Algoritmos y Estructuras de Datos II

Algoritmos voraces sobre grafos

Clase de hoy

1. Árboles generadores de costo mínimo
   - Algoritmo de Prim

2. Camino de costo mínimo
   - Algoritmo de Dijkstra

Árbol generador de costo mínimo

- Sea $G = (V, A)$ un grafo conexo no dirigido con un costo no negativo asociado a cada arista.
- Se dice que $T \subseteq A$ es un árbol generador (intuitivamente, un tendido) si el grafo (V, T) es conexo y no contiene ciclos.
- Su costo es la suma de los costos de sus aristas.
- Se busca T tal que su costo sea mínimo.

Árbol generador de costo mínimo

- Sea $G = (V, A)$ un grafo conexo no dirigido con un costo no negativo asociado a cada arista.
- Se dice que $T \subseteq A$ es un árbol generador (intuitivamente, un tendido) si el grafo (V, T) es conexo y no contiene ciclos.
- Su costo es la suma de los costos de sus aristas.
- Se busca T tal que su costo sea mínimo.

Árbol generador de costo mínimo

- Sea $G = (V, A)$ un grafo conexo no dirigido con un costo no negativo asociado a cada arista.
- Se dice que $T \subseteq A$ es un árbol generador (intuitivamente, un tendido) si el grafo (V, T) es conexo y no contiene ciclos.
- Su costo es la suma de los costos de sus aristas.
- Se busca T tal que su costo sea mínimo.

Árbol generador de costo mínimo

- Sea $G = (V, A)$ un grafo conexo no dirigido con un costo no negativo asociado a cada arista.
- Se dice que $T \subseteq A$ es un árbol generador (intuitivamente, un tendido) si el grafo (V, T) es conexo y no contiene ciclos.
- Su costo es la suma de los costos de sus aristas.
- Se busca T tal que su costo sea mínimo.

Árbol generador de costo mínimo

- El problema de encontrar un árbol generador de costo mínimo tiene numerosas aplicaciones en la vida real.
- Cada vez que se quiera realizar un tendido (eléctrico, telefónico, etc) se quieren unir distintas localidades de modo que requiera el menor costo en instalaciones (por ejemplo, cables) posible.
- Se trata de realizar el tendido siguiendo la traza de un árbol generador de costo mínimo.

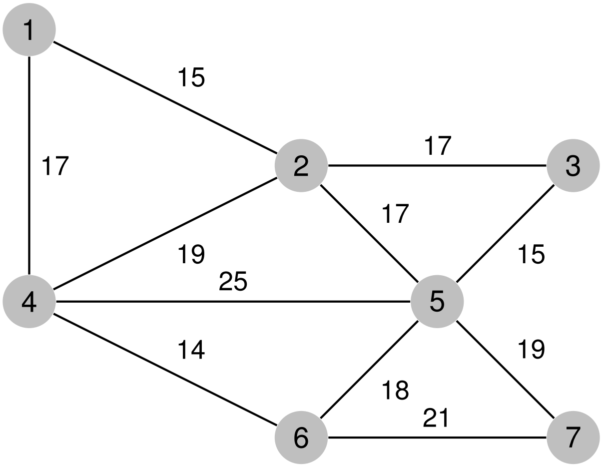
Árbol generador de costo mínimo

- El problema de encontrar un árbol generador de costo mínimo tiene numerosas aplicaciones en la vida real.
- Cada vez que se quiera realizar un tendido (eléctrico, telefónico, etc) se quieren unir distintas localidades de modo que requiera el menor costo en instalaciones (por ejemplo, cables) posible.
- Se trata de realizar el tendido siguiendo la traza de un árbol generador de costo mínimo.

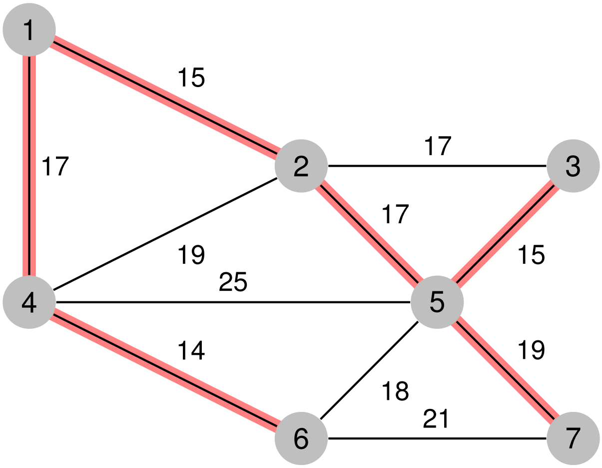
Árbol generador de costo mínimo

- El problema de encontrar un árbol generador de costo mínimo tiene numerosas aplicaciones en la vida real.
- Cada vez que se quiera realizar un tendido (eléctrico, telefónico, etc) se quieren unir distintas localidades de modo que requiera el menor costo en instalaciones (por ejemplo, cables) posible.
- Se trata de realizar el tendido siguiendo la traza de un árbol generador de costo mínimo.

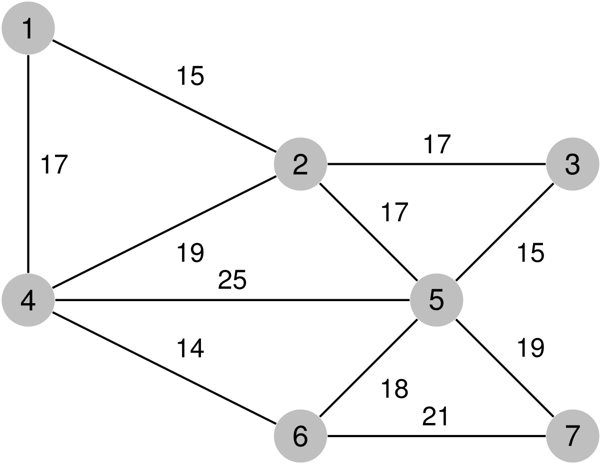
Ejemplo



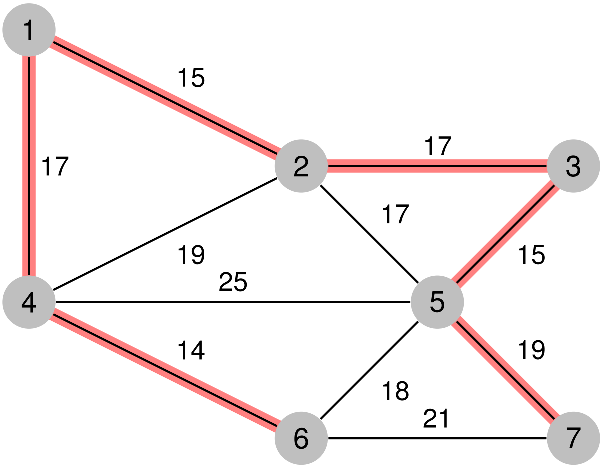
Ejemplo



Ejemplo



Ejemplo



Dos estrategias

Hay dos grandes ideas de cómo resolverlo:

- La de Prim: se parte desde un vértice origen y se va extendiendo el tendido a partir de ahí:
  - en cada paso se une el tendido ya existente con alguno de los vértices aún no alcanzados, seleccionando la arista de menor costo capaz de incorporar un nuevo vértice
- La de Kruskal: se divide el grafo en distintas componentes (originariamente una por cada vértice) y se van uniendo componentes,
  - en cada paso se selecciona la arista de menor costo capaz de unir componentes.

Este año no veremos en detalle el algoritmo de Kruskal.

Dos estrategias

Hay dos grandes ideas de cómo resolverlo:

- La de Prim: se parte desde un vértice origen y se va extendiendo el tendido a partir de ahí:
  - en cada paso se une el tendido ya existente con alguno de los vértices aún no alcanzados, seleccionando la arista de menor costo capaz de incorporar un nuevo vértice
- La de Kruskal: se divide el grafo en distintas componentes (originariamente una por cada vértice) y se van uniendo componentes,
  - en cada paso se selecciona la arista de menor costo capaz de unir componentes.

Este año no veremos en detalle el algoritmo de Kruskal.

Dos estrategias

Hay dos grandes ideas de cómo resolverlo:

- La de Prim: se parte desde un vértice origen y se va extendiendo el tendido a partir de ahí:
  - en cada paso se une el tendido ya existente con alguno de los vértices aún no alcanzados, seleccionando la arista de menor costo capaz de incorporar un nuevo vértice
- La de Kruskal: se divide el grafo en distintas componentes (originariamente una por cada vértice) y se van uniendo componentes,
  - en cada paso se selecciona la arista de menor costo capaz de unir componentes.

Este año no veremos en detalle el algoritmo de Kruskal.

Dos estrategias

Hay dos grandes ideas de cómo resolverlo:

- La de Prim: se parte desde un vértice origen y se va extendiendo el tendido a partir de ahí:
  - en cada paso se une el tendido ya existente con alguno de los vértices aún no alcanzados, seleccionando la arista de menor costo capaz de incorporar un nuevo vértice
- La de Kruskal: se divide el grafo en distintas componentes (originariamente una por cada vértice) y se van uniendo componentes,
  - en cada paso se selecciona la arista de menor costo capaz de unir componentes.

Este año no veremos en detalle el algoritmo de Kruskal.

Dos estrategias

Hay dos grandes ideas de cómo resolverlo:

- La de Prim: se parte desde un vértice origen y se va extendiendo el tendido a partir de ahí:
  - en cada paso se une el tendido ya existente con alguno de los vértices aún no alcanzados, seleccionando la arista de menor costo capaz de incorporar un nuevo vértice
- La de Kruskal: se divide el grafo en distintas componentes (originariamente una por cada vértice) y se van uniendo componentes,
  - en cada paso se selecciona la arista de menor costo capaz de unir componentes.

Este año no veremos en detalle el algoritmo de Kruskal.

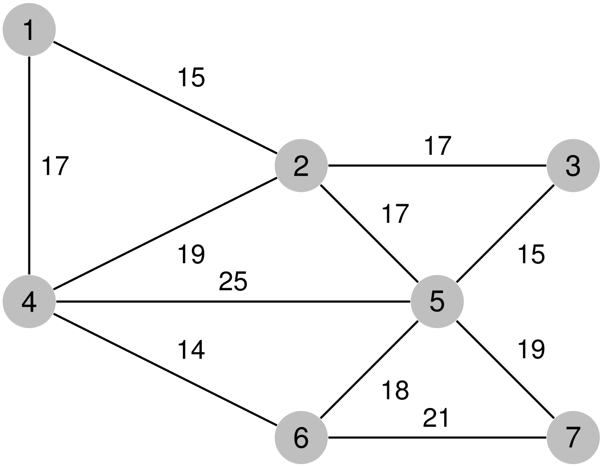
Dos estrategias

Hay dos grandes ideas de cómo resolverlo:

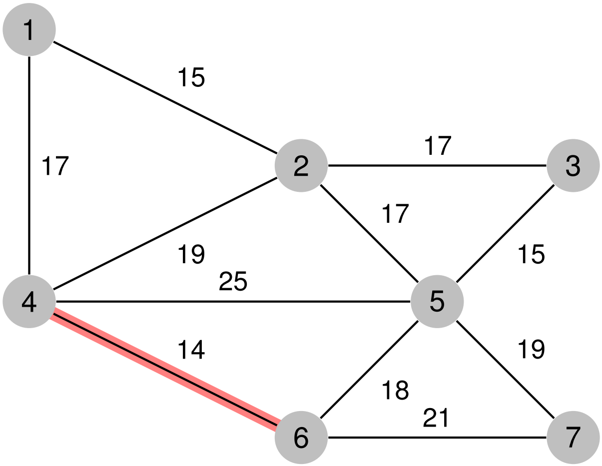
- La de Prim: se parte desde un vértice origen y se va extendiendo el tendido a partir de ahí:
  - en cada paso se une el tendido ya existente con alguno de los vértices aún no alcanzados, seleccionando la arista de menor costo capaz de incorporar un nuevo vértice
- La de Kruskal: se divide el grafo en distintas componentes (originariamente una por cada vértice) y se van uniendo componentes,
  - en cada paso se selecciona la arista de menor costo capaz de unir componentes.

Este año no veremos en detalle el algoritmo de Kruskal.

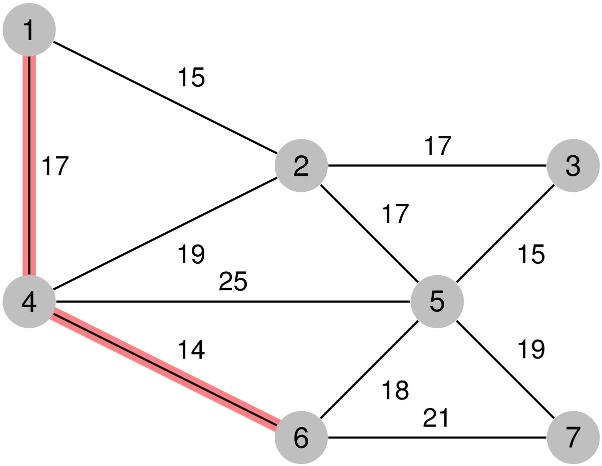
Algoritmo de Prim



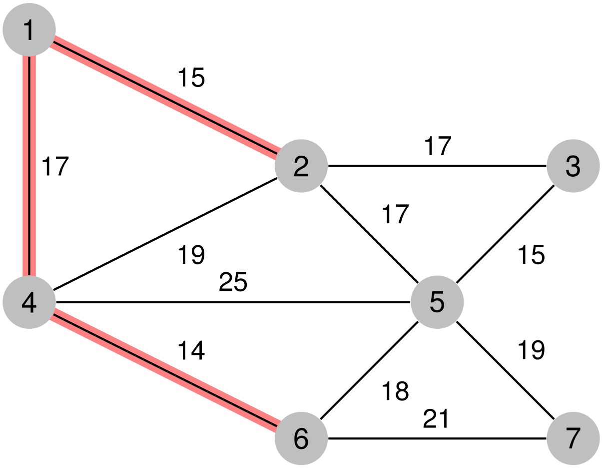
Algoritmo de Prim



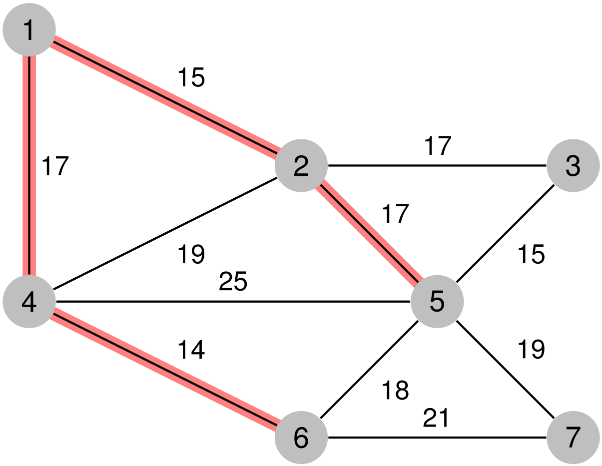
Algoritmo de Prim



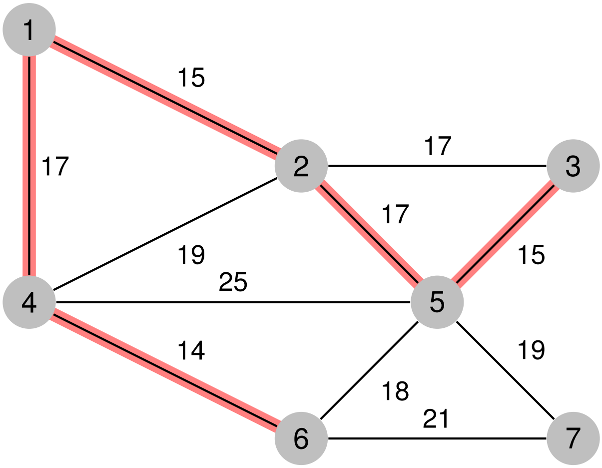
Algoritmo de Prim



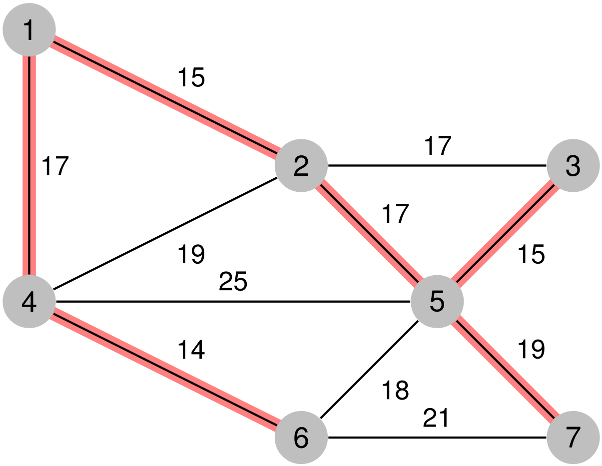
Algoritmo de Prim



Algoritmo de Prim



Algoritmo de Prim



Implementación del Algoritmo de Prim

Podemos representar los grafos como una tupla con dos conjuntos: uno para los vértices y otro para las aristas.

```
type Vertex = Nat

type Edge = tuple
                v1 : Vertex
                v2 : Vertex
                cost : Nat
            end tuple

type Graph = tuple
                vertices : Set of Vertex
                edges : Set of Edge
             end tuple
```

Implementación del Algoritmo de Prim

```
fun Prim(G : Graph, k: Vertex) ret T: Set of Edge
    var c: Edge
    var C: Set of Vertex
    C:= copy_set(G.vertices)
    elim(C,k)
    T:= empty_set()
    do (not is_empty_set(C)) →
            c := "selecciono arista de costo mínimo tal que
                c.v1 ∈ C y c.v2 ∉ C, ó c.v2 ∈ C y c.v1 ∉ C"
            if member(c.v1,C)
              then elim(C,c.v1)
              else elim(C,c.v2)
            add(T,c)
    fi
    od
end fun
```

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.

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.

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 costos de los caminos de menor costo a cada uno de los vértices.

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 costos de los caminos de menor costo a cada uno de los vértices.

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 costos de los caminos de menor costo a cada uno de los vértices.

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 costos de 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
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

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.

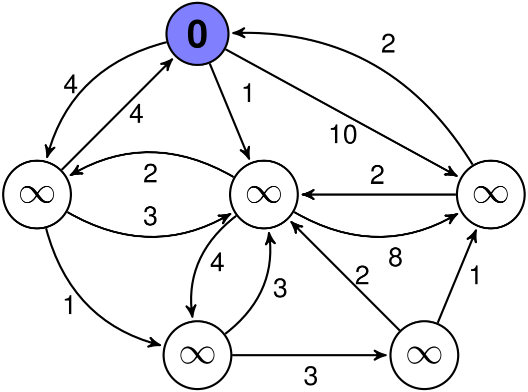
Algoritmo de Dijkstra

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.

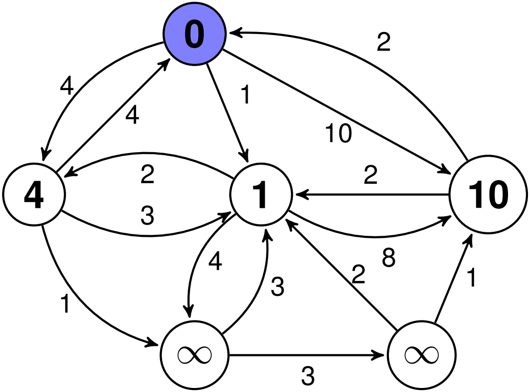
Algoritmo de Dijkstra

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



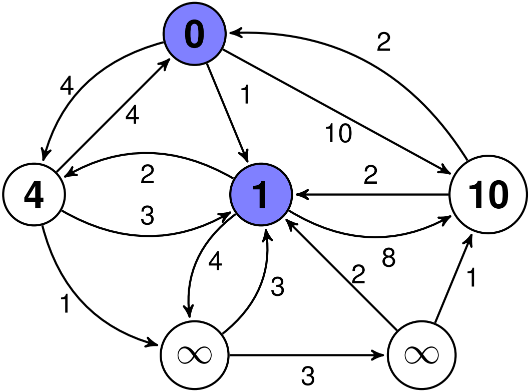
Algoritmo de Dijkstra

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



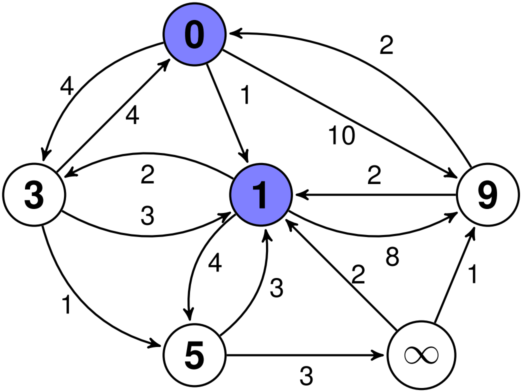
Algoritmo de Dijkstra

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



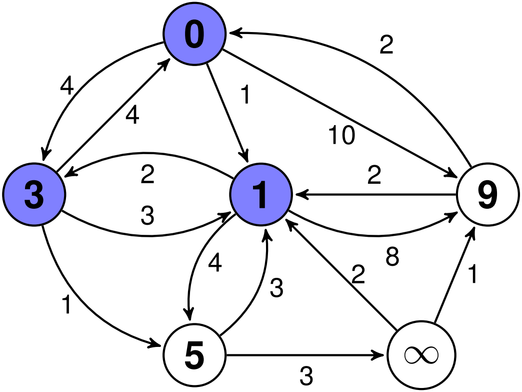
Algoritmo de Dijkstra

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



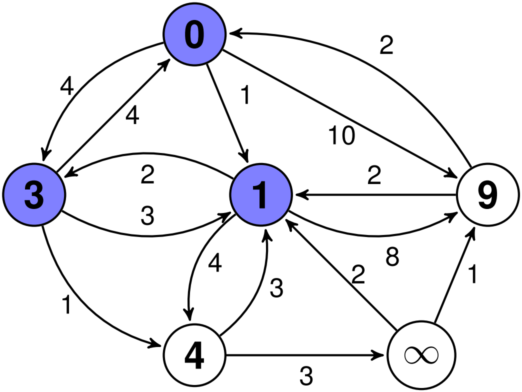
Algoritmo de Dijkstra

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



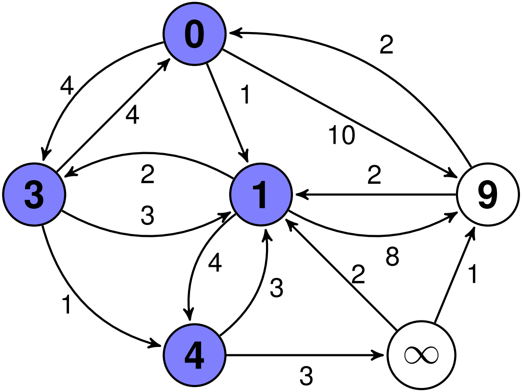
Algoritmo de Dijkstra

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



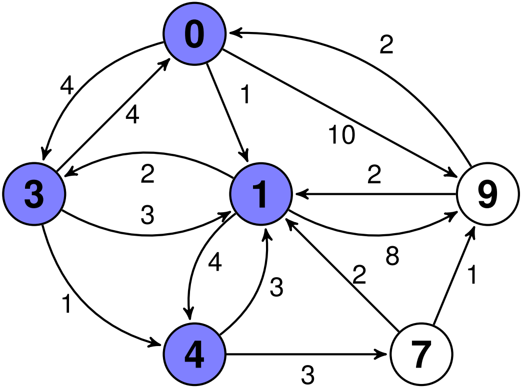
Algoritmo de Dijkstra

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



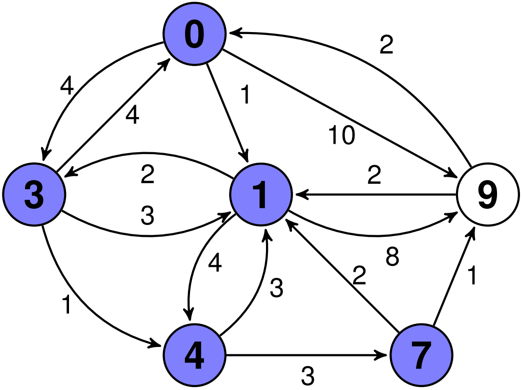
Algoritmo de Dijkstra

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



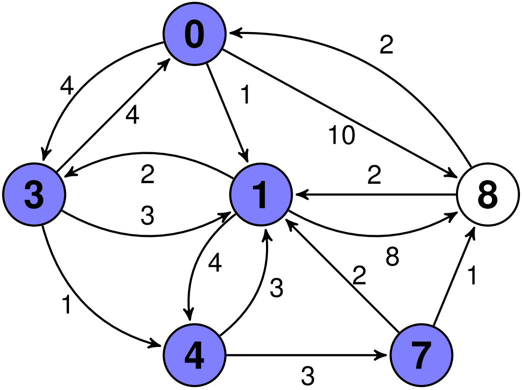
Algoritmo de Dijkstra

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



El algoritmo

- Asumiremos que el grafo viene dado por el conjunto de vértices $V = \{1, 2, \ldots, n\}$
- y los costos por una matriz L : **array**[1..n,1..n] **of** Nat,
- 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

- Asumiremos que el grafo viene dado por el conjunto de vértices $V = \{1, 2, \ldots, n\}$
- y los costos por una matriz L : **array**[1..n,1..n] **of** Nat,
- 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

- Asumiremos que el grafo viene dado por el conjunto de vértices $V = \{1, 2, \ldots, n\}$
- y los costos por una matriz L : **array**[1..n,1..n] **of** Nat,
- 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

- Asumiremos que el grafo viene dado por el conjunto de vértices $V = \{1, 2, \ldots, n\}$
- y los costos por una matriz L : **array**[1..n,1..n] **of** Nat,
- 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

- Asumiremos que el grafo viene dado por el conjunto de vértices $V = \{1, 2, \ldots, n\}$
- y los costos por una matriz L : **array**[1..n,1..n] **of** Nat,
- 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

- Asumiremos que el grafo viene dado por el conjunto de vértices $V = \{1, 2, \ldots, n\}$
- y los costos por una matriz L : **array**[1..n,1..n] **of** Nat,
- 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

- La versión que daremos del algoritmo, 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** Nat,
- 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.

El algoritmo

- La versión que daremos del algoritmo, 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** Nat,
- 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.

El algoritmo

- La versión que daremos del algoritmo, 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** Nat,
- 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.

El algoritmo

- La versión que daremos del algoritmo, 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** Nat,
- 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.

El algoritmo

- La versión que daremos del algoritmo, 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** Nat,
- 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 Nat, v: Nat)
                                             ret D: array[1..n] of Nat
    var c: Nat
    var C: Set of Nat
    for i := 1 to n do add(C,i) od
    elim(C,v)
    for j:= 1 to n do D[j]:= L[v,j] od
    do (not is_empty_set(C))→
            c:= “elijo elemento c de C tal que D[c] sea mínimo”
            elim(C,c)
            for j in C do D[j]:= min(D[j],D[c]+L[c,j]) od
    od
end fun
```

Algoritmo de Dijkstra

- La implementación sigue el esquema de los algoritmos voraces.
- En cada paso, elijo el vértice c al que puedo llegar con menor costo.
- Luego actualizo el costo para llegar a cada uno de los demás vértices, habiendo “aprendido” el mejor camino para ir hasta c.
- Esta actualización se realiza calculando el mínimo entre lo que me costaba antes ir hasta cada vértice j, y lo que me cuesta si voy primero a c, y luego de ahí hasta j.

Modificando levemente la implementación dada, se puede obtener una versión que devuelve el camino hacia cada vértice, no solo su costo. No la veremos en detalle este año.

Algoritmo de Dijkstra

- La implementación sigue el esquema de los algoritmos voraces.
- En cada paso, elijo el vértice c al que puedo llegar con menor costo.
- Luego actualizo el costo para llegar a cada uno de los demás vértices, habiendo "aprendido" el mejor camino para ir hasta c.
- Esta actualización se realiza calculando el mínimo entre lo que me costaba antes ir hasta cada vértice j, y lo que me cuesta si voy primero a c, y luego de ahí hasta j.

Modificando levemente la implementación dada, se puede obtener una versión que devuelve el camino hacia cada vértice, no solo su costo. No la veremos en detalle este año.

Algoritmo de Dijkstra

- La implementación sigue el esquema de los algoritmos voraces.
- En cada paso, elijo el vértice c al que puedo llegar con menor costo.
- Luego actualizo el costo para llegar a cada uno de los demás vértices, habiendo "aprendido" el mejor camino para ir hasta c.
- Esta actualización se realiza calculando el mínimo entre lo que me costaba antes ir hasta cada vértice j, y lo que me cuesta si voy primero a c, y luego de ahí hasta j.

Modificando levemente la implementación dada, se puede obtener una versión que devuelve el camino hacia cada vértice, no solo su costo. No la veremos en detalle este año.

Algoritmo de Dijkstra

- La implementación sigue el esquema de los algoritmos voraces.
- En cada paso, elijo el vértice c al que puedo llegar con menor costo.
- Luego actualizo el costo para llegar a cada uno de los demás vértices, habiendo “aprendido” el mejor camino para ir hasta c.
- Esta actualización se realiza calculando el mínimo entre lo que me costaba antes ir hasta cada vértice j, y lo que me cuesta si voy primero a c, y luego de ahí hasta j.

Modificando levemente la implementación dada, se puede obtener una versión que devuelve el camino hacia cada vértice, no solo su costo. No la veremos en detalle este año.

Algoritmo de Dijkstra

- La implementación sigue el esquema de los algoritmos voraces.
- En cada paso, elijo el vértice c al que puedo llegar con menor costo.
- Luego actualizo el costo para llegar a cada uno de los demás vértices, habiendo "aprendido" el mejor camino para ir hasta c.
- Esta actualización se realiza calculando el mínimo entre lo que me costaba antes ir hasta cada vértice j, y lo que me cuesta si voy primero a c, y luego de ahí hasta j.

Modificando levemente la implementación dada, se puede obtener una versión que devuelve el camino hacia cada vértice, no solo su costo. No la veremos en detalle este año.

Algoritmo de Dijkstra

- La implementación sigue el esquema de los algoritmos voraces.
- En cada paso, elijo el vértice c al que puedo llegar con menor costo.
- Luego actualizo el costo para llegar a cada uno de los demás vértices, habiendo "aprendido" el mejor camino para ir hasta c.
- Esta actualización se realiza calculando el mínimo entre lo que me costaba antes ir hasta cada vértice j, y lo que me cuesta si voy primero a c, y luego de ahí hasta j.

Modificando levemente la implementación dada, se puede obtener una versión que devuelve el camino hacia cada vértice, no solo su costo. No la veremos en detalle este año.