Temas Avanzados de Programación Dinámica

Agustín Santiago Gutiérrez

Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires

Training Camp Argentina 2020

Contenidos

1. Dinámicas con subconjuntos
   - Idea
   - Ejemplos
2. Tabla aditiva multidimensional
   - Idea
   - Ejemplos
3. Dinámicas con frente
   - Idea
   - Ejemplos
4. Algunas optimizaciones de DP
   - Convex Hull Trick
   - “Alien Trick” (AKA multiplicador de Lagrange)
5. Optimizaciones de DP basadas en argmin monótono (Bonus, solo si sobra tiempo)
   - Optimización de Knuth
   - Optimización de Divide and Conquer

“Pay heed to the tales of old wives. It may well be that they alone keep in memory what it was once needful for the wise to know.”

J. R. R. Tolkien, The Lord of the Rings

“Memory is the thing you forget with.”

Alexander Chase, Perspectives, 1966.

“I cannot but remember such things were,
That were most precious to me.”

William Shakespeare, Macbeth
Act IV, scene 3, line 222

Contenidos

1. Dinámicas con subconjuntos
   - Idea
   - Ejemplos
2. Tabla aditiva multidimensional
   - Idea
   - Ejemplos
3. Dinámicas con frente
   - Idea
   - Ejemplos
4. Algunas optimizaciones de DP
   - Convex Hull Trick
   - “Alien Trick” (AKA multiplicador de Lagrange)
5. Optimizaciones de DP basadas en argmin monótono (Bonus, solo si sobra tiempo)
   - Optimización de Knuth
   - Optimización de Divide and Conquer

Idea

- El estado contiene un **subconjunto** $S \subseteq T$ de algún conjunto T relevante al problema
- Implementación: número de 0 a $2^n - 1$ (máscara de bits: `0` a `(1<<n)-1`)
  - $\cup$ es el `|`
  - $\cap$ es el `&`
  - $\{i\}$ es `1<<i`
  - T es `(1<<n)-1`
  - $\emptyset$ es `0`
  - $i \in S$ es `((1<<i)&S != 0)` o `((S>>i)&1)`
  - S^c es `T&(~S)` o `T-S`
  - $A - B = A \cap B^c$ es `A&(~B)` o `A&(T-B)`
- Notar que el estado podría tener más cosas, o más de un subconjunto

Contenidos

1. Dinámicas con subconjuntos
   - Idea
   - Ejemplos
2. Tabla aditiva multidimensional
   - Idea
   - Ejemplos
3. Dinámicas con frente
   - Idea
   - Ejemplos
4. Algunas optimizaciones de DP
   - Convex Hull Trick
   - “Alien Trick” (AKA multiplicador de Lagrange)
5. Optimizaciones de DP basadas en argmin monótono (Bonus, solo si sobra tiempo)
   - Optimización de Knuth
   - Optimización de Divide and Conquer

Ejemplos

- Matching perfecto de costo mínimo en grafo completo: $O(N2^N)$ [Contest 3: C. Looking for Order]

Ejemplos

- Matching perfecto de costo mínimo en grafo completo: $O(N2^N)$ [Contest 3: C. Looking for Order]
- De hecho $O(N\phi^N)$, donde $\phi = \frac{1+\sqrt{5}}{2} \approx 1{,}618$

Ejemplos

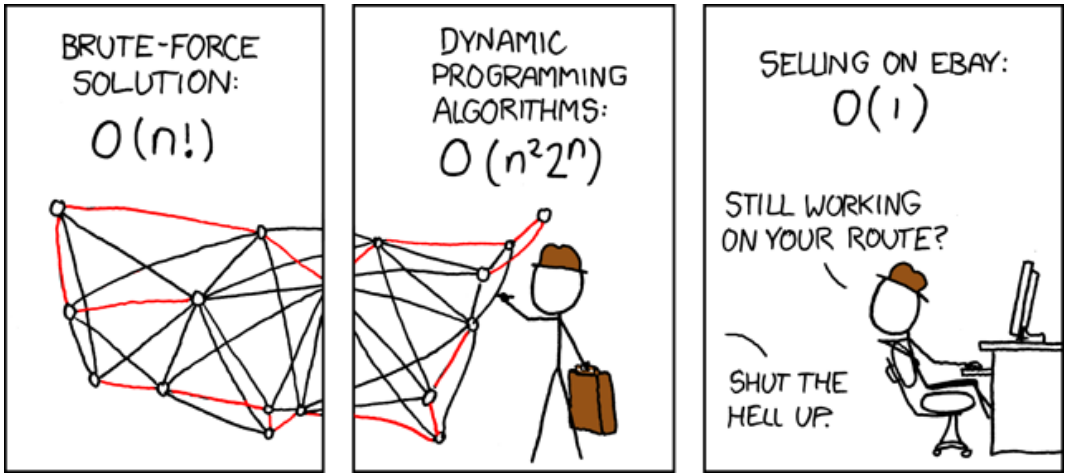
- Matching perfecto de costo mínimo en grafo completo: $O(N2^N)$ [Contest 3: C. Looking for Order]
- De hecho $O(N\phi^N)$, donde $\phi = \frac{1+\sqrt{5}}{2} \approx 1{,}618$
- Minimum set cover: $O(M2^N)$, para M conjuntos de N elementos

Ejemplos

- Matching perfecto de costo mínimo en grafo completo: $O(N2^N)$ [Contest 3: C. Looking for Order]
- De hecho $O(N\phi^N)$, donde $\phi = \frac{1+\sqrt{5}}{2} \approx 1{,}618$
- Minimum set cover: $O(M2^N)$, para M conjuntos de N elementos
- Bin Packing: $O(N2^N)$

Ejemplos

- Matching perfecto de costo mínimo en grafo completo: $O(N2^N)$ [Contest 3: C. Looking for Order]
- De hecho $O(N\phi^N)$, donde $\phi = \frac{1+\sqrt{5}}{2} \approx 1{,}618$
- Minimum set cover: $O(M2^N)$, para M conjuntos de N elementos
- Bin Packing: $O(N2^N)$
- TSP: $O(N^2 2^N)$



- Todas son **muchísimo** más eficientes que el backtracking directo

Contenidos

1. Dinámicas con subconjuntos
   - Idea
   - Ejemplos
2. Tabla aditiva multidimensional
   - Idea
   - Ejemplos
3. Dinámicas con frente
   - Idea
   - Ejemplos
4. Algunas optimizaciones de DP
   - Convex Hull Trick
   - “Alien Trick” (AKA multiplicador de Lagrange)
5. Optimizaciones de DP basadas en argmin monótono (Bonus, solo si sobra tiempo)
   - Optimización de Knuth
   - Optimización de Divide and Conquer

Sumas parciales (1D)

- *dp* se inicializa igual que el vector de entrada
- Se hace `dp(i) += dp(i-1)` para $i > 0$, en orden

Con este cómputo $O(N)$ está la suma de todos los **anteriores**

Sumas parciales (2D)

- *dp* se inicializa igual que la matriz de entrada
- Se hace `dp(i,j) += dp(i-1,j)` para $i > 0$, para todos los j
- **Luego** se hace `dp(i,j) += dp(i,j-1)` para $j > 0$, y todo i

Con este cómputo $O(NM)$ está la suma de todos los **anteriores** en la matriz: $dp(i,j) = \sum_{a=0}^{i} \sum_{b=0}^{j} v(a,b)$

Es exactamente la idea del teorema de Fubini (con sumatorias).

Código multidimensional 5D

Ejemplo conceptual para 5 dimensiones. Misma idea, acumular en cada una por separado sucesivamente:

- Poner en *dp* la entrada
- Hacer:

```
for dim = 0 to 4
 for a = 0 to na-1
  for b = 0 to nb-1
   for c = 0 to nc-1
    for d = 0 to nd-1
     for e = 0 to ne-1
      pa = a-(dim==0); pb = b-(dim==1); pc = c-(dim==2);
      pd = d-(dim==3); pe = e-(dim==4);
      if (pa >= 0 && pb >= 0 && pc >= 0 &&
          pd >= 0 && pe >= 0)
       dp(a,b,c,d,e) += dp(pa,pb,pc,pd,pe)
```

Para una matriz de T celdas en total y con d dimensiones, este cómputo $O(Td)$ / $O(Td^2)$ calcula la suma de todos los **anteriores**

Contenidos

1. Dinámicas con subconjuntos
   - Idea
   - Ejemplos
2. Tabla aditiva multidimensional
   - Idea
   - Ejemplos
3. Dinámicas con frente
   - Idea
   - Ejemplos
4. Algunas optimizaciones de DP
   - Convex Hull Trick
   - “Alien Trick” (AKA multiplicador de Lagrange)
5. Optimizaciones de DP basadas en argmin monótono (Bonus, solo si sobra tiempo)
   - Optimización de Knuth
   - Optimización de Divide and Conquer

Sumas para los subconjuntos

- Un subconjunto de T, que tiene n elementos, es un elemento de una matriz de $2 \times 2 \times \cdots \times 2$ (n veces) [hipercubo]
- Esto pues la máscada de bits nos da las n coordenadas, 0 o 1
- Los subconjuntos de un elemento, son a su vez los anteriores en dicha matriz
- Con un for de 1 a n, y luego pasando por los 2^n conjuntos haciendo una suma, se computan todas las sumas sobre subconjuntos de cada subconjunto:

```
for i = 0 to n-1
 for mask = 0 to (1«n) - 1
  if (mask & (1«i))
   dp(mask) += dp(mask - (1«i))
```

Sumas para los divisores

- Sea S es un conjunto cerrado por divisores, y P la lista de primos que aparecen en los números de S
- Un número tiene $|P|$ coordenadas, los exponentes de los primos.
- Los divisores son justamente los “menores”, vistos con las coordenadas
- Se puede sumar los valores sobre los divisores usando la idea de tabla aditiva $|P|$ dimensional:

```
for p ∈ P
 for x ∈ S (DE MENOR A MAYOR)
  if (p divide a x)
   dp(x) += dp(x / p)
```

Contenidos

1. Dinámicas con subconjuntos
   - Idea
   - Ejemplos
2. Tabla aditiva multidimensional
   - Idea
   - Ejemplos
3. Dinámicas con frente
   - Idea
   - Ejemplos
4. Algunas optimizaciones de DP
   - Convex Hull Trick
   - “Alien Trick” (AKA multiplicador de Lagrange)
5. Optimizaciones de DP basadas en argmin monótono (Bonus, solo si sobra tiempo)
   - Optimización de Knuth
   - Optimización de Divide and Conquer

Idea

- Tenemos que considerar formas de “llenar” un tablero.
- Como siempre en programación dinámica, queremos “olvidar lo más posible”
- Al ir llenando el tablero, suele alcanzar con saber únicamente la situación en el **frente** por dónde vamos llenando.
- Por ejemplo si queremos llenar un tablero con 0 y 1 pero sin que haya dos 1 pegados:

? ? ? ?
? ? ? ?
? ? ? ?
? ? ? ?

Idea

- Tenemos que considerar formas de “llenar” un tablero.
- Como siempre en programación dinámica, queremos “olvidar lo más posible”
- Al ir llenando el tablero, suele alcanzar con saber únicamente la situación en el **frente** por dónde vamos llenando.
- Por ejemplo si queremos llenar un tablero con 0 y 1 pero sin que haya dos 1 pegados:

0 ? ? ?
? ? ? ?
? ? ? ?
? ? ? ?

Idea

- Tenemos que considerar formas de “llenar” un tablero.
- Como siempre en programación dinámica, queremos “olvidar lo más posible”
- Al ir llenando el tablero, suele alcanzar con saber únicamente la situación en el **frente** por dónde vamos llenando.
- Por ejemplo si queremos llenar un tablero con 0 y 1 pero sin que haya dos 1 pegados:

0 ? ? ?
1 ? ? ?
? ? ? ?
? ? ? ?

Idea

- Tenemos que considerar formas de “llenar” un tablero.
- Como siempre en programación dinámica, queremos “olvidar lo más posible”
- Al ir llenando el tablero, suele alcanzar con saber únicamente la situación en el **frente** por dónde vamos llenando.
- Por ejemplo si queremos llenar un tablero con 0 y 1 pero sin que haya dos 1 pegados:

0 ? ? ?
1 ? ? ?
0 ? ? ?
? ? ? ?

Idea

- Tenemos que considerar formas de “llenar” un tablero.
- Como siempre en programación dinámica, queremos “olvidar lo más posible”
- Al ir llenando el tablero, suele alcanzar con saber únicamente la situación en el **frente** por dónde vamos llenando.
- Por ejemplo si queremos llenar un tablero con 0 y 1 pero sin que haya dos 1 pegados:

0 ? ? ?
1 ? ? ?
0 ? ? ?
1 ? ? ?

Idea

- Tenemos que considerar formas de “llenar” un tablero.
- Como siempre en programación dinámica, queremos “olvidar lo más posible”
- Al ir llenando el tablero, suele alcanzar con saber únicamente la situación en el **frente** por dónde vamos llenando.
- Por ejemplo si queremos llenar un tablero con 0 y 1 pero sin que haya dos 1 pegados:

0 1 ? ?
1 ? ? ?
0 ? ? ?
1 ? ? ?

Idea

- Tenemos que considerar formas de “llenar” un tablero.
- Como siempre en programación dinámica, queremos “olvidar lo más posible”
- Al ir llenando el tablero, suele alcanzar con saber únicamente la situación en el **frente** por dónde vamos llenando.
- Por ejemplo si queremos llenar un tablero con 0 y 1 pero sin que haya dos 1 pegados:

0 1 ? ?
1 0 ? ?
0 ? ? ?
1 ? ? ?

Idea

- Tenemos que considerar formas de “llenar” un tablero.
- Como siempre en programación dinámica, queremos “olvidar lo más posible”
- Al ir llenando el tablero, suele alcanzar con saber únicamente la situación en el **frente** por dónde vamos llenando.
- Por ejemplo si queremos llenar un tablero con 0 y 1 pero sin que haya dos 1 pegados:

0 1 ? ?
1 0 ? ?
0 0 ? ?
1 ? ? ?

Idea

- Tenemos que considerar formas de “llenar” un tablero.
- Como siempre en programación dinámica, queremos “olvidar lo más posible”
- Al ir llenando el tablero, suele alcanzar con saber únicamente la situación en el **frente** por dónde vamos llenando.
- Por ejemplo si queremos llenar un tablero con 0 y 1 pero sin que haya dos 1 pegados:

0 1 ? ?
1 0 ? ?
0 0 ? ?
1 0 ? ?

Idea

- Tenemos que considerar formas de “llenar” un tablero.
- Como siempre en programación dinámica, queremos “olvidar lo más posible”
- Al ir llenando el tablero, suele alcanzar con saber únicamente la situación en el **frente** por dónde vamos llenando.
- Por ejemplo si queremos llenar un tablero con 0 y 1 pero sin que haya dos 1 pegados:

0	1	0	?
1	0	?	?
0	0	?	?
1	0	?	?

Idea

- Tenemos que considerar formas de “llenar” un tablero.
- Como siempre en programación dinámica, queremos “olvidar lo más posible”
- Al ir llenando el tablero, suele alcanzar con saber únicamente la situación en el **frente** por dónde vamos llenando.
- Por ejemplo si queremos llenar un tablero con 0 y 1 pero sin que haya dos 1 pegados:

0 1 0 ?
1 0 0 ?
0 0 ? ?
1 0 ? ?

Estado

- El estado tendrá la posición (x, y) actual en el tablero.
- Además, para un tablero de $N \times M$, guardará N valores (o a veces $N + 1$) con el frente.
- Si los valores son binarios como en el ejemplo hay $O(NM2^N)$ estados, pero en cambio hay 2^{NM} tableros.

Contenidos

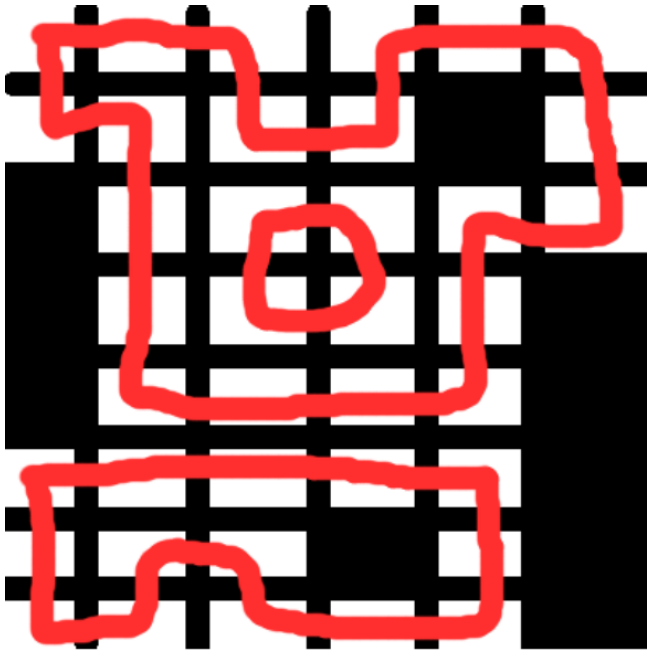
1. Dinámicas con subconjuntos
   - Idea
   - Ejemplos
2. Tabla aditiva multidimensional
   - Idea
   - Ejemplos
3. Dinámicas con frente
   - Idea
   - Ejemplos
4. Algunas optimizaciones de DP
   - Convex Hull Trick
   - “Alien Trick” (AKA multiplicador de Lagrange)
5. Optimizaciones de DP basadas en argmin monótono (Bonus, solo si sobra tiempo)
   - Optimización de Knuth
   - Optimización de Divide and Conquer

Ejemplos

- Cantidad de maneras de cubrir con dominós, un tablero con agujeros en posiciones dadas

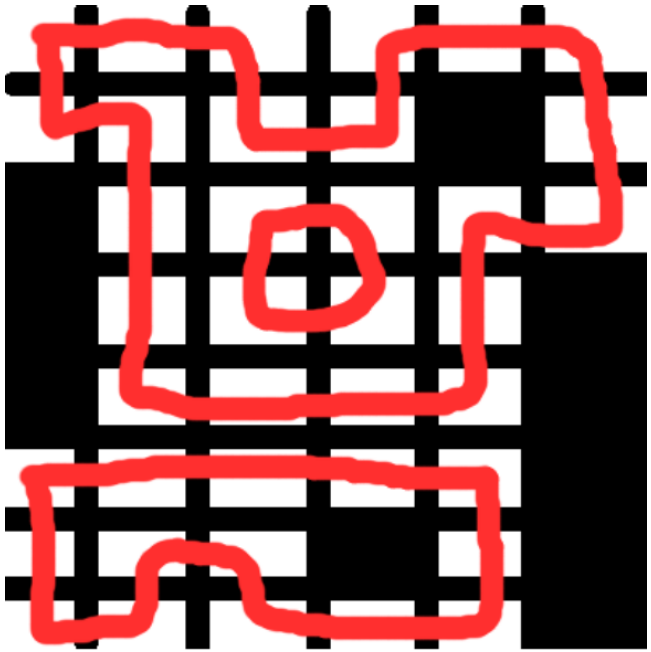
Ejemplos

- Cantidad de maneras de cubrir con dominós, un tablero con agujeros en posiciones dadas
- Cantidad de maneras de cubrir con “tuberías” cerradas (ciclos), un tablero con agujeros en posiciones dadas



Ejemplos

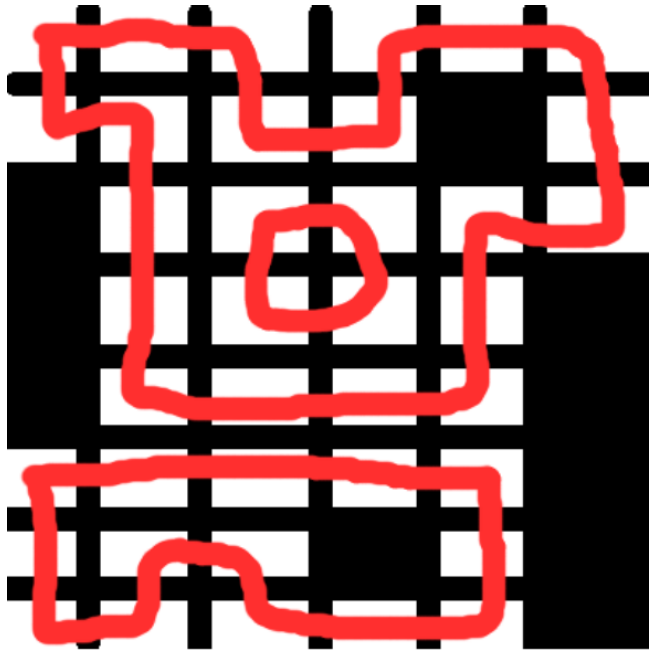
- Cantidad de maneras de cubrir con dominós, un tablero con agujeros en posiciones dadas
- Cantidad de maneras de cubrir con “tuberías” cerradas (ciclos), un tablero con agujeros en posiciones dadas



- El frente anterior tiene $O(2^N)$ valores posibles. ¿Y si quisiéramos saber la mínima cantidad de ciclos necesarios?

Ejemplos

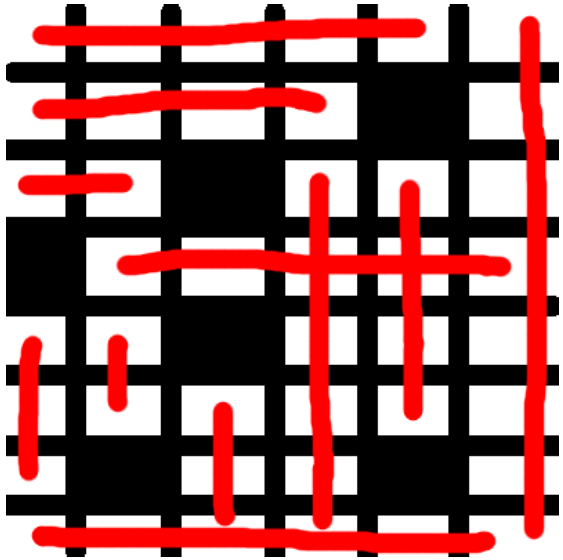
- Cantidad de maneras de cubrir con dominós, un tablero con agujeros en posiciones dadas
- Cantidad de maneras de cubrir con “tuberías” cerradas (ciclos), un tablero con agujeros en posiciones dadas



- El frente anterior tiene $O(2^N)$ valores posibles. ¿Y si quisiéramos saber la mínima cantidad de ciclos necesarios?
- Se puede con un frente con $O(3^N)$ valores posibles (desafío)

Ejemplo final: pintar mural

- Hay un mural (grilla con obstáculos). Podemos hacer pintadas verticales u horizontales. Podemos pasar varias veces por la misma casilla, pero nunca por un obstáculo. Calcular mínima cantidad de pinceladas para pintar todas las casillas sin obstáculo.

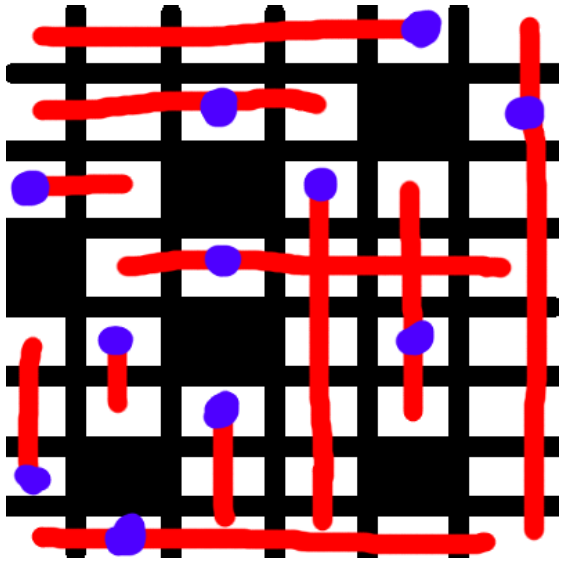


Ejemplo final: pintar mural

- También se puede resolver con una DP con frente: $O(NM2^N)$

Ejemplo final: pintar mural

- También se puede resolver con una DP con frente: $O(NM2^N)$
- Existe en este caso una solución polinomial con matching máximo bipartito:



Contenidos

1. Dinámicas con subconjuntos
   - Idea
   - Ejemplos
2. Tabla aditiva multidimensional
   - Idea
   - Ejemplos
3. Dinámicas con frente
   - Idea
   - Ejemplos
4. Algunas optimizaciones de DP
   - Convex Hull Trick
   - “Alien Trick” (AKA multiplicador de Lagrange)
5. Optimizaciones de DP basadas en argmin monótono (Bonus, solo si sobra tiempo)
   - Optimización de Knuth
   - Optimización de Divide and Conquer

CHT aplicado a DP

- Ya vimos en otra charla que CHT es una técnica e idea geométrica independiente de DP, que responde queries.
- Es muy común sin embargo usar esta estructura para optimizar algoritmos de DP.

CHT aplicado a DP: Ejemplos

- Tengo N vendedores ordenados en fila. Cada vendedor tiene un entero a_i (posiblemente negativo). Yo voy pasando y a algunos les compro. Si le compro al i, obtengo un beneficio $i^2 a_j$, donde j es el comerciante anterior al cual le compré (o bien $a_j = 0$ si es a i al primero que compro). ¿Cuál es el máximo beneficio posible?

CHT aplicado a DP: Ejemplos

- Tengo N vendedores ordenados en fila. Cada vendedor tiene un entero a_i (posiblemente negativo). Yo voy pasando y a algunos les compro. Si le compro al i, obtengo un beneficio $i^2 a_j$, donde j es el comerciante anterior al cual le compré (o bien $a_j = 0$ si es a i al primero que compro). ¿Cuál es el máximo beneficio posible?
- Propuesta: `dp[j]` = "Beneficio óptimo de los siguientes comerciantes, si empiezo comprando en el j"
- Propuesta: $dp(j) = max_{i=j+1}^{n}\ i^2 a_j + dp(i)$

CHT aplicado a DP: Ejemplos

- Tengo N vendedores ordenados en fila. Cada vendedor tiene un entero a_i (posiblemente negativo). Yo voy pasando y a algunos les compro. Si le compro al i, obtengo un beneficio $i^2 a_j$, donde j es el comerciante anterior al cual le compré (o bien $a_j = 0$ si es a i al primero que compro). ¿Cuál es el máximo beneficio posible?
- Propuesta: `dp[j]` = "Beneficio óptimo de los siguientes comerciantes, si empiezo comprando en el j"
- Propuesta: $dp(j) = max_{i=j+1}^{n}\ i^2 a_j + dp(i)$
- Si a medida que calculo voy insertando las rectas $y = i^2 x + dp(i)$ (que están ordenadas por pendiente) queda $O(N \lg N)$ con CHT

CHT aplicado a DP: Ejemplos

- Tengo N vendedores ordenados en fila. Cada vendedor tiene un entero a_i (posiblemente negativo). Yo voy pasando y a algunos les compro. Si le compro al i, obtengo un beneficio $i^2 a_j$, donde j es el comerciante anterior al cual le compré (o bien $a_j = 0$ si es a i al primero que compro). ¿Cuál es el máximo beneficio posible?
- Propuesta: `dp[j]` = "Beneficio óptimo de los siguientes comerciantes, si empiezo comprando en el j"
- Propuesta: $dp(j) = max_{i=j+1}^{n}\ i^2 a_j + dp(i)$
- Si a medida que calculo voy insertando las rectas $y = i^2 x + dp(i)$ (que están ordenadas por pendiente) queda $O(N \lg N)$ con CHT
- Otro ejemplo: Problema K. The Fair Nut and Rectangles del contest 3.

Contenidos

1. Dinámicas con subconjuntos
   - Idea
   - Ejemplos
2. Tabla aditiva multidimensional
   - Idea
   - Ejemplos
3. Dinámicas con frente
   - Idea
   - Ejemplos
4. Algunas optimizaciones de DP
   - Convex Hull Trick
   - “Alien Trick” (AKA multiplicador de Lagrange)
5. Optimizaciones de DP basadas en argmin monótono (Bonus, solo si sobra tiempo)
   - Optimización de Knuth
   - Optimización de Divide and Conquer

¿Qué es Alien Trick? Un poco de historia

- Es una idea muy ingeniosa para reemplazar un parámetro de DP, por una búsqueda binaria.
- Se la conoce así porque “se volvió viral” con el problema Aliens de la IOI Rusia 2016.
- La técnica ya era conocida desde antes, aunque casi no se usaba en programación competitiva:

¿Qué es Alien Trick? Un poco de historia

- Es una idea muy ingeniosa para reemplazar un parámetro de DP, por una búsqueda binaria.
- Se la conoce así porque “se volvió viral” con el problema Aliens de la IOI Rusia 2016.
- La técnica ya era conocida desde antes, aunque casi no se usaba en programación competitiva:
  - Se menciona en “Apuntes de Programación Dinámica 1995” de Fabio Vicentini, FCEN-UBA

¿Qué es Alien Trick? Un poco de historia

- Es una idea muy ingeniosa para reemplazar un parámetro de DP, por una búsqueda binaria.
- Se la conoce así porque “se volvió viral” con el problema Aliens de la IOI Rusia 2016.
- La técnica ya era conocida desde antes, aunque casi no se usaba en programación competitiva:
  - Se menciona en “Apuntes de Programación Dinámica 1995” de Fabio Vicentini, FCEN-UBA
  - Tiene una estrecha relación con los archi-conocidos multiplicadores de Lagrange, por lo que en algunos textos matemáticos se encuentra con ese nombre.

¿Qué es Alien Trick? Un poco de historia

- Es una idea muy ingeniosa para reemplazar un parámetro de DP, por una búsqueda binaria.
- Se la conoce así porque “se volvió viral” con el problema Aliens de la IOI Rusia 2016.
- La técnica ya era conocida desde antes, aunque casi no se usaba en programación competitiva:
  - Se menciona en “Apuntes de Programación Dinámica 1995” de Fabio Vicentini, FCEN-UBA
  - Tiene una estrecha relación con los archi-conocidos multiplicadores de Lagrange, por lo que en algunos textos matemáticos se encuentra con ese nombre.
  - En otros textos en cambio, se le llamaría Parameter Search.

¿Qué es Alien Trick? Un poco de historia

- Es una idea muy ingeniosa para reemplazar un parámetro de DP, por una búsqueda binaria.
- Se la conoce así porque “se volvió viral” con el problema Aliens de la IOI Rusia 2016.
- La técnica ya era conocida desde antes, aunque casi no se usaba en programación competitiva:
  - Se menciona en “Apuntes de Programación Dinámica 1995” de Fabio Vicentini, FCEN-UBA
  - Tiene una estrecha relación con los archi-conocidos multiplicadores de Lagrange, por lo que en algunos textos matemáticos se encuentra con ese nombre.
  - En otros textos en cambio, se le llamaría Parameter Search.
  - Los competidores de olimpíadas chinos ya conocían y usaban este truquito desde antes y lo llamaban “wqs binary search”, pues el competidor wqs lo contó en un training camp chino y se volvió popular allí.

Problema de ejemplo para fijar ideas

- Supongamos el siguiente problema: dado un arreglo de n números distintos, separarlo en **a lo sumo** k subarreglos minimizando la suma de las diferencias (absolutas) **entre elementos consecutivos en un mismo subarreglo**.
- Este problema es fácil de resolver golosamente en $O(n \lg n)$, pero lo ignoraremos para tener un ejemplo fácil de entender.
- Podemos plantear $dp(n, k)$ como antes y queda $dp(n, k) = min(dp(n-1, k-1), dp(n-1, k) + |a_n - a_{n-1}|)$, de complejidad $O(nk)$.
- Utilizaremos la técnica de multiplicador de Lagrange para acelerar esta dp.

Idea central

- La idea central es eliminar por completo el parámetro k, pero **introducir un costo** $\lambda \geq 0$ **por cada subarreglo** que se usa.
- Así, el nuevo problema $f_\lambda(n)$ es la forma óptima de partir los primeros n en subarreglos:
  - La cantidad ya no está restringida, es libre.
  - Pero cada subarreglo distinto implica un costo λ
  - DP lineal: $f_\lambda(n) = min(f_\lambda(n-1) + \lambda, f_\lambda(n-1) + |a_n - a_{n-1}|)$
- Intuitivamente, transformamos una **restricción** estricta, en un simple **costo** ajustable λ.
- ¿Para qué sirve este nuevo problema?

Lemas fundamentales

- Notamos $k_\lambda(n)$ a la cantidad de subarreglos que utiliza una solución óptima para el problema $f_\lambda(n)$ (argmin en diapo previa).
- k_λ se puede calcular en la misma dp lineal de f_λ
- Las funciones f_λ y k_λ tienen propiedades clave:

Lemas fundamentales

- Notamos $k_\lambda(n)$ a la cantidad de subarreglos que utiliza una solución óptima para el problema $f_\lambda(n)$ (argmin en diapo previa).
- k_λ se puede calcular en la misma dp lineal de f_λ
- Las funciones f_λ y k_λ tienen propiedades clave:
  - Si $0 \leq \lambda < \mu$ entonces $k_\lambda \geq k_\mu$ (“más caros, menos uso”)

Lemas fundamentales

- Notamos $k_\lambda(n)$ a la cantidad de subarreglos que utiliza una solución óptima para el problema $f_\lambda(n)$ (argmin en diapo previa).
- k_λ se puede calcular en la misma dp lineal de f_λ
- Las funciones f_λ y k_λ tienen propiedades clave:
  - Si $0 \leq \lambda < \mu$ entonces $k_\lambda \geq k_\mu$ (“más caros, menos uso”)
  - Si x es una solución óptima de $f_\lambda(n)$, entonces x es una solución óptima de $f(n, k_\lambda(n))$

Utilidad de los lemas fundamentales

- Por el segundo lema, si encontramos algún λ tal que $k_\lambda(n)$ tome el valor k de nuestro problema original, la solución que encontramos al calcular f_λ es óptima para el problema original.

Utilidad de los lemas fundamentales

- Por el segundo lema, si encontramos algún λ tal que $k_\lambda(n)$ tome el valor k de nuestro problema original, la solución que encontramos al calcular f_λ es óptima para el problema original. f
- ¡Pero por el primer lema, se puede buscar tal λ con búsqueda binaria!
- La complejidad final resulta entonces $N \lg MAX$

Pero hay un problema...

- En la explicación anterior se asume que va a existir un λ tal que k_λ tenga el valor que deseamos.
- Si existe, lo vamos a encontrar y ya sabemos que la respuesta va a ser óptima. Pero en general, podría no existir...

Pero hay un problema...

- En la explicación anterior se asume que va a existir un λ tal que k_λ tenga el valor que deseamos.
- Si existe, lo vamos a encontrar y ya sabemos que la respuesta va a ser óptima. Pero en general, podría no existir... f
- Teorema: Si $a(k) =$ “Solución al problema usando **exactamente** k subarreglos” es estrictamente convexa como función de k, entonces existe el λ deseado.
- Estrictamente convexa significa $a(k+1) - a(k) < a(k+2) - a(k+1)$ (“derivada creciente”)
- Intuitivamente: Cada subarreglo adicional que nos dan tiene que ser “menos útil” que el anterior.
- Si la función es convexa aunque no estrictamente, hay que tener cierto cuidado pero aún puede utilizarse la misma técnica.

Sobre convexidad

- Como casi siempre en optimización de DP, demostrar la convexidad no siempre es fácil.
- Pero casi siempre es posible convencerse bastante intuitivamente.

Multiplicador de Lagrange (en general)

- Ahora que vimos el ejemplo y la idea general, plantiemos lo mismo un poco más en general.
- Buscamos una solución x en un espacio de posibles soluciones X
- Llamamos $f(x)$ al costo que buscamos minimizar
- Hay una función $h(x) > 0$ de **restricción**, que generalmente mide "cuánto recurso usamos"
- Llamamos $a(k) = min_{h(x)=k} f(x)$ (valor de la mejor solución que "usa una cantidad k de recurso")
- En general lo que buscamos calcular es $a(k)$, o a veces el análogo con $h(x) \leq k$

Multiplicador de Lagrange (cont.)

- Definimos $f_\lambda(x) = f(x) + \lambda h(x)$, para $\lambda \geq 0$
- De las definiciones anteriores, una expresión para el valor óptimo es f_λ es: $opt(\lambda) = f_\lambda(x_{optimo}) = min_{k=1}^{n} a(k) + \lambda k$
- Llamamos k_λ al menor valor posible de k en el que se alcanza el mínimo de la fórmula anterior
- Los lemas fundamentales son:
  - $opt(\lambda)$ es estrictamente creciente
  - Si $0 \leq \lambda < \mu$ entonces $k_\lambda \geq k_\mu$
  - Si $opt(\lambda) = f_\lambda(x_{optimo})$, entonces x_{optimo} es una solución óptima para el problema $min_{h(x) \leq h(x_{optimo})} f(x)$

Existencia de λ

- ¿Cuándo podemos garantizar que habrá un λ adecuado?
- La clave está en analizar cuándo se reduce k_λ
- Si $k_\lambda = t$, de $opt(\lambda) = min_{k=1}^{n} a(k) + \lambda k$ se deduce que k_λ cambiará cuando sea $a(t-1) - a(t) \leq \lambda$
- Si a es estrictamente convexa, estas condiciones se cumplen todas en distintos valores de λ, así que k_λ pasa por todos los valores.
- También se observa que si a toma valores enteros, entonces basta con probar valores enteros de λ.

Caso *a* convexa no estricta

- Si la función a es convexa no estrictamente, puede pasar $k_\lambda - k_{\lambda+1} > 1$.
- En ese caso, todos los k que cumplen $k_{\lambda+1} \leq k \leq k_\lambda$ optimizan f_λ, así que sus valores coinciden con el calculado la obtener $k_{\lambda+1}$
- Esto permite calcular los distintos $a(k)$ en este rango, pues forman una progresión aritmética con diferencia λ.
- Reconstruir la solución y no solo el valor en este caso puede ser un problema más complicado, ya que tenemos la dp ejecutada que obtuvo un cierto k, pero queremos otro.

Patrón general (resumen)

- Tenemos una clásica dp del estilo $dp(n, k) =$ “Forma óptima de resolver para los n primeros elementos, usando **a lo sumo** k”
- Nos queda una DP $O(nk^2)$, con suerte $O(nk)$, pero ya no podemos bajar más a menos que cambiemos el estado.
- Solución: Introduciendo un multiplicador de Lagrange λ, reducimos el estado a solamente $dp(n)$, pero pudiendo controlar el k mediante búsqueda binaria variando λ.

Lectura adicional

- http://serbanology.com/show_article.php?art=The%20Trick%20From%20Aliens
- https://codeforces.com/blog/entry/68778
- https://ioinformatics.org/files/ioi2016solutions.pdf
- https://codeforces.com/blog/entry/49691
- Problema Aliens para intentar: https://wcipeg.com/problem/ioi1623

Contenidos

1. Dinámicas con subconjuntos
   - Idea
   - Ejemplos
2. Tabla aditiva multidimensional
   - Idea
   - Ejemplos
3. Dinámicas con frente
   - Idea
   - Ejemplos
4. Algunas optimizaciones de DP
   - Convex Hull Trick
   - “Alien Trick” (AKA multiplicador de Lagrange)
5. Optimizaciones de DP basadas en argmin monótono (Bonus, solo si sobra tiempo)
   - Optimización de Knuth
   - Optimización de Divide and Conquer

Ejemplo motivador

- Dados n valores enteros distintos v_i, junto a sus frecuencias f_i, dar un árbol binario de búsqueda óptimo para los valores.

Ejemplo motivador

- Dados n valores enteros distintos v_i, junto a sus frecuencias f_i, dar un árbol binario de búsqueda óptimo para los valores.
- $dp(i,j) = \min_{k=i}^{j-1} dp(i,k) + dp(k+1,j) + sum_f(i,j)$
- Complejidad: $O(n^3)$
- ¿Se podrá mejorar?

Contexto

- ¡Sí! Con la optimización de Knuth
- Dado un algoritmo de dp de dos parámetros cualquiera, es decir $dp(i, j)$ con $0 \leq i, j \leq N$ (ejemplo $i \leq j$ si es **DP en rangos**)
- Si su recursión tiene la forma $dp(i, j) = \min_k g(i, j, k)$ para cierta g que solo usa valores $dp(a, b)$ con $a \geq i$ y con $b \leq j$
- Podemos definir $K(i, j)$ como el menor k en donde se alcanza el mínimo de la expresión para $dp(i, j)$
- Para escribir las complejidades, notaremos $|K|$ a la cantidad de valores posibles para k en la expresión. Típicamente $|K|$ es aproximadamente N
- En las típicas dp en rango, k se mueve entre i y j.

Condición de Knuth

- $K(i, j-1) \leq K(i, j) \leq K(i+1, j)$
- Equivalentemente: K es **monótona en ambos parámetros**.
- En criollo para DPs en rangos:
  - Si agregamos un elemento por **izquierda**, el K se mueve **a la izquierda**
  - Si agregamos un elemento por **derecha**, el K se mueve **a la derecha**
- Llamamos a la anterior la *Condición de Knuth*
- Suele ser mucho más difícil demostrar que se cumple, que convencerse o intuir que así será

Optimización de Knuth

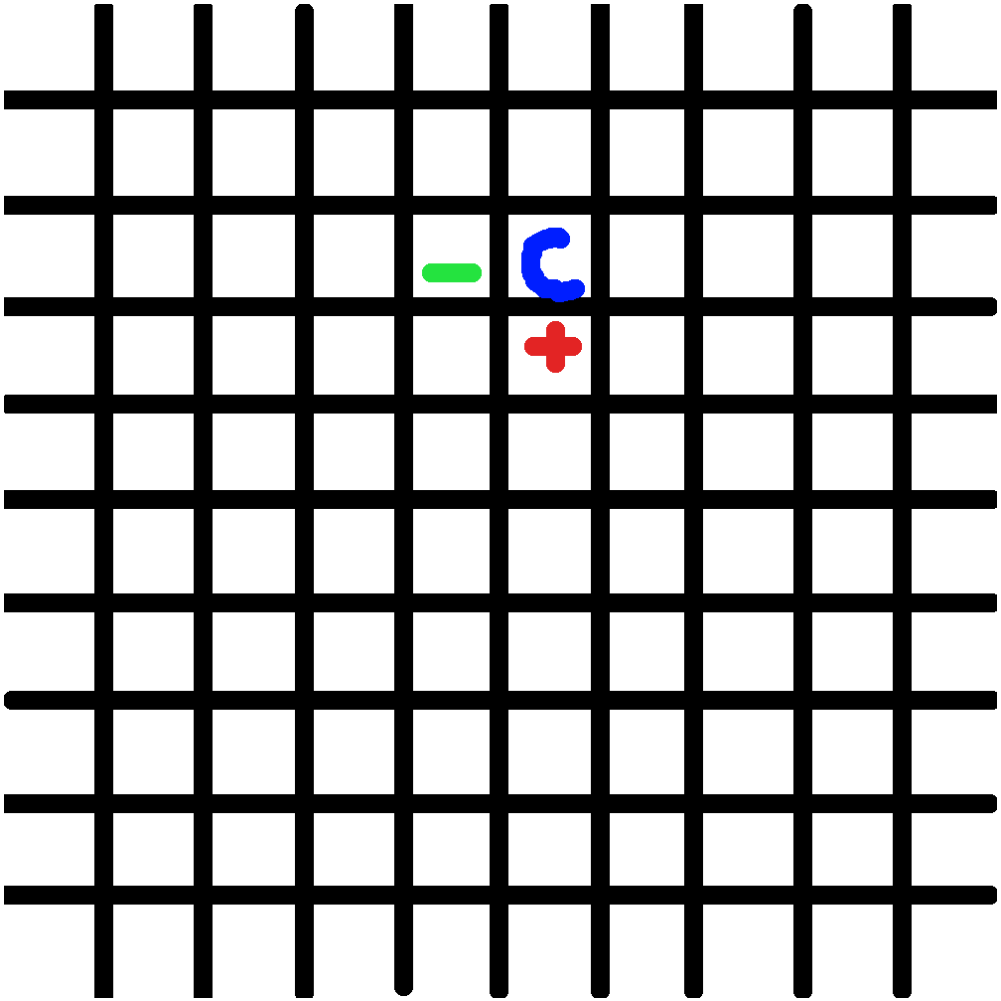
- Como vale la condición de Knuth, es muy simple cambiar en el código la recursión usando las cotas para iterar menos:
- $dp(i,j) = \min_{k=K(i,j-1)}^{K(i+1,j)} g(i,j,k)$
- Los K los podemos ir calculando en el mismo algoritmo junto a los valores dp.
- Las cotas sirven para iterar menos... ¿Pero estamos mejorando la complejidad asintótica?

Optimización de Knuth

- Como vale la condición de Knuth, es muy simple cambiar en el código la recursión usando las cotas para iterar menos:
- $dp(i,j) = \min_{k=K(i,j-1)}^{K(i+1,j)} g(i,j,k)$
- Los K los podemos ir calculando en el mismo algoritmo junto a los valores dp.
- Las cotas sirven para iterar menos... ¿Pero estamos mejorando la complejidad asintótica?
- Teorema: Con este sencillísimo cambio al código básico, el algoritmo es $O(N(N+|K|))$ (la cota anterior era $O(N^2|K|)$
- Demostración: La sumatoria de los costos es telescópica en 2D (dibujos)
- Si evaluar g no es $O(1)$, el costo son $O(N(N+|K|))$ llamadas a g

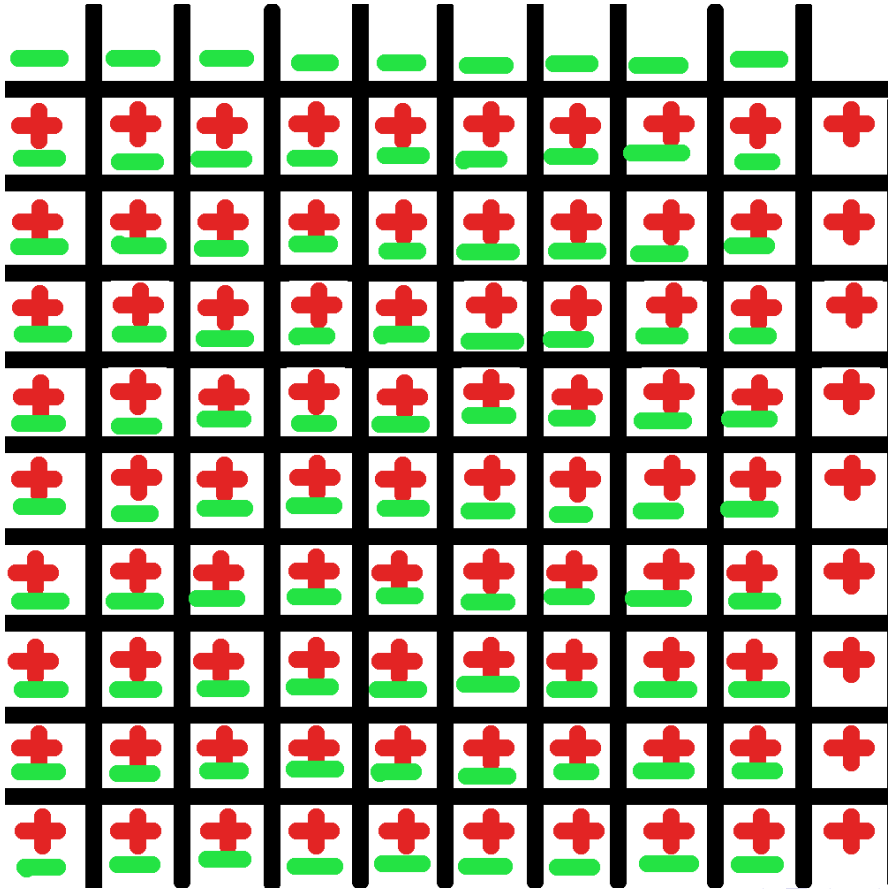
Optimización de Knuth (dibujos)

- Costo para calcular $dp(i, j)$: $K(i+1, j) - K(i, j-1) + 1$



Optimización de Knuth (dibujos)

- A lo largo de casi toda la matriz, los términos K se cancelan.
- Las N^2 casillas pagan el término 1: Total $O(N^2)$
- Las $O(N)$ casillas de borde pagan $O(|K|)$ Total $O(N|K|)$



Comentarios

- El dibujo muestra una matriz “completa” de $N \times N$
- La demostración aplica si “borde” $O(N)$ y tamaño $O(N^2)$
- Ejemplo: la parte triangular superior (dp en rangos)

Ejercicio

- Ejercicio: Verificar que la condición de Knuth aplica en la recursión que vimos antes
- Intuitivamente tiene muchísimo sentido, ¡pero demostrarlo es mucho más difícil que programarlo!

Contenidos

1. Dinámicas con subconjuntos
   - Idea
   - Ejemplos
2. Tabla aditiva multidimensional
   - Idea
   - Ejemplos
3. Dinámicas con frente
   - Idea
   - Ejemplos
4. Algunas optimizaciones de DP
   - Convex Hull Trick
   - “Alien Trick” (AKA multiplicador de Lagrange)
5. Optimizaciones de DP basadas en argmin monótono (Bonus, solo si sobra tiempo)
   - Optimización de Knuth
   - Optimización de Divide and Conquer

Ejemplo motivador

- Dados n valores x_i enteros positivos, el costo de un intervalo $[i, j)$ es $\sum_{i \leq a < b < j} x_a x_b$. O sea, sumar los productos de a pares.
- Particionar el arreglo $[0, n)$ en k intervalos, minimizando la suma de los k costos.

Ejemplo motivador

- Dados n valores x_i enteros positivos, el costo de un intervalo $[i, j)$ es $\sum_{i \leq a < b < j} x_a x_b$. O sea, sumar los productos de a pares.
- Particionar el arreglo $[0, n)$ en k intervalos, minimizando la suma de los k costos.
- $dp(n, k) = \min_{i=0}^{n-1} dp(i, k-1) + val(i, n)$
- Complejidad: $O(N^2 K)$
- ¿Se podrá mejorar?

Contexto

- ¡Sí! Con la optimización de Divide and Conquer
- Dado un algoritmo de dp con dos parámetros, es decir $dp(n, k)$ con $0 \leq n \leq N$ y $0 \leq k \leq K$
- Si su recursión tiene la forma $dp(n, k) = \min_i g(n, k, i)$ para cierta g que solo usa los $dp(j, k')$ con $k' < k$
- Podemos definir $I(n, k)$ como el menor i en donde se alcanza el mínimo de la expresión para $dp(n, k)$
- Para escribir las complejidades, notaremos $|I|$ a la cantidad de valores posibles para i en la expresión. Típicamente $|I|$ es aproximadamente N

Condición de Divide and Conquer

- $I(n,k) \leq I(n+1,k)$
- Equivalentemente: I es **monótona en n**.
- En criollo para DPs de particionar: si para k fijo agrando el rango, el último punto de corte también (mejor dicho: no retrocede)
- Llamamos a la anterior la *Condición de Divide and Conquer*
- Igual que antes, suele ser mucho más difícil demostrar que se cumple, que convencerse o intuir que así será

Optimización de Divide and Conquer

- Esta optimización no es tan simple de implementar como la de Knuth, pero la idea también es sencilla.
- Supongamos que para calcular todos los $dp(n, k)$ para k fijo, calculamos primero el $dp(n', k)$:
  - Para los $n > n'$, alcanza con probar el i **desde** $I(n', k)$
    Es decir, no más de $L_1 = |I| - I(n', k) + 1$ valores
  - Para los $n < n'$, alcanza con probar el i **hasta** $I(n', k)$
    Es decir, no más de $L_2 = I(n', k) + 1$ valores
- Esto nos parte el rango $[0, N)$ que debíamos calcular en dos restantes: $[0, n')$ y $[n' + 1, N)$.
- En la primera parte hay que probar hasta L_1 valores, y en la segunda hasta L_2 valores.

Si profundizamos...

- Podemos seguir partiendo estos rangos en dos recursivamente
- En el paso k tendremos 2^k rangos, cada uno con hasta L_i opciones factibles para el i.
- Observación: En cada paso, los L_i suman $O(N + |I|)$
- Por lo tanto, procesar cada paso es $O(N + |I|)$
- Partiendo siempre a la mitad, serán $O(\lg N)$ pasos y la complejidad es $O((N + |I|) \lg N)$
- El algoritmo final cuesta $O(K(N + |I|) \lg N)$ evaluaciones de g

Ejercicio

- Ejercicio: Verificar que la condición de Divide and Conquer aplica en la recursión que vimos antes
- Pasa lo mismo que antes: Es más fácil intuirlo que probarlo

Resumen

Optimización de Knuth:

- Prototípico para DPs en rangos
- Punto óptimo debe ser monótono en ambas variables
- Requiere cuadrados o "similares": $O(N)$ borde, $O(N^2)$ estados
- Le saca un $|I|$ (ejemplo: $N^3 \to N^2$)

Optimización de Divide and Conquer:

- Prototípico para DPs "de particionar"
- Punto óptimo debe ser monótono en una variable
- Cualquier forma (por ejemplo sirve aún si $K << N$)
- Cambia un $N|I|$ por $(N + |I|) \lg N$ (ejemplo: $KN^2 \to KN \lg N$)