Examen de Algorítmica y Complejidad (Plan 2014)

30 de junio de 2021

Nº matrícula:_____ **Nombre:** _____

Apellidos: _____

Problema. El famoso periodista político Tony Iron en su afamado programa de TV *Red Hot* ha contratado una empresa para que le diseñen su *Pactometer*, un algoritmo que permita generar el conjunto de pactos posibles entre un grupo de partidos políticos que se han presentado a las elecciones. El *Pactometer* debe tener en cuenta tanto el número de escaños conseguidos por cada partido como el desgaste político que supone pactar con otros partidos. Para un total de N diputados, será necesario que en el pacto haya, al menos, $((N/2)+1)$ diputados para obtener la mayoría absoluta. **Siempre se debe elegir como partido principal para formar gobierno el que tenga mayor número de diputados electos**. De cada partido (p_i) se conoce el nº de diputados (d_i) obtenidos en las elecciones $(N \geq d_i \geq 1)$, así como su orientación política (que se mide en una escala entre 1 y 10: 1 extrema izquierda, 10 extrema derecha).

El nivel de perjuicio que supondrá para un partido político p_i, pactar con otro p_j, dependerá de la distancia entre sus ideologías políticas. $Perjuicio(p_i, p_j) = (|o_i - o_j|)$, es una función que **devuelve un valor entre [0..1]**, y que depende de los valores de (o_i, o_j) que representan las orientaciones de cada uno de estos partidos. Para determinar si un partido realiza o no un pacto, existirá un umbral **μ € [0..1]**, de modo que si $Perjuicio(p_i, p_j) \geq \mu$ el pacto entre p_i y p_j no será aceptable.

Se pide implementar una **estrategia voraz** que dado el nº de diputados del Parlamento (N), un conjunto de partidos políticos y un umbral de pactos, devuelva (cuando sea posible) el pacto de gobierno alcanzado. *El objetivo de la estrategia consistirá en* ***minimizar*** *el perjuicio derivado de pactar con otros partidos,* así como ***maximizar*** *el número de diputados que se consiguen tras el pacto.*

SE PIDE: Implementar el método *ArrayList<Partido> greedyPactometer(ArrayList<Partido> misPartidos, int mayoriaABS, float umbralPacto)* empleando una estrategia voraz, así como todos los métodos auxiliares necesarios.

```
public class Partido {
    String nombre;
    int diputados;
    float orientacion;
    Partido(String nombre, int diputados, float orientacion){
        this.nombre = nombre;
        this.diputados = diputados;
        this.orientacion = orientacion;
    }
    float Perjuicio(Partido p2){
       return (Math.abs(this.getOrientacion()-p2.getOrientacion()) / 9)
    }
    /* getters y setters */
}
```

```
ArrayList<Partido> greedyPactometer(ArrayList<Partido> misPartidos, int mayoriaABS,
                                    float umbralPacto){
   ArrayList<Partido> solucion = new ArrayList<Partido>();
   Partido candidato = null;
   int numDiputados =0;
   //Se selecciona el partido de Gobierno == partido con mayor nº de escaños
   Partido partidoPrincipal = seleccionarPartidoGobierno(misPartidos);
   solucion.add(partidoPrincipal);
```

```
    misPartidos.remove(partidoPrincipal);
    numDiputados += partidoPrincipal.getDiputados();

    while ((numDiputados<mayoriaABS) && misPartidos.size()>0){
       candidato = seleccionarCandidato(misPartidos, partidoPrincipal);
       misPartidos.remove(candidato);
       if (partidoPrincipal.Perjuicio(candidato) < umbralPacto){
          solucion.add(candidato);
          numDiputados = numDiputados + candidato.getDiputados();
       }
    }
    if (numDiputados < mayoriaABS)
         return null;
    else
         return solucion;
}

Partido seleccionarPartidoGobierno(ArrayList<Partido> partidos){
    //selecciono el partido con más diputados para formar gobierno
    //en caso de empate me quedo con el primer partido político
    Partido p = null;
    int mejor = 0;
    for (int i=0; i<partidos.size(); i++){
       if (partidos.get(i).getDiputados()> mejor){
          p = partidos.get(i);
          mejor = p.getDiputados();
       }
    }
    return p;
}

Partido seleccionarCandidato(ArrayList<Partido> partidos, Partido partidoPrincipal){
    Partido p = partidos.get(0);
    float perjuicioMejor = partidoPrincipal.Perjuicio(partidos.get(0));
    for (int i=1; i< partidos.size(); i++){
      // se minimiza el perjuicio
       if (partidoPrincipal.Perjuicio(partidos.get(i)) < perjuicioMejor){
          p = partidos.get(i);
          perjuicioMejor = partidoPrincipal.Perjuicio(p);
       }
      // en caso de empate seleccionamos el p. político que más diputados nos proporcione
       else if ((partidoPrincipal.Perjuicio(partidos.get(i)) == perjuicioMejor) &&
                 p.getDiputados() < partidos.get(i).getDiputados()){
          p = partidos.get(i);
          perjuicioMejor = partidoPrincipal.Perjuicio(p);
       }
    }
    return p;
}
```