Capítulo 6 ALGORITMOS VORACES

Nombre: Jean Carlos Iñiguez

* **CARACTERÍSTICAS GENERALES DE LOS ALGORITMOS VORACES**

Los algoritmos voraces buscan resolver problemas de optimización seleccionando en cada paso el candidato más prometedor, con la intención de construir una solución óptima de forma progresiva. **Estas son sus características principales:**

* **Conjunto de candidatos**: El problema se resuelve eligiendo elementos de un conjunto inicial (por ejemplo, monedas, aristas de un grafo, tareas) que servirán para formar la solución.
* **Acumulación de candidatos:** A medida que avanza el algoritmo, se mantienen dos conjuntos:
  1. **Seleccionados:** candidatos que ya forman parte de la solución.
  2. **Rechazados:** candidatos descartados por no ser factibles.
* **Funciones importantes:**
  1. **Función de solución:** Verifica si el conjunto actual de candidatos forma una solución válida (aunque no necesariamente óptima).
  2. **Función de factibilidad:** Determina si el conjunto actual puede ser completado para formar una solución válida.
  3. **Función de selección:** Indica cuál es el candidato más prometedor para añadir en el siguiente paso.
  4. **Función objetivo:** Evalúa la calidad de la solución (por ejemplo, minimizar el número de monedas o el tiempo de procesamiento). Esta función no se usa directamente durante la selección, sino para medir el resultado final.
* **Proceso paso a paso:**
  1. Se comienza con un conjunto vacío.
  2. En cada paso, se selecciona el mejor candidato disponible según la función de selección.
  3. Si añadirlo hace el conjunto no factible, se rechaza ese candidato.
  4. Si sigue siendo factible, se añade el candidato al conjunto seleccionado.
  5. Se verifica si el conjunto seleccionado ya es una solución completa.
  6. La primera solución completa encontrada suele ser óptima cuando el algoritmo voraz es correcto.
* **GRAFOS: ÁRBOLES DE RECUBRIMIENTO MÍNIMO:**

El problema del árbol de recubrimiento mínimo (o árbol generador mínimo) consiste en:

* Dado un grafo G = (N, A) conexo y no dirigido, con pesos o costes no negativos en las aristas, se desea encontrar un subconjunto T de aristas tal que:
  1. Todos los nodos queden conectados.
  2. El coste total de las aristas en T sea el mínimo posible.
  3. T tenga exactamente n - 1 aristas (siendo *n* el número de nodos), formando así un árbol.

Propiedades y consideraciones clave:

* Si una solución tiene más de *n - 1* aristas, entonces contiene al menos un ciclo, por lo que puede simplificarse eliminando una arista sin perder la conexión, mejorando el resultado.
* Si hay varias soluciones de igual coste, se prefiere la que tiene menos aristas.
* A este grafo G' = (N, T) se le llama árbol de recubrimiento mínimo.
* Este tipo de problemas tiene muchas aplicaciones prácticas, como:
  + Diseño de redes (por ejemplo, tender líneas telefónicas entre ciudades con el menor coste).

Estrategias voraces posibles:

Existen dos enfoques voraces que resultan eficaces en este problema:

1. **Selección de aristas directamente:**  
   Se parte de un conjunto vacío y se selecciona en cada paso la arista más corta aún no usada, sin importar su ubicación en el grafo.
2. **Expansión desde un nodo:**  
   Se parte de un nodo arbitrario y se extiende un árbol seleccionando siempre la arista más corta que conecte un nodo nuevo al árbol.

Ambas estrategias llevan correctamente a un árbol de recubrimiento mínimo en este caso, lo que no es común en todos los problemas.

* **GRAFOS: CAMINOS MÍNIMOS:**

**Objetivo del problema:**

Dado un grafo dirigido G = (N, A) con pesos no negativos en sus aristas y un nodo origen, se desea hallar el camino de menor longitud (o coste) desde el origen hasta cada uno de los demás nodos.

**Idea principal:**

Este problema se resuelve de forma eficiente mediante un algoritmo voraz conocido como algoritmo de Dijkstra.

**Conceptos clave:**

* Se utilizan dos conjuntos de nodos:
  + S: nodos cuya distancia mínima desde el origen ya ha sido determinada.
  + C: nodos candidatos, es decir, los que aún no han sido procesados.
* Siempre se cumple que N = S ∪ C.

**Estructura del algoritmo:**

1. **Inicialización:**
   * Se toma un nodo origen (generalmente el nodo 1).
   * El conjunto S comienza solo con el nodo origen.
   * El arreglo D[i] representa la distancia conocida más corta desde el origen al nodo *i*.
   * La matriz L[i, j] contiene las longitudes de las aristas:
     + L[i, j] = coste si existe una arista de *i* a *j*.
     + L[i, j] = ∞ si no existe.
2. **Iteración:**
   * En cada paso, se selecciona de C el nodo con menor distancia conocida al origen (según D).
   * Ese nodo se mueve a S.
   * Se actualizan las distancias de sus vecinos en C usando el principio: