Taller_1

April 24, 2023

1 Ejercicios Estructuras Secuenciales - UNCode

Integrantes:

- Andrés Felipe Infante Hernández
- Manuel Camilo Rincon Blanco
- Joshua Cardona Toro
- Camilo Chitivo Cerinza

1.1 Cuestionario

1. Digan que casos de prueba fallaron, cuál creen que era la causa y cómo la solucionaron.
2. ¿Cuál les pareció el ejercicio más difícil y por qué?
3. ¿Cuál de las estructuras secuenciales le gusta más y por qué?

Deben copiar el codigo en el documento y adjuntar pantallazos de la ejecución en UNCode y en sus locales si tienen.

1.2 Solución

1.2.1 Fallos

- Imprimía un espacio al final.

Se solucionó imprimiendo todos los nodos excepto el último y este último se imprimía de la siguiente forma.

```
print('node', end='')
```

- El uso de la lista simplemente enlazada recorría toda la lista para buscar la cola, lo que causaba que el tiempo de ejecución fuera más alto.

Se solucionó utilizando una lista doblemente enlazada.

- La creación del método que indicaba si en la lista existía un solo nodo, esto, dado que no se sabía como especificar las condiciones de como eran las condiciones para que se indicara esto.

Se solucionó indicando que cuando la lista el nodo `head` apuntara a sí mismo o a `None` en el apuntador al siguiente y al anterior (en lista doblemente enlazada) indicara que eso es que solo hay un elemento en la lista.

- Optimización, en algunas pruebas el tiempo de ejecución era muy alto, y se debía optimizar.

Se solucionó evitando bucles innecesarios y evitando duplicar los datos si no era extrictamente necesario, y usando algoritmos más eficientes.

- Se imprimía dos veces el último nodo.

Fue movida la creación y adición del último nodo fuera del bucle `for`, y colocado después del bucle, de manera tal que se agregue solo una vez al final de la lista enlazada.

- Se imprimían apellidos separados por comas, y no entre espacios

Fue empleado el método `join` para suprimir la impresión entrecomillada y separada por comas de cada nodo: e.g.

```
print(' '.join(list2))
```

donde `''.join(list2)` se utiliza para unir las cadenas en la lista separadas por un espacio en blanco `' '`, es decir, cada elemento de la lista se separa por un espacio en blanco en la salida impresa.

- Complicaciones al borrar la cabeza o la cola con una función universal.

Se solucionó creando funciones especiales cuando era necesario borrar la cabeza o cola.

- Al borrar los nodos se cambiaba el puntero de los nodos de la mitad y terminaba apuntando a un nodo inexistente.

Se solucionó cambiando la manera de recorrer la lista.

1.2.2 Dificultad

El más difícil fue el segundo ejercicio dado que:

- Manejar dos listas a la vez puede ser confuso, es fácil perder la ubicación de los nodos.
- El borrar los nodos es más difícil dado que cada nodo tiene un puntero extra.
- Al borrar los nodos se cambiaba el puntero de los nodos de la mitad y terminaba apuntando a un nodo inexistente.
- Eliminaba el primer elemento común de la lista enlazada 1, pero no de la lista 2. No consideraba el primer elemento común, saltandose la impresión del `"1"` en la entrada `"6 1 3 9 2 7 ; 1 2 4 3 1 6"`. Esto fue resuelto modificando el ciclo que removía elementos para que también revise el nodo de la lista enlazada 2.

1.2.3 Preferencia

La mejor es la lista doblemente enlazada:

- Es mucho más fácil de recorrerla en ambos sentidos.
- Al añadir elementos al final de la lista no hay necesidad de recorrerla toda.
- Ocupa poca memoria a pesar de su flexibilidad.
- Se puede ajustar el puntero siguiente en función del anterior para saltar al nodo que deseamos eliminar.

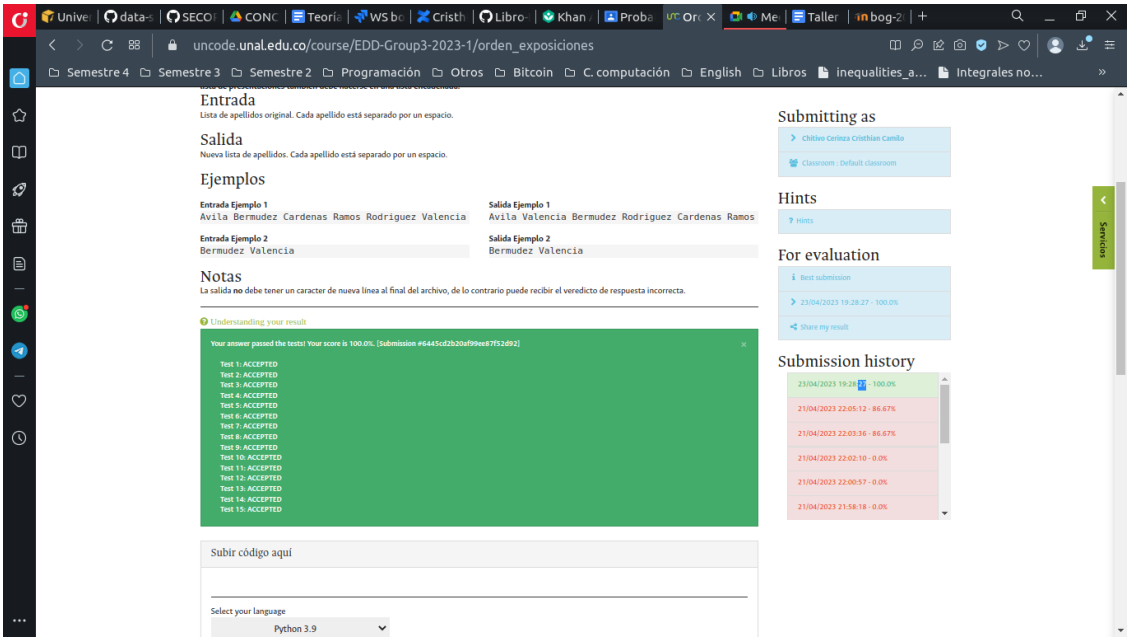
1.2.4 Códigos

Orden de Exposiciones

```
class node:
    def __init__(self, data = None, next=None, prev=None):
        self.data = data
        self.next = next
        self.prev = prev

# Creamos la clase linked_list
class linked_list:
    def __init__(self):
        self.head = None

    def is_empty(self):
        return self.head == None

    def there_is_one(self):
        if self.head == None:
            return False
        else:
            flag = ((self.head.next == None) and (self.head.prev == None)) or
                    ((self.head.next == self.head) and (self.head.prev == self.head))
            return flag

    def add_at_front(self, data):
        if self.is_empty():
            self.head = node(data=data, next=None, prev=None)
        elif self.there_is_one():
            self.head = node(data=data, next=self.head, prev=self.head)
            self.head.prev.next = self.head
            self.head.prev.prev = self.head
        else:
            self.head = node(data=data, next=self.head, prev=self.head.prev)
            self.prev.next = self.head
            self.next.prev = self.head

    def add_at_end(self, data):
        if self.is_empty():
            self.head = node(data=data, next=None, prev=None)
        elif self.there_is_one():
            #self.head.data = None
            self.head.next = node(data=data, next=self.head, prev=self.head)
            self.head.prev = self.head.next
        else:
            self.head.prev.next = node(data=data, next=self.head, prev=self.head.prev)
            self.head.prev = self.head.prev.next

    def delete_first_node(self):
        if self.is_empty():
            return
```

```
        elif self.there_is_one():
            temp = self.head
            self.head.data = None
            self.head.next = None
            self.head.prev = None
            self.head = None
            return temp.data
        else:
            self.head.next.prev = self.head.prev
            self.head.prev.next = self.head.next
            temp = self.head
            self.head = self.head.next
            if self.there_is_one():
                self.head.next = None
                self.head.prev = None
            return temp.data

    def delete_last_node(self):
        if self.is_empty():
            return
        if self.there_is_one():
            temp = str(self.head.data)
            self.head.data = None
            self.head.next = None
            self.head.prev = None
            self.head = None
            return temp
        else:
            temp = self.head.prev
            self.head.prev = self.head.prev.prev
            self.head.prev.next = self.head
            if self.there_is_one():
                self.head.next = None
                self.head.prev = None
            return temp.data

    def print_list(self):
        if self.is_empty():
            return
        elif self.there_is_one():
            print(self.head.data, end='')
        else:
            node = self.head
            stop = self.head.prev
            while node != stop:
                print(node.data, end=' ')
                node = node.next
            print(node.data, end='')
```

```
############################################################

m = input()

s = linked_list()

for i in m.split():
    s.add_at_end(i)

del m

while not s.there_is_one():

    print(s.delete_first_node(), end=' ')
    if s.there_is_one():
        break
    print(s.delete_last_node(), end=' ')

print(s.delete_last_node(), end='')
```



Elementos Comunes

```
#include <iostream>
using namespace std;
struct Node{
    int data;
    Node* next;
    Node* previous;
```

```
};

Node* createNewList(int number){
    Node* head = new Node;
    head -> data = number;
    head -> next = NULL;
    head -> previous = NULL;
    return head;
}
Node* createNewNode(Node* current, int Number){
    Node* newNode = new Node;
    newNode -> data = Number;
    current -> next = newNode;
    newNode -> previous = current;
    newNode -> next = NULL;
    return newNode;
}
int returnCurrentNode(Node* current){
    return current ->data;
}
void deleteCurrentNode(Node* current){
    current->previous->next = current->next;
    current ->next->previous = current->previous;
}
Node* deleteHead(Node* head){
    head = head->next;
    return head;
}
Node* deleteTail(Node* tail){
    tail =tail->previous;
    return tail;
}

int main(){
    int inicialista;
    cin >> inicialista;
    Node* head = createNewList(inicialista);
    Node* current = head;
    int listlength = 0;
    while(cin >> inicialista){
        current = createNewNode(current,inicialista);
        listlength++;
    }
    Node* tail = current;
    Node* currentTail = tail;
    current = head;
    listlength = listlength/2;
    for(int i = listlength;i>=0;i--){
```

```
        if(current->data == currentTail->data){
            if(current == head){
                head = deleteHead(head);
                tail = deleteTail(tail);
                current = current->next;
                currentTail = currentTail->previous;
                listlength--;
            }else{
            deleteCurrentNode(current);
            deleteCurrentNode(currentTail);
            current = current->next;
            currentTail = currentTail->previous;
            listlength--;
            }
        }else{
            current = current->next;
            currentTail = currentTail->previous;
        }
        }
    listlength*=2;
    current = head;
    for(int i = listlength;i>=0;i--){
        if(i==(listlength/2)){
           cout << current ->data;
           current = current->next;
            cout << endl;
        }else{
        cout << current ->data << " ";
        current = current->next;
        }
    }
    cout << tail->data;
    return 0;
}
```

Elementos Comunes

Enunciado del problema

Dados los elementos de dos listas encadenadas diferentes pero con la misma longitud `n` donde `2<=n<=10000`. Se deben recorrer ambas listas al tiempo: la primera lista se recorre de izquierda a derecha y la segunda de derecha a izquierda. Si en alguna iteración se encuentra que dos elementos de ambas listas son iguales, entonces ese elemento se debe eliminar de ambas listas. Por último, se deben imprimir los elementos que quedaron en cada lista.

Restricciones

- Longitud `n` de cada lista: `2<=n<=10000`
- Cada elemento de las listas está entre 0 y 9

Entrada

Elementos de las dos listas encadenadas separados por espacios. Los elementos de la primera lista se separan de los elementos de la segunda lista mediante un salto de línea.

Salida

Elementos resultantes de la primera lista (separados por espacios) seguidos de un salto de línea y luego los elementos de la segunda lista (separados por espacios).

Ejemplos

Entrada Ejemplo 1

```
6 1 3 9 2 7
1 2 4 3 1 6
```

Salida Ejemplo 1

```
9 7
1 4
```

Entrada Ejemplo 2

```
1 2 3
4 5 6
```

Salida Ejemplo 2

```
1 2 3
4 5 6
```

Notas

Importante: La solución implementada debe hacer uso de listas encadenadas..

Entendiendo tus resultados

¡Tu respuesta es exitosa! Tu calificación es 100.0%. [Envío de evaluación #6442fae148310de539ba4986]

Test 1: ACCEPTED
Test 2: ACCEPTED
Test 3: ACCEPTED
Test 4: ACCEPTED
Test 5: ACCEPTED
Test 6: ACCEPTED
Test 7: ACCEPTED
Test 8: ACCEPTED
Test 9: ACCEPTED
Test 10: ACCEPTED
Test 11: ACCEPTED
Test 12: ACCEPTED
Test 13: ACCEPTED
Test 14: ACCEPTED
Test 15: ACCEPTED

Información

Fecha de entrega	Sin fecha de envío
Estado	Tarea completada con éxito
Calificación	100.0%
Promedio ponderado	1.0
Intentos	6
Tiempo límite de envío	Sin Límite

Submitting as

Cardona Toro Joshua

Salón : Default classroom

Pistas

Pistas

Evaluación

Mejor envío

21/04/2023 16:06:41 - 100.0%

Share my result

Historial de tareas enviadas

21/04/2023 16:06:41 - 100.0%
21/04/2023 16:05:56 - 0.0%
21/04/2023 16:02:09 - 0.0%
16/04/2023 18:22:47 - 0.0%
16/04/2023 17:40:35 - 0.0%
15/04/2023 10:19:16 - 0.0%