Tema de casă
-Drum in grid-

Trandafir Daniela-Georgiana

May 2020

Profesor: Costin Bădică
Profesor laborator: Cristinel Ungureanu

- Specializarea : Calculatoare cu predare în limba Română
- Anul : *I*
- Grupa : 1.3 B

UCV Facultatea de Automatică,Calculatoare şi Electronică

1 Enunţul Problemei

Drum in grid
Se consideră un teren în formă de grid pătratic de dimensiuni $N \times N$. Fiecare locaţie din grid este caracterizată printr-un număr întreg pozitiv ce reprezintă cota locaţiei respective (înălţimea unui punct de pe teren faţă de un plan orizontal de referinţă). Se cere găsirea unui drum din colţul din stânga-sus al terenului până în colţul din dreapta-jos al terenului astfel încât:

i) deplasarea de-alungul drumului se face doar către dreapta sau în jos;

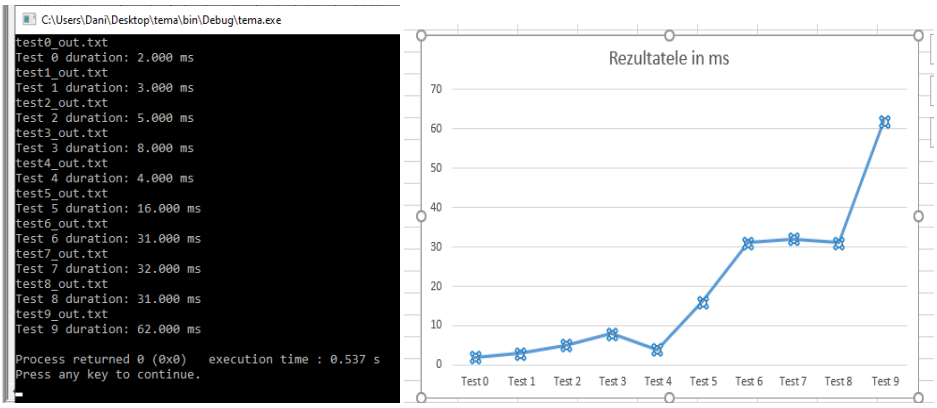
ii) costul drumului, calculat ca suma valorilor absolute ale diferenţelor dintre cotele consecutive de-alungul drumului, să fie minimă. Se vor implementa doi algoritmi diferiţi.

2 Algoritmi

2.1 Algoritm I

min_cost(n)
1.$inf \leftarrow 2000000000$
2.**For** $i = 0, n$ **do**
3. $sum[i][0] \leftarrow inf$
4. $sum[0][i] \leftarrow inf$
5. 6. $sum[0][1] \leftarrow 0$, $sum[1][0] \leftarrow 0$
7.**For** $i = 1, n$ **do**
8. **For** $j = 1, n$ **do**
9. $sum[i][j] \leftarrow minim(absolut(cost[i][j], cost[i][j-1]) + sum[i][j-1]$,
absolut(cost[i][j], cost[i-1][j]) + sum[i-1][j])
10. $result \leftarrow sum[n][n] - cost[1][1]$
11.**Return:** result

Complexitatea algoritmului este: $O(n^2)$
În urma generării testului pentru timpul de execuţie am obtinut următoarele rezultate:

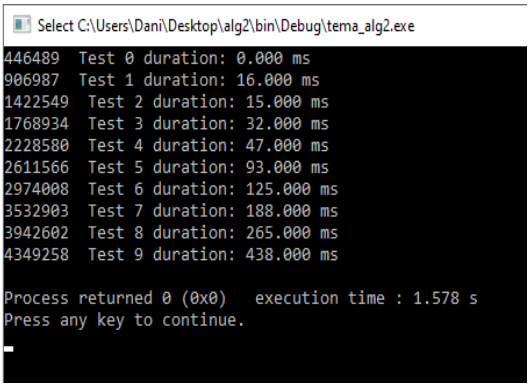


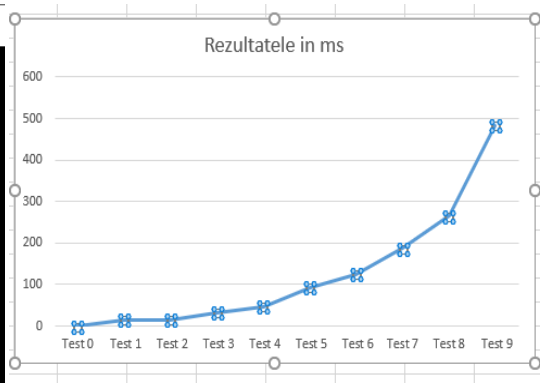
2.2 Algoritm II

solve(n)
$inf \leftarrow 2000000000$
$head.next \leftarrow NULL$
For $i = 0, n-1$
    For $i = 0, n-1$
        $d[i][j] \leftarrow inf$
$d[0][0] \leftarrow inf$
$aux.first \leftarrow 0, aux.second \leftarrow 0$
push_element_end(head,aux)
While list_empty(head) = 1 **do**
    $aux \leftarrow$ **pop_element_begining**($head$)
    $i \leftarrow aux.first$
    $j \leftarrow aux.second$
    If $i + 1 < n$
        If $d[i+1][j] = inf$
            $aux.first \leftarrow i + 1$
            $aux.second \leftarrow j$
            ***p*ush_element_end**(head,aux)
    d[i+1][j] = minim(d[i+1][j], d[i][j]+absolut(a[i][j], a[i+1][j]))

    if j+1 <n **then**
        if d[i][j+1] = inf **then**
            $aux.first \leftarrow i$
            $aux.second \leftarrow j + 1$
            ***p*ush_element_end**(head,aux)
    d[i][j+1] = minim(d[i][j+1], d[i][j]+absolut(a[i][j], a[i][j+1]))
Return: d[n-1][n-1]

Complexitatea algoritmului este: $O(n^2)$
În urma generării testului pentru timpul de execuţie am obtinut următoarele rezultate:





3 Date experimentale

generator(n)
$n \leftarrow (test + 1) * 50$
for $linie = 1, n$ **do**
for $coloana = 1, n$ **do**
rand()

Pentru generarea automată am creat un modul **generator.c** pe care l-am introdus ca funcţie în ambele programe.
Algoritmul generează dimensiunea matricei **n x n**, un număr direct proporţional cu numărul testului , şi apoi valorile fiecărui element, numere aleatorii între 0 si RAND_MAX. Dat fiind intervalul de generare al funcţiei **rand()** $(0, 2^{15} - 1)$ rezultă faptul că datele de intrare sunt valide pentru problema dată.

4 Proiectarea aplicaţiei experimentale

4.1 Structura

Aplicaţia creată este organizată în module, fiecare conţinând funcţii particulare.

4.2 Date de intrare

Datele de intrare sunt sub forma unei matrice pătratice.Fiecare element din matrice reprezintă o locaţie in grid si este caracterizat printr-un numar întreg pozitiv ce reprezintă cota locaţiei respective.

4.3 Date de ieşire

Programul generează o valoare minimă ce reprezintă costul drumului, calculat ca suma valorilor absolute ale diferenţelor dintre cotele locaţilor consecutive de-alungul drumului.

4.4 Modulele aplicatiei

⇒ În cadrul funcţei main se citesc datele de intrare: valoarea <n> ce reprezintă dimensiunea matricei si apoi toate valorile din matrice. Se apelează apoi funcţia pentru generarea rezultatului şi se afisează rezultatul.
⇒ În cadrul funcţiei homework1.c, respectiv homework2.c pentru al doilea algoritm se aplează toate funcţiile necesare:
I
minim - funcţie care returnează minimul dintre două numere naturale;
absolut - functie care returnează diferenţa absolută dintre două numere;
solve - funcţia care creează o matrice ce va fi populată cu costurile minime pentru a ajunge la fiecare poziţie. Aceasta returnează valoarea finală prin parametrul <solve>.

II

minim - funcţie care returnează minimul dintre două numere naturale;
absolut - functie care returnează diferenţa absolută dintre două numere;
push_element_begining - funcţie care adaugă elemente la începutul unei cozi;
push_element_end - funcţie care adaugă elemente la finalul unei cozi;
pop_element_begining - funcţie care şterge elemente de la finalul cozii;
list_empty - funcţie care returnează valoarea 1 daca este populată, respectiv valoarea 0 dacă nu este populată;
solve - functia care populează o matrice cu o valoare foarte mare si apoi cu o căutare Breadth First Search adaugă in coada noduri si află valoarea minimă pentru fiecare element. Rezultatul final se află in matricea creată: d[n-1][n-1];
⇒ În cadrul functiei generator.c se genează valorile semnificative randomizate pentru fiecare dintre teste.

5 Rezultate & Concluzii

5.1 Rezultate

Input 0:
solve1: 7 ms solve2: 3 ms
Input 1:
solve1: 19 ms solve2: 27 ms
Input 2:
solve1: 47 ms solve2: 71 ms
Input 3:
solve1: 100 ms solve2: 139 ms
Input 4:
solve1: 119 ms solve2: 175 ms
Input 5:
solve1: 167 ms solve2: 269 ms
Input 6:
solve1: 291 ms solve2: 475 ms
Input 7:
solve1: 367 ms solve2: 503 ms
Input 8:
solve1: 617 ms solve2: 967 ms
Input 9:
solve1: 639 ms solve2: 802 ms

Am observat că soluţia care foloseşte o coadă are un timp mai mare de execuţie faţă de prima soluţie. Acest lucru este datorat faptului că implentarea cozii alocă memoria diferit faţă de rezolvarea în care folosesc matrice.

5.2 Concluzii

Această temă m-a ajutat să înteleg mai bine limbajul C şi în special limbajul Python în care mai lucrasem foarte puţin până acum şi să îmi aprofundez cunoştinţele pentru creearea de programe modulare.

În creearea temei am întâmpinat dificultăţi atunci când am generat testele si le-am populat cu datele semnificative pentru problema mea, dar am reuşit sa scriu şi ulterior sa citesc datele din fiecare.
În Python am reuşit să salvez input-urile şi output-urile într-un folder numit tests creat în interiorul programului.

6 Referinţe bibliografice

1. Data Structures and Algorithms in Python- autori: Michael T. Goodrich,Roberto Tamassia, Michael H. Goldwasser
2. *GeeksforGeeks*
3. *PythonDocs*
4. *Overleaf*
4. Introduction to Algorithms-autori: H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein