**Ejercicio 3:**

Tras 6 largos años de construcción, todo está preparado para la hora H, 21 de diciembre de 2012 a las 0:00. Durante los últimos años un gran grupo de ingenieros, arquitectos, jefes de estado y miembros del ejército han estado preparando un dispositivo de extracción nunca visto. 4 grandes naves interestelares llamadas GeoEx, con una **capacidad para 200 personas cada una**, están listas para su partida al planeta Alpha 8. Este planeta tiene unas condiciones muy parecidas a las del planeta Tierra y por tanto, óptimas para la vida tal y como la conocemos. Sin embargo, este grupo de expertos está teniendo numerosos problemas a la hora de alojar a los diferentes viajeros en cada una de las naves.

Se han vendido diferentes tipos de billete, los cuales dan derecho a viajar en alguna de las naves que se fletarán para el viaje. A la hora de la venta de plazas, el gobierno ha elaborado un mapa donde se recoge el DNI de la persona (que suponemos único) asociado al tipo de localidad vendida, teniendo una estructura parecida a la siguiente:

| | |
|---|---|
| 77896754-L | A |
| 53457986-A | A |
| 19742597-T | B |
| 23786420-H | C |

Igualmente, cada tipo de localidad da acceso a algunas naves, mientras que otras naves están restringidas para determinados perfiles de acceso. El conjunto de naves permitidas para cada categoría ha quedado recogido en otro documento parecido al anterior que se accede mediante la clase de utilidad **GestionNavesPlaza**. Esta clase posee los siguientes métodos:

- Set<Plaza> getPlazasAsociadas(Nave e): devuelve las plazas compatibles con la nave que se pasa por parámetro
- Boolean esPlazaCompatible(Nave e, Plaza p): determina si un viajero con una plaza de tipo p puede viajar en la nave e.

Por otro lado, determinados usuarios de los vuelos, generalmente los que más han pagado por sus plazas, **desean viajar junto a sus seres queridos**. Para poder satisfacer estas necesidades, se ha elaborado un listado que almacena los DNI de cada usuario y junto al número de familiares que desean viajar con él. Esto es una restricción fuerte y debemos tenerla en cuenta a la hora de hacer la asignación, de manera que si añadimos a un pasajero debemos añadir también a sus familiares, si los tiene. Para ello se debe hacer uso del siguiente mapa que contiene la información:

| | |
|---|---|
| 77896754-L | 2 |
| 53457986-A | 1 |
| 19742597-T | 0 |
| 23786420-H | 0 |

En este punto, la comisión de expertos le ha contratado a usted para que realice un algoritmo que permita llevar a cabo una asignación óptima, de manera que se cumplan las restricciones impuestas y que cada usuario pueda viajar en una nave que le corresponda. Para ello, debe basarse en el algoritmo de Backtracking **teniendo en cuenta que la mejor solución será aquella que mayor cantidad de pasajeros permita situar en sus respectivas naves**, teniendo en cuenta también a los familiares de los usuarios que compraron las plazas.

Tenga en cuenta que cada usuario sólo puede viajar en una nave interestelar y que deberá suponer inicializadas y completadas aquellas estructuras que permiten almacenar la información pertinente para el sistema.

**Se pide:**

- Completar la ficha adjunta siguiendo la información del problema detallado anteriormente

- Implemente los siguientes métodos pertenecientes a la clase EstadoNavesImpl:
  - Boolean add(Nave e)

- Implemente los siguientes métodos pertenecientes a la clase ProblemaNavesImpl:
  - EstadoNaves getEstadoInicial()
  - Boolean esSolucion(SolucionNaves s)
  - Iterable<Nave> getAlternativas(EstadoNaves e)

| ***Ficha 1*** | |
|---|---|
| ***Problema Fin del Mundo*** | |
| *Técnica: Backtracking* | |
| *Tamaño: N* | |
| *Propiedades Compartidas* | *Map<String, Plaza> plazas,*<br>*GestionNavesPlaza naves,*<br>*Map<String, Integer> numeroFamiliares,*<br>*Set<DNI> conjuntoUsuarios*<br>*Set<Nave> navesDisponibles*<br>*Integer capacidadMaximaNaves* |
| *Propiedades del Estado* | *Map<Nave, Integer> plazasOcupadas*<br>*List<String> usuariosAsignados*<br>*List <String> usuariosPorAsignar* |
| *Solución: SolucionNaves* | |
| *Alternativas (EstadoNave e):*<br>$Nave\ n \parallel naves.admitePlaza(n, plazas.get(e.getUsuariosPorAsignar.last))$ | |
| *Add($c^{asignacion,usuariosAsignados,usuariosPorAsignar}$): // TODO*<br>*ultimoUsuario = usuariosPorAsignar.last*<br>*usuariosPorAsignar.remove(ultimoUsuario)*<br>*usuariosAsignados.add(ultimoUsuario)*<br>*plazasOcupadas.set(asignacion,*<br>*plazasOcupadas.get(c)+(1+numeroFamiliares.get(ultimoUsuario)))*<br>*Remove:*<br>*String ultimoAñadido = usuariosAsignados. get(usuariosAsignados.last();*<br>*plazasOcupadas.set(plazasOcupadas.get (ultimoAñadido) – (1 +*<br>*numeroFamiliares.get(ultimoAñadido)));*<br>*usuariosPorAsignar.add(ultimoAñadido)*<br>*usuariosAsignados.remove(ultimoAñadido)* | |
| *Estado inicial:*<br>*plazasOcupadas = {}; // Inicializado con todas las naves con valor asociado 0*<br>*usuariosAsignados = {};*<br>*usuariosPorAsignar = conjuntoUsuarios;* | |

- Implemente los siguientes métodos pertenecientes a la clase EstadoNavesImpl:
  - Boolean add(Nave e)

```
public Boolean add(Nave e){
        String ultimoUsuario = usuariosPorAsignar.remove(usuariosPorAsignar.size()-1);
        usuariosAsignados.add(ultimoUsuario);
        plazasOcupadas.put(e, plazasOcupadas.get(e) + 1+
              numeroFamiliares.get(ultimoUsuario));
        return true;
}
```

- Implemente los siguientes métodos pertenecientes a la clase ProblemaNavesImpl:
  - EstadoNaves getEstadoInicial()

```
public EstadoNaves getEstadoInicial(){
        plazasOcupadas = Maps.newHashMap();

        for (Nave nave: navesDisponibles){
              plazasOcupadas.put(nave, 0);
        }

        usuariosAsignados = Lists.newArrayList();
        usuariosPorAsignar = conjuntoUsuarios;

}
```

  - Boolean esSolucion(SolucionNaves s)

```
public Boolean esSolucion(SolucionNaves s){
        Boolean ret = true;
        for (Integer elems: s.getPlazasOcupadas().values()){
              if (elems > capacidadMaximaNaves)
                    ret = false;
        }

        if (!conjuntoUsuarios.containsAll(s.getUsuariosAsignados())){
              ret = false;
        }

        return ret;
}
```

  - Iterable<Nave> getAlternativas(EstadoNaves e)

```
public Iterable<Nave> getAlternativas(EstadoNaves e){
        Set<Nave> ret = Sets.newHashSet();
        String siguienteUsuario = e.usuariosPorAsignar.get(usuariosPorAsignar.size()-
              1));
        for (Nave nave: navesDisponibles){
              if (naves.esPlazaCompatible(nave,  plazas.get(siguienteUsuario)){
                    if (capacidadMaximaNaves - plazasOcupadas.get(nave) >= 1 +
                    numeroFamiliares.get(siguienteUsuario)){
                          ret.add(nave);
                    }
```

```
                }
            }
            return ret;
}
```

**Puntuación: 4 puntos**