Tecnológico de Monterrey.

Campus Querétaro.

TC2038. Análisis y diseño de Algoritmos A

M.C. Ramona Fuentes Valdéz

rfuentes@tec.mx

12

Técnicas de diseño de algoritmos

Algoritmos voraces (*Greedy*)

- Suelen ser los más fáciles de implementar
- Usan un enfoque miope (*toman decisiones basándose en la información que tienen disponible de modo inmediato*).
- Se usan típicamente para resolver problemas de optimización.
  - ❖ Hay muchos problemas que no se pueden resolver con un enfoque tan "*grosero*".

Características generales

- Buscan resolver un problema de forma óptima basados en una lista de solución candidatos.
- A medida que avanza el algoritmo, se van acumulando dos conjuntos: un conjunto con candidatos rechazados y otro con candidatos seleccionados.
- **Función de solución:** función que comprueba si un cierto conjunto de candidatos constituye una solución (*ignorando si es o no óptima por el momento*).
- **Función de factibilidad:** comprueba si un cierto conjunto de candidatos es factible.
- **Función de selección**: indica cuál de los candidatos restantes es más prometedor.
- **Función objetivo:** da el valor de la solución que hemos hallado (*no aparece explícitamente en el algoritmo voraz*).

IMTA Dr. Alberto González JavaTpoint - Greedy Algorithm Introduction Data Structures - Greedy Algorithms Basics of Greedy Algorithms

13

Técnicas de diseño de algoritmos

Algoritmos voraces (Greedy)

- Suelen ser los más fáciles de implementar
- Usan un enfoque miope (*toman decisiones basándose en la información que tienen disponible de modo inmediato*).
- Se usan típicamente para resolver problemas de optimización.
  - ❖ Hay muchos problemas que no se pueden resolver con un enfoque tan "*grosero*".

Algoritmo

```
función voraz(C:conjunto):conjunto
  // C es el conjunto de candidatos
  S ← ∅  // S es el conjunto solución
  mientras C != ∅ hacer {
       x ← seleccionar(C)
       C ← C \ {x}
       si factible (S ∪ {x}) entonces S ← S ∪ {x}
       si solución(S) entonces devolver S
       si_no devolver "no hay soluciones"
  }
```


14

Técnicas de diseño de algoritmos

Algoritmos voraces (Greedy)

- Suelen ser los más fáciles de implementar
- Usan un enfoque miope (*toman decisiones basándose en la información que tienen disponible de modo inmediato*).
- Se usan típicamente para resolver problemas de optimización.
  - ❖ Hay muchos problemas que no se pueden resolver con un enfoque tan "*grosero*".

Ejemplo:

- Se tiene un conjunto de A dígitos y se quiere formar con ellos, sin repetir, el mayor número entero que sea posible.

Algoritmo:

1. Iniciar con un entero E formado por 0 dígitos
2. Si el conjunto A es vacío ir a 7
3. Seleccionar el mayor dígito d del conjunto A
4. Concatenar d al entero E
5. Borrar d de A
6. Ir a 2
7. Regresar E como la solución del problema

Sea A = {3, 8, 1, 7, 9}

E =

For *i* : 1 to *k* do
Escoger el número más grande y eliminarlo de la entrada.
Endfor

15

Técnicas de diseño de algoritmos

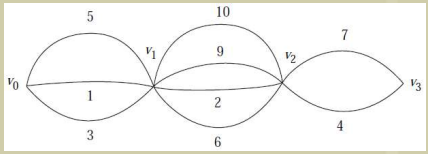
Algoritmos voraces (*Greedy*)

- Suelen ser los más fáciles de implementar
- Usan un enfoque miope (*toman decisiones basándose en la información que tienen disponible de modo inmediato*).
- Se usan típicamente para resolver problemas de optimización.

❖ Hay muchos problemas que no se pueden resolver con un enfoque tan "*grosero*".

Ejemplo:

- Se solicita encontrar la ruta más corta de v_0 a v_3.



Algoritmo:

```
función voraz(C:conjunto):conjunto
 // C es el conjunto de candidatos
 S ← ∅  // S es el conjunto solución
 mientras C != ∅ hacer {
    x ← seleccionar(C)
    C ← C \ {x}
    si factible (S ∪ {x}) entonces S ← S ∪ {x}
    si solución(S) entonces devolver S
    si_no devolver "no hay soluciones"
 }
```

Solución:

- Encontrar la ruta más corta entre v_i y v_{i+1}, desde $i = 0$ hasta 2.

  Es decir, primero se determina la ruta más corta entre v_0 y v_1, luego entre v_1 y v_2, y así sucesivamente.

16

Técnicas de diseño de algoritmos

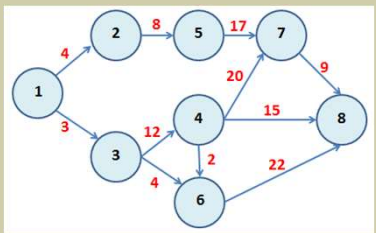
Algoritmos voraces (*Greedy*)

- Suelen ser los más fáciles de implementar
- Usan un enfoque miope (*toman decisiones basándose en la información que tienen disponible de modo inmediato*).
- Se usan típicamente para resolver problemas de optimización.

❖ Hay muchos problemas que no se pueden resolver con un enfoque tan "*grosero*".

Ejemplo:

- Se solicita encontrar la ruta más corta del nodo 1 al nodo 8.



Algoritmo:

```
función voraz(C:conjunto):conjunto
 // C es el conjunto de candidatos
 S ← ∅  // S es el conjunto solución
 mientras C != ∅ hacer {
    x ← seleccionar(C)
    C ← C \ {x}
    si factible (S ∪ {x}) entonces S ← S ∪ {x}
    si solución(S) entonces devolver S
    si_no devolver "no hay soluciones"
 }
```

Solución:

- Ruta 1 – 3 – 6 – 8:

  3 + 4 + 22 = 29 km

17

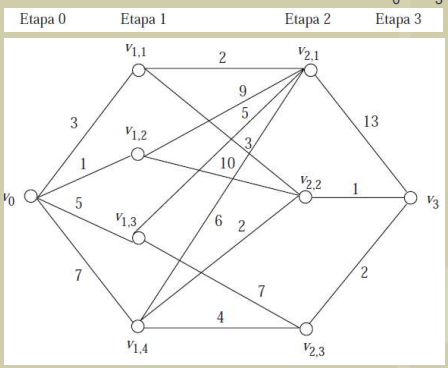
Técnicas de diseño de algoritmos

Algoritmos voraces (*Greedy*)

- Suelen ser los más fáciles de implementar
- Usan un enfoque miope (*toman decisiones basándose en la información que tienen disponible de modo inmediato*).
- Se usan típicamente para resolver problemas de optimización.
  - ❖ Hay muchos problemas que no se pueden resolver con un enfoque tan "*grosero*".

Ejemplo:

- Se solicita encontrar la ruta más corta de v_0 a v_3.



Solución:

- Ruta $v_0 \rightarrow v_{1,2} \rightarrow v_{2,1} \rightarrow v_3$

  Longitud total

  $1 + 9 + 13 = 23$

  no es una solución óptima

- Ruta $v_0 \rightarrow v_{1,1} \rightarrow v_{2,2} \rightarrow v_3$

  Longitud total

  $3 + 3 + 1 = 7$

¿Qué falla?

18

Técnicas de diseño de algoritmos

Algoritmos voraces (*Greedy*)

Ejercicio: Problema de dar cambio

Caso 1.

- Supón que vives en un país donde sólo están disponibles las monedas de 50, 25, 10 y 5.
- Debes diseñar un algoritmo para pagar una cantidad *utilizando el menor número posible de monedas*.

Algoritmo

1. Hacer la cantidad d = 0 (cantidad formada) y n = 0 (número de monedas usadas)
2. Si la cantidad d = D ir a 4
3. Seleccionar la moneda más grande M que al ser sumada a d dé una cantidad menor a D
   a. Hacer d = d + D y n = n + 1
   b. Ir a 2
4. Regresar n como la respuesta
5. FIN

Problema:

Para pagar 175.

Solución:

- 3 monedas de 50
- 1 moneda de 25.

19

Técnicas de diseño de algoritmos

Algoritmos voraces (Greedy)

Ejercicio: Problema de dar cambio

Caso 2.

- Supón que vives en un país donde sólo están disponibles las monedas de 25, 20, 10 y 5.
- Debes diseñar un algoritmo para pagar una cantidad *utilizando el menor número posible de monedas*.

Algoritmo

1. Hacer la cantidad d = 0 (cantidad formada) y n = 0 (número de monedas usadas)
2. Si la cantidad d = D ir a 4
3. Seleccionar la moneda más grande M que al ser sumada a d dé una cantidad menor a D
   a. Hacer d = d + D y n = n + 1
   b. Ir a 2
4. Regresar n como la respuesta
5. FIN

Problema:

Para pagar 40.

Solución:

- 3 monedas. ⟩ *no es una solución óptima*

No funciona el algoritmo

Algoritmo correcto

Se dice que un algoritmo es correcto cuando es capaz de resolver correctamente todas las instancias de un problema dado.

20

Técnicas de diseño de algoritmos

Algoritmos voraces (Greedy)

Ejercicio: Problema de dar cambio

Caso 3.

- Supón que vives en un país donde sólo están disponibles las monedas de 10, 7 y 1.
- Debes diseñar un algoritmo para pagar una cantidad *utilizando el menor número posible de monedas*.

Algoritmo

1. Hacer la cantidad d = 0 (cantidad formada) y n = 0 (número de monedas usadas)
2. Si la cantidad d = D ir a 4
3. Seleccionar la moneda más grande M que al ser sumada a d dé una cantidad menor a D
   a. Hacer d = d + D y n = n + 1
   b. Ir a 2
4. Regresar n como la respuesta
5. FIN

Problema:

Para pagar 15.

Solución:

- 6 monedas. ⟩ *no es una solución óptima*

Solución mejor:

- 3 monedas.

21

Técnicas de diseño de algoritmos

Algoritmos voraces (*Greedy*)

Ejercicio: Problema de dar cambio

- Supón que vives en un país donde sólo están disponibles las monedas de 100, 25, 10, 5 y 1.
- Debes diseñar un algoritmo para pagar una cantidad *utilizando el menor número posible de monedas.*
- *Ejemplo*, para pagar 289 → mejor solución son 2 monedas de 100, 3 de 25, 1 de 10 y 4 de 1).

Algoritmo

```
función devolver_cambio(n):conjunto de monedas
  // da el cambio de n unidades utilizando el menor número posible
  // de monedas. La constante C especifica las monedas disponibles
  const C={100, 25, 10, 5, 1}
  S ← {}  // S es un conjunto que contendrá la solución
  sumaS ← 0  // es la suma de los elementos de S
  mientras sumaS != n hacer
    x ← el mayor elemento de C tal que sumaS + x <= n
    si no existe ese elemento entonces
      devolver "no encuentro la solución"
    S ← S ∪ x  // una moneda de valor x
    sumaS ← sumaS + x
  devolver S
```

- Los **candidatos** (C) son el conjunto de monedas. El número de candidatos debe ser finito.
- La ***función de solución*** comprueba si el valor de las monedas seleccionadas es exactamente el valor que hay que pagar.
- Un conjunto de monedas será factible si su valor total no sobrepasa la cantidad que hay que pagar (***función de factibilidad***).
- La ***función de selección*** toma la moneda con valor más alto que quede en el conjunto de candidatos.
- La ***función objetivo*** cuenta el número de monedas utilizado en la solución.

22

Técnicas de diseño de algoritmos

Algoritmos voraces (*Greedy*)

Ejercicio: Problema de dar cambio

- Supón que vives en un país donde sólo están disponibles las monedas de 100, 25, 10, 5 y 1.
- Debes diseñar un algoritmo para pagar una cantidad *utilizando el menor número posible de monedas.*
- *Ejemplo*, para pagar 289 → mejor solución son 2 monedas de 100, 3 de 25, 1 de 10 y 4 de 1).

Algoritmo

```
función devolver_cambio(n):conjunto de monedas
  // da el cambio de n unidades utilizando el menor número posible
  // de monedas. La constante C especifica las monedas disponibles
  const C={100, 25, 10, 5, 1}
  S ← {}  // S es un conjunto que contendrá la solución
  sumaS ← 0  // es la suma de los elementos de S
  mientras sumaS != n hacer
    x ← el mayor elemento de C tal que sumaS + x <= n
    si no existe ese elemento entonces
      devolver "no encuentro la solución"
    S ← S ∪ x  // una moneda de valor x
    sumaS ← sumaS + x
  devolver S
```

- El algoritmo es "voraz" porque en cada paso **selecciona la mayor** de las monedas que pueda encontrar, *sin preocuparse* por lo correcto de la decisión a la larga.
- Nunca cambia de opinión (una vez que se ha incluido una moneda al conjunto, la moneda permanece ahí).
- **¿Qué pasa si el suministro de alguna de las monedas está limitado?**

23

Técnicas de diseño de algoritmos

Algoritmos voraces (Greedy)

- Suelen ser los más fáciles de implementar
- Usan un enfoque miope (*toman decisiones basándose en la información que tienen disponible de modo inmediato*).
- Se usan típicamente para resolver problemas de optimización.

Algunas aplicaciones:

- Problema del vendedor ambulante (*Salesman*)
- Algoritmo de árbol de expansión mínimo de Prim
- Algoritmo de árbol de expansión mínimo de Kruskal
- Algoritmo de árbol de expansión mínimo de Dijkstra
- Grafos - Coloreo
- Grafos - Vértices
- Problema de la mochila
- Problema de asignación de tiempos

Conclusiones

- El análisis depende de cada algoritmo concreto, pero siempre se identifican las funciones elementales.
- En la práctica los algoritmos voraces suelen ser bastante rápidos, encontrándose dentro de órdenes de complejidad polinomiales.
- La clave es la función de selección.
- Puede no encontrar la solución óptima, o no encontrarla definitivamente.

IMTA Dr. Alberto González JavaTpoint - Greedy Algorithm Introduction Data Structures - Greedy Algorithms Basics of Greedy Algorithms