****

**Universidad Nacional de General Sarmiento**

**Programación III**

**Trabajo Práctico:**

**Fuerza Bruta**

**Profesores:**

* Patricia Bagnes.
* Javier Marenco.

**Grupo:**

* Andrés Geiser
  + 2018/42239150
  + geiserandres00@gmail.com
* Hernán Rodrigo Fernández
  + 2017/41584220
  + fernandezhr\_1998@hotmail.com

**Objetivo**

El objetivo que planteamos resolver en este trabajo práctico consiste en implementar una aplicación que muestre la conformación optima de un equipo de desarrollo a partir de una cierta cantidad de empleados con distintos puestos prefijados por el usuario y los respectivos lazos de convivencia entre ellos, lo cual determinara la posibilidad de generar un equipo optimo con las condiciones dadas.

**Responsabilidad de clases**

El proyecto cuenta con 6 paquetes en los que están distribuidas las clases según su funcionalidad:

* **Archivos**: donde se encuentran los archivos que contienen la lista de los nombres y apellidos que se utilizan para generar empleados.
* **Controlador**: donde se encuentran las clases encargadas de comunicar los distintos componentes visuales con aquellas que implementan la lógica interna. Donde se define el comportamiento de los componentes de la interfaz.
* **Iconos**: donde se encuentran los archivos que se utilizan como iconos en los componentes de la interfaz visual.
* **Modelo**: donde se encuentran las clases encargadas de poner en funcionamiento la lógica interna del programa y la suite de Test correspondiente a las mismas.
* **Principal**: contiene a la clase principal que se encarga de inicializar las clases visuales y desencadenar el funcionamiento del programa.
* **Vista**: donde se encuentran las clases visuales, que contienen el código que implementa la interfaz correspondiente al MainFrame y a los distintos paneles.

A continuación, presentaremos atención a cada una de las clases de los paquetes más importantes, describiendo sus variables internas, sus métodos y su rol general dentro del programa.

**Paquete Modelo**

**Clase Empleado**

Esta clase modelara a un empleado el cual tiene un nombre y un puesto dentro del equipo.

**Variables de clase**

* *String nombre*: representa el nombre del empleado.
* *String puesto*: representa el puesto del empleado.

Luego la clase tiene métodos getNombre() y getPuesto() que son llamados por otras clases, también posee un método equals para verificar que dos empleados sean distintos.

**Clase Requerimiento**

Esta clase será utilizada para guardar los requerimientos por puestos establecidos por el usuario, solo posee métodos de set() y get() para sus variables.

**Variables de clase**

* *int minLideres,maxLideres* : representa la cantidad mínima y máxima de líderes de proyecto que debe tener el equipo. Por defecto es inicializada en 1.
* *int minArquitectos, maxArquitectos*: representa la cantidad mínima y máxima de arquitectos que debe tener el equipo. Por defecto es inicializada en 1.
* *int minProgramadores, maxProgramadores*: representa la cantidad mínima y máxima de programadores que debe tener el equipo. Por defecto es inicializada en 1.
* *int minTesters, maxTesters*: representa la cantidad mínima y máxima de testers que debe tener el equipo. Por defecto es inicializada en 1.

**Clase Instancia**

Esta clase representa un estado actual del problema a resolver, contiene la lista de empleados, las relaciones entre ellos y los requerimientos de los puestos especificados.

**Variables de clase**

* *Grafo malasRelaciones:* contiene las relaciones entre los empleados.
* *ArrayList<Empleados> empleados*: contiene a todos los empleados generados por el usuario.
* *Requerimiento requerimiento*: contiene las especificaciones por puesto que fija el usuario.

Luego la clase contiene un método sonIncompatibles(int i, int j) que es utilizado para verificar la relación entre dos empleados de manera auxiliar y los demás métodos retornan la lista de empleados y las cotas especificadas en requerimientos.

**Clase Assert, testModelo y testSolverEquipo**

Estas clases contienen la suite de test que controlan los métodos de modelo y solver que contienen 24 y 11 test respectivamente, luego la clase Assert es utilizada para probar si dos listas de empleados son iguales, es decir que representan el mismo equipo solución.

**Clase grafo**

Esta clase será utilizada para modelar las distintas relaciones con la estructura de grafo, donde una buena relación significará que existe una arista entre ambos vértices y de lo contrario habrá mala relación.

**Variables de clase**

* *boolean [][] A*: representa la matriz de adyacencia interna del grafo.

**Métodos de clase**

* *void agregarArista(int i, int j)* : se encarga de agregar una arista entre los vértices **i** y **j** colocando en esa posición de la matriz true si existe arista (para nuestro problema true si tienen buena relación laboral).
* *void borrarArista(int i, int j)* : se encarga de eliminar la arista entre los vértices **i** y **j** volviendo false la posición correspondiente de la matriz **A**.
* *boolean existeArista(int i, int j):* verifica si existe una arista entre los vértices **i** y **j**. Es utilizado para verificar la relación entre dos empleados.
* *set<Integer> vecinos(int i):* retorna una colección con aquellos vértices tengan relación con el vértice **i**.

**Clase Modelo**

Esta clase contiene la lógica interna principal del programa, organiza la carga de empleados, las relaciones entre los empleados y gestiona la solución del equipo de desarrollo final.

**Variables de clase**

* *ArrayList<Empleado> empleado*: contiene la lista de todos los empleados que se generaron.
* *Grafo malasRelaciones*: representa a los vértices como empleados y a las aristas como buenas relaciones entre sí.
* *Requerimiento requerimiento*: contiene los valores máximos y mínimos elegidos para cada puesto dentro del equipo.
* *ArrayList<Empleado> solución*: contiene a los empleados validos que cumplen los requisitos y que son parte del equipo final.
* *Solver solver*: referencia a la clase que se utilizara para modelar el equipo óptimo.

**Métodos de clase**

* *void agregarEmpleado(String nombre, String puesto):* si el empleado es válido y si la lista aun no fue confirmada lo agrega a *empleados*.
* *void eliminarEmpleado(String nombre):* si el empleado es válido y si la lista aun no fue confirmada lo elimina de *empleados.*
* *void confirmarListaDeEmpleados()*: si los empleados actuales de *empleados* cumplen con los requisitos mínimos inicializa *malasRelaciones* con el correspondiente tamaño.
* *void agregarMalaRelacion(String nombreE1, String nombreE2) y eliminarMalaRelacion*: si la lista fue confirmada y los dos empleados existen agrega (buena relación) o elimina(mala relación) un la arista entre los dos empleados.
* *void resolver():* inicializa a *solver* y llama a .resolver con una nueva instancia armada con la lista de empleados, malasRelaciones y los requerimientos por puestos, almacenando en *solucion* a los empleados que integran el equipo final*.*
* *String getEstadisticas()*: devuelve una cadena con las estadísticas finales de la ejecución del solver

**Clase Solver**

Esta clase implementa un algoritmo de fuerza bruta basado en clique máxima, pero con adiciones propias de nuestro problema. Dado un grafo g encuentra la clique que contiene el grupo mas grande de empleados que cumplen los requisitos.

**Variables de clase**

* *Instancia instancia*: representa el conjunto de datos que recibe la clase para buscar el equipo óptimo.
* *Set<Integer> conjuntoActual:* contendrá los índices de los empleados que se formaran para conseguir el equipo óptimo.
* *ArrayList<Empleado>:*contendrá los empleados que pertenezcan a la solución final, es decir al equipo óptimo con las condiciones dadas.
* *Int totalIteraciones y double Tiempototal:*representan variables de estadística para controlar el tiempo total de ejecución y la cantidad de iteraciones del caso base.

**Métodos de clase**

* *ArrayList<Empleado> resolver():* inicializa a los conjuntos, llama al método encargado de la recursión con el elemento 0 y devuelve el equipo óptimo formado con los empleados de la *instancia*.
* *void recursión(int inicial):* implementa el algoritmo general recursivo de fuerza bruta, se lo llama inicialmente con algún índice desde el que comenzara a armar todos los equipos posibles. Cuando el inicial sea igual al tamaño del equipo de la instancia, es decir la cantidad de empleados sea igual al inicial, se comprueba que entre los empleados del equipo formado actualmente no existan incompatibilidades y que se cumplan con los requisitos preestablecidos de los distintos puestos.

**(Creo que haría falta tal vez dibujar un esquema para verlo más fácil)**

Luego se forma el equipo máximo que cumple los requisitos, de contener mas empleados que la solución optima esta pasa a ser la mejor solución encontrada. De esta manera se lanzará la recursión con todos los iniciales posibles para lograr encontrar el equipo optimo que cumpla con los requisitos. La cantidad de veces que se ejecutara el caso base corresponde a 2*n* con *n* igual a la cantidad de empleados del conjunto inicial.

**(Un poco incompleto! ¿Deberíamos poner una foto mostrando la complejidad del caso base?)**