# APSP – All-Pairs Shortest Path

Santiago Cifuentes - Ezequiel Companeetz - Dafne Yudcovsky

Departamento de Computación
Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires

$2^{do}$ Cuatrimestre de 2024

# Programa de hoy

1. Algoritmos APSP
   - Floyd-Warshall
   - Dantzig
   - ¿Más?
2. Ejercicios
   - Optimizando canciones
   - 200 por hora
   - String Problem
   - Homero

# APSP (All-Pairs Shortest Paths)

## Objetivo

En esta clase veremos algoritmos cuyo objetivo es encontrar la longitud de todos los caminos mínimos.

# APSP (All-Pairs Shortest Paths)

## Objetivo

En esta clase veremos algoritmos cuyo objetivo es encontrar la longitud de todos los caminos mínimos.

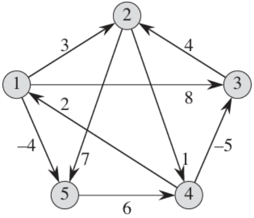
¿Cuál es la complejidad mínima que deberíamos esperar?

# APSP (All-Pairs Shortest Paths)

## Objetivo

En esta clase veremos algoritmos cuyo objetivo es encontrar la longitud de todos los caminos mínimos.

¿Cuál es la complejidad mínima que deberíamos esperar? $O(n^2)$, ya que es el tamaño del output.




$$\begin{pmatrix} 0 & 1 & -3 & 2 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix}$$

# Floyd-Warshall (APSP)

## Intuición

Sea $G$ un grafo con vértices numerados $\{1, \ldots, n\}$.

# Floyd-Warshall (APSP)

## Intuición

Sea $G$ un grafo con vértices numerados $\{1, \dots, n\}$.

- Sea $C^k_{i,j}$ el conjunto de todos los caminos entre $i$ y $j$ que solo usan nodos en el conjunto $\{1, \dots, k\}$, y sea $p$ el camino mínimo de $C^k_{i,j}$.

# Floyd-Warshall (APSP)

## Intuición

Sea $G$ un grafo con vértices numerados $\{1, \dots, n\}$.

- Sea $C^k_{i,j}$ el conjunto de todos los caminos entre $i$ y $j$ que solo usan nodos en el conjunto $\{1, \dots, k\}$, y sea $p$ el camino mínimo de $C^k_{i,j}$.
- ¿Qué relación hay entre $C^{k-1}_{i,j}$ y $p$? ¿Puede $p$ también ser el camino mínimo de $C^{k-1}_{i,j}$?

# Floyd-Warshall (APSP)

## Intuición

Sea $G$ un grafo con vértices numerados $\{1, \dots, n\}$.

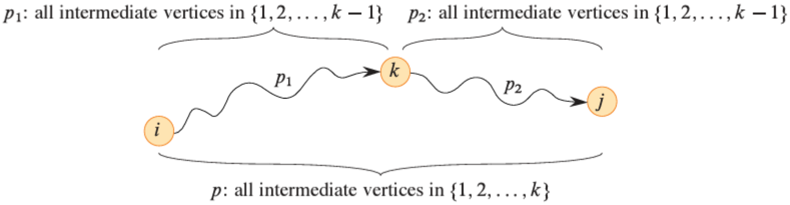
- Sea $C^k_{i,j}$ el conjunto de todos los caminos entre $i$ y $j$ que solo usan nodos en el conjunto $\{1, \dots, k\}$, y sea $p$ el camino mínimo de $C^k_{i,j}$.
- ¿Qué relación hay entre $C^{k-1}_{i,j}$ y $p$? ¿Puede $p$ también ser el camino mínimo de $C^{k-1}_{i,j}$?
- Hay dos opciones: $k \notin p$ o $k \in p$

# Floyd-Warshall

- Si $k \notin p$ entonces $p$ también es un camino mínimo en $C_{i,j}^{k-1}$.

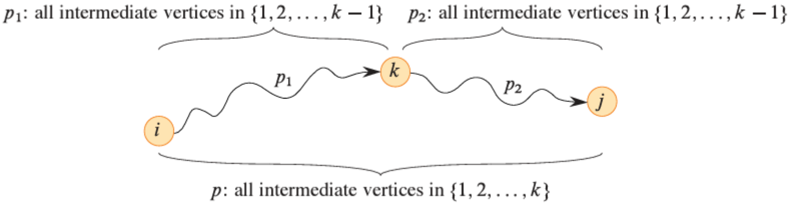
# Floyd-Warshall

- Si $k \notin p$ entonces $p$ también es un camino mínimo en $C_{i,j}^{k-1}$.
- Caso contrario, podemos descomponer a $p$



# Floyd-Warshall

- Si $k \notin p$ entonces $p$ también es un camino mínimo en $C^{k-1}_{i,j}$.
- Caso contrario, podemos descomponer a $p$




La primer porción es un camino mínimo de $i$ a $k$ que usa nodos en el conjunto $\{1, \ldots, k-1\}$, mientras que la segunda es un camino mínimo entre $k$ y $j$ con nodos en el conjunto $\{1, \ldots, k-1\}$.

# Floyd-Warshall

## Formulación

Sea $d_{ij}^{(k)}$ el peso del camino mínimo de $i$ a $j$ usando como vertices intermedios $\{1, 2, \ldots, k\}$

$$d_{ij}^{(k)} = \begin{cases} w_{ij} & \text{if } k = 0 \,, \\ \min\left(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\right) & \text{if } k \geq 1 \,. \end{cases}$$

# Floyd-Warshall

## Formulación

Sea $d_{ij}^{(k)}$ el peso del camino mínimo de $i$ a $j$ usando como vertices intermedios $\{1, 2, \ldots, k\}$

$$d_{ij}^{(k)} = \begin{cases} w_{ij} & \text{if } k = 0 \, , \\ \min\left(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\right) & \text{if } k \geq 1 \, . \end{cases}$$

## Invariante

Típicamente Floyd-Warshall se implementa bottom-up, manteniendo en cada iteración $k$ la matriz $d^k$. Es decir, en cada paso $k$ se tiene acceso a los caminos mínimos que solo usan nodos dentro del conjunto $\{1, \ldots, k\}$.

# Algoritmo

**Algorithm 1:** Floyd-Warshall

**Input:** $G = (V, E)$
**Output:** $d$
**for** $i, j = 1$ ***to*** $n$ ´ **do**
  $d[i][j][0] := w_{ij}$ ;
**for** $k \leftarrow 1$ ***to*** $n$ **do**
  **for** $i \leftarrow 1$ ***to*** $n$ **do**
    **for** $j \leftarrow 1$ ***to*** $n$ **do**
      **if** $d[i][j][k-1] > d[i][k][k-1] + d[k][j][k-1]$ **then**
        $d[i][j][k] := d[i][k][k-1] + d[k][j][k-1]$ ;
        $next[i][j] := k$ ;

¿Que valor guarda next?

# Análisis

- ¿Complejidad temporal?

# Análisis

- ¿Complejidad temporal? $O(n^3)$.

# Análisis

- ¿Complejidad temporal?$O(n^3)$.
- ¿Complejidad espacial?

# Análisis

- ¿Complejidad temporal?$O(n^3)$.
- ¿Complejidad espacial? $O(n^3)$

# Análisis

- ¿Complejidad temporal?$O(n^3)$.
- ¿Complejidad espacial? $O(n^3)$ ¿Se puede mejorar?

# Análisis

- ¿Complejidad temporal?$O(n^3)$.
- ¿Complejidad espacial? $O(n^3)$ ¿Se puede mejorar? Si, a $O(n^2)$.

# Análisis

- ¿Complejidad temporal?$O(n^3)$.
- ¿Complejidad espacial? $O(n^3)$ ¿Se puede mejorar? Si, a $O(n^2)$.
- ¿Qué pasa si el grafo tiene aristas de peso negativo?

# Análisis

- ¿Complejidad temporal?$O(n^3)$.
- ¿Complejidad espacial? $O(n^3)$ ¿Se puede mejorar? Si, a $O(n^2)$.
- ¿Qué pasa si el grafo tiene aristas de peso negativo? Nada, en ningún momento del razonamiento nos importó.

# Análisis

- ¿Complejidad temporal?$O(n^3)$.
- ¿Complejidad espacial? $O(n^3)$ ¿Se puede mejorar? Si, a $O(n^2)$.
- ¿Qué pasa si el grafo tiene aristas de peso negativo? Nada, en ningún momento del razonamiento nos importó.
- ¿Sirve para detectar ciclos negativos?

# Análisis

- ¿Complejidad temporal?$O(n^3)$.
- ¿Complejidad espacial? $O(n^3)$ ¿Se puede mejorar? Si, a $O(n^2)$.
- ¿Qué pasa si el grafo tiene aristas de peso negativo? Nada, en ningún momento del razonamiento nos importó.
- ¿Sirve para detectar ciclos negativos? Si, ver la diagonal.

**Lema**

Si $G$ tiene un ciclo de longitud negativa entonces $d^n_{i,i} < 0$ para algún $i$.

# Dantzig

## Estrategia

Llamemos $G_k$ al subgrafo de $G$ inducido por los nodos $\{1, \ldots, k\}$, y sea $D^k$ la matriz de distancias de $G_k$ ¿Cómo se relacionan $D^k$ y $D^{k+1}$?

# Dantzig

- Distancia de $i \in \{1, \ldots, k\}$ a $k + 1$:

# Dantzig

- Distancia de $i \in \{1, \ldots, k\}$ a $k+1$: hay que ir hasta un $j$ y luego tomar el eje $j \to k+1$. Es decir, $D^{k+1}_{i,k+1} = \min_{1 \leq j \leq k}(D^k_{i,j} + w_{j,k+1})$.

# Dantzig

- Distancia de $i \in \{1, \ldots, k\}$ a $k+1$: hay que ir hasta un $j$ y luego tomar el eje $j \to k+1$. Es decir, $D^{k+1}_{i,k+1} = \min_{1 \leq j \leq k}(D^k_{i,j} + w_{j,k+1})$.
- Distancia de $k+1$ a $i \in \{1, \ldots, k\}$:

# Dantzig

- Distancia de $i \in \{1, \ldots, k\}$ a $k+1$: hay que ir hasta un $j$ y luego tomar el eje $j \to k+1$. Es decir, $D^{k+1}_{i,k+1} = \min_{1 \leq j \leq k}(D^{k}_{i,j} + w_{j,k+1})$.
- Distancia de $k+1$ a $i \in \{1, \ldots, k\}$: análogo.

# Dantzig

- Distancia de $i \in \{1, \ldots, k\}$ a $k+1$: hay que ir hasta un $j$ y luego tomar el eje $j \to k+1$. Es decir, $D^{k+1}_{i,k+1} = \text{mín}_{1 \leq j \leq k}(D^k_{i,j} + w_{j,k+1})$.
- Distancia de $k+1$ a $i \in \{1, \ldots, k\}$: análogo.
- Distancia de $i$ a $j$, ambos menores a $k+1$:

# Dantzig

- Distancia de $i \in \{1, \ldots, k\}$ a $k+1$: hay que ir hasta un $j$ y luego tomar el eje $j \to k+1$. Es decir, $D^{k+1}_{i,k+1} = \text{mín}_{1 \leq j \leq k}(D^k_{i,j} + w_{j,k+1})$.
- Distancia de $k+1$ a $i \in \{1, \ldots, k\}$: análogo.
- Distancia de $i$ a $j$, ambos menores a $k+1$: uso el camino viejo que no pasaba por $k+1$, o bien paso por $k+1$ combinando dos caminos óptimos. Es decir, $D^{k+1}_{i,j} = \text{mín}(D^k_{i,j}, D^{k+1}_{i,k+1} + D^{k+1}_{k+1,j})$.

# Dantzig

## Algoritmo de Dantzig (1966)

**para** $k$ **desde** 1 **a** $n-1$ **hacer**
  **para** $i$ **desde** 1 **a** $k$ **hacer**
    $L_{i,k+1} := \text{mín}_{1 \leq j \leq k}(L_{i,j} + L_{j,k+1})$
    $L_{k+1,i} := \text{mín}_{1 \leq j \leq k}(L_{k+1,j} + L_{j,i})$
  **fin para**
  $t := \text{mín}_{1 \leq i \leq k}(L_{k+1,i} + L_{i,k+1})$
  **si** $t < 0$ **entonces**
    **retornar** "Hay ciclos de longitud negativa"
  **fin si**
  **para** $i$ **desde** 1 **a** $k$ **hacer**
    **para** $j$ **desde** 1 **a** $k$ **hacer**
      $L_{i,j} := \text{mín}(L_{i,j}, L_{i,k+1} + L_{k+1,j})$
    **fin para**
  **fin para**
**fin para**
**retornar** $L$

# Análisis

- ¿Invariante?

# Análisis

- ¿Invariante? Al finalizar la iteración $k$ tenemos las distancias del grafo inducido por los primeros $k$ nodos.

# Análisis

- ¿Invariante? Al finalizar la iteración $k$ tenemos las distancias del grafo inducido por los primeros $k$ nodos.
- ¿Complejidad temporal?

# Análisis

- ¿Invariante? Al finalizar la iteración $k$ tenemos las distancias del grafo inducido por los primeros $k$ nodos.
- ¿Complejidad temporal? $O(n^3)$.

# Análisis

- ¿Invariante? Al finalizar la iteración $k$ tenemos las distancias del grafo inducido por los primeros $k$ nodos.
- ¿Complejidad temporal? $O(n^3)$.
- ¿Complejidad espacial?

# Análisis

- ¿Invariante? Al finalizar la iteración $k$ tenemos las distancias del grafo inducido por los primeros $k$ nodos.
- ¿Complejidad temporal? $O(n^3)$.
- ¿Complejidad espacial? $O(n^3)$ a lo bestia, se puede hacer en $O(n^2)$.

# Análisis

- ¿Invariante? Al finalizar la iteración $k$ tenemos las distancias del grafo inducido por los primeros $k$ nodos.
- ¿Complejidad temporal? $O(n^3)$.
- ¿Complejidad espacial? $O(n^3)$ a lo bestia, se puede hacer en $O(n^2)$.
- ¿Detecta ciclos de longitud negativa?

# Análisis

- ¿Invariante? Al finalizar la iteración $k$ tenemos las distancias del grafo inducido por los primeros $k$ nodos.
- ¿Complejidad temporal? $O(n^3)$.
- ¿Complejidad espacial? $O(n^3)$ a lo bestia, se puede hacer en $O(n^2)$.
- ¿Detecta ciclos de longitud negativa? Sip.

# Un momento... (Dato)

Ambos algoritmos que vimos tienen complejidad $O(n^3)$ y devuelven los caminos todos a todos ¿No podríamos haber usado algún algoritmo uno a todos *n* veces?

# Un momento... (Dato)

Ambos algoritmos que vimos tienen complejidad $O(n^3)$ y devuelven los caminos todos a todos ¿No podríamos haber usado algún algoritmo uno a todos $n$ veces?

- Usando Dijkstra:

# Un momento... (Dato)

Ambos algoritmos que vimos tienen complejidad $O(n^3)$ y devuelven los caminos todos a todos ¿No podríamos haber usado algún algoritmo uno a todos $n$ veces?

- Usando Dijkstra: $O(n \times m \log n) = O(nm \log n)$.

# Un momento... (Dato)

Ambos algoritmos que vimos tienen complejidad $O(n^3)$ y devuelven los caminos todos a todos ¿No podríamos haber usado algún algoritmo uno a todos $n$ veces?

- Usando Dijkstra: $O(n \times m \log n) = O(nm \log n)$.
- Usando Bellman-Ford:

# Un momento... (Dato)

Ambos algoritmos que vimos tienen complejidad $O(n^3)$ y devuelven los caminos todos a todos ¿No podríamos haber usado algún algoritmo uno a todos $n$ veces?

- Usando Dijkstra: $O(n \times m \log n) = O(nm \log n)$.
- Usando Bellman-Ford: $O(n \times nm) = O(n^2 m)$.

# Un momento... (Dato)

Ambos algoritmos que vimos tienen complejidad $O(n^3)$ y devuelven los caminos todos a todos ¿No podríamos haber usado algún algoritmo uno a todos $n$ veces?

- Usando Dijkstra: $O(n \times m \log n) = O(nm \log n)$.
- Usando Bellman-Ford: $O(n \times nm) = O(n^2 m)$.

El segundo siempre es malo, pero el primero le gana a $O(n^3)$ en grafos ralos. Sin embargo, no siempre se puede usar...

# Algoritmo de Johnson

## Intuición

Si tenemos un grafo con pesos negativos podemos modificar el costo en los nodos de tal forma que los pesos ahora sean positivos, pero los caminos mínimos sigan siendo los mismos. Luego, podemos usar Dijkstra.

# Algoritmo de Johnson

**Intuición**

Si tenemos un grafo con pesos negativos podemos modificar el costo en los nodos de tal forma que los pesos ahora sean positivos, pero los caminos mínimos sigan siendo los mismos. Luego, podemos usar Dijkstra.

¿Cuánto cuesta encontrar el valor por el cual modificar los nodos?

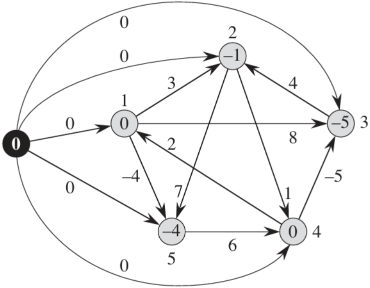
# Algoritmo de Johnson

## Intuición

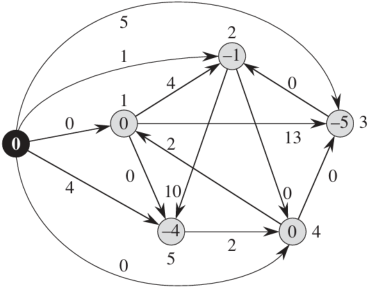
Si tenemos un grafo con pesos negativos podemos modificar el costo en los nodos de tal forma que los pesos ahora sean positivos, pero los caminos mínimos sigan siendo los mismos. Luego, podemos usar Dijkstra.

¿Cuánto cuesta encontrar el valor por el cual modificar los nodos? $O(nm)$ (es literalmente usar Bellman-Ford). La complejidad final es $O(nm + nm \log n) = O(nm \log n)$

# Algoritmo de Johnson




(a)




(b)

# Último dato

¿Se puede calcular APSP en $O(n^{3-\varepsilon})$?

# Último dato

¿Se puede calcular APSP en $O(n^{3-\varepsilon})$?

Subcubic Equivalences Between Path, Matrix, and Triangle Problems*

Virginia Vassilevska Williams† Ryan Williams‡

# Optimizando Canciones

## Enunciado

Rasta, que venció el bajón luego de comer tantos alfajores, está aprendiendo a tocar la guitarra. Como le gusta irse de campamento con amigos mochileros, se está esforzando en mejorar su velocidad para tocar acordes para los fogones.

Existen k acordes en total, y la canción con la que está practicando tiene una sucesión de n acordes. A Rasta le toman $S_{ij}$ segundos pasar del acorde i al acorde j, pero a veces le es más rápido pasar por acordes intermedios.

Se asume que Rasta empieza con la mano acomodada en el primer acorde. ¿Cuál es el mínimo tiempo que le toma tocar la canción?

Prueben codearlo y ponerlo acá:
`https://codeforces.com/gym/102646/problem/C`

# Optimizando Canciones

- Por ejemplo, si la canción va del acorde 6 al 7, capaz Rasta hace más rápido yendo del 6 al 3, y luego del 3 al 7.

# Optimizando Canciones

- Por ejemplo, si la canción va del acorde 6 al 7, capaz Rasta hace más rápido yendo del 6 al 3, y luego del 3 al 7.
- Definamos la canción C = $c_1, c_2, ..., c_n$
- Dado que Rasta puede hacer más rápido pasando por acordes intermedios, es posible que pueda tocar la canción C en menos de $\sum_{i=1}^{n-1} S_{c_i c_{i+1}}$ tiempo.

# Optimizando Canciones

- Podemos imaginar un grafo G = (V, E) de k vértices, con una arista i $\rightarrow$ j de peso $S_{ij}$.

# Optimizando Canciones

- Podemos imaginar un grafo G = (V, E) de k vértices, con una arista i $\rightarrow$ j de peso $S_{ij}$.
- Buscamos saber, para todo par de acordes i, j, el mínimo tiempo que necesita Rasta para pasar de i a j.
- Esto es nada menos que la matriz de distancias de G.

# Optimizando Canciones

- Ahora tenemos que usar algún algoritmo para computar distancias todos con todos.

# Optimizando Canciones

- Ahora tenemos que usar algún algoritmo para computar distancias todos con todos.
- ¿Floyd-Warshall o Johnson?

# Optimizando Canciones

- Ahora tenemos que usar algún algoritmo para computar distancias todos con todos.
- ¿Floyd-Warshall o Johnson?
- Para decidir cuál, ¿Nuestro grafo es denso o ralo?

# Optimizando Canciones

- Ahora tenemos que usar algún algoritmo para computar distancias todos con todos.
- ¿Floyd-Warshall o Johnson?
- Para decidir cuál, ¿Nuestro grafo es denso o ralo? como hay un $S_{ij}$ $\forall i \neq j$, es completo, así que $E \in \theta(V^2)$.
- Usando FW, el valor buscado es $t = \sum_{i=1}^{n-1} \delta(c_i, c_{i+1})$.

# Optimizando Canciones

- Ahora tenemos que usar algún algoritmo para computar distancias todos con todos.
- ¿Floyd-Warshall o Johnson?
- Para decidir cuál, ¿Nuestro grafo es denso o ralo? como hay un $S_{ij}$ $\forall i \neq j$, es completo, así que $E \in \theta(V^2)$.
- Usando FW, el valor buscado es $t = \sum_{i=1}^{n-1} \delta(c_i, c_{i+1})$.
- Complejidad: $O(k^3 + n)$, donde $k^3$ cuesta calcular la matriz con DP y n para recorrer la canción y sumar.

# Optimizando Canciones - Modificado

## Enunciado

Después de un rato de practicar, Rasta se dio cuenta que la canción todavía no estaba muy fluída y decidió pedir ayuda a sus amigos.

Tuki y Sasha, que también iban al campamento, le sugirieron que podría cambiar un acorde para disminuir el tiempo que le toma tocar la canción. ¿Qué cambio podría minimizar el tiempo total?

# Optimizando Canciones - Modificado

- Una manera de responder esta pregunta es yendo acorde por acorde y probando cambiarlo por cada uno de los otros k-1.

# Optimizando Canciones - Modificado

- Una manera de responder esta pregunta es yendo acorde por acorde y probando cambiarlo por cada uno de los otros k-1.
- En particular, si cambió el i-ésimo acorde por e, tengo que $t_{e \to i} = t - d(c_{i-1}, c_i) - d(c_i, c_{i+1}) + d(c_{i-1}, e) + d(e, ci + 1)$
- Salvedad: hay que definir que d = 0 si $c_{i-1} = 0$ o $c_{i+1} = n + 1$

# Optimizando Canciones - Modificado

- Podemos aprovechar una propiedad de la matriz de distancias: La desigualdad triangular. Dada una métrica (la que utilizamos para medir las distancias en el grafo):
  $\forall i, j, q \;\; d(i,j) \leq d(i,q) + d(q,j)$

# Optimizando Canciones - Modificado

- Podemos aprovechar una propiedad de la matriz de distancias: La desigualdad triangular. Dada una métrica (la que utilizamos para medir las distancias en el grafo):
  $\forall i, j, q \;\; d(i, j) \leq d(i, q) + d(q, j)$
- Luego, no probamos todos los acordes, sino que vemos cuál nos conviene repetir.

# Optimizando Canciones - Modificado

- Notar que cambiar un acorde por una repetición no necesariamente da un total menor, puede ser igual.
- Tomemos como ejemplo el enunciado. Sean $S_{6,7} = 3$, $S_{6,3} = 1$, $S_{3,7} = 1$, y supongamos que la canción es: $c_1, ..., 6, 3, 7, ... C_n$
- Si repetimos el 6 (o el 7) tenemos: $c_1, ..., 6, 6, 7, ..., c_n$, donde el costo $6 \rightarrow 7 = 2$ porque Rasta sigue pasado por el acorde 3 pero no lo toca.

# Optimizando Canciones - Modificado

## Algoritmo

1. Calculamos la matriz de distancias y el valor t original.

# Optimizando Canciones - Modificado

## Algoritmo

1. Calculamos la matriz de distancias y el valor t original. $O(k^3 + n)$.
2. Para cada posición i > 1, calculamos el $t_{c_{i-1} \to c_i}$ (el total de cambiar $c_i$ por repetir $c_{i-1}$, y nos quedamos con el mínimo).

# Optimizando Canciones - Modificado

## Algoritmo

1. Calculamos la matriz de distancias y el valor t original. $O(k^3 + n)$.
2. Para cada posición i > 1, calculamos el $t_{c_{i-1} \to c_i}$ (el total de cambiar $c_i$ por repetir $c_{i-1}$, y nos quedamos con el mínimo). $O(n)$
3. Costo total: $O(k^3 + n)$.

# 200 por hora

## Enunciado

Tuki se quiere ir de vacaciones, y observa que, a diferencia de cuando era joven, ahora puede ir por mejores rutas, lo cual le permite llegar en menos tiempo. Aparte, con su auto actual puede ir más rápido.

Él conoce el mapa del Buenos Aires de su infancia, y tiene una línea de tiempo indicando qué localidades fueron creadas desde entonces (junto a sus caminos) y las veces que cambió de auto (junto con las velocidades medias de los mismos). Tuki está interesado en saber cuánto tardaba en hacer ciertas rutas, de acuerdo al desarrrollo de los caminos hasta el momento y el auto que tenía.

# 200 por hora: entrada

## Entrada

La entrada consta de una secuencia de queries de 3 formas.

- `Agregar localidad` $l$, con un lista de rutas $r_l$.
- `Cambiar auto` $a$, donde $a$ indica la velocidad media del auto nuevo.
- `¿Cuánto se tarda en ir de la ciudad` $i$ `a la` $j$`?`.

# 200 por hora: ejemplo

# 200 por hora: solución

- Tenemos que proponer una estructura de datos para ir procesando las queries y guardando información. Vayamos una por una.

# 200 por hora: solución

- Tenemos que proponer una estructura de datos para ir procesando las queries y guardando información. Vayamos una por una.
- ¿Qué podemos hacer cuando recibimos `Agregar localidad` *l*?

# 200 por hora: solución

- Tenemos que proponer una estructura de datos para ir procesando las queries y guardando información. Vayamos una por una.
- ¿Qué podemos hacer cuando recibimos `Agregar localidad` /?
  **E1** Agregar el nodo al grafo con sus ejes ¿Complejidad?

# 200 por hora: solución

- Tenemos que proponer una estructura de datos para ir procesando las queries y guardando información. Vayamos una por una.
- ¿Qué podemos hacer cuando recibimos `Agregar localidad` $l$?
  - **E1** Agregar el nodo al grafo con sus ejes ¿Complejidad? $O(n)$.

# 200 por hora: solución

- Tenemos que proponer una estructura de datos para ir procesando las queries y guardando información. Vayamos una por una.
- ¿Qué podemos hacer cuando recibimos `Agregar localidad` $l$?
  - **E1** Agregar el nodo al grafo con sus ejes ¿Complejidad? $O(n)$.
  - **E2** Agregar la localidad y **recalcular** las distancias mínimas, a lo Dantzig ¿Complejidad?

# 200 por hora: solución

- Tenemos que proponer una estructura de datos para ir procesando las queries y guardando información. Vayamos una por una.
- ¿Qué podemos hacer cuando recibimos `Agregar localidad` $l$?
  - **E1** Agregar el nodo al grafo con sus ejes ¿Complejidad? $O(n)$.
  - **E2** Agregar la localidad y **recalcular** las distancias mínimas, a lo Dantzig ¿Complejidad? $O(n^2)$.
- `Cambiar auto` $a$.

# 200 por hora: solución

- Tenemos que proponer una estructura de datos para ir procesando las queries y guardando información. Vayamos una por una.
- ¿Qué podemos hacer cuando recibimos `Agregar localidad` $l$?
  - **E1** Agregar el nodo al grafo con sus ejes ¿Complejidad? $O(n)$.
  - **E2** Agregar la localidad y **recalcular** las distancias mínimas, a lo Dantzig ¿Complejidad? $O(n^2)$.
- `Cambiar auto` $a$.
  - **E1** Guardar la velocidad del auto actual.

# 200 por hora: solución

- Tenemos que proponer una estructura de datos para ir procesando las queries y guardando información. Vayamos una por una.
- ¿Qué podemos hacer cuando recibimos `Agregar localidad` *l*?
  - **E1** Agregar el nodo al grafo con sus ejes ¿Complejidad? $O(n)$.
  - **E2** Agregar la localidad y **recalcular** las distancias mínimas, a lo Dantzig ¿Complejidad? $O(n^2)$.
- `Cambiar auto` *a*.
  - **E1** Guardar la velocidad del auto actual.
  - **E2** Ídem.

# 200 por hora: solución

- ¿Cuánto de $i$ a $j$?

# 200 por hora: solución

- `¿Cuánto de` $i$ `a` $j$`?`

  **E1** Hay que calcular la distancia ¿Complejidad?

# 200 por hora: solución

- `¿Cuánto de` $i$ `a` $j$`?`
  **E1** Hay que calcular la distancia ¿Complejidad? $O(m \log n)$.

# 200 por hora: solución

- `¿Cuánto de` $i$ `a` $j$`?`
  - **E1** Hay que calcular la distancia ¿Complejidad? $O(m \log n)$.
  - **E2** Ya tenemos calculado todo ¿Complejidad?

# 200 por hora: solución

- ¿Cuánto de $i$ a $j$?
  - **E1** Hay que calcular la distancia ¿Complejidad? $O(m \log n)$.
  - **E2** Ya tenemos calculado todo ¿Complejidad? $O(1)$.

# 200 por hora: solución

- `¿Cuánto de` $i$ `a` $j$`?`
  - **E1** Hay que calcular la distancia ¿Complejidad? $O(m \log n)$.
  - **E2** Ya tenemos calculado todo ¿Complejidad? $O(1)$.
- ¿Qué solución es mejor?

# 200 por hora: solución

- `¿Cuánto de` $i$ `a` $j$`?`
  - **E1** Hay que calcular la distancia ¿Complejidad? $O(m \log n)$.
  - **E2** Ya tenemos calculado todo ¿Complejidad? $O(1)$.
- ¿Qué solución es mejor? Depende de las queries.

# 200 por hora: solución

- Sean $L$ las queries de agregar localidad, $A$ las de actualizar auto y $Q$ las de consulta. Las complejidades de cada estrategia son
  - **E1** $O(Ln + A + Qm \log n) \sim O(n^2 + A + Qm \log n)$.
  - **E2** $O(Ln^2 + A + Q) \sim O(n^3 + A + Q)$.

# 200 por hora: solución

- Sean $L$ las queries de agregar localidad, $A$ las de actualizar auto y $Q$ las de consulta. Las complejidades de cada estrategia son
  **E1** $O(Ln + A + Qm \log n) \sim O(n^2 + A + Qm \log n)$.
  **E2** $O(Ln^2 + A + Q) \sim O(n^3 + A + Q)$.
- ¿Cuándo es mejor **E1**?

# 200 por hora: solución

- Sean $L$ las queries de agregar localidad, $A$ las de actualizar auto y $Q$ las de consulta. Las complejidades de cada estrategia son
  **E1** $O(Ln + A + Qm \log n) \sim O(n^2 + A + Qm \log n)$.
  **E2** $O(Ln^2 + A + Q) \sim O(n^3 + A + Q)$.
- ¿Cuándo es mejor **E1**? Cuando $Q$ es chico. Caso contrario, es mejor **E2**.

# String problem[1]

## Enunciado

Sea $\Sigma$ un conjunto de símbolos, y sean $a, b \in \Sigma^\star$ dos cadenas de símbolos de $\Sigma$. Para cada par de símbolos $s_1, s_2$ sabemos el costo $w_{s_1 s_2}$ de reemplazar $s_1$ por $s_2$ (el cual puede ser infinito, indicando que no se puede hacer ese cambio **directo**).
En cada paso podemos tomar un caracter $s_1$ de $a$ y cambiarlo por otro $s_2$ pagando el costo asociado $w_{s_1,s_2}$. Nos piden decidir si es posible obtener $b$ a partir de $a$, y el menor costo que se debe pagar en casa de que sea posible.

[1] https://codeforces.com/contest/33/problem/B

# String problem: solución

- Claramente, si $|a| \neq |b|$ entonces la respuesta es que no se puede obtener $b$ a partir de $a$.

# String problem: solución

- Claramente, si $|a| \neq |b|$ entonces la respuesta es que no se puede obtener $b$ a partir de $a$.
- ¿Podremos descomponer el problema?

# String problem: solución

- Claramente, si $|a| \neq |b|$ entonces la respuesta es que no se puede obtener $b$ a partir de $a$.
- ¿Podremos descomponer el problema? Se puede pensar a cada posición $i$ de la cadena $a$ como un problema distinto, ya que las posiciones quedan fijas.

# String problem: solución

- Claramente, si $|a| \neq |b|$ entonces la respuesta es que no se puede obtener $b$ a partir de $a$.
- ¿Podremos descomponer el problema? Se puede pensar a cada posición $i$ de la cadena $a$ como un problema distinto, ya que las posiciones quedan fijas.
- Fijemos un $i$ ¿Cómo encontramos la forma más barata de ir de $a_i$ a $b_i$?

# String problem: solución

- Claramente, si $|a| \neq |b|$ entonces la respuesta es que no se puede obtener $b$ a partir de $a$.
- ¿Podremos descomponer el problema? Se puede pensar a cada posición $i$ de la cadena $a$ como un problema distinto, ya que las posiciones quedan fijas.
- Fijemos un $i$ ¿Cómo encontramos la forma más barata de ir de $a_i$ a $b_i$?
- Podemos armar un grafo $G$ de $|\Sigma|$ nodos y $O(|\Sigma|^2)$ ejes donde el eje de $s_1$ a $s_2$ indica el costo asociado a reemplazar $s_1$ por $s_2$. Luego, la distancia mínima en $G$ de $a_i$ a $b_i$ nos dice el costo mínimo a pagar.

# String problem: solución

- Podemos precalcular todas las distancias con Floyd ¿Qué complejidad final queda?

# String problem: solución

- Podemos precalcular todas las distancias con Floyd ¿Qué complejidad final queda? $O(|\Sigma|^3 + |a|)$.

# String problem: solución

- Podemos precalcular todas las distancias con Floyd ¿Qué complejidad final queda? $O(|\Sigma|^3 + |a|)$.
- ¿Y si no precalculamos?

# String problem: solución

- Podemos precalcular todas las distancias con Floyd ¿Qué complejidad final queda? $O(|\Sigma|^3 + |a|)$.
- ¿Y si no precalculamos? Podemos usar Dijkstra en cada paso, lo cual nos da una complejidad $O(|a||\Sigma|^2 \log |\Sigma|)$.

# String problem: solución

- Podemos precalcular todas las distancias con Floyd ¿Qué complejidad final queda? $O(|\Sigma|^3 + |a|)$.
- ¿Y si no precalculamos? Podemos usar Dijkstra en cada paso, lo cual nos da una complejidad $O(|a||\Sigma|^2 \log |\Sigma|)$.
- ¿Cuál es mejor?

# String problem: solución

- Podemos precalcular todas las distancias con Floyd ¿Qué complejidad final queda? $O(|\Sigma|^3 + |a|)$.
- ¿Y si no precalculamos? Podemos usar Dijkstra en cada paso, lo cual nos da una complejidad $O(|a||\Sigma|^2 \log |\Sigma|)$.
- ¿Cuál es mejor? Depende de $|a|$: si $|a|$ es grande entonces conviene precalcular, sino no.

# Manic Moving[2]

## Enunciado

Homero trabaja para MP (Muchos Productos) y saliendo al trabajo piensa en los locales que tiene que visitar hoy con su camión. Conoce el grafo $G$ de la ciudad, y los locales $l_1, \ldots, l_k$ que debe visitar, **en ese orden**.

Como los paquetes son grandes, solo puede llevar dos a la vez (es decir, tiene que volver a central de vez en cuando para cargar nuevos paquetes).

Sabiendo lo que Homero tarda en moverse entre cada par de nodos de la ciudad, debemos indicar cuál es el menor tiempo que le toma repartir todos los productos.

[2] https://codeforces.com/gym/101223/attachments

# Manic Moving: solución

- Supongamos que Homero solo puede llevar un paquete a la vez ¿Cuál es la solución?

# Manic Moving: solución

- Supongamos que Homero solo puede llevar un paquete a la vez ¿Cuál es la solución? Tiene que concatenar caminos mínimos de la central $c$ a $l_1$, luego de vuelta a la central, luego a $l_2$, etc.

# Manic Moving: solución

- Supongamos que Homero solo puede llevar un paquete a la vez ¿Cuál es la solución? Tiene que concatenar caminos mínimos de la central $c$ a $l_1$, luego de vuelta a la central, luego a $l_2$, etc.
- Ahora supongamos que tiene espacio extra, y va a $l_1$ con el segundo paquete ¿Qué opciones tiene?

# Manic Moving: solución

- Supongamos que Homero solo puede llevar un paquete a la vez ¿Cuál es la solución? Tiene que concatenar caminos mínimos de la central $c$ a $l_1$, luego de vuelta a la central, luego a $l_2$, etc.
- Ahora supongamos que tiene espacio extra, y va a $l_1$ con el segundo paquete ¿Qué opciones tiene? Puede ir directo a $l_2$, o pasar por central para cargar el paquete de $l_3$.

# Manic Moving: solución

- Supongamos que Homero solo puede llevar un paquete a la vez ¿Cuál es la solución? Tiene que concatenar caminos mínimos de la central $c$ a $l_1$, luego de vuelta a la central, luego a $l_2$, etc.
- Ahora supongamos que tiene espacio extra, y va a $l_1$ con el segundo paquete ¿Qué opciones tiene? Puede ir directo a $l_2$, o pasar por central para cargar el paquete de $l_3$.
- Uno podría argumentar, sin pérdida de generalidad, que si sale con dos paquetes entonces, visita dos lugares seguidos y después vuelve.

# Manic Moving: solución

- Supongamos que Homero solo puede llevar un paquete a la vez ¿Cuál es la solución? Tiene que concatenar caminos mínimos de la central $c$ a $l_1$, luego de vuelta a la central, luego a $l_2$, etc.
- Ahora supongamos que tiene espacio extra, y va a $l_1$ con el segundo paquete ¿Qué opciones tiene? Puede ir directo a $l_2$, o pasar por central para cargar el paquete de $l_3$.
- Uno podría argumentar, sin pérdida de generalidad, que si sale con dos paquetes entonces, visita dos lugares seguidos y después vuelve.
- Luego en cada paso hay dos opciones:
  - Agarra un paquete y va y vuelve al siguiente local.
  - Agarra dos paquetes y visita los dos locales, luego vuelve.

# Manic Moving: solución

- Podemos escribir una fórmula recursiva que resuelva el problema

$$hom(i) = \begin{cases} \infty & i > k+1 \\ 0 & i = k+1 \\ \min(hom(i+1) + d(c, l_i) + d(l_i, c), & \\ hom(i+2) + d(c, l_i) + d(l_i, l_{i+1}) + d(l_{i+1}, c)) & \end{cases}$$

# Manic Moving: solución

- Podemos escribir una fórmula recursiva que resuelva el problema

$$hom(i) = \begin{cases} \infty & i > k+1 \\ 0 & i = k+1 \\ \min(hom(i+1) + d(c, l_i) + d(l_i, c), & \\ hom(i+2) + d(c, l_i) + d(l_i, l_{i+1}) + d(l_{i+1}, c)) & \end{cases}$$

¿Complejidad?

# Manic Moving: solución

- Podemos escribir una fórmula recursiva que resuelva el problema

$$hom(i) = \begin{cases} \infty & i > k+1 \\ 0 & i = k+1 \\ \min(hom(i+1) + d(c, l_i) + d(l_i, c), & \\ hom(i+2) + d(c, l_i) + d(l_i, l_{i+1}) + d(l_{i+1}, c)) & \end{cases}$$

¿Complejidad? Depende de cómo se calculen los $d(\cdot, \cdot)$. Los podemos precalcular en $O(n^3)$ y luego *hom* se calcula en $O(k) = O(n)$.

# Fin

Fin