ANÁLISIS Y DISEÑO DE ALGORITMOS

Programación Dinámica

Práctica 6 de laboratorio

Esta práctica ocupa dos entregas:

Entrega 1: Versiones recursivas (ingenua y memoización) y la gestión de argumentos.
Entrega 2: Práctica completa.

Fechas de entrega: Respectivamente, hasta los días 3 y 10 de abril, 23:55h.

Encaminamiento óptimo

Una empresa que ofrece servicios de almacenamiento en la nube dispone de n servidores (que también llamaremos nodos) donde aloja los archivos de sus clientes. Los nodos están conectados entre sí mediante un *bus* privado de comunicaciones bidireccional. El bus está compuesto por nodos intermedios, cada uno con dos conexiones hacia ambos vecinos; y dos nodos terminales, uno en cada extremo, con una única conexión al nodo vecino.

Por cuestión de costes, solo una cantidad m de servidores ($0 < m \leq n$) pueden estar conectados directamente con el exterior. Llamaremos "puerta de enlace" a un nodo de almacenamiento que dispone de conectividad exterior directa. Si un nodo no es puerta de enlace entonces se le asignará la que está más cerca (que será la puerta de enlace asociada a ese nodo sin conexión exterior directa). Cada archivo que un cliente solicita para su descarga se envía desde el nodo que lo aloja hasta su puerta de enlace asociada, teniendo que atravesar (si es el caso) todos los nodos intermedios que los separa.

Por otra parte, cada nodo i está caracterizado por un parámetro $c_i \in \mathbb{R}^+$ que indica su capacidad de almacenamiento. Cuanto más alto es su valor, más archivos puede albergar el nodo.

Definimos *tráfico estimado* de una instalación como $\sum_{i=1}^{n} c_i \cdot d_{ij}$, donde $d_{ij} \in \mathbb{R}^+$ es la distancia que separa el nodo i de su puerta de enlace asociada (el nodo j). Si el nodo i es, él mismo, puerta de enlace (es decir, $i = j$), entonces $d_{ij} = 0$ (por lo tanto, los nodos que son puerta de enlace no aumentan el tráfico estimado ya que no hacen uso del bus privado).

Se pretende seleccionar los m nodos que actuarán de puerta de enlace hacia el exterior y, para el resto de los nodos, indicar cuáles serán sus puertas de enlace asociadas. Todo ello con el objetivo de minimizar el tráfico estimado de la instalación (para simplificar, supondremos que el valor de m ha sido prefijado).

Por ejemplo, supongamos que se dispone de $n = 7$ servidores de almacenamiento cuyas distancias se representan en la figura 1. Para cada nodo, numerados de 0 a 6, se muestra también su capacidad de almacenamiento (en forma de subíndice). Si el número de puertas de enlace a instalar, valor preestablecido, fuera $m = 3$ y estas se ubicaran en los nodos 1, 3 y 6, el tráfico estimado de la instalación de la figura vendría dado por la expresión $16{\cdot}2{+}15{\cdot}2{,}5{+}13{\cdot}1{+}14{\cdot}(1{+}1) = 110{,}5$; sin embargo, esta no es la mejor elección de nodos con enlace al exterior directo puesto que si se seleccionaran los nodos 1, 4 y 6 se tendría una estimación de tráfico menor ($16{\cdot}2{+}15{\cdot}2{,}5{+}25{\cdot}1{+}14{\cdot}1 = 108{,}5$) y que correspondería en este caso con la mejor disposición posible.

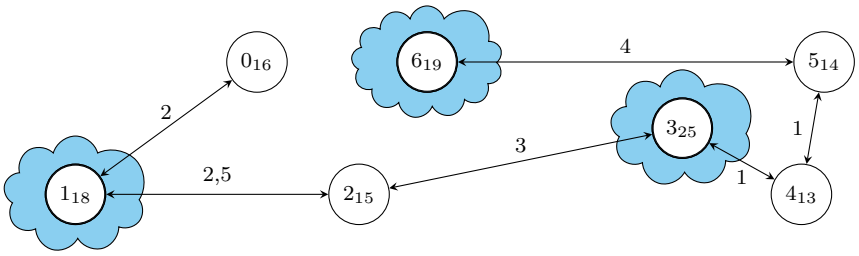

Figura 1: Posible ubicación de tres puertas de enlace (en los vértices dibujados dentro de una nube). Las puertas de enlace asociadas a los nodos 4 y 5 son el nodo 3 (ya que es el nodo más cercano con conexión externa). Las asociadas a los nodos 0 y 2 son el nodo 1. **Esta ubicación de las tres puertas de enlace no es la mejor.**

Puesto que en este caso se puede establecer un orden entre todos los nodos, las distancias entre ellos vendrán dadas en un vector d donde $d_0 = 0$ y cada $d_i \in R^+$ contiene la distancia del nodo i al nodo origen (etiquetada con el número 0), por lo tanto se cumplirá $d_i > d_{i-1}, \forall i : 1 \leq i \leq n-1$. Por otra parte, la capacidad de almacenamiento de cada nodo vendrá dada en un vector c con $c_i \in \mathbb{R}^+$.

La solución de *programación dinámica* para este problema, al que llamaremos *encaminamiento óptimo*, requiere deducir la expresión recursiva que calcula el valor del tráfico estimado para la mejor ubicación posible de las puertas de enlace (tráfico estimado mínimo, met). Considerando n nodos de almacenamiento y m puertas de enlace, $\text{met}(m, n)$ se puede obtener recursivamente de la siguiente manera:

- Si solo hay que colocar una puerta de enlace ($m = 1$), el procedimiento para decidir dónde instalarla consistiría en probar las n posibilidades para quedarse con la mejor. Es decir, la evaluación de $\text{met}(m, n)$ requiere calcular los n valores de tráfico estimado según se instale esa puerta de enlace en cada uno de los n nodos y seleccionar el mínimo de todos ellos (que corresponderá al nodo en el que hay que instalar la puerta de enlace). Llamando $\text{ogw}(0, n)$[1] a dicho mínimo tendremos que $\text{met}(1, n) = \text{ogw}(0, n)$.
  No obstante, para obtener el caso general de la recurrencia conviene generalizar la nueva función introducida de la siguiente manera: $\text{ogw}(k, n)$ es el mínimo tráfico estimado si colocamos, en la mejor ubicación posible, una única puerta de enlace entre los nodos k y $n-1$.[2]

- Si hay que instalar más de una puerta de enlace ($m > 1$), supongamos que la primera de ellas da servicio a los últimos $n-k$ nodos. El tráfico estimado en este tramo es $\text{ogw}(k, n)$. Se trata entonces de asignar las $m-1$ puertas de enlace restantes entre los primeros k nodos (obsérvese que siempre tiene que haber al menos tantos nodos como puertas de enlace por instalar, es decir, $m - 1 \leq k \leq n - 1$), en los que el tráfico estimado es $\text{met}(m-1, k)$. La suma mínima, para todos los posibles valores de k, de esas dos cantidades $s_k = \text{met}(m-1, k) + \text{ogw}(k, n)$ nos permite definir la expresión recursiva que buscamos.

En esta práctica se pide, aplicar el método *divide y vencerás* y su extensión a *programación dinámica* para obtener la mínima estimación de tráfico de la instalación según la mejor ubicación posible de las m puertas de enlace y, a partir de esta información, la puerta de enlace asociada a cada uno de los nodos, es decir, la tabla de encaminamiento.

Para resolver este ejercicio se debe implementar los siguientes algoritmos:[3]

[1] El nombre `ogw` se ha tomado como acrónimo de *one gateway*.

[2] Para facilitar la obtención de la mejor ubicación de la puerta de enlace, la función $\text{ogw}(k, n)$ debería devolver no solo el tráfico estimado mínimo, también el índice del nodo en la que se debe colocar la puerta de enlace.

[3] Como es lógico, las soluciones de todos los algoritmos que muestran la mínima estimación de tráfico deben coincidir.

1. Recursivo sin almacén (ineficiente —también llamada versión ingenua—)
2. Recursivo con almacén (memoización)
3. Iterativo con almacén que hace uso de una tabla para almacenar los resultados intermedios.
4. Iterativo con almacén con complejidad espacial mejorada.
5. Por último deberá obtenerse la tabla de encaminamiento mencionada.

- **Nombres, en el código fuente, de algunas funciones importantes.**

  Para una correcta identificación en el código de las cinco funciones que implementan los algoritmos mencionados en el apartado anterior, deberán ser nombradas de manera preestablecida.

  Siguiendo el mismo orden en el que se han enumerado, los nombres de las funciones de `C++` deben ser: `met_naive`; `met_memo`; `met_it_matrix`; `met_it_vector` y `met_parser`, respectivamente.

  Se deja total libertad para añadir los parámetros que se consideren y también para escoger el tipo de datos de retorno.

  No es imprescindible, para valorar esta práctica, que esas cinco funciones estén en el código pero si no están (o están con otro nombre) entonces se considerará que no han sido desarrolladas, con la consiguiente merma en su calificación.

- **Nombre del programa, opciones y sintaxis de la orden.**

  El programa a realizar se debe llamar **met**. La orden tendrá la siguiente sintaxis:

  `met` *`[-t] [--ignore-naive]`* `-f` *`fichero_entrada`*

  - La opción `-f`, la única de uso obligado, se utiliza para suministrar el nombre del fichero donde está la instancia del problema a resolver. En el caso de que no se suministre o se produzca algún tipo de error al tratar de abrirlo (no existe, sin permisos para abrirlo, etc.) se advertirá con un mensaje de error. No es necesario controlar posibles errores en el contenido del fichero de entrada ya que siempre se ajustará fielmente al formato establecido, que se describe en el apartado "Entrada al programa".
  - Las opciones `--ignore-naive` y `-t` también se describen más adelante.
  - Las opciones no son excluyentes entre sí, ninguna de ellas. Además podrán aparecer en cualquier orden.
  - En el caso de que se haga uso de la orden con una sintaxis distinta a la descrita se emitirá un mensaje de error advirtiéndolo y a continuación se mostrará la sintaxis correcta. Se deben considerar en este caso únicamente las opciones inexistentes; la duplicidad de opciones puede ser ignorada y tratada por tanto como si se hubiera escrito solo una vez. No es necesario indicar todos los errores sintácticos que pueda contener la orden, basta con hacerlo solo con el primer error que se detecte.
  - Para todos los mensajes de error, incluso aquel que informa del uso apropiado de la orden, debe utilizarse la salida estándar de error, es decir, `cerr`.
  - Ante cualquier circunstancia de error en las opciones, el programa deberá terminar advirtiéndolo según se acaba de describir.

- **Salida del programa y descripción de las opciones:**

  En todas las formas posibles de utilizar la mencionada orden, la salida del programa será siempre a la salida estándar,[4] es decir, al terminal si no se redirige, y consistirá, en primer lugar y en la primera línea: la solución obtenida mediante los cuatro algoritmos anteriormente citados: recursivo sin almacén, memoización, iterativo con tabla e iterativo con complejidad espacial mejorada. Cada uno de los cuatro valores se mostrarán en la misma línea, separados por un espacio en blanco y siguiendo ese orden. No obstante, si se incorpora a la orden la opción `--ignore-naive` se deberá sustituir el valor que corresponde a la solución recursiva sin almacén por el carácter guion ('`-`');[5] por lo tanto, en este caso, la primera línea contendrá, en primer lugar dicho carácter y a continuación, los tres resultados restantes.

  En la segunda línea de la salida, se mostrará la tabla de encaminamiento, es decir, los n valores numéricos que corresponden a las puertas de enlace asociadas a cada uno de los nodos. Debe seguirse el mismo formato anterior: un espacio en blanco para separar los distintos valores.

  En la tercera línea deberá mostrarse el mínimo tráfico estimado calculado a partir de la tabla de encaminamiento mostrada en la línea anterior.

  Por otra parte, si se hace uso de la opción `-t` se mostrará, a continuación (cuarta línea y siguientes) las tablas que almacenan los resultados intermedios en las versiones con memoización e iterativa con matriz (en este orden y con los encabezados que pueden verse en los ejemplos de salida). Para los subproblemas que no han sido resueltos en cualquiera de las versiones, se mostrará el carácter guion ('`-`'). El separador entre valores de una misma fila será el espacio en blanco (uno solo), por lo tanto no es necesario que las columnas estén visualmente bien estructuradas.

- **Entrada al programa**

  El problema a resolver se suministrará codificado en un fichero de texto cuyo nombre se recogerá a través de la opción `-f`. Su formato y contenido será:

  - Línea 1 del fichero: Valores n y m, en este orden.
  - Línea 2: Capacidad de almacenamiento de cada nodo: una lista de n números reales positivos incluyendo el 0.
  - Línea 3: Distancias con respecto al nodo origen (nodo 0): una lista de n números reales mayores que cero, excepto el primero que siempre será 0.

  Por tanto, el fichero contendrá 3 líneas que finalizarán con un salto de línea, salvo en todo caso, la última línea. El carácter separador entre números siempre será el espacio en blanco.

  A través de *Moodle* se puede descargar un archivo comprimido con varios ejemplos y sus soluciones, entre ellos está `0.problem` y `1.problem` utilizados a continuación para describir el formato de la salida.

- **Formato de salida. Ejemplos de ejecución.**

  Considérese el fichero `1.problem`; se trata del problema utilizado como ejemplo en la presentación de este problema.

  A continuación se muestran posibles formas de utilizar en la terminal la orden descrita y la salida que el programa debe mostrar (en los ejemplos, las salidas `cout` y `cerr` están dirigidas a la terminal -siguiendo el comportamiento predeterminado-).

  Es imprescindible ceñirse al formato y texto de salida mostrado, incluso en lo que se refiere a los saltos de línea o carácter separador, que en todos los casos es el espacio en blanco (aunque

[4] Véanse los ejemplos de ejecución y los ficheros de *test* suministrados para conocer los detalles de la sintaxis de salida.

[5] Por su elevado coste computacional (evidentemente el programa tampoco deberá hacer la llamada a esta función).

no hay problema si hay más de uno). La última línea debe terminar con un salto de línea (y solo uno).

```
$met -f 1.problem
108.5 108.5 108.5 108.5
1 1 1 4 4 4 6
108.5


$met -t --ignore-naive -f 1.problem
- 108.5 108.5 108.5
1 1 1 4 4 4 6
108.5
Memoization matrix:
0 32 69.5  192  244     -     -
-  0   32 69.5 82.5 108.5     -
-  -    -    -    -     - 108.5
Iterative matrix:
0 32 69.5  192  244   305   419
-  0   32 69.5 82.5 108.5 203.5
-  -    0   32   45    71 108.5


$met -f 0.problem -t --ignore-naive -t
- 0 0 0
0
0
Memoization matrix:
0
Iterative matrix:
0
$
```

```
*ALGUNOS EJEMPLOS DE SITUACIONES DE ERROR*

$met -f
ERROR: missing fiename.
Usage:
met  [-t] [--ignore-naive] -f file


$met
Usage:
met  [-t] [--ignore-naive] -f file


$met -f -t
ERROR: can't open file: -t.
Usage:
met  [-t] [--ignore-naive] -f file


$met -f 0.problem -t -a -b
ERROR: unknown option -a.
Usage:
met  [-t] [--ignore-naive] -f file


$met -f anonexistentfile -t
ERROR: can't open file: anonexistentfile.
Usage:
met  [-t] [--ignore-naive] -f file
```

- **Normas para la entrega.**

ATENCIÓN: Estas normas son de obligado cumplimiento para que esta práctica sea evaluada.

Es imprescindible ceñirse al formato y texto de salida descrito, incluso en lo que se refiere a los saltos de línea o carácter separador, que en todos los casos es el espacio en blanco. No debe añadirse texto o valores adicionales. Si estos requisitos no se cumplen, es posible que falle una de las fases de la evaluación de este trabajo, que se hace de manera automática y en tal caso la práctica tampoco será evaluada.

1. El programa debe realizarse en un único archivo fuente con nombre `met.cc`.
2. Se debe entregar únicamente los ficheros `met.cc` y `makefile` (para generar el archivo ejecutable a través de la orden `make` sin añadir nada más). **Sigue escrupulosamente los nombres de ficheros y funciones que se citan en este enunciado. Solo hay que entregar esos dos archivos (en ningún caso se entregarán archivos de *test*).**
3. Es imprescindible que no presente errores ni de compilación ni de interpretación (según corresponda), en los ordenadores del laboratorio asignado y en el sistema operativo *GNU/Linux*.[6] Se tratará de evitar también cualquier tipo de aviso (*warning*).
4. Todos los ficheros que se entregan deben contener el nombre del autor y su DNI (o NIE) en su primera línea (entre comentarios apropiados según el tipo de archivo).
5. Se comprimirán en un archivo `.tar.gz` cuyo nombre será el DNI del alumno, compuesto de 8 dígitos y una letra (o NIE, compuesto de una letra seguida de 7 dígitos y otra letra). Por ejemplo: `12345678A.tar.gz` o `X1234567A.tar.gz`. **Solo se admite este formato de compresión y solo es válida esta forma de nombrar el archivo.**
6. En el archivo comprimido **no debe haber subcarpetas**, es decir, al extraer sus archivos estos deben quedar guardados en la misma carpeta donde está el archivo que los contiene.
7. La práctica hay que subirla a *Moodle* respetando las fechas expuestas en el encabezado de este enunciado.
8. En la primera entrega solo es necesario que estén implementadas ambas versiones recursivas y, si es el caso, la salida de la tabla de memoización (aunque no hay inconveniente si se realiza algo más). La gestión de opciones del programa debe estar hecha en su totalidad. El resultado que se debe mostrar, en cualquiera de las dos entregas, para los casos aún sin hacer es "?". Por ejemplo, la siguiente salida corresponde a lo mínimo que se pide para la primera entrega:

```
$met -t -f 1.problem
108.5 108.5 ? ?
?
?
Memoization matrix:
0 32 69.5  192  244     -     -
-  0   32 69.5 82.5 108.5     -
-  -    -    -    -     - 108.5
Iterative matrix:
?
```

9. Si se hace la práctica completa para la primera entrega, tambien habrá que realizar la segunda entrega con lo mismo.

[6] Si trabajas con tu propio ordenador o con otro sistema operativo asegúrate de que este requisito se cumple (puedes comprobarlo haciendo uso del compilador *online* de `https://godbolt.org`).