**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 realiza un algoritmo que recorra todos los vuelos, asignandolos a una tripulación en caso de ser un vuelo adyacente, y al llegar hasta un nodo hoja, se vuelve a una iteración atrás para asignar el vuelo previo a otra tripulación. Este algoritmo se repite hasta que se completen

## Pseudocódigo del Algoritmo de Resolución del Problema

***Algoritmo RealizarVuelos:***

***Entrada:*** *etapa: Entero, todosVueloVector: VectorTDA<Vuelo>,*

*, todosVueloConjuntos: Conjunto<Vuelo>, vuelosRealizados: ConjuntoTDA<Vuelo>, costoActual: Entero, mejorCosto: Entero, costoAgregar: Entero, tripulaciones: VectorTDA <Tripulación>*

***Salida:*** *tripulaciones*

*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*

*//Guardo el costo en el atributo de un objeto para evitar problemas*

*Tripulacion(0).setCosto(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*

*//Recorre todos los vuelos*

*Por cada Vuelo en todosVuelosVector*

*//Selecciona la siguienteTripulacion*

*Por cada tripulación en Tripulaciones*

*//Analiza si la tripulación puede realizar ese vuelo*

*Si esAdyacente (vuelo,vuelosYaHechos, todosVuelos, tripulación)*

*vuelosYaHechos 🡨 Agregar(vuelo)*

*Si etapa > 1*

*costoAgregar 🡨 calcularCosto(vuelo, tripulación)*

*Fin Si*

*tripulación 🡨 AgregarACaminoTemp(vuelo)*

*tripulaciones = realizarVuelos(todosVuelosVector, todosVuelosConjunto, vuelosrealizados, costoActual, tripulaciones(0).ObtenerCosto, tripulaciones)*

*vuelosYaHecho 🡨 Quitar(vuelo)*

*tripulación 🡨 QuitarEnCaminoTemp(vuelo)*

*Fin si*

*Fin por*

*Fin por*

*devolver tripulaciones*

***Algoritmo esAdyacente***

*Entrada (vueloHacer: Vuelo, vuelosYaHechos: ConjuntoTDA<Vuelo>, todosVuelos: ConjuntoTDA<Vuelo>, ultimoVueloTripulacion: Vuelo)*

*Salida (boolean)*

*Si vuelosYaHechos.pertenece(vueloHacer)*

*Devolver falso*

*//Si el vuelo sale después de que yo llegue al aeropuerto*

*Si vueloHacer.FechaDespegue > tripulación.UltimoVueloFechaAterrizaje*

*//Si ya esperaron las dos horas de espera requeridas*

*Si (calcularCosto(vueloUltimaTripulacion, VueloHacer+2)>2*

*//Si el vuelo sale del aeropuerto en el que me encuentro*

*Si vueloHacerAeropuertoOrigen == tripulación.UltimoVueloAeropuertoDestino*

*Devolver verdadero*

*Sino*

*Devolver falso*

*Sino*

*Devolver falso*

*Sino*

*Devolver falso*

***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*

*Devolver diferenciaHoras-2*

***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*

*VectorTDA<Vuelo> temp = inicializarVector*

*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 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 costo*

*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

Costo Algoritmo Principal realizarVuelos:

O(MX!)

M: Cantidad de tripulaciones

X: Cantidad de vuelos

El costo es dado porque recorre por cada tripulacion todos los vuelos, y al realizar un vuelo se elimina de la consideracion para la siguiente etapa, se reduce en uno.

**Costo Algoritmo esAdyacente:**

Constante

**Costo Algoritmo calcularCosto:**

Constante

**Costo Algoritmo LeerDatosVuelos:**

O(N)

**Costo Algoritmo LeerDatosTripulaciones:**

O(2N)

# Conclusiones

Tras la realización del proyecto llegamos a la conclusión de que estos tipos de problemas mTSP resultan prácticos para diversos casos en la vida real. Una de estas situaciones puede ser la entrega de paquetes a diversos puntos en la ciudad. Además, también comprendimos que aplicar distintas condiciones de lo que se considera un “mejor camino” resulta sencillo, pudiendo modificar el algoritmo según las necesidades que tengamos, pero con una estructura 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

Repositorio del programa

<https://github.com/xlucasve/TPO-ProgramacionIII.git>

Link de respaldo:

<https://github.com/xlucasve/TPO-ProgramacionIII>