Entregable 4\_5

**Materia:** Programación 3.

**Alumno:** Nicolás Contreras.

**Enunciado**:

Dada una lista de empleados, la empresa necesita conformar dos grupos de trabajos para diferentes áreas. Los empleados se deben distribuir de forma tal que los dos grupos resultantes tengan **fuerzas de trabajo** iguales, o lo más similar posible. La fuerza de trabajo de un grupo se calcula como la suma de la fuerza de trabajo de los empleados que componen el grupo. De esta forma, el ideal es conformar dos grupos de empleados G1 y G2, tal que se **minimice**la diferencia de la fuerza de trabajo entre ambos grupos.  
Todos los empleados deben ser distribuidos entre ambos grupos de trabajo.  
  
Por ejemplo, dado 4 empleados con las siguientes fuerzas de trabajo:  
Empleados: [ e1: 30, e2: 43, e3: 78, e4: 58 ]  
  
Los grupos de trabajos más equiparados serían:  
Grupo 1: [e1, e3] - Fuerza de trabajo 108  
Grupo 2: [e2, e4] - Fuerza de trabajo 101

# Greedy

* **¿Qué son los candidatos?**

En este enunciado los candidatos están conformados por la lista de los empleados totales, los cuales deben agruparse en dos grupos.

* **¿Cuál es el criterio Greedy para seleccionar el próximo candidato (incluyendo aquellos criterios que se pensaron y descartaron durante el desarrollo del trabajo)?**

El criterio utilizado para seleccionar el próximo candidato se basa en elegir en cada paso el empleado cuya fuerza de trabajo se encuentre más próxima al resultado de la resta entre las fuerzas de los grupos de trabajo.

Ejemplo:

Candidatos [e1: 30, e2: 43, e3: 78, e4: 58]

Grupo1 fuerza de trabajo = 40 y Grupo2 fuerza de trabajo = 60.

Diferencia de Fuerzas = 60-40 = 20.

Candidato elegido = e1 ya que posee la fuerza (30) más próxima a 20;

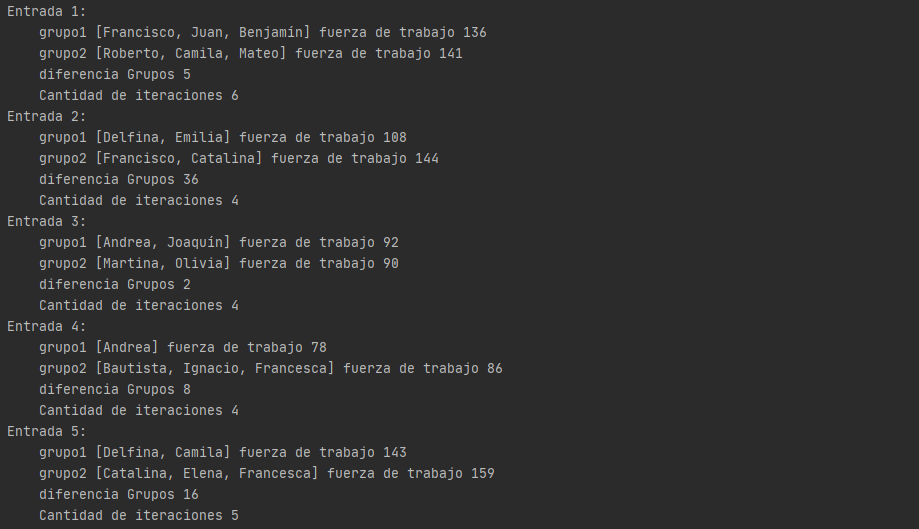
* **¿Cuándo es factible agregar un candidato a la solución?**

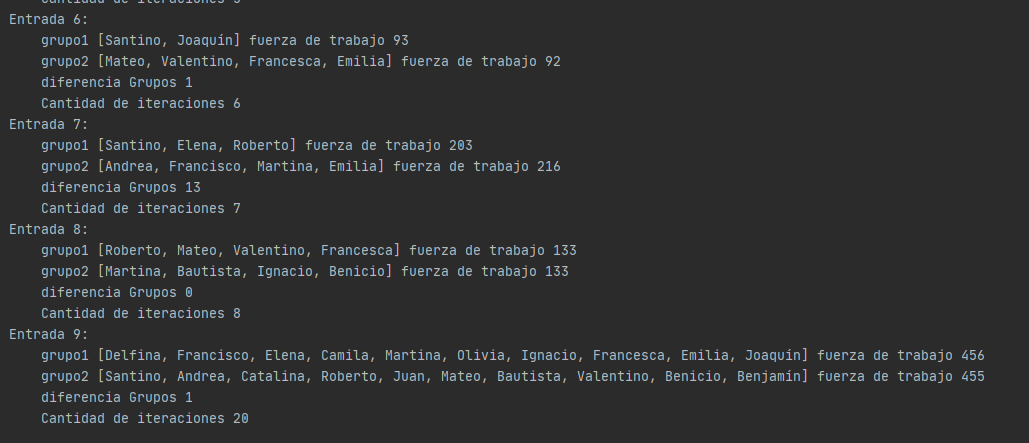
Debido a las condiciones de este enunciado agregar un candidato a la solución siempre va a ser factible, ya que todos los empleados deben pertenecer a alguno de los dos grupos. Es decir todos los empleados deben ser acomodados.

* **¿Cuándo alcanzamos una solución al problema?**

La solución es alcanzada una vez que todos los empleados fueron distribuidos, por lo tanto en este punto nuestra lista de candidatos se encontrara vacía debido a que en cada paso el candidato luego de ser seleccionado e incorporado a la solución, es removido de la misma( lista de candidatos).

**Resultados arrojados por el algoritmo:**

****

****

# Backtracking

* **¿Qué es un estado y qué datos contiene?**

Un estado está formado por toda la información que va a cambiar de un llamado recursivo del algoritmo al otro, es decir de un nodo padre a un nodo hijo en el árbol de exploración del algoritmo.

Dicha información en este algoritmo está formada por:

Una solución parcial en la cual se almacenan los grupos con sus respectivos empleados hasta el momento y la diferencia de fuerza entre ambos.

El empleado que estoy procesando en ese momento (este va ser siempre el que se encuentra en la posición 0 en nuestra lista de candidatos, ya que al momento de seleccionar el empleado este es removido, luego se llama recursivamente para el grupo 1 y para el grupo 2 y por último el empleado es incorporado nuevamente en la posición 0 de la lista).

* **¿Cuándo un estado es final?**

Un estado es final cuando todos los empleados fueron distribuidos en los grupos, por lo tanto la lista de empleados quedo vacía (this.empleados.size() == 0).

En nuestro árbol de exploración los encontraremos como las hojas, es decir nodos que no cuentan con hijos.

* **¿Cuándo un estado es solución?**

Un estado solución es todo aquel estado final en el cual la solución parcial que posee el mismo tiene menos diferencia de fuerzas entre ambos grupos que la solución (considerada optima) del problema hasta ese momento.

(parcial.getDiferenciaGrupos() < this.solucion.getDiferenciaGrupos() || this.solucion.getDiferenciaGrupos() == -1 ).

* **¿Cuáles son los hijos de un estado y qué modificaciones debo aplicar desde un estado para ir a un estado hijo?**

Los estados hijos son aquellos estados resultantes de tomar una decisión entre los llamados recursivos de nuestro algoritmo. Por lo tanto cada estado en nuestro árbol de exploración tendrá dos estados hijos resultantes de tomar la decisión de incorporar el empleado actual al grupo1 (primer llamado recursivo) o al grupo2 (segundo llamado recursivo).

Estas decisiones de incorporar el empleado a un grupo determinado son las causantes de la generación de nuevos estados hijos, es decir para ir de un estado a un estado hijo debemos debemos incorporar el empleado actual a uno de los dos grupos, otorgándole a ese estado hijo el empleado y la solución parcial resultante de la decisión tomada sobre a que grupo será incorporado.

* **¿Qué poda se le puede aplicar al problema?**

La poda aplicada en este problema contempla que se siga llamando recursivamente siempre y cuando la solución (optima del problema) no tenga como valor la diferencia de fuerza de los grupos un 0, ya que en este punto esa solución no va a poder mejorarse.

Debido a que los valores menores al 0 escapan de los números naturales y no puede encontrarse una solución con diferencia de grupos negativa (en el problema las fuerzas de los empleados toman valores positivos únicamente).

Podemos observar el resultado de aplicar poda en la entrada 8, la cual cuenta con una cantidad de iteraciones de 511 y luego al aplicar la poda esta disminuye a 63 iteraciones.

Figura sin poda

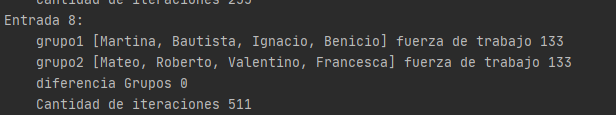
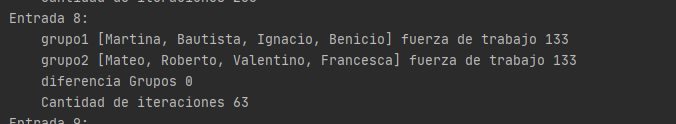
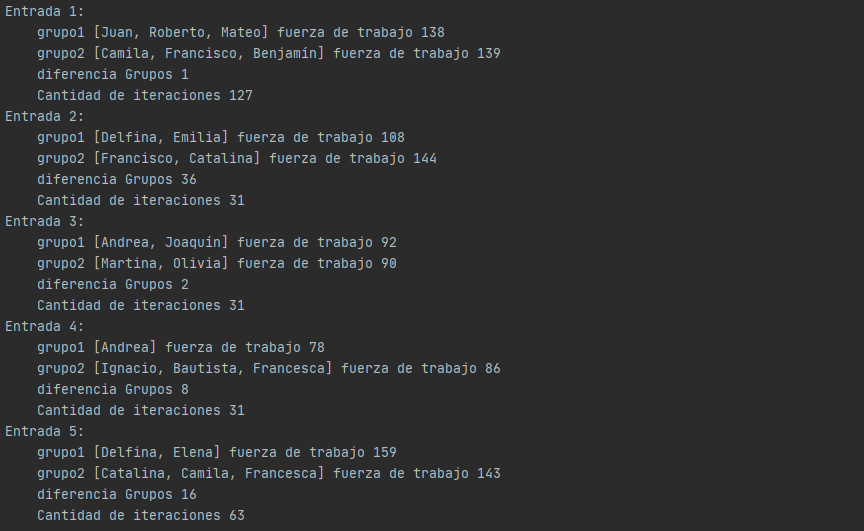
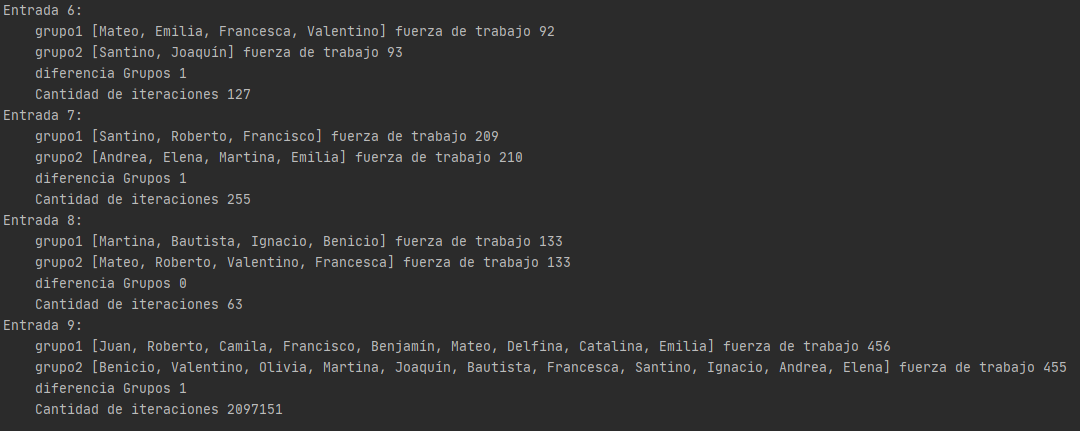
****

Figura con poda



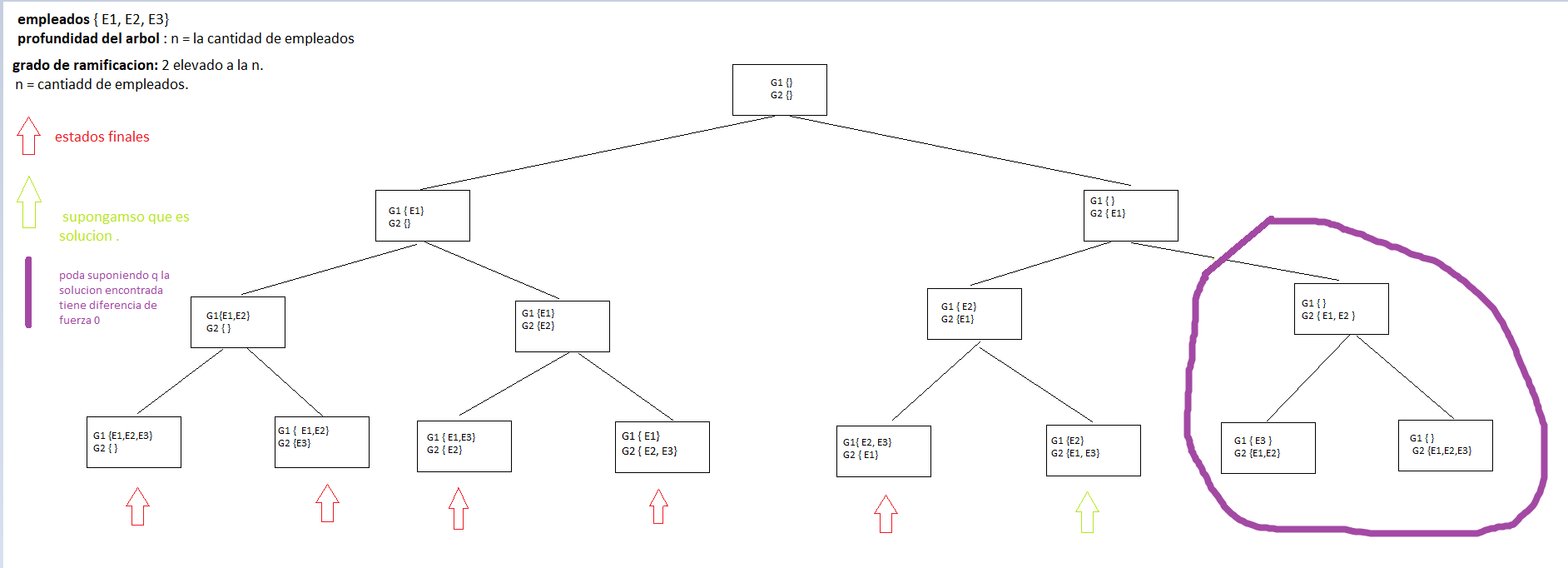
**Resultados arrojados por el algoritmo:**

****

****

**Árbol de exploración del algoritmo:**

Elaborado en paint (complejidad para realizarlo prolijo y de buen tamaño).

****