Camino Mínimo:

Todos a todos y C.M. en DAGs

Técnicas de Diseño de Algoritmos \ AED3

1er Cuatrimestre 2024

Optimizando Canciones

Timmy está aprendiendo a tocar la guitarra y se esfuerza en mejorar su velocidad para tocar acordes. Existen K acordes, y la canción con la que está practicando consta de n acordes. A Timmy le toman S_{ij} segundos pasar del acorde i al acorde j, pero a veces le es más rápido pasar por otro(s) acorde(s) intermedio(s).

De Codeforces: https://codeforces.com/problemset/gymProblem/102646/C

Optimizando Canciones

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

$S_{67} = 3\,s$
$S_{63} = 1\,s$
$S_{37} = 1\,s$

Se asume que Timmy comienza con la mano ya acomodada para el primer acorde.

¿Cuál es el mínimo tiempo que le puede ~~tomar~~ tocar la canción?

Optimizando Canciones

acordes

Canción $C = c_1, c_2, \ldots\ldots, c_M$

Dado que Timmy puede hacer más rápido pasando por acordes intermedios, es posible que pueda tocar la canción C en menos de

$$\sum_{i=1}^{M-1} S_{c_i c_{i+1}}$$ tiempo.

¿Qué habría que calcular?

Optimizando Canciones - Calculando distancias

Podemos imaginar um grafo S de k vērtices, com uma arista $i \to j$ de peso s_{ij}.

Buscamos, para todo par de acordes i, j, conocer el mínimo tiempo que necesita Timmy para pasar de i a j.

Esto es nada menos que la matriz de distancias de S.

Optimizando Canciones - Calculando distancias

En este caso, nos va a convenir usar algún algoritmo todos a todos, como Floyd-Warshall, ya que hay un s_{ij} para todo $i \neq j$.

Con esto, el valor buscado es $t = \sum_{i=1}^{n-1} d(c_i, c_{i+1})$ y lo obtenemos en $O(\underbrace{K^3}_{\text{calcular matriz}} + \underbrace{n}_{\text{recorrer la canción y sumar}})$

Optimizando Canciones - Un Cambio

Supongamos que Timmy ya conoce $d(i,j)$, el mínimo tiempo para cambiar del acorde i al j, para todo $1 < i,j < k$, y tocar la canción le toma t tiempo. Ahora, está dispuesto a cambiar un acorde de la canción por otro para disminuir un poco ese tiempo. ¿Qué cambio minimizará el tiempo total?

Optimizando Canciones – Un Cambio

tiempo total que tarda si cambia el acorde en la posición i por e $(i \in 1 \ldots n)$

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 cambio el i-ésimo acorde por e, tengo que

Para que funcione con la primera y la última canción podemos definir
$d(c_0, x) = 0$
$d(x, c_{n+1}) = 0$
para todo x

$$t_{i \leftarrow e} = t + \underbrace{\Big(d(c_{i-1}, e) - d(c_{i-1}, c_i)\Big)}_{\text{diferencia entre ir de } c_{i-1} \text{ a } e \text{ en vez de } c_i} + \underbrace{\Big(d(e, c_{i+1}) - d(c_i, c_{i+1})\Big)}_{\text{diferencia de ir a } c_{i+1} \text{ desde } e \text{ en vez de } c_i}$$

Optimizando Canciones – Propiedad

Pero aquí podemos aprovechar una propiedad de la matriz de distancias: La desigualdad triangular.

$$\forall i,j,q \quad d(i,j) \leq d(i,q) + d(q,j)$$

Luego, no es necesario probar todos los acordes en todas las posiciones, en cada posición i basta con comparar cambiar C_i por repetir C_{i-1}.

Optimizando Canciones - Propiedad

Notar que repetir el acorde no necesariamente mejora estrictamente el tiempo, puede dar igual.

Tomemos el ejemplo del planteo. Sean $S_{67}=3$, $S_{63}=1$, $S_{37}=1$ y supongamos que la canción es

$$c_1, \dots, 6 \overset{1}{\curvearrowright} 3 \overset{1}{\curvearrowright} 7, \dots c_n$$

Si repetimos el 6, tenemos

$$c_1, \dots, 6 \overset{0}{\curvearrowright} 6 \overset{2}{\curvearrowright} 7, \dots c_n$$

Este es un 2 porque $d(6,7)=2$ es decir, Timmy sigue pasando por el acorde 3 pero no lo toca

Optimizando Canciones – Algoritmo.

$O(k^3+m)$ 1. Calculamos la matriz de distancias y el valor t original.

2. Para cada posición $1 < i \leq m$

$O(m)$ calculamos $t_{c_i \leftarrow c_{i-1}}$, el total de cambiar c_i por repetir c_{i-1}, y nos quedamos con el que dé mínimo.

Costo total: $O(k^3+m)$

Calles

Tenemos un mapa con n ciudades y m rutas entre ellas. Cada ruta e tiene un cierto costo de peaje $p(e) \geq 0$ y se tarda $r(e) \in \mathbb{N}$ minutos en recorrerla. Queremos el camino de v a w de menor costo que no tome más de T minutos.

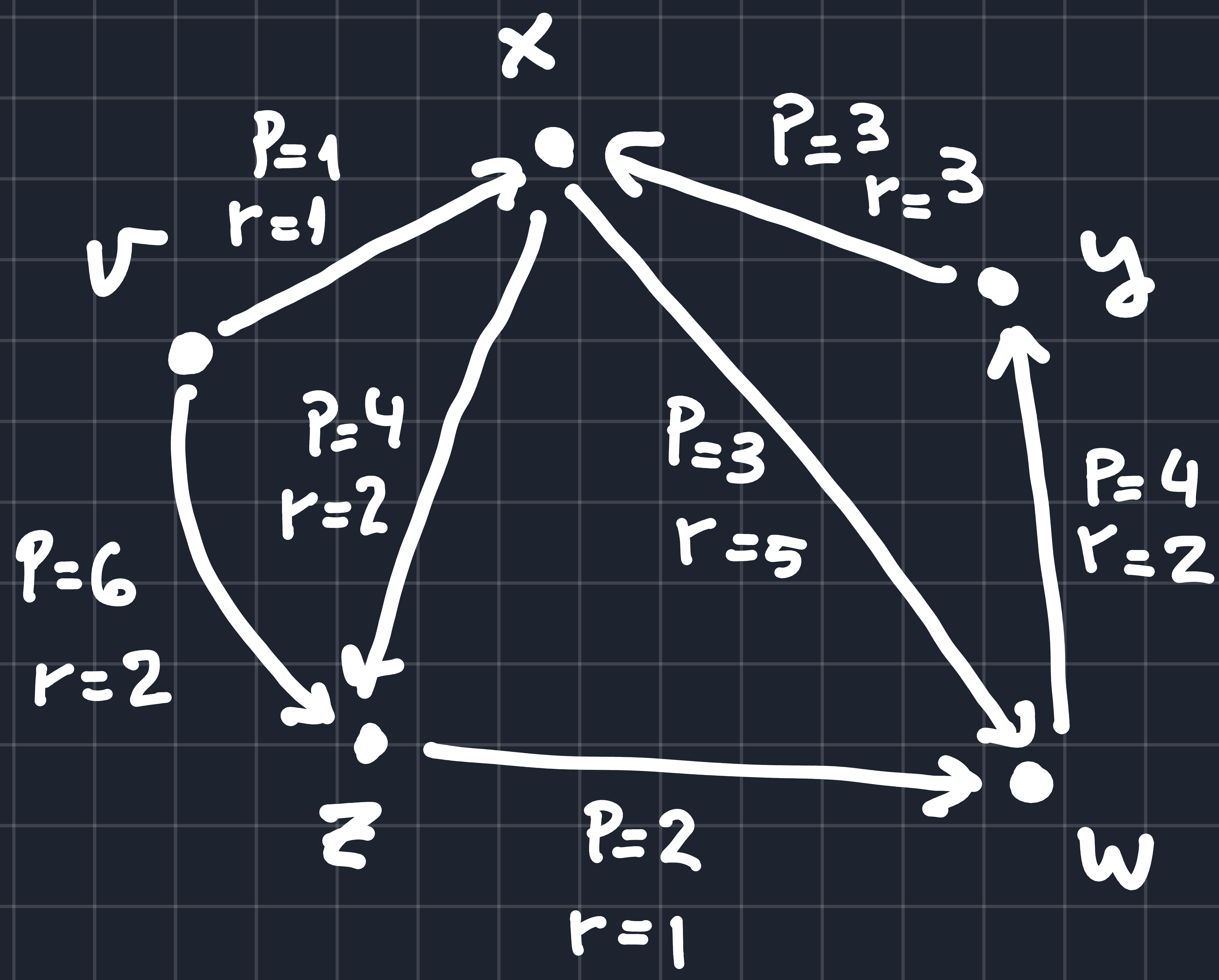
Nota: T es un valor en $O(n^{O(1)})$

Calles - Ejemplo

$G(V,E)$

$p: E \to \mathbb{R}_{\geq 0}$

$r: E \to \mathbb{N}$



Supongamos $T=5$

Hay 3 caminos de v a w:

v, x, w: $P=4$, $r=6$ ✗ SE PASA

$$\boxed{v, x, z, w: \; P=7, \; r=4}$$

v, z, w: $P=8$, $r=3$

el más barato que entra en tiempo

Notar que no es lo mismo que tomar el camino más barato de entre los que menos tardan.

Calles - Dos parámetros

Tenemos que ver cómo manejar
los dos pesos de la arista.

Calles - Idea com modelado

- Representamos cada vértice $T+1$ veces.

- ie. si partimos de $G=(V,E)$ definimos $G'=(V',E')$
  com $V'=\{(z,t') : z \in V, 0 \leq t' \leq T\}$

- $(z_1,t_1) \to (z_2,t_2) \in E'$ sii $z_1 \to z_2 \in E$ y
  $r(z_1 \to z_2) = t_2 - t_1$

- Se define la función de peso $P' : E' \to \mathbb{R}_{\geq 0}$
  $P'((z_1,t_1) \to (z_2,t_2)) = P(z_1 \to z_2)$

NOTA! Esto es polinomial porque dijimos que $T \in O(n^{O(1)})$

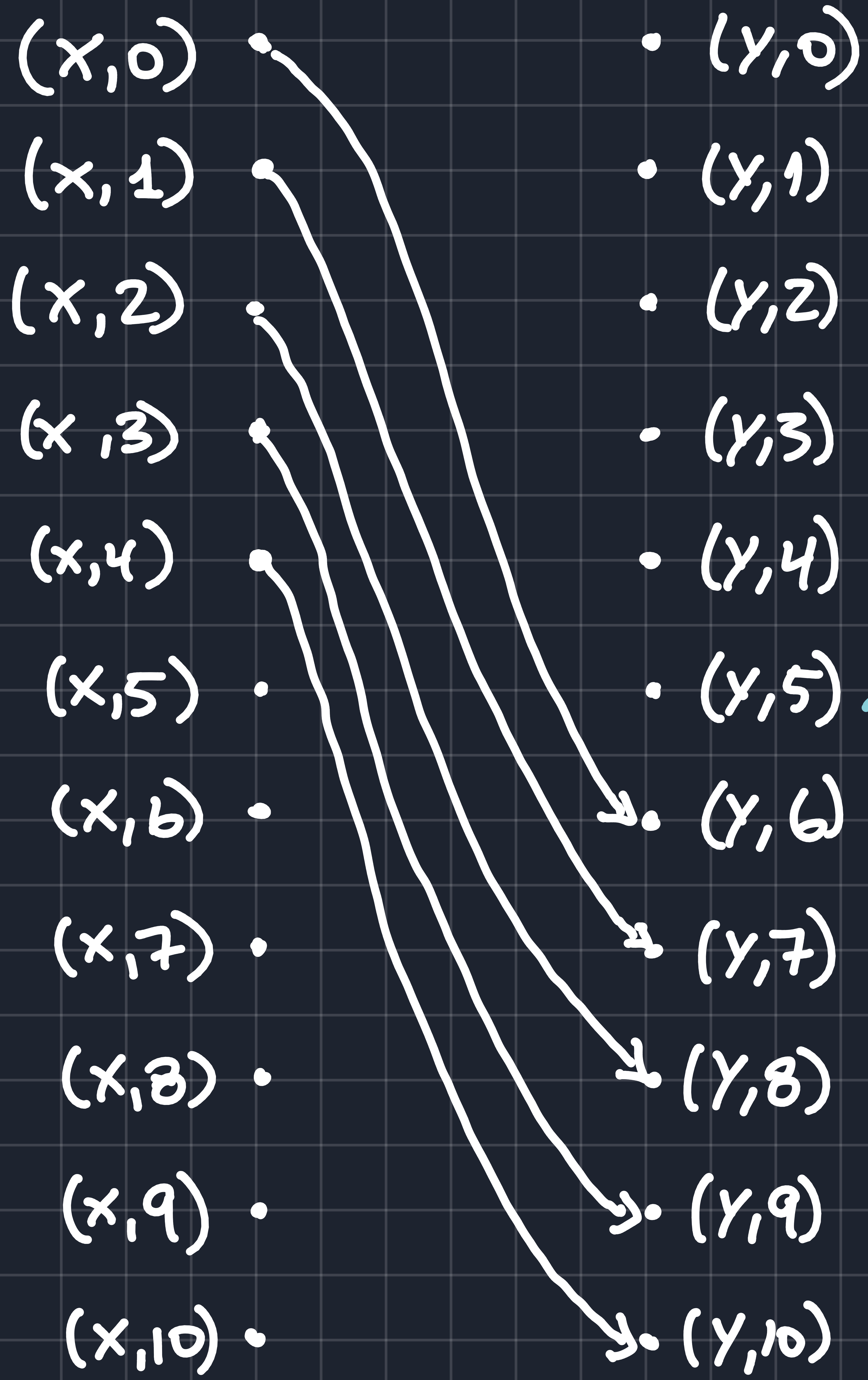
Supongamos la arista

$x \longrightarrow y$

$p=3$
$w=6$

y $T=10$.

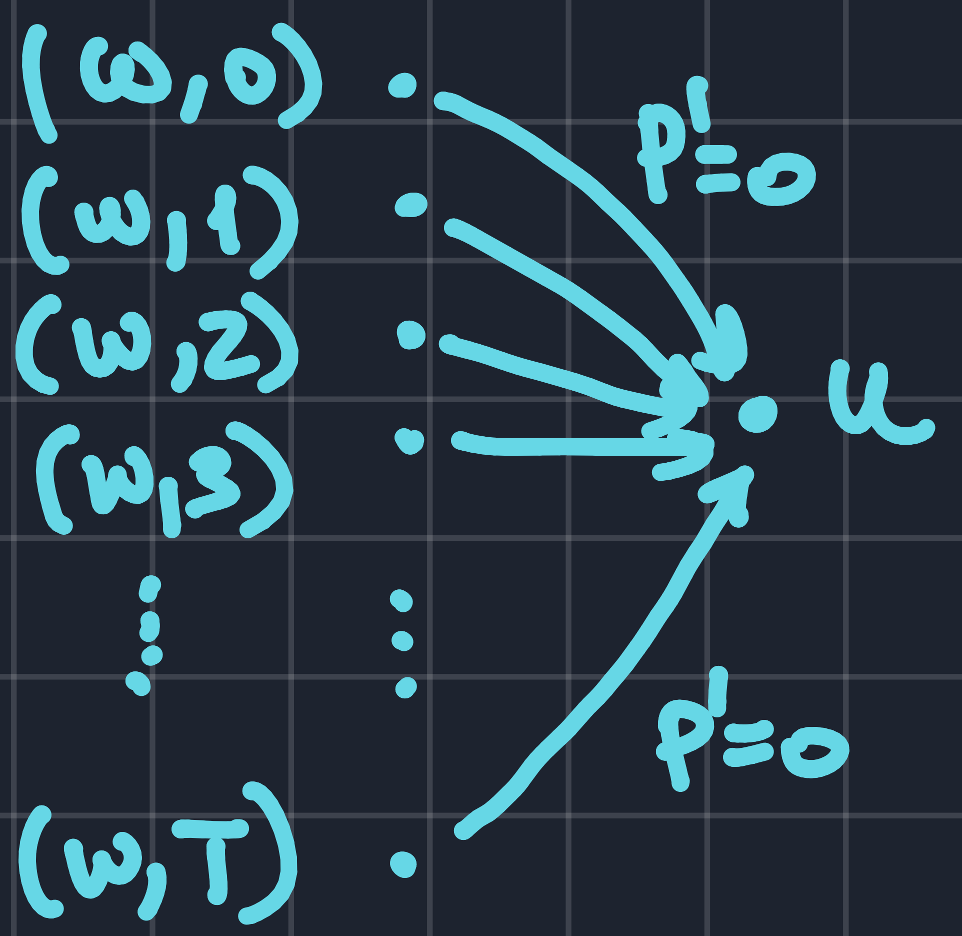
Se transforma en:



Si bien no reciben aristas desde x, podría haber otra arista en G permita que estos vertices sean alcanzables.

Calles - Tenemos grafo, ¿y ahora?

Buscamos el mínimo de los caminos de $(v, 0)$ a (w, t^*) para algún t^* (los que se pasan de tiempo no están representados en G')



¿Cómo lo obtengo?

Puedo usar Dijkstra y buscar la distancia mínima a alguno de los (w, t^*)

Puedo ahorrarme la búsqueda creando un vértice sumidero u (con aristas de peso 0)

Calles - Mejor que Dijkstra

Observación clave: Este grafo es un DAG

¿Por qué?

Toda arista une a un vértice con un t a otro con un t mayor

(todo viaje toma tiempo positivo)

Calles - Solución en DAG

Por ser DAG, G' tiene un orden topológico, y puedo usarlo para resolver camino mínimo con programación dinámica en tiempo lineal.

Calles - Solución en DAG

Por ser DAG, G' tiene un orden topológico,[1] y puedo usarlo para resolver camino mínimo con programación dinámica en tiempo lineal.[2]

(1) Para obtener el orden topológico se puede usar DFS, en este caso desde $(v,0)$ alcanza

(2) Algoritmo bottom-up.

(la distancia en este caso el el mínimo costo de peaje a pagar)

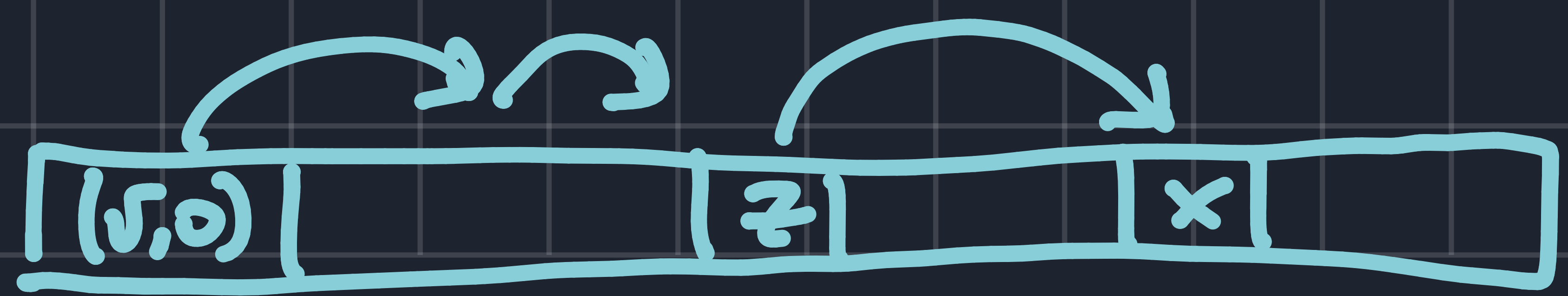
inicializo la distancia $d(x)=\infty$ para todo x alcanzable desde $(v,0)$, y $d((v,0))=0$

para todo x en o.t. desde $(v,0)$:

  para todo z con arista hacia x:

    $d(x)=\min(d(x), d(z)+p'(z\to x))$

Por ser DAG, cuando llego a x todo vértice alcanzable desde $(v,0)$ tiene su d calculada, y los no alcanzables tienen ∞



Calles - Algoritmo

hay T+1 vértices por cada vértice de G

$$|V(G')| = (T+1)|V(G)| + 1 \in O(Tn)$$

$O(|V(G')| + |E(G')|) = O(T(n+m))$

1. Construyo G', agregando el vértice sumidero u.

$$|E(G')| = \sum_{e \in E(G)} (T + 1 - r(e)) + T + 1 \in O(Tm)$$

por cada arista de G hay $(T+1) - r(e)$ aristas en G', y se suman T+1 aristas hacia u.

$O(T(n+m))$

2. Calculo el orden topológico de G'

$O(T(n+m))$

3. Resuelvo camino mínimo desde $(v,0)$ a u usando el orden topológico.

$$O(T(n+m))$$

Esto es polinomial <u>en tanto T es "chico"</u>