5 Voracidad y grafos

ALGORÍTMICA Y COMPLEJIDAD

Camilo Palazuelos Calderón

Curso 2023-2024

Este material se publica bajo la licencia Creative Commons BY-NC-SA 4.0

```
VORAZ(X):
    Y ⟵ ∅                                  ▷ Conjunto solución
    while not ESSOLUCIÓN(Y)
        if X = ∅
            return NULL
        x ⟵ un elemento de X               ▷ ¿Cuál?
        X ⟵ X \ {x}                        ▷ x no se vuelve a considerar
        if ESFACTIBLE(Y ∪ {x})             ▷ ¿Añadir x lleva a solución?
            Y ⟵ Y ∪ {x}
    return Y
```

Qué son los algoritmos voraces

- Suelen utilizarse para resolver problemas de optimización
- El diseño iterativo es la manera natural de pensar en ellos
  - En cada iteración, se añade un elemento a la solución parcial
  - La decisión de añadir un elemento particular a la solución parcial es *irrevocable*
  - Al final de cada iteración, se comprueba si la solución parcial constituye una solución

Cuándo usar algoritmos voraces

- ¡NUNCA!
  - Al menos, no deben usarse si no se dispone de una demostración de que el algoritmo *siempre* conduce a la solución

- Si se verifica el principio de optimalidad
  - Si las subsecuencias de la secuencia de decisiones óptima son también óptimas
  - *No* es condición suficiente para garantizar la existencia de una función de selección que conduzca iterativamente a la solución

```
MOCHILA(P[1..n], V[1..n], p_máx):
    Y[1..n] ← (0,…,0)                          ▷ Solución
    p_act ← 0                                  ▷ Peso total actual
    n_act ← 0                                  ▷ N.º de objetos actual
    while not (p_act = p_máx or n_act = n)
        i ← índice de un objeto                ▷ ¿Cuál?
        if p_act + P[i] ≤ p_máx                ▷ El i-ésimo objeto cabe
            Y[i] ← 1
            p_act ← p_act + P[i]
        else
            Y[i] ← (p_máx – p_act) / P[i]
            p_act ← p_máx
        n_act ← n_act + 1
    return Y[1..n]
```

El problema de la mochila más sencillo

- Disponemos de n objetos con peso y valor y de una mochila que soporta un peso p
  - Los objetos, si se rompen, ven reducido su peso y valor en la misma proporción
  - El objetivo es llenar la mochila maximizando el valor total de los objetos introducidos
- ¿Función de selección adecuada?
- ¿Coste temporal?

Búsquedas en grafos

- Dados un grafo $G = (V, E)$ y un vértice x de V, ¿qué vértices son alcanzables desde x?
- El tipo de búsqueda y su coste temporal vienen determinados por la implementación de B
  - *Pila:* Búsqueda en profundidad
  - *Cola:* Búsqueda en anchura
  - *Cola de prioridad:* Búsqueda primero el mejor
- BUSCAR produce árboles generadores que dependen de
  - La implementación de B
  - El vértice de partida
  - El orden en que se visitan los vecinos de un vértice
- ¿Coste temporal?

```
BUSCAR(V, E, x):
    B ← ()
    INTRODUCIR(x, B)
    while not VACÍA(B)
        v ← EXTRAER(B)
        if not MARCADO(v)
            MARCAR(v)
            for each (v, w) in E
                INTRODUCIR(w, B)
```

```
DIJKSTRA(V, E, x):
    P[1..n] ⟵ (0, …, 0)
    D[1..n] ⟵ (∞, …, ∞)
    D[x] ⟵ 0
    C ⟵ ()
    INSERTAR((x, D[x]), C)
    while not VACÍA(C)
        v ⟵ EXTRAERMÍN(C)
        for each (v, w) in E
            if D[v] + PESO(v, w) < D[w]
                P[w] ⟵ v
                D[w] ⟵ D[v] + PESO(v, w)
                INSERTAR((w, D[w]), C)
    return (P[1..n], D[1..n])
```

El problema del camino más corto

- Dados un grafo con pesos no negativos y un vértice *x* del grafo, ¿cuál es el camino más corto, es decir, aquel cuya suma de pesos es mínima, entre *x* y los demás vértices del grafo?
- Algoritmo de Dijkstra
  - Se basa en BUSCAR con *cola de prioridad*
- ¿Coste temporal?