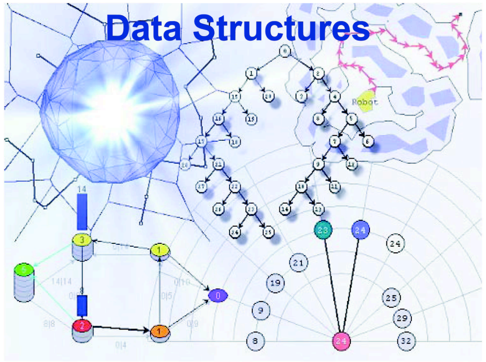
BBM 201
DATA STRUCTURES

Lecture 7:

Introduction to the Lists

(Array-based linked lists)

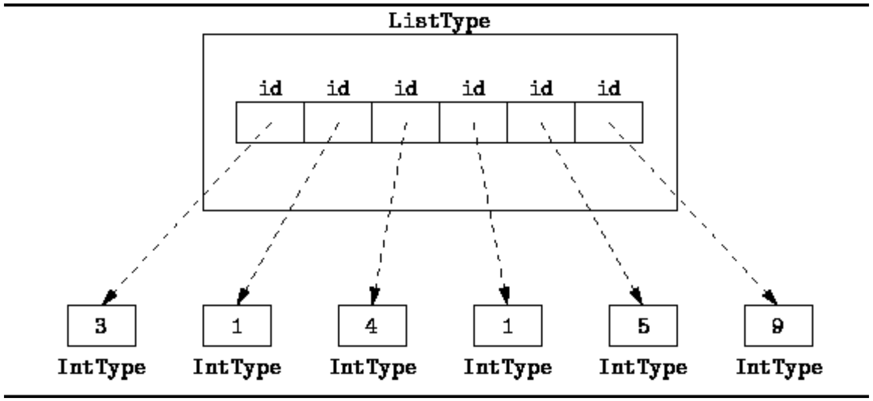


2019-2020 Fall

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 circular queue, if the i^{th} item is in L_i, (i+1)st item is in $(L_i+c)\%n$.
  - In a stack, if the top item is in L_T, the below item is in L_T-c.



Insertion and deletion: O(1)

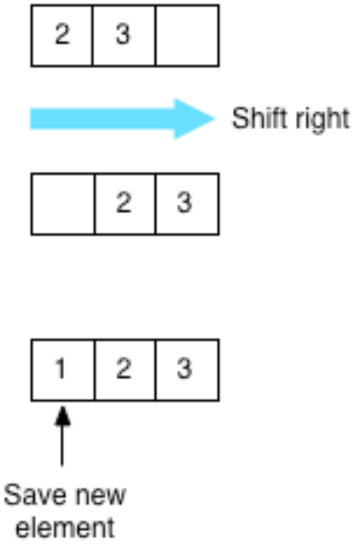
Sequential Access
(ascending or descending)

Example 1:

- Alphabetically ordered lists:

Ape	Butterfly	Cat	Dog	Mouse

- Delete 'Ape', what happens?
- Delete 'Cat', what happens?
- Add 'Bear', what happens?
- Add 'Chicken', what happens?



Sequential Access
(ascending or descending)

Example 1:

- Delete ‘Ape’

t0	Ape	Butterfly	Cat	Dog	Mouse
t1		Butterfly	Cat	Dog	Mouse
t2	Butterfly		Cat	Dog	Mouse
t3	Butterfly	Cat		Dog	Mouse
t4	Butterfly	Cat	Dog		Mouse
t5	Butterfly	Cat	Dog	Mouse	

Sequential Access
(ascending or descending)

Example 1:

- Add ‘Ant’

t0	Ape	Butterfly	Cat	Dog	Mouse	
t1	Ape	Butterfly	Cat	Dog		Mouse
t2	Ape	Butterfly	Cat		Dog	Mouse
t3	Ape	Butterfly		Cat	Dog	Mouse
t4	Ape		Butterfly	Cat	Dog	Mouse
t5		Ape	Butterfly	Cat	Dog	Mouse
t6	Ant	Ape	Butterfly	Cat	Dog	Mouse

- What if array is full?

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.

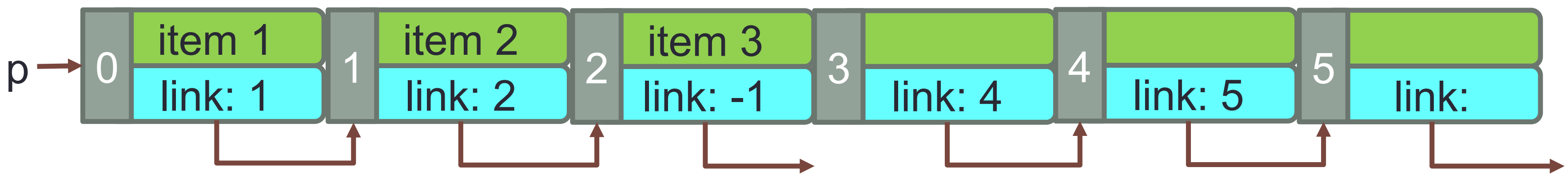
Sorted items

- 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?

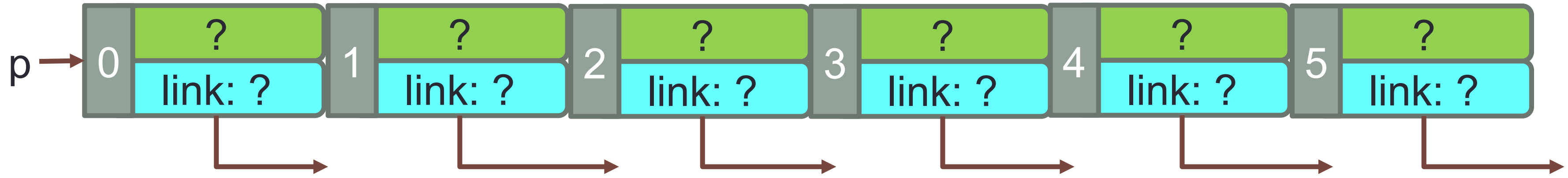
Towards the Linked List

- Idea :
  - Each data item has a link (index) to the next data item
  - Each free item slot has a link to the next free item slot
  - We remember the locations of the first item and the first free slot



first = 0
free_ = 3

Initialisation



first = ?
free_ = ?

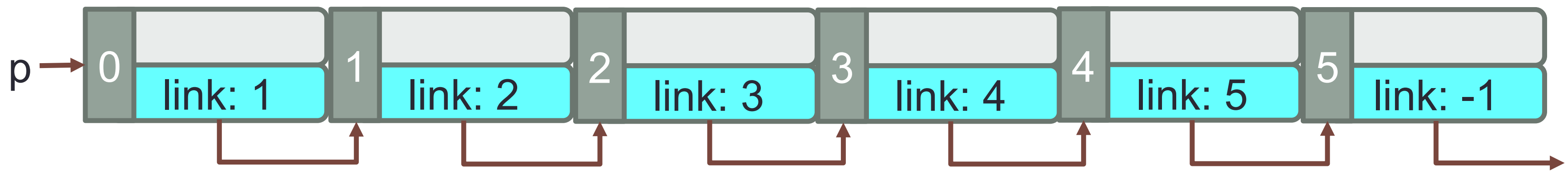
```
#include <stdio.h>
#include <string.h>

#define MAX_LIST 6
#define TRUE 1
#define FALSE 0
#define NULL -1
```

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

item linkedlist[MAX_LIST];
int first;
int free_;
```

Initialisation

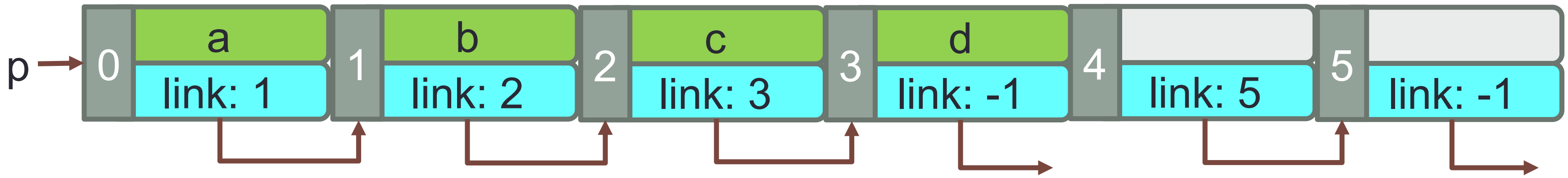


first = NULL
free_ = 0

```
void initialise()
{
    first = NULL;
    free_ = 0;
    for (int i = 0; i < MAX_LIST; i++)
        linkedlist[i].link = i + 1;

    linkedlist[MAX_LIST - 1].link = NULL;
}
```

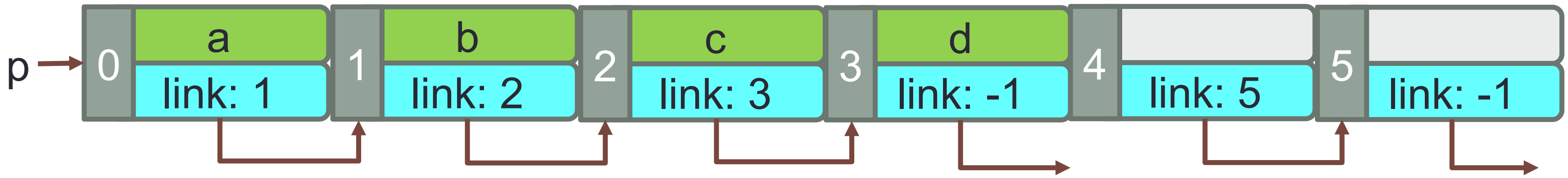
get a free item slot



first = 0
free_ = 4

```
int get_free_slot(int* slot) {
    if (free_ == NULL)  // All slots are occupied
        return FALSE;
    else      // Return the first free slot
        *slot = free_;
    return TRUE;
}
```

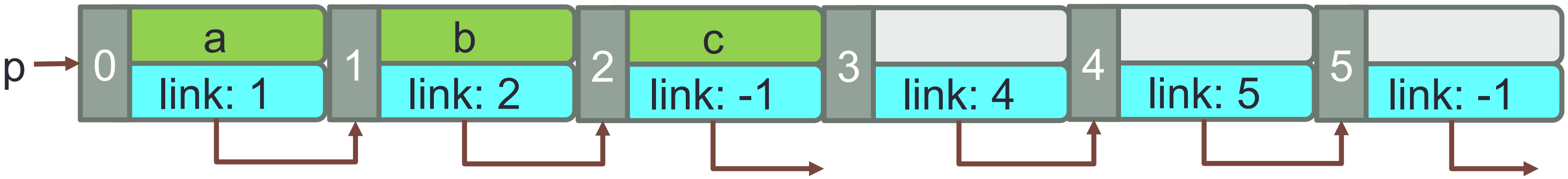
find item



first = 0
free_ = 4

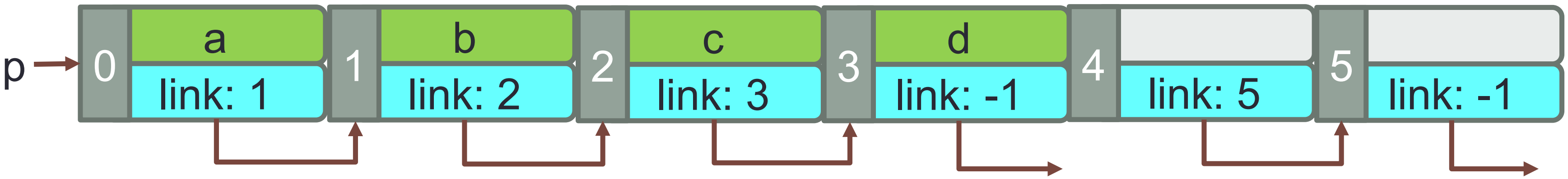
```
int find_item(char name[]) {
    if (first == NULL) // list is empty
        return NULL;
    else { // iterate over items
        for (int next = first; next != NULL; next = linkedlist[next].link)
            if (strcmp(linkedlist[next].name, name) == 0)
                return next;
    }
    return NULL;
}
```

insert an item



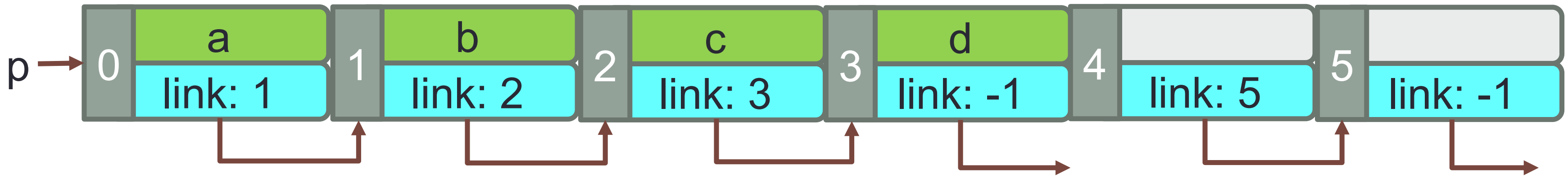
first = 0
free_ = 3

insert("d")



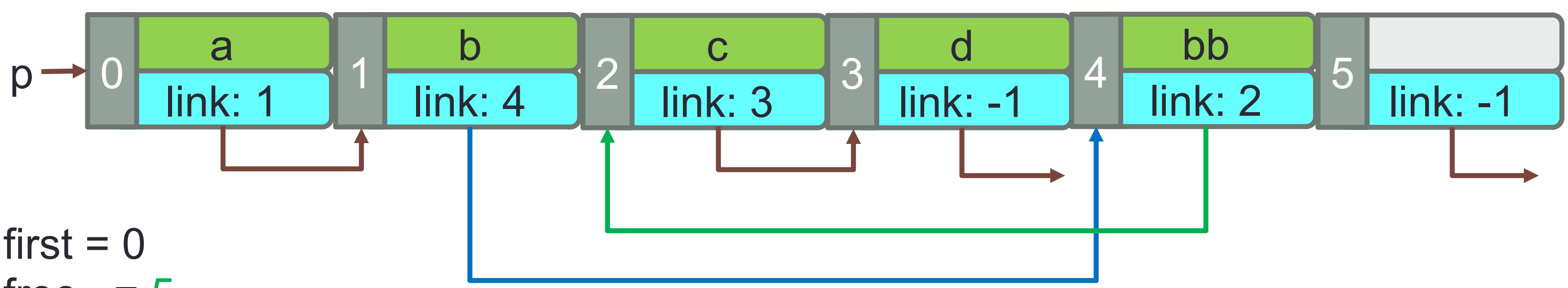
first = 0
free_ = 4

insert an item



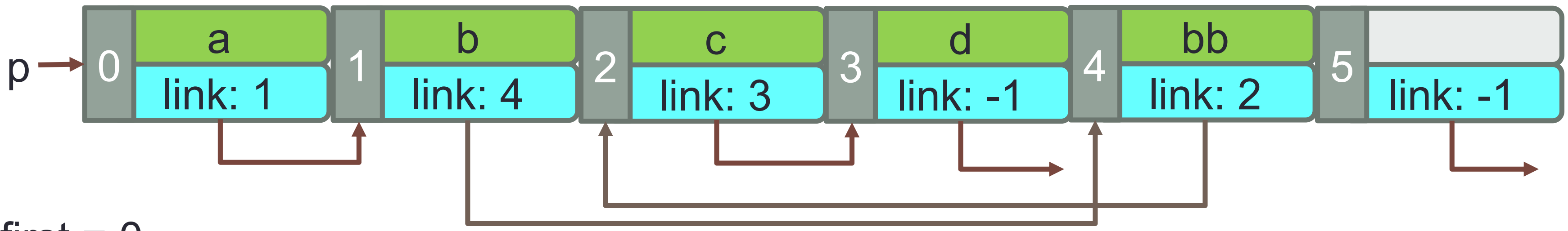
first = 0
free_ = 4

insert("bb")



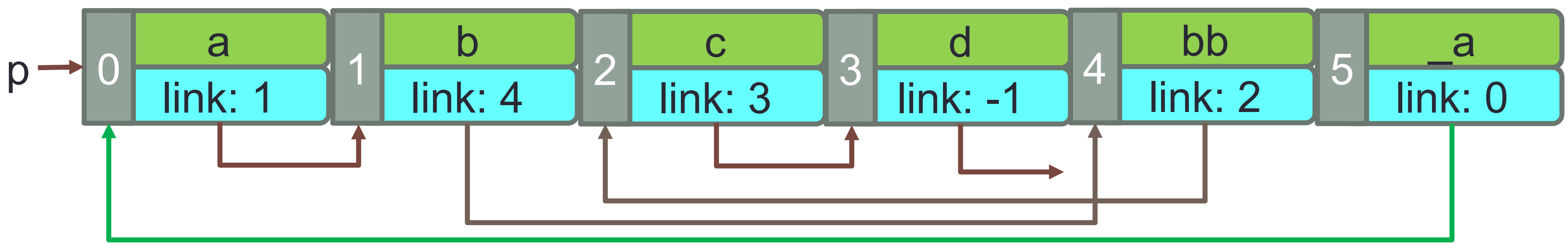
first = 0
free_ = 5

insert an item



first = 0
free_ = 5

insert("_a")



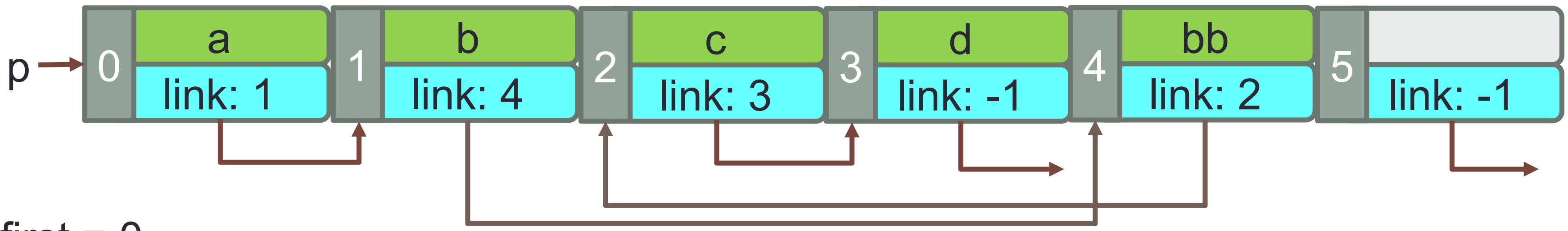
first = 5
free_ = NULL

```
int insert_item(char name[]) {
    int free_slot;
    if (get_free_slot(&free_slot)) {  // get_free_slot successful
        strcpy(linkedlist[free_slot].name, name);
        free_ = linkedlist[free_slot].link;
        int next = first;
        int prev = NULL;
        while (next != NULL && strcmp(linkedlist[next].name, name) < 0) {
            prev = next;
            next = linkedlist[next].link;
        }

        if (prev == NULL) { // Insert as the first item
            linkedlist[free_slot].link = first;
            first = free_slot;
        } else {
            linkedlist[free_slot].link = next;
            linkedlist[prev].link = free_slot;
        }

        printf("Item %s inserted. \n", name);
        return TRUE;
    }
    else {
        // No free slot exists
        printf("No free slot exists.\n");
        return FALSE;
    }
}
```

delete an item

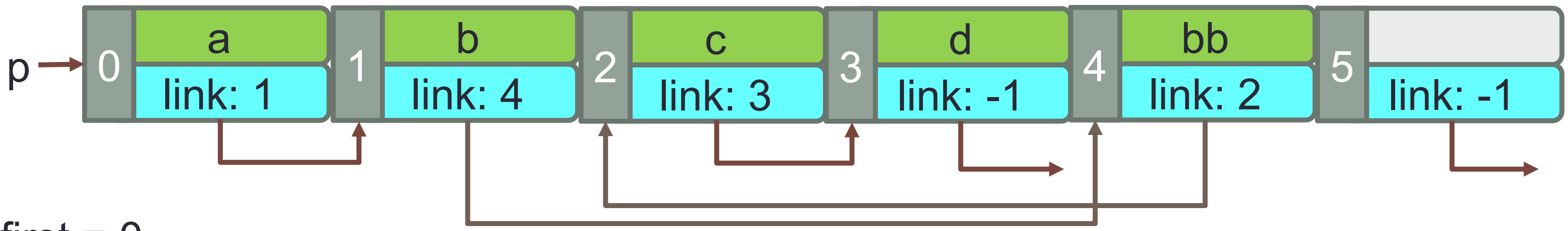


first = 0
free_ = 5

delete("e")

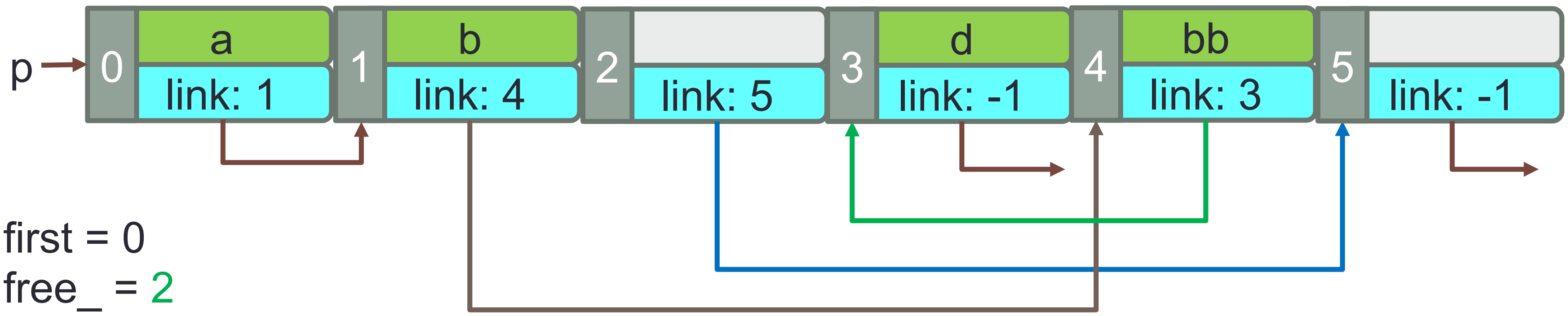
NOT FOUND!

delete an item



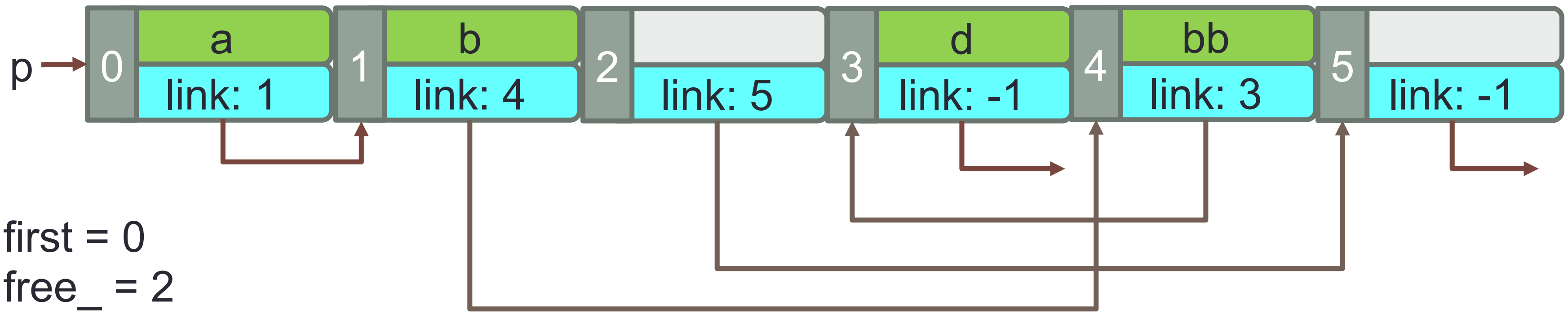
first = 0
free_ = 5

delete("c")



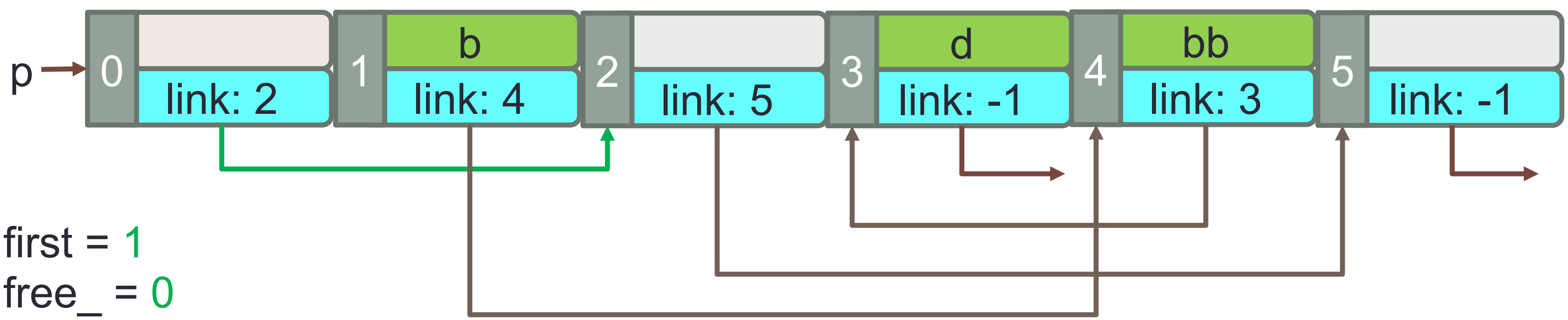
first = 0
free_ = 2

delete an item



first = 0
free_ = 2

delete("a")



first = 1
free_ = 0

```
int delete_item(char name[]) {
    int next = first;
    int prev = NULL;
    while (next != NULL && strcmp(linkedlist[next].name, name) != 0) {
        prev = next;
        next = linkedlist[next].link;
    }

    if (prev == NULL)  // Deleting the first item
        first = linkedlist[first].link;
    else if (next != NULL) // Deleting normal item
        linkedlist[prev].link = linkedlist[next].link;
    else
        return FALSE;

    linkedlist[next].link = free_;
    free_ = next;
    return TRUE;
}
```

Example Code

- https://onlinegdb.com/rJk3gW76B

References

- Data Structures Notes, Mustafa Ege.
- Fundamentals of Data Structures in C. Ellis Horowitz, Sartaj Sahni, and Susan Anderson-Freed, 1993.