Computing Laboratory

Stacks, Queues and Linked Lists

Malay Bhattacharyya

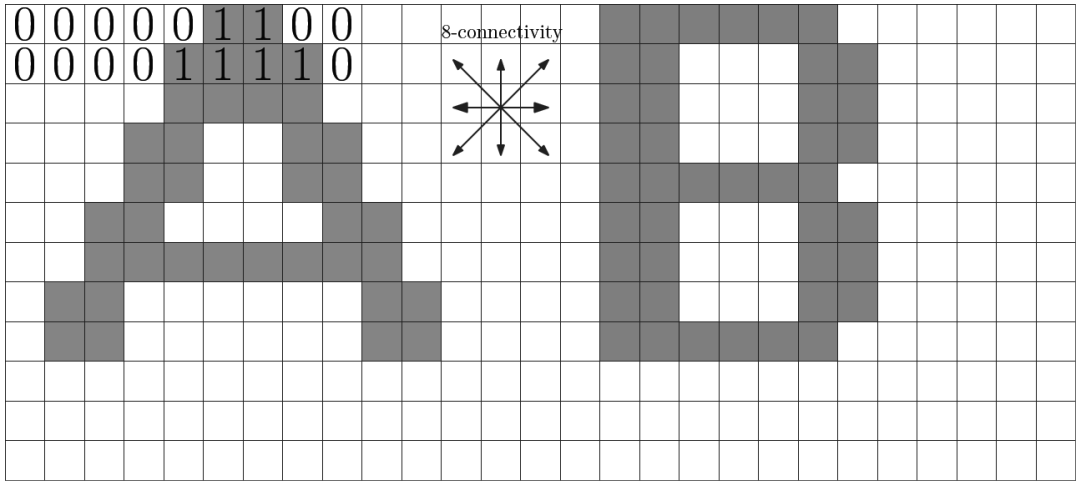
Assistant Professor

Machine Intelligence Unit
Indian Statistical Institute, Kolkata

November, 2021

Motivating example

Problem: Find the *connected components* in an image.



Data structure: Stack (LIFO)
Operations:

- `PUSH`: insert at the top
- `POP`: remove and return element from the top
- `LENGTH` or `HEIGHT`
- `ISEMPTY`, `ISFULL`

Motivating example

Problem: Maintain a list of patients waiting to consult a doctor.
Data structure: Queue (FIFO)
Operations:

- `INSERT` or `ENQUEUE`: insert at the end
- `DELETE` or `DEQUEUE`: remove and return element from the front
- `LENGTH` or `SIZE`
- `ISEMPTY`, `ISFULL`

Motivating example

Problem: Maintain a database of student information for the mentor committee: roll number, name, stream, courses taken, etc.
Data structure: Linked list (AIAO)
Operations:

- `INSERT`
  - at the beginning, end, other position (by index)
  - before / after a specific element
- `DELETE`
  - at the beginning, end, other position (by index)
  - before / after a specific element
- `LENGTH`: number of elements in the list
- `GET`: return the value of an item by index
- `SEARCH`: lookup
- `ITERATE` or `FOREACH`: to do something
- `SORT`: on a specified attribute

Traditional implementation


```
typedef struct{
    ...
}DATA;

typedef struct node{
    DATA data;
    struct node *next;
}NODE;
```

Traditional implementation

Functions:

- `create_node()`
- `insert()`
  - insert at beginning / end
  - insert in front of given node
  - insert after given node
- `delete()`
  - delete from beginning / end
  - delete given node

Traditional implementation

```
NODE *create_node(DATA d){
    NODE *nptr;
    if(NULL == (nptr = Malloc(1, NODE)))
       ERR_MESG("out of memory");
    nptr->data = d;
    nptr->next = NULL;
    return nptr;
}
```

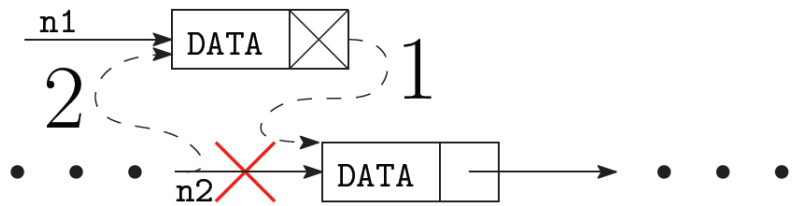
Note: Recall the definition of the following macro.

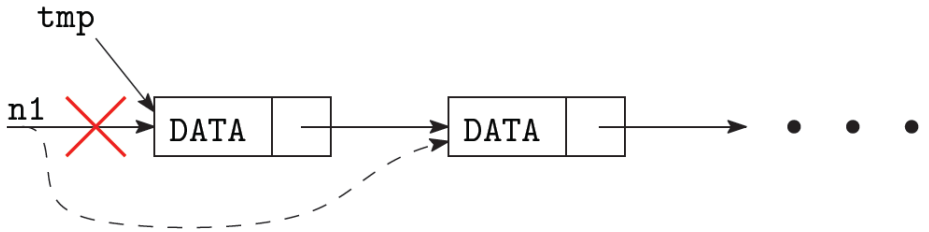
```
#define Malloc(n,type) (type *)malloc((unsigned) ((n)*sizeof(type)))
```

Traditional implementation

```
void insert(NODE *n1, NODE **n2){
    /* Insert n1 in front of n2 */
    if(n1 != NULL){
        n1->next = *n2;
        *n2 = n1;
    }
}

void delete(NODE **n1){
    NODE *tmp;
    if(n1 != NULL && *n1 != NULL){
        tmp = *n1;
        *n1 = tmp->next;
        free(tmp);
    }
}
```





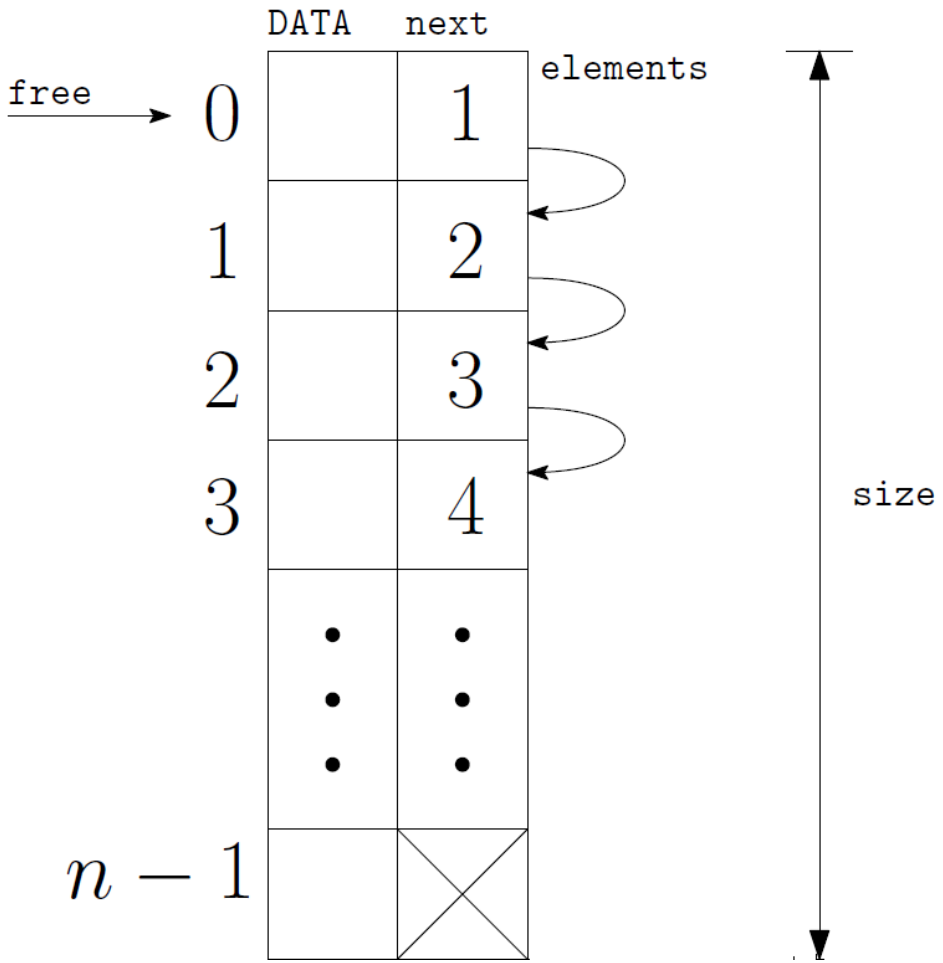
Note: Check whether boundary cases are correctly handled or not.

Alternative implementation

```
typedef struct{
    ...
}DATA;

typedef struct{
    DATA data;
    int next;
}NODE;

typedef struct{
    int head, free;
    int length, size;
    NODE *elements;
}LIST;
```



Alternative implementation

```
LIST create_list(int n){
    int i;
    LIST l;
    if(NULL == (l.elements = Malloc(n, NODE)))
        ERR_MESG("out of memory");
    for(i = 0; i < n-1; i++)
        l.elements[i].next = i+1;
    l.elements[n-1].next = -1;
    l.head = -1;     // Starting point
    l.free = 0;      // Current starting point after freeing elements
    l.length = 0;    // Current number of elements in the linked list
    l.size = n;      // Capacity of the linked list
    return l;
}
```

Alternative implementation

```
/* Insert d in front of node */
void insert(LIST *l, DATA *d, int *node){
    int position = l->free;
    if(-1 == position){
        /* No space left */
        l->size *= 2;
        if(NULL == Realloc(l->elements, l->size, NODE))
            ERR_MESG("out of memory");
        l->free = l->size/2;
        for(i = l->size/2; i < l->size - 1; i++)
            l->elements[i].next = i+1;
        l->elements[l->size - 1].next = -1;
        position = l->free;
    }
    l->free = l->elements[l->free].next;
    l->elements[position].data = *d;
    l->elements[position].next = *node;
    *node = position;
    l->length++;
    }
}
```

Alternative implementation

```
void delete(LIST *l, int node){
    int tmp;
    if(-1 != node) {
        tmp = l->elements[node].next;
        if(-1 != tmp){
            l->elements[node].next = l->elements[tmp].next;
            l->elements[tmp].next = l->free;
            l->free = tmp;
            l->length--;
        }
    }
}
```

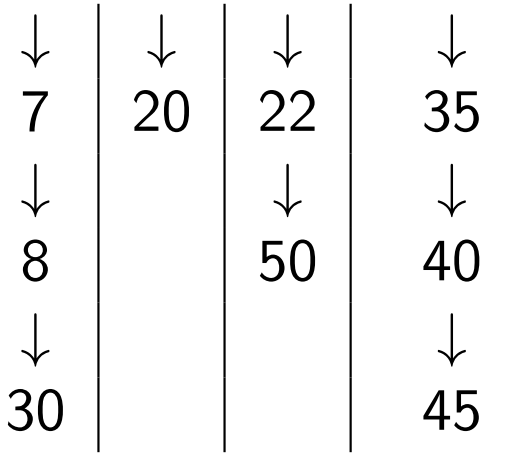
Problems – Day 9

1. Given a list of numbers (provided as command line arguments), write a program to compute the nearest larger value for the number at position i (nearness is measured in terms of the difference in array indices). For example, in the array $[1, 4, 3, 2, 5, 7]$, the nearest larger value for 4 is 5. Implement a naive, $O(n^2)$ time algorithm, as well as an $O(n)$ time algorithm for this problem. Compare the run times of your algorithms.
2. Given a set of sorted numbers as user input, write a program to remove the duplicate elements.
3. Write a program that takes a linked list and an index (starting with 0) as user inputs, and prints the alternating elements in the linked list starting from the given index.

Problems – Day 9

4 Write a program that takes a linked list of linked lists, and creates a single flattened linked list, as shown in the example below.

Input



Output

$5 \rightarrow 7 \rightarrow 8 \rightarrow 30 \rightarrow 10 \rightarrow 20 \rightarrow 19 \rightarrow 22 \rightarrow 50 \rightarrow 28 \rightarrow 35 \rightarrow 40 \rightarrow 45$

Input file format:

```
4 # Number of lists
5 7 8 30 # List 1
10 20 # List 2
19 22 50 # List 3
28 35 40 45 # List 4
```

Problems – Day 9

5 There are N petrol pumps $P_1, P_2, \ldots, P_N$ arranged in a clockwise direction along a circular road. Consider a truck with a fuel tank that is initially empty, but which has infinite capacity. The truck will initially fuel up at some P_i and move in a clockwise direction along the circular road. Each time it reaches a petrol pump, it will take up all the petrol available at that pump.

Write a program to determine the first P_i such that if the truck starts from P_i, it will be able to completely traverse the circle and return to P_i.

The amount of petrol that every petrol pump has (in litres), and the distance from that petrol pump to the next petrol pump (in km) will be given to you as command line arguments. Assume that the truck needs 1 litre of fuel to travel 1 km.

Problems – Day 9

Example: Let there be 4 petrol pumps having 4, 6, 7, and 4 litres of petrol respectively. Let the distance between these pumps be 6, 5, 3, and 5 km respectively. Then your program should output 2, since the first pump from which the truck can complete a circular tour is the 2nd petrol pump.

Also think about when such a tour will not be possible.