Universidad Rey Juan Carlos

ETSII Escuela Técnica Superior de Ingeniería Informática

Algoritmos Avanzados

-Práctica 3a-

Tecnicas de busqueda
Vuelta Atrás

Curso 2022/2023

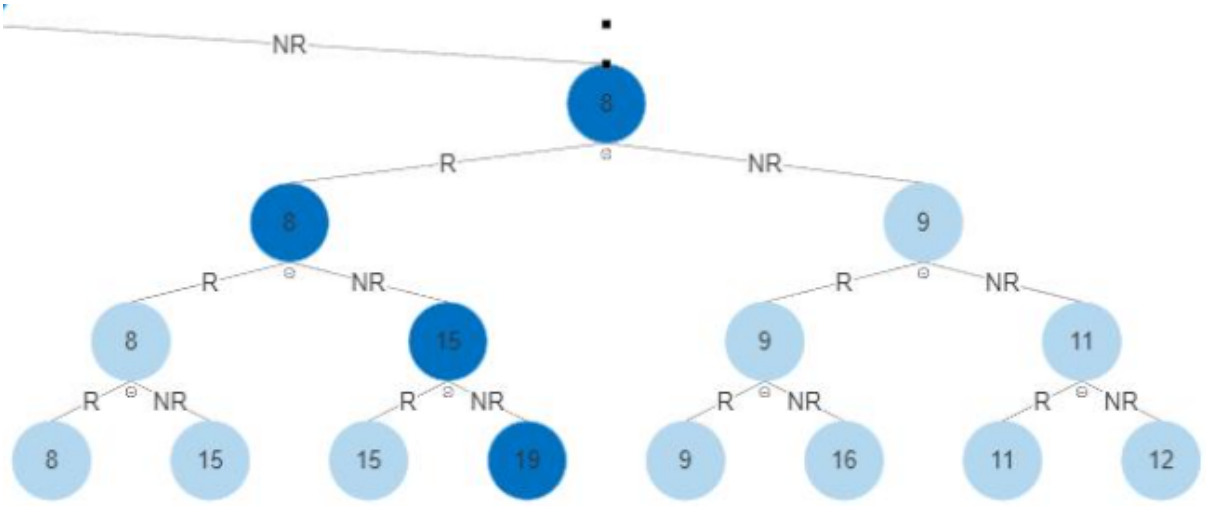
GII

Alberto Radlowski Nova
Pablo Rodea Piornedo

Técnica de vuelta atrás

Diseño de árbol de búsqueda

árbol (ÁRBOL COMPLETO en el Link)



Esta imagen es la parte derecha de nuestro árbol, el árbol completo tiene 4 niveles, tantos niveles como la longitud de los vectores que obtenemos por enunciado.

El árbol como se puede apreciar es un árbol binario el cual contempla en cada nodo la opción que tiene de reiniciar el sistema o de continuar sin reinicio.

Comprobación de validez

En nuestro método solamente comprobamos las soluciones definitivas (de los nodos hoja) para saber cual es la óptima, ya que según el enunciado no necesitamos realizar una comprobación de validez parcial en cada nodo (No es el problema de la mochila por ej).

Código de vuelta atrás

```
public static int procesar3 (int[] ds, int[] cs){
    int [] combinaciones = new int[ds.length];
    //Llama a metodo auxiliar
    return aux(ds, cs, 0, combinaciones, 0);
}
```

Método auxiliar

```
public static int aux(int[] ds, int[] cs, int etapa, int [] combinaciones,
int solSubArbol){

   for(int i=0; i<=1; i++){
       //Array que guarda de forma temporal la solucion del subarbol y se
actualiza en cada iteracion
       combinaciones[etapa] = i;
       //Si es una hoja
       if(etapa == ds.length-1){
           int resultado = 0;
           int indice = 0;
           //Se comprueba si la combinacion del subArbol actual es mejor
que la solucion optima hallada hasta ahora
           for(int j=0; j<cs.length; j++){
               if(combinaciones[j] == 1 ){
                   resultado += Math.min(ds[j], cs[indice]);
                   indice++;
               }else{
                   indice = 0;
               }
           }
           if(resultado > solSubArbol){
               solSubArbol = resultado;
           }

       }else{
           //Si no es una hoja, recorre de forma recursiva el subarbol
hasta la profundidad de ds.length-1
           solSubArbol = aux(ds, cs, etapa+1, combinaciones, solSubArbol);
       }
   }
   //Marca que la combinación ha sido evaluada
   combinaciones[etapa] = -1;
   //Devuelve el resultado del subarbol
   return solSubArbol;
}
```

Comparación de optimalidad

Material del experimento

- **Algoritmo “Procesar”**, es el algoritmo propuesto por el profesor el cual reinicia el servidor en el momento que cs[indice]<ds[i].

- **Algoritmo “Procesar2”**, en este caso es un algoritmo más optimizado ya que calcula las medias de cs y ds y solo reinicia cuando se queda por debajo de estos valores.

- **Algoritmo “Procesar3”**, aquí hemos utilizado backtracking para resolver un árbol binario de búsqueda, para ello, se comprueban todas las combinaciones posibles hasta llegar a un nodo hoja y va arrastrando el valor hacia arriba de la llamada recursiva actualizándose en caso de encontrar una combinación mejor.

  Estas combinaciones están definidas en el array “combinaciones” cuyo índice “i” se actualiza a 1 si se está probando esa combinación (sin reinicio), a 0 si ha reiniciado y a -1 si ya la ha probado y está volviendo a subir por la rama del árbol.

  Rangos de valores: Tamaño **“cs”** y **“ds”** = 15, **ds** con valores repetidos sin orden, y **cs** con valores repetidos ordenados de forma descendente, número de ejemplos = 1000.

Conclusión

- **“Procesar”** y **“Procesar2”** son algoritmos heurísticos, por tanto son inexactos ya que no realizan una búsqueda exhaustiva de todas las combinaciones posibles si no que intentan acercarse a la solución óptima mediante una aproximación.

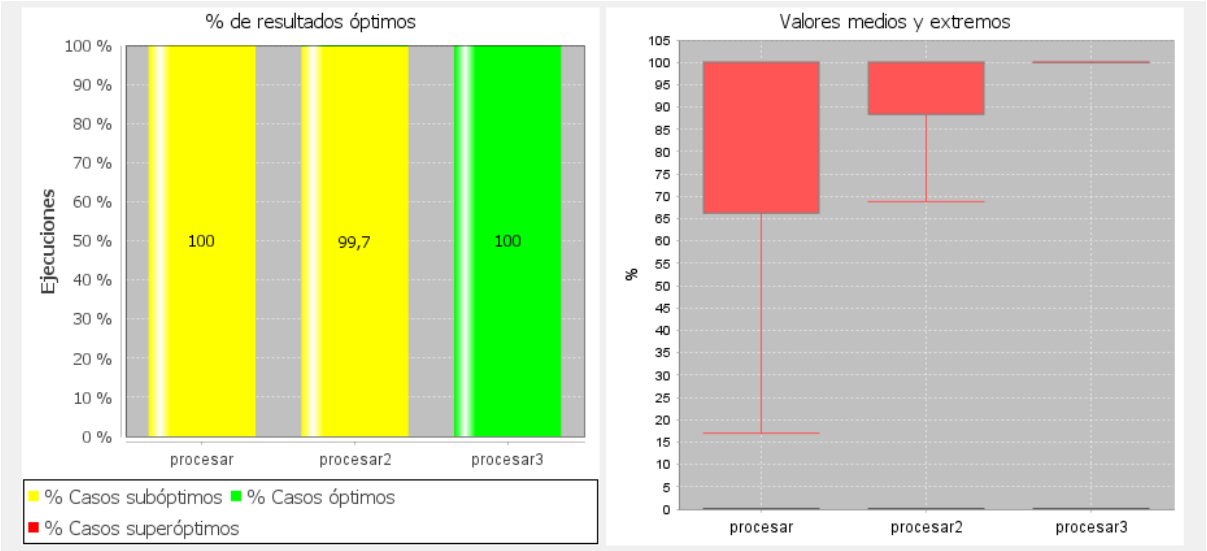
- **“Procesar3”** en cambio sí que es un algoritmo exacto ya que mediante el uso de backtracking prueba todas las combinaciones posibles y termina hallando la solución óptima en el 100% de los casos.

Evidencias

Medidas	procesar	procesar2	procesar3
Núm. ejecuciones	1000	1000	1000
Núm. ejec. válidas del método	1000	1000	1000
Núm. ejec. válidas en total	1000	1000	1000
% Soluciones subóptimas	100,00 %	99,70 %	0,00 %
% Soluciones óptimas	0,00 %	0,30 %	100,00 %
% Soluciones sobreóptimas	0,00 %	0,00 %	0,00 %
% Diferencia media subóptima	66,25 %	88,36 %	0.00 %
% Diferencia máxima subóptima	16,98 %	68,83 %	0,00 %
% Diferencia media sobreóptima	0,00 %	0,00 %	0,00 %
% Diferencia máxima sobreóptima	0,00 %	0,00 %	0,00 %

En la tabla podemos ver como **“procesar3”** el algoritmo que usa backtracking es óptimo el 100% de las veces en cambio los otros 2 no.

Tambíen se puede observar que a pesar de que **“Procesar”** y **“Procesar2”** no son exactos, **“Procesar2”** es un mejor algoritmo heurístico ya que la media de su solución con respecto al backtracking de **“Procesar3”** es de un 88% de la solución óptima en cambio **“Procesar”** se queda en el 66% de media.



En el gráfico podemos observar como **“Procesar3”** es siempre óptimo.

Además de nuevo se puede observar claramente como **“Procesar2”** da mejores resultados que **“Procesar”** ya que según los bigotes del gráfico **“Procesar”** llega a dar valores incluso de solo el 17% de la solución óptima.

En cambio el algoritmo **“Procesar2”** da resultados menos dispares acercándose bastante generalmente a la solución óptima (88%) y cuyos peores resultados se alejan a solo el 68% de la solución óptima de **“Procesar3**”

También se puede observar como **“Procesar”** ha obtenido la solución óptima en el 0% de los casos y **“Procesar2”** en el 0,3% de los 1000 casos probados.

Conclusiones

Como conclusión final de la realización de la práctica hemos sacado en claro que hay diversas formas de realizar este algoritmo, la definitiva no ha sido la primera que hemos intentado.

Este formato de resolución ha sido nuestro preferencial ya que con la herramienta debug hemos podido observar en tiempo real gracias a nuestro vector posición, en que iteración del árbol de búsqueda se encuentra el programa, lo cual ha hecho mucho más sencilla su realización.