Gyökös felbontás módszer
(Square Root Decomposition)

szerző: Nikházy László
előadő: Gyimesi Péter

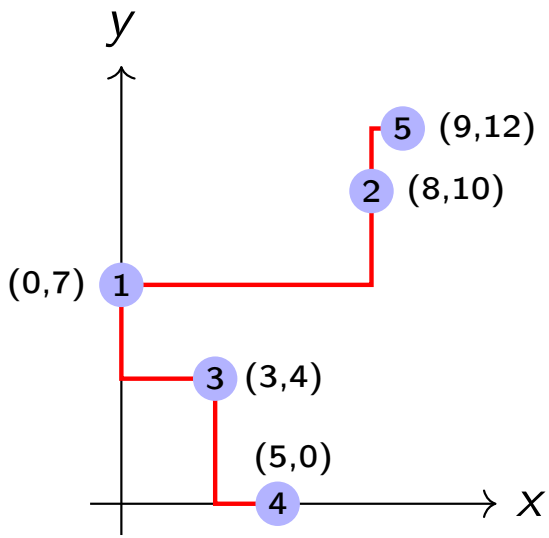
2025. szeptember 11.

1. feladat: Pontok a síkon (Codeforces - 576C)

Adott N $(1 \leq N \leq 10^6)$ különböző (x_i, y_i) pont egész koordinátákkal $(\mathbf{0 \leq x_i, y_i \leq 10^6}, 1 \leq i \leq N)$. Az a és b indexű pontok közötti Manhattan-távolság: $\texttt{dist}(a, b) = |x_a - x_b| + |y_a - y_b|$.
Egy Hamilton-út egy $p_1, \ldots, p_N$ permutáció a $\{1, \ldots, N\}$ számokból.
Az út hossza: $\sum_{i=1}^{N-1} \texttt{dist}(p_i, p_{i+1})$.
Adjunk meg bármilyen Hamilton-utat, amelynek hossza **legfeljebb** $\mathbf{25 \times 10^8}$. Nem kell a legrövidebb utat adni, és biztos van jó megoldás.

Példa

Bemenet	Kimenet
5 0 7 8 10 3 4 5 0 9 12	4 3 1 2 5



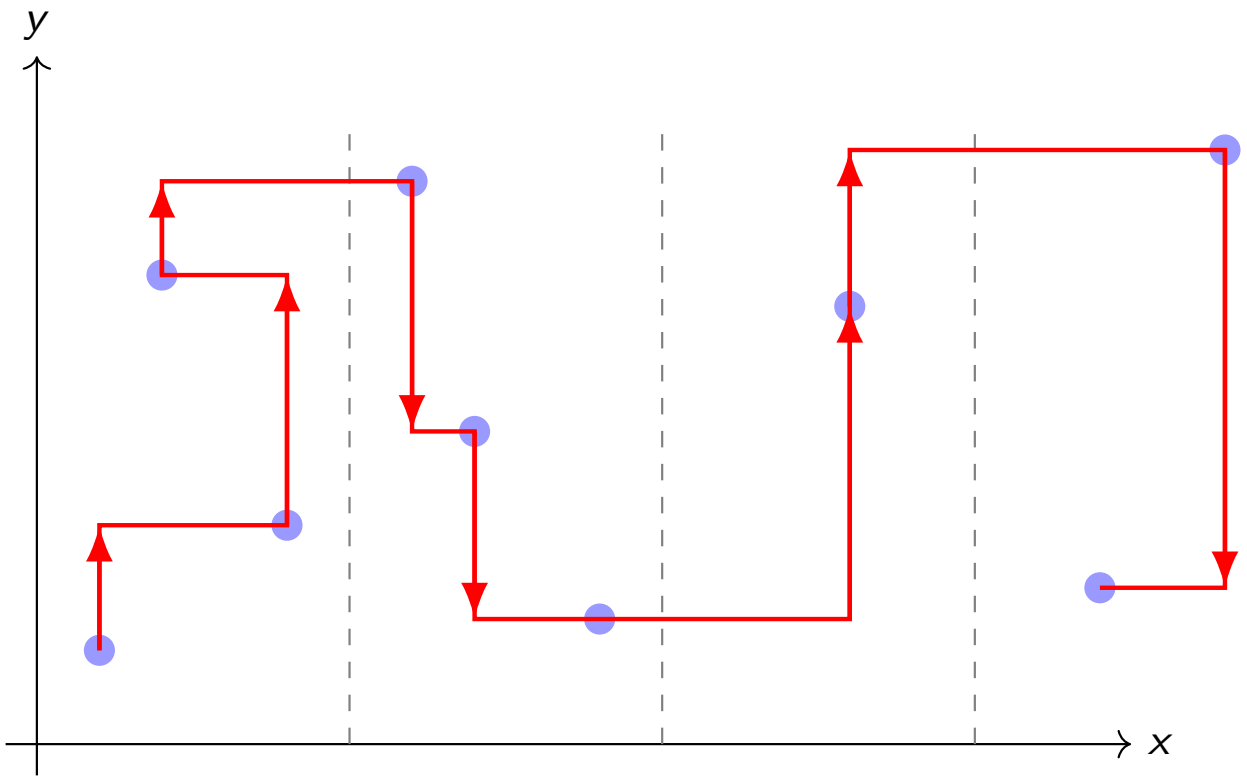
A példában a teljes hossz:
$\texttt{dist}(4, 3) + \texttt{dist}(3, 1) + \texttt{dist}(1, 2) + \texttt{dist}(2, 5) =$
$(|5-3| + |0-4|) + (|3-0| + |4-7|) + (|0-8| + |7-10|) + (|8-9| + |10-12|) = 26$

1. feladat: Pontok a síkon - Megoldási ötlet

Osszuk fel a $10^6 \times 10^6$-os négyzetet függőleges csíkokkal 1000 darab $10^3 \times 10^6$ téglalapra. Számozzuk meg a csíkokat balról jobbra, és haladjunk végig a pontokon csíkonként:

- páros sorszámú csíkban a pontokat y szerint növekvő sorrendben járjuk be,
- páratlan sorszámú csíkban pedig y szerint csökkenő sorrendben.

Így kígyózó módon végigmegyünk az összes ponton, és garantáltan kapunk egy megfelelő hosszúságú Hamilton-utat.
Meg tudod indokolni, hogy ez miért nem lesz hosszabb, mint $25 \cdot 10^8 = 2.5 \cdot 10^9$?



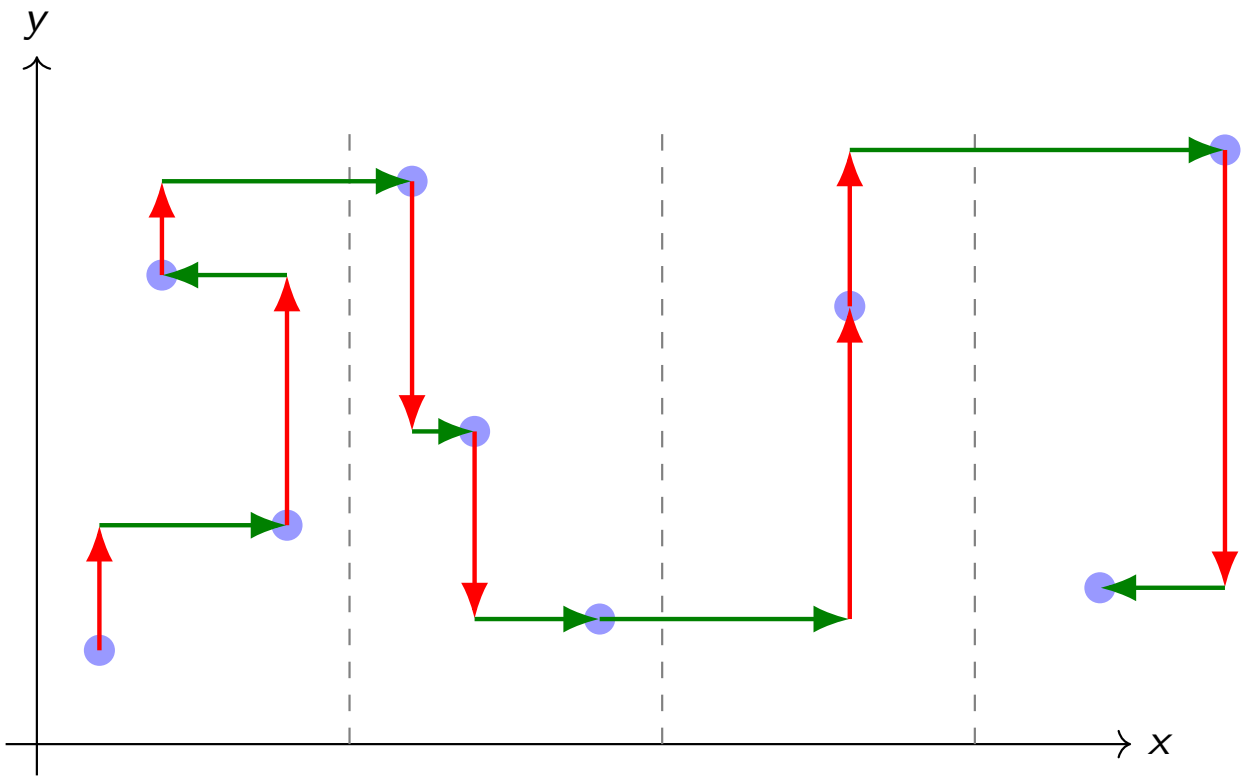
1. feladat: Pontok a síkon - Indoklás a helyességre

Vizsgáljuk meg a maximális út hosszát.

- A y koordináta szerint legfeljebb 10^6-ot mozgunk minden csíkban, és 1000 csík van. Ez összesen $\leq 10^9$.
- Az x koordináta szerint a csíkon belüli lépésekhez ≤ 1000, a következő csíkba átlépéshez pedig ≤ 2000 egységet használunk. Számolhatunk úgy, hogy pontonként ≤ 1000 plusz még csíkonként ≤ 1000 egységet mozgunk, ami összesen $\leq 10^9 + 10^6$.

Így a teljes útra: maximális hossz $\leq 10^9 + 10^9 + 10^6 \leq 2.5 \cdot 10^9$

Milyen algoritmussal csinálnád?



1. feladat: Pontok a síkon - Kód

Ötlet: Egyetlen rendezéssel állítjuk elő a megfelelő sorrendet. Az x koordinátát leosztjuk $C = 1000$-rel, így csíkon belül az x azonos lesz és csak az y koordináta számít: a csík paritásától függően növekvő vagy csökkenő y szerint rendezzük.

```
1  #include <bits/stdc++.h>
2  using namespace std;
3
4  const int C = 1000;
5  struct point { int x, y, id; };
6
7  int main() {
8      ios::sync_with_stdio(false); cin.tie(nullptr);
9      int n; cin >> n;
10     vector<point> pts(n);
11     for (int i = 0; i < n; i++) {
12         cin >> pts[i].x >> pts[i].y; pts[i].id = i;
13         pts[i].x /= C;
14     }
15     sort(pts.begin(), pts.end(), [](const point &a, const point &b) {
16         if (a.x != b.x) return a.x < b.x;
17         return (a.x % 2 == 0) ? (a.y < b.y) : (a.y > b.y);
18     });
19     for (auto &p : pts) cout << p.id + 1 << "␣";
20 }
```

2. feladat: Tömb lekérdezések (SPOJ - GIVE AWAY)

Adott egy N egész számot tartalmazó tömb ($A = A_1, A_2, \ldots A_N$), és Q lekérdezés ($1 \leq N, Q \leq 10^5, 1 \leq A_i \leq 10^9$). Kétféle lekérdezés van:

- **`0 i j c`**: Számoljuk meg, hány elem van az $[i, j]$ indextartományban, amely c-nél nagyobb vagy egyenlő.
- **`1 i c`**: A tömb i-edik elemét állítsuk c-re.

Bemenet: Első sor: N. Második sor: A tömb (N egész szám). Harmadik sor: Q. Következő Q sor: lekérdezések.

Példa

Bemenet	Kimenet
5	0
1 2 3 4 5	1
3	
0 1 5 10	
1 2 20	
0 1 3 10	

Magyarázat:

- 1. lekérdezés: nincs 10-nél nagyobb vagy egyenlő elem az 1-5 indexek között $\rightarrow$ 0
- 2. lekérdezés: a 2. elem 20-ra változik
- 3. lekérdezés: az 1-3 indexek között csak $A_2 \geq 10 \rightarrow 1$

2. feladat: Tömb lekérdezések - Megoldási ötlet

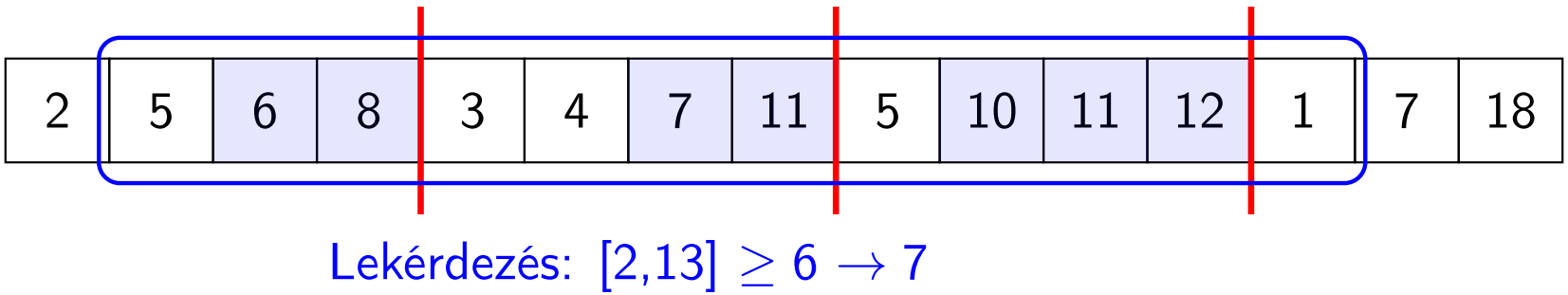
- A tömböt $\sqrt{N}$ darab $\sqrt{N}$ méretű **blokk**ra osztjuk.
- Minden blokkot külön-külön **rendezve** tárolunk.

Hogyan gyorsítja ez a kérdések megválaszolását? Mit kell csinálni módosításkor?

Példa: $A = [5, 2, 8, 6, 11, 7, 3, 4, 5, 12, 11, 10, 18, 1, 7]$

5	2	8	6	11	7	3	4	5	12	11	10	18	1	7

Blokkosítva, rendezve (blokkméret $= 4 \approx \sqrt{15}$):



Lekérdezés: $[2,13] \geq 6 \rightarrow 7$

2. feladat: Tömb lekérdezések - Megoldás és komplexitás

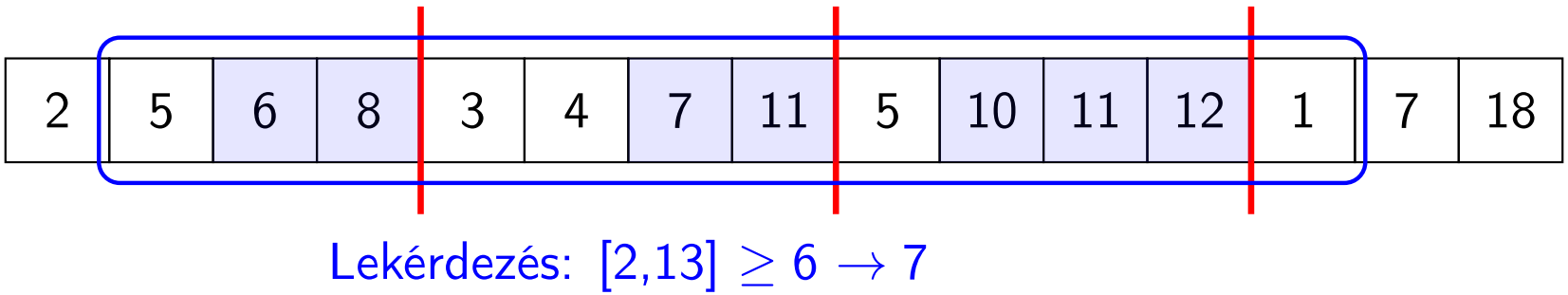
Lekérdezés ([i,j] tartományban hány $\geq c$):

- A tartomány elején és végén lévő két blokk ellenőrzése lineárisan: $O(\sqrt{N})$
- A tartományon belüli teljes blokkokból az elemek számolása bináris kereséssel: $O(\sqrt{N} \log \sqrt{N})$

Módosítás ($A[i] = val$):

- Az érintett blokk újrarendezése: $O(\sqrt{N} \log \sqrt{N})$

Összesített komplexitás: Minden lekérdezés és módosítás $O(\sqrt{N} \log \sqrt{N})$, így a teljes algoritmus elég gyors $N, Q \leq 10^5$ esetén.



Lekérdezés: [2,13] $\geq 6 \rightarrow 7$

2. feladat: Tömb lekérdezések - Kód

```
1  int main() {
2      ios::sync_with_stdio(false); cin.tie(nullptr);
3      int n; cin >> n;
4      vector<int> A(n); for (int &a : A) cin >> a;
5
6      int B = sqrt(n) + 1;
7      vector<vector<int>> blocks(B);
8      for (int i = 0; i < n; i++) blocks[i / B].push_back(A[i]);
9      for (auto &b : blocks) sort(b.begin(), b.end());
10
11     auto query = [&](int l, int r, int c) {
12         int res = 0;
13         for (int i = l; i <= r; i++) {
14             if (i % B == 0 && i + B - 1 <= r) {
15                 auto &b = blocks[i / B];
16                 res += b.end() - lower_bound(b.begin(), b.end(), c);
17                 i += B - 1;
18             } else {
19                 if (A[i] >= c) res++;
20             }
21         }
22         return res;
23     };
```

2. feladat: Tömb lekérdezések - Kód

```
24
25      auto update = [&](int i, int val) {
26          auto &b = blocks[i / B];
27          auto it = lower_bound(b.begin(), b.end(), A[i]);
28          *it = val; A[i] = val;
29          sort(b.begin(), b.end());
30      };
31
32      int Q; cin >> Q;
33      while (Q--) {
34          int t, i, j, c;
35          cin >> t >> i;
36          if (t == 0) {
37              cin >> j >> c;
38              cout << query(i - 1, j - 1, c) << "\n";
39          } else {
40              cin >> c;
41              update(i - 1, c);
42          }
43      }
```

3. feladat: Fekete-fehér négyzetrács (CPH 27.1.)

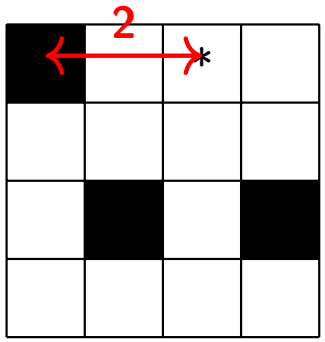
Adott egy $R \times C$ méretű rács ($1 \leq R, C \leq 400$), kezdetben minden cella fehér, kivéve a bal felső (1,1) cellát, ami fekete. Ezután N lépés történik ($1 \leq N < R \cdot C$), minden lépésben adott egy (i, j) fehér cella, melyre:

- ki kell számolni a legkisebb Manhattan-távolságát bármely fekete cellától,
- ezután a cella feketére változik.

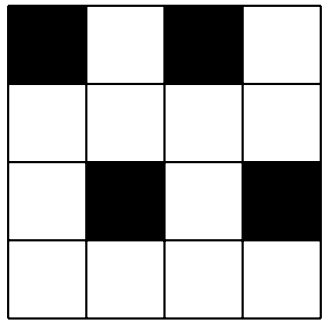
Példa

Bemenet	Kimenet
4 4	3
3	2
3 2	2
3 4	
1 3	

Az utolsó lépésben kiválasztott cella (*):



A kiválasztott cella feketére festése:



3. feladat: Fekete-fehér négyzetrács - Megoldási ötlet

Két algoritmus, amik külön-külön túl lassúak:

- **1. algoritmus:** BFS a teljes rácson a fekete cellákból indulva, így meglesz minden fehér cella távolsága a legközelebbi fekete cellától. Idő: $O(R \cdot C)$, utána bármely cella minimum távolsága $O(1)$-ben lekérdezhető.
- **2. algoritmus:** Fenntartunk egy listát a feketére festett cellákról. Minden lépésben kiszámoljuk a távolságot a lista minden elemétől, majd a cellát hozzáadjuk a listához. Idő: $O(k)$, ahol k a lista hossza.

A két algoritmust kombináljuk. Hogyan?

3. feladat: Fekete-fehér négyzetrács - Megoldás

- **1. algoritmus:** BFS a teljes rácson a fekete cellákból indulva.
- **2. algoritmus:** Fenntartunk egy listát a feketére festett cellákról.

A két algoritmust kombináljuk. Először az N lekérdezést $\sim \sqrt{N}$ **batch**-re osztjuk, minden batch $\sim \sqrt{N}$ lekérdezésből áll.

Batch feldolgozás:

- A batch elején futtatjuk az **1. algoritmus**t. $O(R \cdot C)$ idő, $\sqrt{N}$-szer fut le.
- A batch lekérdezéseit feldolgozzuk a **2. algoritmus**sal. (A batch végén töröljük a listát.) $O(\sqrt{N})$ idő, N-szer fut le.
- Egy cella minimális távolsága mindig az 1. vagy a 2. algoritmus által számolt érték (a kettő közül a kisebb).

4. feladat: Feketére festett négyzetek (AtCoder ABC335F)

Van N négyzet egy sorban, 1-től N-ig számozva és adott egy N hosszúságú sorozat, $A = (A_1, A_2, \ldots, A_N)$. Kezdetben az 1-es négyzet fekete, a többi fehér, és egy bábu áll az 1-es négyzeten.
Ismételjük a következő lépést akárhányszor (esetleg 0-szor):

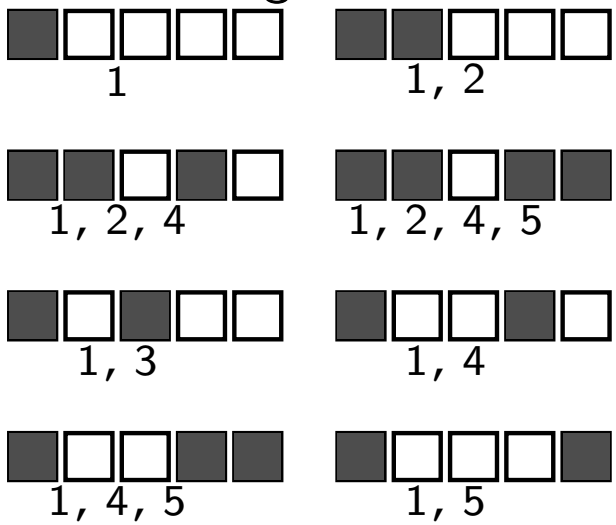
- Ha a bábu az i-edik négyzeten áll, válasszunk egy pozitív x számot és lépjünk a $i + A_i \cdot x$-edik négyzetre, feltéve, hogy $i + A_i \cdot x \leq N$.
- A cél négyzetet fessük feketévé.

Hány különböző színezés érhető el a folyamat végén, modulo 998244353?

Példa

Bemenet	Kimenet
5 1 2 3 1 1	8

A 8 lehetséges színezés:



4. feladat: Feketére festett négyzetek - Megoldási ötlet

A fekete mezők halmaza megfelel annak, ahogyan a bábu halad, ezért elegendő csak a lépéseket számolni.
A probléma DP-vel oldható meg:

```
dp = {1,0,...,0};
for(int i = 1; i <= N; i++)
    for(int j = i + A[i]; j <= N; j += A[i])
        dp[j] += dp[i];
cout << sum(dp) << "\n";
```

Ha az A_i értékek nagyok, a belső ciklus nem fut sokszor, így a módszer ésszerű. Ha azonban $A_i = (1, 1, \ldots, 1)$, a komplexitás $O(N^2)$, ezért lassú.

Kicsi A_i értékekre jó lenne gyorsítani a DP-t, hogyan?

4. feladat: Feketére festett négyzetek - DP optimalizálás

A DP átmenetekhez:

- $i < j$ esetén `dp[i]` → `dp[j]` akkor és csak akkor történik, ha $(j - i)$ osztható A_i-vel, azaz $i \bmod A_i = j \bmod A_i$.
- Ezt kihasználva definiálhatunk egy segédtömböt:

$$\texttt{dp2[d][r]} = \sum \text{dp[i] azoknál i-knél, ahol } i \bmod d = r$$

  Ez lehetővé teszi, hogy minden *kis* A_i esetén gyorsan frissítsük az érintett dp-értékeket, anélkül hogy végig kellene iterálni minden j-t.

- A *kis* és *nagy* A_i értékeket a b konstans választja el:

  $A_i \leq b$: a `dp2` tömb segítségével $O(b)$ idő alatt frissíthetünk, összességében $O(N \cdot b)$.

  $A_i > b$: a hagyományos iterációval $O(N/b)$ időt igényel, összességében $O(N \cdot N/b)$.

- Az optimális $b \approx \sqrt{N}$, így a teljes algoritmus $O(N\sqrt{N})$-ban fut.

4. feladat: Feketére festett négyzetek - Kód

```cpp
#include <bits/stdc++.h>
using namespace std;

const int MOD = 998244353, B = 500;
void add(int &a, int b) { a += b; if (a >= MOD) a -= MOD; }

int dp[200005], dp2[505][505];

int main() {
    ios::sync_with_stdio(false); cin.tie(nullptr);
    int n; cin >> n;

    dp[0] = 1;
    for (int i = 0; i < n; i++) {
        int a; cin >> a;
        for (int d = 1; d <= B; d++) add(dp[i], dp2[d][i % d]);
        if (a <= B) add(dp2[a][i % a], dp[i]);
        else for (int j = i + a; j < n; j += a) add(dp[j], dp[i]);
    }

    int res = 0;
    for (int i = 0; i < n; i++) add(res, dp[i]);
    cout << res << "\n";
}
```

Gyökös felbontás (SQRT decomposition) - Gyakori típusok

1. **Blokkolás:** Adatok csoportosítása $\sqrt{N}$ darab, $\sqrt{N}$ méretű blokkba.

2. **Kombinálás:** Két lassú algoritmus hatékony összekombinálása hatékonnyá, tipikusan kicsi és nagy értékek szerint szétválasztva.

3. **Batchelés:** Query + update feladatoknál $\sqrt{Q}$ lekérdezésenként preprocesszálunk, majd a változásokat ehhez képest kezeljük.

+1. **Mo-algoritmus:** Offline query-k sorrendje átrendezhető, és így megfelelően csoportosíthatjuk őket, hogy gyors legyen feldolgozás.

Tipp: a blokkméret, illetve a kicsi-nagy határ legyen egy konstans változó, amit könnyű módosítani.

+1. feladat: SPOJ DQUERY

Adott egy N hosszú sorozat: $A = (A_1, A_2, \ldots, A_N)$ és Q darab lekérdezés ($1 \leq N, Q \leq 10^5, \ 1 \leq A_i \leq 10^6$).
Minden lekérdezésben adott egy (i, j) pár, és az a kérdés, hogy hány **különböző szám** szerepel az $A_i, A_{i+1}, \ldots, A_j$ részintervallumban?

Példa

Bemenet	Kimenet
5 1 1 2 1 3 3 1 5 2 4 3 5	3 2 3

+1. feladat: SPOJ DQUERY - Megoldási ötlet

Ötlet: Mindig tartsunk nyilván egy $[L, R]$ *aktív intervallumot*, amelyhez az aktuális választ ismerjük.

Hogyan lépünk egyik lekérdezéstől a másikig?

A bal (L) és jobb (R) határokat egyenként mozgatjuk:

- ha L-t jobbra visszük, elemeket törlünk az intervallumból,
- ha L-t balra visszük, elemeket adunk hozzá,
- ha R-t jobbra visszük, elemeket adunk hozzá,
- ha R-t balra visszük, elemeket törlünk.

Az intervallum bővítése/szűkítésekor frissítjük a statisztikát:

- `count[x]` = hányszor szerepel x az aktív intervallumban,
- `different`: különböző elemek száma.

Probléma: Ha a lekérdezéseket sorrendben dolgozzuk fel, a határokat nagyon sokat kell mozgatni $\Rightarrow$ túl lassú.
Ötlet: Rendezzük át a lekérdezéseket, hogy összességében ne túl sokat mozogjanak a határok.

+1. feladat: SPOJ DQUERY - Mo algoritmus

Ötlet: Lekérdezések átrendezése.

Rendezési szabály (nagyon hasonlít az 1. példához - Pontok a síkon):

Osszuk fel a tömböt $\sqrt{N}$ hosszú blokkokra.

Két lekérdezést $[L_1, R_1]$, $[L_2, R_2]$ így hasonlítunk össze:

- ha $\lfloor L_1/\sqrt{N} \rfloor < \lfloor L_2/\sqrt{N} \rfloor$, akkor $[L_1, R_1]$ előbb jön,
- ha ugyanabban a blokkban vannak: $R_1 < R_2$ dönt.

Miért jó ez?

- A bal határ (L) általában legfeljebb $\sqrt{N}$-et mozog, blokkhatároknál lehet legfeljebb $2\sqrt{N}$, összességében $O(Q\sqrt{N})$.
- A jobb határ (R) egy blokkon belül csak egy irányba mozog, összesen legfeljebb N-et, és blokkok között is legfeljebb N-et, összesen $O(N\sqrt{N})$.
- Összes komplexitás: $O((N+Q)\sqrt{N})$.

Alkalmazás: minden léptetéskor az intervallum bővítése/szűkítése történik, aminél $O(1)$ művelettel tudjuk frissíteni az aktuális választ.

+1. feladat: SPOJ DQUERY - Mo algoritmus - Kód

```
1  #include <bits/stdc++.h>
2  using namespace std;
3
4  struct Query { int l, r, idx; };
5
6  int main() {
7      int N; cin >> N;
8      vector<int> A(N);
9      for (int &x : A) cin >> x;
10
11     int Q; cin >> Q;
12     vector<Query> qs(Q);
13     for (int i = 0; i < Q; i++) {
14         cin >> qs[i].l >> qs[i].r;
15         qs[i].l--, qs[i].r--, qs[i].idx = i;
16     }
17
18     int B = sqrt(N);
19     sort(qs.begin(), qs.end(), [&](auto a, auto b) {
20         int block_a = a.l / B, block_b = b.l / B;
21         return block_a == block_b ? a.r < b.r : block_a < block_b;
22     });
```

+1. feladat: SPOJ DQUERY - Mo algoritmus - Kód

```
23      vector<int> cnt(1'000'001), ans(Q);
24      int distinct = 0, L = 0, R = -1;
25
26      auto add = [&](int i) {
27          if (++cnt[A[i]] == 1) distinct++;
28      };
29      auto remove = [&](int i) {
30          if (--cnt[A[i]] == 0) distinct--;
31      };
32
33      for (auto q : qs) {
34          while (L > q.l) add(--L);
35          while (R < q.r) add(++R);
36          while (L < q.l) remove(L++);
37          while (R > q.r) remove(R--);
38          ans[q.idx] = distinct;
39      }
40
41      for (int x : ans) cout << x << "\n";
42  }
```

Feladatok

Feladatok csoportos munkára:

A. **POI 2017 – Shipping Containers**
`https://szkopul.edu.pl/problemset/problem/oNnWY6ZuzzhvG-jCmijiXkIk/site/?key=statement`

B. **XOR and Favourite Number**
`https://codeforces.com/contest/617/problem/E`

C. **Holes**
`https://codeforces.com/contest/13/problem/E`

D. **JOI 2018 – Bitaro's Birthday**
`https://oj.uz/problem/view/JOI18_bitaro`

Lekötő feladatok:

L1. **Induced Subgraph Queries**
`https://codeforces.com/contest/2129/problem/E`

L2. **Xenia and Tree**
`https://codeforces.com/contest/342/problem/E`

L3. **IOI 2011 - Dancing Elephants**
`https://oj.uz/problem/view/IOI11_elephants`