Práctica 5: Recorrido mínimo

Compilado: 29 de mayo de 2024

Algoritmo de Dijkstra

1. Dado un digrafo D con pesos $c\colon E(D) \to \mathbb{N}$ y dos vértices s y t, decimos que una arista $v \to w$ es *st-eficiente* cuando $v \to w$ pertenece a algún camino mínimo de s a t. Sea $d(\cdot, \cdot)$ la función que indica el peso de un camino mínimo entre dos vértices.

   a. Demostrar que $v \to w$ es *st*-eficiente si y sólo si $d(s, v) + c(v \to w) + d(w, t) = d(s, t)$.

   b. Usando el inciso anterior, proponga un algoritmo eficiente que encuentre el mínimo de los caminos entre s y t que no use aristas *st*-eficientes. Si dicho camino no existe, el algoritmo retorna $\perp$.

2. Diseñar un algoritmo eficiente que, dado un digrafo G con pesos no negativos, dos vértices s y t y una cota c, determine una arista de peso máximo de entre aquellas que se encuentran en algún recorrido de s a t cuyo peso (del recorrido, no de la arista) sea a lo sumo c. **Demostrar** que el algoritmo propuesto es correcto.

3. Diseñar un algoritmo eficiente que, dado un digrafo pesado G y dos vértices s y t, determine el recorrido minimo de s a t que pasa por a lo sumo una arista de peso negativo. **Demostrar** que el algoritmo propuesto es correcto.

4. Sea G un digrafo con pesos positivos que tiene dos vértices especiales s y t. Para una arista $e \notin E(G)$ con peso positivo, definimos $G + e$ como el digrafo que se obtiene de agregar e a G. Decimos que e mejora el camino de s a t cuando $d_G(s, t) > d_{G+e}(s, t)$. Diseñar un algoritmo eficiente que, dado un grafo G y un conjunto de aristas $E \notin E(G)$ con pesos positivos, determine cuáles aristas de E mejoran el camino de s a t en G. **Demostrar** que el algoritmo es correcto.

5. Sea G un digrafo con pesos positivos que tiene dos vértices especiales s y t. Decimos que una arista $e \in E(G)$ es *crítica* para s y t cuando $d_G(s, t) < d_{G-e}(s, t)$. Diseñar un algoritmo eficiente que, dado G, determine las aristas de G que son críticas para s y t. **Demostrar** que el algoritmo es correcto. **Ayuda:** pensar en el subgrafo P de G que está formado por las aristas de caminos mínimos de G (el "grafo de caminos mínimos").

6. En muchas aplicaciones se necesita encontrar caminos de *peso multiplicativo* mínimo en un digrafo D pesado con una función positiva $c\colon E(G) \to \mathbb{R}_{>1}$. Formalmente, el peso multiplicativo de un camino $v_1, \ldots, v_k$ es la multiplicatoria de los pesos de sus aristas. Este tipo de caminos se buscan, por ejemplo, cuando los pesos de las aristas representan probabilidades[1] de eventos independientes y se quiere encontrar una sucesión de eventos con probabilidad máxima/mínima. Modelar el problema de camino de peso multiplicativo mínimo como un problema de camino mínimo. **Demostrar** que el algoritmo es correcto. **Ayuda:** transformar el peso de cada arista usando una operación conocida que sea creciente y transforme cualquier multiplicatoria en una sumatoria.

[1] No se incluye un ejercicio en particular de este tema debido a que Estadística Computacional no es correlativa con TDA.

Algoritmo de Bellman-Ford y SRDs

7. Para organizar el tráfico, la ciudad de Ciclos Positivos ha decidido implementar las cabinas de peaje inverso. La idea de estas cabinas es incentivar la circulación de vehículos por caminos alternativos, estableciendo un monto que se le paga a le conductore de un vehículo cuando pasa por la cabina. Estas cabinas inversas se suman a las cabinas regulares, donde le conductore paga por pasar por la cabina. La ciudad sabe que estas nuevas cabinas pueden dar lugar al negocio del "ciclo puré", que consiste en transitar *eternamente* por la ciudad a fin de obtener una ganancia que tienda a infinito. Para evitar esta situación, que genera costos y tráfico adicional, lo cual será aprovechado para desgastar a la administración ante la opinion pública, la ciudad quiere evaluar distintas alternativas antes de llevar las cabinas inversas a la práctica.

   a. Modelar el problema de determinar si la ciudad permite el negocio del ciclo puré cuando el costo de transitar por cada cabina i de peaje es c_i ($c_i < 0$ si la cabina es inversa) y el costo que cuesta viajar de forma directa de cada cabina i a cada cabina j es $c_{ij} > 0$.
   b. Dar un algoritmo para resolver el problema del inciso anterior, indicando su complejidad temporal.

   El sistema arrojó que ninguna de las configuraciones deseadas para desincentivar el tráfico evita el negocio de los ciclos puré. Desafortunadamente, el plan se filtró a la prensa y comenzaron las peleas mediáticas. A fin de obtener cierto rédito, desde el departamento de *marketing* sugieren transformar la idea de cabinas inversas en cabinas mixtas. Cuando un vehículo pasa por una cabina mixta, se le paga a le conductore si se le cobró a le conductore en la cabina anterior; caso contrario, le conductore paga. Obviamente, desde *marketing* sugieren que se le pague a le conductore cuando la cabina mixta sea la primera cabina recorrida para bajar los malos humores.

   c. Modelar el problema de determinar si la ciudad permite el negocio de los ciclos puré cuando se aplica la nueva configuración para las cabinas. Además de la información utilizada para el problema original, ahora se conoce cuáles cabinas son mixtas: notar que el monto de cobro es c_i y el monto de pago es $-c_i$ para la cabina mixta i (con $c_i > 0$).

8. Un *sistema de restricciones de diferencias (SRD)* es un sistema $\mathcal{S}$ que tiene m inecuaciones y n incógnitas $x_1, \ldots, x_n$. Cada inecuación es de la forma $x_i - x_j \leq c_{ij}$ para una constante $c_{ij} \in \mathbb{R}$; por cada par i, j existe a lo sumo una inecuación (por qué?). Para cada SRD $\mathcal{S}$ se puede definir un digrafo pesado $D(\mathcal{S})$ que tiene un vértice v_i por cada incógnita x_i de forma tal que $v_j \to v_i$ es una arista de peso c_{ij} cuando $x_i - x_j \leq c_{ij}$ es una inecuación de $\mathcal{S}$. Asimismo, $\mathcal{S}$ tiene un vértice v_0 y una arista $v_0 \to v_i$ de peso 0 para todo $1 \leq i \leq n$.

   a. Demostrar que si $D(\mathcal{S})$ tiene un ciclo de peso negativo, entonces $\mathcal{S}$ no tiene solución.
   b. Demostrar que si $D(\mathcal{S})$ no tiene ciclos de peso negativo, entonces $\{x_i = d(v_0, v_i) \mid 1 \leq i \leq n\}$ es una solución de $D(\mathcal{S})$. Acá $d(v_0, v_i)$ es la distancia desde v_0 a v_i en $D(\mathcal{S})$.
   c. A partir de los incisos anteriores, proponer un algoritmo que permita resolver cualquier SRD. En caso de no existir solución, el algoritmo debe mostrar un conjunto de inecuaciones contradictorias entre sí.

9. Sea S una cadena con n paréntesis que abren y n paréntesis que cierran. Dada una longitud ℓ impar, decimos que $s\colon \{1, \ldots, n\} \to \mathbb{N}$ es un *ℓ-posicionamiento uniforme* de S si $s(i)$ es par y

al escribir el i-ésimo paréntesis que abre en $s(i)$ y el i-ésimo paréntesis que cierra en $s(i)+\ell$, $1 \leq i \leq n$, se obtiene una escritura válida de S. Por ejemplo, si $S = ((()()())())()$ y $\ell = 15$, entonces $s(1) = 0$, $s(2) = 6$, $s(3) = 10$, $s(4) = 16$, $s(5) = 20$, $s(6) = 22$ y $s(7) = 36$ es un 15-posicionamiento uniforme de S. Definir un SRD que permita resolver el problema de determinar si una cadena dada S tiene un ℓ-posicionamiento uniforme cuando $\ell > 0$ impar también es dado. El mejor SRD que conocemos tiene $O(n)$ inecuaciones y, por lo tanto, permite resolver el problema en $O(n^2)$ de aplicando el algoritmo del ejercicio anterior.

10. Tenemos a n clientes de un supermercado $\{c_1, c_2, ..., c_n\}$ y queremos asignarle a cada uno, una caja para hacer fila. Las cajas están ordenadas en una línea y numeradas de izquierda a derecha de la 1 a la M y se encuentran separadas por pasillos. Durante el proceso de asignación algunos clientes se pelean entre sí y son separados por seguridad. Si dos clientes c_i y c_j pelean, los guardias les dicen que tienen que ponerse en filas distintas que se encuentren separadas por $K_{ij} > 0$ pasillos intermedios, para que no se vuelvan a pelear. Notar que cuando seguridad separa una pelea naturalmente hay un cliente que queda más a la izquierda (cerca de la caja 1) y el otro más a la derecha (cerca de la caja M). Con la restricción de no volver a acercarse, ese orden ya no puede cambiar. A su vez hay pares de clientes c_k y c_m que son amigos y no queremos que haya más que $L_{km} = L_{mk} \geq 0$ pasillos intermedios entre las filas de c_k y c_m. ¿será posible asignarlos a todos?

    a. Modelar el problema utilizando un sistema de resticciones de diferencias (no olviden justificar).

    b. Proponer un algoritmo polinomial que lo resuelva.

    c. ¿Qué complejidad tiene el algoritmo propuesto? Para la respuesta, tener en cuenta la cantidades m_1 y m_2 de amistades y peleas, respectivamente.

    Nota: K_{ij} de alguna manera captura la intensidad de la pelea y L_{ij} captura (inversamente) la intensidad de la amistad. Es posible que dos amigos se peleen y en ese caso hay que cumplir las dos condiciones. Si eso pasa solo puede haber soluciones si $K_{ij} \leq L_{ij}$. Para todo par de clientes sabemos si son amigos o si se pelearon, la intensidad de cada relación. Además, para aquellos clientes que se pelearon, conocemos cuál cliente quedó a la izquierda y cuál a la derecha.
    Ayuda: Si tenemos n variables x_i en un SRD y queremos acotarlas entre A y B ($x_i \in [A, B]$) podemos agregar una variable auxiliar z, sumar restricciones del tipo $A \leq x_i - z \leq B$ y luego correr la solución para que z sea 0.

11. Nuevamente tenemos a n clientes de un supermercado $\{c_1, c_2, \ldots, c_n\}$ y queremos asignarle a cada uno una caja para hacer fila. Esta vez, las cajas están ordenadas en forma circular, numeradas de la 1 a la M y se encuentran separadas por pasillos. Entre la caja M y la 1 hay una valla que impide pasar de una a la otra. Durante el proceso de asignación algunos clientes se pelean entre sí y son separados por seguridad. Si dos clientes c_i y c_j se pelean, los guardias les dicen que tienen que ponerse en filas distintas que se encuentren separadas por al menos $K_{ij} > 0$ pasillos intermedios en ambos sentidos del círculo, para que no se vuelvan a pelear. Notar que cuando seguridad separa una pelea naturalmente hay un cliente que queda en un número de caja más bajo y el otro en un número de caja más alto. Con la restricción de no volver a acercarse y la valla entre las cajas M y 1 ese orden ya no puede cambiar. ¿Será posible asignarlos a todos?

    a. Modelar el problema utilizando un sistema de restricciones de diferencias. Para el modelo, notar que sabemos qué clientes se pelearon. Más aún, si c_i y c_j se pelearon, sabemos quién

entre c_i y c_j quedó del lado de las cajas con menor numeración. En este escenario no hay restricciones por amistad.

b. Proponer un algoritmo polinomial que lo resuelva.

c. ¿Qué complejidad tiene el algoritmo propuesto? Para la respuesta, tener en cuenta la cantidad m_1 de peleas.

12. Adaptar el modelo del Ejercicio 8 para el caso en que se reemplazan las inecuaciones por ecuaciones; en este caso, el sistema se describe con un grafo $G(\mathcal{S})$ en lugar del digrafo $D(\mathcal{S})$. Luego, considerar el problema C del Torneo Argentino de Programación 2017[2], cuyo texto se copia abajo.

a. Modelar el problema con un SRD $\mathcal{S}$ que utilice únicamente ecuaciones. En el mejor modelo que conocemos cada incógnita aparece una única vez en forma positiva y una única vez en forma negativa y, por lo tanto, cada componente de $G(\mathcal{S})$ es un ciclo.

b. Proponer un algoritmo de tiempo $O(n)$ para resolver el problema, donde n es la cantidad de incógnitas en $\mathcal{S}$.

Carolina tiene la costumbre de juntarse todas las tardes a tomar mate con sus amigos. Como han vivido equivocados toda su vida, les gusta tomar mate dulce. Últimamente, se han preocupado por su ingesta calórica y han decidido probar un nuevo edulcorante cero calorías que salió al mercado: el Ingrediente Caramelizador de Productos Cebables (ICPC). El ICPC tiene la extraña propiedad de que al aplicarlo dura exactamente K cebadas endulzando el mate y luego se evapora completamente.

Carolina y sus amigos se ubican alrededor de una mesa circular, y se numeran del 0 al $N-1$ en el sentido de las agujas del reloj. Luego comienzan a tomar mate durante varias rondas. En cada ronda, ella ceba un mate para cada integrante, comenzando por la persona 0 y continuando en orden ascendente hasta llegar a la persona $N-1$. Por lo tanto, luego de que toma la persona $N-1$ es nuevamente el turno de la persona 0. Carolina decide una cantidad fija entera y positiva E_i de ICPC para agregar al mate antes de cebar a la persona i. La cantidad de ICPC que recibe cada persona en su mate será entonces la suma de lo agregado por la cebadora en las últimas K cebadas. Formalmente, la cantidad de ICPC que recibe la persona i a partir de la segunda ronda es

$$T_i = \sum_{d=0}^{K-1} E_{(i-d) \text{ mód } N}$$

donde x mód N es un entero entre 0 y $N-1$ que indica el resto de x en la división entera por N.

Por ejemplo, si la ronda constara de $N = 5$ amigos, la duración del edulcorante fuera de $K = 3$ cebadas y las cantidades de ICPC agregado fueran $E_0 = 10$, $E_1 = 4$, $E_2 = 0$, $E_3 = 2$ y $E_4 = 1$, entonces las cantidades de ICPC que recibirían los amigos serían $T_0 = 13$, $T_1 = 15$, $T_2 = 14$, $T_3 = 6$ y $T_4 = 3$.

Carolina conoce muy bien los gustos de sus amigos y quisiera complacerlos a todos. Dado un arreglo $G_0, G_1, \ldots, G_{N-1}$ con las cantidades de edulcorante que quieren recibir los N amigos, ustedes deben determinar si existe un arreglo $E_0, E_1, \ldots, E_{N-1}$

[2] Autor: Nicolás Álvarez, Universidad Nacional del Sur

con las cantidades de ICPC a agregar antes de cebar a cada persona, tal que a partir de la segunda ronda todos los amigos estén satisfechos (esto es, $T_i = G_i$ para $i = 0, 1, \ldots, N-1$).

Algoritmo de Floyd-Warshall y recorrido mínimo en DAGs

13. Decimos que una matriz cuadrada, simétrica y positiva $M \in \mathbb{N}^2$ es de *Floyd-Warshall (FW)* si existe un grafo G tal que M es el resultado de aplicar FW a G. Describir un algoritmo para decidir si una matriz M es FW. En caso afirmativo, el algoritmo debe retornar un grafo G con la mínima cantidad de aristas posibles tal que el resultado de FW sobre G sea M. En caso negativo, el algoritmo debe retornar alguna evidencia que pruebe que M no es FW.

14. Dado un digrafo D con pesos $c\colon E(D) \to \mathbb{R}$ que no tiene ciclos de peso negativo, queremos encontrar la arista $v \to w$ que sea *st*-eficiente para la mayor cantidad de pares s y t. Proponer un algoritmo eficiente y "simple de programar" para resolver este problema. **Ayuda:** verificar que la propiedad del Ejercicio 1*a* también es cierta en este caso.

15. Dados dos vértices v y w de un grafo pesado G, el *intervalo* entre v y w es el conjunto $I(v, w)$ que contiene a todos los vértices que están en algún recorrido mínimo entre v y w. Un conjunto de vértices D es *geodésico* cuando $\bigcup_{v,w \in D} I(v, w) = V(G)$. Diseñar e implementar un algoritmo de tiempo $O(n^3)$ que, dado un grafo pesado y conexo G y un conjunto de vértices D de G, determine si D es geodésico.

16. Sea D un digrafo conexo que no tiene ciclos dirigidos, v el único vértice de D con grado de entrada 0[3] y $c\colon E(D) \to \mathbb{Z}$ una función de pesos.

    a. Definir una función recursiva $d\colon V(D) \to \mathbb{Z}$ tal que $d(w)$ es el peso del camino mínimo de v a w para todo $w \in V(D)$. **Ayuda:** considerar que el camino mínimo de v a w se obtiene yendo de v hacia z y luego tomando la arista $z \to w$, para algún vecino de entrada z de w; notar que la función recursiva está bien definida porque D no tiene ciclos.

    b. Diseñar un algoritmo de programación dinámica top-down para el problema de camino mínimo en digrafos sin ciclos y calcular su complejidad.

    c. (Integrador y opcional) Diseñar un algoritmo de programación dinámica bottom-up para el problema. **Ayuda:** computar d de acuerdo a un orden topológico $v = v_1, \ldots, v_n$ donde $v_i \to v_j$ solo si $i < j$. Este orden se puede computar en $O(n + m)$ (guía 3).

17. En el *problema del vuelto* tenemos una cantidad ilimitada de monedas de distintos valores $w_1, \ldots, w_k$ y queremos dar un vuelto v utilizando la menor cantidad de monedas posibles (ver Teórica 2). Por ejemplo, si los valores son $w_1 = 1$, $w_2 = 5$, y $w_3 = 12$ y el vuelto es $v = 15$, entonces el resultado es 3 ya que alcanza con dar 3 monedas de \$5. Modelar este problema como un problema de camino mínimo e indicar un algoritmo eficiente para resolverlo. El algoritmo sobre el modelo debe tener complejidad $O(vk)$. **Opcional:** discutir cómo se relaciona este modelo con el algoritmo de programación dinámica correspondiente.

18. En el *problema de gestión de proyectos* tenemos un proyecto que se divide en n etapas $v_1, \ldots, v_n$. Cada etapa v_i consume un tiempo $t_i \geq 0$. Para poder empezar una etapa v_i se requiere que

[3] Dicho vértice siempre existe; ver guías 2 y 3.

primero se hayan terminado un conjunto $N(v_i)$ de etapas v_j tales que $j < i$. Por simplicidad, la etapa v_1 se usa como indicador de inicio del proyecto y, por lo tanto, consume un tiempo $t_1 = 0$ y es requerida por todas las otras etapas. Análogamente, la etapa v_n indica el final del proyecto por lo que consume tiempo $t_n = 0$ y requiere la finalización del resto de las etapas. Una etapa es *crítica* cuando cualquier atraso en la misma provoca un retraso en la finalización del proyecto. Modelar el problema de encontrar todas las etapas críticas de un proyecto como un problema de camino mínimo e indicar qué algoritmo usaría para resolverlo. El mejor algoritmo que conocemos toma tiempo lineal en la cantidad de datos necesarios para describir un proyecto.

Invariantes de los algoritmos

19. Solo para este ejercicio, la clase de los grafos *densos* está formada por todos los grafos con $\Omega(n^2)$ aristas, mientras que la clase de los grafos *ralos* está formada por todos los grafos con $O(n)$ aristas. Justificar qué algoritmo de camino mínimo conviene usar para cada uno de los siguientes problemas, explicitando su implementación:

    a. Encontrar un camino mínimo de uno a todos en un grafo ralo (resp. denso) cuyos pesos son todos iguales y no negativos.

    b. Encontrar un camino mínimo de todos a todos en un grafo ralo (resp. denso) cuyos pesos son todos iguales y no negativos.

    c. Encontrar un camino mínimo de uno a todos en un grafo ralo (resp. denso) cuyos pesos son no negativos.

    d. Encontrar un camino mínimo de todos a todos en un grafo ralo (resp. denso) cuyos pesos son no negativos.

    e. Determinar si un grafo ralo (resp. denso) tiene ciclos de peso negativo; no suponer que el grafo es conexo.

    f. Encontrar un recorrido mínimo de uno a todos en un grafo ralo (resp. denso).

    g. Encontrar un recorrido mínimo de todos a todos en un grafo ralo (resp. denso).

20. Se tiene un digrafo D donde cada arista $v \to w$ representa un camino directo de una locación v a otra w. La arista $v \to w$ tiene un *tiempo de viaje* $t(v \to w)$ que indica que si un vehículo parte de v en el instante t, entonces llega a w en el instante $t + t(v \to w)$. Asimismo, $v \to w$ tiene un *tiempo de apertura* $s(v \to w) \geq 0$ que indica que ningún vehículo puede empezar a cruzar $v \to w$ antes del instante $s(v \to w)$. Dado un vértice v y para todo $w \in V(G)$, queremos determinar el instante más temprano $d(w) \geq 0$ en que un vehículo puede llegar a w si empieza su recorrido en v (Figura 1).

    a. Suponiendo que $t(\cdot) \geq 0$, diseñar un algoritmo eficiente para el problema que esté basado en el algoritmo de Dijkstra; justificar su correctitud. **Ayuda:** mantener como invariante una partición V, W de $V(G)$ tal que: $d(x)$ es correcto para todo $x \in V$ y $d(y) \geq d(x)$ para todo $y \in W$. Luego, determinar qué vértice de W puede agregarse a V satisfacendo el invariante.

    b. Sin suponer que $t(\cdot) \geq 0$ (i.e., algunas aristas "vuelven el tiempo atrás") pero suponiendo que $t(C) \geq 0$ para todo ciclo C, diseñar un algoritmo eficiente para el problema que esté basado en el algoritmo de Bellman-Ford; justificar su correctitud. **Ayuda:** definir una función recursiva $d^k(w)$ que para cada w indique (una cota inferior de) qué tan temprano se puede llegar a w si se permiten recorrer hasta k aristas.

c. Justifique brevemente si el algoritmo de Floyd-Warshall se puede adaptar en forma sencilla para la versión todos a todos del problema.

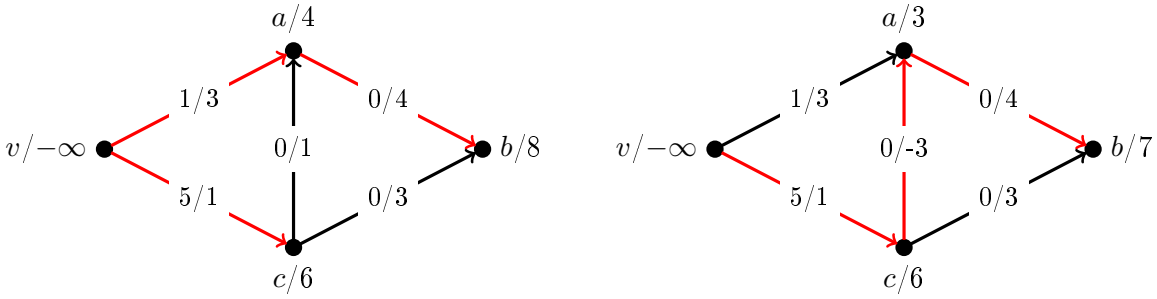

Figura 1. *Ejemplos de redes con tiempos de apertura en la que los caminos mínimos están formados por las aristas en rojo. Cada arista $x \to y$ está etiquetada con $s(x \to y)/t(x \to y)$ mientras que cada vértice x está etiquetado con $x/d(x)$*