## Range minimum query (RMQ)

Enunțul problemei: Se dă un șir de valori numerice și mai multe interogări de aflare minim pe secvențe oarecare din șir.

Notăm cu `v` tabloul în care se memorează elementele șirului și cu `n` numărul de elemente. Presupunem că acestea sunt stocate începând cu poziția `1`. Notăm cu `m` numărul de întrebări și considerăm că fiecare dintre acestea este precizată prin indicii de început și de final notați `st` și `dr` (`st <= dr`).

Observăm că întrebările se pun fără ca între timp șirul să se mai modifice. Încadrăm deci problema la capitolul “operații statice pe șiruri”.

**Soluția brută** presupune ca la fiecare interogare să determinăm maximul din secvența dată, printr-o parcurgere de la indicele `st` la indicele `dr`.

```
fin>>n>>m;
for (i=1;i<=n;i++)
    fin>>v[i];
for (i=1;i<=m;i++) {
    fin>>st>>dr;
    minim = INF;
    for (j=st;j<=dr;j++)
        if (v[j] < minim)
            minim = v[j];
    fout<<minim<<"\n";
}
```

Timpul de calcul este de ordinul `n*m`, iar pe date mari programul va rula lent.

**O soluție mai bună** se poate obține folosind arbori de intervale. Structura de date necesită însă cunoștințe de programare mai avansată și oferă timp de calcul de ordin logaritmic (în `n`) pentru fiecare interogare, precum și timp de calcul de ordin $n \log_2 n$ pentru pregătirea structurii.

Prezentăm în continuare o soluție care necesită o preprocesare cu $n \log_2 n$ timp și memorie dar pentru care rezultatul la fiecare întrebare se poate obține în timp constant.

Întrucât elementele vectorului nu se mai modifică în timpul interogărilor, putem reorganiza de la început datele într-un mod convenabil tipului nostru de interogări (minim pe secvențe).

Probabil ne gândim mai întâi la sume parțiale. Dacă am fi avut nevoie de sume pe secvențe, puteam precalcula un șir de sume parțiale (`s[i]` = suma elementelor din `v` de la poziția `1` la poziția `i`) și la fiecare interogare am fi răspuns în timp constant (`s[dr]-s[st-1]`). Calculul lui

`s` ar necesita timp constant, realizându-se la citire printr-o instrucțiune suplimentară de atribuire (`s[i] = s[i-1] + v[i]`).

Fiind vorba despre minime, realizarea de "minime parțiale" nu ne-ar ajuta (minimul din secvența de la `1` la `i` s-ar obține ușor, `minim[i] = min (v[i], minim[i-1])`) dar nu ne-ar fi util pentru a obține minimul din secvența de la `st` la `dr` (nu putem decide minimul din secvență folosind `minim[dr]` și `minim[st]`, ca în cazul sumei)).

Vom folosi altă abordare, care ne oferă **timp de executare constant pentru fiecare intrerogare**.
Pentru fiecare poziție `i` din `v`, vom calcula mai multe minime. *Pentru fiecarte secvență de lungime **putere de 2** care începe la poziția `i`, vom calcula minimul dintre elementele ei*.

De exemplu, pentru datele de intrare: `n = 11` și `v = (4, 2, 7, 8, 3, 5, 9, 1, 6, 2, 9)`, și pentru poziția `3` (acolo unde este valoarea `7`) vom valcula `4` valori:

- minimul unei secvențe de lungime `1` ce începe la poziția `3` (secvența: `7`), cu valoarea `7`;
- minimul unei secvențe de lungime `2` ce începe la poziția `3` (secvența: `7`, `8`), cu valoarea `7`;
- minimul unei secvențe de lungime `4` ce începe la poziția `3` (secvența: `7`, `8`, `3`, `5`), cu valoarea `3`;
- minimul unei secvențe de lungime `8` ce începe la poziția `3` (secvența: `7`, `8`, `3`, `5`, `9`, `1`, `6`, `2`), cu valoarea `1`;

Deci, pentru fiecare poziție din `v` vom avea maxim $\log_2 n$ minime de calculat (atâtea puteri de `2` sunt mai mici sau egale decât `n`).
**Astfel, vom obține o matrice `r[p][i]` = minimul din `v` pentru elementele dintr-o secvență de lungime $2^p$ care începe la poziția `i`.**
Valorile de pe linia `0` a acestei matrice vor fi minimele unor secvențe de lungime `1` care încep în șirul dat la indicele din dreptul valorii, valorile de pe linia `1` reprezintă minimele unor secvențe de lungime `2`, cele de pe linia `2` reprezintă minimele pentru secvențe de lungime `4` etc.

Pentru vectorul `v` exemplificat mai sus, matricea `r` ar fi:

| v: | 4 | 2 | 7 | 8 | 3 | 5 | 9 | 1 | 6 | 2 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| r: | | | | | | | | | | | |
| $2^0$ | 4 | 2 | 7 | 8 | 3 | 5 | 9 | 1 | 6 | 2 | 9 |
| $2^1$ | 2 | 2 | 7 | 3 | 3 | 5 | 1 | 1 | 6 | 2 | 9 |
| $2^2$ | 2 | 2 | 3 | 3 | 1 | 1 | 1 | 1 | 2 | 2 | 9 |
| $2^3$ | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 2 | 2 | 9 |

Deci indicii de coloană reprezintă indici din `v`, adică locuri unde încep secvențele de lungime putere de `2` pentru care se calculează minime. Indicii de linie reprezintă exponenți ai puterii de `2` ce dă lungimea secvențelor pentru care se stochează informații pe acea linie.
O altă observație: elementele de pe linia `0` a lui `r` sunt chiar elementele șirului dat.

Promitem așadar că dacă am reușit să memorăm minime pentru un număr de ordin $n \log_2 n$ secvențe dintre cele $n^2$ care sunt în total, am putea apoi să aflăm minimul din oricare altă secvență în timp constant (fără repetiții suplimenrare la fiecare interogare).

Să vedem mai departe cum realizăm acestea. Mai întâi vom arăta cum construim valorile din `r` iar în etapa a doua vom arăta cum ne folosim de `r` pentru a realiza interogările.

**Modul de calcul pentru `r`.**
Așadar linia `0` o inițializăm cu valorile șirului de la intrare. În practică putem face asta direct de la citire, astfel nici nu mai e nevoie de declararea vectorului `v`.

```
int r[17][100010];
fin>>n;
for (i=1;i<=n;i++)
    fin>>r[0][i];
```

Observăm modul în care declarăm matricea în care structurăm datele pentru **`rmq`** (numărul de linii este de ordin $\log_2$ din numărul de coloane).

Pentru a construi elementele de pe celelalte linii este esențială următoarea observație: orice secvență pentru care calculăm minimul pe linia `i` (deci care are lungimea $2^i$) se obține concatenând două secvențe pentru care avem informații pe linia `i-1` (deci de lungime $2^{i-1}$).

Pentru o mai bună înțelegere este utilă analizarea datelor din următorul tabel:

| indici: | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| v: | 4 | 2 | 7 | 8 | 3 | 5 | 9 | 1 | 6 | 2 | 9 |
| r: | | | | | | | | | | | |
| $2^0$ | 4 | 2 | 7 | 8 | 3 | 5 | 9 | 1 | 6 | 2 | 9 |
| $2^1$ | 2 | 2 | 7 | 3 | 3 | 5 | 1 | 1 | 2 | 2 | 9 |
| $2^2$ | 2 | 2 | 3 | 3 | 1 | 1 | 1 | 1 | 2 | 2 | 9 |
| $2^3$ | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 2 | 2 | 9 |

Concret, elementul `r[3][2]` trebuie să memoreze minimul din toată secvența hașurată sus, adică de lungime $2^3$ și care începe la poziția `2`. Această secvență este formată din cea de lungime $2^2$ care începe la poziția `2` și din cea de lungime $2^2$ care începe la poziția `6`. Minimele din aceste două jumătăți ale ei se află în elementele `r[2][2]` respectiv `r[2][6]`.
Așadar, valoarea `r[3][2] = min (r[2][2], r[2][6])`.

Privind la general, un element de pe linia `p` se calculează ca minimul a două elemente de pe linia `p-1`.
`r[p][i] = min (r[p-1][i], r[p-1][i+2^(p-1)])`.
Având deja linia `0` cunoscută nu ne rămâne decât să calculăm valorile linie cu linie, în ordinea crescătoare a indicilor de linii. Valoarea $2^{p-1}$ se poate calcula direct folosind operatorul de deplasare la stânga pe biți ( `1 << (p-1)` ).
O ultimă observație înainte de a prezenta secvența de cod: la determinarea minimului dintre cele două elemente de pe linia anterioară trebuie să ne asigurăm că există cel din dreapta, fiind suficient să testăm ca `i+(1 << (p-1))` să fie mai mic sau egal cu `n`.

```
for (p=1; (1<<p) <= n; p++)
    for (i=1;i<=n;i++) {
        r[p][i] = r[p-1][i];
        if (i +(1<<(p-1))<=n && r[p][i] > r[p-1][i+ (1<<(p-1))])
            r[p][i] = r[p-1][i + (1<<(p-1))];
}
```

Ca detaliu de implementare, am putea stoca într-o variabilă valoarea coloanei elementului din dreapta de pe linia anterioară, astfel obținem un cod mai clar și mai rapid.

```
for (p=1; (1<<p) <= n; p++)
    for (i=1;i<=n;i++) {
        r[p][i] = r[p-1][i];
        int j = i +(1<<(p-1));
        if (j<=n && r[p][i] > r[p-1][j])
            r[p][i] = r[p-1][j];
}
```

Este acum mai vizibil că am construit o structură de date care ocupă memorie de ordin $n*\log_2 n$ și timpul de executare pentru acest lucru este de același ordin.

Trecem acum la **etapa a doua**, să vedem cum o folosim structura construită.
Noi primim la fiecare interogare capetele unui interval, [`st`, `dr`] care, evident, nu este neapărat de lungime putere de `2`.

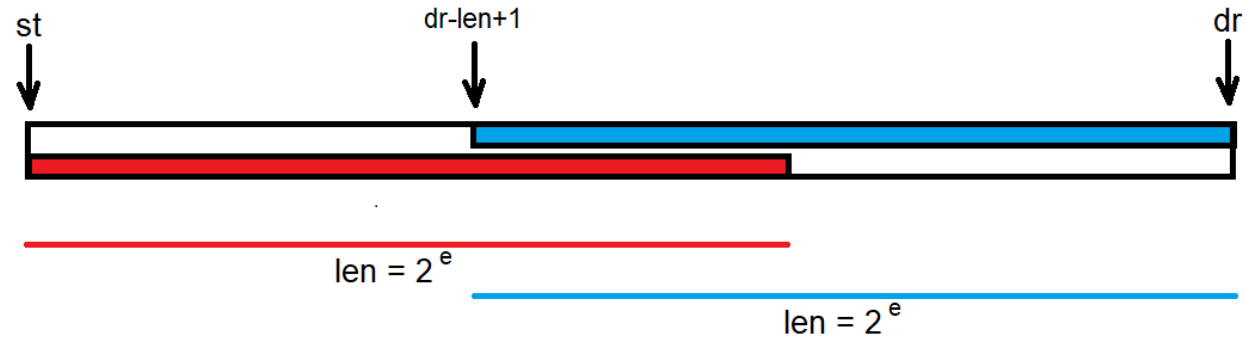
O primă ideea ar fi să descompunem numărul `L = dr-st+1` (lungimea intervalului) în baza `2`, adică scriindu-l ca sumă de puteri de `2`, putem să aflăm minimul său folosind minimele din intervalele corespunzătoare, acestea fiind de lungime putere de `2` (deci le regăsim în `r`).

Descompunerea unui număr în baza `2` are însă timp de calcul de ordin logaritmic, ori noi ne-am propus să răspundem la fiecare interogare în timp constant.

Pentru acest lucru, să considerăm cea mai mare putere de `2` mai mică sau egală decât `L` (notăm cu `len` această valoare). Această putere de `2` este totodată mai mare decât jumătatea lungimii intervalului (altfel nu ea ar fi cea mai mare putere de `2` mai mică sau egală cu `L`, ci dublul ei).
Acum să consideră, intervalul de lungime `len` care începe la poziția `st` și pe cel de lungime `len` care se termină la poziția `dr`.

St = 10
Dr = 30  lungimea secventei este 21. Cea mai mare putere de 2 mai mica sau egala cu 21 este 16. Afirm ca Aceasta putere de 2 este mai mare decat jumatatea lungimii secventei

R




Aceste două intervale acoperă în întregime intervalul [`st`, `dr`], iar faptul că se suprapun la mijloc nu ne încurcă. Pentru fiecare dintre cele două intervale noi avem câte un element în `r` unde este stocat minimul secvenței corespunzătoare.

Să notăm cu e exponentul puterii lui `2` egală cu len: $2^e = len$

Elementul din `r` care stochează minimul din secvența care începe la poziția `st` și are lungimea $2^e$ este `r[e][st]` iar elementul care stochează minimul din secvența de lungime `len` care se termină la poziția `dr` este `r[e][ dr - (1<<e) +1 ]`. Formula provine din faptul că noi trebuie să indicăm începutul secvenței (care are lungimea `len` și se termină la poziția `dr`) - nu uităm că `len` este o putere de `2`.
Astfel, la o interogare de forma: care este minimul din secvența care începe la poziția `st` și se termină la poziția `dr`, răspunsul este: `min (r[e][st], r[e][ dr - (1<<e) +1 ])`, adică minimul dintre două valori precalculate deja anterior.

Ultimul lucru care ne mai încurcă este: cum aflăm valoarea `e` (exponentul celei mai mari puteri de `2` mai mică sau egală cu o valoare dată).

O primă soluție ar fi să ne folosim de funcții de calcul al logaritmului care se găsesc în biblioteca de funcții matematice. Acestea implică operații cu numere reale și sunt în general lente, lucru vizibil mai ales dacă sunt apelate de multe ori.
Soluția pe care noi o alegem este să precalculăm valorile `e` pentru fiecare număr de la `1` la `n`. Astfel, vom construi un vector `E`, în care `E[i]` = exponentul celei mai mari puteri de `2` mai mică sau egală cu `i`. `E[i] = 1 + E[i/2]`. Intuitiv este destul de evident (la dublarea valorii, exponentul puterii lui `2` crește cu `1`). Pornind cu `E[1] = 0`, putem calcula dinainte vectorul `E` ca o etapă de precalculare:

```
E[1] = 0;
for (i=2;i<=n;i++)
    E[i] = 1 + E[i/2];
```

Programul complet este:

```
#include <fstream>
#define INF 1000000000
using namespace std;
ifstream fin ("rmq.in");
ofstream fout("rmq.out");
int r[17][100010];
int E[100010];
int n, m, i, j, st, dr, minim, p, e, len;
int main () {
    fin>>n>>m;
    for (i=1;i<=n;i++)
        fin>>r[0][i];

    for (p=1; (1<<p) <= n; p++)
        for (i=1;i<=n;i++) {
            r[p][i] = r[p-1][i];
            j = i + (1<<(p-1));
            if (j <= n && r[p][i] > r[p-1][j])
                r[p][i] = r[p-1][j];
        }
    E[1] = 0;
    for (i=2;i<=n;i++)
        E[i] = 1 + E[i/2];

    for (i=1;i<=m;i++) {
        fin>>st>>dr;
        e = E[dr-st+1];
        len = (1<<e);
        fout<<min(r[e][st], r[e][dr-len+1])<<"\n";
    }
}
```

## RMQ 2d

Vom aplica tehnica folosind o problemă de pe infoarena.ro, devenită clasică: https://infoarena.ro/problema/plantatie.

Aici avem dată o matrice și interogările sunt submatrice **pătratice** ale ei pentru care trebuie să determinăm elementul maxim (evident că structurile pot fi folosite la fel și pentru maxim și pentru minim).

Vom nota cu `A` matricea dată (pentru simplitate în problema plantație ea este pătratică, cu `n` linii și `n` coloane). Pentru o interogare vom folosi notația (`i`, `j`, `lat`) cu semnificația: să se determine valoarea maximă din submatricea pătratică având colțul stânga-sus (`i`,`j`) și latura `lat`.

În etapa de precalculare, pentru fiecare poziție (`i`,`j`) din matricea dată `A`, vom calcula maximele din toate submatricele pătratice care au colțul stânga sus în (`i`,`j`) și care au latura putere de `2`. Vor fi $\log_2 n$ astfel de submatrice.

Așadar vom avea o matrice tridimensională de dimensiune $n^2 \log_2 n$.

Așa cum la cazul unidimensional pentru fiecare poziție de `i` determinam minimele din secvențe care încep la poziția `i` și au lungime putere de `2`, la cazul bidimensional pentru fiecare poziție (`i`,`j`) din matrice determinăm maximele din toate submatricele pătratice care au colțul stânga-sus (`i`,`j`) și au laturi cu lungimea putere de `2`.

Pentru o mai bună înțelegere, să exemplificăm.
Fie matricea `A`:

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | | | r[0][2][3] | 3 |
| 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | | | r[1][2][3] | 5 |
| 3 | 3 | 4 | 3 | 3 | 3 | 3 | 3 | 3 | | | r[2][2][3] | 7 |
| 3 | 5 | 3 | 4 | 6 | 3 | 3 | 3 | 3 | | | r[3][2][3] | 8 |
| 3 | 3 | 3 | 3 | 3 | 3 | 5 | 3 | 3 | | | | |
| 7 | 3 | 7 | 3 | 3 | 3 | 3 | 3 | 3 | | | | |
| 7 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | | | | |
| 3 | 3 | 3 | 3 | 3 | 3 | 3 | 8 | 3 | | | | |
| 2 | 3 | 3 | 3 | 8 | 3 | 3 | 3 | 3 | | | | |
| 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | | | | |

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 3 | 3 | 1 | 3 | 3 | 3 | 2 | 3 | | | | |

În partea dreaptă avem toate valorile calculate pentru submatrice cu colțul stânga-sus în poziția 2,3.

Pentru a realiza precalcularea ne folosim de aceeași observație ca și la vectori: o putere de 2 se poate compune din exact două puteri de 2 cu exponent cu 1 mai mic. Aici un pătrat cu latură putere $2^p$ îl vom descompune în cele 4 "sferturi" care sunt fiecare pătrate de latură $2^{p-1}$.

Astfel r[p][i][j] va fi maximul dintre următoarele 4 valori:
r[p-1][i][j]
r[p-1][i+$2^{p-1}$][j]
r[p-1][i][j+$2^{p-1}$]
r[p-1][i+$2^{p-1}$][j+$2^{p-1}$]

Extinzând de la vectori, ne imaginăm că avem $\log_2 n$ matrice (fiecare de nXm) una peste alta, și un element al unei matrice (aflată pe poziția p în șirul matricelor) se calculează în funcție de 4 elemente ale matricei anterioare (cea de pe poziția p-1).

Matricea 0 este chiar matricea dată (valorile corespund unor submatrice pătratice de latură $1 = 2^0$).

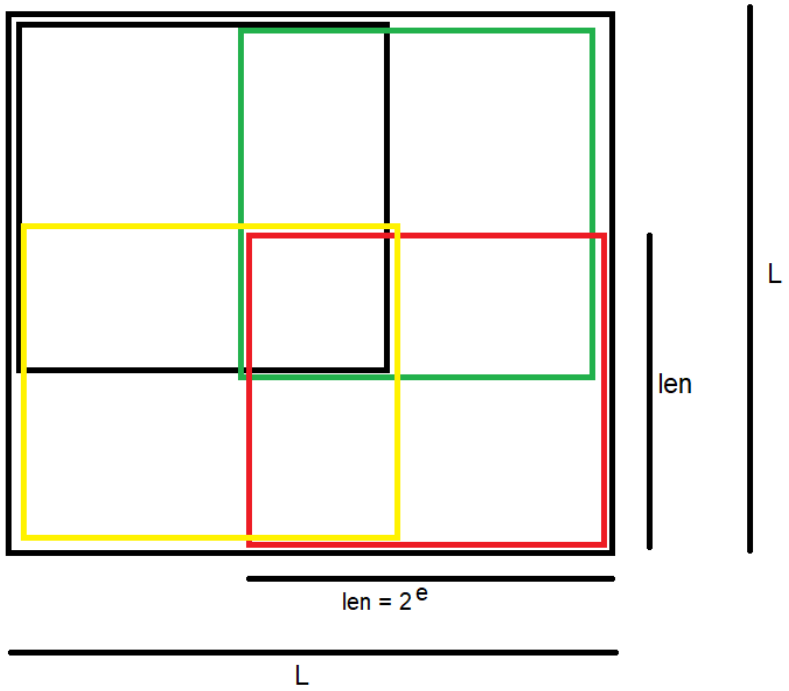
Pentru a le calcula pe celelalte una din cealaltă avem următorul cod în care implementăm cele spuse mai sus.

```
for (int p=1,lat=2; lat<=n; p++,lat*=2){
    for (int i1=1;i1<=n-lat+1;i1++){
        for (int j1=1;j1<=n-lat+1;j1++){
            i2=i1+(lat>>1);
            j2=j1+(lat>>1);
            r[p][i1][j1] = maxim(
                            r[p-1][i1][j1],
                            r[p-1][i2][j1],
                            r[p-1][i1][j2],
                            r[p-1][i2][j2]));
        }
    }
}
```

Ca detaliu de implementare, am ales ca în forul cu log pași să calculăm simultan puterea de 2 și exponentul.

Odată realizată precalcularea vom folosi un raționament similar pentru a răspunde la interogare. Reluăm contextul: avem precalculată matricea tridimensională r și avem o interogare de forma (i, j, L) care specifică o submatrice pătratică de latură L și care are colțul stânga sus (i,j) pentru care dorim să aflăm minimul.

Vom considera `len` = cea mai mare putere de `2` mai mică sau egală cu `L` (notăm cu `e` exponentul acestei puteri). Vom determina minimul cerut folosind `4` valori din `r` care corespund la `4` submatrice pătratice de latură $2^{e-1}$ și care se suprapun peste colțuri ale submatricei pătratice de latură `L` pe care facem interogarea. Evident că aceste `4` submatrice se suprapun puțin în mijloc dar acest lucru nu ne încurcă, este important că ele asigură minimul din toată submatricea de dimensiune `L` și că nu iau în calcul elemente din afara ei.




Iată o sursă completă dintre cele trimise pe infoarena la problema plantație.

```
#include <fstream>
#define DIM 502

using namespace std;

int n,m,i1,i2,j1,j2,lat,L,len,k;
int r[10][DIM][DIM];
int E[DIM];

inline int maxim(int a, int b){
    return (a>b ? a : b);
}

int main(){
    ifstream fin("plantatie.in");
    ofstream fout("plantatie.out");
    fin>>n>>m;
    for (int i=1;i<=n;i++){
        for (int j=1;j<=n;j++){
```

```
            fin>>r[0][i][j];
        }
    }
    for (int p=1,lat=2; lat<=n; p++,lat*=2){
        for (int i1=1;i1<=n-lat+1;i1++){
            for (int j1=1;j1<=n-lat+1;j1++){
                i2=i1+(lat>>1);
                j2=j1+(lat>>1);
                r[p][i1][j1] = max(
                                    max(r[p-1][i1][j1],r[p-1][i2][j1]),
                                    max(r[p-1][i1][j2],r[p-1][i2][j2])
                                );
            }
        }
    }
    E[1] = 0;
    for (int i=2;i<=n;i++)
        E[i] = 1 + E[i/2];

    for (;m;m--){
        fin>>i1>>j1>>L;
        k = E[L];
        len = (1<<k); /// cea mai mare putere de 2 <= latura
                      /// patratului de la interogare
        i2 = i1+L-len;
        j2 = j1+L-len;
        fout<<max(
                  max(r[k][i1][j1],r[k][i1][j2]),
                  max(r[k][i2][j1],r[k][i2][j2])
                )<<"\n";
    }
    return 0;
}
```

În concluzie avem timp de ordin $n^2 \log_2 n$ in etapa de preprocesare (+memorie de același ordin) dar obținem timp constant de răspuns la interogări.

**RMQ2d cu interogări pe dreptunghiuri**

Așa cum este scris în titlu, diferența față de problema anterioară este că avem interogare pe un dreptunghi și nu pe un pătrat.

Vom face analiza folosind problema euclid de pe infoarena (https://www.infoarena.ro/problema/euclid).

Enunțul pe scurt: se dă o matrice cu $m$ linii și $n$ coloane (cu elemente naturale) și trebuie să determinăm un dreptunghi cu $h$ linii și $w$ coloane în care cmmdc al tuturor elementelor sale să fie cât mai mare posibil.

Este acum ocazia să subliniem că trucul de RMQ funcționează nu numai la maxime și minime pe scevențe ci și la cmmdc (precalculăm cmmdc pentru toate secvențele de lungime putere de 2). Revenind la problema pe care o analizăm în acest paragraf, vă invităm să studiați în primul rând descrierea oficială a soluției sale, foarte frumoasă, care se găsește pe infoarena (https://www.infoarena.ro/summer-challenge-2007/solutii/runda-2).

Venim și noi aici, în continuare, cu câteva explicații.
Deci, o primă abordare este să construim o matrice cu 4 dimensiuni, în care un element D[i][j][ei][ej] reprezintă informația dorită (cmmdc la noi) pentru un dreptunghi ce are colțul stânga-sus la linia i și coloana j și care are $2^{ei}$ linii și $2^{ej}$ coloane. Astfel, dacă avem o interogare pe o submatrice cu h linii și w coloane, ne interesează cea mai mare putere de 2 mai mică sau egală cu h (să notăm eh exponentul ei) și cea mai mare putere de 2 mai mică sau egală w (notăm ew exponentul ei). Pentru a afla cmmdc pentru un dreptunghi de dimensiuni (h,w) ne interesează valorile precalculate din cele 4 dreptunghiuri de dimensiuni ($2^{eh}$, $2^{ew}$) suprapuse peste colțurile dreptunghiului de la interogare.
La această soluție avem nevoie de memorie de ordin $m*n*\log_2 m*\log_2 n$. Timpul de calcul este de același ordin, în etapa de preprocesare.

A doua abordare reduce cu un log factorul de timp și de memorie.
În prima etapă vom calcula informații pentru secvențe de lungime putere de 2 pe linii și apoi determinăm cmmdc din toate secvențele de lungime w, pe linii. Deci nu mai gandim cmmdc pe submatrice ci doar pe vectori (tratam fiecare linie ca pe un vector). Așadar putem obține o structură L în care L[i][j] = cmmdc al elementelor din secvența de lungime w care începe pe linia i la coloana j. Deci fiecare element din L ține acum informația dintr-o secvență de lungime w din matricea dată.
Rămâne acum să aplicăm asupra matricei L același raționament, dar pe coloane și ne interesează cmmdc pe scevențe de lungime putere de 2 pe coloane din L și în final obținem cmmdc pe secvențe de lungime h pe coloane din L și evident soluția este maximul dintre cmmdc din aceste secvențe.

O soluție comentată detaliat, pe a doua idee, găsiți mai jos.

```
#include <cstdio>
#define DIM 401
using namespace std;
int D[9][DIM][DIM];
int L[DIM][DIM];
int E[DIM];
int m, n, h, w, i, j, t, e, sol, T;

int cmmdc(int a, int b) {
    int r;
    while (b) {
```

```
        r = a%b;
        a = b;
        b = r;
    }
    return a;
}

int maxim(int a, int b) {
    return a > b ? a : b;
}

int main () {
    FILE *fin  = fopen("euclid.in","r");
    FILE *fout = fopen("euclid.out","w");

    /// precalculare: E[i] = exponentul celei mai mari
    /// puteri de 2 mai mica sau egala cu i
    E[1] = 0;
    for (i=2;i<=400;i++)
        E[i] = 1 + E[i/2];
    fscanf(fin,"%d",&T);
    for (t = 1;t<=T; t++) {
        sol = 0;
        fscanf(fin,"%d%d%d%d",&m,&n,&h,&w);
        for (i=1;i<=m;i++)
            for (j=1;j<=n;j++) {
                fscanf(fin,"%d",&D[0][i][j]);
            }
/// calculam in D cmmdc pentru secvente
/// de lungime putere de 2, pe linii
        for (e=1;(1<<e) <= n; e++)
            for (i=1;i<=m;i++)
                for (j=1;j<=n;j++) {
                    D[e][i][j] = D[e-1][i][j];
                    if (j+(1<<(e-1)) <= n)
                        D[e][i][j] = cmmdc(D[e-1][i][j],
                                           D[e-1][i][j+(1<<(e-1))]);
                }
/// interogam D si obtinem in L cmmdc pe secvente
/// de lungime w, pe linii
        for (i=1;i<=m;i++)
            for (j=1;j+w-1<=n;j++) {
                e = E[w];
                L[i][j] = cmmdc(D[e][i][j], D[e][i][j+w - (1<<e)]);
            }

/// aplicam asupra lui L (care practic a contractat
/// rezultatul de pe w coloane in una singura) acelasi
/// rationament dar pe coloane
/// obtinem in D cmmdc pe secvente din L
/// de lungime putere de 2, pe coloane
```

```
        for (i=1;i<=m;i++)
            for (j=1;j+w-1<=n;j++)
                D[0][i][j] = L[i][j];
/// interogand noul D pe secvente de lungime h
/// si luand maximul dintre rezultate,
/// obtinem rezultatul
        for (e=1; (1<<e) <= m; e++)
            for (j=1;j+w-1<=n;j++) {
                for (i=1;i<=m;i++) {
                    D[e][i][j] = D[e-1][i][j];
                    if (i+(1<<(e-1)) <= m)
                        D[e][i][j] = cmmdc(D[e][i][j],
                                           D[e-1][i+(1<<(e-1))][j]);
                }
            }

        for (j=1;j+w-1<=n;j++) {
            for (i=1;i+h-1 <= m; i++) {
                e = E[h];
                sol = maxim (sol, cmmdc(D[e][i][j],
                                        D[e][i+h- (1<<e)][j]));
            }
        }
        fprintf(fout,"Case #%d: %d\n",t,sol);
    }

    return 0;
}
```

*Craiova, 14 ianuarie 2022*