Soluţie imax

Soluţia are ca principiu o programare dinamică în $O(n^3)$. Se fixează o coloană `ic1` reprezentând coloana de start a dreptunghiului de mijloc. Se parcurg liniile începând cu n către `1`(variabila `ir`),construindu-se un vector `T` .

Vectorul `T` va reţine în elementul `T[i]` aria unui obiect format din ultimele 2 subtablouri, care se extinde pană la linia 1, pentru care coloanele dreptunghiului de mijloc sunt situate între `ic1` şi `i` (obiectul descrie forma literei `T` răsturnate).

Pentru cei doi indici fixaţi `ic1` şi `ir` se parcurge matricea de două ori: o dată se parcurg liniile de la `ir + 1` la `n` pentru actualizarea elementelor lui `T`, după care revine cu o nouă parcurgere, de la `ir - 1` la `1` pentru adăugarea subtabloului `1`.

```
#include <stdio.h>
#include <algorithm>

using namespace std;

#define MAX_N 155
#define FIN "imax.in"
#define FOUT "imax.out"

int N, M, Res, A[MAX_N][MAX_N], L[MAX_N][MAX_N], R[MAX_N][MAX_N];

int main(void)
{
    int i, j, ic1, ic2, ir, br, tr, best, l, r;

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

    scanf("%d %d", &N, &M);
    fill(A[0], A[0]+M+2, 1);
    fill(A[N+1], A[N+1]+M+2, 1);
    for (i = 1; i <= N; i++)
    {
        A[i][0] = A[i][M+1] = 1;
        for (j = 1; j <= M; j++)
            scanf("%d", A[i]+j);
    }

    for (i = 1; i <= N; i++)
    {
        L[i][0] = 0;
        for (j = 1; j <= M; j++)
            L[i][j] = A[i][j] ? j : L[i][j-1];
        R[i][M+1] = M+1;
        for (j = M; j >= 1; j--)
            R[i][j] = A[i][j] ? j : R[i][j+1];
    }

    for (ic1 = 1; ic1 <= M; ic1++)
    {
        int T[MAX_N] = {0};
        for (ir = N; ir >= 1; ir--)
        {
            for (i = R[ir][ic1]; i <= M; i++) T[i] = 0;
            if (A[ir][ic1]) continue;
            ic2 = R[ir][ic1]-1; best = 0;
            l = 1; r = M;
```

```
            for (br = ir+1; br <= N; br++)
            {
                l = max(l, L[br][ic1]+1);
                r = min(r, R[br][ic1]-1);
                if (l >= ic1 || r <= ic1) break;
                if (r <= ic2)
                {
                    if (best)
                        for (i = r; i <= ic2; i++)
                            T[i] = max(T[i], best+ir*(i-ic1+1));
                    ic2 = r-1;
                }
                best = max(best, (br-ir)*(r-l+1));
            }
            if (best)
                for (i = ic1; i <= ic2; i++)
                    T[i] = max(T[i], best+ir*(i-ic1+1));

            ic2 = R[ir][ic1]-1; best = 0;
            l = 1; r = M;
            for (tr = ir-1; tr >= 1; tr--)
            {
                l = max(l, L[tr][ic1]+1);
                r = min(r, R[tr][ic1]-1);
                if (l >= ic1 || r <= ic1) break;
                if (r <= ic2)
                {
                    if (best)
                    for (i = r; i <= ic2; i++)
                        if (T[i]) Res = max(Res, best+T[i]-(ir-1)*(i-ic1+1));
                    ic2 = r-1;
                }
                best = max(best, (ir-tr)*(r-l+1));
            }
            if (best)
                for (i = ic1; i <= ic2; i++)
                    if (T[i]) Res = max(Res, best+T[i]-(ir-1)*(i-ic1+1));
        }
    }

    printf("%d\n", Res);

    return 0;

}
```

Soluţie Casute

Problema cere a se determina intr-un graf aciclic cel mai mic stramos comun intre oricare doua noduri. Acesta este unic datorita faptului ca inaltimile sunt distincte.
O solutie $O(N^3)$ este evidenta si ar fi obtinut maxim 40-50 de puncte cu diverse optimizari. Pentru a obtine 100 de puncte este necesara o solutie $O(N*M)$. La primul pas, se calculeaza pentru fiecare nod din graf o lista de noduri care pot fi accesate din acest nod. Aceastea se pot calcula folosind N parcurgeri DF sau BF.
Fie n un nod cu grad interior 0 in graf si $L[n]$ lista nodurilor accesibile din n. Este clar ca, initial,
$LCA(n, L[n][i])=n$ pentru orice i. Pentru fiecare nod adiacent m cu n vom actualiza $LCA(m, x)$ cu $LCA(n, x)$ pentru fiecare nod x din graf si vom scoate nodul n din graf. In implementare, se foloseste o coada cu nodurile care au gradul 0 in momentul respectiv.

Soluţie repeat

Solutiile pentru primele doua teste se genereaza usor folosind metoda backtracking implementata intr-o maniera bruta. Insa pentru a obtine solutii optime pentru toate testele este nevoie de niste rafinamente.
Se observa ca numarul minim de secvente repetate distincte este suficient de mic (pentru toate testele mai mic sau egal cu 3). Astfel ne gandim sa limitam aceasta valoare de la bun inceput si de a opri cautarea atunci cand numarul de secvente repetate distincte o depaseste. In acest sens se poate folosi containerul map din STL pentru a calcula acest numar la fiecare pas. Totusi aceasta metoda va fi destul de lenta si nu va putea genera solutiile optime intr-un timp rezonabil decat pentru primele 5 teste.
Pentru a accelera cautarea se poate mentine tot un map care contine pentru fiecare secventa repetata numarul de aparitii distincte ale acesteia. La fiecare pas al cautarii structura se actualizeaza in mod corespunzator (in functie de ultima valoare adaugata la sirul generat pana la pasul respectiv) si se verifica daca satisface limita impusa.
Pentru testul maxim solutia se genereaza in mai putin de un minut.

```
#include <cstdio>
#include <cstdlib>
#include <cstring>
#include <set>
#include <string>
#include <map>
#include <algorithm>
using namespace std;

map<string, int> A;
int a[20005];
int N, BMAX, i, j, stp, Res = 9999999, Ans;
string aux;

int eq(int i, int l)
{   int j;
    for (aux = "", j = 0; j < l; j++)
    {
        if (a[i+j] != a[i+l+j]) return 0;
        aux += '0'+a[i+j];
    }
    return 1;
}
```

```
void add(int x)
{
    int l, i;
    map<string, int> :: iterator it;
    for (l = 1; l <= (x+1)/2; l++)
        if (eq(x-2*l+1, l))
        {
            it = A.find(aux);
            if (it == A.end())
                A[aux] = 1;
            else
                ++it->second;
        }
}

void rem(int x)
{
    int l, i;
    map<string, int> :: iterator it;
    for (l = 1; l <= (x+1)/2; l++)
        if (eq(x-2*l+1, l))
        {
            it = A.find(aux);
            if (it == A.end()) { printf("not good\n"); exit(0); }
            if (it->second == 1)
                A.erase(it);
            else
                --it->second;
        }
}

void dfs(int x)
{
    if (x == N)
    {
        for (int i = 0; i < N; i++)
            printf("%d", a[i]);
        printf("\n");
        exit(0);
    }

    a[x] = 0;
    add(x);
    if (A.size() <= BMAX) dfs(x+1);
    rem(x);

    a[x] = 1;
    add(x);
    if (A.size() <= BMAX) dfs(x+1);
    rem(x);
}

int main(int argc, char* argv[])
{
    freopen(argv[3], "w", stdout);
    N = atoi(argv[1]), BMAX = atoi(argv[2]);
    dfs(0);
    return 0;
}
```