Universidad Rey Juan Carlos

ETSII Escuela Técnica Superior de Ingeniería Informática

Algoritmos Avanzados

-Práctica 2-

Algoritmos Heurísticos

Curso 2022/2023

GII

Alberto Radlowski Nova
Pablo Rodea Piornedo

Especificación del problema

Precondición

- Array de entrada de cs debe estar ordenados de manera descendente.
- Los números de “ds” y “cs” no deben ser negativos.
- La longitud de los vectores “cs” y “ds” debe ser la misma.
- Los números introducidos han de ser enteros.
- Los vectores “ds” y “cs” no pueden ser vacíos (al menos para el algoritmo 2).

Postcondición

- Función Objetivo: maximizar cantidad de datos procesados.
- El valor de salida debe ser no negativo.
- El valor de salida debe ser un número entero.

Diseño e Implementación de Algoritmos Heurísticos

Algoritmo Heurístico - enunciado práctica

```
public static int procesar (int[] ds, int[] cs){

    int resultado = 0;
    int indice = 0;

    for(int i = 0; i < ds.length; i++){
        //Si no reinicia
        if(cs[indice] > ds[i]){
            resultado += ds[i];
            indice++;
        }
        //Si reinicia
        else{
            indice = 0;
        }
    }
    return resultado;
}
```

Este es el algoritmo que venía detallado en el enunciado de la práctica, cuyo criterio de reinicio del vector cs es simplemente que el vector ds en una posición [i] sea en cualquier momento menor que el vector cs en la misma posición [i].

Algoritmo Heurístico - alternativo

```
public static int procesar2 (int[] ds, int[] cs){

    int resultado = 0;
    int indice = 0;
    int mediads = 0;
    int mediacs = 0;

    //Media del array ds
    for(int i = 0; i < ds.length; i++){
        mediads += ds[i];
    }
    mediads = mediads / ds.length;

    //Media cs
    for(int i = 0; i < cs.length; i++){
        mediacs += cs[i];
    }
    mediacs = mediacs / cs.length;

    for(int i = 0; i < ds.length; i++){
        //No reinicia
        if(cs[indice] > ds[i]){
            resultado += ds[i];
            indice++;
        //Si es el ultimo dia no se reinicia(coge lo que puede)
        } else if (i == ds.length - 1) {
            resultado += Math.min(cs[indice], ds[i]);
        }
        else{
            //Si es un día que merece la pena no reiniciar
            if(ds[i] > mediads && cs[indice] > mediacs){
                resultado += Math.min(ds[i], cs[indice]);
            }
            //Reinicia Servidor (la recompensa es muy baja si
no se hace)
            else{
                indice = 0;
            }
        }
    }
```

Esta versión del algoritmo ha sido planteada con el objetivo de producir mejores soluciones que la del descrito en el enunciado.

La forma en la que hemos producido nuestro algoritmo ha sido seleccionando un criterio con el cual determinar la eficiencia del mismo en diferentes puntos, en nuestro caso hemos calculado la media del vector ds y cs, si el valor de turno en el vector ds es menor que la media, se considera que no es rentable seguir operando y se manda un reinicio, análogamente se reinicia según la media de cs.

Si por el contrario considera que es rentable porque los valores de ds[i] y cs[i] son mayores que sus respectivas medias sumará al resultado de datos procesados el menor de los valores de éstos.

Por último es importante comentar que hemos añadido un par de líneas de código que mejoran la eficiencia, las cuales evitan que si es el último día (último elemento de la lista) no se reinicie aunque el valor sea menor a nuestra media calculada ya que cualquier reinicio en este día obtendría un resultado peor.

Hemos asumido que podemos hacer una excepción en este caso y así mejorar algo la eficiencia.

Experimentación con la optimalidad de los algoritmos

Material del experimento

- Los algoritmos heurísticos que hemos usado se llaman “procesar” y “procesar2”.
- Datos Algorex:
    - Criterio de experimentación: optimalidad
    - Objetivo del problema: maximizar
    - Tipo de medidas: relativas
    - Tamaño vectores “cs” y “ds”: 30
    - Rango de valores de 0 a 30
    - Ejemplos ejecutados: 1000
    - Vector de “ds” con números mezclados
    - V de “cs” ordenado de manera descendente
    - Se permiten repeticiones en ambos

Conclusión

Ambos son algoritmos inexactos puesto que se trata de algoritmos heurísticos.

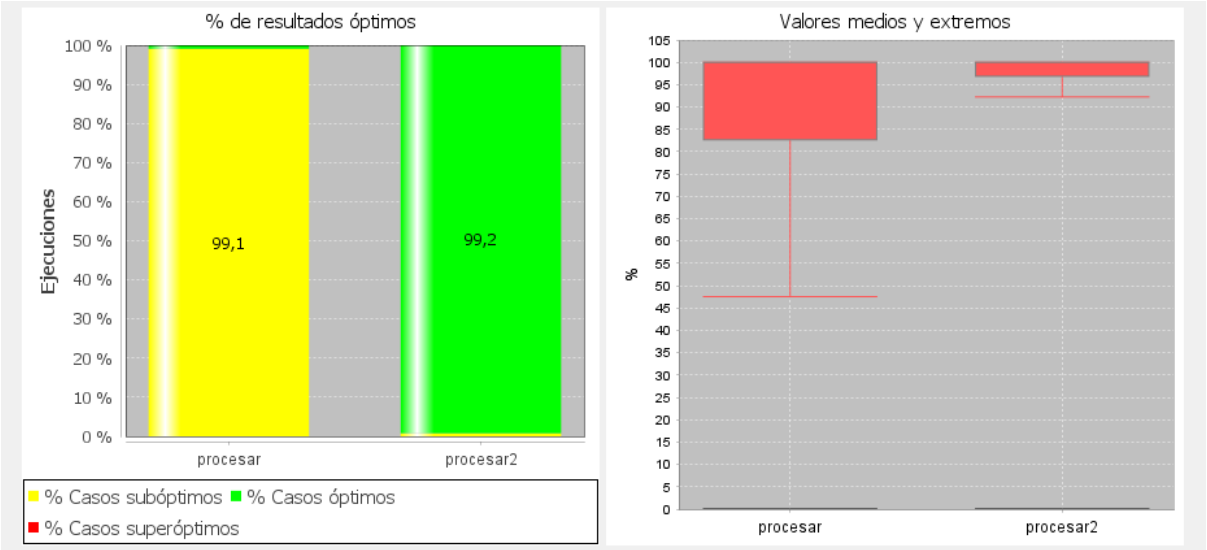
No calculan la solución óptima exacta sino una aproximación a ésta en la mayoría de los casos.

Hemos podido comprobar con los resultados que “procesar” se queda la mayoría de las veces bastante lejos de la solución óptima exacta en cambio “procesar2” se acerca bastante generalmente.

Evidencias

Medidas	procesar	procesar2
Núm. ejecuciones	1000	1000
Núm. ejec. válidas del método	1000	1000
Núm. ejec. válidas en total	1000	1000
% Soluciones subóptimas	99,10 %	0,80 %
% Soluciones óptimas	0,90 %	99,20 %
% Soluciones sobreóptimas	0,00 %	0,00 %
% Diferencia media subóptima	82,66 %	96,88 %
% Diferencia máxima subóptima	47,55 %	92,25 %
% Diferencia media sobreóptima	0,00 %	0,00 %
% Diferencia máxima sobreóptima	0,00 %	0,00 %

En esta tabla se puede observar como en el 99% de los casos “procesar2” obtiene un resultado mejor que “procesar”
Además en las veces que “procesar2” queda por debajo es por muy poca diferencia con respecto a “procesar” por lo que “procesar2” es generalmente un algoritmo que maximiza mejor el resultado.



En este gráfico se puede observar de manera más clara lo comentado anteriormente que “procesar2” es mejor en el 99% de los casos y cuando no, difiere un 8% aproximadamente según el bigote de “procesar2”.

En cambio “procesar” llega a tener resultados incluso un 50% por debajo que “procesar2”

Contraejemplos “procesar”

Algorex:

ds -> {1,18,25,27,5,12,29,24,28,14,5,27,24,8,12,16,16,25,29,10,6,10,27,13,0,17,19,26,14,8}

cs -> {26,25,24,23,23,23,21,21,20,18,17,17,17,14,13,13,12,10,10,10,9,8,7,5,3,3,2,2,1,1}

Con estos datos de entrada de Algorex “procesar” obtiene un resultado subóptimo = 252, muy lejos del resultado exacto porque según su criterio de reinicio desperdicia mucha cantidad de datos aunque cs sea simplemente menor a ds en 1 unidad ej: ds = 20 y cs = 19 -> según el algoritmo habría que reiniciar pero no es para nada óptimo porque se desperdicia la oportunidad de procesar 19 datos.

Deducido:

ds -> {7,2,4,4,9,2,3,6,6,4}
cs -> {9,7,7,6,6,6,3,1,0,0}

En este contraejemplo más sencillo ocurre lo mismo que en el contraejemplo anterior, reinicia de manera ineficiente aunque tenga la posibilidad de procesar una cantidad considerable de datos dando como resultado 32 que se aleja bastante de lo máximo posible

Contraejemplos “procesar2”

Deducido:

ds -> {9, 4, 4, 1, 8, 8, 5, 5, 5, 2}
cs -> {10, 9, 7, 7, 6, 5, 4, 4, 3, 1}

En este ejemplo más sencillo “procesar2” es subóptimo con respecto a “procesar” porque todos los números están muy cerca de la media y escoja el que escoja la cantidad de datos perdidos va a ser grande.

Algorex:

ds -> {10,25,8,29,17,23,28,22,11,5,19,0,11,6,16,22,12,22,20,1,1,23,16,24,26,27,8,1,5,11}

cs -> {29,29,29,28,24,22,20,19,18,16,15,14,14,13,12,12,11,10,9,9,7,7,7,5,2,1,1,1,0,0}

Aquí se puede observar de forma análoga que todos los números de ds y cs están muy cerca de la media y por tanto no existe una decisión óptima en este caso para este algoritmo que no haga que se pierda una cantidad considerable de datos procesados.

Conclusiones

Hemos tenido una serie de incidencias a la hora de realizar el algoritmo heurístico y a la hora de probar este mismo con AlgorEx.

La dificultad a la hora de la implementación ha venido dada debido a que en una primera instancia hemos intentado copiar nuestro vector cs en un array que habíamos creado nuevo, ordenando este nuevo array. Más tarde nos dimos cuenta que esta no era una forma eficiente de realizar el algoritmo y derrotaba el propósito de la heurística.

Finalmente elegimos operar simplemente con los índices reiniciando sus posiciones cuando era necesario reiniciar el servidor.

Por último hemos encontrado problemas a la hora de realizar la experimentación con AlgorEx ya que los resultados que hemos obtenido de la comparación de ambos algoritmos salían increíblemente dispares, lo que nos ha hecho dudar en un primer momento, pero lo hemos acabado atribuyendo a que el segundo algoritmo era mucho más eficaz que el primero.