COM-11103 Estructuras de Datos Avanzadas
Semestre enero – mayo 2023
Grupo 02. Primer Examen Parcial

Miércoles 22 de febrero 2023 Duración:1:15

CU: 201598 Nombre: Ibarrarán Arnaldo Carlos Armando

Documento de reporte.

==========

Reinas:

[5.0] Ejer 1.1) Reporte resultados (copy & paste) y reporte conclusiones.

Estrategia: (idea general del ejercicio)

Diseño: (dónde se modifica el código y métodos de qué objeto o clase se agregan)

Resultados:

Conclusiones:

[3.0] Ejer 1.2) Reporte resultados y reporte sus conclusiones.

Estrategia: (idea general del ejercicio)

Diseño: (dónde se modifica el código y métodos de qué objeto o clase se agregan)

Resultados:

Conclusiones:

[2.0] Ejer 2) Reporte resultados y reporte sus conclusiones.

Estrategia: (idea general del ejercicio)

Diseño: (dónde se modifica el código y métodos de qué objeto o clase se agregan)

Resultados:

Conclusiones:

==========
Mochila.

[3.5] Ejer 1) Reporte resultados.
Estrategia: (idea general del ejercicio)

Para obtener más “mejores soluciones” que la primera, debemos almacenar varias soluciones y compararlas entre sí. Almacenar cada solución es poco práctico, por lo que es mejor guardar las mejores n soluciones. Podemos mantener una lista ordenada de las mejores soluciones y para cada nueva solución que se encuentre, compararla con las existentes en la lista. Si la nueva solución es mejor que alguna en la lista, tomará su lugar y las demás soluciones se desplazarán hacia abajo. Por ejemplo, si la nueva solución tiene una suma de valores mayor que la segunda mejor solución, tomará su lugar y la segunda mejor solución se moverá al tercer lugar, la tercera al cuarto y así sucesivamente. Para hacer esto, es conveniente definir un objeto llamado "Solución" que almacene la suma de pesos, la suma de valores y la representación en binario del subconjunto evaluado.

Diseño: (dónde se modifica el código y métodos de qué objeto o clase se agregan)

- Se agrega clase “Soluciones”
- Se agrega método desplazaDerechaDesde
- Se agrega método imprimeInstrucciones
- Se separa declaración de variables de su inicialización.
- Se cambia N=15 a N=10.
- Se eliminan variables de valor_max, peso_max y strComb_max.
- Se agregan vaiables numEjecuciones, sol, topSols, i, sols, argSemilla y validArgs.
- Se agregan ifs anidados para verificar ejecutar el ejercicio deseado de manera correcta.
- Se envuelve el código de evaluación, excepto la iniciación de parámetros para generación, en un for, para obtener numEjecuciones ejecuciones del ejercicio.
- Se envuelve el for anterior y la iniciación de parámetros para generación, en un if con condición validArgs.
- En la búsqueda del máximo, se elimina el if de comparación individual y se agrega un while para comparar con las topSols soluciones.
- Se agrega un if posterior para, en caso de que la solución pertenezca entre las topSols, se agregue en el lugar correcto y se desplace a las demás.
- Se envuelve la impresión de resultados en un while que cicla a través de las sols encontradas.
- Se agrega un if para imprimir una leyenda en caso de que no se encuentre ninguna solución o no tantas como se deseaban.

Resultados:

Se obtiene en primer lugar el mismo subconjunto que aquel que se obtiene con el algoritmo original con la misma semilla. Todos los lugares siguientes están ordenados descendentemente de acuerdo con el valor total, el peso total se acerca al límite. El tiempo promedio por ejecución fue de 2660210 nanosegundos. Lo impreso por el programa se guardó en el archivo salEj1.txt

Conclusiones:

Se obtuvieron los resultados esperados, lo que demuestra que este método es confiable. Además, resulta muy eficiente para muestras de relativamente pocos artículos, 10 en este caso, y de pocas "mejores soluciones" deseadas, 10 en este caso.

La complejidad del algoritmo original es 2^n, que es la cantidad de posibles soluciones (subconjuntos), pero ahora al comparar cada solución válida con 10 "mejores soluciones", la complejidad O será 2^n <= O <= 2^n * topSols. Esto hace que el algoritmo pueda tener altos costos computacionales por el aumento de artículos o el aumento de cantidad de mejores soluciones pedidas.

En suma, aunque la manera en que está estructurado le hace tener una complejidad entre exponencial y exponencial por lineal, lo cual podría hacer que el aumento de artículos o "mejores soluciones" pedidas quiebre al algoritmo, el algoritmo es confiable, rápido para dígitos de baja magnitud y fácil de escribir y documental, por lo que tiene un bajo costo de implementación y, por lo tanto, se recubre de valor.

[3.5] Ejer 2) Reporte resultados.

Estrategia: (idea general del ejercicio)

Para obtener las mejores topSols soluciones, permitiendo que cada artículo esté 0, 1 o 2 (nVeces) veces, podemos suponer que en nuestro conjunto de artículos existen "clones" de estos. Estos "clones" serán iguales porque tendrán el mismo peso y valor que los originales. Para poder llenar nuestras listas de pesos y valores con los "clones". Como los originales fueron producidos por un generador con una semilla, podemos repetir este ciclo reiniciando nuestro generador y dándole la misma semilla. También deberemos considerar que el número de artículos cambia, entonces deberemos de considerar que el número de artículos se repetirá n veces y en base a eso, definir la longitud de las listas de pesos y valores y la cantidad de subconjuntos que deberán ser evaluados. Ahora, al evaluar las soluciones, tenemos que verificar que no sean equivalentes, es decir que, en una, esté el artículo original sin el clon y, en la otra, el clon sin el original.

Diseño: (dónde se modifica el código y métodos de qué objeto o clase se agregan)

- Se declara la variable nVeces y se le da un valor por defecto de 1, para que el código que hace funcionar este inciso no intervenga con la ejecución del ejercicio 1.
- Se modifica la inicialización de las listas de pesos y valores, dándoles ahora un tamaño de N*nVeces.
- Se modifica el método genPesosYValores. Se le da un nuevo parámetro (int start) con el que llenará las listas de pesos y valores con N artículos comenzando en el índice start.
- Se envuelve la inicialización del generador de números al azar y la llamada al método genPesosYValores en un ciclo for que se repite nVeces.
- Se se le da a N un nuevo valor: N*nVeces.
- Se agrega un método de tipo boolean a la clase “Solución”.
- Se agrega un condicional antes de desplazar para verificar que la nueva solución no sea equivalente.

Resultados:

- Caso numVeces=0
  Se obtiene en cada caso que no hay ninguna solución para todas las ejecuciones. El tiempo promedio por ejecución fue de 333810 nanosegundos. Lo impreso por el programa se guardó en el archivo salEj2-0.txt

- Caso numVeces=1
  Se obtiene en primer lugar el mismo subconjunto que aquel que se obtiene con el algoritmo original con la misma semilla. Todos los lugares siguientes están ordenados descendentemente de acuerdo con el valor total, el peso total se acerca al límite. El tiempo promedio por ejecución fue de 2673020 nanosegundos. Lo impreso por el programa se guardó en el archivo salEj2-1.txt

- Caso numVeces=2
  Se obtiene en primer lugar el mismo subconjunto que aquel que se obtiene con el algoritmo original con la misma semilla. Todos los lugares siguientes están ordenados descendentemente de acuerdo con el valor total, el peso total se acerca al límite. El tiempo promedio por ejecución fue de 2.7371706E8 nanosegundos. Lo impreso por el programa se guardó en el archivo salEj2-2.txt

Conclusiones:

Se obtuvieron los resultados esperados, lo que demuestra que este método es confiable. Además, en el caso nVeces=0, resulta muy eficiente con un tiempo promedio de ejecución que no supera la milésima de segundo. Para el caso nVeces=1, con la misma cantidad de artículos y soluciones

deseadas, la eficiencia sigue siendo buena y las impresiones son casi inmediatas, aunque toma alrededor de 10 veces el tiempo promedio por ejecución a lo que toma el primero, pero no sobrepasa la centésima de segundo. En el caso nVeces=2, la reducción en velocidad de las impresiones es notable a simple vista y considerable matemáticamente, pues este método toma alrededor de 100 veces el tiempo que toma el caso 1 y 1000 veces el tiempo que toma el caso 0, por lo que supera la décima parte se segundo sin llegar a este. Aun así, el tiempo que toma para imprimir sigue siendo relativamente poco.

La complejidad del algoritmo original es 2^n, que es la cantidad de posibles soluciones (subconjuntos), pero ahora al comparar cada solución válida con 10 “mejores soluciones”, la complejidad O será 2^n <= O <= 2^n * topSols.

En el caso nVeces=0, habría solo un subconjunto que analizar, el vacío, pero este no es una solución, por lo que el algoritmo lo que evita operación alguna, y hace que la complejidad tienda a cero, por lo que el tiempo de ejecución siempre será muy pequeño sin importar la cantidad de artículos dados o el número de “mejores soluciones” pedidas. Sin embargo, lo devuelto es algo que se podía obtener trivialmente y no aporta información nueva.

En el caso nVeces=1, lo obtenido es equivalente a lo del ejercicio 1, por lo que las conclusiones son las mismas.

En el caso nVeces=2, la complejidad O cambia a 2^(2n) <= O <= 2^(2n) * topSols. Por lo que se vuelve un algoritmo muy lento cuando el número de artículos sigue siendo relativamente pequeño. Se usó una muestra de 10 artículos, pero el tiempo que se hubiera requerido para evaluar 11 hubiera sido 4096 veces mayor. Se convierte en un algoritmo muy caro computacionalmente con un aumento mínimo de artículos.

En suma, en el caso nVeces=0 tenemos un buen algoritmo, pues, aunque su resultado es obvio, es fácil de implementar, muy baráto computacionalmente y fácil de modificar para tratar con casos más relevantes. En el caso nVeces=1 tenemos un buen algoritmo por las mismas razones que lo teníamos en el ejercicio 1. En el caso nVeces=2 tenemos un algoritmo funcional y fácil de implementar, por lo cual, a pesar de las limitaciones computacionales que tiene, le otorga valor.

[3.0] Ejer 3) Reporte sus observaciones y conclusiones.

Estrategia: (idea general del ejercicio)

Para poder comparar, necesitamos alguna medida, alguna cuantificación. Podremos evaluar cada algoritmo si tomamos en cuenta los siguientes aspectos:

- Funcionalidad (calidad de datos regresados)
- Complejidad
- Tiempo de ejecución

- Magnitudes de datos que pueden manejar
- Facilidad de implementación

Ahora simplemente describiremos los algoritmos de cada ejercicio basándonos en estos aspectos.

Diseño: (dónde se modifica el código y métodos de qué objeto o clase se agregan)

- Se declaran t0 y t1.
- Se evalúa el tiempo al iniciar las ejecuciones (t0) y al terminarlas todas (t1).
- Se agrega una impresión del tiempo promedio de ejecución, la cual se obtiene al dividir (t1-t0) entre el número de ejecuciones.

Resultados:

Para el programa del ejercicio 1 (algoritmo 1) tenemos:

- Funcionalidad (calidad de datos regresados): Buena, muy confiable, pero para un problema específico.
- Complejidad: Buena, no es muy escalable.
- Tiempo de ejecución: Buena.
- Cantidad de datos que pueden manejar: Poca, solo puede tratar con magnitudes pequeñas.
- Facilidad de implementación: Buena

Para el programa del ejercicio 2 (algoritmo 2) tenemos:

- Funcionalidad (calidad de datos regresados): Buena, muy confiable y ofrece resultados para tres casos distintos de un mismo problema.
- Complejidad: Regular, casi no es escalable, pues uno de los casos tomará demasiado tiempo.
- Tiempo de ejecución: Regular, pues es buena en dos de tres casos y mala en uno.
- Cantidad de datos que pueden manejar: Poca, solo puede tratar con magnitudes pequeñas.
- Facilidad de implementación: Buena

Comparación:

- Funcionalidad (calidad de datos regresados): Algoritmo 2 es mejor
- Complejidad: Algoritmo dos es mejor en primer caso, igual en el segundo y peor en el tercero
- Tiempo de ejecución: Algoritmo dos es mejor en primer caso, igual en el segundo y peor en el tercero.
- Magnitud de datos que pueden manejar: Algoritmo dos es mejor en primer caso, igual en el segundo y peor en el tercero.
- Facilidad de implementación: Algoritmo 1 es mejor.

Conclusiones:

Los algoritmos son de calidad equivalente en cuanto a su complejidad, tiempo de ejecución y magnitud de datos que son capaces de manejar, ya que el algoritmo 2 posee un caso que devuelve la misma salida que la que imprime el algoritmo 1. El algoritmo uno destaca en cuanto a su facilidad de implementación y el 2 en cuanto a su funcionalidad. Es por todo esto que resulta claro que no podemos pronunciar a un algoritmo mejor que otro, pues todo depende de lo que se necesita al implementarlo.