1. Conceptos básicos algorítmicos

Complejidad temporal:

- Constante -> O(1)
- Logarítmico -> O(log n)
- Lineal -> O(n)
- Cuasilineal -> O(n log n)
- Cuadrático -> O(n^2)
- Cúbico -> O(n^3)
- Polinómico -> O(n^k)
- Exponencial -> O(k^n)
- (*) -> O(n^½), O(n!), O(n^n)

Symbol	$L = \lim_{n\to\infty} \frac{f(n)}{g(n)}$	intuition
$f(n) = O(g(n))$	$L < \infty$	$f \leq g$
$f(n) = \Omega(g(n))$	$L > 0$	$f \geq g$
$f(n) = \Theta(g(n))$	$0 < L < \infty$	$f = g$
$f(n) = o(g(n))$	$L = 0$	$f < g$
$f(n) = \omega(g(n))$	$L = \infty$	$f > g$

Un algoritmo es factible si su coste es polinómico.

Given an integer $n > 0$ and a real $a > 1$ and $a \neq 0$:

- Arithmetic summation: $\sum_{i=0}^{n} i = \frac{n(n+1)}{2}$.
- Geometric summation: $\sum_{i=0}^{n} a^i = \frac{1-a^{n+1}}{1-a}$.

Logarithms and Exponents: For $a, b, c \in \mathbb{R}^+$,

- $\log_b a = c \Leftrightarrow a = b^c \Rightarrow \log_b 1 = 0$
- $\log_b ac = \log_b a + \log_b c$, $\log_b a/c = \log_b a - \log_b c$.
- $\log_b a^c = c \log_b a \Rightarrow c^{\log_b a} = a^{\log_b c} \Rightarrow 2^{\log_2 n} = n$.
- $\log_b a = \log_c a / \log_c b \Rightarrow \log_b a = \Theta(\log_c a)$

Stirling: $n! = \sqrt{2\pi n}(n/e)^n + 0(1/n) + \gamma \Rightarrow n! + \omega((n/2)^n)$.

n-Harmonic: $H_n = \sum_{i=1}^{n} 1/i \sim \ln n$.

ALGORITMO 1: BÚSQUEDA EN PROFUNDIDAD (BFS)

- Definición: Búsqueda en profundidad
- Coste: O(|V| + |E|) lista de adyacencia // O(|V|^2) matriz de adyacencia

ALGORITMO 2: BÚSQUEDA EN ANCHURA (DFS)

- Definición: Búsqueda en anchura
- Coste: O(|V| + |E|) lista de adyacencia // O(|V|^2) matriz de adyacencia

ALGORITMO 3: KOSARAJU-SHARIR'S ALGORITHM

- Definición: Para encontrar componentes fuertemente conexos dentro de los digrafos
- Coste: Coste: O(|V| + |E|)

ALGORITMO 4: TARJAN'S ALGORITHM

- Definición: Para encontrar componentes fuertemente conexos dentro de los digrafos
- Coste: Coste: O(|V| + |E|)

TEOREMAS MAESTROS (SUSTRACTIVA Y DIVISORA)

Teorema mestre I

Sigui $T(n) = \begin{cases} f(n), & \text{si } 0 \leq n < n_0 \\ a \cdot T(n-c) + g(n), & \text{si } n \geq n_0 \end{cases}$

on $n_0 \in \mathbb{N}$, $c \geq 1$, f és una funció arbitrària i $g \in \Theta(n^k)$ per a $k \geq 0$.

Aleshores

$$T(n) \in \begin{cases} \Theta(n^k), & \text{si } a < 1 \\ \Theta(n^{k+1}), & \text{si } a = 1 \\ \Theta(a^{n/c}), & \text{si } a > 1 \end{cases}$$

Teorema mestre II

Sigui $T(n) = \begin{cases} f(n), & \text{si } 0 \leq n < n_0 \\ a \cdot T(n/b) + g(n), & \text{si } n \geq n_0 \end{cases}$

on $n_0 \in \mathbb{N}$, $b > 1$, f és una funció arbitrària i $g \in \Theta(n^k)$ per a $k \geq 0$.

Sigui $\alpha = \log_b(a)$. Aleshores,

$$T(n) \in \begin{cases} \Theta(n^k), & \text{si } \alpha < k \\ \Theta(n^k \log n), & \text{si } \alpha = k \\ \Theta(n^\alpha), & \text{si } \alpha > k \end{cases}$$

ALGORITMO 5: SELECTION SORT

- Definición: Para seleccionar el i-èsimo elemento más pequeño de una lista desordenada
- Coste: O(n)

ALGORITMO 6: COUNTING SORT

- Definición: Ordenación de una lista contando los elementos. Necesita saber el rango en el que se encuentran los elementos. [0, r]
- Coste: O(n + r)

ALGORITMO 7: RADIX SORT

- Definición: Ordenación de una lista mediante la base de sus elementos. Necesita saber el número de dígitos de los elementos (d) y la base en la que se encuentran (b). (decimal, binario, …). Ordena primero el dígito menos significativo del elemento y luego los otros de forma creciente. (321, 234 -> primero 1 y con 4, luego 2 con 3 y finalmente 3 con 2)
- Coste: O(d(n + b))

LOWER BOUND -> un problema tiene tiempo LOWER BOUND L(n) cuando no hay ningún algoritmo que puede solucionarlo en tiempo < L(n) para cualquier input e de tamaño n.

UPPER BOUND -> un problema tiene tiempo UPPER BOUND T(n) cuando hay un algoritmo A tal que para cualquier input x de tamaño n, A(x) da la solución correcta en <= T(n) pasos.

Para demostrar el UPPER BOUND de un problema solo hay que producir un algoritmo A tal que lo solucione, mientras que para demostrar el LOWER BOUND, se tendría que comprobar que para todos los algoritmos existentes, no son capaces de solucionarlo en tiempo < L(n).

2. ALGORITMOS VORACES O GREEDY

Algoritmos diseñados para resolver problemas de optimización combinatoria. En cada paso se escoge la mejor opción, las elecciones son basadas en elecciones previas (no de elecciones futuras) y nunca se hace backtrack. Y la solución es una secuencia de elementos.

2 características de los algoritmos voraces:

- Se puede llegar a un óptimo global seleccionando los óptimos locales.
- Se va construyendo la solución con soluciones parciales óptimas.

Es importante definir bien el criterio de elección de los elementos que forman parte de la secuencia de la solución. Y se tiene que:

- Demostrar por qué este es el criterio que lleva al resultado óptimo (argumento del intercambio, suponer que no es el óptimo y llegar a una contradicción).
- Demostrar la correctesa de este criterio.
- Citar el coste lineal para solucionar el problema con este criterio.

Problemas de programación de tareas o actividades

Dado un conjunto de n tareas (con diferentes características) que necesitan ser procesados por un sistema individual, proporcionar una programación de las tareas, tales que todos puedan ser ejecutados y optimizar algún criterio.

1. **Problema de programación de intervalos**: las tareas tienen tiempo de inicio y tiempo de fin, y el objetivo es hacer una programación de estas tareas tales que se intente ejecutar el máximo número posible de tareas sin que ninguna de ellas se solapen con otras. Solución: escoger las tareas que tengan un tiempo de fin más temprano y que no solape con las que ya se han seleccionado. Coste -> O(n log n)
2. **Selección de actividad ponderada**: las tareas tienen tiempo de inicio y tiempo de fin y una ponderación, y el objetivo es hacer una programación de estas tareas tales que se intente maximizar la ponderación total. NO SE PUEDE SOLUCIONAR CON UN GREEDY, se solucionará con programación dinámica.
3. **Problema de minimización de retrasos**: las tareas tienen un tiempo de procesado y un tiempo límite, y el objetivo es hacer una programación de todas las tareas y determinar a qué tiempo debe comenzar cada tarea para tal de minimizar el tiempo de finalización de una tarea exceda de su tiempo límite. El retraso de una tarea es 0 cuando el tiempo de procesado es menor que el tiempo límite, sinó, es la resta de estos dos tiempos. El objetivo es minimizar el retraso. Criterio: procesar primero las tareas urgentes (las que tienen tiempo límite más reciente). Coste -> O(n log n)

Compresión de datos

Dado como entrada un texto T sobre un alfabeto finito Σ. Queremos representar T con la menor cantidad de bits posible. El objetivo de la compresión de datos es reducir el tiempo de transmisión de archivos grandes y el espacio para almacenarlos. Si usamos codificación de longitud variable, necesitamos un sistema fácil de codificar y decodificar.

Código prefijo

Dado un conjunto de símbolos Σ, un código de prefijo, es φ: Σ → {0,1} + (símbolos a la cadena de bits) donde para distintos x, y ∈ Σ, φ (x) no es un prefijo de φ (y).

- φ (A) = 1 y φ (C) = 101 entonces φ no es un código de prefijo.
- φ (A) = 1, φ (T) = 01, φ (G) = 000, φ (C) = 001 es un código de prefijo.

Árbol de prefijo

Un árbol de prefijos T es un árbol binario con las siguientes propiedades:

- Una hoja por símbolo.
- Borde izquierdo etiquetado como 0 y borde derecho etiquetado como 1.
- Las etiquetas en la ruta desde la raíz hasta una hoja especifican el código del símbolo en esa hoja.

La longitud de codificación

- Given a text S on Σ, with $|S| = n$, and a prefix code ϕ, $B(S)$ is the length of the encoded text.
- For $x \in \Sigma$, define the frequency of x as

$$f(x) = \frac{\text{number occurrencies of } x \in S}{n}$$

  Note: $\sum_{x\in\Sigma} f(x) = 1$.
- We get the formula,

$$B(S) = \sum_{x\in\Sigma} n\, f(x)|\phi(x)| = n \sum_{x\in\Sigma} f(x)|\phi(x)|.$$

- $\alpha(S) = \sum_{x\in\Sigma} f(x)|\phi(x)|$ is the average number of bits per symbol or compression factor.

Código Huffman

Los símbolos con baja frecuencia se colocarán más abajo que el símbolo con alta frecuencia.

Dadas las frecuencias f (x) para todo x ∈ Σ, el algoritmo mantiene una lista ordenada dinámica en una cola de prioridad Q. Construye un árbol de abajo hacia arriba:

- Inserta los símbolos como hojas con la clave f.
- Extrae los dos primeros elementos de Q y los une mediante un nuevo nodo virtual cuya clave es la suma de las f de sus hijos.
- Inserta el nuevo nodo en Q.

Cuando Q tenga el tamaño 1, el árbol resultante será el árbol de prefijos de un código de prefijo óptimo.

Coste -> O(n log n)

Algoritmos de aproximación

Muchas veces, el greedy produce una solución factible con un valor cercano a la solución óptima. En muchos casos prácticos, cuando es difícil encontrar el óptimo global, los greedy pueden producir una solución factible suficientemente buena: una aproximación a la solución óptima. Un algoritmo de aproximación para el problema siempre calcula una salida válida cercana. Las heurísticas también podrían dar buenas soluciones, pero no tienen una garantía teórica de cercanía. Greedy es una de las técnicas algorítmicas utilizadas para diseñar algoritmos de aproximaciones.

Árboles de expansión mínimos (MST)

- Un árbol en n nodos tiene n - 1 aristas.
- Cualquier grafo no dirigido conectado con n vértices y n - 1 aristas es un árbol.
- Un grafo no dirigido es un árbol si hay un único camino entre cualquier par de nodos.

Sea G = (V, E) un grafo no dirigido:

- G' = (V', E') es un subgrafo de G si V' ⊆ V y E' ⊆ E.
- Un subgrafo G' = (V', E') de G es de expansión si V' = V.
- Un árbol de expansión de G es un subgrafo de expansión que es un árbol.

Cualquier gráfico conexo tiene un árbol de expansión.

Problema del árbol de expansión mínimo -> dado un grafo con pesos, encontrar un árbol MST que minimice la suma de pesos.

- Un camino es una secuencia de consecutivos aristas.
- Un ciclo es un camino que termina en una arista conectada al vértice inicial y con ningún otro vértice repetido.
- Un corte es una partición de V en dos conjuntos S y V - S.
- El conjunto de corte de un corte es el conjunto de aristas con un extremo en S y el otro en V - S. cut(S, V − S) = {e = (u,v) ∈ E | u pertenece a S v no pertenece a S}

Un MST T en G tiene las siguientes propiedades:

- La propiedad de corte: e ∈ T ⇔ e es la arista con menos peso de algún corte en G.
- La propiedad del ciclo: e ∈ T ⇔ e es la arista más pesado en algún ciclo en G.

Los algoritmos MST usan dos reglas para agregar / descartar aristas.

La implicación ⇐ de la propiedad de corte produce la regla azul (incluir), que nos permite incluir de forma segura en T una arista de peso mínimo de algún corte identificado.

La implicación ⇒ de la propiedad del ciclo producirá la regla roja (excluir) que nos permite excluir de T una arista de peso máximo de algunos ciclos identificados.

Esquema greedy:

Dado G, aplicar las reglas rojo y azul hasta tener n - 1 aristas azules, estas serán las que forman el MST.

ALGORITMO 8: JARNIK-PRIM ALGORITHM

- Definición: encontrar un árbol MST (usa cola de prioridad Q)
- Coste:
  - Q an unsorted array -> O(|V|^2).
  - Q a heap -> O(|E| lg |V|).
  - Q a Fibonacci heap -> O(|E| + |V| lg |V|).

ALGORITMO 9: KRUSKAL ALGORITHM

- Definición: encontrar un árbol MST (usa estructura de datos UNION-FIND)
- Coste: O(m lg n).

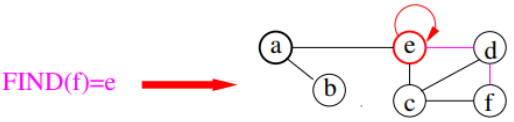
Uso de Union-Find para el algoritmo de Kruskal

Union-Find es una estructura de datos para mantener una partición dinámica de un conjunto. Es una de las estructuras de datos más elegantes del conjunto de herramientas algorítmicas. Union-Find hace posible diseñar algoritmos de tiempo casi lineales para problemas que de otro modo serían inviables.

Union-Find supports three operations on partitions of a set:
MAKESET (x): creates a new set containing the single element x.

UNION (x, y): Merge the sets containing x and y, by using their union.

FIND (x): Return the representative of the set containing x.



UNION (x, y) = UNION (FIND(x), FIND(y))

Algoritmo de Kruskal con UNION-FIND -> La ordenación tarda O(m log n). Y la parte que queda es una secuencia de n MAKESET y O(m) operaciones de FIND/UNION.

Usando un árbol enraizado por conjunto, en una UNION manteniendo el representante del conjunto más grande, y haciendo compresión de ruta durante un FIND.

- MAKESET tarda O (1).
- Cualquier secuencia entremezclada de k FIND y UNION tarda O (kα (n)).

α (n) es la función de Ackerman inversa que crece extremadamente lentamente. Para aplicaciones prácticas se comporta como una constante.

Algoritmo de Kruskal con UNION-FIND -> La ordenación tarda O(m log n). Y la parte que queda es una secuencia de n MAKESET y O(m) operaciones de FIND/UNION, n + O(mα(n)) = O(n + m). Y tarda O(n + m log n).

Clustering

Clustering: proceso de encontrar una estructura interesante en un conjunto de datos. Dada una colección de objetos, se les organiza en grupos coherentes similares con respecto a algunos (función de distancia d(·, ·)). La función de distancia no tiene por qué ser necesariamente la distancia física (euclidiana). La interpretación de d(·, ·) es que para dos objetos cualesquiera x, y, cuanto mayor sea d(x, y), menos similares serán x e y. Hay muchos problemas en la agrupación, pero para la mayoría de ellos, d (·, ·) debe tener una métrica: d(x, x) = 0 y d (x, y) > 0, para x != y; d(x, y) = d (y, x); d(x, y) + d(y, z) ≤ d(x, z). Si x, y son dos especies, podemos definir d(x, y) como los años en que divergieron en el curso de la evolución. Dado un conjunto de puntos de datos U = {x1, x2, ..., xn} junto con una función de distancia d en X, y dado un k> 0, un k-agrupamiento es una partición de X en k subconjuntos disjuntos.