

UNIVERSIDAD NACIONAL DE INGENIERÍA

Análisis y Diseño de Algoritmos

Qué hacer con un problema difícil

Poda, aproximación, heurísticas y repaso

Gerson Garrido

Semana 15

Hoy

Sesión A

1. Las preguntas antes de elegir
2. El viajero con datos de verdad
3. Backtracking con poda
4. Aproximación: cobertura de vértices
5. Viajero métrico: duplicar el árbol
6. Sin desigualdad triangular
7. Heurísticas: 2-opt
8. La comparación completa

Sesión B

9. Programación lineal como idea **[se puede cortar]**
10. Repaso: ¿qué técnica sirve?
11. Qué sigue

> **Lectura**
>
> Apuntes de la semana 15. Código en `code/week-15`.

SECCIÓN 01

Las preguntas antes de elegir

01

“Es NP-completo” no es el final

Salvo que P = NP, ningún algoritmo es rápido **y** exacto en todos los casos.

“Es NP-completo” no es el final

Salvo que P = NP, ningún algoritmo es rápido **y** exacto en todos los casos.

Hay que hacer algo, y conviene no mezclar tres cosas:

“Es NP-completo” no es el final

Salvo que P = NP, ningún algoritmo es rápido **y** exacto en todos los casos.

Hay que hacer algo, y conviene no mezclar tres cosas:

- **Búsqueda exacta con poda:** da siempre el óptimo. Su tiempo puede ser exponencial.

“Es NP-completo” no es el final

Salvo que P = NP, ningún algoritmo es rápido **y** exacto en todos los casos.

Hay que hacer algo, y conviene no mezclar tres cosas:

- **Búsqueda exacta con poda:** da siempre el óptimo. Su tiempo puede ser exponencial.
- **Aproximación:** polinomial, con una garantía *demostrada*, sin conocer el óptimo.

“Es NP-completo” no es el final

Salvo que P = NP, ningún algoritmo es rápido **y** exacto en todos los casos.

Hay que hacer algo, y conviene no mezclar tres cosas:

- **Búsqueda exacta con poda:** da siempre el óptimo. Su tiempo puede ser exponencial.
- **Aproximación:** polinomial, con una garantía *demostrada*, sin conocer el óptimo.
- **Heurística:** suele dar buenas respuestas. No promete nada.

“Es NP-completo” no es el final

Salvo que P = NP, ningún algoritmo es rápido **y** exacto en todos los casos.

Hay que hacer algo, y conviene no mezclar tres cosas:

- **Búsqueda exacta con poda:** da siempre el óptimo. Su tiempo puede ser exponencial.
- **Aproximación:** polinomial, con una garantía *demostrada*, sin conocer el óptimo.
- **Heurística:** suele dar buenas respuestas. No promete nada.

Las tres sobre el mismo problema: el viajero, con las capitales regionales del Perú.

Cuatro preguntas

Pregunta	Si la respuesta es sí	Qué se pierde
¿La entrada es pequeña?	Held-Karp, poda	Nada, mientras siga chica
¿Tiene estructura especial?	Un algoritmo para ese caso	Solo sirve para ese caso
¿Sirve algo cercano, con garantía?	Aproximación	El óptimo
¿Sirve algo cercano, sin garantía?	Heurística	La garantía
¿Hay un solucionador?	Traducir el problema	El control sobre el tiempo

No se excluyen

Una heurística da un buen recorrido, y ese recorrido ayuda a una búsqueda exacta a podar.

Casos especiales que ya vimos

- **Cobertura de vértices en árboles** (semana 9). En general es NP-difícil. En un árbol, programación dinámica lineal: quitar un nodo separa partes independientes.

Casos especiales que ya vimos

- **Cobertura de vértices en árboles** (semana 9). En general es NP-difícil. En un árbol, programación dinámica lineal: quitar un nodo separa partes independientes.

- **Mochila con números pequeños** (semana 7). $O(nW)$, pseudopolinomial. La dificultad estaba en los números enormes.

Casos especiales que ya vimos

- **Cobertura de vértices en árboles** (semana 9). En general es NP-difícil. En un árbol, programación dinámica lineal: quitar un nodo separa partes independientes.
- **Mochila con números pequeños** (semana 7). $O(nW)$, pseudopolinomial. La dificultad estaba en los números enormes.
- **Fronteras** (semana 14). 2-SAT frente a 3-SAT, 2-coloreo frente a 3-coloreo.

Casos especiales que ya vimos

- **Cobertura de vértices en árboles** (semana 9). En general es NP-difícil. En un árbol, programación dinámica lineal: quitar un nodo separa partes independientes.
- **Mochila con números pequeños** (semana 7). $O(nW)$, pseudopolinomial. La dificultad estaba en los números enormes.
- **Fronteras** (semana 14). 2-SAT frente a 3-SAT, 2-coloreo frente a 3-coloreo.

La demostración de NP-completitud habla del problema general. El suyo puede ser un caso particular.

SECCIÓN 02

El viajero con datos de verdad

02

Los datos y la distancia

26 lugares, sedes de gobierno regional y Lima. Coordenadas de GeoNames, CC BY 4.0. Subconjuntos fijos: las primeras 10, 15 y 20 filas de `capitales.csv`.

Gran círculo, con la fórmula de haversine

$$h = \sin^2 \frac{\varphi_2 - \varphi_1}{2} + \cos\varphi_1 \cos\varphi_2 \sin^2 \frac{\lambda_2 - \lambda_1}{2} \qquad d = 2R \arcsin\sqrt{h} \qquad R = 6371 \text{ km}$$

No es una carretera

Es el camino más corto sobre una esfera. Es un modelo, elegido porque tiene una propiedad que vamos a necesitar.

La desigualdad triangular

Viajero métrico

Distancias simétricas, no negativas, y para tres nodos cualesquiera:

$$d(a, c) \leq d(a, b) + d(b, c)$$

La desigualdad triangular

Viajero métrico

Distancias simétricas, no negativas, y para tres nodos cualesquiera:

$$d(a, c) \leq d(a, b) + d(b, c)$$

El gran círculo la cumple: $d(a, c)$ es la curva más corta de a a c, y pegar la de a a b con la de b a c también es una curva de a a c.

La desigualdad triangular

Viajero métrico

Distancias simétricas, no negativas, y para tres nodos cualesquiera:

$$d(a, c) \leq d(a, b) + d(b, c)$$

El gran círculo la cumple: $d(a, c)$ es la curva más corta de a a c, y pegar la de a a b con la de b a c también es una curva de a a c.

El problema aparece al **redondear**.

Redondear hacia arriba sí la conserva

build/metric_demo, extracto

```
Tres puntos sobre el ecuador:
  reales: ab = 1000.400, bc = 1000.400, ac = 2000.800
  al más cercano: 1000 + 1000 < 2001
  hacia arriba: 1001 + 1001 >= 2001
```

Afirmación

Si $a \leq b + c$, con reales, entonces $\lceil a \rceil \leq \lceil b \rceil + \lceil c \rceil$.

Redondear hacia arriba sí la conserva

build/metric_demo, extracto

```
Tres puntos sobre el ecuador:
  reales: ab = 1000.400, bc = 1000.400, ac = 2000.800
  al más cercano: 1000 + 1000 < 2001
  hacia arriba: 1001 + 1001 >= 2001
```

Afirmación

Si $a \leq b + c$, con reales, entonces $\lceil a \rceil \leq \lceil b \rceil + \lceil c \rceil$.

1. $a \leq b + c \leq \lceil b \rceil + \lceil c \rceil$.

Redondear hacia arriba sí la conserva

build/metric_demo, extracto

```
Tres puntos sobre el ecuador:
  reales: ab = 1000.400, bc = 1000.400, ac = 2000.800
  al más cercano: 1000 + 1000 < 2001
  hacia arriba: 1001 + 1001 >= 2001
```

Afirmación

Si $a \le b + c$, con reales, entonces $\lceil a \rceil \le \lceil b \rceil + \lceil c \rceil$.

1. $a \le b + c \le \lceil b \rceil + \lceil c \rceil$.
2. El lado derecho es un entero que no queda debajo de a.

Redondear hacia arriba sí la conserva

build/metric_demo, extracto

```
Tres puntos sobre el ecuador:
  reales: ab = 1000.400, bc = 1000.400, ac = 2000.800
  al más cercano: 1000 + 1000 < 2001
  hacia arriba: 1001 + 1001 >= 2001
```

Afirmación

Si $a \leq b + c$, con reales, entonces $\lceil a \rceil \leq \lceil b \rceil + \lceil c \rceil$.

1. $a \leq b + c \leq \lceil b \rceil + \lceil c \rceil$.
2. El lado derecho es un entero que no queda debajo de a.
3. $\lceil a \rceil$ es el menor entero así. Por lo tanto, $\lceil a \rceil \leq \lceil b \rceil + \lceil c \rceil$. ■

Y mirando todas las ternas

build/metric_demo, extracto

```
más cercanos: Callao y Lima, 11595.553 m
más lejanos: Tacna y Tumbes, 1953471.901 m
lo más cerca de un entero que queda una distancia: 0.000237 m
peor error relativo de tratar los grados como un plano: 3.577 %

Desigualdad triangular, d(a,c) <= d(a,b) + d(b,c), en todas las ternas:
  double, en metros: 0 de 15600 ternas la rompen
  entero, redondeado hacia arriba: 0 de 15600 ternas la rompen
  entero, al más cercano: 0 de 15600 ternas la rompen
la terna más justa: Callao, Chiclayo, Piura, le sobran 0.003 m
```

Y mirando todas las ternas

build/metric_demo, extracto

```
más cercanos: Callao y Lima, 11595.553 m
más lejanos: Tacna y Tumbes, 1953471.901 m
lo más cerca de un entero que queda una distancia: 0.000237 m
peor error relativo de tratar los grados como un plano: 3.577 %

Desigualdad triangular, d(a,c) <= d(a,b) + d(b,c), en todas las ternas:
  double, en metros: 0 de 15600 ternas la rompen
  entero, redondeado hacia arriba: 0 de 15600 ternas la rompen
  entero, al más cercano: 0 de 15600 ternas la rompen
la terna más justa: Callao, Chiclayo, Piura, le sobran 0.003 m
```

Aquí, ni el redondeo al más cercano la rompe, pero eso fue suerte de las posiciones.

Y mirando todas las ternas

build/metric_demo, extracto

```
más cercanos: Callao y Lima, 11595.553 m
más lejanos: Tacna y Tumbes, 1953471.901 m
lo más cerca de un entero que queda una distancia: 0.000237 m
peor error relativo de tratar los grados como un plano: 3.577 %

Desigualdad triangular, d(a,c) <= d(a,b) + d(b,c), en todas las ternas:
  double, en metros: 0 de 15600 ternas la rompen
  entero, redondeado hacia arriba: 0 de 15600 ternas la rompen
  entero, al más cercano: 0 de 15600 ternas la rompen
la terna más justa: Callao, Chiclayo, Piura, le sobran 0.003 m
```

Aquí, ni el redondeo al más cercano la rompe, pero eso fue suerte de las posiciones. Usamos **metros enteros, hacia arriba**: comparar recorridos es exacto y cada mejora baja al menos 1 m.

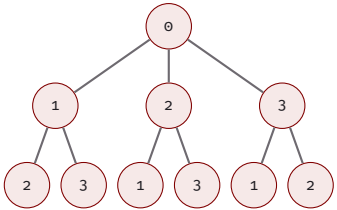
SECCIÓN 03

Backtracking con poda

03

El árbol de búsqueda

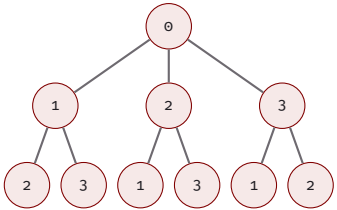
Cada nodo es un recorrido parcial: salió de 0 y lleva visitadas algunas ciudades.



El árbol de búsqueda

Cada nodo es un recorrido parcial: salió de 0 y lleva visitadas algunas ciudades.

Sin poda, con $n = 10$, son 986410 nodos.
Con 26 ciudades, las hojas son $25! \approx 1{,}55 \times 10^{25}$.

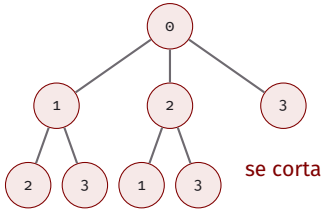


El árbol de búsqueda

Cada nodo es un recorrido parcial: salió de 0 y lleva visitadas algunas ciudades.

Sin poda, con $n = 10$, son 986410 nodos. Con 26 ciudades, las hojas son $25! \approx 1{,}55 \times 10^{25}$.

Si una rama no puede bajar del mejor recorrido conocido, **se corta**.



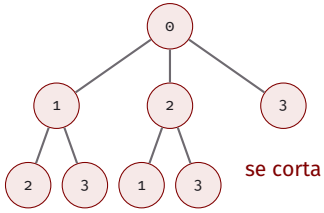
El árbol de búsqueda

Cada nodo es un recorrido parcial: salió de 0 y lleva visitadas algunas ciudades.

Sin poda, con $n = 10$, son 986410 nodos. Con 26 ciudades, las hojas son $25! \approx 1{,}55 \times 10^{25}$.

Si una rama no puede bajar del mejor recorrido conocido, **se corta**.

Es la **poda** *(pruning)*, y en su forma general, **ramificación y acotamiento** *(branch and bound)*.



La búsqueda

pseudocódigo

```
algoritmo buscar(ultimo, resto, largo)
    si resto está vacío entonces
        mejor ← mín(mejor, largo + d(ultimo, o))
        devolver
    fin si
    si largo + cota(ultimo, resto) ≥ mejor entonces
        devolver
    fin si
    para cada v en resto, de más cerca a más lejos de ultimo hacer
        buscar(v, resto - {v}, largo + d(ultimo, v))
    fin para
fin algoritmo
```

La búsqueda

pseudocódigo

```
algoritmo buscar(ultimo, resto, largo)
    si resto está vacío entonces
        mejor ← mín(mejor, largo + d(ultimo, o))
        devolver
    fin si
    si largo + cota(ultimo, resto) ≥ mejor entonces
        devolver
    fin si
    para cada v en resto, de más cerca a más lejos de ultimo hacer
        buscar(v, resto - {v}, largo + d(ultimo, v))
    fin para
fin algoritmo
```

`mejor` empieza en infinito. Probar primero lo más cercano no cambia el resultado: encuentra pronto un buen recorrido, y eso poda más.

Cuándo una cota es válida

Condición

`cota(ultimo, resto)` nunca pasa del largo de la mejor forma de terminar. Es una **cota inferior**.

Teorema. Con esa condición, el algoritmo devuelve el óptimo.

Cuándo una cota es válida

Condición

`cota(ultimo, resto)` nunca pasa del largo de la mejor forma de terminar. Es una **cota inferior**.

Teorema. Con esa condición, el algoritmo devuelve el óptimo.

1. Se poda solo si largo + cota $\geq$ mejor.

Cuándo una cota es válida

Condición

`cota(ultimo, resto)` nunca pasa del largo de la mejor forma de terminar. Es una **cota inferior**.

Teorema. Con esa condición, el algoritmo devuelve el óptimo.

1. Se poda solo si largo + cota $\geq$ mejor.
2. Todo recorrido de esa rama mide largo más lo que cueste terminar, y eso es al menos la cota.

Cuándo una cota es válida

Condición

`cota(ultimo, resto)` nunca pasa del largo de la mejor forma de terminar. Es una **cota inferior**.

Teorema. Con esa condición, el algoritmo devuelve el óptimo.

1. Se poda solo si largo + cota $\geq$ mejor.
2. Todo recorrido de esa rama mide largo más lo que cueste terminar, y eso es al menos la cota.
3. Así, ninguno es estrictamente mejor que `mejor`, que ya es un recorrido real.

Cuándo una cota es válida

Condición

`cota(ultimo, resto)` nunca pasa del largo de la mejor forma de terminar. Es una **cota inferior**.

Teorema. Con esa condición, el algoritmo devuelve el óptimo.

1. Se poda solo si largo + cota $\geq$ mejor.
2. Todo recorrido de esa rama mide largo más lo que cueste terminar, y eso es al menos la cota.
3. Así, ninguno es estrictamente mejor que `mejor`, que ya es un recorrido real.
4. Ningún corte pierde el óptimo. ■

Cotas que sirven y cotas que no

La más simple es 0: se poda si lo ya recorrido alcanza a `mejor`. Es válida, pero no mira lo que falta.

Cotas que sirven y cotas que no

La más simple es 0: se poda si lo ya recorrido alcanza a `mejor`. Es válida, pero no mira lo que falta.

Error común

Usar una “cota” que se puede pasar, como el largo de un recorrido aproximado o un promedio. Poda de más, y da una respuesta que no es óptima, sin avisar.

Cotas que sirven y cotas que no

La más simple es 0: se poda si lo ya recorrido alcanza a `mejor`. Es válida, pero no mira lo que falta.

Error común

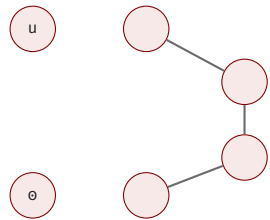
Usar una “cota” que se puede pasar, como el largo de un recorrido aproximado o un promedio. Poda de más, y da una respuesta que no es óptima, sin avisar.

La diferencia

Una cota floja solo cuesta tiempo. Una cota que se pasa cuesta la correctitud.

La cota de árbol

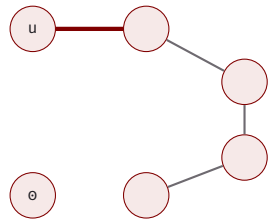
Cualquier forma de terminar es `ultimo` $\to r_1 \to \cdots \to r_k \to 0$.



La cota de árbol

Cualquier forma de terminar es `ultimo` $\to r_1 \to \cdots \to r_k \to 0$.

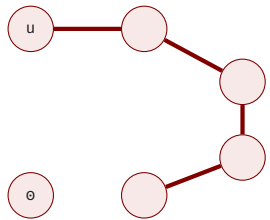
- La primera arista mide al menos $\min d(\texttt{ultimo}, r)$.



La cota de árbol

Cualquier forma de terminar es `ultimo` $\to r_1 \to \cdots \to r_k \to 0$.

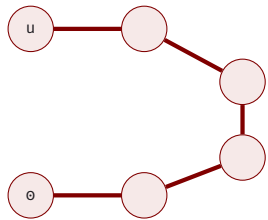
- La primera arista mide al menos mín $d(\texttt{ultimo}, r)$.
- Las del medio son un camino por todo `resto`: un árbol de expansión, que pesa al menos A.



La cota de árbol

Cualquier forma de terminar es `ultimo` $\to r_1 \to \cdots \to r_k \to 0$.

- La primera arista mide al menos mín $d(\texttt{ultimo}, r)$.
- Las del medio son un camino por todo `resto`: un árbol de expansión, que pesa al menos A.
- La última mide al menos mín $d(r, 0)$.



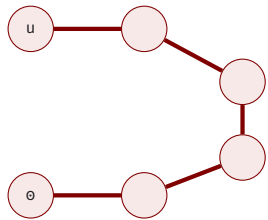
u = ultimo

La cota de árbol

Cualquier forma de terminar es `ultimo` $\to r_1 \to \cdots \to r_k \to 0$.

- La primera arista mide al menos $\min d(\texttt{ultimo}, r)$.
- Las del medio son un camino por todo `resto`: un árbol de expansión, que pesa al menos A.
- La última mide al menos $\min d(r, 0)$.

$$\text{cota} = \min_r d(\texttt{ultimo}, r) + A + \min_r d(r, 0)$$



u = ultimo

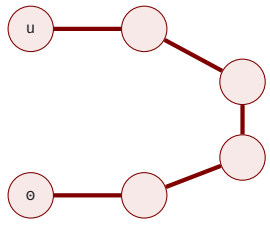
La cota de árbol

Cualquier forma de terminar es `ultimo` $\to r_1 \to \cdots \to r_k \to 0$.

- La primera arista mide al menos $\min d(\texttt{ultimo}, r)$.
- Las del medio son un camino por todo `resto`: un árbol de expansión, que pesa al menos A.
- La última mide al menos $\min d(r, 0)$.

$$\text{cota} = \min_r d(\texttt{ultimo}, r) + A + \min_r d(r, 0)$$

No usa la desigualdad triangular. Cuesta $O(r^2)$ por nodo.



u = ultimo

Medir la poda: contar nodos

build/bound_demo

n	sin poda	lo recorrido	con cota de árbol	costo (m)
6	326	290	86	2646313
8	13700	6290	535	3085034
10	986410	146810	3157	3167199
11	9864101	802246	8567	3177709
12	-	4768125	23473	3180694
13	-	24676909	49358	3200397

Medir la poda: contar nodos

build/bound_demo

n	sin poda	lo recorrido	con cota de árbol	costo (m)
6	326	290	86	2646313
8	13700	6290	535	3085034
10	986410	146810	3157	3167199
11	9864101	802246	8567	3177709
12	-	4768125	23473	3180694
13	-	24676909	49358	3200397

Contar nodos es mejor que medir segundos: el número no depende de la máquina. Las tres podas dan el mismo costo que Held-Karp en 200 instancias de las pruebas.

Con los casos grandes

build/bound_bench: una corrida, -O2, depende de la máquina

n	poda	nodos	costo	tiempo
15	lo recorrido	622800727	3225.0 km	21.44 s
15	con cota de árbol	71523	3225.0 km	0.02 s
20	con cota de árbol	13042082	4640.2 km	4.93 s
26	con cota de árbol	18575558	5621.6 km	14.91 s

Con los casos grandes

build/bound_bench: una corrida, -O2, depende de la máquina

n	poda	nodos	costo	tiempo
15	lo recorrido	622800727	3225.0 km	21.44 s
15	con cota de árbol	71523	3225.0 km	0.02 s
20	con cota de árbol	13042082	4640.2 km	4.93 s
26	con cota de árbol	18575558	5621.6 km	14.91 s

Con 26 terminó, y Held-Karp necesitaría unos 6,7 GB de tabla. El árbol podado depende de los datos: en el peor caso sigue siendo exponencial.

Poda no es aproximación

La poda cambia el tiempo, no la respuesta

Una búsqueda con poda es un algoritmo **exacto**. No hay factor ni “casi”: da el óptimo, y no sabemos de antemano cuánto va a tardar.

Poda no es aproximación

La poda cambia el tiempo, no la respuesta

Una búsqueda con poda es un algoritmo **exacto**. No hay factor ni “casi”: da el óptimo, y no sabemos de antemano cuánto va a tardar.

Los programas especializados usan cotas mucho más fuertes y un buen recorrido inicial, y han resuelto instancias de miles de ciudades.

Poda no es aproximación

La poda cambia el tiempo, no la respuesta

Una búsqueda con poda es un algoritmo **exacto**. No hay factor ni “casi”: da el óptimo, y no sabemos de antemano cuánto va a tardar.

Los programas especializados usan cotas mucho más fuertes y un buen recorrido inicial, y han resuelto instancias de miles de ciudades.

Son tan exponenciales como esta en el peor caso. Solo que los casos que importan no son el peor.

SECCIÓN 04

Aproximación: cobertura de vértices

04

Qué es un algoritmo de aproximación

Algoritmo de aproximación ***(approximation algorithm)***

Corre en tiempo polinomial y tiene **factor** $\alpha \geq 1$ si, para **toda** instancia,

$$\text{valor que devuelve} \leq \alpha \cdot \text{OPT}$$

Qué es un algoritmo de aproximación

Algoritmo de aproximación ***(approximation algorithm)***

Corre en tiempo polinomial y tiene **factor** $\alpha \geq 1$ si, para **toda** instancia,

$$\text{valor que devuelve} \leq \alpha \cdot \text{OPT}$$

Medir el resultado en 100 instancias no da un factor. Se **demuestra**. ¿Cómo, si OPT es lo que no se sabe calcular?

Qué es un algoritmo de aproximación

Algoritmo de aproximación *(approximation algorithm)*

Corre en tiempo polinomial y tiene **factor** $\alpha \geq 1$ si, para **toda** instancia,

$$\text{valor que devuelve} \leq \alpha \cdot \text{OPT}$$

Medir el resultado en 100 instancias no da un factor. Se **demuestra**. ¿Cómo, si OPT es lo que no se sabe calcular?

La idea de toda esta parte

No se compara contra OPT. Se compara contra una **cota inferior** $L \leq \text{OPT}$ que sí se puede calcular.

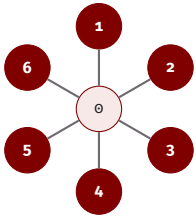
$$\text{valor} \leq \alpha L \leq \alpha \cdot \text{OPT}$$

Una idea parecida que falla

Por cada arista sin cubrir, elegir una de sus puntas.

build/cover_demo, extracto

```
Estrella con centro 0 y 6 hojas:
  una sola punta: {1, 2, 3, 4, 5, 6}
  matching maximal, 1 aristas: {0, 1}
  mayor grado: {0}
  óptimo por fuerza bruta: 1
```

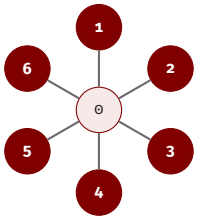


Una idea parecida que falla

Por cada arista sin cubrir, elegir una de sus puntas.

build/cover_demo, extracto

```
Estrella con centro 0 y 6 hojas:
  una sola punta: {1, 2, 3, 4, 5, 6}
  matching maximal, 1 aristas: {0, 1}
  mayor grado: {0}
  óptimo por fuerza bruta: 1
```



Con $n - 1$ hojas, da $n - 1$ frente a 1. Ningún factor constante le sirve: escoger una punta “por casualidad” no dice nada de lo que hace el óptimo.

La idea que suena peor: las dos puntas

pseudocódigo

```
algoritmo cobertura_por_matching(aristas)
    C ← vacío
    M ← vacío
    para cada arista {u, v} en aristas hacer
        si u no está en C y v no está en C entonces
            M ← M + {{u, v}}
            C ← C + {u, v}
        fin si
    fin para
    devolver C
fin algoritmo
```

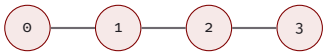
La idea que suena peor: las dos puntas

pseudocódigo

```
algoritmo cobertura_por_matching(aristas)
    C ← vacío
    M ← vacío
    para cada arista {u, v} en aristas hacer
        si u no está en C y v no está en C entonces
            M ← M + {{u, v}}
            C ← C + {u, v}
        fin si
    fin para
    devolver C
fin algoritmo
```

M es un **matching**: sus aristas no comparten nodos. Y es **maximal**: no le cabe ninguna arista más.

Maximal no es máximo



Maximal no es máximo



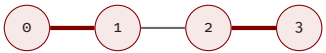
primero la arista 1-2

Maximal: no le cabe nada más sin sacar algo. Aquí tiene tamaño 1, y da $\{1, 2\}$.

Maximal no es máximo



primero la arista 1-2



primero la arista 0-1

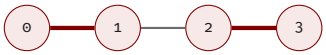
Maximal: no le cabe nada más sin sacar algo. Aquí tiene tamaño 1, y da $\{1, 2\}$.

Máximo: no hay otro más grande. Aquí tiene tamaño 2, y da $\{0, 1, 2, 3\}$.

Maximal no es máximo



primero la arista 1-2



primero la arista 0-1

Maximal: no le cabe nada más sin sacar algo. Aquí tiene tamaño 1, y da $\{1, 2\}$.

Máximo: no hay otro más grande. Aquí tiene tamaño 2, y da $\{0, 1, 2, 3\}$.

Basta con maximal, que sale con una sola pasada por las aristas. El máximo es otro problema, y aquí no hace falta.

Factor 2

Teorema. C es una cobertura, con $|C| \leq 2 \cdot$ OPT. Técnica: la cota inferior es $|M|$.

Factor 2

Teorema. C es una cobertura, con $|C| \leq 2 \cdot$ OPT. Técnica: la cota inferior es $|M|$.

1. **Es una cobertura.** Al revisar una arista, o ya tenía una punta en C, o se agregaron las dos.

Factor 2

Teorema. C es una cobertura, con $|C| \leq 2 \cdot$ OPT. Técnica: la cota inferior es $|M|$.

1. **Es una cobertura.** Al revisar una arista, o ya tenía una punta en C, o se agregaron las dos.
2. **OPT** $\geq |M|$**.** Toda cobertura C^* tiene una punta de cada arista de M.

Factor 2

Teorema. C es una cobertura, con $|C| \leq 2 \cdot$ OPT. Técnica: la cota inferior es $|M|$.

1. **Es una cobertura.** Al revisar una arista, o ya tenía una punta en C, o se agregaron las dos.
2. **OPT** $\geq |M|$**.** Toda cobertura C^* tiene una punta de cada arista de M.
3. Las aristas de M no comparten nodos, así que esas puntas son todas distintas: una por arista.

Factor 2

Teorema. C es una cobertura, con $|C| \leq 2 \cdot$ OPT. Técnica: la cota inferior es $|M|$.

1. **Es una cobertura.** Al revisar una arista, o ya tenía una punta en C, o se agregaron las dos.
2. **OPT** $\geq |M|$**.** Toda cobertura C^* tiene una punta de cada arista de M.
3. Las aristas de M no comparten nodos, así que esas puntas son todas distintas: una por arista.
4. $|C| = 2|M|$**.** Cada arista de M puso dos nodos, sin repetir.

Factor 2

Teorema. C es una cobertura, con $|C| \leq 2 \cdot$ OPT. Técnica: la cota inferior es $|M|$.

1. **Es una cobertura.** Al revisar una arista, o ya tenía una punta en C, o se agregaron las dos.
2. **OPT** $\geq |M|$**.** Toda cobertura C^* tiene una punta de cada arista de M.
3. Las aristas de M no comparten nodos, así que esas puntas son todas distintas: una por arista.
4. $|C| = 2|M|$**.** Cada arista de M puso dos nodos, sin repetir.
5. $|C| = 2|M| \leq 2 \cdot$ OPT. ■

Factor 2

Teorema. C es una cobertura, con $|C| \leq 2 \cdot$ OPT. Técnica: la cota inferior es $|M|$.

1. **Es una cobertura.** Al revisar una arista, o ya tenía una punta en C, o se agregaron las dos.
2. **OPT** $\geq |M|$**.** Toda cobertura C^* tiene una punta de cada arista de M.
3. Las aristas de M no comparten nodos, así que esas puntas son todas distintas: una por arista.
4. $|C| = 2|M|$**.** Cada arista de M puso dos nodos, sin repetir.
5. $|C| = 2|M| \leq 2 \cdot$ OPT. ■

Se alcanza exactamente

Con una sola arista, elige 2 nodos, y OPT = 1. El análisis no tiene holgura. Nunca se calculó OPT.

¿Y el mayor grado?

La regla natural: elegir el nodo que cubre más aristas sin cubrir, y repetir.

build/cover_demo, extracto

```
300 grafos al azar, 16 nodos, cada arista con probabilidad 1/4:
  promedio del óptimo: 8.71333
  promedio del matching: 13.18, peor razón 2
  promedio del mayor grado: 9, peor razón 1.25
```

¿Y el mayor grado?

La regla natural: elegir el nodo que cubre más aristas sin cubrir, y repetir.

build/cover_demo, extracto

```
300 grafos al azar, 16 nodos, cada arista con probabilidad 1/4:
  promedio del óptimo: 8.71333
  promedio del matching: 13.18, peor razón 2
  promedio del mayor grado: 9, peor razón 1.25
```

En estos grafos gana con mucha ventaja. Si esto fuera todo lo que sabemos, la elegiríamos sin pensar.

¿Y el mayor grado?

La regla natural: elegir el nodo que cubre más aristas sin cubrir, y repetir.

build/cover_demo, extracto

```
300 grafos al azar, 16 nodos, cada arista con probabilidad 1/4:
  promedio del óptimo: 8.71333
  promedio del matching: 13.18, peor razón 2
  promedio del mayor grado: 9, peor razón 1.25
```

En estos grafos gana con mucha ventaja. Si esto fuera todo lo que sabemos, la elegiríamos sin pensar.

Pero hay una familia de grafos en su contra.

La trampa del mayor grado

k nodos a la izquierda. Para cada i de 2 a k, un grupo de $\lfloor k/i \rfloor$ nodos a la derecha, cada uno unido a i de la izquierda, sin repetir dentro del grupo.

build/cover_demo, extracto

k	nodos	aristas	mayor grado	matching	mayor grado / k
6	14	27	8	12	1.33333
12	35	115	23	24	1.91667
24	84	467	60	48	2.5
48	198	1882	150	96	3.125
96	459	7561	363	192	3.78125
192	1047	30298	855	384	4.45312

La trampa del mayor grado

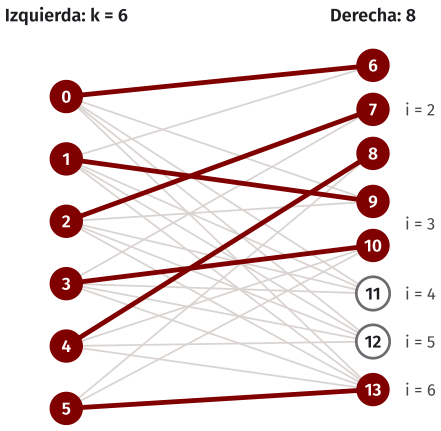
k nodos a la izquierda. Para cada i de 2 a k, un grupo de $\lfloor k/i \rfloor$ nodos a la derecha, cada uno unido a i de la izquierda, sin repetir dentro del grupo.

build/cover_demo, extracto

k	nodos	aristas	mayor grado	matching	mayor grado / k
6	14	27	8	12	1.33333
12	35	115	23	24	1.91667
24	84	467	60	48	2.5
48	198	1882	150	96	3.125
96	459	7561	363	192	3.78125
192	1047	30298	855	384	4.45312

Con $k = 24$ ya pasa de 2, y sigue creciendo. El matching, mientras tanto, da justo $2k$.

La trampa con $k = 6$: seis aristas disjuntas



Matching maximal: |M| = 6
Ninguna arista gruesa comparte un nodo con otra.
Cada una le cuesta un nodo propio a toda cobertura.

OPT = 6
OPT ≥ |M| = 6, por lo anterior.
OPT ≤ 6: la izquierda ya es una cobertura.

Lo que da el matching: |C| = 12 = 2 · 6
Las puntas de cada arista gruesa: los nodos rellenos.
Justo su garantía: el doble del óptimo.

Lo que da el mayor grado: 8
Elige los 8 de la derecha: aquí gana, con 8 / 6 = 1,33.
Con k = 24 ya pierde: da 60, y el matching, 48.

Por qué, y qué dice exactamente

- Con los grupos $2, \ldots, i$ por elegir, un nodo del grupo i tiene grado i, y uno de la izquierda, a lo sumo $i - 1$. La regla elige toda la derecha, sin importar cómo desempate.

Por qué, y qué dice exactamente

- Con los grupos $2, \ldots, i$ por elegir, un nodo del grupo i tiene grado i, y uno de la izquierda, a lo sumo $i - 1$. La regla elige toda la derecha, sin importar cómo desempate.
- La izquierda es una cobertura: OPT $\leq k$.

Por qué, y qué dice exactamente

- Con los grupos $2, \ldots, i$ por elegir, un nodo del grupo i tiene grado i, y uno de la izquierda, a lo sumo $i - 1$. La regla elige toda la derecha, sin importar cómo desempate.
- La izquierda es una cobertura: OPT $\leq k$.
- El matching de la tabla tiene k aristas: OPT $\geq k$.

Por qué, y qué dice exactamente

- Con los grupos $2, \ldots, i$ por elegir, un nodo del grupo i tiene grado i, y uno de la izquierda, a lo sumo $i - 1$. La regla elige toda la derecha, sin importar cómo desempate.
- La izquierda es una cobertura: OPT $\leq k$.
- El matching de la tabla tiene k aristas: OPT $\geq k$.
- Elige $\sum_i \lfloor k/i \rfloor$ nodos, y al dividir entre k, eso crece como $\log k$.

Por qué, y qué dice exactamente

- Con los grupos $2, \ldots, i$ por elegir, un nodo del grupo i tiene grado i, y uno de la izquierda, a lo sumo $i - 1$. La regla elige toda la derecha, sin importar cómo desempate.
- La izquierda es una cobertura: OPT $\leq k$.
- El matching de la tabla tiene k aristas: OPT $\geq k$.
- Elige $\sum_i \lfloor k/i \rfloor$ nodos, y al dividir entre k, eso crece como $\log k$.

Las dos cosas son ciertas

Tiene una garantía $O(\log n)$ (Erickson, J.3), y no tiene un factor constante. Midió mucho mejor que el matching en grafos al azar, y su peor caso garantizado es peor.

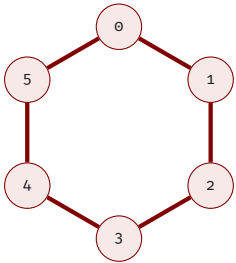
SECCIÓN 05

Viajero métrico: duplicar el árbol

05

La cota inferior: el árbol de expansión mínima

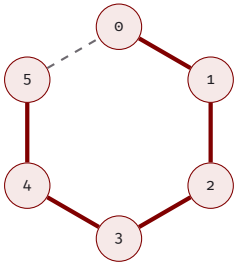
Teorema. Con pesos no negativos, $A \leq \text{OPT}$, donde A es el peso del árbol de expansión mínima.



La cota inferior: el árbol de expansión mínima

Teorema. Con pesos no negativos, $A \leq \text{OPT}$, donde A es el peso del árbol de expansión mínima.

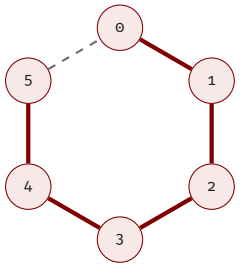
1. Tomen el recorrido óptimo y quítenle una arista.



La cota inferior: el árbol de expansión mínima

Teorema. Con pesos no negativos, $A \leq \text{OPT}$, donde A es el peso del árbol de expansión mínima.

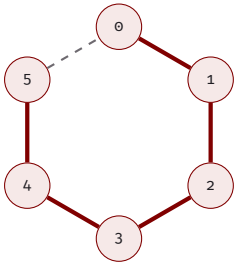
1. Tomen el recorrido óptimo y quítenle una arista.
2. Queda un camino por todos los nodos: un árbol de expansión.



La cota inferior: el árbol de expansión mínima

Teorema. Con pesos no negativos, $A \leq \text{OPT}$, donde A es el peso del árbol de expansión mínima.

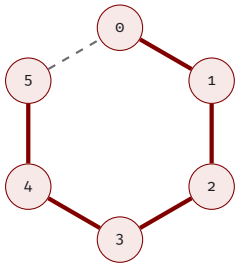
1. Tomen el recorrido óptimo y quítenle una arista.
2. Queda un camino por todos los nodos: un árbol de expansión.
3. Su peso es a lo sumo OPT, porque no hay pesos negativos.



La cota inferior: el árbol de expansión mínima

Teorema. Con pesos no negativos, $A \leq \text{OPT}$, donde A es el peso del árbol de expansión mínima.

1. Tomen el recorrido óptimo y quítenle una arista.
2. Queda un camino por todos los nodos: un árbol de expansión.
3. Su peso es a lo sumo OPT, porque no hay pesos negativos.
4. A es el menor de todos los árboles de expansión. ■

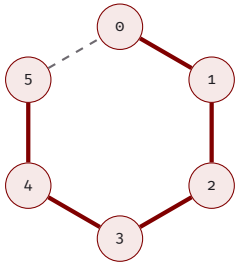


La cota inferior: el árbol de expansión mínima

Teorema. Con pesos no negativos, $A \leq \text{OPT}$, donde A es el peso del árbol de expansión mínima.

1. Tomen el recorrido óptimo y quítenle una arista.
2. Queda un camino por todos los nodos: un árbol de expansión.
3. Su peso es a lo sumo OPT, porque no hay pesos negativos.
4. A es el menor de todos los árboles de expansión. ■

Se calcula con Prim o Kruskal, en tiempo polinomial.



Duplicar el árbol

pseudocódigo

```
algoritmo arbol_duplicado(d, n)
    T ← árbol de expansión mínima de d, con raíz en 0
    caminata ← recorrer T con DFS, escribiendo cada nodo al llegar y al volver a él
    recorrido ← caminata, conservando de cada nodo solo su primera aparición
    devolver recorrido
fin algoritmo
```

Duplicar el árbol

pseudocódigo

```
algoritmo arbol_duplicado(d, n)
    T ← árbol de expansión mínima de d, con raíz en 0
    caminata ← recorrer T con DFS, escribiendo cada nodo al llegar y al volver a él
    recorrido ← caminata, conservando de cada nodo solo su primera aparición
    devolver recorrido
fin algoritmo
```

- La caminata pasa por cada arista dos veces: mide exactamente $2A$. Pero repite nodos.

Duplicar el árbol

pseudocódigo

```
algoritmo arbol_duplicado(d, n)
    T ← árbol de expansión mínima de d, con raíz en o
    caminata ← recorrer T con DFS, escribiendo cada nodo al llegar y al volver a él
    recorrido ← caminata, conservando de cada nodo solo su primera aparición
    devolver recorrido
fin algoritmo
```

- La caminata pasa por cada arista dos veces: mide exactamente $2A$. Pero repite nodos.
- **Atajar:** si va a volver a un nodo ya visitado, salta directo al siguiente nuevo.

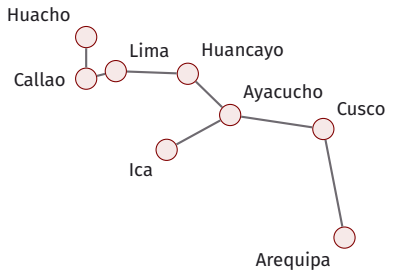
Duplicar el árbol

pseudocódigo

```
algoritmo arbol_duplicado(d, n)
    T ← árbol de expansión mínima de d, con raíz en 0
    caminata ← recorrer T con DFS, escribiendo cada nodo al llegar y al volver a él
    recorrido ← caminata, conservando de cada nodo solo su primera aparición
    devolver recorrido
fin algoritmo
```

- La caminata pasa por cada arista dos veces: mide exactamente $2A$. Pero repite nodos.
- **Atajar:** si va a volver a un nodo ya visitado, salta directo al siguiente nuevo.
- El resultado es el preorden del árbol.

Con ocho capitales

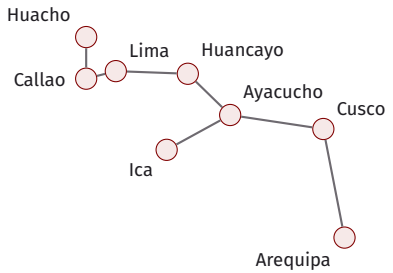


Posiciones aproximadas, con Lima y Callao separadas para verlas.

build/double_tree_demo, extracto

```
peso del árbol: 1249,2 km
Caminata, cada arista dos veces:
  Lima Callao Huacho Callao Lima Huancayo
  Ayacucho Ica Ayacucho Cusco Arequipa
  Cusco Ayacucho Huancayo Lima
largo de la caminata: 2498,4 km
Atajando las repeticiones:
  Lima Callao Huacho Huancayo Ayacucho Ica
  Cusco Arequipa y de vuelta a Lima
largo del recorrido: 2261,4 km
óptimo, con Held-Karp: 1925,9 km
```

Con ocho capitales

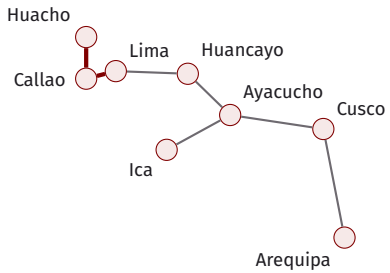


Posiciones aproximadas, con Lima y Callao separadas para verlas.

build/double_tree_demo, extracto

```
peso del árbol: 1249,2 km
Caminata, cada arista dos veces:
  Lima Callao Huacho Callao Lima Huancayo
  Ayacucho Ica Ayacucho Cusco Arequipa
  Cusco Ayacucho Huancayo Lima
largo de la caminata: 2498,4 km
Atajando las repeticiones:
  Lima Callao Huacho Huancayo Ayacucho Ica
  Cusco Arequipa y de vuelta a Lima
largo del recorrido: 2261,4 km
óptimo, con Held-Karp: 1925,9 km
```

Con ocho capitales

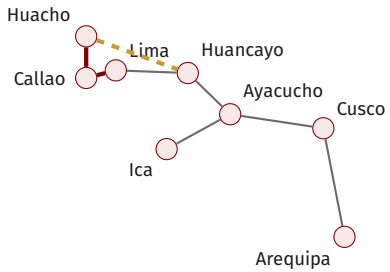


Posiciones aproximadas, con Lima y Callao separadas para verlas.

build/double_tree_demo, extracto

```
peso del árbol: 1249,2 km
Caminata, cada arista dos veces:
  Lima Callao Huacho Callao Lima Huancayo
  Ayacucho Ica Ayacucho Cusco Arequipa
  Cusco Ayacucho Huancayo Lima
largo de la caminata: 2498,4 km
Atajando las repeticiones:
  Lima Callao Huacho Huancayo Ayacucho Ica
  Cusco Arequipa y de vuelta a Lima
largo del recorrido: 2261,4 km
óptimo, con Held-Karp: 1925,9 km
```

Con ocho capitales



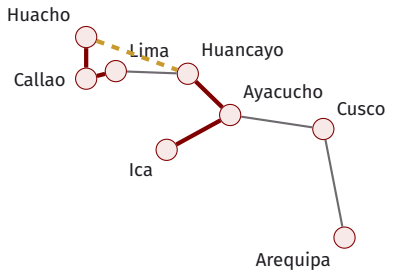
Posiciones aproximadas, con Lima y Callao separadas para verlas.

build/double_tree_demo, extracto

```
peso del árbol: 1249,2 km
Caminata, cada arista dos veces:
  Lima Callao Huacho Callao Lima Huancayo
  Ayacucho Ica Ayacucho Cusco Arequipa
  Cusco Ayacucho Huancayo Lima
largo de la caminata: 2498,4 km
Atajando las repeticiones:
  Lima Callao Huacho Huancayo Ayacucho Ica
  Cusco Arequipa y de vuelta a Lima
largo del recorrido: 2261,4 km
óptimo, con Held-Karp: 1925,9 km
```

Huacho, Callao, Lima, Huancayo se ataja a Huacho → Huancayo: dos veces la desigualdad.

Con ocho capitales

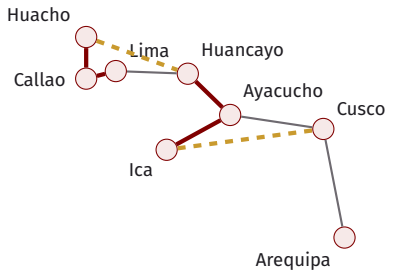


Posiciones aproximadas, con Lima y Callao separadas para verlas.

build/double_tree_demo, extracto

```
peso del árbol: 1249,2 km
Caminata, cada arista dos veces:
  Lima Callao Huacho Callao Lima Huancayo
  Ayacucho Ica Ayacucho Cusco Arequipa
  Cusco Ayacucho Huancayo Lima
largo de la caminata: 2498,4 km
Atajando las repeticiones:
  Lima Callao Huacho Huancayo Ayacucho Ica
  Cusco Arequipa y de vuelta a Lima
largo del recorrido: 2261,4 km
óptimo, con Held-Karp: 1925,9 km
```

Con ocho capitales



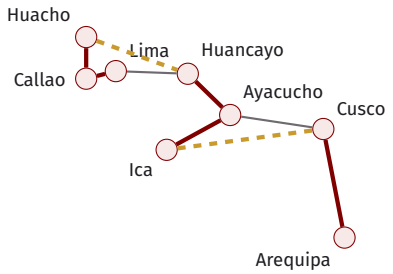
Posiciones aproximadas, con Lima y Callao separadas para verlas.

build/double_tree_demo, extracto

```
peso del árbol: 1249,2 km
Caminata, cada arista dos veces:
  Lima Callao Huacho Callao Lima Huancayo
  Ayacucho Ica Ayacucho Cusco Arequipa
  Cusco Ayacucho Huancayo Lima
largo de la caminata: 2498,4 km
Atajando las repeticiones:
  Lima Callao Huacho Huancayo Ayacucho Ica
  Cusco Arequipa y de vuelta a Lima
largo del recorrido: 2261,4 km
óptimo, con Held-Karp: 1925,9 km
```

Ica, Ayacucho, Cusco se ataja a Ica → Cusco.

Con ocho capitales

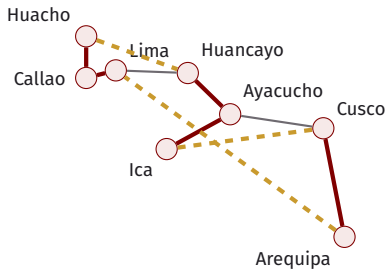


Posiciones aproximadas, con Lima y Callao separadas para verlas.

build/double_tree_demo, extracto

```
peso del árbol: 1249,2 km
Caminata, cada arista dos veces:
  Lima Callao Huacho Callao Lima Huancayo
  Ayacucho Ica Ayacucho Cusco Arequipa
  Cusco Ayacucho Huancayo Lima
largo de la caminata: 2498,4 km
Atajando las repeticiones:
  Lima Callao Huacho Huancayo Ayacucho Ica
  Cusco Arequipa y de vuelta a Lima
largo del recorrido: 2261,4 km
óptimo, con Held-Karp: 1925,9 km
```

Con ocho capitales



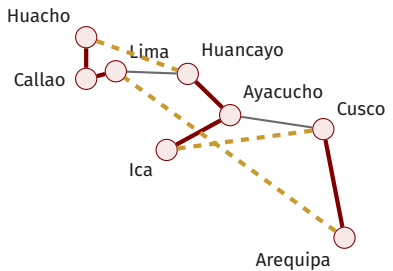
Posiciones aproximadas, con Lima y Callao separadas para verlas.

build/double_tree_demo, extracto

```
peso del árbol: 1249,2 km
Caminata, cada arista dos veces:
  Lima Callao Huacho Callao Lima Huancayo
  Ayacucho Ica Ayacucho Cusco Arequipa
  Cusco Ayacucho Huancayo Lima
largo de la caminata: 2498,4 km
Atajando las repeticiones:
  Lima Callao Huacho Huancayo Ayacucho Ica
  Cusco Arequipa y de vuelta a Lima
largo del recorrido: 2261,4 km
óptimo, con Held-Karp: 1925,9 km
```

Arequipa, Cusco, Ayacucho, Huancayo, Lima: tres veces la desigualdad.

Con ocho capitales



Posiciones aproximadas, con Lima y Callao separadas para verlas.

build/double_tree_demo, extracto

```
peso del árbol: 1249,2 km
Caminata, cada arista dos veces:
  Lima Callao Huacho Callao Lima Huancayo
  Ayacucho Ica Ayacucho Cusco Arequipa
  Cusco Ayacucho Huancayo Lima
largo de la caminata: 2498,4 km
Atajando las repeticiones:
  Lima Callao Huacho Huancayo Ayacucho Ica
  Cusco Arequipa y de vuelta a Lima
largo del recorrido: 2261,4 km
óptimo, con Held-Karp: 1925,9 km
```

$1925{,}9 \leq 2261{,}4 \leq 2498{,}4 = 2A.$

Factor 2 en el viajero métrico

Teorema. Con distancias simétricas, no negativas y con la desigualdad triangular, el recorrido mide a lo sumo $2 \cdot$ OPT.

Factor 2 en el viajero métrico

Teorema. Con distancias simétricas, no negativas y con la desigualdad triangular, el recorrido mide a lo sumo $2 \cdot$ OPT.

1. **Es un recorrido.** El árbol contiene a todos, y se conserva una aparición de cada uno.

Factor 2 en el viajero métrico

Teorema. Con distancias simétricas, no negativas y con la desigualdad triangular, el recorrido mide a lo sumo $2 \cdot$ OPT.

1. **Es un recorrido.** El árbol contiene a todos, y se conserva una aparición de cada uno.
2. **La caminata mide** $2A$**.** Cada arista dos veces. Por la simetría, las dos cuestan lo mismo.

Factor 2 en el viajero métrico

Teorema. Con distancias simétricas, no negativas y con la desigualdad triangular, el recorrido mide a lo sumo $2 \cdot$ OPT.

1. **Es un recorrido.** El árbol contiene a todos, y se conserva una aparición de cada uno.
2. **La caminata mide** $2A$**.** Cada arista dos veces. Por la simetría, las dos cuestan lo mismo.
3. **Atajar no alarga.** Entre dos nodos consecutivos x, y del recorrido, la caminata va por $x, z_1, \ldots, z_k, y$. Con la desigualdad, k veces:

$$d(x, y) \leq d(x, z_1) + d(z_1, z_2) + \cdots + d(z_k, y)$$

Factor 2 en el viajero métrico

Teorema. Con distancias simétricas, no negativas y con la desigualdad triangular, el recorrido mide a lo sumo $2 \cdot$ OPT.

1. **Es un recorrido.** El árbol contiene a todos, y se conserva una aparición de cada uno.
2. **La caminata mide** $2A$**.** Cada arista dos veces. Por la simetría, las dos cuestan lo mismo.
3. **Atajar no alarga.** Entre dos nodos consecutivos x, y del recorrido, la caminata va por $x, z_1, \ldots, z_k, y$. Con la desigualdad, k veces:

$$d(x, y) \leq d(x, z_1) + d(z_1, z_2) + \cdots + d(z_k, y)$$

4. **Conclusión.** Recorrido $\leq 2A \leq 2 \cdot$ OPT. ■

Dónde se usa cada hipótesis

La desigualdad triangular se usa una sola vez: al atajar

$A \leq \text{OPT}$ no la necesita, y la caminata de $2A$ tampoco. Sin ella, una caminata barata con repetidos no tiene por qué seguir siendo barata sin ellos.

Dónde se usa cada hipótesis

La desigualdad triangular se usa una sola vez: al atajar

$A \leq \text{OPT}$ no la necesita, y la caminata de $2A$ tampoco. Sin ella, una caminata barata con repetidos no tiene por qué seguir siendo barata sin ellos.

Error común

Omitir el atajo y entregar la caminata. Mide exactamente $2A$, y no es una solución: repite ciudades. El verificador del laboratorio la rechaza.

Dónde se usa cada hipótesis

La desigualdad triangular se usa una sola vez: al atajar

$A \leq \text{OPT}$ no la necesita, y la caminata de $2A$ tampoco. Sin ella, una caminata barata con repetidos no tiene por qué seguir siendo barata sin ellos.

Error común

Omitir el atajo y entregar la caminata. Mide exactamente $2A$, y no es una solución: repite ciudades. El verificador del laboratorio la rechaza.

Se conoce algo mejor: Christofides, de 1976, tiene factor 3/2 en el caso métrico. No lo vemos: aquí interesa la forma del argumento.

El factor, comprobado con las capitales

n	A (km)	Duplicado (km)	$2A$ (km)	Óptimo (km)	Duplicado / óptimo
10	1918,2	3386,4	3836,4	3167,2	1,069
15	2175,8	3359,2	4351,7	3225,0	1,042
20	3080,2	5142,1	6160,4	4640,2	1,108
26	4187,4	6762,8	8374,9	5621,6	1,203

El factor, comprobado con las capitales

n	A (km)	Duplicado (km)	$2A$ (km)	Óptimo (km)	Duplicado / óptimo
10	1918,2	3386,4	3836,4	3167,2	1,069
15	2175,8	3359,2	4351,7	3225,0	1,042
20	3080,2	5142,1	6160,4	4640,2	1,108
26	4187,4	6762,8	8374,9	5621,6	1,203

Las pruebas lo comprueban paso a paso en 12 subconjuntos y 300 instancias métricas al azar.

El factor, comprobado con las capitales

n	A (km)	Duplicado (km)	$2A$ (km)	Óptimo (km)	Duplicado / óptimo
10	1918,2	3386,4	3836,4	3167,2	1,069
15	2175,8	3359,2	4351,7	3225,0	1,042
20	3080,2	5142,1	6160,4	4640,2	1,108
26	4187,4	6762,8	8374,9	5621,6	1,203

Las pruebas lo comprueban paso a paso en 12 subconjuntos y 300 instancias métricas al azar.

Medición y garantía

1,203 es lo que midió *esta* instancia. 2 es lo que vale *siempre*.

SECCIÓN 06

Sin desigualdad triangular

06

Sin ella, ningún factor constante

Teorema. Si existe un algoritmo polinomial con factor constante α para el viajero general, entonces P = NP.

Se reduce el **ciclo hamiltoniano**, que se sabe NP-completo. No lo demostramos aquí.

La construcción: G **con** $n \geq 3$ **nodos,** $W = \lceil \alpha n \rceil + 1$

peso$(u, v) = 1$ si $\{u, v\}$ es arista de G. peso$(u, v) = W$ si no lo es.

Sin ella, ningún factor constante

Teorema. Si existe un algoritmo polinomial con factor constante α para el viajero general, entonces P = NP.

Se reduce el **ciclo hamiltoniano**, que se sabe NP-completo. No lo demostramos aquí.

La construcción: G con $n \geq 3$ nodos, $W = \lceil \alpha n \rceil + 1$

peso$(u, v) = 1$ si $\{u, v\}$ es arista de G. peso$(u, v) = W$ si no lo es.

W tiene $O(\log n)$ bits y hay $O(n^2)$ pesos: es polinomial. Se responde "sí hay ciclo" si y solo si el recorrido mide a lo sumo αn.

Las dos direcciones

- **Si G tiene ciclo hamiltoniano.** Ese ciclo mide n, y OPT $= n$. El algoritmo devuelve a lo sumo $\alpha \cdot$ OPT $= \alpha n$. Responde sí.

Las dos direcciones

- **Si G tiene ciclo hamiltoniano.** Ese ciclo mide n, y OPT $= n$. El algoritmo devuelve a lo sumo $\alpha \cdot$ OPT $= \alpha n$. Responde sí.
- **Si no lo tiene.** Todo recorrido usa un par que no es arista: si no, sería un ciclo hamiltoniano. Mide al menos $W + (n - 1) > \alpha n$. Responde no.

Las dos direcciones

- **Si G tiene ciclo hamiltoniano.** Ese ciclo mide n, y OPT $= n$. El algoritmo devuelve a lo sumo $\alpha \cdot$ OPT $= \alpha n$. Responde sí.
- **Si no lo tiene.** Todo recorrido usa un par que no es arista: si no, sería un ciclo hamiltoniano. Mide al menos $W + (n - 1) > \alpha n$. Responde no.
- Comparar con αn decide un problema NP-completo en tiempo polinomial. ■

Las dos direcciones

- **Si G tiene ciclo hamiltoniano.** Ese ciclo mide n, y OPT $= n$. El algoritmo devuelve a lo sumo $\alpha \cdot$ OPT $= \alpha n$. Responde sí.
- **Si no lo tiene.** Todo recorrido usa un par que no es arista: si no, sería un ciclo hamiltoniano. Mide al menos $W + (n - 1) > \alpha n$. Responde no.
- Comparar con αn decide un problema NP-completo en tiempo polinomial. ■

¿Por qué no contradice el factor 2?

Estos pesos no cumplen la desigualdad: $1 + 1 < W$. Esa libertad es justo la que hace que un solo par malo arruine un recorrido.

Corriendo la reducción, con $\alpha = 2$

build/hamilton_demo

```
Un ciclo de 6 nodos con la numeración desordenada: sí hay ciclo hamiltoniano
  n = 6, W = 13, alfa·n = 12
  óptimo: 6 <= 12
  árbol duplicado: 0 3 1 5 4 2, largo 30

Dos nodos unidos a otros cuatro, K(2,4): no hay ciclo hamiltoniano
  n = 6, W = 13, alfa·n = 12
  óptimo: 30 > 12
  árbol duplicado: 0 2 1 3 4 5, largo 30
```

Corriendo la reducción, con $\alpha = 2$

build/hamilton_demo

```
Un ciclo de 6 nodos con la numeración desordenada: sí hay ciclo hamiltoniano
  n = 6, W = 13, alfa·n = 12
  óptimo: 6 <= 12
  árbol duplicado: 0 3 1 5 4 2, largo 30

Dos nodos unidos a otros cuatro, K(2,4): no hay ciclo hamiltoniano
  n = 6, W = 13, alfa·n = 12
  óptimo: 30 > 12
  árbol duplicado: 0 2 1 3 4 5, largo 30
```

En el primero, el árbol duplicado da cinco veces el óptimo. Aquí no tiene ninguna garantía.

Corriendo la reducción, con $\alpha = 2$

build/hamilton_demo

```
Un ciclo de 6 nodos con la numeración desordenada: sí hay ciclo hamiltoniano
  n = 6, W = 13, alfa·n = 12
  óptimo: 6 <= 12
  árbol duplicado: 0 3 1 5 4 2, largo 30

Dos nodos unidos a otros cuatro, K(2,4): no hay ciclo hamiltoniano
  n = 6, W = 13, alfa·n = 12
  óptimo: 30 > 12
  árbol duplicado: 0 2 1 3 4 5, largo 30
```

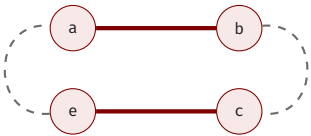
En el primero, el árbol duplicado da cinco veces el óptimo. Aquí no tiene ninguna garantía. NP-difícil no predice cuánto se aproxima: viajero métrico, factor 2; viajero general, ningún factor constante, salvo que P = NP.

SECCIÓN 07

Heurísticas: 2-opt

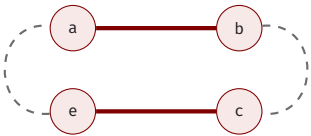
07

El movimiento

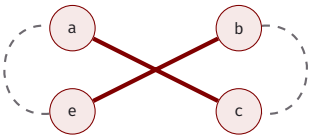


antes

El movimiento

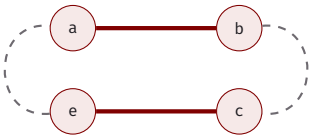


antes

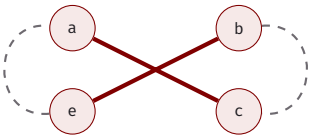


después: el tramo de b a c, al revés

El movimiento



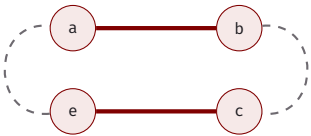
antes



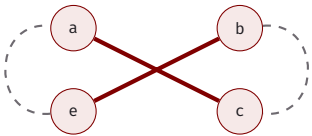
después: el tramo de b a c, al revés

Quitar dos aristas sin nodos en común parte el recorrido en dos caminos, y hay una sola forma distinta de volver a unirlos en un recorrido. La otra, *a* con *e* y *b* con *c*, deja dos ciclos.

El movimiento



antes



después: el tramo de b a c, al revés

Quitar dos aristas sin nodos en común parte el recorrido en dos caminos, y hay una sola forma distinta de volver a unirlos en un recorrido. La otra, a con e y b con c, deja dos ciclos.

$$\text{ganancia} = d(a,b) + d(c,e) - d(a,c) - d(b,e)$$

2-opt

pseudocódigo

```
algoritmo dos_opt(d, t, n)
    mejoro ← verdadero
    mientras mejoro hacer
        mejoro ← falso
        para i desde 1 hasta n - 2 hacer
            para j desde i + 1 hasta n - 1 hacer
                a, b, c, e ← t[i - 1], t[i], t[j], t[(j + 1) mod n]
                si d(a, c) + d(b, e) < d(a, b) + d(c, e) entonces
                    invertir t[i] a t[j]
                    mejoro ← verdadero
                fin si
            fin para
        fin para
    fin mientras
    devolver t
fin algoritmo
```

Termina, y no promete más

Teorema. Con distancias enteras no negativas, termina en un **óptimo local**: ningún cambio de dos aristas lo acorta.

Termina, y no promete más

Teorema. Con distancias enteras no negativas, termina en un **óptimo local**: ningún cambio de dos aristas lo acorta.

1. Cada cambio aplicado baja el largo en al menos 1, porque es entero.

Termina, y no promete más

Teorema. Con distancias enteras no negativas, termina en un **óptimo local**: ningún cambio de dos aristas lo acorta.

1. Cada cambio aplicado baja el largo en al menos 1, porque es entero.
2. El largo es un entero no negativo: solo puede bajar una cantidad finita de veces.

Termina, y no promete más

Teorema. Con distancias enteras no negativas, termina en un **óptimo local**: ningún cambio de dos aristas lo acorta.

1. Cada cambio aplicado baja el largo en al menos 1, porque es entero.
2. El largo es un entero no negativo: solo puede bajar una cantidad finita de veces.
3. Al terminar, una pasada completa no aplicó nada: ningún par da ganancia positiva. ■

Termina, y no promete más

Teorema. Con distancias enteras no negativas, termina en un **óptimo local**: ningún cambio de dos aristas lo acorta.

1. Cada cambio aplicado baja el largo en al menos 1, porque es entero.
2. El largo es un entero no negativo: solo puede bajar una cantidad finita de veces.
3. Al terminar, una pasada completa no aplicó nada: ningún par da ganancia positiva. ■

No dice que sean pocos cambios, ni que el resultado sea el óptimo.

Termina, y no promete más

Teorema. Con distancias enteras no negativas, termina en un **óptimo local**: ningún cambio de dos aristas lo acorta.

1. Cada cambio aplicado baja el largo en al menos 1, porque es entero.
2. El largo es un entero no negativo: solo puede bajar una cantidad finita de veces.
3. Al terminar, una pasada completa no aplicó nada: ningún par da ganancia positiva. ■

No dice que sean pocos cambios, ni que el resultado sea el óptimo.

> **Por qué metros enteros**
>
> Con `double`, una ganancia de 10^{-12} puede ser ruido de redondeo. Con enteros, $<$ significa lo que dice.

Heurística no es aproximación

Si empieza en el árbol duplicado, nunca empeora, así que mide a lo sumo $2 \cdot$ OPT. La garantía viene del **punto de partida**, no de 2-opt.

Heurística no es aproximación

Si empieza en el árbol duplicado, nunca empeora, así que mide a lo sumo $2 \cdot$ OPT. La garantía viene del **punto de partida**, no de 2-opt.

Error común

Decir “aproximación” para una heurística que no tiene garantía demostrada.

Heurística no es aproximación

Si empieza en el árbol duplicado, nunca empeora, así que mide a lo sumo $2 \cdot \mathrm{OPT}$. La garantía viene del **punto de partida**, no de 2-opt.

Error común

Decir “aproximación” para una heurística que no tiene garantía demostrada.

- “2-opt es una aproximación”: **falso**, tal como está dicho.

Heurística no es aproximación

Si empieza en el árbol duplicado, nunca empeora, así que mide a lo sumo $2 \cdot$ OPT. La garantía viene del **punto de partida**, no de 2-opt.

Error común

Decir “aproximación” para una heurística que no tiene garantía demostrada.

- “2-opt es una aproximación”: **falso**, tal como está dicho.
- “2-opt me dio 1,05 veces el óptimo en mis datos”: una **medición** de esos datos.

Heurística no es aproximación

Si empieza en el árbol duplicado, nunca empeora, así que mide a lo sumo $2 \cdot \mathrm{OPT}$. La garantía viene del **punto de partida**, no de 2-opt.

Error común

Decir “aproximación” para una heurística que no tiene garantía demostrada.

- “2-opt es una aproximación”: **falso**, tal como está dicho.
- “2-opt me dio 1,05 veces el óptimo en mis datos”: una **medición** de esos datos.
- “Desde el árbol duplicado, a lo sumo $2 \cdot \mathrm{OPT}$”: un **teorema**, por el punto de partida.

Paso a paso, con 20 ciudades

build/two_opt_demo, extracto

```
Las primeras 20 filas. Árbol duplicado, 5142.1 km:
  invertir de la posición 2 a la 3: 5133.0 km
  invertir de la posición 4 a la 19: 5033.3 km
  invertir de la posición 8 a la 9: 4867.3 km
óptimo, con Held-Karp: 4640.2 km

2-opt desde 2000 recorridos al azar, las mismas 20 filas:
  mejor 4640.2 km, mediana 4640.2 km, peor 4882.0 km
  llegaron al óptimo: 1215 de 2000
  peor largo / óptimo: 1.052
```

Paso a paso, con 20 ciudades

build/two_opt_demo, extracto

```
Las primeras 20 filas. Árbol duplicado, 5142.1 km:
  invertir de la posición 2 a la 3: 5133.0 km
  invertir de la posición 4 a la 19: 5033.3 km
  invertir de la posición 8 a la 9: 4867.3 km
óptimo, con Held-Karp: 4640.2 km

2-opt desde 2000 recorridos al azar, las mismas 20 filas:
  mejor 4640.2 km, mediana 4640.2 km, peor 4882.0 km
  llegaron al óptimo: 1215 de 2000
  peor largo / óptimo: 1.052
```

- Tres cambios, y queda en 4867,3 km: **un óptimo local, no el óptimo.**

Paso a paso, con 20 ciudades

build/two_opt_demo, extracto

```
Las primeras 20 filas. Árbol duplicado, 5142.1 km:
  invertir de la posición 2 a la 3: 5133.0 km
  invertir de la posición 4 a la 19: 5033.3 km
  invertir de la posición 8 a la 9: 4867.3 km
óptimo, con Held-Karp: 4640.2 km

2-opt desde 2000 recorridos al azar, las mismas 20 filas:
  mejor 4640.2 km, mediana 4640.2 km, peor 4882.0 km
  llegaron al óptimo: 1215 de 2000
  peor largo / óptimo: 1.052
```

- Tres cambios, y queda en 4867,3 km: **un óptimo local, no el óptimo.**
- Dónde termina depende de dónde empieza. Es una medición de estas 20 ciudades: no se extiende a un caso nuevo.

Los tres cambios, sobre las capitales



Posición y ciudad

0	Abancay
1	Cusco
2	Arequipa
3	Moquegua
4	Ayacucho
5	Huancavelica
6	Huancayo
7	Cerro de Pasco
8	Huánuco
9	Huaraz
10	Cajamarca
11	Chachapoyas
12	Moyobamba
13	Iquitos
14	Chiclayo
15	Piura
16	Huacho
17	Callao
18	Lima
19	Ica

Largo

5142,1 km

Las primeras 20 filas. El recorrido del árbol duplicado mide 5142,1 km.

Proyección esquemática de longitud y latitud, no un mapa. Los largos son de gran círculo.

Los tres cambios, sobre las capitales



Posición y ciudad

0	Abancay
1	**Cusco**
2	**Moquegua**
3	**Arequipa**
4	**Ayacucho**
5	Huancavelica
6	Huancayo
7	Cerro de Pasco
8	Huánuco
9	Huaraz
10	Cajamarca
11	Chachapoyas
12	Moyobamba
13	Iquitos
14	Chiclayo
15	Piura
16	Huacho
17	Callao
18	Lima
19	Ica

Largo

5133,0 km

invertir de la posición 2 a la 3

a = Cusco
b = Arequipa
c = Moquegua
e = Ayacucho

salen: a-b, c-e
entran: a-c, b-e
el tramo: mide lo mismo

Cusco, Arequipa, Moquegua, Ayacucho pasa a Cusco, Moquegua, Arequipa, Ayacucho: baja 9,1 km.

Proyección esquemática de longitud y latitud, no un mapa. Los largos son de gran círculo.

Los tres cambios, sobre las capitales



Posición y ciudad

0	**Abancay**
1	Cusco
2	Moquegua
3	**Arequipa**
4	**Ica**
5	Lima
6	Callao
7	Huacho
8	Piura
9	Chiclayo
10	Iquitos
11	Moyobamba
12	Chachapoyas
13	Cajamarca
14	Huaraz
15	Huánuco
16	Cerro de Pasco
17	Huancayo
18	Huancavelica
19	**Ayacucho**

Largo

5033,3 km

invertir de la posición 4 a la 19

a = Arequipa
b = Ayacucho
c = Ica
e = Abancay

salen: a-b, c-e
entran: a-c, b-e
el tramo: mide lo mismo

Se invierten 16 ciudades, y solo cambian dos aristas. Baja a 5033,3 km.

Proyección esquemática de longitud y latitud, no un mapa. Los largos son de gran círculo.

Los tres cambios, sobre las capitales



Posición y ciudad

0	Abancay
1	Cusco
2	Moquegua
3	Arequipa
4	Ica
5	Lima
6	Callao
7	**Huacho**
8	**Chiclayo**
9	**Piura**
10	**Iquitos**
11	Moyobamba
12	Chachapoyas
13	Cajamarca
14	Huaraz
15	Huánuco
16	Cerro de Pasco
17	Huancayo
18	Huancavelica
19	Ayacucho

Largo

4867,3 km

invertir de la posición 8 a la 9

a = Huacho
b = Piura
c = Chiclayo
e = Iquitos

salen: a-b, c-e
entran: a-c, b-e
el tramo: mide lo mismo

Tercer cambio: baja a 4867,3 km. Ya ningún cambio de dos aristas lo acorta.

Proyección esquemática de longitud y latitud, no un mapa. Los largos son de gran círculo.

Los tres cambios, sobre las capitales



Posición y ciudad

0	Abancay
1	Cusco
2	Moquegua
3	Arequipa
4	Ica
5	Lima
6	Callao
7	Huacho
8	Chiclayo
9	Piura
10	Iquitos
11	Moyobamba
12	Chachapoyas
13	Cajamarca
14	Huaraz
15	Huánuco
16	Cerro de Pasco
17	Huancayo
18	Huancavelica
19	Ayacucho

Largo

4867,3 km

Óptimo: 4640,2 km

aristas de un recorrido óptimo que este no tiene

Es un óptimo local, y no es el óptimo: 4867,3 km frente a 4640,2 km.

Proyección esquemática de longitud y latitud, no un mapa. Los largos son de gran círculo.

El 2-opt de los ejemplos

tour.h, two_opt_plain

```
for (int i = 1; i + 1 < n; i++) {
    for (int j = i + 1; j < n; j++) {
        std::reverse(t.begin() + i, t.begin() + j + 1);
        const long long candidate = tour_length(d, t);
        if (candidate < current) {
            current = candidate;
            moves.push_back({i, j, candidate});
            improved = true;
        } else {
            std::reverse(t.begin() + i, t.begin() + j + 1);
        }
    }
}
```

El 2-opt de los ejemplos

tour.h, two_opt_plain

```cpp
for (int i = 1; i + 1 < n; i++) {
    for (int j = i + 1; j < n; j++) {
        std::reverse(t.begin() + i, t.begin() + j + 1);
        const long long candidate = tour_length(d, t);
        if (candidate < current) {
            current = candidate;
            moves.push_back({i, j, candidate});
            improved = true;
        } else {
            std::reverse(t.begin() + i, t.begin() + j + 1);
        }
    }
}
```

Mide todo en cada intento: $O(n)$, no el $O(1)$ del pseudocódigo. Con 26 ciudades no se nota. Con mil lugares, sí.

SECCIÓN 08

La comparación completa

08

Largo, con las primeras n filas

Método	**Largo** $n = 10$	$n = 15$	$n = 20$	$n = 26$	
Árbol, A (no es un recorrido)	1918,2	2175,8	3080,2	4187,4	km
Árbol duplicado	3386,4	3359,2	5142,1	6762,8	km
Duplicado + 2-opt	3178,9	3259,5	4867,3	5719,3	km
Vecino más cercano	3513,3	4736,1	6218,7	6799,2	km
Vecino + 2-opt	3178,9	3225,0	4827,8	6344,0	km
Óptimo	3167,2	3225,0	4640,2	5621,6	km

Largo, con las primeras n filas

Método	**Largo** $n = 10$	$n = 15$	$n = 20$	$n = 26$	
Árbol, A (no es un recorrido)	1918,2	2175,8	3080,2	4187,4	km
Árbol duplicado	3386,4	3359,2	5142,1	6762,8	km
Duplicado + 2-opt	3178,9	3259,5	4867,3	5719,3	km
Vecino más cercano	3513,3	4736,1	6218,7	6799,2	km
Vecino + 2-opt	3178,9	3225,0	4827,8	6344,0	km
Óptimo	3167,2	3225,0	4640,2	5621,6	km

Razón al óptimo: duplicado, de 1,042 a 1,203. Duplicado + 2-opt, de 1,004 a 1,049. Vecino, de 1,109 a 1,469.

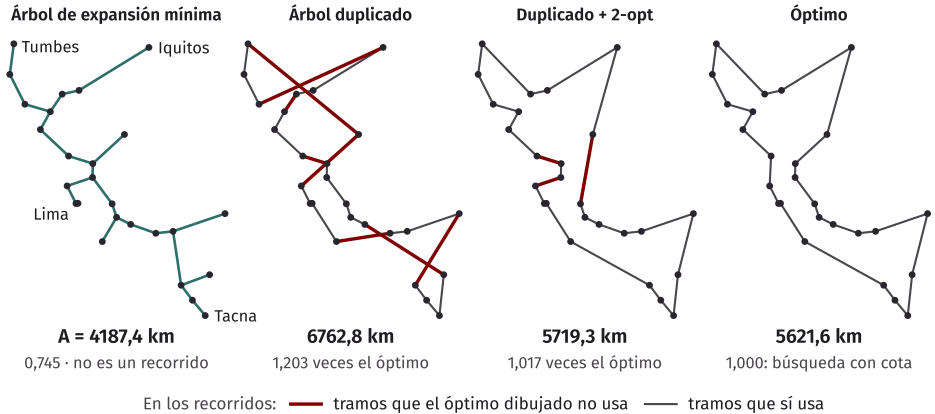
Largo, con las primeras n filas

Método	**Largo** $n = 10$	$n = 15$	$n = 20$	$n = 26$	
Árbol, A (no es un recorrido)	1918,2	2175,8	3080,2	4187,4	km
Árbol duplicado	3386,4	3359,2	5142,1	6762,8	km
Duplicado + 2-opt	3178,9	3259,5	4867,3	5719,3	km
Vecino más cercano	3513,3	4736,1	6218,7	6799,2	km
Vecino + 2-opt	3178,9	3225,0	4827,8	6344,0	km
Óptimo	3167,2	3225,0	4640,2	5621,6	km

Razón al óptimo: duplicado, de 1,042 a 1,203. Duplicado + 2-opt, de 1,004 a 1,049. Vecino, de 1,109 a 1,469.

No hay un ganador entre los dos arranques: con 15 y 20, mejor el vecino; con 26, mejor el árbol, 1,017 frente a 1,128. Son mediciones de cuatro instancias, no una ley.

Las 26 capitales, dibujadas



Proyección esquemática de longitud y latitud, no un mapa. Los largos son de gran círculo.

Tiempo: los exactos

Método	$n = 10$	$n = 15$	$n = 20$	$n = 26$
Árbol, atajos, 2-opt, vecino	< 1 ms	< 1 ms	< 1 ms	< 1 ms
Held-Karp	< 1 ms	0,008 s	0,466 s	no se corrió
Búsqueda con cota	0,001 s	0,023 s	7,708 s	23,516 s

`make bench`, -O2, en el entorno del curso, en una máquina compartida. Menor de tres corridas, salvo la búsqueda con 20 y 26, que corrió una vez. Otra corrida de la misma búsqueda, en la misma medición, tardó 4,93 s y 14,91 s. Las cuentas de nodos no cambian, y los tiempos sí.

Tiempo: los exactos

Método	$n = 10$	$n = 15$	$n = 20$	$n = 26$
Árbol, atajos, 2-opt, vecino	< 1 ms	< 1 ms	< 1 ms	< 1 ms
Held-Karp	< 1 ms	0,008 s	0,466 s	no se corrió
Búsqueda con cota	0,001 s	0,023 s	7,708 s	23,516 s

`make bench`, -O2, en el entorno del curso, en una máquina compartida. Menor de tres corridas, salvo la búsqueda con 20 y 26, que corrió una vez. Otra corrida de la misma búsqueda, en la misma medición, tardó 4,93 s y 14,91 s. Las cuentas de nodos no cambian, y los tiempos sí.

Held-Karp no se corrió con 26: su tabla necesitaría unos 6,7 GB.

Tiempo: los exactos

Método	$n = 10$	$n = 15$	$n = 20$	$n = 26$
Árbol, atajos, 2-opt, vecino	< 1 ms	< 1 ms	< 1 ms	< 1 ms
Held-Karp	< 1 ms	0,008 s	0,466 s	no se corrió
Búsqueda con cota	0,001 s	0,023 s	7,708 s	23,516 s

`make bench`, -O2, en el entorno del curso, en una máquina compartida. Menor de tres corridas, salvo la búsqueda con 20 y 26, que corrió una vez. Otra corrida de la misma búsqueda, en la misma medición, tardó 4,93 s y 14,91 s. Las cuentas de nodos no cambian, y los tiempos sí.

Held-Karp no se corrió con 26: su tabla necesitaría unos 6,7 GB.

Se combinan

2-opt desde varios puntos, en milisegundos, y ese largo como `mejor` inicial de una búsqueda exacta. Si no termina a tiempo, ya hay un recorrido.

Cuánta memoria toma Held-Karp

build/memory_bench: el pico del proceso, ru_maxrss

```
tabla para n = 20                                  la cuenta         pico
2^19 x 20, int                                      40.0 MiB     43.1 MiB
2^19 x 20, long long                                80.0 MiB     83.1 MiB
2^20 x 20, int                                      80.0 MiB     83.1 MiB
2^20 x 20, long long                               160.0 MiB    163.1 MiB
2^20 filas de 20, vector de vectores, long long    160.0 MiB    203.0 MiB
```

Cuánta memoria toma Held-Karp

build/memory_bench: el pico del proceso, ru_maxrss

```
tabla para n = 20                                  la cuenta         pico
2^19 x 20, int                                      40.0 MiB     43.1 MiB
2^19 x 20, long long                                80.0 MiB     83.1 MiB
2^20 x 20, int                                      80.0 MiB     83.1 MiB
2^20 x 20, long long                               160.0 MiB    163.1 MiB
2^20 filas de 20, vector de vectores, long long    160.0 MiB    203.0 MiB
```

- Fijar la ciudad 0 baja las filas de 2^{20} a 2^{19}.

Cuánta memoria toma Held-Karp

build/memory_bench: el pico del proceso, ru_maxrss

```
tabla para n = 20                                  la cuenta         pico
2^19 x 20, int                                      40.0 MiB     43.1 MiB
2^19 x 20, long long                                80.0 MiB     83.1 MiB
2^20 x 20, int                                      80.0 MiB     83.1 MiB
2^20 x 20, long long                               160.0 MiB    163.1 MiB
2^20 filas de 20, vector de vectores, long long    160.0 MiB    203.0 MiB
```

- Fijar la ciudad 0 baja las filas de 2^{20} a 2^{19}.
- 20 tramos no pasan de unos 4×10^8 m: cabe en `int`.

Cuánta memoria toma Held-Karp

build/memory_bench: el pico del proceso, ru_maxrss

```
tabla para n = 20                                  la cuenta         pico
2^19 x 20, int                                      40.0 MiB     43.1 MiB
2^19 x 20, long long                                80.0 MiB     83.1 MiB
2^20 x 20, int                                      80.0 MiB     83.1 MiB
2^20 x 20, long long                               160.0 MiB    163.1 MiB
2^20 filas de 20, vector de vectores, long long    160.0 MiB    203.0 MiB
```

- Fijar la ciudad 0 baja las filas de 2^{20} a 2^{19}.
- 20 tramos no pasan de unos 4×10^8 m: cabe en `int`.
- Una matriz contigua no paga 40 MiB de un millón de vectores pequeños.

Cuánta memoria toma Held-Karp

build/memory_bench: el pico del proceso, ru_maxrss

tabla para n = 20	la cuenta	pico
2^19 x 20, int	40.0 MiB	43.1 MiB
2^19 x 20, long long	80.0 MiB	83.1 MiB
2^20 x 20, int	80.0 MiB	83.1 MiB
2^20 x 20, long long	160.0 MiB	163.1 MiB
2^20 filas de 20, vector de vectores, long long	160.0 MiB	203.0 MiB

- Fijar la ciudad 0 baja las filas de 2^{20} a 2^{19}.
- 20 tramos no pasan de unos 4×10^8 m: cabe en `int`.
- Una matriz contigua no paga 40 MiB de un millón de vectores pequeños.

Límite del laboratorio: 256 MiB. El juez no lo mide; el enunciado dice cómo comprobarlo.

SECCIÓN 09

Programación lineal como idea

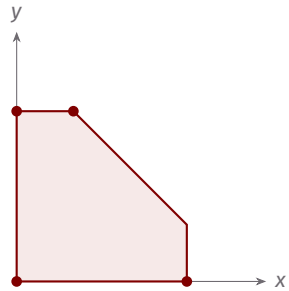
09

[se puede cortar] Programación lineal

Programación lineal ***(linear programming)***

Valores reales para unas variables, que optimicen una función lineal con restricciones lineales.

máx $3x + 2y$ con $x + y \leq 4$, $x \leq 3$, $y \leq 3$, $x, y \geq 0$.



Hecho a mano. En el entorno no hay un solucionador.

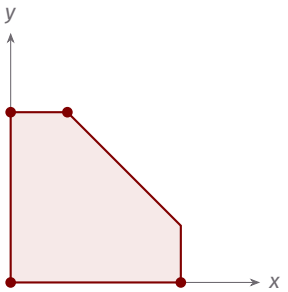
[se puede cortar] Programación lineal

Programación lineal *(linear programming)*

Valores reales para unas variables, que optimicen una función lineal con restricciones lineales.

máx $3x + 2y$ con $x + y \leq 4$, $x \leq 3$, $y \leq 3$, $x, y \geq 0$.

Cada restricción deja afuera un semiplano. El máximo se alcanza en una esquina.

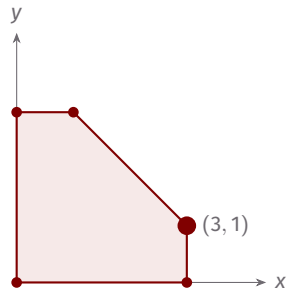


Hecho a mano. En el entorno no hay un solucionador.

[se puede cortar] Programación lineal

Programación lineal *(linear programming)*

Valores reales para unas variables, que optimicen una función lineal con restricciones lineales.



máx $3x + 2y$ con $x + y \leq 4$, $x \leq 3$, $y \leq 3$, $x, y \geq 0$.

Cada restricción deja afuera un semiplano. El máximo se alcanza en una esquina.

Valores: 0, 9, **11**, 9, 6. El óptimo es 11, en $(3, 1)$.

Hecho a mano. En el entorno no hay un solucionador.

[se puede cortar] Se resuelve en tiempo polinomial

- **Elipsoide** (Khachiyan, 1979) y **puntos interiores** (Karmarkar, 1984): polinomiales en los bits de la entrada.

[se puede cortar] Se resuelve en tiempo polinomial

- **Elipsoide** (Khachiyan, 1979) y **puntos interiores** (Karmarkar, 1984): polinomiales en los bits de la entrada.
- **Símplex** (Dantzig): camina de esquina en esquina. Muy usado y muy rápido en la práctica. **No se conoce** ninguna versión con cota polinomial de peor caso, y hay ejemplos en los que las reglas clásicas tardan un tiempo exponencial.

[se puede cortar] Se resuelve en tiempo polinomial

- **Elipsoide** (Khachiyan, 1979) y **puntos interiores** (Karmarkar, 1984): polinomiales en los bits de la entrada.
- **Símplex** (Dantzig): camina de esquina en esquina. Muy usado y muy rápido en la práctica. **No se conoce** ninguna versión con cota polinomial de peor caso, y hay ejemplos en los que las reglas clásicas tardan un tiempo exponencial.
- Hay solucionadores maduros de código abierto, como HiGHS y GLPK.

[se puede cortar] Se resuelve en tiempo polinomial

- **Elipsoide** (Khachiyan, 1979) y **puntos interiores** (Karmarkar, 1984): polinomiales en los bits de la entrada.
- **Símplex** (Dantzig): camina de esquina en esquina. Muy usado y muy rápido en la práctica. **No se conoce** ninguna versión con cota polinomial de peor caso, y hay ejemplos en los que las reglas clásicas tardan un tiempo exponencial.
- Hay solucionadores maduros de código abierto, como HiGHS y GLPK.

> **Lo mismo de hoy, otra vez**
>
> Lo que funciona en la práctica y lo que está garantizado en el peor caso no siempre coinciden.

[se puede cortar] Flujo máximo como programa lineal

Una variable $f(u, v)$ por arista:

$$
\begin{array}{rll}
\text{maximizar} & \sum_w f(s, w) - \sum_u f(u, s) & \\
\text{sujeto a} & \sum_w f(v, w) - \sum_u f(u, v) = 0 & \text{para cada nodo } v \neq s, t \\
& f(u, v) \leq c(u, v) & \text{para cada arista} \\
& f(u, v) \geq 0 & \text{para cada arista}
\end{array}
$$

[se puede cortar] Flujo máximo como programa lineal

Una variable $f(u, v)$ por arista:

$$
\begin{array}{rll}
\text{maximizar} & \sum_w f(s, w) - \sum_u f(u, s) & \\
\text{sujeto a} & \sum_w f(v, w) - \sum_u f(u, v) = 0 & \text{para cada nodo } v \neq s, t \\
& f(u, v) \leq c(u, v) & \text{para cada arista} \\
& f(u, v) \geq 0 & \text{para cada arista}
\end{array}
$$

Con capacidades enteras, las **esquinas** de este poliedro son enteras: una propiedad de la matriz de estas restricciones, que no demostramos.

[se puede cortar] Flujo máximo como programa lineal

Una variable $f(u, v)$ por arista:

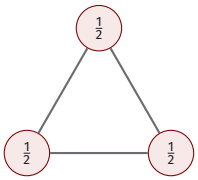
$$
\begin{array}{rll}
\text{maximizar} & \sum_w f(s, w) - \sum_u f(u, s) & \\
\text{sujeto a} & \sum_w f(v, w) - \sum_u f(u, v) = 0 & \text{para cada nodo } v \neq s, t \\
& f(u, v) \leq c(u, v) & \text{para cada arista} \\
& f(u, v) \geq 0 & \text{para cada arista}
\end{array}
$$

Con capacidades enteras, las **esquinas** de este poliedro son enteras: una propiedad de la matriz de estas restricciones, que no demostramos. Habla de las esquinas: como en la semana 12, también hay flujos máximos con fracciones.

[se puede cortar] Con enteros es otro problema

Con reales, polinomial. **Con enteros, NP-difícil.**
Desde cobertura de vértices:

$$\text{mín} \sum_v x(v), \quad x(u) + x(v) \geq 1, \quad x(v) \in \{0, 1\}$$

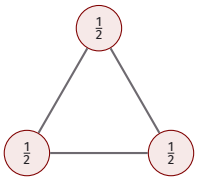


[se puede cortar] Con enteros es otro problema

Con reales, polinomial. **Con enteros, NP-difícil.**
Desde cobertura de vértices:

$$\text{mín} \sum_v x(v), \quad x(u) + x(v) \geq 1, \quad x(v) \in \{0, 1\}$$

Una cobertura y una solución entera se corresponden, con el mismo valor.



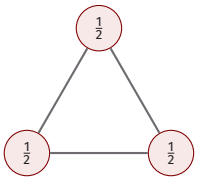
[se puede cortar] Con enteros es otro problema

Con reales, polinomial. **Con enteros, NP-difícil.**
Desde cobertura de vértices:

$$\text{mín} \sum_v x(v), \quad x(u) + x(v) \geq 1, \quad x(v) \in \{0, 1\}$$

Una cobertura y una solución entera se corresponden, con el mismo valor.

Sin "entero": con $\frac{1}{2}$ en cada nodo del triángulo, vale $\frac{3}{2}$. El óptimo entero es 2.



[se puede cortar] Con enteros es otro problema

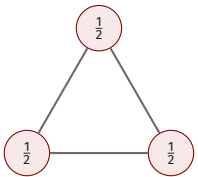
Con reales, polinomial. **Con enteros, NP-difícil.**
Desde cobertura de vértices:

$$\text{mín} \sum_v x(v), \quad x(u) + x(v) \geq 1, \quad x(v) \in \{0, 1\}$$

Una cobertura y una solución entera se corresponden, con el mismo valor.

Sin "entero": con $\frac{1}{2}$ en cada nodo del triángulo, vale $\frac{3}{2}$. El óptimo entero es 2.

Pero sirve de cota: tomen los $x(v) \geq \frac{1}{2}$. Es una cobertura de tamaño $\leq 2 \cdot \text{LP} \leq 2 \cdot \text{OPT}$. Otro factor 2.



SECCIÓN 10

Repaso: ¿qué técnica sirve?

10

Dado un problema nuevo

Señal	Técnica	Se justifica con
Se parte en pedazos del mismo tipo, y se combinan	Divide y vencerás	Recurrencia
Una decisión local es segura	Greedy	Intercambio
Decisiones dependientes, subproblemas repetidos	Prog. dinámica	Subproblema e inducción
Ir de un lugar a otro, con pesos	Caminos mínimos	Dijkstra, Bellman-Ford
Conectar todo al menor costo	Árbol de expansión	Propiedad del corte
Asignar con cupos, elegir con dependencias	Flujo o corte	Reducción, dos direcciones
Nada de eso, y se parece a uno del mapa	NP-dificultad	Reducción desde uno difícil
Es NP-difícil y hay que responder	Lo de hoy	Poda, aproximación, heurística

Dado un problema nuevo

Señal	Técnica	Se justifica con
Se parte en pedazos del mismo tipo, y se combinan	Divide y vencerás	Recurrencia
Una decisión local es segura	Greedy	Intercambio
Decisiones dependientes, subproblemas repetidos	Prog. dinámica	Subproblema e inducción
Ir de un lugar a otro, con pesos	Caminos mínimos	Dijkstra, Bellman-Ford
Conectar todo al menor costo	Árbol de expansión	Propiedad del corte
Asignar con cupos, elegir con dependencias	Flujo o corte	Reducción, dos direcciones
Nada de eso, y se parece a uno del mapa	NP-dificultad	Reducción desde uno difícil
Es NP-difícil y hay que responder	Lo de hoy	Poda, aproximación, heurística

Primero, los límites: $n = 20$ permite exponencial, y $n = 10^6$ pide $O(n \log n)$.

Pregunta 5 de repaso

- **50 médicos a turnos con cupos.** Flujo. Red $s \to$ médico $\to$ turno $\to t$, con capacidades. La integralidad da una asignación entera. El greedy ya falló en la semana 12.

Pregunta 5 de repaso

- **50 médicos a turnos con cupos.** Flujo. Red $s \to$ médico $\to$ turno $\to t$, con capacidades. La integralidad da una asignación entera. El greedy ya falló en la semana 12.

- **12 clientes, orden óptimo.** Viajero, pero con entrada pequeña: Held-Karp hace $(n-1)(n-2)2^{n-3} = 56320$ transiciones. Aproximar no tiene sentido.

Pregunta 5 de repaso

- **50 médicos a turnos con cupos.** Flujo. Red $s \to$ médico $\to$ turno $\to t$, con capacidades. La integralidad da una asignación entera. El greedy ya falló en la semana 12.
- **12 clientes, orden óptimo.** Viajero, pero con entrada pequeña: Held-Karp hace $(n-1)(n-2)2^{n-3} = 56320$ transiciones. Aproximar no tiene sentido.
- **Máximo de clases sin cruces, entre 10000.** Greedy por hora de fin, de la semana 4. $O(n \log n)$, con argumento de intercambio.

Pregunta 5 de repaso

- **50 médicos a turnos con cupos.** Flujo. Red $s \to$ médico $\to$ turno $\to t$, con capacidades. La integralidad da una asignación entera. El greedy ya falló en la semana 12.
- **12 clientes, orden óptimo.** Viajero, pero con entrada pequeña: Held-Karp hace $(n-1)(n-2)2^{n-3} = 56320$ transiciones. Aproximar no tiene sentido.
- **Máximo de clases sin cruces, entre 10000.** Greedy por hora de fin, de la semana 4. $O(n \log n)$, con argumento de intercambio.
- **Alinear dos cadenas de ADN.** Programación dinámica sobre pares de prefijos, de la semana 7, en $O(nm)$. Con un millón de bases por cadena, la tabla entera no cabe. Para el costo solo, bastan dos filas.

Qué sigue

Este curso cierra aquí su contenido. La semana 16 es el examen final.

Qué sigue

Este curso cierra aquí su contenido. La semana 16 es el examen final.

- **Teoría de la Computación.** Qué se puede calcular. Allí está la demostración de Cook-Levin.

Qué sigue

Este curso cierra aquí su contenido. La semana 16 es el examen final.

- **Teoría de la Computación.** Qué se puede calcular. Allí está la demostración de Cook-Levin.
- **Probabilidad y algoritmos aleatorizados.** Algoritmos que tiran monedas, y cómo se analizan.

Qué sigue

Este curso cierra aquí su contenido. La semana 16 es el examen final.

- **Teoría de la Computación.** Qué se puede calcular. Allí está la demostración de Cook-Levin.
- **Probabilidad y algoritmos aleatorizados.** Algoritmos que tiran monedas, y cómo se analizan.
- **Algoritmos Avanzados.** Más técnicas, más estructuras y mejores factores de aproximación.

SECCIÓN 11

Laboratorio

11

Laboratorio: dos problemas

`lab/peru_exact`, hasta 20 ciudades.

- El largo mínimo **y** el recorrido.
- El verificador mide de nuevo lo que escriben.
- La semana 10 da solo el costo: piensen cómo reconstruir.

`lab/peru_tour`, hasta 1500 lugares.

- Válido, no más de 2*A*, y ningún cambio de dos aristas lo acorta.
- `two_opt_plain` es correcto, y demasiado lento en los casos grandes.
- El verificador mira el resultado, no el método.

La tabla es de ustedes

Largo y tiempo con 10, 15 y 20 ciudades, y con las 26 solo los métodos que terminen. Comprueben el factor. Sus óptimos deben coincidir con los del capítulo.

Lo que hay que llevarse

- **Cuatro preguntas:** ¿pequeña?, ¿estructura?, ¿cercano?, ¿solucionador?
- **Poda:** una cota inferior válida corta ramas sin perder el óptimo. Sigue siendo exacto.
- **Aproximación:** polinomial, con garantía demostrada contra una cota inferior de OPT.
- **Cobertura:** matching maximal, las dos puntas. $|C| = 2|M| \leq 2 \cdot$ OPT.
- **Viajero métrico:** $A \leq$ OPT, caminata $2A$, atajar no alarga. Factor 2.
- **Sin desigualdad triangular:** ningún factor constante, salvo que P = NP.
- **2-opt:** termina en un óptimo local. No garantiza nada, salvo por dónde empezó.
- **Mediciones:** valen para esos datos. Las garantías valen siempre.
- **[se puede cortar] Programación lineal:** con reales, polinomial. Con enteros, NP-difícil.
- **Repaso:** la técnica sale de la estructura y de los límites.

Para la próxima clase

- Leer los apuntes de la semana 15, y hacer sus ejercicios. La semana 16 es el examen final.
- **Laboratorio:** `lab/peru_exact` y `lab/peru_tour`.

Preguntas

Demuestren que un matching maximal da una cobertura de vértices con factor 2.
¿Por qué el árbol de expansión mínima es una cota inferior para el viajero?
Demuestren el factor 2 de duplicar el árbol en un caso métrico. ¿Dónde se usa la desigualdad triangular?
¿Cuál es la diferencia entre una aproximación y una heurística?
Elijan una técnica para cada uno: 50 médicos a turnos con cupos; 12 clientes en orden óptimo; clases sin cruces, entre 10000; alinear dos cadenas de ADN.