Actividad 2: Montículos y cola de prioridad

Carmen Witsman García

Hay varios estudiantes en una escuela que esperan ser atendidos. Pueden tener lugar dos tipos de eventos con ellos, ENTER y SERVED, que se describen a continuación.

- ENTER: un alumno con alguna prioridad entra a la cola para ser atendido.
- SERVED: el estudiante con la prioridad más alta es servido (eliminado) de la cola.

Se asigna un identificador único a cada estudiante que ingresa a la cola. La cola sirve a los estudiantes según los siguientes criterios (criterios de prioridad):

- El estudiante que tenga el promedio acumulado de calificaciones más alto (CGPA) recibe el servicio primero.
- Cualquier estudiante que tenga el mismo CGPA será atendido por su nombre en orden alfabético ascendente que distingue entre mayúsculas y minúsculas.
- Todos los estudiantes que tengan el mismo CGPA y el mismo nombre serán atendidos en orden ascendente de identificación.

Implementaremos las dos siguientes clases:

- `Student`

  • El constructor con un identificador de estudiante, nombre y una CGPA

  • `get_id` - devuelve el _id del estudiante

  • `get_name` - devuelve el nombre del estudiante

  • `get_cgpa` - devuelve el CGPA del estudiante

- `PriorityQueue`
  - `process_events` - Procesa todos los eventos dados y muestra todos los estudiantes que aún no han sido atendidos en el orden de prioridad
  - `enter` - Añade un estudiante a la cola de prioridad
  - `served` - Extrae un estudiante de la cola de prioridad

Restricciones de datos:

- 0 <= CGPA <= 4.0
- 1 <= id <= 105

- 2 <= |nombre| <= 30

Formato de salida:

- Se muestran los nombres de los estudiantes que aún no se han servido por orden de prioridad con `process_events()`. Si no hay estudiantes, devuelve 'EMPTY'.

1. Clase Student

Con esta clase podremos crear estudiantes, para añadir posteriormente a la cola de prioridad.

In [7]:
```python
# Creamos la clase Student
class Student:

    def __init__(self, cgpa, name, id):
        """
        Método constructor
        """
        # Restricción de CGPA
        if (0<=float(cgpa)<=4.0) == False:
            raise Exception("La nota media acumulada debe estar comprendida entr
        self.cgpa = float(cgpa)

        # Restricción de Nombre
        if (2<=len(str(name))<=30) == False:
            raise Exception("El nombre no puede contener más de 30 caracteres, n
        self.name = str(name)

        # Restricción de ID
        if (1<=int(id)<=105) == False:
            raise Exception("El número de id debe estar comprendido entre 1 y 10
        self.id = int(id)


    def __lt__(self, other):
        """
        Establece los criterios de prioridad de los estudiantes en
        la cola de prioridad.
        """
        if self.cgpa != other.cgpa:             # Si distinto CGPA
            return self.cgpa > other.cgpa       #    Mayor CGPA = Prioridad
        elif self.cgpa == other.cgpa:           # Si mismo CGPA
            if self.name != other.name:         #    Si distinto nombre
                return self.name < other.name   #    Orden alfabético ascendente
            if self.name == other.name:         #    Si mismo nombre
                return self.id < other.id       #    Orden de ID ascendente = Pr

    def get_id(self):
        """
        Devuelve el ID del estudiante.
        """
        return self.id

    def get_name(self):
        """
```

```
        Devuelve el nombre del estudiante.
        """
        return self.name

    def get_cgpa(self):
        """
        Devuelve la nota media acumulada del estudiante.
        """
        return self.cgpa
```

1.1. Ejemplo de uso

In [23]:
```
# Definimos la clase Student como "e"
e = Student

# Creamos un nuevo estudiante
e1 = e(4.0,"Carmen", 1)

# Obtenemos sus datos
print("Nombre:", e1.get_name(), "\t  CGPA:", e1.get_cgpa(), "\tID: ", e1.get_id(
```

```
Nombre: Carmen    CGPA: 4.0     ID:  1
```

2. Clase PriorityQueue

Con esta clase podremos crear una cola de prioridad en la que los estudiantes serán añadidos en un montículo atendiendo a las prioridades especificadas en el método `__lt__` de la clase `Students`.

Para ello, utilizaremos el módulo `heapq` de Python, aplicándolo a una lista que usaremos de soporte para crear la cola de prioridad.

In [66]:
```
# Importamos el módulo heapq
from heapq import *

# Creamos la clase PriorityQueue
class PriorityQueue(Student):

    def __init__(self):
        """
        Método constructor.
        """
        self.heap = [] # Crea lista vacía (heap)

    def enter(self, student):
        """
        Introduce estudiantes en la cola y lo devuelve.
        """
        # Excepción si no se introduce estudiantes de la clase Student
        if not isinstance(student, Student):
            raise Exception("Debes introducir un estudiante de la clase Student"

        # Usamos la función heappush para introducir los estudiantes
        # atendiendo a los criterios de prioridad
        heappush(self.heap, student)
        s = student
```

```
        return [s.get_name(), s.get_cgpa(), s.get_id()]

    def served(self):
        """
        Elimina de la cola al estudiante de mayor prioridad y lo devuelve.
        """
        h = heappop(self.heap)
        h
        return [h.get_name(), h.get_cgpa(), h.get_id()]

    def process_events(self):
        """
        Devuelve los estudiantes de la cola en orden de prioridad.
        """
        # La cola no está vacía
        if self.heap != []:
            for estudiante in self.heap:
                print("Nombre: ", estudiante.name, # Muestra los estudiantes en
                      "\tCGPA: ", estudiante.cgpa,
                      "\tID: ", estudiante.id)
        # La cola está vacía
        else:
            return "EMPTY"

    def clear(self):
        """
        Limpia la cola.
        """
        return self.heap.clear()
```

2.1. Ejemplo de uso

In [68]:

```
# Definimos las clases
pq = PriorityQueue()
s = Student

# Estudiantes nuevos
e1 = s(2.1, "Joaquina", 1)
e2 = s(4.0, "Carmen", 2)
e3 = s(3.5, "Zendaya", 3)
e4 = s(3.5, "Aníbal", 4)
e5 = s(2.6, "Eugenio", 5)
e6 = s(2.6, "Eugenio", 6)
e7 = s(1.1, "Piplup", 7)

Estudiantes = [e1, e2, e3, e4, e5, e6, e7]

# Se introducen a la cola
for estudiante in Estudiantes:
    pq.enter(estudiante)

# Vemos la cola
pq.process_events()

# Eliminamos elemento prioritario
print("\nSe elimina: ", pq.served())

# Limpiamos la cola
pq.clear()
```

```
# Mostramos cola vacía
print("\nCola vacía: ")
pq.process_events()
```

```
Nombre:  Carmen         CGPA:  4.0      ID:  2
Nombre:  Aníbal         CGPA:  3.5      ID:  4
Nombre:  Zendaya        CGPA:  3.5      ID:  3
Nombre:  Joaquina       CGPA:  2.1      ID:  1
Nombre:  Eugenio        CGPA:  2.6      ID:  5
Nombre:  Eugenio        CGPA:  2.6      ID:  6
Nombre:  Piplup         CGPA:  1.1      ID:  7

Se elimina:  ['Carmen', 4.0, 2]

Cola vacía:
```

Out[68]: 'EMPTY'

Observamos que en la cola que acabamos de mostrar se cumplen todos los criterios de prioridad:

- El estudiante de mayor nota media acumulada (`Carmen`) está en la cima (mayor prioridad)
- Los estudiantes con misma nota media se ordenan alfabéticamente de manera ascendente (`Anibal` > `Zendaya`)
- Los estudiantes con misma nota media y mismo nombre se ordenan por número de ID ascendente (`Eugenio 5` > `Eugenio 6`)

Lista de eventos

Mostraremos el funcionamiento de la cola de prioridad, siguiendo la dinámica "ENTER (estudiante entra a la cola), SERVED (estudiante prioritario sale de la cola)".

In [65]:
```
e = Student
pq = PriorityQueue()

# Añadimos estudiante 2
e2 = e(3.8, "Sandro", 2)
print("ENTER\t", pq.enter(e2))

# Añadimos estudiante 3
e3 = e(2.5, "Eugenio", 3)
print("ENTER\t", pq.enter(e3))

# Añadimos estudiante 4
e4 = e(1.5, "Karlos", 4)
print("ENTER\t", pq.enter(e4))

# Servimos (eliminamos por prioridad)
pq.served()
print("SERVED")

# Añadimos estudiante 5
```

```python
e5 = e(1.1, "Emmanuel", 5)
print("ENTER\t", pq.enter(e5))

# Añadimos estudiante 6
e6 = e(3.9, "Zendaya", 6)
print("ENTER\t", pq.enter(e6))

# Añadimos estudiante 7
e7 = e(3.9, "Coscu", 7)
print("ENTER\t", pq.enter(e7))

pq.served()
print("SERVED")

# Añadimos estudiante 8
e8 = e(2.5, "Eugenio", 8)
print("ENTER\t", pq.enter(e8))

# Servimos (eliminamos por prioridad)
pq.served()
print("SERVED")
# Servimos (eliminamos por prioridad)
pq.served()
print("SERVED")

# Mostramos
print("\nPersonas en la cola (por orden de prioridad):\n")
pq.process_events()
```

```
ENTER    ['Sandro', 3.8, 2]
ENTER    ['Eugenio', 2.5, 3]
ENTER    ['Karlos', 1.5, 4]
SERVED
ENTER    ['Emmanuel', 1.1, 5]
ENTER    ['Zendaya', 3.9, 6]
ENTER    ['Coscu', 3.9, 7]
SERVED
ENTER    ['Eugenio', 2.5, 8]
SERVED
SERVED

Personas en la cola (por orden de prioridad):

Nombre:  Eugenio        CGPA:  2.5      ID:  8
Nombre:  Karlos         CGPA:  1.5      ID:  4
Nombre:  Emmanuel       CGPA:  1.1      ID:  5
```