Tema 6. Algoritmos voraces

Tema 6. Algoritmos voraces

Competencias

- CTEC3. Capacidad para evaluar la complejidad computacional de un problema, conocer estrategias algorítmicas que puedan conducir a su resolución y recomendar, desarrollar e implementar aquella que garantice el mejor rendimiento de acuerdo con los requisitos establecidos.

Tema 6. Algoritmos voraces

Objetivos del tema

- Estudiar el método de los Algoritmos Voraces (método greedy o método devorador).
- Aplicabilidad del método en problemas de optimización cuando la solución se puede obtener a trozos.
- Hay que demostrar cuando la solución obtenida es óptima (no siempre lo es).
- No hay vuelta atrás cuando se tiene un trozo de solución.
- A veces se usan para obtener soluciones subóptimas.
- Ejemplos ilustrativos.

Tema 6. Algoritmos voraces

Introducción

- Mantiene la idea de la división pero ahora se divide la solución.
- Se aplica en problemas cuya solución se puede obtener a trozos.
- Cada trozo se obtiene buscando el óptimo entre los trozos aún no seleccionados.
- Se suele aplicar en problemas de optimización.
- Cada vez que se selecciona un trozo, éste ya es definitivo sin verse afectado por lo que ocurra después.
- Ejemplo del problema del cambio, resaltando que no es óptimo.

Tema 6. Algoritmos voraces

Algoritmo *cambio*(n)

inicio

$C = \{100, 50, 20, 10, 5, 2, 1\}$

$S \leftarrow \phi$ *Solucion*

$s \leftarrow 0$ *suma parcial*

mientras $s \neq n$ **hacer**

$x \leftarrow maximo(C)$ *tal que* $s + x \leq n$

si *existe*(x) **entonces**

$S \leftarrow S \cup \{x\}$

$s \leftarrow s + x$

sino

devolver *no encuentro solucion*

finsi

finmientras

fin

Tema 6. Algoritmos voraces

El método general

- Problema de optimización en el que la solución se construye partiendo de un conjunto de candidatos (monedas).
- Se usan dos conjuntos:
  - El primero contiene candidatos evaluados y seleccionados.
  - El segundo contiene los evaluados y rechazados.
- Una función comprueba si los candidatos seleccionados hasta el momento constituyen una solución del problema. (Las monedas seleccionadas ya son iguales a la cantidad que se quiere conseguir).
- Otra función comprueba si hay candidatos puede mejorar la solución haciendo crecer el conjunto. (Ver monedas que se pueden seleccionar).

Tema 6. Algoritmos voraces

El método general(II)

- Función de selección del mejor de los candidatos que pueden mejorar la solución (seleccionar la moneda de más valor dentro de las seleccionables).
- Función objetivo a optimizar que proporciona el valor de la solución encontrada (número de monedas empleadas en el cambio).
- Tiempo de ejecución $O(n^2)$ u $O(n^3)$:
    - n candidatos (se reducen en 1 en cada iteración).
    - Función de selección y objetivo constante o lineal.
    - Número de candidatos que forman la solución.

Tema 6. Algoritmos voraces

Algoritmo *voraz*(C)

inicio

  $S \leftarrow \phi$ *Solucion*

  mientras $C \neq \phi$ **y no** *solucion*(S) **hacer**

    $x \leftarrow$ *seleccionar*(C)

    $C \leftarrow C - \{x\}$

    si *viable*$(S \cup \{x\})$ **entonces**

      $S \leftarrow S \cup \{x\}$

    finsi

  finmientras

  si *solucion*(S) **entonces**

    devolver S

  sino

    devolver *No hay solucion*

  finsi

fin

Tema 6. Algoritmos voraces

Elementos de un algoritmo voraz en el caso del problema del cambio

- Los candidatos son el conjunto de monedas disponibles, suponiendo que no hay límite para ninguna de ellas.
- La función de solución comprueba si el valor de las monedas seleccionadas es igual al valor que hay que conseguir.
- Un conjunto de monedas es viable si no sobrepasa la cantidad buscada.
- La función de selección elige la moneda de más valor que quede en el conjunto de candidatos.
- La función objetivo contabiliza el número de monedas usadas en la selección.

Tema 6. Algoritmos voraces

El problema de la mochila

- Problema:
  - Mochila de volumen V.
  - Materiales divisibles m_i de volumen v_i y de precio unitario p_i
  - Llenar la mochila con máximo coste.
- Solución:
  - Llenar la mochila comenzando con el material de mayor precio y cuando se agota éste, si queda volumen disponible, seleccionar el de siguiente mayor precio, y así hasta que se llene la mochila.
  - El máximo se puede seleccionar en cada iteración $O(kn)$, o se pueden ordenar los materiales según el precio $O(nlogn)$.

Tema 6. Algoritmos voraces

Algoritmo *Mochila*$(n, V; D;)$
inicio
    $resto \leftarrow V$
    para i **de** 1 **a** n **hacer**
        $D(i).usado \leftarrow$ "*nada*"
    finpara
    repetir
        $precioMaximo \leftarrow 0$, $materialMaximo \leftarrow 0$, $materialDisponible \leftarrow falso$
        para i **de** 1 **a** n **hacer**
            si $D(i).usado =$ "*nada*" **entonces**
                $materialDisponible \leftarrow cierto$
                si $D(i).precio > precioMaximo$ **entonces**
                    $precioMaximo \leftarrow D(i).precio$
                    $materialMaximo \leftarrow i$
                finsi
            finsi

        finpara

Tema 6. Algoritmos voraces

Comprobamos si el material de maximo coste cabe en la mochila

si $materialDisponible = cierto$ **entonces**

si $resto \geq D(materialMaximo).volumen$ **entonces**

$D(materialMaximo).usado \leftarrow$ "*total*"

$resto \leftarrow resto - D(materialMaximo).volumen$

sino

$D(materialMaximo).usado \leftarrow$ "*parcial*"

$resto \leftarrow 0$

finsi

finsi

hasta que $resto = 0$ **o** $materialDisponible = falso$

fin

Tema 6. Algoritmos voraces

Minimización del tiempo de espera

- Problema:
  - En un determinado servicio se han de atender a n clientes, y de antemano se conoce el tiempo t_i que se atiende a cada uno. ¿En qué orden deben ser atendidos los clientes para que la suma de los tiempos de todos los clientes que están en el servicio (tiempo de espera y tiempo de atención) sea mínima?.
- Solución:
  - La estrategia voraz a seguir consiste en atender en cada paso al cliente no atendido con menor tiempo de atención.

Tema 6. Algoritmos voraces

Minimización del tiempo de espera (II)

- Demostración:
  - Supongamos que un algoritmo va construyendo la secuencia óptima paso a paso.
  - Después de haber calculado la secuencia óptima $(i_1, i_2, ..., i_m)$ para los m primeros clientes, supongamos que se añade a la misma el cliente j, con tiempo de atención t_j. El crecimiento del tiempo total en el servicio T será:
    $t_{i1} + t_{i2} + t_{i3} + \cdots + t_{im} + t_j$
  - Para minimizar este crecimiento, dado que un algoritmo voraz no reconsidera sus decisiones y los tiempos previos seleccionados ya no se pueden cambiar, lo único factible es el minimizar t_j
- Ejemplo de la cinta de casette.

Tema 6. Algoritmos voraces

Minimización del tiempo de espera (III)

- Ejemplo: Supongamos que se tienen tres clientes, y los tiempos de atención son $t_1 = 5$, $t_2 = 10$, $t_3 = 3$.
- Las posibilidades de atención son:
  - 1, 2, 3. El tiempo total en servicio sería: $5 + (5 + 10) + (5 + 10 + 3) = 38$.
  - 1, 3, 2. El tiempo total en servicio sería: $5 + (5 + 3) + (5 + 3 + 10) = 31$.
  - 2, 1, 3. El tiempo total en servicio sería: $10 + (10 + 5) + (10 + 5 + 3) = 43$.
  - 2, 3, 1. El tiempo total en servicio sería: $10 + (10 + 3) + (10 + 3 + 5) = 41$.
  - 3, 1, 2. El tiempo total en servicio sería: $3 + (3 + 5) + (3 + 5 + 10) = 29$.
  - 3, 2, 1. El tiempo total en servicio sería: $3 + (3 + 10) + (3 + 10 + 5) = 34$.
- La secuencia óptima es la $3, 1, 2$ cuyo tiempo total en el servicio es 29.

Tema 6. Algoritmos voraces

Planificación de tareas a plazo fijo

- Problema:
  - Se tienen n tareas (t_i) y cada una se realiza en una unidad de tiempo y solo genera beneficio (b_i) si se ejecuta antes de un plazo (p_i).
  - Calcular secuencia de tareas de más beneficio.

Tema 6. Algoritmos voraces

Planificación de tareas a plazo fijo(II)

- Un conjunto de tareas es factible si al menos una secuencia del conjunto se puede realizar en plazos.
- Se puede obtener una solución seleccionando en cada paso la tarea aún no seleccionada de mayor beneficio, siempre que la secuencia resultante sea factible.
- Un conjunto T de tareas es factible si y solo si la permutación de ese conjunto de tareas en orden creciente de plazos de ejecución también lo es.
- Se demuestra que la solución que obtiene una permutación ordenada en orden creciente de plazos es la solución óptima.

Tema 6. Algoritmos voraces

Algoritmo *secuencia*$(p, n; ; k, s)$

inicio

$p(0) \leftarrow 0$

$s(0) \leftarrow 0$

$k \leftarrow 1$

$s(1) \leftarrow 1$ `La tarea 1 se selecciona siempre`

para i **de** 2 **a** n **hacer** `en orden decreciente de los beneficios`

`busca tarea y prueba insertarla sin que las ya seleccionadas`

`queden fuera de plazo y salgan de la solucion`

$r \leftarrow k$ `almacena la última tarea seleccionada`

mientras $p(s(r)) > p(i)$ **y** $p(s(r)) \neq r$ **hacer**

`la primera parte del predicado busca la posición de inserción`

`la segunda comprueba si la tarea r, ya colocada, puede ser desplazada`

`sin violar plazos`

$r \leftarrow r - 1$

finmientras

Tema 6. Algoritmos voraces

```
Encuentra la posición de inserción comparando con las
ya seleccionadas e inserta ó inserta porque no se cumplirían plazos
```

si $p(s(r)) \leq p(i)$ **y** $p(i) > r$ **entonces**

```
la primera parte del predicado comprueba que ha encontrado
la posición de inserción
la segunda comprueba que la tarea se inserta sin violar su plazo
se inserta i en la posicion r+1
```

para j **de** k **a** $r+1$ **inc** -1 **hacer**

```
Desplaza una posición las tareas desplazables
```

$s(j+1) \leftarrow s(j)$

finpara

```
Inserta la tarea nueva en la posición r+1
```

$s(r+1) \leftarrow i$

```
Pasa a evaluar la siguiente tarea
```

$k \leftarrow k+1$

finsi

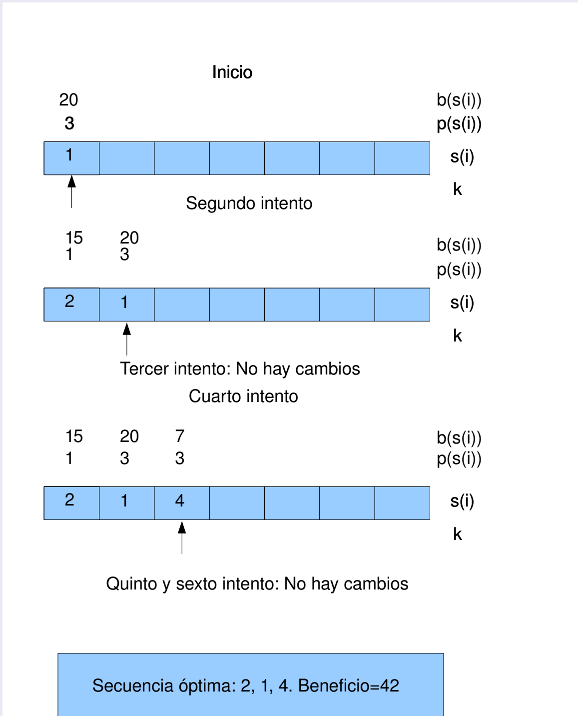
finpara

fin

Tema 6. Algoritmos voraces

Ejemplo (Planificación de tareas a plazo fijo)

- $n = 6$ *beneficio* $b_i = \{20, 15, 10, 7, 5, 3\}$ *plazos* $p_i = \{3, 1, 1, 3, 1, 3\}$



Tema 6. Algoritmos voraces

Algoritmo de Kruskal.

- Problema: Obtener el árbol abarcador de coste mínimo en un grafo conexo y no dirigido de n nodos.
- Solución:
  - Ordenar los lados de menor a mayor y seleccionar $n-1$ lados en orden creciente siempre y cuando enlacen dos componentes conexas distintas.
  - Al finalizar hay una componente conexa que enlaza todos los nodos.

Tema 6. Algoritmos voraces

Algoritmo *Kruskal*(*GRAFOG*; ; *GRAFOL*)

inicio

ordenar(*CL*) `ordena crecientemente el conjunto de lados`

$L \leftarrow \phi$ `Inicialmente ningún lado forma parte de la solucion`

inicializar n conjuntos `Inicialmente hay tantos conjuntos como nodos`

repetir

$(u, v) \leftarrow$ `Lado mas corto no considerado`

uconjunto $\leftarrow$ *buscar*(u) `Conjunto al que pertenece nodo u`

vconjunto $\leftarrow$ *buscar*(v) `Conjunto al que pertenece nodo v`

si *uconjunto* $\neq$ *vconjunto* **entonces**

fusionar(*uconjunto*, *vconjunto*) `Se fusionan los conjuntos de u y v`

$L \leftarrow L + (u, v)$ `El lado (u,v) se añade al grafo solucion`

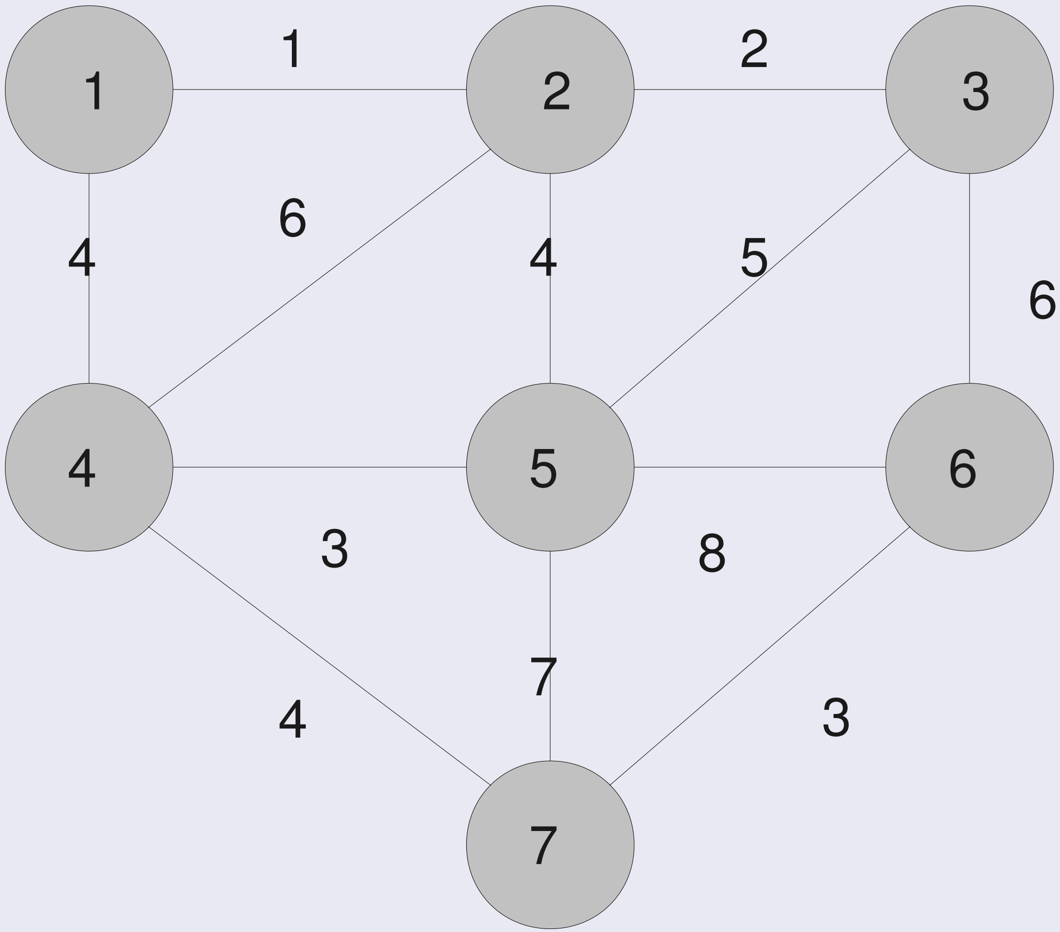
finsi

hasta que L *tenga* $n - 1$ *lados*.

fin

Tema 6. Algoritmos voraces

Ejemplo (Ejemplo algoritmo de Kruskal)



Tema 6. Algoritmos voraces

Viajante de comercio.

- Problema: Consiste en recorrer todos los nodos de un grafo conexo no dirigido, volviendo al nodo de partida y sin pasar dos veces por el mismo nodo, a un coste mínimo.
- Se puede usar un algoritmo voraz para obtener una solución aproximada.
- Se seleccionan n lados de forma que:
  - En orden creciente.
  - Sin formar ciclos (ciclo sólo al seleccionar el último).
  - Sin que más de dos lados confluyan en un nodo.
- Ver ejemplo de los apuntes.