Coduri Gray

Negruşeri Cosmin

Acest articol prezintă noţiunea de cod Gray şi unele aplicaţii ale lui în probleme de la concursurile de programare.

Un cod Gray de ordin n este un şir care conţine toate numerele de la 0 la 2^n-1 astfel încât oricare două numere consecutive din şir diferă în reprezentarea binară prin exact un bit. Pentru $n \geq 2$ există mai multe coduri Gray distincte. Un cod mai des întâlnit, numit în engleză binary reflected Gray code, mai jos ne când vorbim despre codul Gray ne vom referi de fapt la acest cod. Modul de construcţie al acestui cod este unul intuitiv: la fiecare pas adăugăm numărul adăugat anterior căruia îi modificăm cel mai puţin semnificativ bit, astfel ca numărul obţinut să nu mai fi fost adăugat înainte la şir. De exemplu dacă ordinul este doi şi avem la început numărul 0 în şir vom adăuga 1, apoi 3 iar apoi 4. În acest articol vom nota acest cod cu **Gn**. O altă metodă de construcţie care obţine acelaşi cod Gray este de a determina mai întâi pe **Gn-1**, apoi dacă notăm cu **Ğn-1** şirul **Gn-1** inversat, atunci obţinem pe **Gn** dacă punem câte un bit de 0 în faţa fiecărui număr din **Gn-1** iar acestea sunt urmate de numerele din **Ğn-1** cărora le adăugăm câte un bit de 1 ca bit semnificativ, pe scurt **Gn** = 0**Gn-1,** 1**Ğn-1** (1)**.** Observăm că acest cod este unul circular, adică ultimul număr şi primul număr din cod diferă prin exact un bit. Observăm de asemenea că fiecare cod de ordin n îl conţine pe cel de n-1 ca prefix, deci am putea nota prin **G**∞ şirul numerelor naturale în care orice două numere consecutive diferă în reprezentarea binară prin exact un bit şi pentru care şirul primelor 2^n elemente coincide cu **Gn** oricare ar fi acest n un număr natural**.** Fie g(x) al x-lea număr din **G**∞ (prin g(x) notăm codul Gray al numărului x) şi g-1(y) la ce poziţie apare numărul y în şirul **G**∞**.** Ne punem problema calculării eficiente a celor două funcţii. Se poate demonstra prin inducţie că dacă avem un număr x cu reprezentarea binară (…a2 a1 a0)2 şi codul lui Gray cu reprezentarea binară (… b2 b1 b0)2. Atunci avem bi = ai ^ ai+1 (2). De exemplu dacă avem numărul 6 cu reprezentarea binară 110, atunci b0 = a0 ^ a1 = 0 ^ 1 = 1, b1 = a1 ^ a2 = 1 ^ 1 = 0, b2 = a2 ^ a3 = 1^ 0 = 1, deci g(110) = 101. Din această relaţie tragem concluzia că g(i) = x ^ (x / 2) (unde prin / am notat împărţire întreagă). Din (2) obţinem că bi ^ bi+1 ^ bi+2 ^ … = (ai ^ ai+1) ^ (ai+1 ^ ai+2) ^ (ai+2 ^ ai+3) ^… = ai. Această sumă este finită deoarece vom avea un bi egal cu 0 şi un ai egal cu 0, dacă i este de ajuns de mare. Astfel g-1(y) = y ^ y/2 ^ y/4 ^ …
Din cele de mai sus obţinem următoarele metode:

```
int binToGray(int x) {
     return x ^ (x >> 1);
}

int  grayToBin(int y) {
     int ret = 0;
    while (y > 0) {
        ret ^= y;
        y >>= 1;
     }
     return ret;
}
```

Să vedem mai departe câteva probleme.

Problema 1: Turnurile din Hanoi
Avem trei tije şi n discuri de dimensiuni diferite plasate în ordinea descrescătoare a dimensiunilor pe prima tijă. Se doreşte mutarea tuturor discurilor pe cea de a doua tijă, iar mutările acceptate sunt mutarea unui disc de pe o tijă pe alta cu condiţia ca pe tija destinaţie discul din vârf dacă există un asemenea disc să fie mai mare ca discul ce îl mutăm acum.

Rezolvare:
Aceasta este o problemă clasică în predarea recursivităţii, dar ea are o soluţie care se foloseşte de codul Gray. Urmărim pas cu pas ce bit se schimbă de la numerele consecutive ale lui Gn, acest bit corespunde discului pe care îl vom muta (cel mai puţin semnificativ bit corespunde celui mai mic disc, iar cel mai semnificativ bit corespunde celui mai mare disc). Problema este că ştim ce disc să mutăm, dar nu ştim pe ce tijă. Dar există o regulă simplă care ne poate ajuta: un disc nu trebuie plasat pe alt disc ce are aceeaşi paritate. Această strategie rezolvă şi ea în număr minim de paşi problema, acest număr fiind $2^n - 1$.

Problema 2:
Un bec este conectat la n întrerupătoare, fiecarui astfel de întrerupător îi poate fi schimbată starea prin apăsare. Problema este că nu ştim starea lor iniţială. Se cere să se determine o strategie care ar face număr minim de apăsări în cazul cel mai rău, pentru a aprinde becul.

Rezolvare:
Pentru a fi siguri că aprindem becul trebuie în cel mai rău caz să trecem prin toate configuraţiile apăsat/ne-apăsat ale întrerupătoarelor. Pentru a schimba starea curentă trebuie să apăsăm cel puţin un buton, deci în cazul cel mai defavorabil va trebui să facem cel puţin 2^n-1 apăsări. Codul Gray ne dă o posibilitate de a trece prin toate configuraţiile trecând de la una la alta prin o apăsare de buton, rezolvându-ne problema.

Problema 3: Matrix (249 acm.sgu.ru)
Trebuie să aranjaţi numerele de la 0 la $2^{(n+m)} - 1$ într-o matrice cu 2^n rânduri şi 2^m coloane. De asemenea numerele care sunt situate în celule adiacente pe verticală sau orizontală trebuie să difere prin exact un bit în reprezentarea lor binară. Matricea este ciclică, adică pentru fiecare rând celula cea mai din stânga se consideră învecintată cu cea mai din dreapta, de asemenea pentru fiecare coloană celula cea mai de sus se consideră învecinată cu celula cea mai de jos. La intrare se dau numerele n şi m ($0 < n, m; n + m \le 20$). Trebuie să afişaţi o asemenea matrice. De exemplu dacă $n = 1$ şi $m = 1$ o matrice ar fi
```
(0 2)
(1 3)
```

Rezolvare:
O primă metodă ar fi de determinare a codului $Gn+m$ iar apoi de a-l scrie şerpuit în matrice, adică pe de index par de la stânga la dreapta, iar în liniile de index impar de la dreapta la stânga.
De exemplu dacă $n = 2$ şi $m = 2$ avem:
G4 = 0000, 0001, 0011, 0010, 0110, 0111, 0101, 0100, 1100, 1101, 1111, 1110, 1010, 1011, 1001, 1000

0000	0001	0011	0010
0100	0101	0111	0110
1100	1101	1111	1110
1000	1001	1011	1010

Altă modalitate de rezolvare care ne dă de fapt aceeaşi matrice este următoarea:

Întregul din celula (i, j) a matricei va fi în reprezentarea binară rezultatul concatenării reprezentării binare a întregului de index i din codul Gn cu reprezentarea binară a întregului de index j din codul Gm.
De exemplu dacă avem N = 3 şi M = 2

	00	01	11	10
000	00000	00001	00011	00010
001	00100	00101	00111	00110
011	01100	01101	01111	01110
010	01000	01001	01011	01010
110	11000	11001	11011	11010
111	11100	11101	11111	11110
101	10100	10101	10111	10111
100	10000	10001	10011	10010

Este evident că orice două numere din matrice vor fi diferite, orice două numere adiacente în matrice sunt diferite în reprezentarea binară prin exact un bit. În concurs limita de timp era foarte strânsă şi trebuia folosită funcţia prezentată anterior. Deci pe linia i coloana j a matricei scriem numărul (binToGray(i) << M) | binToGray(j).

Problema 4: Divizori (algoritmus)
Vom considera un număr natural N. În şirul A vom aşeza toţi divizorii lui N (2≤N≤2 000 000 000). Se cere să se permute elementele şirului A astfel încât pentru oricare două elemente consecutive A[i] şi A[i+1] să avem fie A[i]=A[i+1]*p, fie A[i+1]=A[i]*p, unde p este un număr prim oarecare. Valoarea p poate diferi de la o pereche de elemente la alta. De exemplu pentru N = 12 o posibilă soluţie ar fi 1 2 3 4 12 6 3.

Rezolvare:
Numărul N are maxim $2*N^{1/2}$ divizori. Dacă descompunem numărul N în factori primi atunci îl vom putea scrie în forma $P1^{E1}*P2^{E2}*...*Pk^{Ek}$, unde P1, P2, ..., Pk sunt numere prime şi E1, E2, ..., Ek numere naturale mai mari ca zero. Un divizor al lui N va fi reprezentat printr-un vector de exponenţi (e1, e2, ..., ek) unde 0≤ei≤Ek. Prin urmare, problema noastra poate fi uşor redusă la următoarea cerinţă: ordonaţi toţi vectorii (e1, e2, ..., ek) unde 0≤ei≤Ek, într-un şir cu proprietatea că diferenţa între doi vectori consecutivi se realizeaza la o singura poziţie a vectorilor şi cele două elemente ale vectorilor de pe poziţia respectiva diferă prin o unitate. Exemplul din enunţul problemei se poate reprezenta astfel:
1 2 4 12 6 3
P1=2 E1=2
P2=3 E2=1
(0,0) (1,0) (2,0) (2,1) (1,1) (0,1)
Această problemă este o generalizare a determinării codului Gray şi poate fi rezolvată generalizând metoda expusă mai sus.
Presupunem că ştim să generăm o soluţie pentru $M=N/(Pk^{Ek})$ (o soluţie pentru un număr cu k-1 factori primi). Să vedem cum generăm o soluţie pentru N. Fie C vectorul soluţie pentru M şi R vectorul C inversat (primul element al lui R este ultimul al lui C, etc.). O soluţie pentru N poate fi obţinută în urmatoarea formă:
$C\ Pk*R\ Pk^2*C\ Pk^3*R\ ...$
Astfel am concatenat Ek coduri pentru M, înmulţind cu Pk şi pe fiecare poziţie impară am pus secvenţa inversată pentru M. Am făcut aceasta pentru că cele două capete ce vor ajunge în codul final consecutive să difere numai prin Pk.

Exemplu:
```
E1=3; E2=2; E3=1
(0,0,0) (1,0,0) (2,0,0) (3,0,0)
```
Soluția pentru un factor prim.
```
(0,0,0) (1,0,0) (2,0,0) (3,0,0) (3,1,0) (2,1,0) (1,1,0) (0,1,0) (0,2,0)
(1,2,0) (2,2,0) (3,2,0)
```
Soluția pentru doi factori primi.
```
(0,0,0) (1,0,0) (2,0,0) (3,0,0) (3,1,0) (2,1,0) (1,1,0) (0,1,0) (0,2,0)
(1,2,0) (2,2,0) (3,2,0)
(3,2,1) (2,2,1) (1,2,1) (0,2,1) (0,1,1) (1,1,1) (2,1,1) (3,1,1) (3,0,1)
(2,0,1) (1,0,1) (0,0,1)
```
Soluția pentru cei trei factori primi.
Complexitatea finală a soluției este O(T) unde T reprezinta numărul de divizori ai lui N.

Problema 5: Sortnet (Mihai Pătraşcu, lot 2002)
O rețea completă de sortare este o rețea de sortare cu `N` fire de adâncime `D` ce are `D*(N/2)` comparatori. Amintim principiul 0-1 care spune că o rețea de sortare sortează orice `N` numere dacă aceasta sortează orice secvență de `N` numere 0 sau 1. Problema noastră cere câte astfel de secvențe de 0 şi 1 nu sunt sortate corect.

Soluția lui Mihai Pătraşcu:
Observăm că după primul strat de comparatori sunt exact 3^(N/2) rezultate posibile, pentru că intrările trecute printr-un comparator se transformă astfel 0, 0 -> 0, 0 ; 0, 1 -> 1, 0; 1, 0 -> 1, 0; 1, 1 -> 1, 1. Pentru a testa fiecare astfel de rezultat dacă e sortat sau nu de următoarele D-1 straturi, nu vor trebui simulate toate în complexitate O(N * (3^N/2) * D). Prin generarea secvențelor conform codului Gray în baza 3 putem doar să urmărim modificarea rezultatului pentru două rezultate succesive în O(D). Un număr în baza trei îl transformăm în unul în baza doi cu număr dublu de cifre, astfel 0 trece în 00, 1 trece în 10 şi 2 trece în 11, acest cod reprezentând un rezultat posibil după evaluarea unei secvențe binare de către primul strat al rețelei. Modificarea unei cifre în baza trei conform codului Gray generalizat pentru numere în această bază, duce la modificarea unei cifre în codul binar a unui bit. Urmărirea modificărilor făcute de acest bit schimbat în evaluările făcute de rețeaua de sortare o putem face în complexitate O(D). Astfel complexitatea finală este O(3^(N/2) * D).

Menționăm că problema este pe siteul infoarena şi acolo o rezolvare O(2^N * D) intră în timp, dar şi pentru aceasta trebuie să folosim codul Gray şi să urmărim modificările date de schimbarea unui bit în evaluare.
Dăm mai jos rezolvarea elegantă şi uşor de înțeles a lui Mircea Paşoi:

```
#include <stdio.h>

#define MAX_N 20
#define MAX_M 33
#define MAX_C 59049
#define FIN "sortnet.in"
#define FOUT "sortnet.out"
#define GRAY(x) ((x) ^ ((x) >> 1))
#define BIT(a, b) (((a) & (1 << (b))) > 0)
#define MIN(a, b) ((a) < (b) ? (a) : (b))
#define MAX(a, b) ((a) > (b) ? (a) : (b))
#define SORTED(x) (((x) & ((x)+1)) == 0)
#define FOR(i, a, b) for (i = (a); i < (b); i++)
```

```
int N, M, G[MAX_M][MAX_N], V[MAX_M], V2[MAX_M], Res;

int works(int n, int a)
{
    int i, b, t;

    V2[0] = n;
    FOR (i, 0, M)
    {
        b = G[i][a];
        if ((BIT(V[i], MAX(a, b)) && !BIT(V[i], MIN(a, b))) ||
            (BIT(V2[i], MAX(a, b)) && !BIT(V2[i], MIN(a, b))))
            a = b;
        V[i] = V2[i];
        V2[i+1] = V[i+1]^(1<<a);
    }
    V[M] = V2[M];

    return SORTED(V[M]);
}

int main(void)
{
    int i, j, k, a, b, bit;

    freopen(FIN, "r", stdin);
    freopen(FOUT, "w", stdout);

    scanf("%d %d\n", &N, &M);
    FOR (i, 0, M) FOR (j, 0, N/2)
    {
        scanf("<%d,%d> ", &a, &b);
        a--; b--;
        G[i][a] = b; G[i][b] = a;
    }

    Res = 1;
    FOR (i, 1, (1<<N))
    {
        k = GRAY(i) ^ GRAY(i-1);
        for (bit = 0; (1<<bit) < k; bit++);
        Res += works(GRAY(i), bit);
    }
    printf("%d\n", Res);

    return 0;
}
```

Bibliografie:

[1] D.E.Knuth TAOC Prefascicle 2A
[2] W.H. Press et. all. Numerical Recipies in C, Cambridge University Press
[3] http://en.wikipedia.org/wiki/Gray_code