Algoritmos voraces

Algoritmia y optimización

Grado en Ingeniería en Inteligencia Artificial

Algoritmos voraces

Introducción

- Hay determinados problemas que se pueden resolver **tomando decisiones secuencialmente**, sin tener en cuenta más posibilidades.
- Un **algoritmo voraz** sigue un criterio de selección que elige una opción **óptima local**, con la esperanza de llegar a una solución **óptima global**.

Algoritmos voraces

Esquema general

1. Inicializar la solución.
2. Repetir hasta completar una solución:
    a. Seleccionar el mejor candidato según un criterio voraz.
    b. Comprobar si este candidato puede ser añadido a la solución.
    c. Si se puede: añadir el candidato a la solución.
3. Devolver la solución obtenida.

El problema de la mochila (continua)

Algoritmos voraces

El problema de la mochila (continua)

Instancia

Valores $(v_1, v_2, \ldots, v_n)$

Pesos $(w_1, w_2, \ldots, w_n)$

Peso máximo W

Problema (versión continua)

$$\arg\max_{\mathbf{x}\in[0,1]} \sum_{i=1}^{n} v_i x_i$$

$$\text{s.t.} \quad \sum_{i=1}^{n} w_i x_i \leq W$$

Algoritmos voraces

El problema de la mochila (continua)

- La versión continua del problema de la mochila se puede resolver mediante un **algoritmo voraz**:

    1. **Ordenar** los elementos en relación **descendente** valor / peso
    2. Iterativamente: añadir la **fracción del objeto** correspondiente que cabe en la mochila.

Algoritmos voraces

El problema de la mochila (continua)

Instancia:

- **Valores**: v = (6, 6, 2, 1)
- **Pesos**: w = (3, 4, 1, 1)
- **Capacidad**: W = 5

Valor máximo: 9.5

Solución: x = (1, 0.25, 1, 0)

Algoritmos voraces

El problema de la mochila (continua)

```
función mochila_continua(W, n, pesos, valores):
    peso_total, valor_total := 0

    ordenar_por_ratio(valores, pesos)
    para i desde 1 hasta n:
        si peso_total + pesos[i] <= W
            fraccion := 1
        si no
            fraccion := (W - peso_total) / pesos[i]
        valor_total := valor_total + valores[i] * fraccion
        peso_total := peso_total + pesos[i] * fraccion

    devuelve valor_total
```

Solución completa: ordenar los id iniciales

Algoritmos voraces

El problema de la mochila

- ¿Funciona este esquema en el caso discreto?
  - Puede dar una solución pero, ¿es óptima?
  - ¿Podemos hablar de *algoritmo*?
- ¿Es óptimo en el caso voraz?
  - Los algoritmos voraces requieren una **demostración de optimalidad** (o un contraejemplo para lo contrario).

Algoritmo de Kruskal

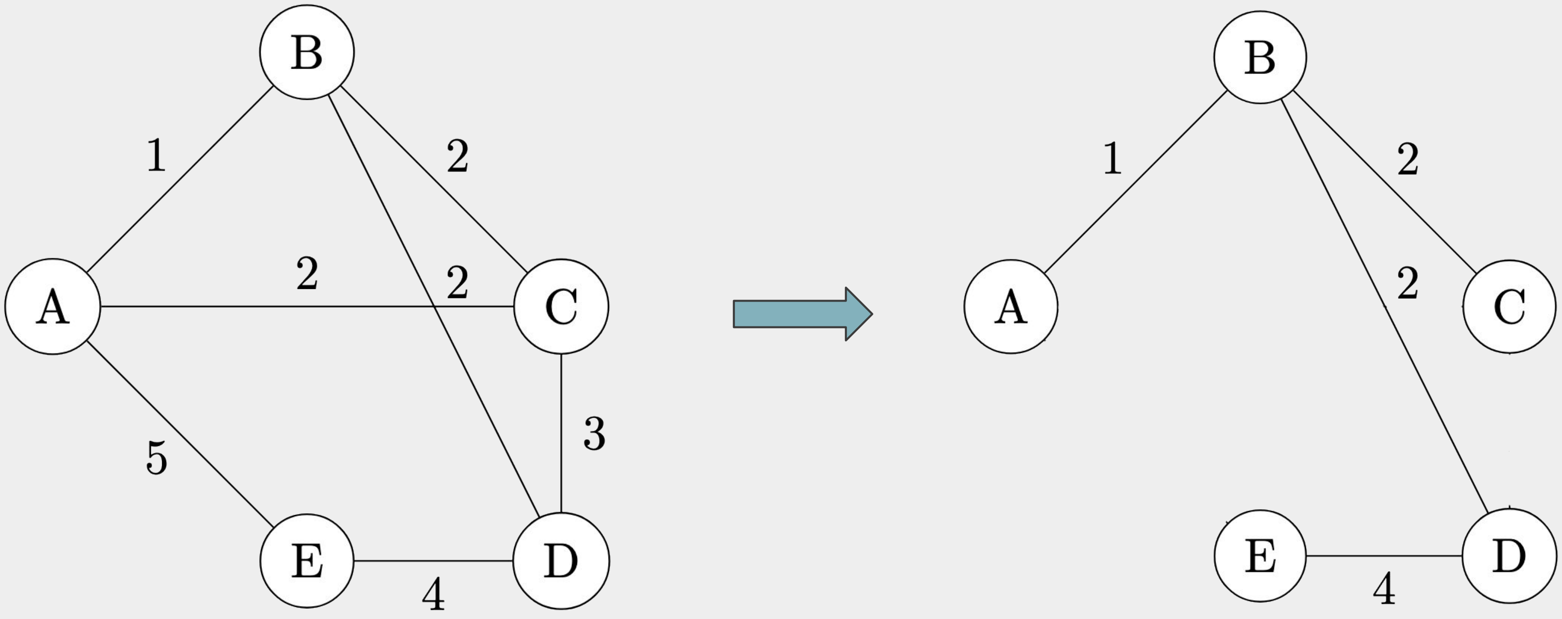
Algoritmos voraces

Árbol de expansión mínima

- Dado un grafo ponderado no dirigido, el **árbol de expansión mínima** (*Minimum Spanning Tree, o MST*) es el **subgrafo** que:
  - Conecta todos los vértices del grafo original.
  - Con el menor peso total posible.
  - Y que no contiene ciclos.
- El MST es fundamental en muchas áreas de la computación.

Algoritmos voraces

Árbol de expansión mínima



Algoritmos voraces

Árbol de expansión mínima: Algoritmo de Kruskal

- El **algoritmo de Kruskal** es un enfoque voraz para calcular el MST.
- Mantiene una **estructura de datos** con todos los sub-árboles producidos hasta el momento.
- Consulta las aristas por **orden creciente** de peso:
  - Si la arista conecta dos vértices de árboles distintos, la incluye en la solución y los dos árboles se unen.
  - Si no, se descarta.

Algoritmos voraces

Árbol de expansión mínima: Algoritmo de Kruskal

1. Inicializar un conjunto de árboles, cada uno con un solo vértice.
2. Crear una lista de todas las aristas del grafo, ordenadas por peso en orden ascendente.
3. Para cada arista en la lista ordenada:
    a. Si la arista conecta dos árboles distintos, añadirla a la solución y unir los dos árboles
4. Repetir el paso 3 hasta que todos los vértices estén conectados en un único árbol.

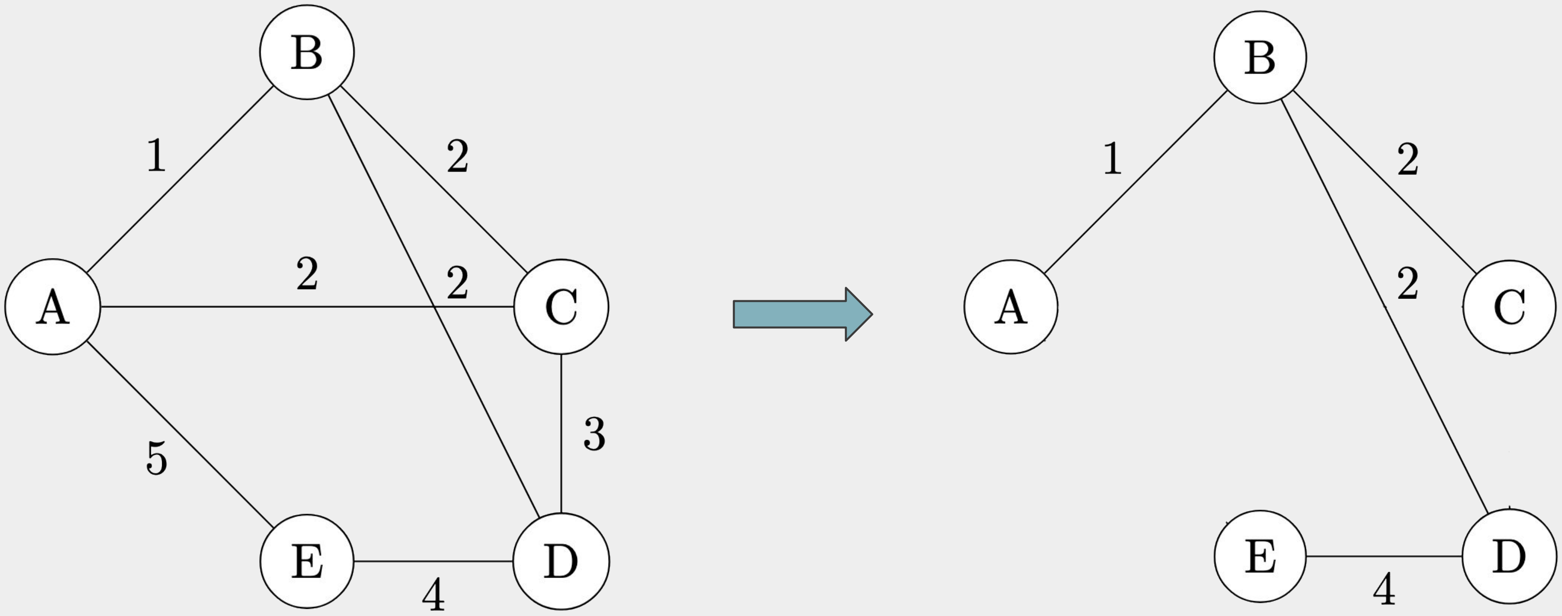
Algoritmos voraces

Árbol de expansión mínima: Algoritmo de Kruskal

```
función kruskal(V,A):
    MST := Ø
    T := estructura de conjuntos
    para i desde 1 hasta |A|
        T[i] = {i}

    ordenar(A)

    para cada (u, v) en A:
        si T[u] != T[v]
            MST = MST ∪ {(u, v)
            unión(T[u],T[v])

    devuelve MST
```

Algoritmos voraces

Árbol de expansión mínima: Algoritmo de Kruskal



Algoritmos voraces

Árbol de expansión mínima: Algoritmo de Kruskal

- ¿Qué estructura de datos utiliza el algoritmo de Kruskal?
  - Estructura ***union-find / disjoint-set***.
    - Operaciones de pertenencia (*find*) y unión (*union*) casi constantes.
    - Complejidad del algoritmo dominada por la ordenación de aristas.
  - Con una **estructura convencional** (vector de pertenencia):
    - Operaciones de pertenencia y unión lineales.
    - Complejidad dominada por el bucle sobre las aristas y operaciones.
- Importancia de usar la **estructura de datos apropiada**.

Ejercicios

Algoritmos voraces

El cambio de monedas

- Dado un conjunto de tipos de monedas ***$D = (d_1, d_2, \ldots, d_n)$***, se busca el **mínimo número de monedas** cuya suma sea una cantidad de dinero **C**.
- ¿Se puede encontrar un **algoritmo voraz**?