AVANSAT · OLIMPIADĂ · LECȚIA 11

Operații pe biți și mulțimi codificate pe biți (bitmask), DP pe submulțimi

Durată: 80 min Nivel: olimpiadă / informatică intensiv Înainte: lecțiile x-08, xi-16, co-05

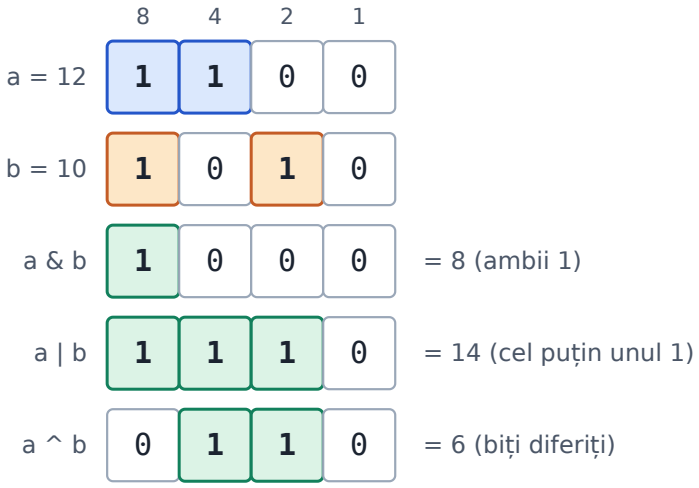
CE VEI ȘTI DUPĂ LECȚIE

- Folosești operatorii pe biți (&, |, ^, ~, <<, >>) și verifici, setezi, ștergi, comuți un bit.
- Reprezinți o mulțime mică (n ≤ 20) ca un număr întreg (mască de biți).
- Enumeri toate submulțimile și submulțimile unei măști.
- Rezolvi o problemă de programare dinamică pe submulțimi (comis-voiajorul, n ≤ 15).

1. Numerele ca șiruri de biți

Un `int` este reținut ca 32 de biți (0 sau 1). Operatorii pe biți lucrează **bit cu bit**, foarte rapid (o instrucțiune de procesor), iar un număr mic poate reprezenta o **mulțime** întreagă. Ambele idei apar des la olimpiadă.

Operator	Nume	Exemplu cu 4 biți
`a & b`	ȘI pe biți	`1100 & 1010 = 1000`
`a | b`	SAU pe biți	`1100 | 1010 = 1110`
`a ^ b`	XOR (sau exclusiv)	`1100 ^ 1010 = 0110`
`~a`	negare (inversează toți biții)	`~0101 = …1010`
`a << k`	deplasare la stânga cu k biți	`0011 << 2 = 1100` (înmulțire cu 2^k)
`a >> k`	deplasare la dreapta cu k biți	`1100 >> 2 = 0011` (împărțire întreagă la 2^k)



a & b: 1 doar unde ambii au 1; a | b: 1 unde cel puțin unul are 1; a ^ b: 1 unde diferă

Operații

Biții se numerotează de la **dreapta**, începând cu 0: bitul 0 are valoarea 1, bitul 1 are valoarea 2, bitul 2 are 4 etc.

Cele patru operații pe un bit

Ce vrei	Cum	Exemplu (x = 5 = 0101, i = 1)
verifici bitul i	`(x >> i) & 1`	`(5 >> 1) & 1 = 0`
setezi bitul i la 1	`x | (1 << i)`	`5 | 2 = 7` (0111)
ștergi bitul i (îl pui 0)	`x & ~(1 << i)`	`7 & ~2 = 5`
comuți bitul i	`x ^ (1 << i)`	`5 ^ 2 = 7`, apoi `7 ^ 2 = 5`

DE CE AȘA?

De ce `1 << i`? Numărul `1 << i` are **un singur** bit de 1, la poziția i. Cu `&` îl „citim”, cu `|` îl „aprindem”, cu `& ~` îl „stingem”, cu `^` îl „comutăm”. Este o „mască” care selectează un bit. **Atenție la tip:** pentru i ≥ 31 folosește `1LL << i`, altfel depășești `int`.

Trucuri utile

Ce	Cum
număr par / impar	`x & 1` (1 = impar)
2^k	`1 << k`
x este putere a lui 2?	`x > 0 && (x & (x - 1)) == 0`
cel mai din dreapta bit de 1	`x & -x` (lowbit, lecția co-07)
numărul de biți de 1	`__builtin_popcount(x)` (sau `__builtin_popcountll` pentru `long long`)
a și b diferă?	`a ^ b != 0`

Ce	Cum
interschimbă a și b fără variabilă	`a ^= b; b ^= a; a ^= b;` (curiozitate)

```cpp
#include <iostream>
using namespace std;

int main()
{
    int n, x = 0;
    cin >> n;
    for (int i = 0; i < n; i++)
    {
        int a;
        cin >> a;
        x ^= a;                          // a ^ a = 0, a ^ 0 = a
    }
    cout << x;                           // numărul care apare o singură dată
    return 0;
}
```

DE ȘTIUT

Problema clasică: într-un șir, fiecare număr apare de două ori, în afară de unul. XOR-ul tuturor elementelor este exact acel număr, pentru că `a ^ a = 0`, `a ^ 0 = a`, iar XOR este comutativ. Memorie O(1), timp O(n).

2. Mulțimi pe biți (mască)

Dacă universul are cel mult ~20 de elemente numerotate `0 … n − 1`, o submulțime se reprezintă printr-un număr în care **bitul i este 1 dacă elementul i este în mulțime**. Exemplu: mulțimea `{0, 2, 3}` este $1101_2 = 13$.

Operația pe mulțimi	Cu mască
mulțimea vidă	`0`
universul `{0, …, n − 1}`	`(1 << n) - 1`
`i` ∈ M?	`(M >> i) & 1`
adaugă `i` în M	`M | (1 << i)`
scoate `i` din M	`M & ~(1 << i)`
reuniune A ∪ B	`A | B`
intersecție A ∩ B	`A & B`
complementul lui A	`((1 << n) - 1) ^ A`
numărul de elemente al lui A	`__builtin_popcount(A)`
A ⊆ B?	`(A & B) == A`

Enumerarea tuturor submulțimilor

O mulțime cu n elemente are 2^n submulțimi, codificate chiar de numerele `0 … 2ⁿ − 1`.

```cpp
#include <iostream>
using namespace std;

int main()
{
    int n, S, v[20];
    cin >> n >> S;
    for (int i = 0; i < n; i++) cin >> v[i];
    int numar = 0;
    for (int masca = 0; masca < (1 << n); masca++)       // fiecare submulțime
    {
        int suma = 0;
        for (int i = 0; i < n; i++)
            if (masca >> i & 1) suma += v[i];            // elementul i este în submulțime
        if (suma == S) numar++;
    }
    cout << numar;                                       // câte submulțimi au suma S
    return 0;
}
```

Pentru `v = {3, 4, 2, 5}` și `S = 7`: submulțimile cu suma 7 sunt `{3, 4}` (masca `0011`) și `{2, 5}` (masca `1100`): **2**. Costul este $O(2^n \cdot n)$: rezonabil pentru $n \le 20$ ($\approx 2 \cdot 10^7$).

Submulțimile unei submulțimi

Pentru a enumera **submăștile** lui `M` (toate submulțimile mulțimii M):

```cpp
#include <iostream>
using namespace std;

int main()
{
    int M;
    cin >> M;
    for (int s = M; ; s = (s - 1) & M)     // următoarea submască, în ordine descrescătoare
    {
        cout << s << ' ';
        if (s == 0) break;                 // 0 (mulțimea vidă) este ultima
    }
    return 0;
}
```

Pentru `M = 13 = 1101₂`, submăștile sunt (zecimal) 13, 12, 9, 8, 5, 4, 1, 0: câte `2^(popcount(M)) = 8`.

> **DE CE AȘA?**
>
> **De ce `(s − 1) & M`?** Scăderea lui 1 din `s` transformă ultimul bit de 1 în 0 și toți biții de la dreapta lui în 1. Prin `& M` păstrăm doar biții care sunt în M. Rezultatul este următoarea submască mai mică. **Cost:** suma, peste toate măștile M, a numărului de submăști este 3^n (fiecare element este: în afara lui M, în M dar nu în s, sau în s).

3. DP pe submulțimi: comis-voiajorul

Problemă. Un comis-voiajor pornește din orașul 0, trebuie să viziteze **toate** cele n orașe (n ≤ 15), exact o dată fiecare, și să revină în 0. Se dau distanțele `d[i][j]`. Care este lungimea minimă a turului?

Încercarea tuturor ordinilor costă n! (pentru n = 15 sunt 10^{12}). Observație: pentru viitor contează doar **mulțimea orașelor deja vizitate** și **orașul în care ne aflăm**, nu și ordinea în care am ajuns.

Stare: `dp[masca][i]` = lungimea minimă a unui drum care pornește din 0, vizitează **exact** orașele din `masca` și se termină în orașul `i` (care e în `masca`).

Pornire: `dp[1][0] = 0` (masca `{0}`, suntem în 0).

Tranziție: din `(masca, i)` mergem într-un oraș nevizitat `j`: `dp[masca | (1 << j)][j] = min(…, dp[masca][i] + d[i][j])`.

Răspuns: `min` peste `i ≠ 0` din `dp[toate][i] + d[i][0]`.

```cpp
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;

int main()
{
    int n;
    cin >> n;
    vector<vector<int>> d(n, vector<int>(n));
    for (auto &rand : d)
        for (auto &x : rand) cin >> x;

    const int INF = 1000000000;
    vector<vector<int>> dp(1 << n, vector<int>(n, INF));
    dp[1][0] = 0;                                          // am vizitat doar orașul 0
    for (int masca = 1; masca < (1 << n); masca++)
        for (int i = 0; i < n; i++)
        {
            if (!(masca >> i & 1) || dp[masca][i] == INF) continue;
            for (int j = 0; j < n; j++)
                if (!(masca >> j & 1))                     // j nevizitat
                {
                    int nou = masca | (1 << j);
                    dp[nou][j] = min(dp[nou][j], dp[masca][i] + d[i][j]);
                }
        }
    int rezultat = INF;
    for (int i = 1; i < n; i++)
        rezultat = min(rezultat, dp[(1 << n) - 1][i] + d[i][0]);   // revenirea în 0
    cout << rezultat;
    return 0;
}
```

Pentru cele 4 orașe din test, tururile sunt: 0→1→3→2→0 = 10 + 25 + 30 + 15 = **80**; 0→1→2→3→0 = 10 + 35 + 30 + 20 = 95; 0→2→1→3→0 = 15 + 35 + 25 + 20 = 95; (și celelalte trei sunt inversele lor, cu aceleași lungimi). Minimul este **80**.

DE CE AȘA?

De ce masca este suficientă? DP-ul are „proprietatea fără memorie”: costul continuării depinde doar de orașele rămase și de orașul curent, nu de cum am ajuns aici. Astfel, 15! ≈ 10^{12} ordini se reduc la $2^{15} \cdot 15$ stări cu 15 tranziții fiecare: $2^{15} \cdot 15 \cdot 15 \approx 7{,}4 \cdot 10^6$ operații. **De ce parcurgem măștile crescător?** Tranziția merge de la o mască mai mică la una mai mare (`masca | (1 << j) > masca`), deci `dp[masca]` este final când îl folosim. **Memorie:** $2^n \cdot n$ valori; pentru n = 20 înseamnă 20 milioane de `int` : 80 MB, deci limita practică este n ≈ 20.

4. Când se folosesc măștile?

Semnal în enunț	Tehnica
n ≤ 20 și „toate submulțimile”	enumerare cu mască, $O(2^n \cdot n)$
„vizitează toate”, „fiecare o dată”, n ≤ 20	DP pe submulțimi
„asignează n sarcini la n persoane”, n ≤ 20	DP pe submulțimi (masca = sarcinile deja asignate)
memorie mică pentru mulți `bool`	`bitset` sau mască
XOR, paritate, „apare o singură dată”	`^`
„numărul de biți de 1”, „puterea lui 2”	`__builtin_popcount` , `x & (x - 1)`

5. Greșeli frecvente

Greșeala	Ce se întâmplă	Corect
`1 << i` cu `i ≥ 31`	depășire (comportament nedefinit)	`1LL << i`
`masca >> i & 1` fără paranteze la combinări	prioritate surprinzătoare: `&` are prioritate mai mică decât `==`	pune paranteze: `((masca >> i) & 1) == 1`
`x << 1` când x e deja mare	depășire	`long long` sau verifică limita
`~x` pe `unsigned` vs `int`	rezultat neașteptat	rămâi pe același tip
`dp[1 << n][n]` pentru n = 25	depășește memoria	n ≤ 20; altfel altă tehnică
uitarea stării inițiale `dp[1][0] = 0`	totul rămâne INF	inițializează starea de pornire
`INF + d` în `int`	depășire	`INF = 10⁹` și verifică `dp != INF`
enumerarea submăștilor cu `s - 1` fără `& M`	iese din submulțime	`(s - 1) & M`

6. La olimpiadă

Măștile apar în două forme: (1) ca **unealtă de viteză** (xor, `__builtin_popcount`, `bitset` pentru memorie sau pentru a „paraleliza" 64 de operații în una) și (2) ca **DP cu stare de submulțime**, de obicei când constrângerea este n ≤ 15–20. Dacă vezi n ≤ 20, gândește-te la 2^n.

7. Exersează pe hârtie

1. Calculează în binar (4 biți): `0110 & 0101`, `0110 | 0101`, `0110 ^ 0101`.

2. Care este valoarea lui `1 << 5`? Dar `100 >> 2`?

3. Cum verifici dacă bitul 3 din `x` este 1? Cum îl pui pe 0?

4. Reprezintă mulțimea `{1, 3, 4}` (elemente de la 0) ca număr.

5. Câte biți de 1 are 29 (11101_2)? Este 64 putere a lui 2? Dar 96?

6. Pentru M = 6 = 110_2, enumeră toate submăștile (în ordinea `s = (s - 1) & M`).

7. Câte submulțimi are o mulțime cu 12 elemente? Cât costă enumerarea cu verificare O(n) pe fiecare?

8. Șirul `5 3 5 8 3` : care număr apare o singură dată? (XOR.)

9. Pentru TSP cu n = 4: câte stări `(masca, i)` există și câte sunt posibile?

10. De ce nu merge DP-ul pe submulțimi la n = 30?

8. Mini-grile

1. `x & (x - 1)` șterge: a) cel mai din stânga bit de 1 b) cel mai din dreapta bit de 1 c) toți biții d) niciun bit

2. Câte submulțimi are o mulțime cu n elemente? a) n b) n^2 c) 2^n d) n!

3. `1 << 3` este: a) 3 b) 6 c) 8 d) 9

4. Operatorul care „comută" un bit este: a) `&` b) `|` c) `^` d) `~`

9. Mai departe (pentru cine vrea mai mult)

- **Gray code:** `i ^ (i >> 1)` produce o ordine în care numerele consecutive diferă printr-un singur bit.
- **Meet in the middle:** pentru n ≈ 40, împarți în două jumătăți de câte 20 și combini.
- **SOS DP (sum over subsets):** sume peste submulțimi în $O(2^n \cdot n)$.
- **`bitset`:** operații pe 64 de biți odată; util pentru accesibilitate în grafuri și pentru DP de tip rucsac.
- **XOR-ul în jocuri (Nim):** o poziție este pierzătoare dacă XOR-ul grămezilor este 0.

Rezumat

- Operatori: `&`, `|`, `^`, `~`, `<<`, `>>`. Bitul i: `(x >> i) & 1`; setare `| (1 << i)`; ștergere `& ~(1 << i)`; comutare `^ (1 << i)`.

- Mulțime mică = număr; `__builtin_popcount` numără elementele; enumerarea tuturor submulțimilor este `for (m = 0; m < (1 << n); m++)`.
- Submăștile lui M: `s = (s - 1) & M`.
- DP pe submulțimi: `dp[masca][ultimul]`; n ≤ 15–20; TSP în $O(2^n \cdot n^2)$.
- XOR găsește numărul care apare o singură dată.

Probleme de lucru (concurs)

Probleme de pe infoarena, CSES, Kilonova sau pbinfo, în ordinea dificultății. Trimite soluțiile pe platformă și bifează ce ai terminat.

- [ ] CSES **Apples** — ușoară
  împarte merele în două grupe cât mai echilibrat: enumerare de submulțimi
  https://cses.fi/problemset/task/1623
- [ ] CSES **Gray Code** — ușoară
  xor între i și i >> 1
  https://cses.fi/problemset/task/2205
- [ ] CSES **Hamiltonian Flights** — dificilă
  DP pe submulțimi: numărarea drumurilor hamiltoniene
  https://cses.fi/problemset/task/1690
- [ ] CSES **Elevator Rides** — dificilă
  DP pe submulțimi: câte curse de lift
  https://cses.fi/problemset/task/1653

Temă (concurs)

Mai grele: încearcă-le după ce le-ai terminat pe cele de mai sus.

- [ ] CSES **Counting Bits** — medie
  biții de 1 de la 1 la n: număr pe fiecare poziție
  https://cses.fi/problemset/task/1146
- [ ] USACO Guide **SOS DP** — dificilă
  sume peste submulțimi (avansat)
  https://usaco.guide/adv/sos-dp

Citește și exersează în plus

- CP-Algorithms — Bit manipulation — trucuri cu biți (engleză) https://cp-algorithms.com/algebra/bit-manipulation.html
- USACO Guide — Bitmasks — măști și DP pe măști, probleme graduate (engleză) https://usaco.guide/gold/bitmasks
- cppreference — std::bitset — biți în loc de bool, pentru memorie mică (engleză) https://en.cppreference.com/w/cpp/utility/bitset

Răspunsuri

Exersează pe hârtie

1. `0110 & 0101 = 0100`; `0110 | 0101 = 0111`; `0110 ^ 0101 = 0011`.

2. `1 << 5 = 32`; `100 >> 2 = 25`.

3. Verificare: `(x >> 3) & 1`. Ștergere: `x & ~(1 << 3)`.

4. `{1, 3, 4}`: biții 1, 3, 4: 2 + 8 + 16 = **26** (11010_2).

5. 29 = 16 + 8 + 4 + 1: **4** biți de 1. 64 este putere a lui 2 (`64 & 63 = 0`); 96 = 64 + 32 nu este (`96 & 95 = 64`).

6. M = 110_2: `s = 6`, apoi `(6 − 1) & 6 = 5 & 6 = 4`, apoi `(4 − 1) & 6 = 3 & 6 = 2`, apoi `(2 − 1) & 6 = 1 & 6 = 0`. Submăștile: **6, 4, 2, 0**.

7. 2^{12} = 4096 submulțimi; cu verificare O(12) pe fiecare: 4096 · 12 ≈ 49 000 operații.

8. `5 ^ 3 ^ 5 ^ 8 ^ 3 = 8`.

9. În total $2^4 \cdot 4 = 64$ de perechi `(masca, i)`. Se folosesc doar cele cu orașul 0 în mască și `i` în mască: starea de pornire este `dp[1][0]`, iar pentru `i ≠ 0` masca are cel puțin biții 0 și i, deci $2^2 = 4$ măști pentru fiecare dintre cele 3 orașe, adică 12 stări, plus starea de pornire.

10. $2^{30} \approx 10^9$ măști, ori 30 de orașe: $3 \cdot 10^{10}$ valori în memorie: imposibil.

Mini-grile

1. b) cel mai din dreapta bit de 1. **2.** c) 2^n. **3.** c) 8. **4.** c) `^`.