Parcial practico 1

May 4, 2023

1 Parcial práctico 1 - UNCode

Integrantes:

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

1.1 Cuestionario

1. Adjuntar código fuente.
2. Resaltar cuales casos les dieron problemas y como se solucionaron.

1.2 Solución

1.2.1 Códigos

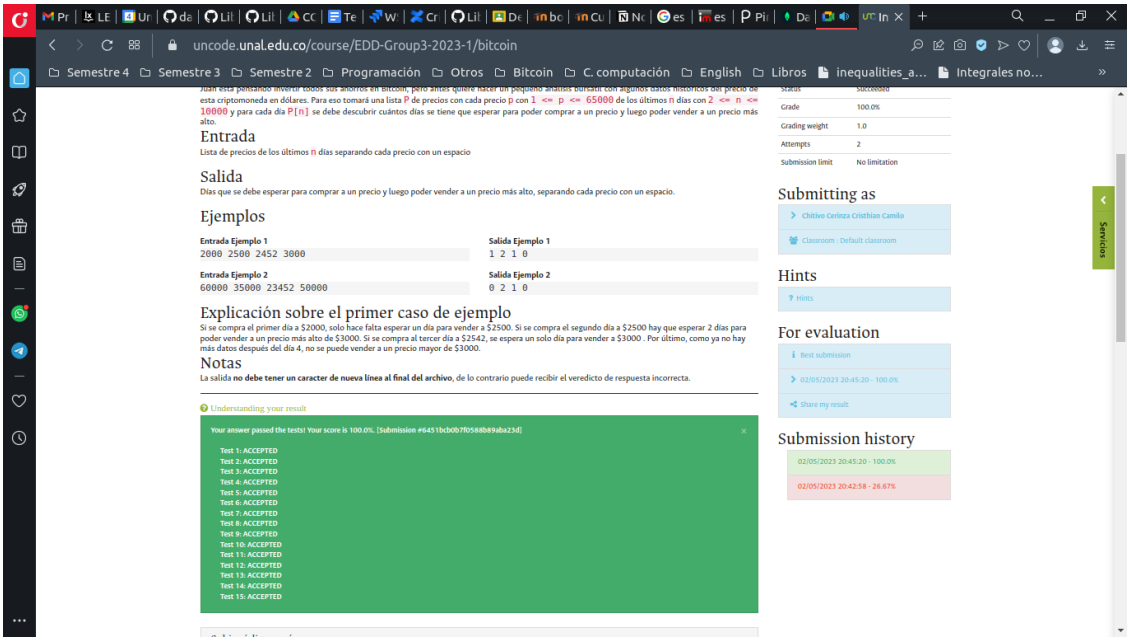
Invertir en Bitcoin

[ ]:
```
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='')

    def count_nodes(self):
        count = 0
        node = self.head
        while self.head.prev != node:
            node = node.next
            count += 1
        if self.head.next == self.head.prev:
            count += 1
        return count

    def count_days(self, large):
        count = 0
        #large = self.count_nodes()
        node = self.head.next
        flag = True
        while count < large:
```

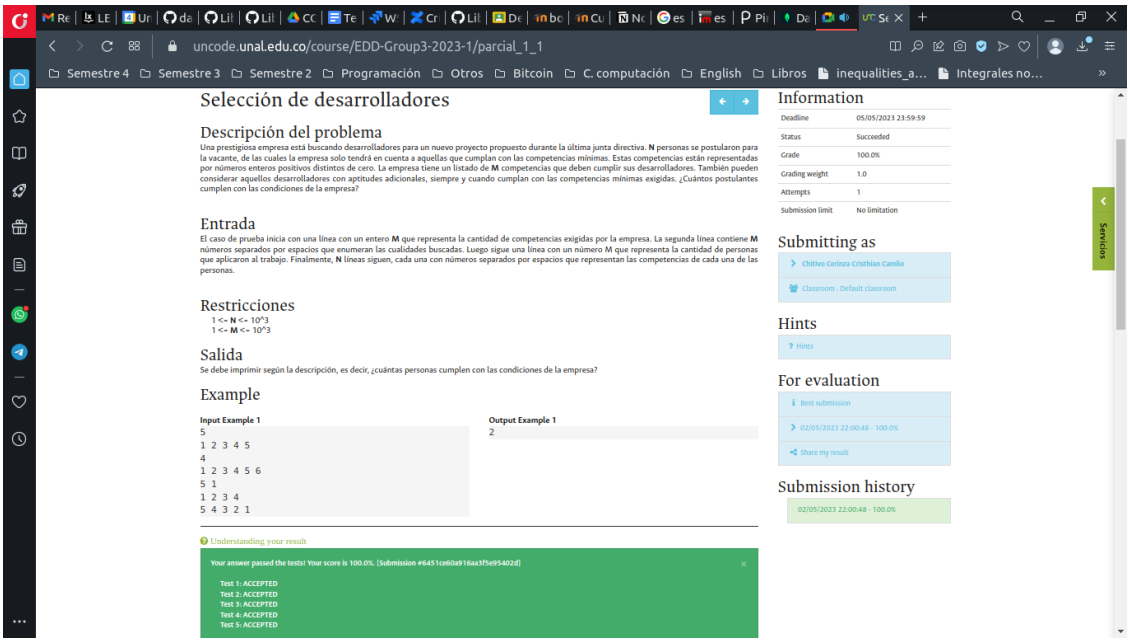
```
            if node.data > self.head.data:
                flag = False
                count += 1
                break
            count += 1
            node = node.next
        if flag:
            count = 0
        return count

############################################################

m = input()

s = linked_list()

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

t = linked_list()
large = s.count_nodes()

while not s.there_is_one():
    days = s.count_days(large)
    t.add_at_end(days)
    s.delete_first_node()
    large -= 1

t.add_at_end(0)

t.print_list()
```



Selección desarrolladores

```
[ ]: class Postulante:
    def __init__(self, competencias):
        self.competencias = competencias
        self.siguiente = None

class Empresa:
    def __init__(self, competencias_requeridas):
        self.cabeza = None
        self.competencias_requeridas = competencias_requeridas

    def insertar_postulante(self, competencias):
        nuevo_postulante = Postulante(competencias)
        if not self.cabeza:
            self.cabeza = nuevo_postulante
        else:
            actual = self.cabeza
            while actual.siguiente:
                actual = actual.siguiente
            actual.siguiente = nuevo_postulante

    def buscar_postulantes(self):
        contador = 0
        actual = self.cabeza
        while actual:
            if set(actual.competencias).issuperset(self.
 ↪competencias_requeridas):
                contador += 1
```

```
            actual = actual.siguiente
        return contador

# Lectura de la entrada
m = int(input())
competencias = list(map(int, input().split()))
n = int(input())

# Creación de la lista enlazada con los postulantes
empresa = Empresa(competencias)
for i in range(n):
    postulante = list(map(int, input().split()))
    empresa.insertar_postulante(postulante)

# Contar los postulantes que cumplen las competencias
busc_postulantes = empresa.buscar_postulantes()
print(busc_postulantes, end='')
```



1.2.2 Problemas y soluciones

En el ejecicio “Invertir Bitcoin”:

- Se requería hacer un conteo en cada iteración del tamaño de la lista y con la función que se tenía requería recorrer la lista entera para contar el número de nodos, lo cual generaba un alto costo en tiempo, por lo cual se optó por una solución secilla que fue contar una sola vez el número de nodos y ese valor asignarlo a una variable y cada vez que se eliminaba un nodo simplemente se restaba a la varible, lo cual ahorro en demasía el tiempo utilizado para resolver el ejercicio.
- Al llegar principalmente a los últimos nodos los cuales no se llegaba a un precio mayor al

comprado para poder venderlo, se debía poner en cero el número de días, lo cual se solucionó creando una variable `flag` que indicara que ya se había recorrido toda la lista y no se encontró un valor más alto, lo cual indicaba que se debía poner cero a el valor de ese nodo.

En el ejercicio "Selección de desarrolladores":

- Se presentaron inconvenientes en los datos de entrada dado que al no haberse aplicado el método `map()`, no lograba ser convertido cada elemento de la entrada en un número entero empleando el `int()`. Posterior a la implementación del método, fue posible realizar cálculos numéricos con los valores de entrada, y evaluarlos como conjuntos. También el `PRESENTATION_ERROR`, el cual fue solventado empleando `end=''` al finalizar la impresión para la supresión del espacio que pudiese imprimirse.
- El atributo `siguiente` no fue en primera instancia establecido correctamente en cada nodo, a lo cual el programa llegaba a entrar en un ciclo infinito o fallar al tratar de acceder a un atributo que no existía. El error fue solventado con el constructor de la clase `Postulante`, se inicializa el atributo siguiente con el valor `None`, y luego en el método `insertar_postulante` de la clase `Empresa`, se recorre la lista enlazada hasta encontrar el último nodo (es decir, aquel cuyo atributo siguiente es `None`), y se establece el atributo siguiente del nuevo nodo en ese último nodo encontrado. De esta manera, hemos garantizado que la lista enlazada esté correctamente enlazada y no presente ciclos infinitos.