## Zidar – Descrierea soluţiei

Aceasta problema se rezolva folosind metoda programarii dinamice.

O solutie triviala este calcularea matricei 4-dimensionale M[i][j][k][l] = numarul maxim de caramizi pe care il poate contine un zid al carui al i-lea strat este cuprins intre coloanele j si k, avand pretul total de constructie egal cu l.

Din pacate, l poate deveni suficient de mare cat sa nu permita o rezolvare eficienta.

Imbunatatirea adusa este schimbarea punctului de vedere, si anume calcularea matricei M[i][j][k][l] = pretul minim al unui zid al carui al i-lea strat este cuprins intre coloanele j si k, folosind l caramizi. Solutia va fi valoarea l maxima (evident, mai mica sau egal cu X) pentru care exista un M[i][j][k][l] $\leq$ T

Pentru aceasta, o solutie banala de complexitatea O(M*$N^4$*X) obtine aproximativ x% din punctaj.

Mentinand pentru fiecare nivel o matrice aditionala Z[i][j] = costul minim al unui zid care se termina pe nivelul respectiv, contine caramida #i si contine in total j caramizi, se poate ajunge la o solutie de complexitate O(M*$N^3$ * X) care obtine punctajul maxim.

De notat ca este posibila reducerea memoriei folosite de la O(M*$N^2$*X) la O($N^2$*X), folosind doar ultima “linie” a matricei 4-dimensionale de mai sus.

## Apel – Descrierea soluţiei

Pentru evaluarea unui apel de funcţie vom înlocui în ordine parametrii formali cu parametrii actuali, apoi vom evalua expresia aritmetică ce explicitează funcţia.

## Excursie – Descrierea soluţiei

Definim tabloul ef, de dimensiuni n x m, cu semnificaţia: ef[i][j] este efortul minim necesar pentru a ajunge la linia i şi coloana j, pe un traseu care porneşte din poziţia iniţială. Tabloul se inţializează cu valori foarte mari, mai puţin valoarea corespunzătoare poziţiei de plecare, care este 0. Starea curentă este definită de: efortul minim de la poziţia de start la cea actuală, poziţia în matrice şi distanţa faţă de punctul de plecare. Se expandează starea curentă. Se încearcă obţinerea unui efort mai mic până la una dintre poziţiile adiacente, pe un traseu care trece prin starea curentă. La acelaşi efort minim obţinut, starea vecină se actualizează, dacă se obţine o distanţă mai mică. Pentru memorarea stărilor curente şi a celor adiacente, se pot utiliza două cozi. În prima coadă se introduce starea iniţială (practic doar poziţiile i şi j). Stările adiacente, care suferă actualizări, se introduc în coada a doua. Când toate stările din prima coadă s-au expandat, se suprascriu valorile din prima coadă cu cele din coada a doua. Acestea din urmă devin astfel stări curente şi se reia procedeul. Se afişează ef[lf][cf].

O implementare de tip backtracking poate obţine 50-70% din punctaj.

Sursa: `zidar.c, zidar.cpp, zidar.pas`