Listas simplemente enlazadas - Parte II
Listas doblemente enlazadas

Sofía Beatriz Pérez
Daniel Agustín Rosso

sperez@iua.edu.ar
drosso@iua.edu.ar

Centro Regional Univesitario Córdoba
Instituto Univeristario Areonáutico

Clase número 7 - Ciclo lectivo 2023

Agenda

Disclaimer

Los siguientes slides tienen el objetivo de dar soporte al dictado de la asignatura. De ninguna manera pueden sustituir los apuntes tomados en clases y/o la asistencia a las mismas.
Es importante mencionar que todos este material se encuentra en un proceso de mejora continua.
Si encuentra bugs, errores de ortografía o redacción, por favor repórtelo a sperez@iua.edu.ar y/o drosso@iua.edu.ar. También puede abrir issues en el repositorio de este link: ▸ infoI_IUA_GitLab

Listas simplemente enlazadas: implementación con funciones I

- ¿Cómo implementamos una función para agregar un nodo al comienzo de la lista?
- Se necesita modificar el contenido, es decir la dirección de memoria almacenada en head
- ¿Cómo permitimos que una función modifique el contenido de una variable puntero?

Listas simplemente enlazadas: implementación con funciones II

- Propuestas de prototipos para insertar un nodo al comienzo de la lista
  - void push(struct Node , int);
  - void push(struct Node *, int);
  - void push(struct Node **, int);
  - void push(struct Node ***, int);

Listas simplemente enlazadas: implementación con funciones III

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  struct Node {
5    int data;
6    struct Node * next;
7  };
8
9
10 void push      (struct Node **, int);
11 void append    (struct Node **, int);
12 void print_list(struct Node*);
13
14
```

Listas simplemente enlazadas: implementación con funciones IV

```
15  void append(struct Node** head, int node_data)
16  {
17    struct Node* new_node = NULL;
18    new_node = (struct Node * ) malloc(sizeof(struct Node));
19    if(new_node==NULL)
20    {
21      printf("No hay memoria disponible");
22      exit(0);
23    }
24    struct Node *temp = *head;
25    new_node->data = node_data;
26
27    new_node->next = NULL;
28
```

Listas simplemente enlazadas: implementación con funciones V

```
29      if (*head == NULL)
30      {
31          *head = new_node;
32          return;
33      }
34
35      while (temp->next != NULL)
36          temp = temp->next;
37
38      temp->next = new_node;
39  }
40
41
42
```

Listas simplemente enlazadas: implementación con funciones VI

```
43  void push(struct Node** head, int node_data)
44  {
45    struct Node* new_node = NULL;
46    new_node = (struct Node * ) malloc(sizeof(struct Node));
47    if (new_node == NULL)
48    {
49      printf("No hay memoria disponible");
50      exit(0);
51    }
52    new_node->data = node_data;
53    new_node->next = (*head);
54    (*head) = new_node;
55  }
56
```

Listas simplemente enlazadas: implementación con funciones VII

```
57
58  void print_list (struct Node *head)
59  {
60     struct Node* temp = NULL;
61     temp=head;
62     while (temp != NULL)
63     {
64        printf("%d\n", temp -> data);
65        temp = temp -> next;
66     }
67  }
68
69
70
```

Listas simplemente enlazadas: implementación con funciones VIII

```
71  void menu(void)
72  {
73     printf("1.- Agregar un nodo al final\n");
74     printf("2.- Agregar un nodo al comienzo\n");
75     printf("3.- Impresion de la lista\n");
76     printf("4.- Salir\n");
77  }
78
79
80
81
82
83
84
```

Listas simplemente enlazadas: implementación con funciones IX

```
85  int main(void)
86  {
87    int dato = dato = 0,op=0;
88    /*Puntero al comienzo de la lista*/
89    struct Node * head = NULL;
90    do {
91      menu();
92      scanf("%d", &op);
93      switch (op) {
94      case 1:
95        printf("Ingrese un dato\n");
96        scanf("%d", &dato);
97        append(&head,dato);
98      break;
```

Listas simplemente enlazadas: implementación con funciones X

```
 99        case 2:
100           printf("Ingrese un dato\n");
101           scanf("%d", &dato);
102           push(&head,dato);
103           break;
104        case 3:
105           print_list(head);
106           break;
107           }
108        }while (op != 4);
109      return (0);
110   }
```

▸ Ver ejemplo completo en gitlab

Listas doblemente enlazadas: introducción

Una lista doble es también una colección de nodos, donde cada uno contiene la parte de datos (almacenamiento útil de la lista) y dos punteros a estructuras del mismo tipo. Uno de estos punteros apunta al nodo previo mientras que el otro hará lo correspondiente con el nodo siguiente.

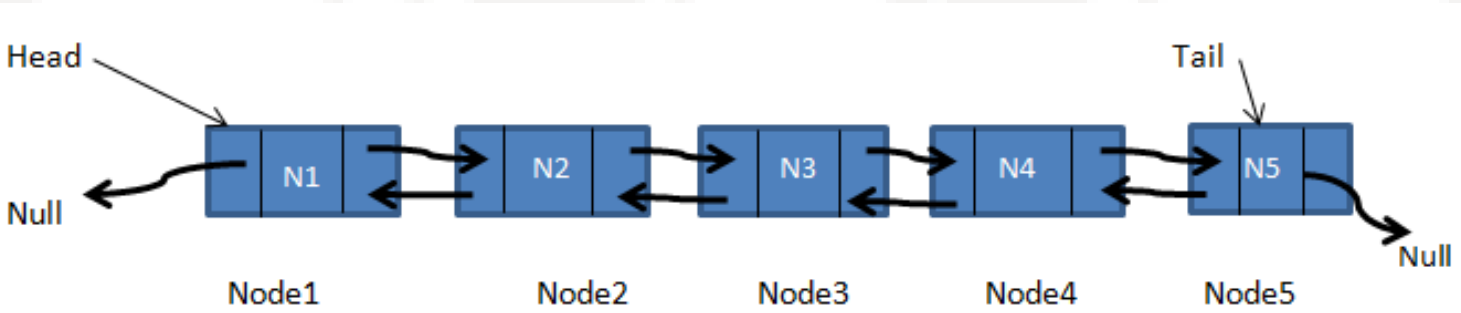

Figure: Representación gráfica de una lista doblemente enlazada.

Listas doblemente enlazadas: operaciones

- Creación de la estructura auto-referenciada
- Inserción
  - Al comienzo de la lista
  - Al final de la lista
  - Antes de un nodo en particular
  - Después de un nodo en particular
- Borrado
  - Primer nodo
  - Último nodo
  - Nodo que contenga un dato en particular
- Recorrido
  - Forward (de derecha a izquierda)
  - Backward (de izquierda a derecha)

Listas doblemente enlazadas: creación de la estructura auto-referenciada I

Cada nodo de la lista, debe contener:

- Una carga útil: por simplicidad se supondrá sólo un dato de tipo entero
- Un puntero a una estrucutura del mismo tipo apuntando al nodo anterior
- Un puntero a una estrucutura del mismo tipo apuntando al nodo siguiente

Listas doblemente enlazadas: creación de la estructura auto-referenciada II

```
1  struct Node
2  {
3      int data;
4      struct Node* next;
5      struct Node* prev;
6  };
```

Para el manejo de esta estructura desde la función principal main(), se definirá un puntero a la estructura de tipo Node:

Listas doblemente enlazadas: creación de la estructura auto-referenciada III

```
1  int main()
2  {
3  /* puntero al comienzo de la lista */
4  struct Node* head = NULL;
5
6  ...
```

Listas doblemente enlazadas: inserción de un nodo al comienzo de la lista I

1. Generar un nuevo nodo, si hay memoria suficiente, almacenar el dato en la variable correspondiente
2. Verificar si "head" es distinto de NULL, es decir, si existen nodos ya insertos en la lista:
   - Verdadero: el puntero al nodo siguiente debe asignarse al nodo que actualmente es el "head" de la lista. Es decir, al que actualmente es el primero. También debe asignarse el puntero prev del "head" al nodo recientemente creado
   - Falso: Asignar los punteros next y prev a NULL.
3. Apuntar con "head" al nuevo nodo.

Listas doblemente enlazadas: inserción de un nodo al comienzo de la lista II

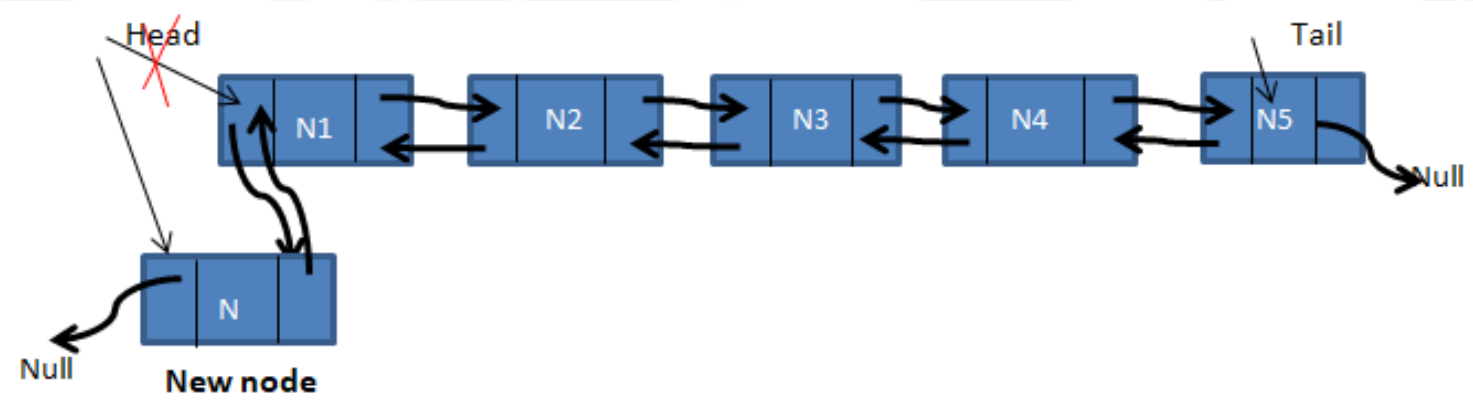

Figure: Inserción al inicio de una lista doble.

Listas doblemente enlazadas: inserción de un nodo al comienzo de la lista III

```
1  void insert_front(struct Node** head, int data)
2  {
3      struct Node* newNode = NULL;
4      try
5      {
6          newNode = new Node;
7          newNode->data = data;
8      }
9      catch (...)
10     {
11         cout << "No hay suficiente memoria";
12         exit(0);
13     }
14
```

Listas doblemente enlazadas: inserción de un nodo al comienzo de la lista IV

```
15      if ((*head) != NULL)
16      {
17          newNode->next = (*head);
18          newNode->prev = NULL;
19          (*head)->prev = newNode;
20      }
21      else
22      {
23          newNode->next = NULL;
24          newNode->prev = NULL;
25      }
26
27      (*head) = newNode;
28  }
```

Listas doblemente enlazadas: inserción de un nodo al comienzo de la lista V

Listas doblemente enlazadas: inserción al final de la lista I

1. Generar un nuevo nodo, si hay memoria suficiente, almacenar el dato en la variable correspondiente.
2. Como el nuevo nodo ingresado va al final de la lista, el puntero a "next" es asignado a NULL
3. Crear un nuevo puntero "last" que apuntará al último elemento de la lista, luego de realizar una búsqueda
4. Verificar si "head" es igual a NULL, es decir, NO existen nodos ya insertos en la lista:
   - Verdadero: el puntero a "prev" es asignado a NULL. Puesto a que el nodo ingresado es el único, head apunta a la dirección de este

Listas doblemente enlazadas: inserción al final de la lista II

- Falso: se busca cual es el último nodo de la lista y se lo apunta mediante "last". Luego, hacemos que el puntero "next" de last, apunte al nuevo nodo y el puntero "prev" del nuevo nodo a last.

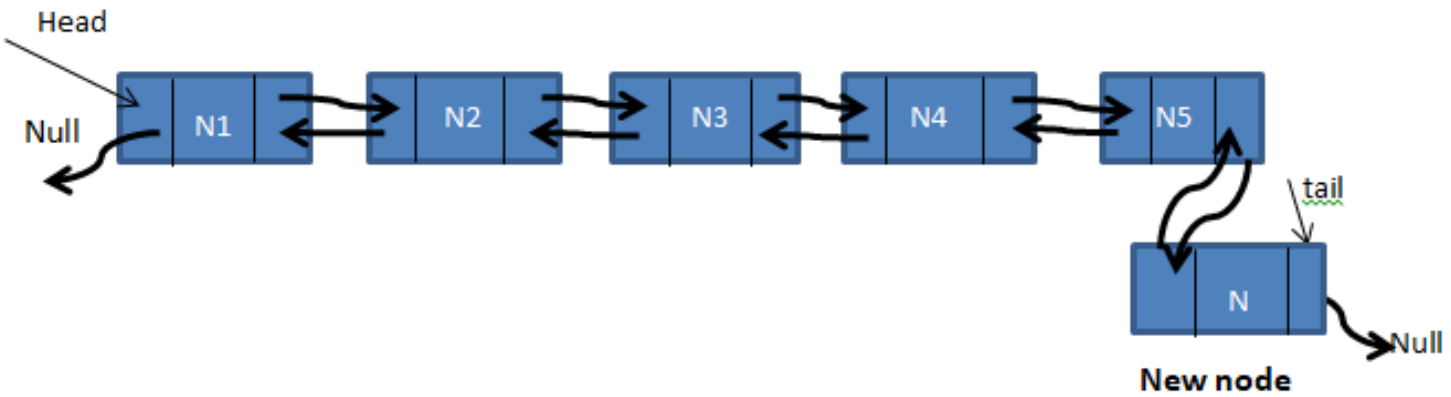

Figure: Inserción al final de una lista doble.

Listas doblemente enlazadas: inserción al final de la lista III

```
1  void insert_end(struct Node ** head, int new_data) {
2    struct Node * newNode = new Node;
3    struct Node * last = NULL;
4
5    try {
6      newNode = new Node;
7      newNode -> data = new_data;
8      newNode -> next = NULL;
9
10   } catch (...) {
11     cout << "No hay suficiente memoria";
12     exit(0);
13   }
14
15   last = * head;
```

Listas doblemente enlazadas: inserción al final de la lista IV

```
16
17      if ( *head == NULL) {
18          newNode -> prev = NULL;
19          *head = newNode;
20      } else {
21          while (last -> next != NULL)
22              last = last -> next;
23      }
24
25      last -> next = newNode;
26      newNode -> prev = last;
27
28  }
```

Listas doblemente enlazadas: inserción antes/después de un nodo en particular I

1. Verificar que el puntero al nodo recibido es valido. Si apunta a NULL, se debe abortar la estrategia
2. Generar un nuevo nodo, si hay memoria suficiente, almacenar el dato en la variable correspondiente.
3. Apuntar el puntero "next" del nuevo nodo, al "next" del nodo previo
4. Apuntar el puntero "next" del nodo previo, al nodo actual
5. Apuntar el puntero "prev" del nuevo nodo al nodo previo
6. Verificar si "next" del nuevo nodo es distinto de NULL
   - Verdadero: apuntamos "prev" del puntero siguiente al nuevo nodo a nuevo nodo

Listas doblemente enlazadas: inserción antes/después de un nodo en particular II

- Falso: lo dejamos en NULL. Es el último nodo de la lista

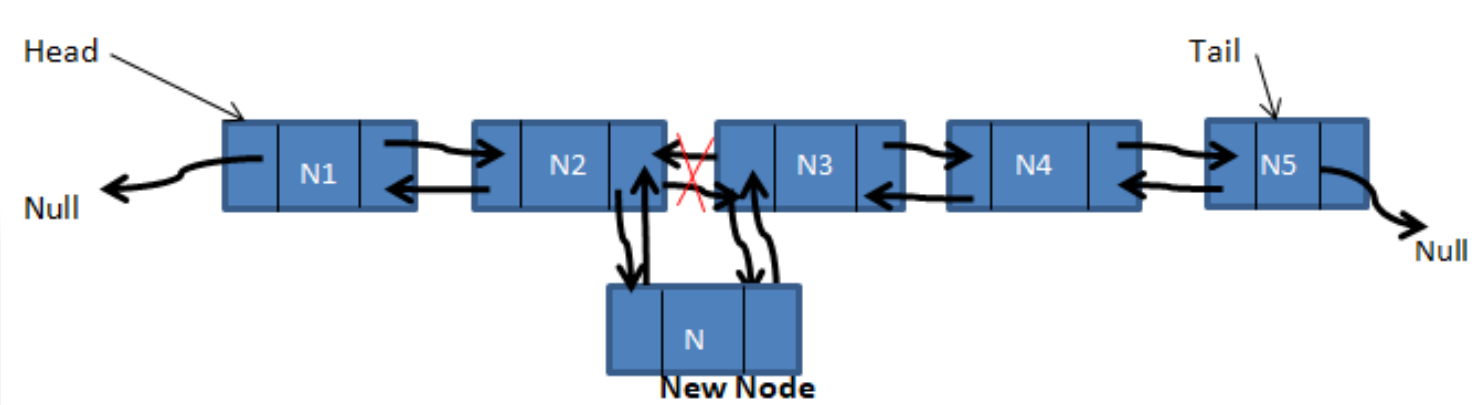

Figure: Inserción al final de una lista doble.

Listas doblemente enlazadas: inserción antes/después de un nodo en particular III

```
1  void insert_After(struct Node * prev_node, int new_d
2  {
3     struct Node * newNode = new Node;
4     newNode -> data = new_data;
5     if (prev_node == NULL)
6     {
7        cout << "Nodo previo invalido";
8        exit(0);
9     }
10
11
12
13
14
```

Listas doblemente enlazadas: inserción antes/después de un nodo en particular IV

```
15
16      try
17      {
18          newNode = new Node;
19          newNode -> data = new_data;
20      } catch (...)
21      {
22          cout << "No hay suficiente memoria";
23          exit(0);
24      }
25
26
27
28      newNode -> next = prev_node -> next;
```

Listas doblemente enlazadas: inserción antes/después de un nodo en particular V

```
29      prev_node -> next = newNode;
30      newNode -> prev = prev_node;
31
32      if (newNode -> next != NULL)
33        newNode -> next -> prev = newNode;
34  }
```

Listas doblemente enlazadas: borrado de un nodo I

- Verificar si el nodo a eliminar es el primero.
  - Verdadero: apuntar "head" a "head" next.
  - Falso:
    - Recorrer la lista hasta encontrar el nodo que se desea eliminar y almacenar un puntero a él en "del". Al hacer esto, también se debe obtener un puntero al nodo anterior al que se desea eliminar
    - Apuntar "next" del nodo previo a "next" del nodo a eliminar
    - Apuntar "del" del nodo siguiente al nodo a eliminar, a "prev" del nodo a eliminar

Listas doblemente enlazadas: borrado de un nodo II

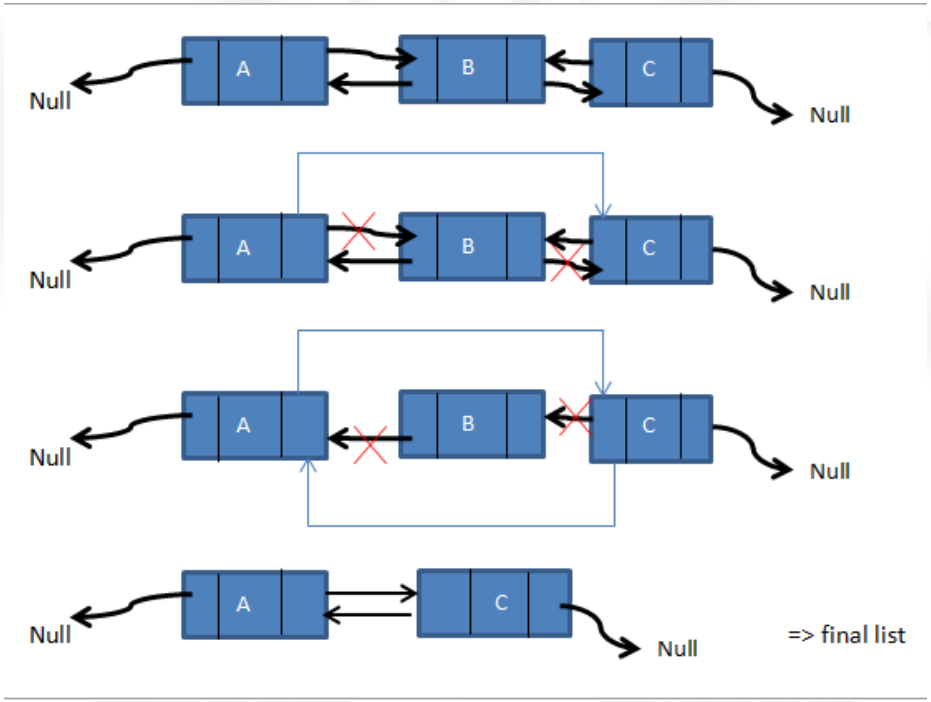

Figure: Eliminando un nodo de una lista doble.

Listas doblemente enlazadas: impresión I

```
1  void printing(struct Node* head)
2  {
3      struct Node* p = NULL;
4      struct Node* q = NULL;
5      printf("Reocorrido forward\n");
6      p=head;
7      while (p != NULL)
8      {
9          printf("%d\n",p->data);
10         q = p;
11         p = p->next;
12     }
13
14
15
```

Listas doblemente enlazadas: impresión II

```
16      printf("Reocorrido Backward\n");
17      while (q != NULL)
18      {
19          printf("%d\n",q->data);
20          q = p->prev;
21      }
22  }
```

¡Muchas gracias!

Consultas:

sperez@iua.edu.ar
drosso@iua.edu.ar