Diseño y Análisis de Algoritmos

TEMA 3. ALGORITMOS VORACES

Universidad
Rey Juan Carlos

Tema 3. Algoritmos Voraces

Introducción

- Motivación: Problema de la devolución del cambio
- Esquema de la técnica
- Aspectos de diseño
- Aplicaciones de los algoritmos voraces
  - Minimización del tiempo en el sistema
  - Problema de la Mochila
  - Planificación con plazo fijo
  - Problemas en grafos (árboles de recubrimiento y caminos más cortos)

Tema 3. Algoritmos Voraces

Motivación

- *Greedy* (adj.): voraz, avaricioso, ávido, codicioso, glotón, …

Dado un problema con n entradas, el objetivo es obtener un subconjunto de estas, de tal forma que se satisfaga una determinada restricción de forma óptimo

- La forma habitual de resolverlo es:
  - Escoger las mejores entradas que verifiquen las restricciones hasta que se encuentra la solución que se busca

Tema 3. Algoritmos Voraces

Motivación

- Supongamos un país cuyo sistema monetario tiene monedas con valores $v_1, v_2, \ldots v_n$. El **problema del cambio** se pueden enunciar como.

"Descomponer una cantidad dada M, en monedas de valores $v_1, v_2, \ldots v_n$, de forma que el número de monedas utilizados sea mínimo"

Motivación

Datos relevantes

- **Candidatos**: Monedas $C = \{v_1, v_2, \ldots, v_n\}$
- **Solución**: La suma de las monedas elegidas son igual al cambio
- **Factibilidad**: la suma de monedas no puede superar al cambio
- **Objetivo**: minimizar las monedas devueltas
- **¿Selección?** Moneda de mayor valor mientras sea factible

Tema 3. Algoritmos Voraces

Motivación

```
funcion cambioVoraz(M: entero, Valor: conjunto):vector
//M: cantidad a devolver;
//Valor: conjunto de valores de las monedas
var
moneda: Valor; Cambio: vector solución;
para moneda desde prim(Valor) hasta ult(Valor) hacer
   Cambio[moneda] ← 0
fin_para
para moneda desde prim(Valor) hasta ult(Valor) hacer
   mientras (Valor[moneda]<=M) hacer
      Cambio[moneda] ← Cambio[moneda] + 1
      M ← M - Valor(moneda)
   fin_mientras
fin_para
devolver Cambio
fin_funcion
```

Esquema de la técnica

Identificar

- Conjunto de **candidatos** y conjuntos de **seleccionados**
- Función de **selección**: elige el candidato idóneo en cada etapa
- Función **solución**: determina si los candidatos seleccionados son una solución
- Función de **factibilidad**: determina si el conjunto de seleccionados es prometedor
- Función **objetivo**: determina el valor de la solución

Tema 3. Algoritmos Voraces

Esquema de la técnica

```
funcion voraz(C: conjunto):conjunto
{C es el conjunto de candidatos y S la solución}
S ← ∅
mientras C ≠ ∅ y no solucion(S) hacer
    x ← seleccionar(C)
    C ← C \ {x}
    si factible(S ∪ {x}) entonces
        S ← S ∪ {x}
    fin_si
fin_mientras
si solucion(S) entonces
    devolver S
si_no
    devolver ∅
fin_si
fin_funcion
```

Esquema de la técnica

Características del esquema

- Se construye una solución **iterativamente**
- Se toma la decisión **óptima** en cada **iteración**
- Una vez analizado un candidato (introducir o excluir), **no se reconsidera** la decisión
- Son voraces porque en cada etapa toman **la mejor decisión** sin preocuparse de mañana

Aspectos de diseño

Ventajas

- La **implementación** de este tipo de algoritmos suele ser **sencilla**
- Producen soluciones de forma muy **eficiente** (complejidad polinómica)
- Encuentran la solución **óptima** para un número **determinado** de problemas

Aspectos de diseño

Desventajas

- No siempre se encuentra la **solución óptima** (por ejemplo, cambio con monedas de 11, 5 y 1)
- No **reconsiderar** decisiones **pasadas** puede conducir a no obtener el **óptimo global** (por ejemplo, el problema del viajante)
- **Encontrar** la función de selección que garantice la optimalidad (por ejemplo, prob. de la mochila)
- **Demostración** formal de la optimalidad (encuentra el óptimo global)

Aplicaciones de AV. Min. del tiempo de espera en el sistema

Supongamos un servidor que tiene que dar servicio a n clientes (procesador, cajero, …). El tiempo requerido por cada cliente t_i es conocido. Se desea ***minimizar el tiempo medio de cada cliente en el sistema***

$$T_{med} = \frac{\sum_{i=1}^{n} t_i}{n}$$

Como *n* es conocido, equivale a minimizar el tiempo total invertido pro cada cliente en el sistema

$$T = \sum_{i=1}^{n} t_i$$

Tema 3. Algoritmos Voraces

Aplicaciones de AV. Min. del tiempo de espera en el sistema

- Conjunto de **candidatos**: los *n* clientes
- Función **solución**: todos los clientes han sido ordenados
- Función de **factibilidad**: si han sido ordenados los clientes o no
- Función **objetivo**: minimizar *T*
- Función de **selección**:¿?

Aplicaciones de AV. Min. del tiempo de espera en el sistema

- Ejemplo con 3 clientes

$n = 3$			
	1	2	3
t	5	10	3

Orden	T	T_{med}
1,2,3	5+(5+10)+(5+10+3) 38	12.67
1,3,2	5+(5+3)+(5+3+10) = 31	10.33
2,1,3	10+(10+5)+(10+5+3) = 43	14.33
2,3,1	10+(10+3)+(10+3+5) = 41	13.67
3,1,2	3+(3+5)+(3+5+10) = 29	9.67
3,2,1	3+(3+10)+(3+5+10) = 34	11.33

Tema 3. Algoritmos Voraces

Aplicaciones de AV. Min. del tiempo de espera en el sistema

- Función de **selección**: Elegir los candidatos por orden creciente de t_i
- El algoritmo voraz se reduce a **ordenar** de forma no decreciente en t_i los n clientes
- En pseudo-código

```
procedimiento tiempoEspera(t[1..n],VAR sol[1..n]);
{bucle voraz}
   para i ← 1 hasta n hacer
       sol[i]←j del cliente no inc. con menor t[j]
      fin_para
fin_ procedimiento
```

Aplicaciones de AV. Problema de la mochila

Supongamos que tenemos n objetos y una mochila. Cada objeto i tiene un peso $w_i > 0$ y un valor $v_i > 0$. La mochila puede llevar un peso que no sobrepase W. Se desea llenar la mochila ***maximizando*** *el valor de los objetos transportados*

$$\max \sum_{i=1}^{n} x_i v_i$$

con las estricciones

$$\sum_{i=1}^{n} x_i w_i \leq W$$

$$0 \leq x_i \leq 1 \text{ con } 1 \leq i \leq n$$

Aplicaciones de AV. Problema de la mochila

- Conjunto de **candidatos**: los *n* objetos
- Función **solución**: cuando no puedan añasirse más fracciones de objetos a la mochila
- Función de **factibilidad**: $\sum_{i=1}^{n} x_i w_i \leq W$
- Función **objetivo**: maximizar $\sum_{i=1}^{n} x_i v_i$
- Función de **selección**:¿?

Aplicaciones de AV. Problema de la mochila

- Ejemplo con 5 objetos

n = 5, W = 100					
w	10	20	30	40	50
v	20	30	66	40	60
v/w	2.0	1.5	2.2	1.0	1.2

Función	x_i	Valor
Max v_i	0, 0, 1, 0.5, 1	146
Min w_i	1, 1, 1, 1, 0	156
Max v_i/w_i	1, 1, 1, 0, 0.8	164

Tema 3. Algoritmos Voraces

Aplicaciones de AV. Problema de la mochila

```
procedimiento mochila(w[1..n],v[1..n],W,VAR x[1..n]);
   {Inicialización}
   para i desde 1 hasta n hacer
      x[i] ← 0
   fin_para
   peso ← 0
   {bucle voraz}
   mientras peso < W hacer
      i ← el mejor objeto restante {mejor tasa v_i/w_i }
      si peso + w[i] ≤ W entonces
         x[i] ← 1
         peso ← peso + w[i]
      sino
         x[i] ← (W - peso)/w[i]
         peso ← W
      fin_si
   fin_mientras
fin_ procedimiento
```

Aplicaciones de AV. Planificación con plazo fijo

Supongamos que tenemos n trabajos, donde cada trabajo i tiene una fecha tope de realización $f_i > 0$ y un beneficio $b_i > 0$

- *Para cualquier trabajo i, el beneficio b_i se gana si y sólo sí se realiza antes (o coincidiendo) con su fecha tope f_i*
- *El trabajo se realiza en una máquina que consume una unidad de tiempo y sólo hay una máquina disponible (i.e., en un instante de tiempo sólo se puede ejecutar una tarea)*

Aplicaciones de AV. Planificación con plazo fijo

- Ejemplo con 5 objetos

n = 4	1	2	3	4
Beneficio	50	10	15	30
Fecha tope	2	1	2	1

Secuencia (T)	Beneficio	Secuencia (T)	Beneficio
<1>	50	<2,1>	60
<2>	10	<2,3>	25
<3>	30	<3,1>	65
<4>	30	<4,1>	80
<1,3>	65	<4,3>	45

Aplicaciones de AV. Planificación con plazo fijo

- Conjunto de **candidatos**: los *n* trabajos a realizar
- Función **solución**: cuando se haya planificado todas las tareas
- Función de **factibilidad**: conjunto *T* de trabajos que todavía se pueden completar antes de su tope
- Función **objetivo**: maximizar $\sum_{i \in T} b_i$
- Función de **selección**: Considerar trabajos en orden decreciente de los beneficios

Tema 3. Algoritmos Voraces

Aplicaciones de AV. Planificación con plazo fijo

- Ordenar por beneficio

n = 4	1	2	3	4
Beneficio	50	10	15	30
Fecha tope	2	1	2	1

n = 4	1	4	3	2
Beneficio	50	30	15	10
Fecha tope	2	1	2	1

Paso	Tarea sel.	Factible?	T
0	----	----	Ø
2	1	<1>	<1>
3	4	<1,4>,<4,1>	<4,1>
4	3	<3,4,1>,<4,3,1>,<4,1,3>	<4,1>
1,3	2	<2,4,1>,<4,2,1>,<4,1,2>	<4,1>