CSE 326
Lecture 5

Spring Quarter 1995

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"

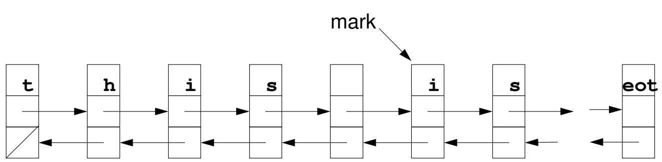
  `this is an example of a text string`

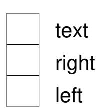
  mark cursor

- 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}

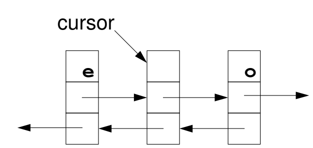
- Move_Left {

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

- Insert(c) {

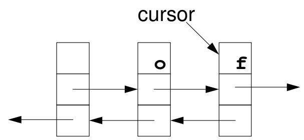
  local pointer p;

  p := new(node);
  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;
  if not(cursor.left = null) then cursor.left.right := cursor.

- If the node before the cursor is to be destroyed then a pointer must be set to it before the delete code above.

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.

`this is an example of a text string`

mark cursor p

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

Multiple Linked Lists

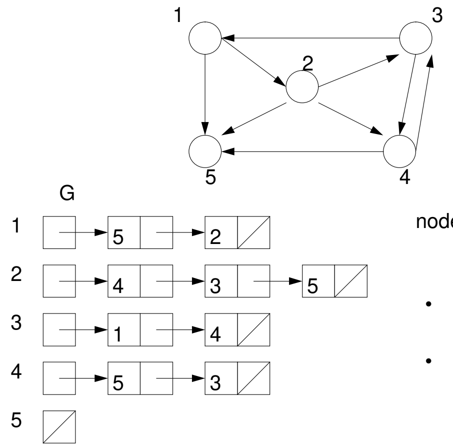
- Many ADTs such as graphs, relations, sparse matrices, radix sort, multivariate polynomials use multiple linked lists.
- There are several options

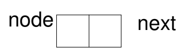
  - array of lists

  - list of lists

  - cross lists

Adjacency List Representation of Directed Graph



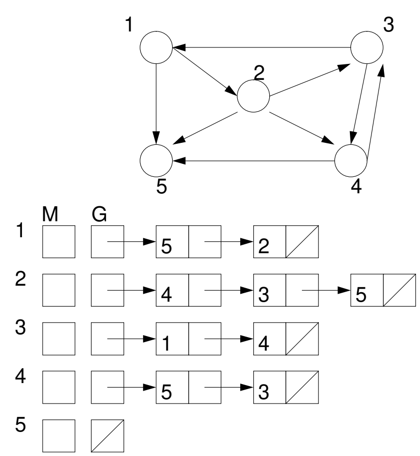


- Array G of unordered lists
- Each list entry corresponds to an edge in the graph.

Reachability by Marking

- Suppose we want to mark all the node in the graph which are reachable from a given node k. Let G[1..n] be the adjacency list representation of the graph and let M[1..n] be the mark array which is initialized to all 0.
- mark(i) {

```
 local pointer x;
M[i] := 1;
x := G[i];
while not(x=null) do {
   if M[x.node] = 0 then mark(x.node);
   x := x.next}}
```

mark(1)

Notes on Reachability

- The marking algorithm visits each node and each edge at most once.
- This marking algorithm is commonly called depth-first-search (DFS) and variants are used in solving many problems.
- Graph reachability is closely related to garbage collection.

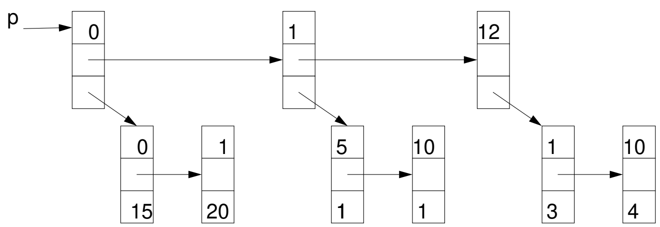
  - the nodes are blocks of memory.

  - the marked blocks are those which are reachable from a variable in the program. The unmarked blocks are garbage.

  - In real application the recursion and any stack should be eliminated.

Multivariate Polynomial as a List of Lists

- $15 + 20y + xy^{10} + xy^5 + 3x^{12} + 4x^{12}y^{10} =$
  $(15 + 20y)x^0 + (y^5 + y^{10})x^1 + (3 + 4y^{10})x^{12}$



Daily Schedule Cross List

	M	Tu	W	Th	Fri
09:30	prep		prep		prep
10:30					
11:30	326	off	326		326
12:30					
01:30		590z		fac	
02:30	off				off
03:30			590b		

Cross List