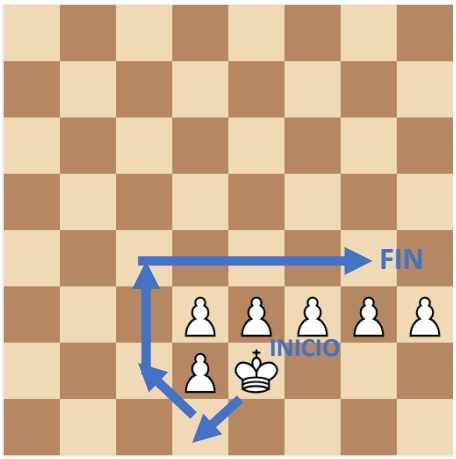
**Apellidos y nombre:**

**Titulación y grupo:**

---

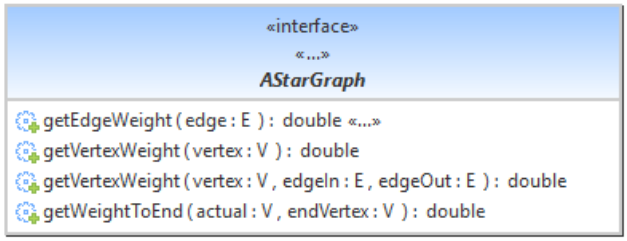
**Enunciado:**

Dado un tablero de ajedrez de nxn casillas con una serie de peones en juego, se desea saber la secuencia mínima de movimientos válidos para llevar al rey desde una casilla inicial a otra final evitando a los peones. El rey puede moverse en cualquier dirección una sola casilla cada vez.




Ejemplo: Si la casilla inicial es la (6,4) y queremos ir a la casilla (4,6), una solución posible es: [(6,4), (7,3),(6,2),(5,2),(4,2),(4,3),(4,4),(4,5),(4,6)]

**Nota: Una posible estimación de la distancia para llevar al rey de una casilla a otra es la mayor de las distancias según filas o columnas.**




Se pide:

1.Resuelva este problema como un problema de caminos mínimos usando grafos virtuales. Rellene los huecos indicados en la ficha.

2.Dentro de la clase que corresponda a sus vértices (que debe implementar VirtualVertex), implemente el método getNeighborListOf()que debe devolver un Set con los vértices vecinos al objeto vértice actual.

3.Dentro de la clase que corresponda a su grafo, implemente el método **public Set<SimpleEdge<TableroAjedrez>> edgesOf()** y el metodo **public double getWeightToEnd(TableroAjedrez startVertex, TableroAjedrez endVertex)**

| ***Problema del Rey: Grafo Virtual y A**** | |
|---|---|
| ***Propiedades Compartidas*** | *TI, Tablero inicial*<br>*TF, Tablero deseado*<br>*N, tamaño del tablero, derivada* |
| ***Propiedades del vértice*** | ***TODO*** |
| ***Aristas: TODO***<br><br>***Vecinos: TODO***<br><br>***Heurística (v, vf): TODO*** | |

**Solucion:**

| ***Problema del Rey: Grafo Virtual y A**** | |
|---|---|
| ***Propiedades Compartidas*** | *TI, Tablero inicial*<br>*TF, Tablero deseado*<br>*N, tamaño del tablero, derivada* |
| ***Propiedades del vértice*** | *i0, entero en [0,N), coordenada i del rey,*<br>*j0, entero en [0,N), coordenada j del rey,*<br>*datos, matriz de enteros, casillas,(opcionales dependiendo de la solución)* |

***Aristas: La arista a representa posible el desplazamiento en i,j del hueco.***

$\boldsymbol{A} = \{\boldsymbol{a} \in \{(\mathbf{1},-\mathbf{1}),(\mathbf{1},\mathbf{0}),(\mathbf{1},\mathbf{1}),(\mathbf{0},-\mathbf{1}),(\mathbf{0},\mathbf{1}),(-\mathbf{1},-\mathbf{1}),(-\mathbf{1},\mathbf{0}),(-\mathbf{1},\mathbf{1})\}|(\boldsymbol{i0},\boldsymbol{j0}) + \boldsymbol{a} \in [\mathbf{0},\boldsymbol{N}) \times [\mathbf{0},\boldsymbol{N}) \;\&\; ¡\, \boldsymbol{hayPeon}(\boldsymbol{i0}+\boldsymbol{a_0},\boldsymbol{j0}+\boldsymbol{a_1})\}$

***Vecinos:*** $(\boldsymbol{i0}+\boldsymbol{a}.\boldsymbol{i},\boldsymbol{j0}+\boldsymbol{a}.\boldsymbol{j}), \forall \boldsymbol{a} \in \boldsymbol{A}$

***Heurística(v, vf)****:*

***Max(ABS(TF.getPosRey().getX()-i0), ABS(TF.getPosRey().getY()-j0))***

***Vértice Origen: TI***

***Vértice Objetivo*** $\boldsymbol{TF}$

***Solución (lv)***

$\boldsymbol{lv}$

- getNeighborListOf()

```
@Override
public Set<TableroAjedrez> getNeighborListOf() {

                List<PairInteger> ls = Lists.newArrayList(
                        PairInteger.create(1, -1),
                        PairInteger.create(1, 0),
                        PairInteger.create(1, 1),
                        PairInteger.create(0, -1),
                        PairInteger.create(0, 1),
                        PairInteger.create(-1, -1),
                        PairInteger.create(-1, 0),
                        PairInteger.create(-1, 1));

        return ls.stream().filter(x -> x.v1 + problema.getI0() >= 0
                && x.v1 + problema.getI0() < ProblemaKing.numFilas
                && x.v2 + problema.getJ0() >= 0
                && x.v2 + problema.getJ0() < ProblemaKing.numFilas
                && !hayPeon(x.v1+problema.getI0(),x.v2+problema.getJ0()))
                .map(x -> TableroAjedrez.create(problema.getVecino(x.v1, x.v2)))
                                .collect(Collectors.<TableroAjedrez> toSet());
        }
```

Se permite el uso del créate y el getVecino como se habían visto en las prácticas

- edgesOf

```
        @Override
        public Set<SimpleEdge<TableroAjedrez>> edgesOf(){
                return this.getNeighborListOf().stream()
                   .map(x->SimpleEdge.create(this,x))
                   .collect(Collectors.<SimpleEdge<TableroAjedrez>>toSet());
        }
```

- getWeightToEnd

```
        @Override
        public double getWeightToEnd(TableroAjedrez
startVertex,TableroAjedrez endVertex) {
                if(startVertex==null || endVertex==null)
                        throw new IllegalArgumentException("Los vértices
inicial y final no pueden ser null");
                return startVertex.getDistance(endVertex);
        }

public double getDistance(TableroAjedrez endVertex) {
                return Math.max(Math.abs(this.getProblema().getI0()-
endVertex.getProblema().getI0()),Math.abs(this.getProblema().getJ0()-
endVertex.getProblema().getJ0()));

}
```