Alcuni metodi poco noti

Problema 1: Vengono dati alcuni sottoinsiemi "belli" di $\{0, 1, \dots, n-1\}$. Per ogni sottoinsieme di $\{0, 1, \dots, n-1\}$, trova quanti sottoinsiemi belli contiene.

Limiti: $n \leq 20$.

Harder version: Contare quanti sottoinsiemi di $\{0, 1, \dots, n-1\}$ sono unione di sottoinsiemi belli.
(BONUS)

Soluzione naive: $O(2^n \times 2^n) = O(4^n)$ troppo lenta.

Soluzione un po' meno naive: Per ogni sottoinsieme S, scorro i suoi $2^{|S|}$ sottoinsiemi e conto i belli.

La complessità è data da

$$\sum_{S \subseteq \{0,1,\dots,n-1\}} 2^{|S|} = \sum_{k=0}^{n} \sum_{|S|=k} 2^k = \sum_{k=0}^{n} 2^k \, \#\{S \subseteq \{0,\dots,n-1\} : |S|=k\}$$

$$= \sum_{k=0}^{n} 2^k \binom{n}{k} = (2+1)^n = 3^n$$

$O(3^n)$. $\sim 5 \cdot 10^9$

DP on subsets: L'idea è "aggiustare" un bit alla volta.

Dato $S \subseteq \{0,1,\dots,n-1\}$; sia $f_k(S) = \#\{T \subseteq S : T$ è bello

$T \cap \{k, k+1, \dots, n-1\}$
"
$S \cap \{k, k+1, \dots, n-1\}\}$

"T può essere diverso da S solo sui primi k elementi"

Notare che $f_n(S)$ è la risposta

$$f_0(S) = \begin{cases} 0 & \text{se } S \text{ non è bello} \\ 1 & \text{se } S \text{ è bello} \end{cases}$$

- $f_{k+1}(S) = f_k(S)$ se $k \notin S$ (infatti $k \notin T \subseteq S$
  $\Rightarrow$ se $T = S$ su $\{k+1, \dots, n-1\}$
  $\Rightarrow$ $T = S$ su $\{k, k+1, \dots, n-1\}$)

- Se $k \in S \Rightarrow f_{k+1}(S) = \overbrace{f_k(S)}^{k \in T} + \overbrace{f_k(S \setminus \{k\})}^{k \notin T \subseteq S \text{ bello}}$

In ogni caso ho transizione $O(1)$ $\Rightarrow$ La complessità è $O(n 2^n)$.

Remark: Posso fare tutto in-place (cioè creare solo un array f)

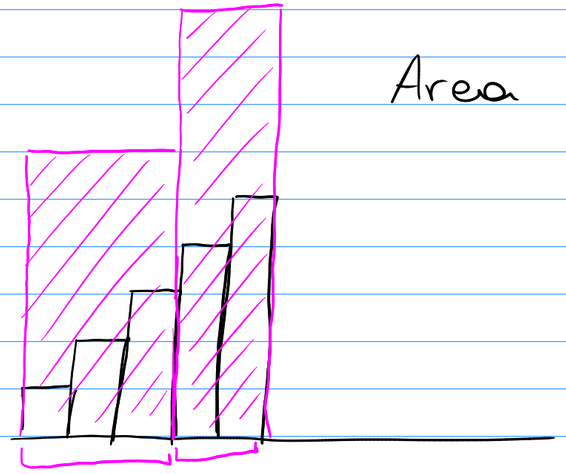
Problema 2: Viene dato un istogramma con N barre alte $a_0, a_1, \dots, a_{N-1}$. Volete semplificarlo e farlo diventare un istogramma con al massimo K barre.

Quanto è l'area minima dell'istogramma risultante

Limiti: $N \le 100{,}000$
$K \le 10$

Esempio:

$n=5, k=2$



Area minima possibile = 36.

(per iniziare, Cercate una soluzione $O(kn^2)$.)

Soluzione $O(KN^2)$: classica dinamica.

Sia $diu_k(n)$ la risposta al problema con $a_0, \dots, a_{n-1}$ e al massimo $k \le K$ barre alla fine ($n \le N$).

$$diu_1(n) = n \cdot (a_0 + a_1 + \dots + a_{n-1})$$

$$diu_{k+1}(n) = \min_{0 \le i < n} diu_k(i) + (n-i)(a_i + a_{i+1} + \dots + a_{n-1}). \quad (*)$$

D&C dp ottimizzazione: (obiettivo $O(KN \log(N))$.

Fissiamo k; sia i_n il minimo indice che realizza il minimo in $(*)$.

Claim: $i_n \le i_{n+1}$ (molto intuitivo, ma va dimostrato)

proof: Prendo $i < i_n$ e mostro che, anche quando calcolo $div_{k+1}(n+1)$ di certo i_n è meglio di i. (chiamo $f(n) = div_k(n)$)

Goal: $f(i) + (n+1-i)(a_i + a_{i+1} + \dots + a_n) > f(i_n) + (n+1-i_n)(a_{i_n} + \dots + a_n)$.

$\Longleftrightarrow$ $f(i) - f(i_n) > (n+1-i_n)(a_{i_n} + \dots + a_n) - (n+1-i)(a_i + \dots + a_n)$. ★

So che vale per $div_{k+1}(n)$:

$f(i) - f(i_n) > (n-i_n)(a_{i_n} + \dots + a_{n-1}) - (n-i)(a_i + \dots + a_{n-1})$.

Avremo finito se ★ $\le$ ☼

$a = n - i_n$
$b = n - i$
$c = a_{i_n} + \dots + a_{n-1}$
$d = a_i + \dots + a_{n-1}$
$x = a_n \ge 0$

$(a+1)(c+x) - (b+1)(d+x) \le ac - bd$

$ac + ax + c + x - bd - bx - d - x \le ac - bd$

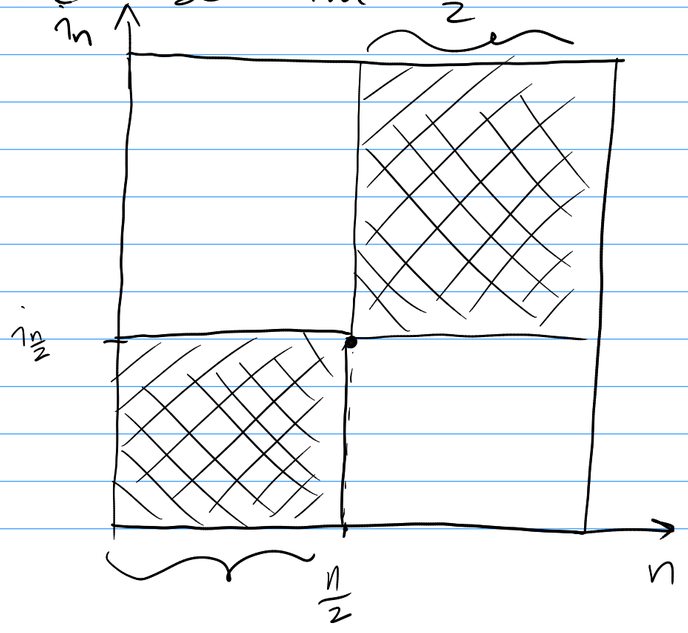
$(a-b)x + c - d \le 0$ che è vera.

A questo punto sappiamo $i_n \leq i_{n+1}$.

(*) $$div_{k+1}(n) = \min_{0 \leq i < n} div_k(i) + \underbrace{(n-i)(a_i + a_{i+1} + \dots + a_{n-1})}_{f(i,n)}.$$

Idea: Inizio dal calcolare $div_{k+1}\left(\frac{n}{2}\right)$ in $O(n)$ e trovo $i_{\frac{n}{2}}$.

Ora so che se $m < \frac{n}{2} \Rightarrow i_m \leq i_{\frac{n}{2}}$; $m > \frac{n}{2} \Rightarrow i_m \geq i_{\frac{n}{2}}$.



Pagando $O(n)$ avete "dimezzato" l'area dove vive il grafico degli $i_0, i_1, \dots i_{N-1}$.

Funzione ricorsiva (k fissato)

solve(l, r, i_l, i_r): $m = \frac{l+r}{2}$

Calcola i_m in $O(i_r - i_l)$

Ricorre su

solve(l, m, i_l, i_r), solve(m, r, i_m, i_r)

La complessità della transizione $k \mapsto k+1$ è $O(N \log N)$

$\Rightarrow$ Complessità totale $O(NK \log(N))$.

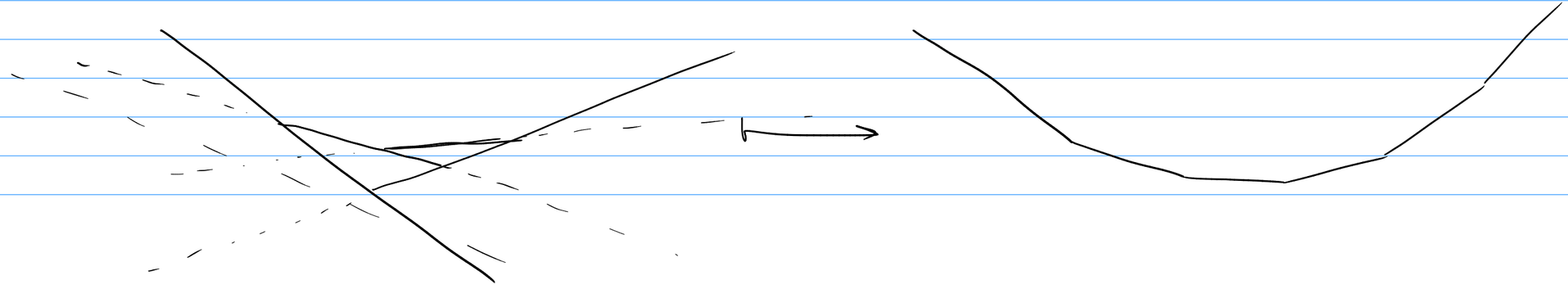
Problema 3: Avete n polinomi quadratici $P_i(x) = a_i x^2 + b_i x + c_i$.
Dovete rispondere a m domande.
La k-esima domanda è: dato x_k, trova il massimo di $P_0(x_k), P_1(x_k), \dots, P_{n-1}(x_k)$

Limiti: $|a|, |b|, |c| \le 10^6$, $|x| \le 10^6$

$n, m \le 2 \cdot 10^5$.

Versione più facile: $a_i = 0$ per ogni i.

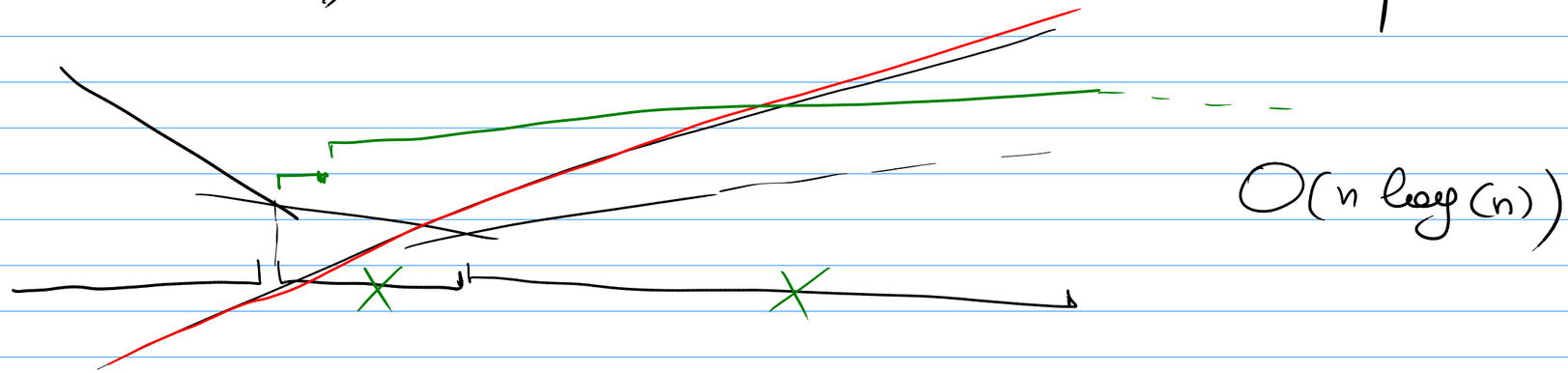
Versione facile: Disegniamo le funzioni (sono rette)



Nel caso facile, 2 modi:

modo 1: Assumo $b_i \leq b_{i+1}$

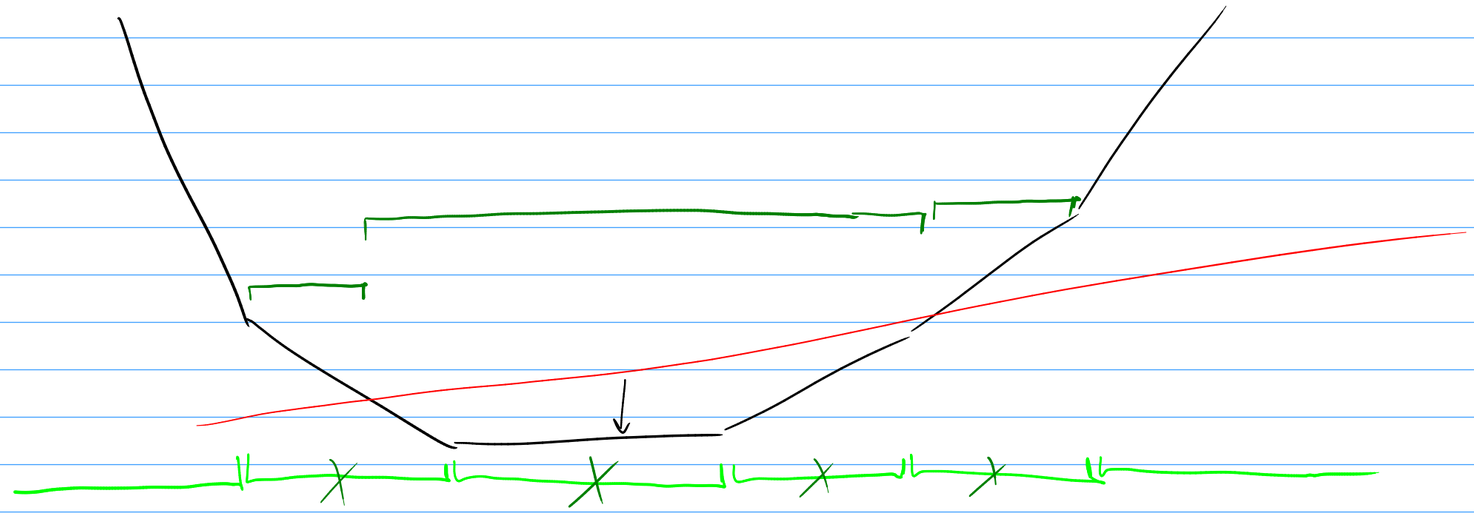
E poi tengo aggiornato il massimo come una successione di intervalli, in ogni intervallo una retta è la maggiore
Osservazione è che quando prendo una nuova retta, devo "solo" buttare fuori dal fondo un po' di rette



$O(n \log(n))$

modo 2: Non chiedo nessun ordinamento; prendo le rette una alla volta, tengo aggiornata una struttura che

tiene intervalli dove una singola retta "governa"



Osservazione: Aggiungere una retta rimuove intervalli consecutivi (e ne aggiunge al massimo 3).
Si può trovare con una ~~binary~~ ternary search un intervallo da rimuovere.

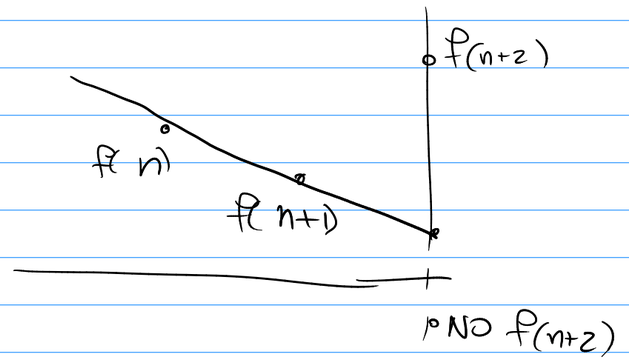
$O(n \log(n))$

Soluzione con polinomi quadratici:

Funzione convessa:

Definizione: f definita sui numeri interi è convessa se Δf è crescente

$(\Delta f)(n) = f(n+1) - f(n)$.



Fatto: Si può trovare il minimo di f convessa con una binary search;

n è il punto di minimo $\iff$ è il primo valore tale $\underline{\Delta f(n) \geq 0}$.

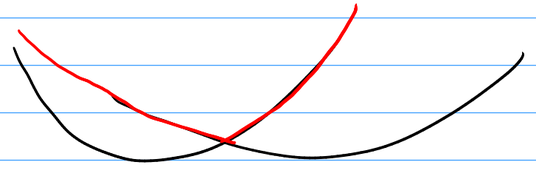
binary search su Δf.

Fatto: Il massimo è cercato agli estremi.

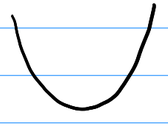
Fatti: f convessa $\Rightarrow$ $\lambda \cdot f$ convessa $(\lambda > 0)$

f, g convesse $\Rightarrow$ $f+g$ convessa

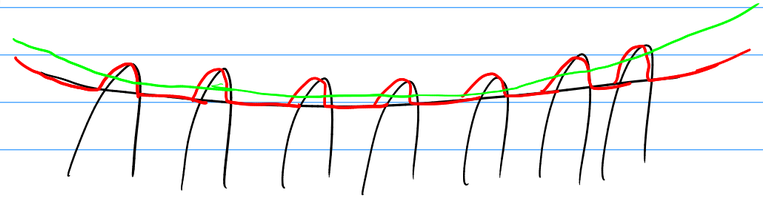
$\max(f, g)$ convessa



Le rivoluzioni sono convesse.

$n \longmapsto n^2$ è convessa  $(\Delta(n^2) = 2n+1$ crescente ✓.$)$

La difficoltà è che aggiungere un polinomio può cambiare drasticamente la struttura del $\max(p_0, \dots, p_{k-1})$



Idea: Assumo a_i siano decrescenti.

Processo in quest'ordine i polinomi e tengo aggiornata $\max(p_0, p_1, \dots, p_{k-1})$. ← Sequenza di intervalli dove governa un certo polinomio.

Dove governa p_k? $\quad p_k > \max(p_0, \dots, p_{k-1}) \iff$

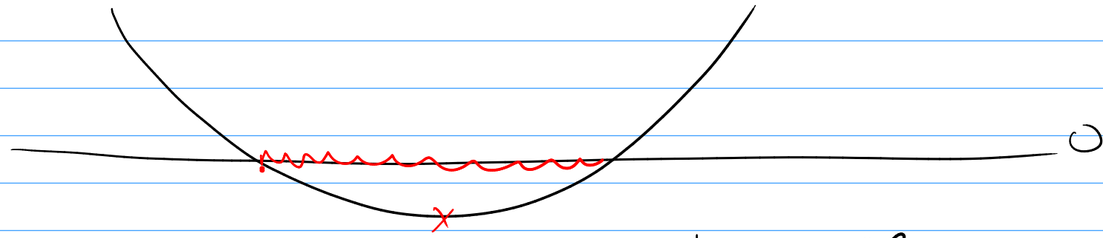
$$0 > \max(p_0 - p_k, p_1 - p_k, \dots, p_{k-1} - p_k)$$

Osservazione cruciale: $p_i - p_k$ è convesso

$$p_i - p_k = \underbrace{(a_i - a_k)}_{<0} \underbrace{x^2}_{\text{convesso}} + \underbrace{(\text{lineare})}_{\text{convesso}} \Big\} \text{ convesso}$$

$\Rightarrow \underbrace{\max(p_0 - p_k, p_1 - p_k, \dots, p_{k-1} - p_k)}_{f}$ è convesso

Devo "solo" vedere dove f è negativa



$O(1)$ ammortizzato elimino intervalli e alla fine ne devo aggiungere al più 3. $\Rightarrow$ Soluzione $O(n \log(n))$

Problema 4: Esattamente come il problema 3, ma le query vengono fatte durante l'aggiunta dei polinomi

Static to dynamic trick: Black box $B(\mathcal{F})$

$\mathcal{F}$: famiglia di polinomi

che risponde a query $\max_{p \in \mathcal{F}} p(x) \longleftarrow x$.

$B(\mathcal{F})$ si costruisce in $O(|\mathcal{F}| \log |\mathcal{F}|)$.

L'idea è la seguente, tengo vari black box.

Inizialmente $\mathcal{F}(\{p_0\})$; $\mathcal{F}(\{p_0, p_1\})$

Ricalcolo tutto

$\mathcal{F}(\{p_0, p_1\})$; $\mathcal{F}(\{p_2\})$

$\mathcal{F}(\{p_0, p_1, p_2, p_3, p_4, p_5, p_6, p_7\})$

L'idea è tenere $O(\log(n))$ black-box con dimensione potenze di 2, esattamente come nella scrittura in base 2 di n.

Immaginiamo di aver processato k polinomi,

$$k = 2^7 + 2^4 + 2^3 + 2^2$$

$$B(\{p_0, p_1, \dots, p_{2^7-1}\}); \quad B(\{p_{2^7}, p_{2^7+1}, \dots, p_{2^7+2^4-1}\})$$

$$\dots \quad B(\{\underbrace{p_{2^7+2^4+2^3}, p_{2^7+2^4+2^3+1}, \dots, p_{k-1}}_{2^2 \text{ elementi}}\})$$

Se devo rispondere ad una query: faccio la query su tutti i $B(\{\})$ e prendo il massimo.

E se devo aggiungere p_k ? $B(\{p_k\})$ $k+1 = 2^7 + 2^4 + 2^3 + 2^2 + 2^0$

// // // p_{k+1} ? $B(\{p_k, p_{k+1}\})$ // $+ 2^1$

p_{k+2} ? $B(\{p_{k+2}\})$ // $+ 2^1 + 2^0$

p_{k+3} ? $B(\{p_{2^7}, p_{2^7+1}, \dots, p_{k+3}\})$ $k+4 = 2^7 + 2^5$

Rispondere alle query: $O(\log(n) \cdot$ "complessità statica$") = O(\log(n)^2)$.

Ricostruire la struttura? Quando aggiungo p_{k-1}, pago $2^{e_{\log(n)}}$ dove 2^e è la massima potenza di 2 che divide k.

Complessità totale: $\sum_{k=1}^{n} 2^{v_2(k)} \log(n) = \sum_{e=0}^{\log_2(n)} 2^e \underbrace{\#\{k \le n : v_2(k) = e\}}_{\frac{n}{2^e}} \log(n)$

$= \sum^{\log(n)} \log(n)\, n = n \log(n)^2$ ☺

$O(n \log^2(n))$ ☺

Morale: Statico → Dinamico pagando un logaritmo.