1) Rellenar las partes de la ficha que no estén rellenas

(3 puntos en total)

Problema esquiadores	
Técnica: Voraz	
Tamaño: t=tiempoDisponible	
Propiedades Compartidas	***pistasIniciales**: SortedSet<Pista>, ordenada por (tiempo/longitud) de mayor a menor* ***accesibilidad**:Map<Pista,SortedSet<Pista>>, ordenada por (tiempo/longitud) de mayor a menor* ***tiempoTotal**: Integer*
Propiedades del Estado	***pistasRecorridas**: List<Pista>* ***distanciaRecorrida**: Integer* ***tiempoDisponible**: Integer*
Solucion: SolucionPistas	
Alternativa: //TODO: Sólo tendrá en cuenta las pistas accesibles desde la última añadida $A_{estado}=\{ a_i \in accesibilidad.get(estado.pistasRecorridas.last)\}$	
Elección: //TODO: Se tendrán en cuenta las propiedades longitud y tiempo de cada Pista $h(estado,a_i)= a_i \in A_{estado}$ *con máximo* $(a_i^{longitud} / a_i^{tiempo})$	
Estado Inicial: //TODO: Habrá que considerar una de las pistas iniciales *pistasRecorridas += pistasIniciales.first* *distanciaRecorrida = pistasIniciales.first.longitud* *tiempoDisponible = tiempoTotal-pistasIniciales.first.tiempo*	
Estado Final: //TODO: Se tendrá que indicar cuándo el esquiador no puede seguir más *tiempoDisponible <= 0*	
Add(p_i): //TODO: Los valores del estado tendrán que actualizarse con la nueva Pista *pistasRecorridas +=* p_i *distanciaRecorrida +=* $p_i^{longitud}$ *tiempoDisponible -=* p_i^{tiempo}	

2) Implementar los métodos

   a. private void voraz()

   (1 punto en total)

```
private void voraz() {
      while (problema.isAlternativa(estado)) {
            Pista alternativa = problema.getAlternativa(estado);
            estado.add(alternativa);
      }
      solucion = estado.getSolucion();
}
```

   b. public boolean isAlternativa(EstadoPista e)

   (0.5 puntos en total)

```
public boolean isAlternativa(EstadoPistas e) {
      //Quedará tiempo, para coger otro telesillas?
      return e.getTiempoDisponible()>0;
}
```

c. public Pista getAlternativa(EstadoPista e)

(1 puntos en total)

```
public Pista getAlternativa(EstadoPistas e) {
	SortedSet<Pista>
		pistasAccesibles=accesibilidad.get(e.getLast());
	return pistasAccesibles.first()
}
```

d. public boolean add(Pista p)

(2.5 puntos en total)

```
public boolean add(Pista p) {
	pistasRecorridas.add(p);
	distanciaRecorrida+=p.getLongitud();
	tiempoDisponible-= p.getTiempo();

	return true;
}
```

e. public boolean esSolucion(SolucionPista s)

(2 puntos en total)

```
public boolean esSolucion(SolucionPistas s) {
	//No queda tiempo para coger otro telesillas
	boolean ret=s.getTiempoRestante()<=0;
	//Los kilometros esquiados son positivos
	ret&=s.getLongitudEsquiada()>=0;

	//cada pista da acceso a la siguiente
	if(ret){
		List<Pista> pistas= s.getPistasRecorridas();
		Pista anterior= pistas.get(0);
		for(int i=1; i<pistas.size() && ret;i++){
			Pista actual= pistas.get(i);
			ret&=accesibilidad.get(
				anterior).contains(actual);
			anterior=actual;
		}
	}
	return ret;
}
```