Lecture 8: Introduction to the Lists
(Array-based linked lists)
Lists
Lists

- We used successive data structures up to now:
  - If $a_{ij}$ in the memory location $L_{ij}$, then $a_{ij+1}$ is in $L_{ij}+c$ ($c$: constant)
  - In a queue, if the $i^{th}$ item is in $L_i$, $i+1$. item is in $(L_i+c)\%n$. (i.e. circular queue)
  - In a stack, if the top item is in $L_T$, the below item is in $L_{T-c}$.

Insertion and deletion: $O(1)$
Sequential Access
(ascending or descending)

Example 1:
• Alphabetically ordered lists:

<table>
<thead>
<tr>
<th>Ape</th>
<th>Butterfly</th>
<th>Cat</th>
<th>Dog</th>
<th>Mouse</th>
</tr>
</thead>
</table>

• Delete ‘Ape’, what happens?
• Delete ‘Cat’, what happens?
• Add ‘Bear’, what happens?
• Add ‘Chicken’, what happens?
Sequential Access
(ascending or descending)

Example 2:
• The result of the multiplication of two polynomials
  • $(x^7+5x^4-3x^2+4)(3x^5-2x^3+x^2+1)$

| 3 | -2 | 1 | 1 | 15 | -10 | 5 | 5 | -9 | 6 | -3 | 12 | ... | ... |
|---|----|---|---|----|-----|---|---|----|---|----|----|     |     |
| 12| 10 | 9 | 7 | 9 | 7   | 6 | 4 | 7  | 5 | 4  | 5   | ... | ... |

• Powers are not ordered. So either we need to sort or shift in order to solve this problem.
Sorted items

• We want to keep the items sorted, and we want to avoid the sorting cost.
  • We may need to sort after each insertion of a new item.
  • Or we need to do shifting.

What is the solution?
Towards the Linked List

• Each item has to have a second data field – link.
  • Each item has two fields: data and link.

Example on the board.
Linked List

#define MAX_LIST 10
#define TRUE 1
#define FALSE 0
#define NULL -1

typedef struct{
    char name[5];
    //other fields
    int link;
}item;

item linkedlist[MAX_LIST];
int free_;
void make_emptylist(void)
{
    int i;
    for(i=0;i<MAX_LIST-1;i++)
        list[i].link=i+1;  // every item points the next

    linkedlist [MAX_LIST-1].link=NULL; // last item
    free_=0;
}
Returns a free item from the list:

```c
int get_item(int* r) {
    if(free_==NULL) //there is no item to get
        return FALSE;
    else{
        *r=free_;   //get the item which is pointed by free_
        free_=linkedlist[free_].link; //points next free item
        return TRUE;
    }
}
```
Linked List
--return item

Free the item:

```c
void return_item(int r)
{
    linkedlist[r].link=free_; //return item that is pointed by r
    free_=r; //free the item
}
```
<table>
<thead>
<tr>
<th></th>
<th>name</th>
<th>link</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td>6</td>
</tr>
<tr>
<td>6</td>
<td></td>
<td>7</td>
</tr>
<tr>
<td>7</td>
<td></td>
<td>8</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td>...</td>
</tr>
</tbody>
</table>

**free_ = 0**

<table>
<thead>
<tr>
<th></th>
<th>name</th>
<th>link</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Arzu</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>Ayse</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>Aziz</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>Bora</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>Kaan</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>Muge</td>
<td>6</td>
</tr>
<tr>
<td>6</td>
<td>Ugur</td>
<td>-1</td>
</tr>
<tr>
<td>7</td>
<td></td>
<td>8</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td>...</td>
</tr>
</tbody>
</table>

**free_ = 7**

List starts at 0 (*list=0*)
<table>
<thead>
<tr>
<th></th>
<th>name</th>
<th>link</th>
</tr>
</thead>
<tbody>
<tr>
<td>[0]</td>
<td>Arzu</td>
<td>1</td>
</tr>
<tr>
<td>[1]</td>
<td>Ayse</td>
<td>2</td>
</tr>
<tr>
<td>[3]</td>
<td>Bora</td>
<td>4</td>
</tr>
<tr>
<td>[6]</td>
<td>Ugur</td>
<td>-1</td>
</tr>
<tr>
<td>[7]</td>
<td>Leyla</td>
<td>5</td>
</tr>
<tr>
<td>[8]</td>
<td>....</td>
<td>9</td>
</tr>
</tbody>
</table>

... .... ...

free_ = 8 ("Leyla" added)
*list = 0

<table>
<thead>
<tr>
<th></th>
<th>name</th>
<th>link</th>
</tr>
</thead>
<tbody>
<tr>
<td>[0]</td>
<td>Eyup</td>
<td>4</td>
</tr>
<tr>
<td>[1]</td>
<td>Ayse</td>
<td>2</td>
</tr>
<tr>
<td>[3]</td>
<td>Bora</td>
<td>0</td>
</tr>
<tr>
<td>[6]</td>
<td>Ugur</td>
<td>-1</td>
</tr>
<tr>
<td>[7]</td>
<td>Leyla</td>
<td>5</td>
</tr>
<tr>
<td>[8]</td>
<td></td>
<td>9</td>
</tr>
</tbody>
</table>

free_ = 0 ("Arzu" deleted)
free_ = 8 ("Eyup" added)
*list = 1
Linked List

--insert item
Linked List
--insert item

**INSERT at the Front**

The new item is inserted before the head of the list.
void insert_item(char name[], int* list)
{
    int r, q, p;
    if(get_item(&r)) {
        strcpy(linkedlist[r].name, name);
        q = NULL;
        p = *list;
        while(p!=NULL && strcmp(linkedlist[p].name, name)<0) { //search right position
            q = p;
            p = linkedlist[p].link;
        }
        if(q==NULL) {  //new item is inserted to the front of the list.
            *list = r;
            linkedlist[r].link = p;
        } else { //new item is inserted in the middle
            linkedlist[q].link = r;
            linkedlist[r].link = p;
        }
    } else printf("\nnot enough free space!!");
}
Linked List

--delete item

Diagram showing the deletion of the item "14" from a linked list.
Linked List
--delete item

Deleting from the front:
void delete_item(char name[], int* list)
{
    int q,p;
    q=NULL;
    p=*list;
    int l;
    while(p!=NULL && (l=strcmp(linkedlist[p].name, name))<0){  //search for the item
        q=p;
        p=linkedlist[p].link;
    }
    if(p==NULL || l>0)  //end of the list
        printf( "\n %s cannot be found!! ", name);
    else if(q==NULL){  //the first item of the list will be deleted.
        *list=linkedlist[p].link;
        return_item(p);
    }
    else{   //get the item pointed by ‘p’
        linkedlist[q].link=linkedlist[p].link;
        return_item(p);
    }
}
References

• Data Structures Notes, Mustafa Ege.