Oszd meg és uralkodj elv

Feladatok, megoldások

Szerző: Nikházy László
Előadó: Németh Zsolt

2025. október 23.

N1. Maximális haszon

Egy vállalat részvényeinek napi árfolyamát ismerjük egy N elemű tömbben: $A[1], A[2], \ldots, A[N]$, ahol $A[i]$ az i-edik nap záróára. Szeretnénk egyetlen alkalommal **vásárolni**, majd később **eladni** pontosan egy darab részvényt. A cél a **lehető legnagyobb haszon** elérése.

Formálisan: a keresett érték

$$\max_{1 \le i < j \le N} (A[j] - A[i])$$

Példa

Bemenet	Kimenet
6 7 1 5 3 6 4	5

Korlátok

$1 \le N \le 200\,000, \quad 1 \le A[i] \le 10^9$

N1. Maximális haszon – Megoldás (Divide and Conquer)

Ötlet: Osszuk a napokat két egyenlő részre. A legjobb vétel–eladás pár háromféleképp alakulhat:

- Mindkét nap a **bal** részben van ⇒ rekurzív megoldás balra
- Mindkét nap a **jobb** részben van ⇒ rekurzív megoldás jobbra
- A vétel balról, az eladás jobbról jön ⇒ haszon = $\mathtt{max}(\text{jobb rész}) - \mathtt{min}(\text{bal rész})$. A bal oldalon a minimumot, a jobb oldalon a maximumot kell megkeresni ⇒ $O(n)$ idő.

Futásidő:

$$T(n) = 2T\left(\frac{n}{2}\right) + O(n) \Rightarrow T(n) = O(n \log n)$$

Megjegyzés: Iteratív ($O(n)$) megoldás: egyetlen bejárás során követjük az eddigi minimumot és az aktuális különbséget. DE: van rekurzív $O(n)$ megoldás is.

N1. Maximális haszon – Megoldás (Lineáris D&C)

Ötlet: A korábbi rekurzív megoldásban a kombinálás $O(n)$ időt vett igénybe. Ezt kiküszöbölhetjük, ha a részfeladat nemcsak a **maximális hasznot**, hanem a **minimum** és **maximum** árfolyamot is visszaadja.

Minden szegmensre: a rekurzió három értéket ad vissza:

$$(\text{maxHaszon}, \text{minÁr}, \text{maxÁr})$$

Kombinálás:

- maxHaszon = $\texttt{max}(\text{bal.maxHaszon}, \ \text{jobb.maxHaszon}, \ \text{jobb.maxÁr} - \text{bal.minÁr})$
- minÁr $= \texttt{min}(\text{bal.minÁr}, \ \text{jobb.minÁr})$
- maxÁr $= \texttt{max}(\text{bal.maxÁr}, \ \text{jobb.maxÁr})$

Alapeset: 1 elem esetén $(0, \ A[i], \ A[i])$

Minden lépés $O(1)$ idő, így

$$T(n) = 2T\left(\frac{n}{2}\right) + O(1) \Rightarrow T(n) = O(n)$$

N2. Majoráns elem

Van egy rejtett, N elemű tömb, amelynek elemei ismeretlenek, de tetszőleges két indexre, i és j, meg tudjuk kérdezni, hogy az elemek **azonosak-e**:

$$\texttt{query}(i, j) \Rightarrow \begin{cases} \texttt{true}, & \text{ha } A[i] = A[j], \\ \texttt{false}, & \text{különben.} \end{cases}$$

A tömbben létezhet egy **majoráns elem**, amely több mint $\frac{N}{2}$ alkalommal fordul elő.
A feladat: állapítsuk meg, **van-e majoráns elem**, és ha igen, **adjunk meg egy olyan indexet**, ahol ez az érték található.

Korlátok

$1 \leq N \leq 200\,000$
A lekérdezések száma legyen minél kevesebb!

Részprobléma: Adott résztömbben eldönteni, hogy van-e benne **majoráns elem**, és ha igen, megadni annak **egy indexét**.

Alapeset: 1 elemű tömbre a majoráns az egyetlen elem.

Kombinálás:

- Ha **nincs** egyik oldalon sem majoráns, akkor az összevont tömbben sem lehet.
- Ha **van** legalább az egyik oldalon majoráns, akkor az összevont tömb majoránsa csak az ottani jelölt(ek) közül kerülhet ki.
    - Vizsgáljuk meg a két jelöltet (ha mindkettő létezik, akár azonosak is lehetnek): minden elemre lekérdezzük, hogy egyezik-e velük, és megszámoljuk az előfordulásaikat.
    - A tömb majoránsa az a jelölt, amely több mint $N/2$ alkalommal fordul elő, különben `nincs majoráns`.

Futásidő és lekérdezésszám:

$$T(n) = 2T\left(\frac{n}{2}\right) + O(n) \quad \Rightarrow \quad T(n) = O(n \log n)$$

Megjegyzés: Van lineáris megoldás is (*Boyer–Moore Majority Vote*).

N2. Majoráns elem – Boyer–Moore Majority Vote

Ötlet: Egyetlen végigjárás során kiválasztható egy **jelölt**, amely – ha létezik majoráns – biztosan az lesz. A módszer csak egyenlőség-vizsgálatokat igényel, így a „rejtett tömb” modellben is használható.

Algoritmus:

1. Inicializáljunk egy-egy változót: jelölt m, számláló $c = 0$
2. Minden indexre $i = 1, 2, \ldots, N$:
   - ha $c = 0$, akkor $m = i$, $c = 1$
   - különben, ha $\texttt{query}(i, m)$ igaz, akkor $c = c + 1$
   - különben $c = c - 1$
3. A bejárás végén m jelöli a lehetséges majoránst.
4. Második bejárásban ellenőrizzük: számoljuk meg, hány i-re igaz $\texttt{query}(i, m)$, és ha ez $> N/2$, akkor $A[m]$ valóban majoráns.

Tulajdonságok:

- Idő- és lekérdezésszám: $O(N)$
- Tárigény: $O(1)$
- A második bejárás szükséges a hitelesítéshez

N3. Permutation Graph

`https://codeforces.com/contest/1696/problem/D`

Adott az $1..n$ számok egy permutációja: $a = [a_1, a_2, \ldots, a_n]$. Hozzunk létre egy **gráfot** n **csúccsal** $(1, 2, \ldots, n)$. Minden $1 \le i < j \le n$ párra tegyünk élt i és j közé, ha az

$$mn(i,j) = \min_{k=i..j} a_k, \quad mx(i,j) = \max_{k=i..j} a_k$$

értékekre teljesül, hogy $(mn(i,j), mx(i,j)) = (a_i, a_j)$ vagy (a_j, a_i).

Milyen hosszú a **legrövidebb út** a gráfban 1-ből n-be?

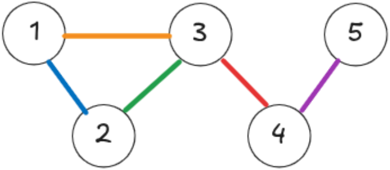
Korlátok

$1 \le n \le 2.5 \cdot 10^5$

Példa

Bemenet	Kimenet
5 1 4 5 2 3	3

Magyarázat: az alábbi ábrán be vannak jelölve azok az (i, j) intervallumok, amelyekre igaz, hogy a minimum és maximum a két szélén van, és ez alapján látható a létrejövő gráf, amiben az 1. és 5. csúcs távolsága 3.



N3. Permutation Graph – Megoldás (Divide and Conquer)

Kulcsötlet: A $[1..n]$ intervallumban az **maximum elem pozícióját** (jelölje k) minden $1 \to n$ útnak tartalmaznia kell. Ha $a_k = n$, akkor bármely $x < k < y$ esetén nincs közvetlen él x és y között, hiszen az $[x..y]$ intervallumban a maximum mindig a_k.

Következmény: Az $1 \to n$ legrövidebb út mindig a k csúcson halad át. Ezért:

$$\text{dist}(1, n) = \text{dist}(1, k) + \text{dist}(k, n)$$

Rekurzió:

- Oldjuk meg rekurzívan: $\text{dist}(1, k)$ és $\text{dist}(k, n)$.
- Bal oldalon ($[1..k]$) jelölje j a minimum indexét. Egyrészt a fentihez hasonló megfontolásból minden $1 \to k$ út keresztül megy rajta. Másrészt látható, hogy van él j és k között, ezért $\textit{dist}(j, k) = 1$, vagyis a rekurziót csak a $\textit{dist}(1, j)$-re kell alkalmazni (ahol majd ismét maximumot keresünk).
- Jobb oldalon ($[k..n]$) ugyanúgy kell megoldani.
- A rekurzió csak a **prefix** és **szuffix minimum/maximum** értékeket használja, ezeket könnyen ki tudjuk számítani előre.

Időkomplexitás: $O(n)$

N3. Permutation Graph – Kód

```
const int N = 250000;
int n, p[N], prefmin[N], prefmax[N], sufmin[N], sufmax[N];

void precalc() {
  prefmin[0] = prefmax[0] = 0;
  for (int i = 1; i < n; i++) {
    prefmin[i] = p[i] < p[prefmin[i - 1]] ? i : prefmin[i - 1];
    prefmax[i] = p[i] > p[prefmax[i - 1]] ? i : prefmax[i - 1];
  }
  sufmin[n - 1] = sufmax[n - 1] = n - 1;
  for (int i = n - 2; i >= 0; i--) {
    sufmin[i] = p[i] < p[sufmin[i + 1]] ? i : sufmin[i + 1];
    sufmax[i] = p[i] > p[sufmax[i + 1]] ? i : sufmax[i + 1];
  }
}

int dist_left(int i) {
  if (i == 0) return 0;
  if (prefmin[i] == 0) return 1;
  return dist_left(prefmax[prefmin[i]]) + 2;
}
int dist_right(int i) {
  if (i == n - 1) return 0;
  if (sufmin[i] == n - 1) return 1;
  return dist_right(sufmax[sufmin[i]]) + 2;
}

void solve() {
  cin >> n;
  for (int i = 0; i < n; i++) cin >> p[i];
  precalc();
  cout << dist_left(prefmax[n - 1]) + dist_right(prefmax[n - 1]) << "\n";
}
```

E1. Részhalmaz összegek (Sum over Subsets)

Adott egy 2^n elemű tömb: $a_0, a_1, \ldots, a_{2^n-1}$, amelynek indexeit bináris alakban, bitmaszkokként képzeljük el. Azt szeretnénk kiszámítani, hogy minden i bitmaszkhoz mi az

$$A_i = \sum_{i \text{ fedi } j\text{-t}} a_j,$$

vagyis az összeg azon a_j értékekre, amelyekre j minden 1-es bite az i-ben is 1. Például $n = 2$ esetén:

$$A_0 = a_0, A_1 = a_0 + a_1, A_2 = a_0 + a_2, A_3 = a_0 + a_1 + a_2 + a_3$$

Korlátok

$1 \leq n \leq 20$
$|a_i| \leq 10^9$

Ötlet (legfelső bit szerinti felosztás):

- Osszuk a maszkokat két részre a legmagasabb bit szerint:
  - *alsó fél*: indexek $0..H-1$ (felső bit = 0),
  - *felső fél*: indexek $H..2H-1$ (felső bit = 1), ahol $H = 2^{n-1}$.
- Rekurzívan számoljuk ki, a felső bit elhagyásával a két félre:

$$A^L = f(a[0..H-1]), \quad A^R = f(a[H..2H-1]) \quad \text{(mindkettő } n-1 \text{ biten)}$$

- **Egyesítés (lineáris):** minden $t \in [0..H-1]$-re

$$A[t] = A^L[t], \qquad A[H+t] = A^L[t] + A^R[t].$$

Indoklás: ha a felső bit 0, akkor a részhalmazok felső bitje is 0 (csak az alsó félből jön hozzájárulás); ha a felső bit 1, akkor a részhalmazok két csoportra esnek (felső bit 0 vagy 1) → összeadódik az alsó és a felső fél eredménye.

Időkomplexitás: $T(N) = 2T\left(\frac{N}{2}\right) + O(N) \implies T(N) = O(N \log N)$, ahol $N = 2^n$ a tömb hossza.

E2. Rendezett intervallumok tartalmazása

Adott n darab intervallum: $[l_1, r_1], [l_2, r_2], \ldots, [l_n, r_n]$. Számoljuk meg azokat az (i, j) párokat, amelyekre:

$$i < j, \quad l_i \leq l_j, \quad r_i \geq r_j.$$

Vagyis a kisebb indexű intervallum teljesen tartalmazza a nagyobb indexűt.

Példa

Bemenet	Kimenet
5 1 7 2 5 -3 6 -4 0 5 5	4

A példában a négy érvényes (i,j) pár: (1, 2), (1, 5), (2, 5), (3, 5).

Korlátok

$1 \leq n \leq 200\,000; \quad -10^9 \leq l_i \leq r_i \leq 10^9.$

Előzmény: Ha nincs $i < j$ feltétel akkor l szerint rendezve söpréssel megoldható („alapeset”).

Ötlet (Divide & Conquer):

- Osszuk két részre az intervallumokat index szerint: *bal* és *jobb* félre.
- **Rekurzív lépés:** Számoljuk meg a bal és jobb félen belüli párokat külön-külön.
- **Egyesítés (Conquer lépés):**
    - Csak azokat a párokat kell kezelni, ahol i bal, j jobb oldalon van.
    - Ehhez az alapesettel analóg **söpréses** megoldást használunk:
        - rendezzük a két fél intervallumait l szerint növekvően;
        - ahogy haladunk rajtuk végig, egy adatszerkezetben (pl. *szegmensfa*, *Fenwick-fa*) tároljuk a bal oldali r értékeket;
        - minden jobb oldali j esetén megszámoljuk, hány tárolt $r_i \geq r_j$ (mert ekkor i tartalmazza j-t).

Futásidő: $T(n) = 2T\left(\frac{n}{2}\right) + O(n \log n) \quad \Rightarrow \quad T(n) = O(n \log^2 n)$.

Megjegyzés: Nem rekurzív megoldás is létezik 2D-s Fenwick-fával vagy szegmensfával, de az $O(n \log^2 n)$.

E3. Intrinsic Interval

`https://codeforces.com/gym/101620` – Problem I

Adott az $1..n$ számok egy **permutációja**: $\pi = (\pi_1, \pi_2, \ldots, \pi_n)$
Egy **valódi intervallum** a permutációban olyan *folytonos szakasz*, amelynek elemeit növekvő sorrendbe rendezve **egymást követő számokat** kapunk.

Példa: $\pi = (3, 1, 7, 5, 6, 4, 2)$ esetén $\pi_3^6 = (7, 5, 6, 4)$ valódi intervallum (4–7 közötti számokat tartalmaz), de $\pi_1^3 = (3, 1, 7)$ nem az.

Feladat: Adott m darab szakasz: $\pi_{x_j}^{y_j}$. Mindegyikhez keressünk egy $\pi_{a_j}^{b_j}$ **legszűkebb valódi intervallumot**, ami tartalmazza azt ($a_j \leq x_j \leq y_j \leq b_j$).

E3. Intrinsic Interval – példa

Példa

Bemenet	Kimenet
7	3 6
3 1 7 5 6 4 2	7 7
3	1 7
3 6	
7 7	
1 3	

Korlátok

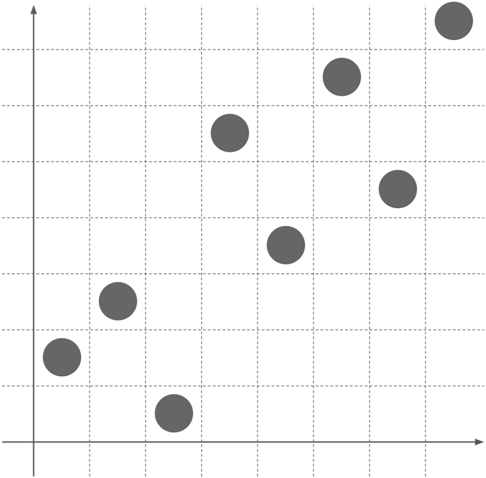
$1 \leq n, m \leq 100\,000$

$1 \leq \pi_i \leq n$, a számok páronként különbözőek.

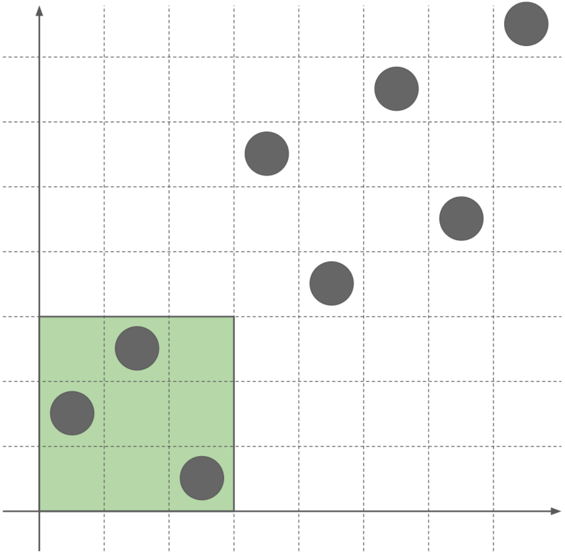
Időlimit: 3 másodperc **Memórialimit:** 512 MiB

E3. Intrinsic Interval – Megoldás

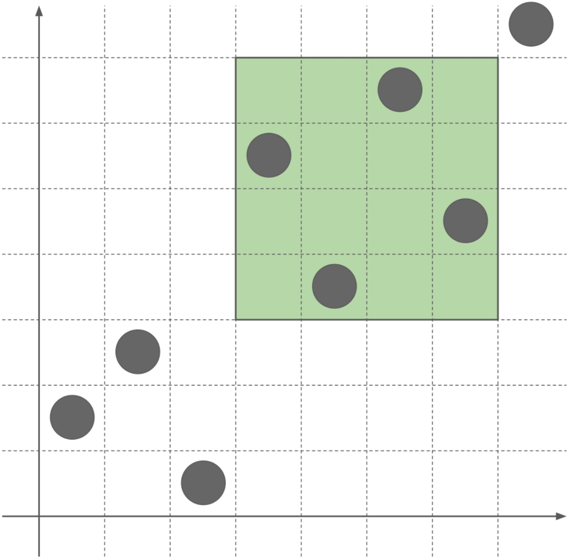
Ahhoz, hogy lássuk, hogyan kell bővíteni az intervallumokat, ábrázoljuk a permutációt két dimenzióban.



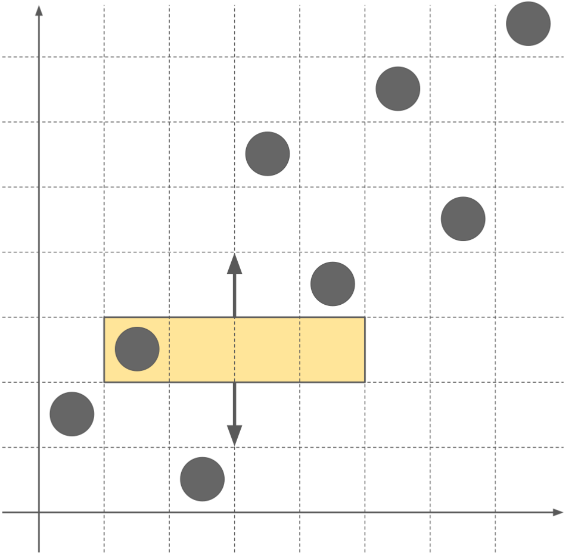
E3. Intrinsic Interval – Jó intervallum



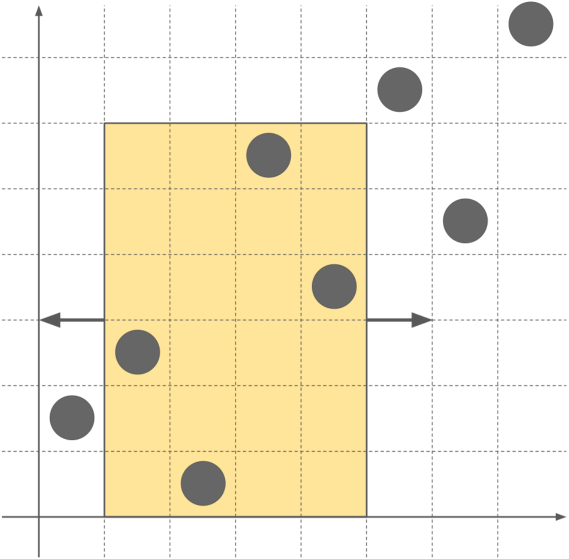
E3. Intrinsic Interval – Jó intervallum



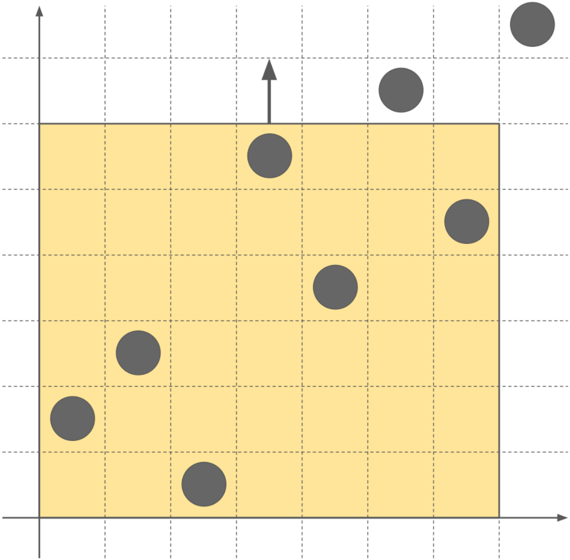
E3. Intrinsic Interval – Intervallum bővítés



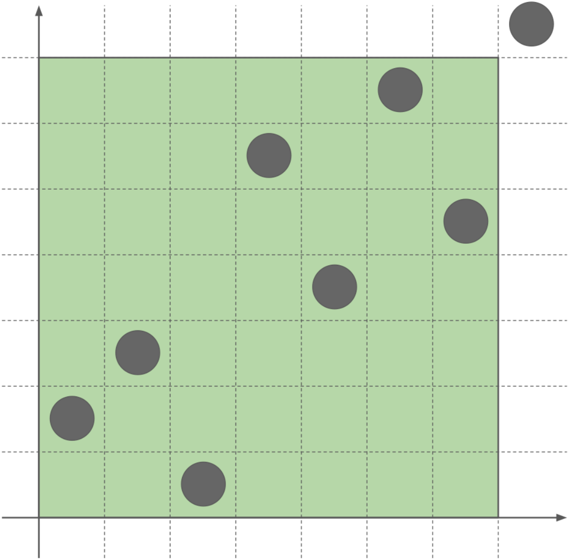
E3. Intrinsic Interval – Intervallum bővítés



E3. Intrinsic Interval – Intervallum bővítés



E3. Intrinsic Interval – Intervallum bővítés



E3. Intrinsic Interval – Divide & Conquer megoldás

Naiv megoldás: egy lekérdezést balra és jobbra bővítve $O(|b-a|)$ időben megoldható, de ez $O(nm)$ lenne – túl lassú.

Ötlet: Végezzünk **oszd meg és uralkodj** stratégiát, és **minden lekérdezést egyszerre** javítsunk.

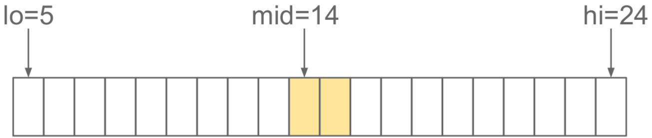
- Az összes lekérdezést induláskor $[1, n]$ intervallumra állítjuk.
- Rekurzív függvény: `Improve(queries, lo, hi)` - a $[lo, hi]$ ablakon belül próbálja szűkíteni a lekérdezések intervallumát.
- A függvény három részre oszt:
    - `lo == hi` esetben visszatér;
    - `mid = (lo + hi)/2`;
    - bal és jobb félen rekurzívan hívja magát;
    - majd a `ImproveViaMid` függvény kezeli azokat, amik átnyúlnak a `mid`-en.

E3. Intrinsic Interval – ImproveViaMid lépés

Központi ötlet: A `ImproveViaMid(queries, lo, mid, hi)` lépés azokat az intervallumokat vizsgálja, amelyek tartalmazzák a $[mid, mid + 1]$ pontpárt, és a $[lo, hi]$-on belül vannak.

- Kiindulás: $[mid, mid + 1]$ mint kezdő intervallum.
- **Balra** bővítjük, és minden új valódi intervallumot eltárolunk.

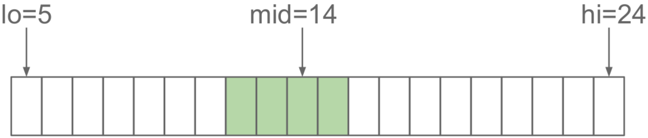
Bal intervallumok:



Központi ötlet: A `ImproveViaMid(queries, lo, mid, hi)` lépés azokat az intervallumokat vizsgálja, amelyek tartalmazzák a $[mid, mid+1]$ pontpárt, és a $[lo, hi]$-on belül vannak.

- Kiindulás: $[mid, mid+1]$ mint kezdő intervallum.
- **Balra** bővítjük, és minden új valódi intervallumot eltárolunk.

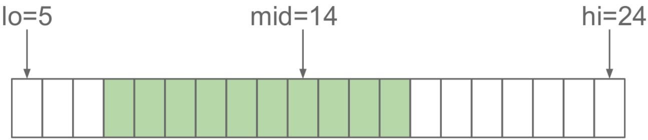
Bal intervallumok: $[12, 15]$,



Központi ötlet: A `ImproveViaMid(queries, lo, mid, hi)` lépés azokat az intervallumokat vizsgálja, amelyek tartalmazzák a $[mid, mid+1]$ pontpárt, és a $[lo, hi]$-on belül vannak.

- Kiindulás: $[mid, mid+1]$ mint kezdő intervallum.
- **Balra** bővítjük, és minden új valódi intervallumot eltárolunk.

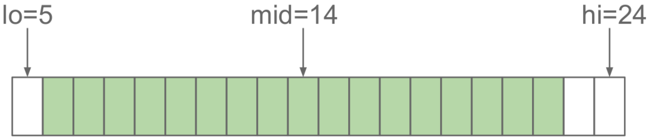
Bal intervallumok: $[12, 15]$, $[8, 17]$,



Központi ötlet: A `ImproveViaMid(queries, lo, mid, hi)` lépés azokat az intervallumokat vizsgálja, amelyek tartalmazzák a $[mid, mid+1]$ pontpárt, és a $[lo, hi]$-on belül vannak.

- Kiindulás: $[mid, mid+1]$ mint kezdő intervallum.
- **Balra** bővítjük, és minden új valódi intervallumot eltárolunk.

Bal intervallumok: $[12, 15]$, $[8, 17]$, $[6, 22]$



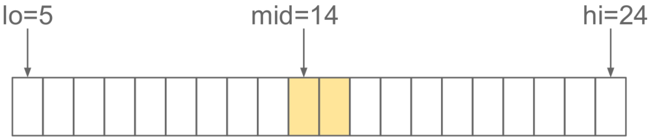
E3. Intrinsic Interval – ImproveViaMid lépés

Központi ötlet: A `ImproveViaMid(queries, lo, mid, hi)` lépés azokat az intervallumokat vizsgálja, amelyek tartalmazzák a $[mid, mid + 1]$ pontpárt, és a $[lo, hi]$-on belül vannak.

- Kiindulás: $[mid, mid + 1]$ mint kezdő intervallum.
- **Balra** bővítjük, és minden új valódi intervallumot eltárolunk.
- **Jobbra** is bővítjük hasonlóan.

Bal intervallumok: $[12, 15]$, $[8, 17]$, $[6, 22]$

Jobb intervallumok:



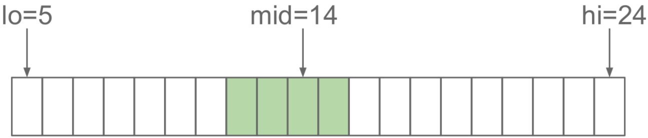
E3. Intrinsic Interval – ImproveViaMid lépés

Központi ötlet: A `ImproveViaMid(queries, lo, mid, hi)` lépés azokat az intervallumokat vizsgálja, amelyek tartalmazzák a $[mid, mid+1]$ pontpárt, és a $[lo, hi]$-on belül vannak.

- Kiindulás: $[mid, mid+1]$ mint kezdő intervallum.
- **Balra** bővítjük, és minden új valódi intervallumot eltárolunk.
- **Jobbra** is bővítjük hasonlóan.

Bal intervallumok: $[12, 15]$, $[8, 17]$, $[6, 22]$

Jobb intervallumok: $[12, 15]$,



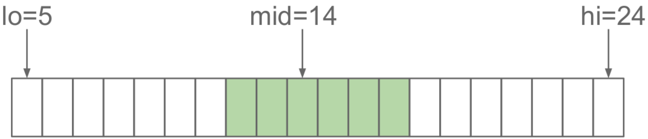
E3. Intrinsic Interval – ImproveViaMid lépés

Központi ötlet: A `ImproveViaMid(queries, lo, mid, hi)` lépés azokat az intervallumokat vizsgálja, amelyek tartalmazzák a $[mid, mid + 1]$ pontpárt, és a $[lo, hi]$-on belül vannak.

- Kiindulás: $[mid, mid + 1]$ mint kezdő intervallum.
- **Balra** bővítjük, és minden új valódi intervallumot eltárolunk.
- **Jobbra** is bővítjük hasonlóan.

Bal intervallumok: $[12, 15]$, $[8, 17]$, $[6, 22]$

Jobb intervallumok: $[12, 15]$, $[12, 17]$,

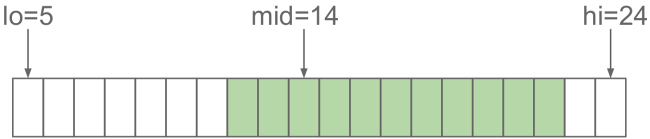


Központi ötlet: A `ImproveViaMid(queries, lo, mid, hi)` lépés azokat az intervallumokat vizsgálja, amelyek tartalmazzák a $[mid, mid + 1]$ pontpárt, és a $[lo, hi]$-on belül vannak.

- Kiindulás: $[mid, mid + 1]$ mint kezdő intervallum.
- **Balra** bővítjük, és minden új valódi intervallumot eltárolunk.
- **Jobbra** is bővítjük hasonlóan.

Bal intervallumok: $[12, 15]$, $[8, 17]$, $[6, 22]$

Jobb intervallumok: $[12, 15]$, $[12, 17]$, $[12, 22]$



E3. Intrinsic Interval – ImproveViaMid lépés

Központi ötlet: A `ImproveViaMid(queries, lo, mid, hi)` lépés azokat az intervallumokat vizsgálja, amelyek tartalmazzák a $[mid, mid+1]$ pontpárt, és a $[lo, hi]$-on belül vannak.

- Kiindulás: $[mid, mid+1]$ mint kezdő intervallum.
- **Balra** bővítjük, és minden új valódi intervallumot eltárolunk.
- **Jobbra** is bővítjük hasonlóan.

$$\text{Bal intervallumok: } [12, 15], [8, 17], [6, 22]$$

$$\text{Jobb intervallumok: } [12, 15], [12, 17], [12, 22]$$

- Minden lekérdezéshez megkeressük a legkisebb bal- és jobb-intervallumot, ami tartalmazza azt – ezek uniója adja az új, szűkebb megoldást.

E3. Intrinsic Interval – Összefoglalás

Pszeudokód:

```
def SolveAll(queries, lo, hi):
    if lo == hi:
        return
    mid = (lo + hi) // 2
    SolveAll([q for q in queries if q.b <= mid], lo, mid)
    SolveAll([q for q in queries if q.a > mid], mid + 1, hi)
    ImproveViaMid(queries, lo, mid, hi)


def ImproveViaMid(queries, lo, mid, hi):
    leftIntervals  = expandLeft(mid, lo, hi)
    rightIntervals = expandRight(mid + 1, lo, hi)
    for (a, b) in queries:
        bestLeft  = smallest in leftIntervals containing a
        bestRight = smallest in rightIntervals containing b
        ans[q] = union(bestLeft, bestRight)
```

Komplexitás: Minden lekérdezés legfeljebb $O(\log n)$ `ImproveViaMid` hívásban szerepel, így $O((n+m)\log n)$.

Extra feladatok

Elméletben:

Kiválasztás rendezett mátrixban

Egy $n \times n$ mátrix elemeit nem ismerjük, de lekérdezhetjük őket egyesével. A mátrix soronként és oszloponként is növekvően rendezett. Keressük meg a nagyság szerint k-adik elemet $O(k)$ kérdéssel.

Kódolásra is:

Minimum Sum
`https://codeforces.com/contest/120/problem/J`

Defender of Childhood Dreams
`https://codeforces.com/contest/1586/problem/F`

Meta-universe
`https://codeforces.com/contest/475/problem/F`