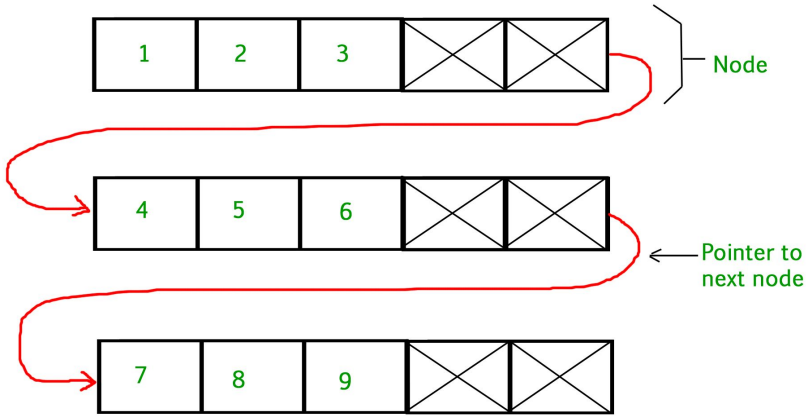
Unrolled Linked List

Integrantes:
1. Huanca Olazabal Cristhian David
2. Deza Sotomayor Fernando David

Tabla de Contenido:

1. **Why do we need unrolled linked list?**

Las simply linked list tienen se toman un tiempo de O(1) en realizar operaciones insert y delete en cualquier nodo dado. Sin embargo, este se toma O(n) en llegar al nodo indicado y proceder con la operación. En este sentido las Unrolled linked list son una gran opción ya que logran hacer la búsqueda de elementos de forma más rápida esto debido a que existen menos nodos a los que saltar además de incrementar velocidad de búsqueda debido al uso de arrays en cada nodo.



2. Advantages

La búsqueda lineal es mucho más rápida

Mejora el tiempo de búsqueda inserción y eliminación

Reduce la carga de almacenamiento al tener menos punteros

3. Disadvantages

Existe una sobrecarga mayor por cada nodo

4. How searching becomes better in unrolled linked lists?

Si tuviéramos una cantidad n de elementos que quisiéramos guardar lo que haríamos sería dividir esa cantidad en sqrt(n) en donde guardaremos una cantidad similar de elementos en cada nodo así al momento de querer acceder a un elemento específico solo buscaríamos en el nodo que contiene el elemento a diferencia de una lista enlazada normal en donde tendríamos que acceder a “n” elementos para encontrar ese nodo.

5. Costo computacional

Operación	Complejidad tiempo (Big-O)
Inserción al principio	O(1)
Inserción al final	O(1)
Inserción en posición	O(n)
Eliminación al principio	O(1)
Eliminación al final	O(1)
Eliminación en posición	O(n)
Búsqueda	O(n)
Recorrido	O(n)

6. Especificación algebraica

- Unrolledlinkedlist: L<T>, T: E, F
  - insert: L x E --> L
  - delete: L X E --> L
  - search: L X E --> Bool
  - trasversal: Q --> void
  - new: --> L
- Axiomas
  - size(new) == 0
  - size(insert(new, E))!= NULL
  - search(insert(new,E),E)==true
  - search(new,E)==false
  - delete(new,E) = new

7. Implementación.

La implementación de la estructura se hará en el lenguaje C.

Definición de la estructura de un Unrolled List:

- Estructura de cada Nodo

```
typedef struct node
{
  struct node* next;
  int* data;
  int cantidad;
} node;
```

- Estructura de la Unrolled List

```
typedef struct UnrollLinkList
{
  node* head;
  node* tail;
  int cantNodos;
  int capacidad;
} UnrollLinkList;
```

7. Implementación : Traversal

```
void traversal(UnrollLinkList* list)
{
  node* current = list -> head;
  for (int i = 0; i < list -> cantNodos; i++)
  {
    for (int j = 0; j < current -> cantidad; j++)
      printf("%d ", current -> data[j]);
    printf("\n");
    current = current -> next;
  }
}
```

```
1 2 3
4 5 6
7 8 9 10 11
```

7. Implementación : Search

```
int search(UnrollLinkList* list, int x)
{
  node* current = list -> head;
  for (int i = 0; i < list -> cantNodos; i++)
  {
    for(int j = 0; j < current -> cantidad; j++)
      if (current -> data[j] == x) return 1;
    current = current -> next;
  }
  return 0;
}
```

```
1 2 3
4 5 6
7 8 9 10 11
El elemento 10 fue encontrado
```

7. Implementación : Insert

```
void insert(UnrollLinkList* list, int x)
{
  if (!list -> head)
  {
    list -> head  = makeNewNodo(list -> capacidad);
    list -> head -> data[list -> head -> cantidad++] = x;
    list -> tail = list -> head;
    list -> cantNodos++;
    return;
  }
  if(list -> tail -> cantidad + 1 < list -> capacidad)
    list -> tail -> data[list -> tail -> cantidad++] = x;
  else
  {
   node* newNodo = makeNewNodo(list -> capacidad);
   int j = 0;
   for (int i = list -> tail -> cantidad / 2 + 1; i <list -> tail -> cantidad; i++)
     newNodo -> data[j++] = list -> tail -> data[i];
   list -> cantNodos++;
   newNodo -> data[j++] = x;
   newNodo -> cantidad = j;
   list -> tail -> cantidad = list -> tail -> cantidad / 2 + 1;
   list -> tail -> next = newNodo;
   list -> tail = newNodo;
  }
}
```

```
Insertando el valor:  1
1
Insertando el valor:  2
1 2
Insertando el valor:  3
1 2 3
Insertando el valor:  4
1 2 3 4
Insertando el valor:  5
1 2 3 4 5
Insertando el valor:  6
1 2 3
4 5 6
Insertando el valor:  7
1 2 3
4 5 6 7
Insertando el valor:  8
1 2 3
4 5 6 7 8
Insertando el valor:  9
1 2 3
4 5 6
7 8 9
Insertando el valor:  10
1 2 3
4 5 6
7 8 9 10
Insertando el valor:  11
1 2 3
4 5 6
7 8 9 10 11
```

7. Implementación : Delete

```
void delete(UnrollLinkList* list, int x)
{
  node* current = list -> head;
  while(current)
  {
    for (int i = 0; i < current -> cantidad; i++)
    {
      if (current -> data[i] == x)
      {
        pop(current, i);
        while(current -> cantidad < (list -> capacidad / 2 + 1) && current -> next)
          current -> data[current -> cantidad++] = pop(current -> next, 0);
        if(current -> next && current -> next -> cantidad < (list -> capacidad / 2 + 1))
        {
          mergeNode(current, current -> next);
          list -> cantNodos--;
        }
        return;
      }
    }
    current = current -> next;
  }
}
```

```
1 2 3
4 5 6
7 8 9 10 11
Eliminando el valor:  1
2 3 4 5 6
7 8 9 10 11
Eliminando el valor:  3
2 4 5 6
7 8 9 10 11
Eliminando el valor:  9
2 4 5 6
7 8 10 11
Eliminando el valor:  10
2 4 5 6
7 8 11
Eliminando el valor:  11
2 4 5 6
7 8
Eliminando el valor:  4
2 5 6 7 8
Eliminando el valor:  5
2 6 7 8
Eliminando el valor:  2
6 7 8
Eliminando el valor:  8
6 7
```

Pop ->

MergeNode ->

```
int pop(node* head, int pos)
{
  int tmp = head -> data[pos];
  for (int i = pos; i < head -> cantidad - 1; i++)
  {
    head -> data[i] = head -> data[i + 1];
  }
  head -> cantidad--;
  return tmp;
}
void mergeNode(node* left, node* right)
{
  for (int i = 0; i < right -> cantidad; i++)
  {
    left -> data[i + left -> cantidad] = right -> data[i];
  }
  left -> cantidad += right -> cantidad;
  left -> next = left -> next -> next;
  free(right -> data);
  free(right);
}
```

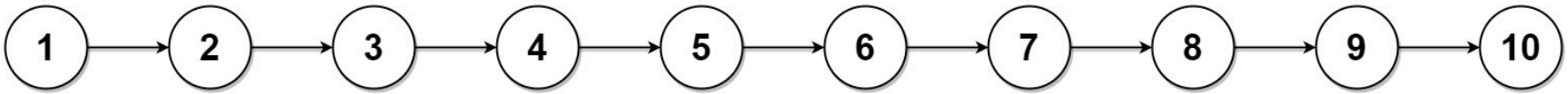
Ejercicio de Programación Competitiva

Given the head of a singly linked list and an integer k, split the linked list into k consecutive linked list parts.

The length of each part should be as equal as possible: no two parts should have a size differing by more than one. This may lead to some parts being null.

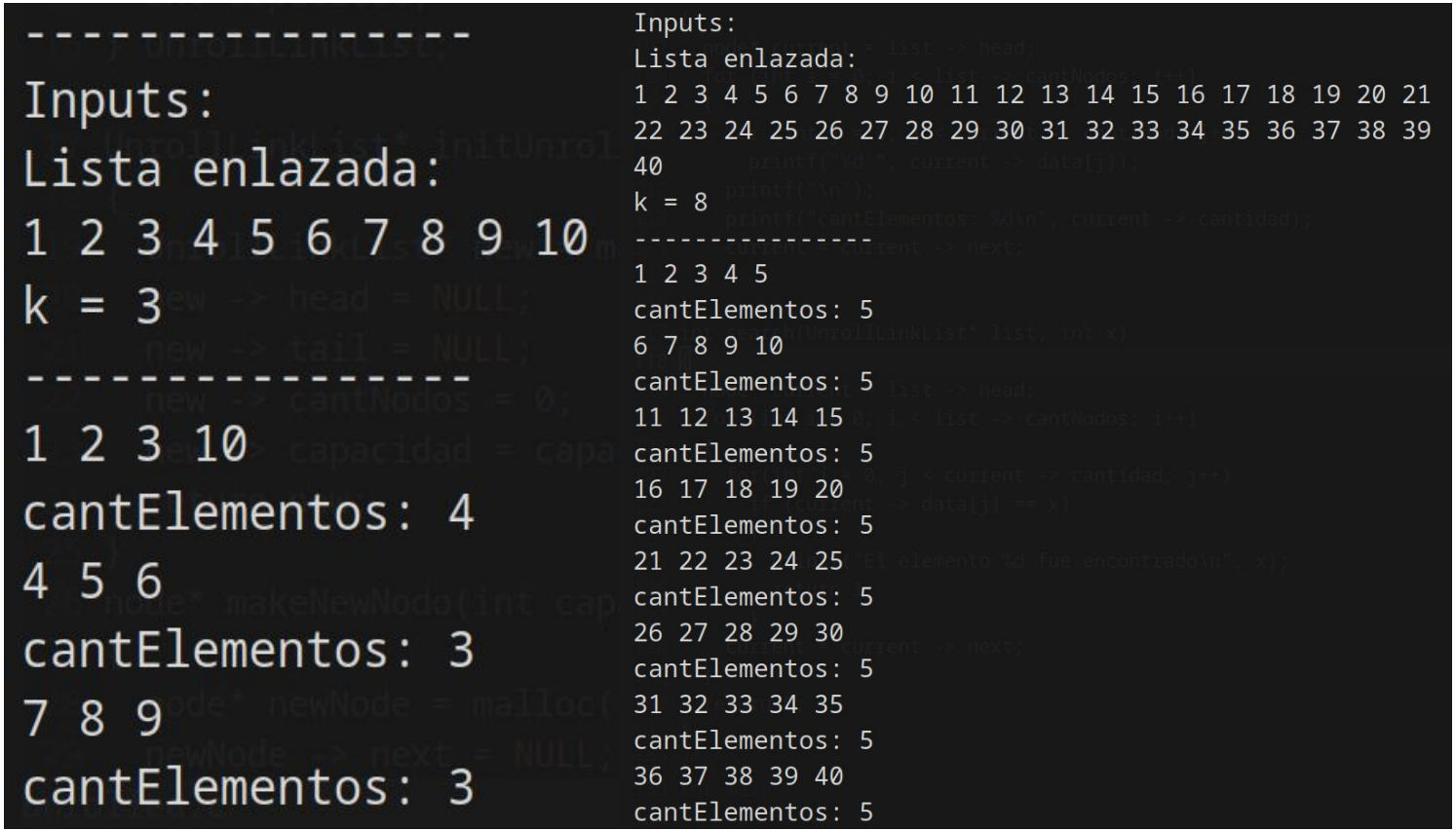
The parts should be in the order of occurrence in the input list, and parts occurring earlier should always have a size greater than or equal to parts occurring later.

Return *an array of the* k *parts*.



Input: head = [1,2,3,4,5,6,7,8,9,10], k = 3
Output: [[1,2,3,10],[5,6,7],[8,9,10]]

Input: head = [1,2,3], k = 5
Output: [[1],[2],[3],[],[]]



Inputs:
Lista enlazada:
1 2 3 4 5 6 7 8 9 10
k = 3
---
1 2 3 10
cantElementos: 4
4 5 6
cantElementos: 3
7 8 9
cantElementos: 3
Inputs:
Lista enlazada:
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21
22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39
40
k = 8
---
1 2 3 4 5
cantElementos: 5
6 7 8 9 10
cantElementos: 5
11 12 13 14 15
cantElementos: 5
16 17 18 19 20
cantElementos: 5
21 22 23 24 25
cantElementos: 5
26 27 28 29 30
cantElementos: 5
31 32 33 34 35
cantElementos: 5
36 37 38 39 40
cantElementos: 5