

# Descriere Sniper

**Autor: Prof. Adrian Panaete**  
**Descrierea soluției: Cătălin Frâncu**

## 1 Numărul de permutări posibile

În primul rînd, cîte permutări distincte putem obține după eliminarea tuturor elementelor?

La prima tăietură avem patru variante de eliminare, fiecare pornind din alt colț. În fiecare variantă, putem reface matricea în două moduri distincte. De exemplu, dacă eliminăm diagonală de la NV spre SE, atunci jumătatea dreaptă poate migra cu o linie în jos sau cu o coloană spre stînga.

La a doua tăietură, și în general la tăieturile cu număr de ordine par, avem de-a face cu o matrice dreptunghiulară,  $m \times n$  unde  $m = n \pm 1$ . Există din nou patru variante de eliminare, pornind din fiecare colț, dar apoi direcția de alipire este impusă, ca să refacem forma pătrată. Figura 1 prezintă un exemplu.

Rezultă că există  $4 \times 2 \times 4 = 32$  de moduri de a reduce latura matricei cu 1. Face excepție cazul final  $n = 2$ , unde există doar 8 moduri de eliminare a celor 4 elemente: odată eliminată o diagonală, rămîne o matrice de  $2 \times 1$  sau de  $1 \times 2$  cu două variante de eliminare.

Este important că toate aceste moduri sînt distincte. Figura 1 arată că pentru a doua tăietură, deși pornim din același colț (19), conținutul diagonalelor este diferit pentru că am făcut alipirea anterioară în moduri diferite.

## 2 Aflarea celei de-a $P$ -a permutări

Așadar, numărul de permutări posibile este o putere a lui 2, mai exact 3 biți pentru  $n = 2$  și cîte 5 biți pentru fiecare unitate a lui  $n$  peste 2. În total

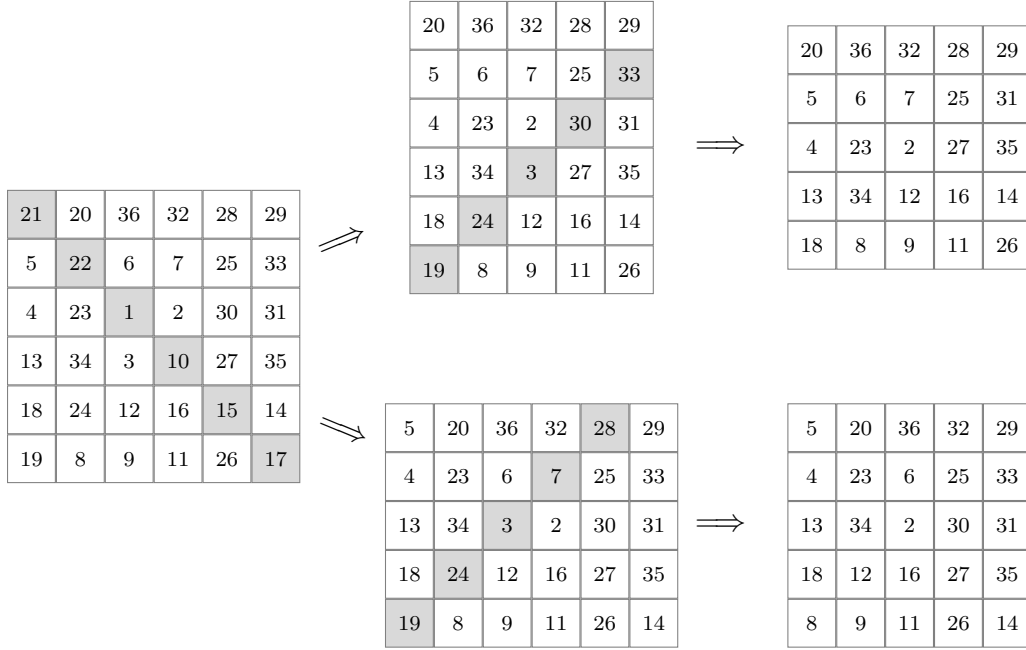


Figura 1: Un exemplu de eliminare pentru  $n = 6$ .

există  $T = 2^{5(n-2)+3}$  permutări distincte iar, pentru a o genera pe a  $P$ -a, putem să procesăm grupe de câte 5 biți din  $P$  (3 la ultima iterație) și să facem eliminarea cu numărul de ordine dat dintre cele 32 posibile. Primele (posibil multe) iterații vor avea numărul de ordine 0, avînd în vedere că  $P$  are 64 de biți, iar  $T$  poate avea mii de biți.

Cum aflăm a  $k$ -a variantă din 32? Ordonarea nu este trivială. După cum se vede în figură, în funcție de direcția primei alipiri, la a doua tăietură putem obține șiruri care coincid, dar numai pînă la jumătate. Elementele care fac departajarea (30 și respectiv 7) se află imediat după centrul matricei. Trebuie să le calculăm valoarea înainte de a face ordonarea variantelor. Vom reveni la acest aspect.

Astfel, putem sorta cele 32 de variante după trei criterii, în ordinea importanței:

1. valoarea din colțul de început al primei tăieturi;
2. valoarea din colțul de început al celei de-a doua tăieturi;

3. valoarea elementului de departajare, dacă există.

### 3 Implementare izotropă

Pentru a evita duplicarea (sau tetraplicarea?) codului, este util să căutăm o reprezentare **izotropă** a datelor: codul să fie același indiferent de colțul din care începem tăierea.

O soluție este să codificăm circular direcțiile, de exemplu 0=est, 1=sud, 2=vest, 3=nord. Atunci orientarea relativă devine independentă de valoarea exactă. „Dreapta” este întotdeauna „direcția următoare modulo 4”. Iată un exemplu. Când tăiem o diagonală, mereu vom merge „la dreapta” și „în jos” relativ la colțul de pornire. În funcție de direcția de alipire aleasă, mereu vom conecta vecinii „de la stînga” și „de la dreapta” elementului eliminat sau pe cei „de deasupra” și „de dedesubt”.

Practic, căpătăm luxul de a scrie codul doar pentru colțul de nord-vest. Tot ce trebuie este să nu folosim direcțiile 0, 1, 2 și 3, ci  $d$ ,  $d + 1$ ,  $d + 2$  și  $d + 3$  raportate la o direcție  $d$  primită ca parametru.

### 4 Soluție în $\mathcal{O}(n^3)$

Pentru 73 de puncte, putem reprezenta matricea în mod natural. La tăierea unei diagonale mutăm  $\mathcal{O}(n^2)$  elemente pentru a reface matricea. Efortul total este  $\mathcal{O}(n^3)$ .

Următoarele două soluții cer o implementare atentă și buna gestiune a unor pointeri.

### 5 Soluție în $\mathcal{O}(n^2)$ (implementare cu rețea de pointeri)

Pentru a reface matricea în  $\mathcal{O}(n)$  după o tăietură, o putem reprezenta ca pe o rețea de pointeri. Construim o matrice convențională, care pe lângă valori mai stochează și patru pointeri către celulele vecine pe cele patru direcții.

La eliminarea unui element, trebuie să conectăm corect vecinii săi pentru ca rețeaua rezultată să fie consecventă.

În această reprezentare, nu putem accesa direct elementele în funcție de coordonate. De aceea, găsirea elementului de departajare, care este aproape de centrul matricei, costă  $\mathcal{O}(n)$ . Acest efort nu este dăunător asimptotic, căci tăierea unei diagonale costă oricum  $\mathcal{O}(n)$ , dar este nevoie să facem 8 astfel de căutări pentru fiecare  $n$ , deci constanta este semnificativă.

Este necesar să menținem evidența celor patru colțuri. Inițial ele se află pe pozițiile 0,  $n - 1$ ,  $n^2 - 1$  și  $n^2 - n$ , apoi încep să migreze pe măsură ce tăiem diagonale.

## 6 Soluție în $\mathcal{O}(n^2)$ (implementare cu pătrate concentrice)

Putem reprezenta matricea și ca  $\lceil n/2 \rceil$  pătrate concentrice. Un prim avantaj al acestei reprezentări este că necesită doar doi pointeri per celulă, la elementele anterior și următor de pe inel. Aceasta reduce memoria cu circa 40%.

În această reprezentare, este de ajutor să stocăm colțurile fiecărui pătrat. Eliminarea unui element se simplifică: trebuie doar să unim elementele anterior și următor, apoi să recalculăm colțurile.

Dar marele avantaj este că putem identifica elementele de departajare în  $\mathcal{O}(1)$ : știm din ce colț începe diagonala considerată și știm că elementele căutate se află în partea opusă, pe inelul central. Aceasta reduce timpul de rulare de 3-4 ori.