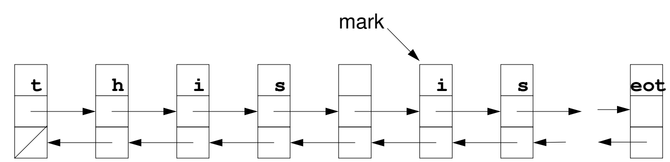
  `this is an example of a text string`

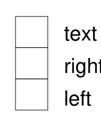
  mark cursor

- Some operations

  - Insert(c): inserts the character c just before the cursor.
  - Delete: deletes character just before cursor.
  - Long_Delete: deletes character from cursor to mark
  - Move_Left, Move_Right, Set_Mark, ...

Doubly Linked List for Text String





Example Operations

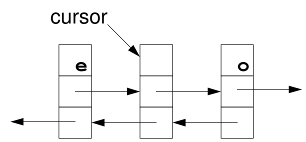
- Move_Right {

  if not(cursor.text = eot) then cursor := cursor.right}
- Move_Left {

  if not (cursor.left = null) then cursor := cursor.left}
- Insert(c) {

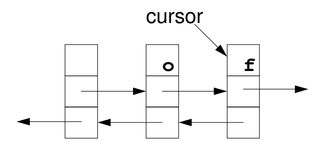
  local pointer p;

  p := new;
  p.text := c;
  p.right := cursor;
  p.left := cursor.left;
  cursor.left := p;
  if not(p.left = null) then
  p.left.right := p}



Delete Operation

- Delete - tread carefully



delete {

if not(cursor.left = null) then
{cursor.left := cursor.left.left;
cursor.left.right := cursor }
}

Long Delete

- If mark = cursor do nothing
- if mark left of cursor then delete mark to just before cursor

`this is an example of a text string`

mark cursor

- if mark right of cursor then delete cursor to just before mark, thereby moving cursor to mark.

`this is an example of a text string`

cursor mark

Testing Order of Cursor and Mark

1. Scan a pointer p from cursor to eot looking for p = mark.

   - may be slow if cursor at beginning of long text.
2. Scan two pointers from cursor, one to the left and one to the right, each one looking for mark.

   - if the cursor and mark are close to each other then this will be much faster.

`this is an example of a text string`

mark L cursor R