Algoritmos y Estructuras de Datos II

Algoritmo de Dijkstra

13 de mayo de 2015

Clase de hoy

Camino de costo mínimo

- Sea $G = (V, A)$ un grafo dirigido con costos no negativos en sus aristas, y sea $v \in V$ uno de sus vértices.
- Se busca obtener los caminos de menor costo desde v hacia cada uno de los demás vértices.

Clase de hoy

1. Problema: camino de costo mínimo
2. Algoritmo de Dijkstra
   - Idea del algoritmo
   - El algoritmo
   - Fundamentación
   - Calculando el camino de costo mínimo

Algoritmo de Dijkstra

Idea

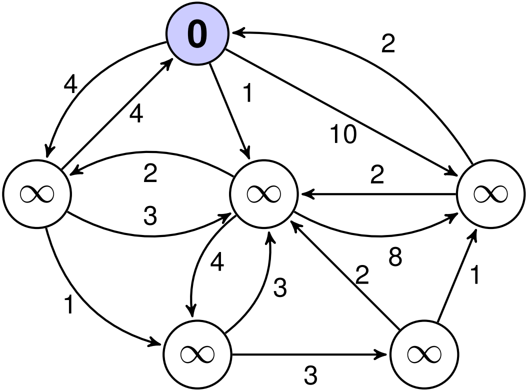
- El algoritmo de Dijkstra realiza una secuencia de n pasos, donde n es el número de vértices.
- En cada paso, “aprende” el camino de menor costo desde v a un nuevo vértice.
- A ese nuevo vértice lo pinta de azul.
- Tras esos n pasos, conoce los caminos de menor costo a cada uno de los vértices.

Algoritmo de Dijkstra
Ejemplo

- Tratemos de entenderlo a través de un ejemplo.
- En casa paso, en los vértices azules anotamos el costo del camino de menor costo de v a ese vértice.
- En casa paso, en los vértices blancos anotamos el costo del camino azul de menor costo de v a ese vértice.
- Un camino azul es uno que a lo sumo tiene al vértice destino blanco, sus otros vértices son azules.

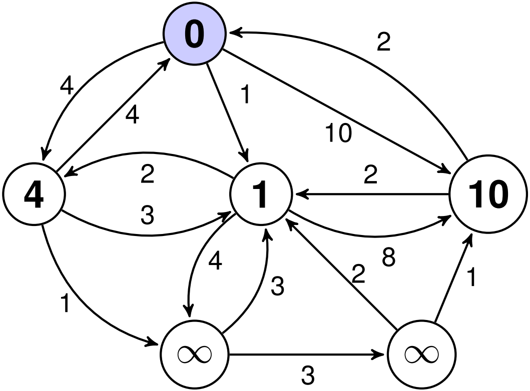
Algoritmo de Dijkstra

Paso 1 (a): sabemos lo que cuesta llegar de v a v



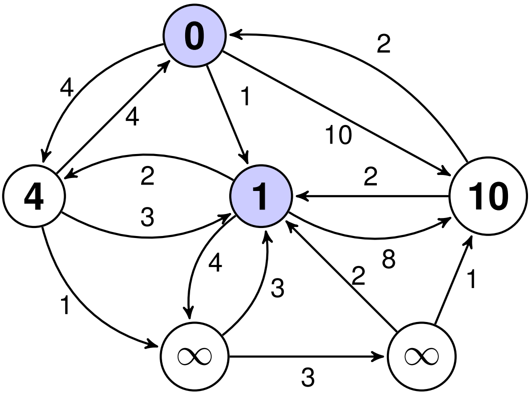
Algoritmo de Dijkstra

Paso 1 (b): Actualizamos los costos de los caminos azules óptimos



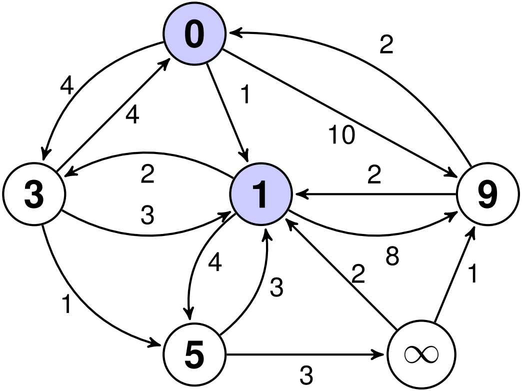
Algoritmo de Dijkstra

Paso 2 (a): sabemos lo que cuesta llegar de v a un nuevo vértice



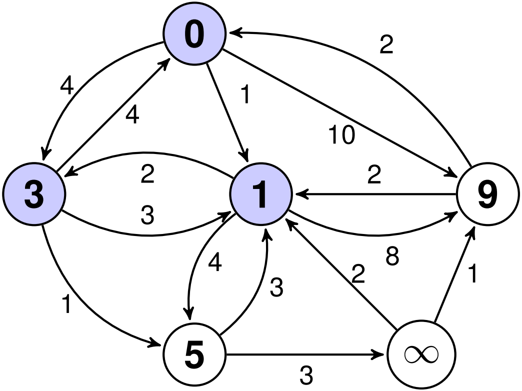
Algoritmo de Dijkstra

Paso 2 (b): Actualizamos los costos de los caminos azules óptimos



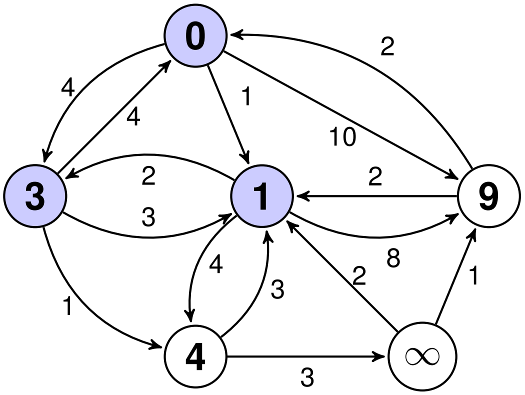
Algoritmo de Dijkstra

Paso 3 (a): sabemos lo que cuesta llegar de *v* a un nuevo vértice



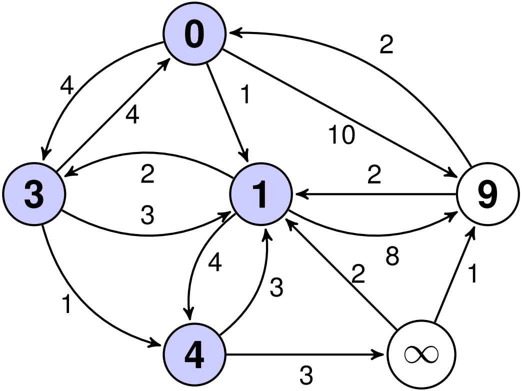
Algoritmo de Dijkstra

Paso 3 (b): Actualizamos los costos de los caminos azules óptimos



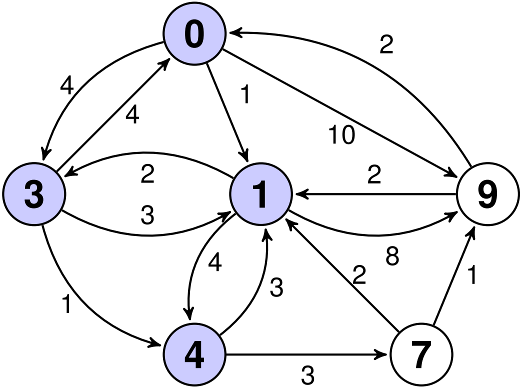
Algoritmo de Dijkstra

Paso 4 (a): sabemos lo que cuesta llegar de *v* a un nuevo vértice



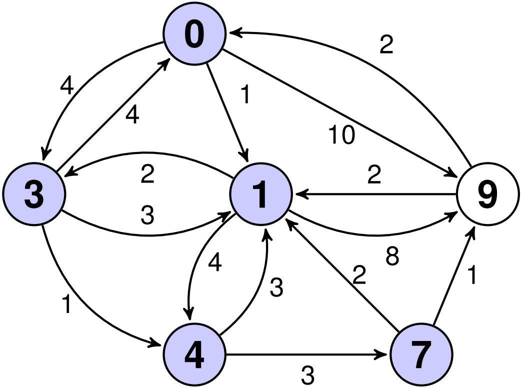
Algoritmo de Dijkstra

Paso 4 (b): Actualizamos los costos de los caminos azules óptimos



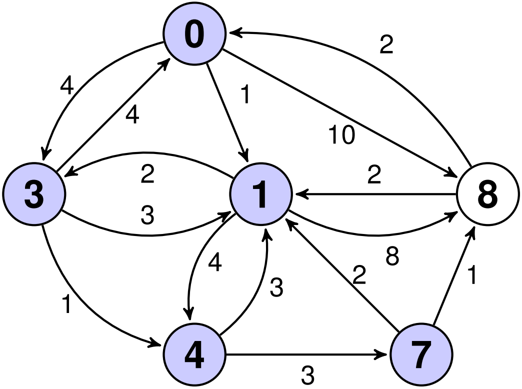
Algoritmo de Dijkstra

Paso 5 (a): sabemos lo que cuesta llegar de v a un nuevo vértice



Algoritmo de Dijkstra

Paso 5 (b): Actualizamos los costos de los caminos azules óptimos



Clase de hoy

El algoritmo

- Asumiremos que el grafo viene dado por el conjunto de vértices $V = \{1, 2, \dots, n\}$
- y los costos por una matriz L : **array**[1..n,1..n] **of** costo,
- que en $L[i, j]$ mantiene el costo de la arista que va de i a j.
- En caso de no haber ninguna arista de i a j, $L[i, j] = \infty$.
- Asumimos $L[j, j] = 0$.
- El algoritmo funciona también para grafos no dirigidos, simplemente se tiene $L[i, j] = L[j, i]$ para todo par de vértices i y j.

El algoritmo

Versión simplificada

- En vez de hallar el camino de costo mínimo desde *v* hasta cada uno de los demás, halla sólo el **costo** de dicho camino.
- Es decir, halla el costo del camino de costo mínimo desde *v* hasta cada uno de los demás.
- El resultado estará dado por un arreglo *D*: **array**[1..n] **of** costo,
- en *D*[*j*] devolverá el costo del camino de costo mínimo que va de *v* a *j*.
- El conjunto *C* es el conjunto de los vértices hacia los que todavía desconocemos cuál es el camino de menor costo.

Algoritmo de Dijkstra

```
fun Dijkstra(L: array[1..n,1..n] of costo, v: nat)
                                        ret D: array[1..n] of costo
    var c: nat
    C:= {1,2,...,n}-{v}
    for j:= 1 to n do D[j]:= L[v,j] od
    do n-2 times → c:= elemento de C que minimice D[c]
                   C:= C-{c}
                   for j in C do D[j]:= min(D[j],D[c]+L[c,j]) od
    od
end fun
```

Clase de hoy

1. Problema: camino de costo mínimo

2. Algoritmo de Dijkstra
   - Idea del algoritmo
   - El algoritmo
   - Fundamentación
   - Calculando el camino de costo mínimo

Vértices especiales

- Llamamos **vértices especiales** a los que no pertenecen a C.
- O sea, a los pintados de azul en nuestra animación anterior.
- Inicialmente el único vértice especial es v.
- Un **camino especial** es un camino cuyos vértices son especiales salvo quizá el último.
- Inicialmente, los caminos especiales son el camino vacío (que va de v a v y tiene costo $L[v, v] = 0$)
- y las aristas que van de v a j que tienen costo $L[v, j]$.

Idea del algoritmo

- En todo momento, D mantiene en cada posición j, el costo del camino **especial** de costo mínimo que va de v a j.
- Inicialmente, por lo dicho en el párrafo anterior, $D[j]$ debe ser $L[v, j]$.
- Eso explica la inicialización de D que se realiza en el primer **for**.

Vértice especial y camino mínimo

- Cuando un vértice c es especial, ya se conoce el costo del camino de costo mínimo que va de v a c,
- y es el que está dado en ese momento por $D[c]$.
- En efecto, esto se cumple inicialmente: el vértice v es el único especial y el valor inicial de $D[v]$, es decir, 0, es el costo del camino de costo mínimo para ir desde v a v.

Invariante

Lo dicho puede expresarse en el siguiente invariante:

$\forall j \notin C.D[j] =$ costo del camino de costo mínimo de v a j
$\forall j \in C.D[j] =$ costo del camino **especial** de costo mínimo de v a j

- Para entender el algoritmo es importante prestar atención a la palabra **especial** (que equivale a **azul**).
- Cuando conocemos el costo del camino **especial** de costo mínimo no necesariamente hemos obtenido lo que buscamos,
- buscamos el costo del camino de costo mínimo, el mínimo de todos, especial o no.

Un nuevo vértice especial

- El algoritmo de Dijkstra elimina en cada ciclo un vértice c de C.
- Para que se mantenga el invariante es imprescindible saber que para ese c
- (que pertenecía a C y por lo tanto por el invariante $D[c]$ era el costo del camino **especial** de costo mínimo de v a c),
- $D[c]$ es en realidad el costo del camino (no necesariamente especial) de costo mínimo de v a c.

¿Cómo podemos asegurarnos de eso?

- El algoritmo elige $c \in C$ de modo de que $D[c]$ sea el mínimo.
- Es decir, elige un vértice c que aún **no es especial** y tal que $D[c]$ es mínimo.
- Sabemos, por el invariante, que $D[c]$ es el costo del camino **especial** de costo mínimo de v a c.
- ¿Puede haber un camino **no especial** de v a c que cueste menos?

¿Puede haber un camino **no especial** de v a c que cueste menos?

- Si lo hubiera, dicho camino necesariamente debería tener, por ser **no especial**, algún vértice intermedio **no especial**.
- Sea w el primer vértice **no especial** que ocurre en ese camino comenzando desde v.
- El camino **no especial** consta de una primera parte que llega a w.
- Esa primera parte es un camino **especial** de v a w, por lo que su costo, dice el invariante, debe ser $D[w]$.
- El costo del camino completo **no especial** de v a c que pasa por w costará al menos $D[w]$ ya que ése es apenas el costo de una parte del mismo.

¿Puede haber un camino **no especial** de v a c cueste menos?

- Dijimos que ese camino pasaría por w como primer vértice **no especial** y por ello costaría al menos $D[w]$.
- Sin embargo, como c fue elegido como el que minimiza (entre los vértices **no especiales**) $D[c]$, necesariamente debe cumplirse $D[c] \leq D[w]$.
- Esto demuestra que no puede haber un camino **no especial** de v a c que cueste menos que $D[c]$.
- Por ello, c puede sin peligro ser considerado un vértice **especial** ya que $D[c]$ contiene el costo del camino (especial o no) de costo mínimo de v a c.

Nuevos caminos especiales

- Inmediatamente después de agregar c entre los vértices **especiales**, es decir, inmediatamente después de eliminarlo de C,
- surgen nuevos caminos **especiales** ya que ahora se permite que los mismos pasen también por el nuevo vértice **especial** c.
- Eso obliga a actualizar $D[j]$ para los j **no especiales** de modo de que siga satisfaciendo el invariante.
- Ahora un camino **especial** a j puede pasar por c.
- Sólo hace falta considerar caminos **especiales** de v a j cuyo último vértice **especial** es c.

¿Por qué?

- Dijimos que sólo hace falta considerar caminos **especiales** de v a j cuyo último vértice **especial** es c.
- ¿Por qué?
- Los caminos **especiales** de v a j que pasan por c y cuyo último vértice **especial** es k no ganan nada por pasar por c
- ya que c está antes de k en esos caminos y entonces el costo del tramo hasta k, siendo k **especial**, sigue siendo como mínimo $D[k]$,
- es decir, en el mejor de los casos lo mismo que se tenía sin pasar por c.

Recalculando D

- Consideremos entonces solamente los caminos **especiales** a j que tienen a c como último vértice **especial**.
- El costo de un tal camino de costo mínimo está dado por $D[c] + L[c, j]$,
- la suma entre el costo del camino de costo mínimo para llegar hasta c ($D[c]$) más el costo de la arista que va de c a j ($L[c, j]$).
- Este costo debe compararse con el que ya se tenía, el que sólo contemplaba los caminos **especiales** antes de que c fuera **especial**.
- Ese valor es $D[j]$.
- El mínimo de los dos es el nuevo valor para $D[j]$.
- Eso explica el segundo **for**.

Últimas consideraciones

- Por último, puede observarse que en cada ejecución del ciclo un nuevo vértice se vuelve **especial**.
- Inicialmente v lo es.
- Por ello, al cabo de n-2 iteraciones, tenemos solamente 1 vértice **no especial**.
- Sea k ese vértice.

Postcondición

El invariante resulta

$$\forall j \neq k. D[j] = \text{costo del camino de costo mínimo de } v \text{ a } j$$
$$D[k] = \text{costo del camino } \textbf{especial} \text{ de costo mínimo de } v \text{ a } k$$

pero siendo k el único vértice **no especial** todos los caminos de v a k (que no tengan ciclos en los que k esté involucrado) son **especiales.** Por ello, se tiene

$$D[k] = \text{costo del camino de costo mínimo de } v \text{ a } k$$

y por consiguiente

$$\forall j. D[j] = \text{costo del camino de costo mínimo de } v \text{ a } j$$

Clase de hoy

Algoritmo de Dijkstra

```
fun Dijkstra(L: array[1..n,1..n] of costo, v: nat)
    ret D: array[1..n] of costo          ret E: array[1..n] of nat
    var c: nat
    C:= {1,2,...,n}-{v}
    for j:= 1 to n do D[j]:= L[v,j] od
    for j:= 1 to n do E[j]:= v od
    do n-2 times → c:= elemento de C que minimice D[c]
                   C:= C-{c}
                   for j in C do
                       if D[c]+L[c,j] < D[j] then D[j]:= D[c]+L[c,j]
                                                  E[j]:= c
                       fi
                   od
    od
end fun
```