Trabajo práctico 2

Fecha de entrega: viernes 9 de octubre, hasta las 18:00 hs.
Fecha de reentrega: viernes 23 de octubre, hasta las 18:00 hs.

Este trabajo práctico consta de varios problemas y para aprobar el trabajo se requiere aprobar todos los problemas. La nota final del trabajo será un promedio ponderado de las notas finales de los ejercicios y el trabajo práctico se aprobará con una nota de 5 (*cinco*) o superior. De ser necesario (o si el grupo lo desea) el trabajo podrá reentregarse una vez corregido por los docentes y en ese caso la reentrega deberá estar acompañada por un *informe de modificaciones*. Este informe deberá detallar brevemente las diferencias entre las dos entregas, especificando los cambios, agregados y/o partes eliminadas del trabajo. Cualquier cambio que no se encuentre en dicho informe podrá no ser tenido en cuenta en la corrección de la reentrega. Para la reentrega del trabajo **podrían pedirse ejercicios adicionales**.

Para cada ejercicio se pide encontrar una solución algorítmica al problema propuesto y desarrollar los siguientes puntos:

1. Describir detalladamente el problema a resolver dando ejemplos del mismo y sus soluciones.
2. Explicar de forma clara, sencilla, estructurada y concisa las ideas desarrolladas para la resolución del problema usando pseudocódigo (que no es código fuente) y/o lenguaje coloquial.
3. Justificar por qué el procedimiento desarrollado en el punto anterior resuelve efectivamente el problema.
4. Deducir una cota de complejidad temporal del algoritmo propuesto (en función de los parámetros que se consideren correctos) y justificar por qué el algoritmo desarrollado para la resolución del problema cumple la cota dada. Utilizar el modelo uniforme salvo que se explícite lo contrario.
5. Dar un código fuente claro que implemente la solución propuesta. El mismo no sólo debe ser correcto sino que además debe estar bien programado. Si bien no se pide que siga ninguna convención de codificación específica, mínimamente el mismo debe tener nombres de variables apropiados y un estilo de indentación coherente. Se deben incluir las partes relevantes del código como apéndice del informe impreso entregado.
6. Dar un conjunto de casos de test que permitan corroborar en la práctica la correctitud del problema. Debe explicarse por qué los tests cubren los casos posibles del problema. **No** deben ser muchísimos casos ni muy grandes, salvo que el tamaño sea una necesidad para determinar la correctitud.
7. Dar un conjunto de casos de test que permitan observar la performance en términos de tiempo del problema. Para esto se deben desarrollar tanto tests de mejor y peor caso como tests generados sin una intencionalidad (detallando cómo fueron generados).
8. Presentar en forma gráfica y ofrecer conclusiones sobre la comparación entre tiempo estimado de corrida según la complejidad temporal calculada, tiempo medido de corrida para los tests patológicos de peor caso, y tiempo medido de corrida para los tests sin intencionalidad.

Respecto de las implementaciones, Java es el lenguaje sugerido por la Cátedra pero se acepta cualquier lenguaje que permita el cálculo de complejidades según la forma vista en la materia (previa consulta con el docente). Debe poder compilarse y ejecutarse correctamente en las máquinas de los laboratorios del Departamento de Computación.

La entrada y salida debe hacerse por medio de archivos. No se considerará correcta una implementación que no pase los tests que se mencionaron en los puntos anteriores. No se deben considerar los tiempos de lectura/escritura al medir los tiempos de ejecución de los programas implementados.

Será valorada la presentación de un análisis de casos críticos para los problemas presentados, podas posibles de ser pertinentes y caminos alternativos que llevaron hasta la solución presentada.

Deberá entregarse un informe impreso que desarrolle los puntos mencionados. Deberá entregarse el mismo informe en formato digital acompañado de los códigos fuentes desarrollados e instrucciones de compilación, de ser necesarias. Estos archivos deberán enviarse a la dirección `emilioOca@gmail.com` indicando *TP 2* y luego los apellidos de los integrantes del grupo.

Problema 1: Saliendo del Freezer

Estamos en el año 2048 y el Pabellón 0+infinito es todo un éxito. Los alumnos de Algoritmos III están contentos porque van a cursar este cuatrimestre en un aula que está en el piso N, que es el más alto de todos. Con los avances de la ciencia y la tecnología, escaleras y ascensores han quedado obsoletos, y la forma de subir de un piso a otro es a través de portales. El nuevo pabellón tiene P portales, cada uno de los cuales permite subir de un piso A a un piso más alto B (para bajar de piso hay que tirarse con paracaídas al piso 0 y luego volver a subir de ser necesario). Uno de los alumnos, que estaba cursando en el segundo cuatrimestre de 2015 y fue congelado por el método de criogenia, acaba de ser descongelado y no puede creer lo buenos que están estos portales, algo que en su época no existía. Luego de completar todos los censos de estudiantes desde el año 2016 en adelante, este alumno quiere usar la mayor cantidad de portales posibles para llegar al piso N y así seguir cursando Algoritmos III. Diseñar un algoritmo de complejidad $O(N^2)$ para calcular la mayor cantidad de portales que puede utilizar el alumno para subir desde planta baja al piso N (sin tirarse nunca con paracaídas). Se asegura que en toda instancia del problema es posible realizar el recorrido deseado, y que no hay más de un portal que comunique el mismo par de pisos.

Problema 2: Algo Rush

El Pabellón 0+infinito acaba de reabrir sus puertas con la novedad de que ahora tiene P portales que son bidireccionales; asimismo, los paracaídas fueron eliminados por considerarse inseguros. Cada piso del renovado pabellón consta de un pasillo de L metros de longitud, y cada portal permite viajar entre posiciones específicas de los pasillos de dos pisos. Más concretamente, cada portal puede describirse por medio de cuatro enteros no negativos A, D_A, B y D_B, los cuales indican que el portal comunica el piso A, a D_A metros del comienzo del pasillo de ese piso, con el piso B, a D_B metros del comienzo del pasillo de ese piso. Los alumnos, acostrumbrados a los portales que sólo permitían subir, están un poco confundidos al poder utilizar un mismo portal tanto para subir como para bajar entre dos pisos, o incluso para moverse entre posiciones diferentes dentro del pasillo de un mismo piso. Todos los alumnos de Algoritmos III quieren llegar primero a la clase, que es en un aula que está al final del piso N (el más alto del pabellón). Diseñar un algoritmo de complejidad $O(NL + P)$ para calcular la mínima cantidad de segundos que se necesitan para llegar del comienzo del pasillo del piso 0 al final del pasillo del piso N, suponiendo que recorrer un metro requiere 1 segundo, y utilizar cualquier portal requiere 2 segundos (en cualquiera de las dos direcciones posibles). Se asegura que en toda instancia del problema es posible realizar el recorrido deseado, y que no hay más de un portal que comunique las mismas posiciones del mismo par de pisos. No obstante, puede haber más de un portal que comunique el mismo par de pisos, y portales que comuniquen posiciones diferentes dentro del pasillo de un mismo piso.

Problema 3: Perdidos en los Pasillos

El Pabellón 0+infinito, nuevamente remodelado, ahora tiene un diseño basado en un conjunto de M pasillos de distintas longitudes con intersecciones en donde se unen dos o más pasillos. Es así que puede modelarse como un grafo con pesos en los ejes, donde cada eje es un pasillo (de peso igual a la longitud del pasillo), y cada vértices es una intersección o un extremo donde termina un pasillo sin unirse con ningún otro. El decano junto con el director del Departamento de Computación, están preocupados porque talvez existen ciclos en dicho grafo, lo que podría perjudicar a los alumnos al hacer que se pierdan buscando las aulas. Por tal motivo el decano decidió clausurar los pasillos que sea necesario de manera tal que no queden ciclos en el grafo que representa al pabellón. El problema es que cuanto más largo es un pasillo, más costoso es clausurarlo. Diseñar un algoritmo de complejidad $O(M \log M)$ para calcular la mínima suma posible de las longitudes de los pasillos que deberían ser clausurados (eventualmente ninguno) para que no existan ciclos formados por tres o más pasillos en el grafo que representa al pabellón. Se asegura que en toda instancia del problema el grafo que representa al pabellón es conexo.