# PRÁCTICA 2 – ESQUEMA ALGORÍTMICO GREEDY O VORAZ: PAVIMENTACIÓN DE CAMINOS

TITO JULIO GONZÁLEZ PÀDIAL, FRANCISCO JAVIER MOTA LÓPEZ, JOSÉ MIGUEL MARTÍN MARTÍN

# CONTENIDO

# INTRODUCCION A LOS ALGORITMOS VORACES

Los algoritmos conocidos como voraces , ávidos o glotones, son aquellos que en cada etapa de su ejecución toman todo lo que pueden sin pararse a analizar las consecuencias.
Se emplean en la resolución de problemas de optimización bajo unas determinadas condiciones, como se menciono anteriormente se va construyendo por etapas y en cada etapa toman la decisión con la información disponible hasta ese momento, una vez efectuada esta decisión no vuelve a ser replanteada.
- Existe un subconjunto de esos n candidatos que satisface ciertas restricciones: se llama solución parcial, factible o prometedora, esta solución es comprobada al final de cada etapa para saber si es la solución total del problema, si funciona bien la implementación del algoritmo la primera solución obtenida suele ser la óptima

Elementos de una solución "greedy":

- Conjunto de candidatos (elementos seleccionables correspondientes a las entradas del problema)
- Solución parcial (candidatos seleccionados)
- Función de selección (determina en cada etapa el mejor candidato del conjunto de seleccionables)
- Función de factibilidad (determina si es posible completar la solución parcial añadiendo candidatos para obtener la solución final)
- Función de solución o criterio que define lo que es una solución (indica si la solución parcial obtenida resuelve el problema)
- Función objetivo (valor de la solucion)

Análisis de tiempos de ejecución :

- n: número de elementos de C
- m: número de elementos de una solución
- Se repite como máximo n veces y como mínimo n que:
  - comprobar solución f(m) generalmente O(1) ó O(m)
  - selección de elemento nuevo g(n) entre O(1) y O(n)
  - solución parcial h(m)
  - unión de elementos j(n,m)

dando como resultado:
$t(n,m) \in O(n*(f(m)+g(n)+h(m)) + m*j(n, m))$

# ESTUDIO DE IMPLEMENTACION

## *Algoritmo de prim*

Teniendo en cuenta que el algoritmo de prim comienza por un vértice y escoge en cada etapa la arista de menor peso que verifique que uno de sus vértices se encuentre en el conjunto de vértices ya conectados y el otro no.

método de empleo:

- Aplica reiteradamente la propiedad de los árboles de recubrimiento de coste mínimo, incorporando en cada paso una arista.
- Se usa un conjunto U de vértices tratados y se selecciona en cada paso la arista mínima que une un vértice de U con otro de su complementario.

- Función objetivo a minimizar: longitud (coste) del árbol de recubrimiento.
- Candidatos: las aristas del grafo.
- Función solución: el conjunto de aristas seleccionado es árbol de recubrimiento de longitud mínima.
- Función factible: el conjunto de aristas no contiene ciclos.
- Función de selección:
  - Seleccionar la arista de menor peso que aún no ha sido seleccionada y que forme un árbol junto con el resto de aristas seleccionadas (Algoritmo de Prim).

PARA UN GRAFO CON N NODOS Y A ARISTAS:
·PRIM:O(N^2)
EN UN GRAFO DENSO SE CUMPLE QUE: A -> N(N-1)/2
· PRIM PUEDE SER MEJOR (DEPENDE DEL VALOR DE LAS CONSTANTES
OCULTAS).
EN UN GRAFO DISPERSO SE CUMPLE QUE: A -> N
· PRIM ES MENOS EFICIENTE

# ESTUDIO DE IMPLEMENTACION

## *Algoritmo de prim*

para la implementación de este algoritmo hemos hecho uso de estructuras arrayList para almacenar los distintos tipos de datos a tratar ya sean listas de nodos o aristas para poder así alcanzar el objetivo de rellena el árbol de recubrimiento yendo de nodo a nodo añadiendo sus arista al conjunto solución comprobando que no se forme ningún ciclo, haciendo uso del método encontrar arista que aglomera las aristas del nodo y va comprobando que sea la de menor coste y el método findPAth para encontrar el camino de aristas y asegurar que se ha completado bien el recorrido.

# ESTUDIO DE IMPLEMENTACION

## *Algoritmo de kruskal*

En el algoritmo Kruskal, se ordenan primero las aristas por orden creciente de peso, y en cada etapa se decide qué hacer con cada una de ellas. Si la arista no forma un ciclo con las ya seleccionadas para la solución, se incluye en ella; si no, se descarta.

- Función objetivo a minimizar: longitud (coste) del árbol de recubrimiento.
- Candidatos: las aristas del grafo.
- Función solución: el conjunto de aristas seleccionado es árbol de recubrimiento de longitud mínima.
- Función factible: el conjunto de aristas no contiene ciclos.
- Función de selección:
  - Seleccionar la arista de menor peso que aún no ha sido seleccionada y que no forme un ciclo (Algoritmo de Kruskal).

método de empleo:

- Inicialmente disponemos de todos los vértices del grafo y ninguna arista seleccionada, por lo cual cada uno de los vértices está asignado a una componente conexa distinta (el propio vértice aislado).
- Conforme se van añadiendo aristas en cada paso del algoritmo, el número de componentes conexas va disminuyendo, y los vértices van siendo asignados a éstas.
- Al igual que el de Prim, se basa también en la propiedad de los árboles de recubrimiento de coste mínimo: partiendo del árbol vacío, se selecciona en cada paso la arista de menor peso que aún no haya sido seleccionada y que no conecte dos nodos de la misma componente conexa (es decir, que no forme un ciclo).
- En cada momento, los vértices que están dentro de una componente conexa en la solución forman una clase de equivalencia, y el algoritmo se puede considerar como la fusión continuada de clases hasta obtener una única componente con todos los vértices del grafo.

PARA UN GRAFO CON N NODOS Y A ARISTAS:
· KRUSKAL: O(A LOG N)
EN UN GRAFO DENSO SE CUMPLE QUE: A -> N(N-1)/2
· KRUSKAL -> O(N^2 LOG N)
EN UN GRAFO DISPERSO SE CUMPLE QUE: A -> N
· KRUSKAL -> O(N LOG N)

# ESTUDIO DE IMPLEMENTACION

## *Algoritmo de kruskal*

para la implementación de este algoritmo hemos tenido en cuenta como se ha expuesto con anterioridad, que vamos rellenando el árbol de recubrimiento con las aristas de menos coste mientras no se forme un ciclo dentro del árbol, asegurando así que recorremos todos los vértices del grafo usando las aristas de menor coste posible en cada etapa. Para ello, y usando una clase java para cada elemento (nodo, arista, grafo) y una clase llamada enlace podemos poner en marcha la implementación del algoritmo.

La primera consideración fue lidiar con los ciclos para ello se uso un método que se encarga de comprobar si existe o no dicho ciclo en el árbol al que se llama para cada etapa dentro del método del algoritmo, donde cada iteración entra una nueva arista al árbol y si no se forma ciclo se queda como parte del conjunto solución y si no se va fuera. para ello el método aplicarkruskal hace uso de ArrayList<String> de los nombres de cada nodo un grafo que sera nuestro árbol de recubrimiento y una ArrayList<Arista> llamada L donde almacenaremos todas las aristas del grafo en orden ascendente para ir  sacando de alli comprobando si hay ciclo y añadirlas al conjunto solución arbol en esta clase además se incluye el método de comprobación de ciclos donde usamos de nuevo un arrayList auxiliar de Enlace en este caso, este método adquiere todos los posibles enlaces de un nodo los guarda en la lista si la lista esta vacía devuelve falso ya que no puede haver ciclo si el nodo no tiene enlaces, y si no comprueba la existencia de enlaces con el nodo inicial de la arista a verificar devolviendo true si hubiese enlace.

Posteriormente se instancia un bucle for donde vamos pasando de enlace a enlace para finalmente hacer una llamada recursiva al metodo para una última comprobación.

teniendo así en el grafo arbol el conjunto de aristas de tamaño n-1 siendo n el numero de nodos que cumple con las características especificadas

# ESTUDIO TEÓRICO

## *Algoritmo de Prim*

Como bien sabemos el orden de complejidad de este algoritmo en el caso de EDAland tanto reducida como ampliada es de O(n^2) siendo para la red reducida de O(21^2), es decir, 441 y para la extendida O(1053^2) siendo de 1108809 en el caso del grafo visto en clase tenemos 12 aristas y 7 nodos por lo que el orden sería de O(n^2) dando como resultado 49 por lo que claramente obtiene a priori unos valores bastante grandes incluso

# ESTUDIO TEÓRICO

## *Algoritmo de kruskal*

Como bien sabemos el orden de complejidad de este algoritmo es en su peor caso del orden O(n^2 log n) y en el mejor O(a log n) siempre y cuando nuestro grafo a tratar no cumpla que a(número de aristas) sea n(n-1)/2 en el caso de EDAland reducida tenemos 21 nodos y 29 aristas por lo que el orden de complejidad del algoritmo para este problema debería ser de O(29 log 21) es decir aproximadamente 38,344 para la red extendida disponemos de 1053 nodos y 2017 aristas por lo que de nuevo tenemos el mejor de los casos siendo del orden de O(2017 log 1053) siendo 6096,238. Para el caso del grafo de las transparencias de clase podemos ver que contamos con 7 nodos y 12 aristas por lo que de nuevo tenemos un orden de O(12 log 7) cuyo valor es 10, 141 siendo estos valores bastante mas reducidos que los del algoritmo de prim.

# FUENTES

## *Páginas consultadas*

- Tema 3 algoritmos voraces EDAII Curso 20/21
- Youtube.com
- sites.google.com