Dinamică pe stări exponențiale

Adesea, la probleme care se rezolvă prin programare dinamică, observam că numărul de stări poate depinde exponențial de anumiți parametri. De multe ori aceste stări sunt de fapt submulțimi ale unei mulțimi. În acest caz, dacă numărul de elemente ale mulțimii este mic, putem memora cumva informații despre toate cele maxim 2^n submulțimi ale sale. Este o practică de a păstra o submulțime ca un vector caracteristic.
De exemplu, dacă mulțimea are elementele {1, 2, 3, 4, 5} vectorul [1, 1, 0, 0, 1] codifică submulțimea {1, 2, 5}.
Este practic să ținem informații pentru toate stările doar dacă numărul lor nu este foarte mare și permite să construim o structură cu câte o componentă pentru fiecare stare. Astfel, codificarea internă a unui `int` (care are `32` de poziții binare) este arhisuficientă pentru a păstra toate stările de care avem nevoie cu limitările descrise mai sus. Astfel, este practic să considerăm elementele mulțimii noastre numerotate de la `0` și să codificăm vectorul caracteristic corespunzător unei submulțimi în biții unui număr. De exemplu, dacă elementele mulțimii noastre sunt {0, 1, 2, 3, 4}, pentru submulțimea {0, 1, 4} am avea nevoie de vectorul caracteristic [1, 1, 0, 0, 1], și pe acesta îl codificăm marcând biții de pe pozițiile `0`, `1` și `4` ale unui `int`. Astfel, obținem valoarea codificată intern `10011` = `19` (aici numerotăm de la dreapta, precum cum notăm cifrele la matematică).
În concluzie, întro structură a noastră `V`, pe poziția `19`, ținem informații despre submulțimea {0, 1, 4}.

Probleme rezolvate

1. Poly, infoarena (https://www.infoarena.ro/problema/poly)

Enunțul pe scurt:
Se dă un șir `V` cu `N` elemente (maxim `30000`). Se consideră mulțimea `M` = {`2, 3, 7, 11, 19, 23, 37`}.
Se cere să aflăm lungimea maximă a unui subșir format cu elemente din `V` și în care oricare două elemente *vecine* ale subșirului nu au divizori comuni dintre elementele lui `M`.

Soluția 1
O primă abordare este asemănătoare cu determinarea subșirului crescător de lungime maximă (varianta de complexitate $o(n^2)$).
Notăm `D[i]` = lungimea maximă a unui subșir format cu elemente din `V` și în care ultimul element este cel de pe poziția `i`.
Pentru a calcula `D[i]` ne gândim după cine putem pune imediat în soluție pe `V[i]`. Astfel, ne interesează poziții `j`, cu (`j<i`), cu proprietatea că `V[i]` și `V[j]` nu au divizori comuni din `M`. Considerăm valoarea `D[j]` maximă dintre aceste elemente și obținem `D[i]` ca fiind `1` plus acest maxim.
Această abordare are complexitatea $n^2*card(M)$, deoarece avem nevoie de încă o trecere prin elementele lui `M` la verificarea dacă `V[i]` și `V[j]` au elemente comune.
Putem scăpa de această trecere cu următoarea observație (utilă și pentru soluția optimă, prezentată după aceasta): nu este necesar să memorăm elementele lui `V` ci, pentru fiecare, să știm ce divizori din mulțimea `M` are. Astfel, fiecare element din `V` poate fi înlocuit cu un număr a cărui scriere în baza `2` corespunde unui vector caracteristic care reprezintă factorii primi din `M` pe care îi conținea acel număr din `V`.

Astfel, verificarea dacă `V[i]` și `V[j]` au factori comuni dintre elementele lui `M` se face printr-un "și pe biți": `V[i] & V[j]`.

Soluția 2.
Când ajungem să luăm în calcul elementul de pe poziția `i` din `V`, avem calculate optime cu subșiruri care conțin elemente de pe pozițiile `1`, `2`, ..., `i-1` din `V`. Schimbăm puțin abordarea și observăm că dacă două elemente din `V` sunt codificate identic, nu ne mai interesează lungimea maximă a unui subșir terminat cu fiecare dintre ele ci lungimea maximă a unui subșir terminat într-un element ce are codificarea comună a lor.
Astfel, la pasul `i`, `D[stare]` va avea semnificația: lungimea maximă a unui subșir format cu elemente dintre primele `i` din `V` și în care ultimul element al subșirului este egal cu `stare` (reamintim că elementele șirului nu mai considerăm că sunt valorile din fișier ci codificarea binară a divizorilor pe care îi au dintre numerele din `M`).
Atunci când ajungem la une element `V[i]`, vom actualiza doar elementul `D[ V[i] ]`. Semnificația este că ne interesează doar subșiruri care se vor termina cu el. Astfel, căutăm stări pentru care nu există factori primi comuni cu `V[i]`, iar `D[i]` va fi `1 + maxim (D[stare])` cu proprietatea că `stare&i` dă `0`.

```
#include <fstream>

using namespace std;

/// memorăm numerele din M pe poziții de la 0 la 6.
int M[] = {2, 3, 7, 11, 19, 23, 37};

int D[128], v[30010], sol, i, j, n, maxim, a;
// D[i][stare] = lungimea maxima a unui subsir
// care foloseste elemete dintre primele i
// si in care ultimul element are condiguratia
// de divizori data de stare

int main () {
    ifstream fin ("poly.in");
    ofstream fout("poly.out");

    fin>>n;

    for (i=1;i<=n;i++){
        fin>>a;
        for (j=0;j<7;j++)
            if (a % M[j] == 0)
                v[i] += (1 << j);
    }

    D[ v[1] ] = 1;
    for (i=2;i<=n;i++) {
        /// la analizarea celui de-al i-lea elementactualizăm D[v[i]]
        for (j=0;j<128;j++) {
            if (( j & v[i] ) == 0) {
                D[ v[i] ] = max (D[ v[i] ], D[j] + 1);

            }
        }
    }

    /// după ce am luat în calcul toate elementele din v, avem în D cel mai lung subșir
    /// terminat în fiecare configurație posibilă (deci ne interesează maximul din D)
```

```
    for (j=0;j<128;j++)
        sol = max(sol, D[j]);


    fout<<sol;
    return 0;
}
```

Soluția de mai sus are timpul de calcul de ordinul $n * 2^{card(M)}$. În acest caz este mai convenabil decât n^2 deoarece `card(M)` etste `7`.

Alte observații:
La pasul `i` nu mai avem în dinamică semnificația de la “Soluția 1”, adică “terminat în `V[i]`” ci “dintre primele `i` elemente”.
Am fi putut fi tentați să construim `D[i][stare]` = lungimea maximă a unui subșir format din primele `i` elemente și terminat cu un element ce are configurația “`stare`”. Dar observăm că la pasul `i` am avea de actualizat un singur element al lui `D` față de pasul `i-1`, așa că nu mai este necesar să păstrăm o copie a tuturor celorlalte elemente de la pasul anterior și modificăm mereu în același vector `D`. Adică, primul indice nu este necesar (și reducem astfel memoria de `n` ori).

2. Morcovi, infoarena (https://www.infoarena.ro/problema/morcovi)

Enunțul pe scurt.
Avem un șir `V` cu `N` numere naturale (maxim `1000`). Se mai dă un șir `S` cu `P` valori naturale (`P` <= `12`). Trebuie să alegem o poziție de pornire din `V`, să executăm toate cele `P` salturi (fiecare exact o dată), însă într-o ordine aleasă de noi. Trebuie să maximizăm suma valorilor pe care aterizăm în urma salturilor (dacă pe o poziție `i` se ajunge de mai multe ori în urma aplicării salturilor, valoatea `V[i]` se adună de fiecare dată la sumă). Un salt de lungime `L` făcut din poziția `i`, face să se ajungă fie pe poziția `i+L`, fie pe poziția `i-L` (fără să ieșim din vector).

La un moment dat ne gândim că suntem la o poziție `i` și ne interesează suma obținută din elementele pe care am aterizat deja. Bineînțeles că este important și ce salturi am executat, pentru că este posibil ca în acea poziție să ajungem cu mai multe variante de a fi executat salturile și asta este important pentru ce mai putem acumula la sumă.

Astfel, putem calcula: `D[i][stare]` = suma maximă obținută să fim la poziția `i` și să fi executat salturile corespunzătoare biților de `1` din “`stare`”.
Inițial, `D[i][0]` = `V[i]` (putem pleca din orice poziție, valoarea din poziția de plecare se adună la sumă și nu am executat niciun salt - adică în `stare` nu avem setați biți de `1`).

Dacă suntem în poziția `[i][stare]`, căutăm să facem încă un salt. Astfel, căutăm biți de `0` în `stare` (între pozițiile `0` și `P-1`). Pentru o poziție `k` cu bit `0`, însemnă că putem face un salt de lungime `S[k]`.
Dacă facem la stânga acest salt, ajungem în configurația `[nextPoz][stare + 2^k]`, cu `nextPoz = i-S[k]`
Dacă facem la dreapta acest salt, ajungem în configurația `[nextPoz][stare + 2^k]`, cu `nextPoz = i+S[k]`

Astfel, la valoarea `D[nextPoz][stare + 2^k]` contribuie valoarea `D[i][stare] + V[nextPoz]`.

Ne interesează să ajungem la final oriunde în vector, dar să fi făcut toate salturile (toți biții din `stare` să fie `1`).
Pentru asta calculăm maximul dintre valorile `D[i][2^p-1]`.
Timpul de calcul este de ordinul: `n * 2^p * p`.

```
#include <fstream>
#define DIM 1005
using namespace std;

int v[DIM];
int d[1<<12][DIM];
int s[13];
int n, sol, i, j, k, p;

int main () {
    ifstream fin ("morcovi.in");
    ofstream fout("morcovi.out");

    fin>>n;
    for (i=1;i<=n;i++)
        fin>>v[i];
    fin>>p;
    for (i=0;i<p;i++)
        fin>>s[i];

    for (i=1;i<=n;i++)
        d[0][i] = v[i];
    int stmax = (1<<p);

    for (j=0;j<stmax;j++) {
        for (i=1;i<=n;i++) {
            // ma aflu in pozitia i si in starea j (am facut salturile date de bitii 1 din j)
            // actualizez alte pozitii adaugand la j cate un bit de 1 pe orice pozitie
            // in care in j este bit 0 (mai fac un salt dintre cele disponibile)
            for (k=0;k<p;k++)
                if (!((j>>k)&1)){
                    // din starea j pot sari cu saltul k
                    // sar la stanga
                    int nxt = i-s[k];
                    if (nxt >= 1) {
                        d[ j+(1<<k) ][nxt]= max(d[ j+(1<<k) ][nxt], d[j][i] + v[ nxt ]);
                    }
                    // sar la dreapta
                    nxt = i+s[k];
                    if (nxt <= n) {
                        d[ j+(1<<k) ][nxt]= max(d[ j+(1<<k) ][nxt], d[j][i] + v[ nxt ]);
                    }
                }
        }
    }

    for (i=1;i<=n;i++)
        sol = max(sol, d[(1<<p)-1][i]);
    fout<<sol;
    return 0;
}
```

3. Smin, infoarena (https://www.infoarena.ro/problema/smin)

Enunțul pe scurt.
Avem `N` cuișoare (maxim `17`) înfipte într-o placă de lemn (`N` puncte în plan, de coordonate laticeale). Nu există `3` cuișoare coliniare. Trebuie să înfășurăm cuișoarele cu benzi elastice bine întinse - fiecare dintre acestea să formeze un poligon convex cu cel puțin `3` vârfuri - încât orice cuișor să se afle fie în vârful, fie în interiorul unui astfel de poligon. Trebuie să minimizăm suma ariilor tuturor poligoanelor (dacă o zonă face parte din mai multe poligoane, aria sa se contorizează de fiecare dată).

Soluție
O observație de bază este că putem folosi ca poligoane doar triunghiuri (nu avem limită de număr de poligoane folosite și orice poligon se poate triangulariza).

La un moment dat ne gândim că am înfășurat anumite cuișoare. Astfel, pentru mulțimea `stare` a celor înfășurate, ținem `D[stare]` = *suma minimă a ariilor unor poligoane care au vârfurile dintre punctele din stare (și care acoperă exact punctele din stare)*.

Pentru a calcula `D[stare]`, alegem pozițiile a `3` biți de `1` din stare și ne gândim că ultima dată am înfășurat triunghiul cu vârfurile în aceste vârfuri (să notăm `x1`, `x2`, `x3` pozițiile acestor vârfuri - numerotare de la `0`).

Astfel, dacă niciunul dintre aceste vârfuri nu fusese înfășurat anterior, la `D[stare]` contribuie `D[stare-`2^{x1}`-`2^{x2}`-`2^{x3}`]` + *aria triunghiului cu vârfurile în punctele* `x1`, `x2`, `x3`.
Este însă posibil ca unele dintre cele `3` vârfuri să fi fost înfășurate deja înainte. De exemplu dacă `x1` și `x3` erau deja înfășurate, la `D[stare]` contribuie `D[stare-`2^{x2}`]` + *aria triunghiului cu vârfurile în punctele* `x1`, `x2`, `x3`.
Sau dacă doar unul dintre vârful `x1` ar fi fost deja înfășurat înainte, la `D[stare]` contribuie `D[stare-`2^{x2}`-`2^{x3}`]` + *aria triunghiului cu vârfurile în punctele* `x1`, `x2`, `x3`.
Cele de mai sus sunt valide pentru că un cuișor poate fi inclus ca vârf în mai multe înfășurări.

Analizând cele de mai sus, observăm că pentru a calcula o stare avem nevoie să știm stări ale căror codificări binare reprezintă numere cu valori mai mici. Astfel, este suficient să considerăm stările în ordine crescătoare a valorii în baza `10` a codificării.

Trebuie evitat să luăm în calcul stări cu `0`, `1` sau `2` biți de `1` (regula este să înfășurăm mereu `3` cuișoare odată).

Sursa

```
#include <fstream>
#include <iomanip>
#define INF 1000000000
```

```
using namespace std;

struct punct {
    int x;
    int y;
};
int b[140010];

int aria(punct p1, punct p2, punct p3) {
    int r = (p2.x - p1.x) * (p3.y - p1.y) - (p3.x - p1.x) * (p2.y - p1.y);
    if (r > 0)
        return r;
    else
        return -r;
}

int D[140000], x[20];
int n, i, k, i1, i2, i3, add;
punct p[20];

int main () {
    ifstream fin ("smin.in");
    ofstream fout("smin.out");

    fin>>n;
    for (i=0;i<n;i++) {
        fin>>p[i].x>>p[i].y;
    }

    /// precalculăm numărul de biți de 1 pentru fiecare valoare care ne interesează
    for (i=1;i<(1<<n);i++)
        b[i] = b[i/2] + i%2;

    for (i=0; i<(1<<n);i++) {
        if (b[i] < 3) {
            D[i] = INF;
            continue;
        }

        k = 0;
        for (int j=0;j<n;j++)
            if ((i >> j) & 1) {
                x[++k] = j;
            }

        if (b[i] == 3) {
            D[i] = aria( p[ x[1] ], p[ x[2] ], p[ x[3] ] );
            continue;
        }

        D[i] = INF;
        for (i1 = 1; i1 <= k-2; i1++)
            for (i2 = i1+1; i2<=k-1; i2++)
                for (i3 = i2+1; i3 <= k; i3++) {
                    add = aria(  p[ x[i1] ], p[ x[i2] ], p[x [i3] ]);

                    /// niciunul dinntre cele 3 nu a fost inconjurat
                    D[i] = min(D[i], add + D[i-(1<<x[i1]) - (1<<x[i2]) - (1<<x[i3]) ]);
                    /// consider ca x[i1] si x[i2] erau deja imprejmuite
                    D[i] = min(D[i], add + D[i - (1<<x[i3])]);
                    D[i] = min(D[i], add + D[i - (1<<x[i2])]);
```

```
                    D[i] = min(D[i], add + D[i - (1<<x[i1])]);
                    D[i] = min(D[i], add + D[i-(1<<x[i1]) - (1<<x[i2])]);
                    D[i] = min(D[i], add + D[i-(1<<x[i1]) - (1<<x[i3])]);
                    D[i] = min(D[i], add + D[i-(1<<x[i2]) - (1<<x[i3])]);
                }
    }

    fout<<setprecision(1)<<fixed<<D[(1<<n) - 1]/2.0<<"\n";

    return 0;
}
```

În sursa de mai sus am lucrat cu dublul ariilor, și doar la final am făcut împărțire reală la `2` (pentru a evita calculul intermediar cu numere reale).

Prof. Marius Nicoli
Slatina, 10 mai 2023