# Algoritmos y Programación II (75.41)
# Trabajo Práctico Nº3

11 de noviembre de 2013

## 1. Introducción

Hemos sido contratados por el mismísimo Santa Claus (Papá Noel, para los amigos), para organizar los preparativos previos al día de Navidad. Como todos saben, el 25 de Diciembre es un día agitado, pero pocos se imaginan que el día 24 también lo es. Dado el ritmo frenético que se debe llevar el día previo, se están buscando soluciones informáticas para resolver de forma óptima los problemas que se presentan.

Es por eso que se nos encargó la creación de un programa (de ahora en más 'Navidalgos') que permita organizar el día previo a Navidad.

El programa va a funcionar de backend para otro sistema. Se necesita que el mismo presente una interfaz de comandos automatizable, que tome su entrada desde la entrada estándar.

## 2. Consigna

En el transcurso del día 24, se debe hacer la recolección de los juguetes para los niños. Estos se encuentran en las diferentes fábricas de juguetes de los Duendes, que trabajan en horarios reducidos. Además, como el trineo mágico queda reservado para el día 25, se debe usar uno más antiguo que, si bien viaja casi tan rápido, no es capaz de soportar tanto peso.

Se deberá decidir, entre el conjunto total de las fábricas, cuáles serán las fábricas que efectivamente serán visitadas. Se tiene como restricción que no se pueden visitar dos fábricas con horarios superpuestos[1].
Dado que no conocemos, a priori, el valor de los objetos en cada fábrica, se desea visitar tantas fábricas como sea posible.

Los viajes se realizan desde y hasta la ubicación del Polo Norte. Aún cuando los viajes se realizan en forma virtualmente instantánea, se debe seleccionar el camino más corto hacia la fábrica, entre todos los posibles (regulaciones internacionales impiden el viaje aleatorio por los cielos).

Además, en cada fábrica se desea llenar el trineo con los objetos de mayor valor de felicidad (moneda corriente en Navidad). Para cada juguete de la fábrica, los duendes proveen en números enteros del valor del juguete y su peso.
Se desea cargar en el trineo los objetos que sumen un valor máximo, sin exceder la capacidad del trineo.

Implementar en Lenguaje a elección (acordando en forma previa con el ayudante a cargo) el sistema que asegurará la mejor de las Navidades, que debe proveer las siguientes operaciones:

- Carga inicial de archivos:
  - Carga de Fábricas de Duendes
  - Carga de juguetes disponibles
  - Carga del mapa de las fábricas y ubicación del Polo Norte
- Servicios para consultas:
  - Listar Fábricas a Visitar, por horario

[1] Se consideran los tiempos de traslado nulos, ya que el trineo es muy rápido. Por lo tanto, la actividad en una fabricar puede terminar en el mismo minuto en el que empieza la actividad en la siguiente fábrica

- Listar camino desde el Polo Norte hacia una fábrica
- Valor de los juguetes retirados de una Fábrica
- Listar camino desde una fábrica hacia el Polo Norte
- Dar valor total de todos los juguetes retirados de todas las fábricas

## 2.1. Protocolo

Este programa formará parte de un sistema automatizado. En lugar de presentar un menú con opciones en la pantalla, se esperará que el programa respete un *protocolo de comunicación*, en el cual se reciben comandos por la entrada estándar (`stdin`). Cada comando recibido debe ser procesado y su resultado debe ser escrito en la salida estándar (`stdout`).

Cada línea de la entrada corresponde a un comando, y todos los comandos respetan el mismo formato. El nombre del comando está separado de los parámetros (si existen) por un espacio, y los parámetros están separados por comas:

```
comando parametro1,parametro2,parametro3,...
```

## 2.2. Servicios para consultas

### 2.2.1. Comando: `listar_fabricas`[2]

**Formato:** `listar_fabricas`

**Descripción:** Lista cantidad óptima y fábricas a visitar.

**Salida:** Cantidad: <número de fábricas>, seguido de un fin de línea, y una fábrica por línea, ordenadas por horario de visita, con el siguiente formato:

```
Cantidad: <cantidad de fábricas a visitar>
<id_fabrica>,<horario de entrada>,<horario de ↩
    ↪salida>
```

**Ejemplo:**

```
listar_fabricas
Cantidad: 3
18,10:00,11:28
1,15:15,21:01
703,21:01,23:59
```

[2]Si existiera más de una configuración óptima, debe elegirse siempre aquella que prefiera: en primera instancia fábricas que finalicen lo antes posible; en segunda instancia fábricas que inicien lo antes posible; y, en última instancia, fábricas con el menor id posible

#### 2.2.2. Comando: `valuar_juguetes`

**Formato:** `valuar_juguetes id_fabrica`

**Descripción:** Indica el valor total máximo de los juguetes a retirar de la fábrica.

**Parámetros:**

`id_fabrica`: Identificador de fábrica para el cual se desea obtener el valor.

**Salida:** Total: <valor total de los juguetes retirados de la fábrica> Sonrisas. Mensaje de error apropiado en caso de error (ver sección 2.3).

**Ejemplo:**

```
valuar_juguetes 10
Total: 4059 Sonrisas
```

#### 2.2.3. Comando: `valuar_juguetes_total`

**Formato:** `valuar_juguetes_total`

**Descripción:** Indica el valor total de los juguetes a retirar, de todas las fábricas. Resulta de sumar los resultados óptimos de cada fábrica.

**Salida:** Total: <valor total de los juguetes retirados> Sonrisas.

**Ejemplo:**

```
valuar_juguetes_total
Total: 345920 Sonrisas
```

#### 2.2.4. Comando: `camino_optimo`

**Formato:** `camino_optimo id_fabrica`

**Descripción:** Indica el camino óptimo (el más corto) desde el Polo Norte hasta la fábrica especificada.

**Parámetros:**

`id_fabrica`: Identificador de la fábrica para la cual se desea obtener el camino óptimo.

**Salida:** Listado de las coordenadas del camino óptimo (uno por línea, separando latitud y longitud con una coma)

**Ejemplo:**

```
camino_optimo 10
-58.37555922960053,-34.6905536694991
-58.37634890620792,-34.68986586444252
-58.37736540162192,-34.68898622444274
-58.37816347192872,-34.68829290276174
-58.37854771235283,-34.68860802809196
-58.37885654180224,-34.68885186966851
-58.37965761692605,-34.68953774089236
-58.37989992288034,-34.68935114117046
```

### 2.2.5. Comando Opcional: `listar_juguetes`

**Formato:** `listar_juguetes id_fabrica`

**Descripción:** Lista los juguetes a retirar de una fábrica, que dan valor de felicidad máximo.

**Salida:** Mensaje de error apropiado en caso de error (ver sección 2.3). en caso contrario, un juguete por línea, ordenados por su identificador, con el siguiente formato:

```
<id_juguete>
```

**Ejemplo:**

```
listar_juguetes
4
8
15
16
23
42
```

### 2.2.6. Comando Opcional: `entrenar_duendes`

**Formato:** `entrenar_duendes arch_juguetes`

**Descripción:** Se desea entrenar a los Duendes para realizar el trabajo de seleccionar los juguetes. Este comando muestra una interfaz gráfica, que le permite a los Duendes seleccionar los objetos que el duende cree que hace óptima la solución.
Se debe permitir seleccionar y quitar los objetos a agregar en el trineo.
Se debe recalcular el porcentaje de valor respecto del óptimo y mostrarlo cuando se selecciona en forma gráfica la opción de comparar.
Se debe recalcular el porcentaje de peso respecto de la capacidad y actualizarlo automáticamente.
No se debe permitir que el peso que agregue duende supere la capacidad del trineo.
El valor final del porcentaje alcanzado se debe considerar como el porcentaje cuando se selecciona en forma gráfica la opción de salir.
Se tiene como limitación que el número de juguetes en el archivo es igual a 10.

**Parámetros:**

`arch_juguetes`**:** Archivo con la información de los juguetes disponibles.

**Formato de Archivo:**

`arch_juguetes`**:** Archivo con la información de los juguetes disponibles.

```
id_juguete1,ruta1,valor,peso
id_juguete2,ruta2,valor,peso
...
id_juguete10,ruta10,valor,peso
```

Donde:

`ruta`**:** Ruta de la imagen del juguete.

`id_juguete`**:** Número entero, Identificador de juguete.

`valor`**:** Número entero, Medida del valor del juguete. Reúne tanto valor monetario como potencial valor sentimental.

`peso`**:** Número entero, Medida de Peso del juguete

**Salida:** <Porcentaje alcanzado por el duende> %, respecto de la solución óptima.

```
        <porcentaje_alcanzado>%
```

**Ejemplo:**

```
        listar_juguetes id_fabrica
        83%
```

Como ejemplo, se muestra la siguiente imagen:

### 2.2.7. Comando Opcional: `graficar_rutas`

**Formato:** `graficar_rutas id_fabrica`

**Descripción:** Guarda un archivo KML (de nombre id_fabrica.KML) las rutas óptimas de ida y de vuelta a la fábrica (desde el Polo Norte) El formato y la información necesaria para generar dicho archivo se encuentran en el archivo de ejemplo que está en la página de la materia.

**Parámetros:**

`id_fabrica`**:** Identificador de fábrica para el cual se desea obtener el valor.

**Salida:** Mensaje de error apropiado en caso de error (ver sección 2.3). `OK`, en caso contrario.

**Ejemplo:**

```
        graficar_rutas 10
        OK
```

## 2.3. Mensajes de error

El sistema debe imprimir los siguientes mensajes de error, en cada uno de los casos de error descriptos a continuación:

**Fabrica inexistente:**

```
Error: la fabrica con id <idf> no existe
```

# 3. Carga de archivos

## 3.1. Formato de los archivos

### 3.1.1. Archivo de Fábricas

```
id_fabrica1,id_esquina1,horario_entrada,horario_salida
...
id_fabricaN,id_esquinaN,horario_entrada,horario_salida
```

Donde:

`id_fabrica`**:** Número entero, Identificador de fábrica.

`id_esquina`**:** Número entero, Identificador de esquina en el mapa de calles en donde está ubicada la fábrica.

`horario_entrada`**:** Número entero que expresa el horario, en minutos desde las 00:00, en el que la fábrica comienzaría a atender a Santa Claus.

`horario_salida`**:** Número entero que expresa el horario, en minutos desde las 00:00, en el que la fábrica terminaría de atender a Santa Claus

### 3.1.2. Archivo de Juguetes

```
id_fabrica1,id_juguete1,valor,peso
id_fabrica1,id_juguete2,valor,peso
...
id_fabrica2,id_juguete18,valor,peso
id_fabrica2,id_juguete19,valor,peso
...
id_fabricaN,id_jugueteM,valor,peso
```

Donde:

`id_fabrica`**:** Número entero, Identificador de fábrica.

`id_juguete`**:** Número entero, Identificador de juguete.

`valor`**:** Número entero, Medida del valor del juguete. Reúne tanto valor monetario como potencial valor sentimental.

`peso`**:** Número entero, Medida de Peso del juguete

### 3.1.3. Mapa de calles

```
N
id_esquina1,x1,y1,latitud1,longitud1
id_esquina2,x2,y2,latitud2,longitud2
id_esquina3,x3,y3,latitud3,longitud3
...
id_esquinaN,xN,yN,latitudN,longitudN
M
id_calle1,esquina_inicial1,esquina_final1
id_calle2,esquina_inicial2,esquina_final2
id_calle3,esquina_inicial3,esquina_final3
...
id_calleM,esquina_inicialM,esquina_finalM
```

Donde:

`N`: Cantidad de esquinas totales en el mapa.

`id_esquina`: Número entero, Identificador de esquina.

`x`: Número con dos decimales (el punto es el separador decimal). Coordenada x expresada en metros. Usar la norma euclídea para calcular la distancia entre dos esquinas.

`y`: Número con dos decimales (el punto es el separador decimal). Coordenada y expresada en metros. Usar la norma euclídea para calcular la distancia entre dos esquinas.

`latitud`: Coordenada de latitud de la esquina

`longitud`: Coordenada de longitud de la esquina.

`M`: Cantidad de calles totales en el mapa.

`id_calle`: Número entero, Identificador de calle.

`esquina_inicial`: Identificador de esquina inicial para la calle.

`esquina_final`: Identificador de esquina final para la calle.

# 4. Consignas Opcionales

Para la aprobación del Trabajo Práctico no es necesario implementar las consignas opcionales. Sin embargo, todo el trabajo adicional será tenido en cuenta en la calificación final.

- Implementar el comando listar_juguetes, que lista qué juguetes se retiran de cada fábrica.

- Implementar el comando entrenar_duendes, que implementa la interfaz gráfica.
- Implementar el comando graficar_rutas, que almacena el archivo KLM correspondiente.

# 5. Pruebas

Junto con la especificación se provee de **pruebas automáticas para el programa completo**. Estas pruebas serán de utilidad para revisar que el programa cumpla con la especificación del protocolo de entrada/salida.

Para la aprobación del Trabajo Práctico es requisito que el programa implementado pase todas las pruebas.

## 5.1. Ejecución de las pruebas

Una vez descomprimido el archivo zip con las pruebas[3], se debe efectuar los siguientes pasos para correr las pruebas:

1. Compilar el programa (supongamos que se llama `tp3`)
2. Ejecutar[4]:

```
$ bash pruebas/correr-pruebas.sh ./tp3
```

Cada una de las pruebas está especificada en un archivo con extensión `.test` dentro de la carpeta de pruebas. Por ejemplo:

```
pruebas/
 \- correr-pruebas.sh
 \- prueba1.test
 \- prueba2.test
```

El script `correr-pruebas.sh` ejecutará el programa una vez por cada prueba, pasándole en la entrada estándar los comandos especificados en la prueba. Luego verificará que la salida estándar del programa sea exactamente igual a la esperada según el protocolo.

[3]Puede descomprimirse en cualquier lugar; para el ejemplo suponemos que se guardó en la misma carpeta que el TP. Es decir, el ejecutable `tp3` quedaría al mismo nivel que la carpeta `pruebas` (que contiene el archivo `correr-pruebas.sh`).

[4]Para correr las pruebas es necesario disponer de un entorno con línea de comandos Bash y herramientas GNU. En Linux seguramente no sea necesario instalar nada. Existen varias implementaciones para Windows; por ejemplo MSYS o Cygwin. Para Mac OSX existe el paquete `coreutils`

# 6. Criterios de aprobación

A continuación describimos criterios y lineamientos que deben respetarse en el desarrollo del trabajo.

## 6.1. Utilización de estructuras de datos

Para la realización de este trabajo es necesario utilizar la estructura de datos grafo vista en clase, además de crear las estructuras adicionales que se consideren necesarias. Puede utilizar todas las estructuras de datos que se crean convenientes, así como aplicar las técnicas de programación vistas en clase para resolver cada problema.

Todas las estructuras deben estar implementadas de la forma más genérica posible y correctamente documentadas.

## 6.2. Programa

El programa debe cumplir los siguientes requerimientos:

- Debe estar adecuadamente estructurado y modularizado, utilizando funciones definidas de la forma más genérica posible, sin caer en lo trivial.
- El código debe ser claro y legible.
- El código debe estar comentado y las funciones definidas, adecuadamente documentadas.
- Además, claro, debe satisfacer la especificación de la consigna y pasar todas las pruebas automáticas.

## 6.3. Informe

El informe deberá consistir de las siguientes partes:

- **Carátula** con la información de los alumnos/as y el ayudante asignado.
- **Análisis y diseño**: Describir la solución elegida para resolver cada problema propuesto, y cómo se lleva a cabo. En particular, mencionar cómo es el flujo del programa, qué algoritmos y estructuras de datos se utilizan, y cuál es el orden de ejecución en tiempo y espacio de cada operación.

- **Implementación**: Incluir aquí *todo* el código fuente utilizado (en formato `monoespaciado`, para facilitar su lectura).
- También *opcionalmente,* toda explicación adicional que consideren necesaria, referencias utilizadas, dificultades encontradas, cambios o mejoras que se podrían hacer a futuro y conclusiones.

El informe debe estar lo más completo posible, con presentación y formato adecuados. Por ejemplo, este enunciado cumple con los requerimientos de un informe bien presentado.

# 7. Entrega

El trabajo consiste en:

- El informe impreso.
- El informe digital, en formato `.pdf`
- Una versión digital de **todos** los archivos de código fuente, separados del informe, en un archivo comprimido (`.zip` o `.tar.gz`).

Los dos últimos deben enviarse a la dirección `tps.7541rw@gmail.com`, colocando como asunto:

```
TP3 - Padrón1 - Apellido1 - Padrón2 - Apellido2
```

Se aclara que por código fuente se entiende todos los archivos necesarias para la (compilación y) ejecución del programa. No deben entregarse nunca archivos `.o` u otros archivos compilados.

El informe impreso debe entregarse en clase. El plazo de entrega vence el **Viernes 29 de Noviembre de 2013**.