**PLANIFICACIÓN DE TRIPULACIONES AÉREAS**

Profesor:

Orosco, Ricardo Fabian

Martínez Saucedo, Ana Carolina

Grupo:

Von Elm, Lucas – LU: 1133639

Losada Santalise, Mauro Ignacio– LU: 1124705

Indurain Moneo, Ignacio – LU: 1137789

Mendieta, Juan Ignacio – LU: 1135071

Buenos Aires, 22 de noviembre de 2021.-

**Tabla de Contenidos**

[Introducción 3](#_Toc262832723)

[Descripción del Problema 3](#_Toc262832724)

[Tipos de Datos Abstractos 3](#_Toc262832725)

[Estrategia de Resolución 3](#_Toc262832726)

[Análisis de Complejidad Temporal 3](#_Toc262832727)

[Conclusiones 3](#_Toc262832728)

[Bibliografía 4](#_Toc262832729)

# Introducción

Se busca desarrollar un algoritmo que dado un listado de vuelos entre aeropuertos en un archivo .csv, y una cantidad N de tripulaciones disponibles, empezando con todas las tripulaciones desde un mismo aeropuerto de origen cubrir todos los vuelos y volver al aeropuerto de origen con las tripulaciones disponibles, minimizando el costo de espera. El costo de espera corresponde al intervalo de horas entre el horario de llegada al aeropuerto por parte de una tripulación y el horario de salida del siguiente vuelo tomado, sumando 1 al costo por cada hora de diferencia. Este costo empieza a acumularse pasadas las dos primeras horas de diferencia, que no se suman.

# Descripción del Problema

## Estrategia de Resolución

Para resolver el problema se utiliza la estrategia de mTSP, una versión con varios viajeros derivada del conocido TSP. Se realizará todas las combinaciones posibles de realizar cada vuelo con todas las tripulaciones para obtener de entre ellas el mínimo costo de espera. Esto se hará utilizando la técnica de backtracking, realizando llamados recursivos para cada movimiento hecho e ir actualizando el mejor resultado a medida que se realizan las combinaciones y se cumplen las condiciones estipuladas.

## Pseudocódigo del Algoritmo de Resolución del Problema

***Entrada:*** *etapa: Entero, todosVuelos: VectorTDA<Vuelo>, vuelosRealizados:ConjuntoTDA<Vuelo>, costoActual: Entero,  mejorCosto: Entero, costoAgregar: Entero, tripulaciones: VectorTDA <Tripulación>*

***Salida:*** *tripulaciones*

***Algoritmo RealizarVuelos:***

*costoActual ← costoActual + costoAgregar*

*//Realizar chequeo de condiciones de mejor Camino*

*Si (vuelosRealizados.capacidad() == todosVuelos.capacidad())*

*Si costoActual < mejorCosto*

*boolean estanDevuelta ← Verdadero*

*//Chequear si cada tripulacion esta devuelta en su punto de origen*

*Por cada tripulacion*

*Si tripulacion.ubicacion != tripulacion.origen*

*estanDevuelta ← Falso*

*Fin si*

*Fin por*

*//Todas las tripulaciones están devuelta en su punto de origen*

*Si estanDevuelta*

*mejorCosto = costoActual*

*//Guardar en cada tripulacion el nuevo mejor camino encontrado*

*Por cada tripulacion*

*tripulacion.camino ← tripulacion.caminoTemp // Guardar Resultado*

*Fin por*

*Fin si*

*Fin si*

*Sino*

*//Realizar Combinación de Vuelos*

*Por cada tripulacion;*

*VectorTDA<Vuelo> adyacentes ← obtenerAdyacentes()*

*Por cada adyacente*

*Si etapa>1*

*costoAgregar = calcularCosto(tripulacion.UltimoVuelo, adyacente)*

*Fin si*

*tripulaciones.recuperarElemento(tripulacion).getCaminoTemp().agregarElemento(etapa, adyacentes.recuperarElemento(adyacente))*

*vuelosRealizados.agregar(adyacentes.recuperarElemento(adyacentes))*

*tripulaciones = realizarVuelos(etapa + 1, todosVuelos, vuelosRealizados, costoActual, mejorCosto, costoAgregar, tripulaciones)*

*tripulaciones.recuperarElemento(tripulacion).getCaminoTemp().eliminarElemento(etapa)*

*vuelosRealizados.sacar(adyacentes.recuperarElemento(adyacente))*

*Fin Por*

*Fin Por*

***Algoritmo ObtenerAdyacentes***

*Entrada (origen: Vuelo, todosVuelos : Conjunto<Vuelo>, vuelosRealizados: Conjunto<Vuelo>)*

*Salida vectorAdyacentes: VectorTDA<Vuelo>*

*VectorTDA<Vuelo> vectorVuelo ← iniacializarVector*

*vectorVuelo ← todosVuelos.aVector*

*VectorTDA<Vuelo> vectorAdyacentes ← inicializarVector(todosVuelos.tamaño)*

*Entero j ← 0*

*Por cada Vuelo en vectorVuelo*

*//El vuelo sale desde el aeropuerto en el que me encuentro*

*//Me encuentro en el aeropuerto al que fui en el vuelo anterior*

*Si Vuelo.AeropuertoOrigen == Origen.AeropuertoDestino*

*//El vuelo sale después de que yo llegue al aeropuerto*

*Si Vuelo.FechaDespegue  > Origen.FechaAterrizaje*

*//El vuelo no fue realizado anteriormente por otra tripulacion*

*Si Vuelo no pertenece a VuelosRealizados*

*vectorAdyacentes ← AgregarVuelo(Vuelo)*

*j++*

*Fin si*

*Fin si*

*Fin si*

*Fin por*

*Devolver vectorAdyacentes*

***Algoritmo CalcularCosto***

*Entrada (vueloLLegada: Vuelo, vueloSaliente: Vuelo)*

*Salida (diferenciaHoras: Entero  || 0: Entero)*

*//Tiene que ser un long ya que devuelve un tiempo en milisegundos muy grande para un Entero*

*Long diferencia ← vueloLlegada.FechaAterrizaje - vueloSaliente.FechaDespegue*

*//1000 ← Pasa de Milisegundos a Segundos*

*//60 ← Pasa de Segundos a Minutos*

*//60 ← Pasa de Minutos a Horas*

*//Nos devuelve la diferencia en forma negativa*

*Long diferenciaHoras ← diferencia / (60 \* 60 \* 1000)*

*diferenciaHoras ← -diferenciaHoras*

*//Calcula si el tiempo de espera corresponde a una multa por el sindicato*

*Si diferenciaHoras > 2*

*Devolver diferenciaHoras-2*

*Sino*

*Devolver 0*

***Algoritmo LeerDatosVuelos***

*Entrada (VuelosDatos: Archivos.csv)*

*Salida Conjunto<Vuelo> conjuntoVuelos*

*lector ← BufferedReader(VuelosDatos)*

*ConjuntoTDA<Vuelo> conjuntoVuelos ← InicializarConjunto(todosVuelos.tamaño)*

*boolean primeraLinea ← Verdadero*

*Mientras (Linea = lector.LeerLinea) No nulo*

*Si primeraLinea*

*primeraLinea ← Falso*

*Fin si*

*Sino*

*//Separa cada datos en el csv con una coma*

*//Cada índice es un dato*

*String[] datos ← Linea.SepararPor(“,”)*

*String nroVuelo ← datos[0]*

*String origen ← datos[1]*

*String destino ← datos[2]*

*//Aplico el formato correspondiente a la Fecha Salida  y Fecha Llegada*

*DateFormat salidaFormat ← Formato de DD-MM-YYYY HH:MM*

*salida ← DateFormat.ADate(datos[3], salidaFormat)*

*DateFormat llegadaFormat ← Formato de DD-MM-YYYY HH:MM*

*llegada ← DateFormat.ADate(datos[4], llegadaFormat)*

*//Creo el vuelo con los datos y lo agrego al conjunto*

*Vuelo vuelo ← nuevo Vuelo(nroVuelo, origen, destino, salida, llegada)*

*conjuntoVuelos ← Agregar(vuelo)*

*Fin sino*

*Fin mientras*

*Devolver conjuntoVuelos*

***Algoritmo LeerDatosTripulaciones***

*Entrada (TripulacionesDatos: Archivo.csv)*

*Salida (tripulaciones: VectorTDA<Tripulaciones>)*

*//Como los vectores tienen un tamaño límite primero cuento la cantidad de tripulaciones que van a haber*

*contador ← BufferedReader(TripulacionesDatos)*

*boolean primeraLinea ← Verdadero*

*Mientras (Linea = contador.LeerLinea) No nulo*

*Si primeraLinea*

*primeraLinea ← Falso*

*Fin si*

*Sino*

*contador ← contador++*

*Fin sino*

*Fin Mientras*

*linea ← BufferedReader(TripulacionesDatos)*

*VectorTDA<Tripulacion> tripulaciones ← inicializarVector(contador)*

*boolean primeraLinea ← Verdadero*

*Entero i ← 0*

*Mientras (Linea = linea.LeerLinea) No nulo*

*Si primeraLinea*

*primeraLinea ← Falso*

*Sino*

*String[] datos ← Linea.SepararPor(“,”)*

*tripulaciones ← agregarElemento(i, nueva Tripulacion(vacio, vacio, 0)*

*Vuelo primero ← nuevo Vuelo(“0, datos[1], datos[1], temp, temp)*

*tripulaciones.tripulacion(i) ← agregarACaminoTemp(0, primero)*

*i ← i++*

*Fin sino*

*Fin Mientras*

*Devolver tripulaciones*

*CAMBIOS A CLASES PROVISTAS POR UNIVERSIDAD*

*CAMBIOS REALIZADOS A CLASE VECTOR*

***Algoritmo estaVacio***

*Salida: Boolean*

*//Se considera Vacio a un vector si el primer elemento esta vacio*

*//Por naturaleza de nuestro proyecto, el primero elemento del vecotr de una tripulacion siempre sera el vuelo con los datos de origen, para iniciar la búsqueda de adyacentes*

*Intentar*

*vector.RecuperarElemento(0)*

*devolver falso*

*//Si devuelve error significa que no encontró ningún elemento, y por ello está vacío*

*Atrapar Error OutOfBoundsException*

*devolver verdadero*

*//Algoritmo necesario para saber cuantos elementos hay en un vector*

*//Útil para no tener que usar el algoritmo de capacidad y tener que cerrar todos los módulso en try y catch*

***Algoritmo cantidadElementos***

*Salida: cantidad: Entero*

*Entero contador ← 0*

*Intentar*

*//Cuenta cada elemento que existe en un vector*

*//Si no encuentra elemento en la posicion entonces lanza error*

*//Al atrapar el error no se suma al contador*

*Por cada elemento hasta CapacidadVector*

*RecuperarElemento(elemento)*

*contador ← contador++*

*Fin por*

*Atrapar Error NullPointerException*

*Ignorar*

*Devolver Contador*

*NUEVAS CLASES NECESARIAS PARA EL FUNCIONAMIENTO DEL PROYECTO*

*CLASE TRIPULACION*

*ATRIBUTOS:*

* *VectorTDA<Vuelo> caminoTemp*
* *VectorTDA<Vuelo> camino*
* *Entero costoCamino*

*ALGORITMOS:*

***Algoritmo ObtenerUbicacion***

*Salida: Ubicacion: String*

*String ultimo*

*Intentar*

*Por cada elemento en caminoTemp.cantidadElementos*

*ultimo ← caminoTemp.recuperarElemento(elemento).obtenerAeropuertoDestino*

*Fin por*

*Atrapar Error NullPointerException*

*Ignorar*

*devolver Ultimo*

*CLASE VUELO*

*Atributos:*

* *String nroVuelo*
* *String aeropuertoOrigen*
* *String aeropuertoDestino*
* *Date fechaDespegue*
* *Date fechaAterrizaje*

## Análisis de Complejidad Temporal

Concluimos que la complejidad temporal del algoritmo es de O(Tn), siendo T la cantidad de tripulaciones disponibles a combinar en los N vuelos disponibles.

# Conclusiones

El problema consiste en recorrer V vuelos entre aeropuertos con N tripulaciones disponibles, visitandolos todos y asegurandose de que cada tripulacion haya vuelto a su aeropuerto de origen. Para resolverlo se utilizó una estrategia de Backtracking basada en el problema de TSP, modificado para avalar las varias tripulaciones disponibles y todas las combinaciones de los vuelos entre ellas existentes. El problema resulta interesante debido a la gran diversidad de casos reales en lo cual se debe aplicar un método similar.

Bibliografía

* <*Solving Airline Crew Scheduling Problems by Branch-And-Cut*> - <Karla L. Hoffman, Manfred Padberg>- Management Sciene – Junio 1993
* < *The multiple traveling salesman problem: an overview of*
* *formulations and solution procedures*> - <Tolga Bektas.>- Omega – Volume 34