Naumencu Mihai

322 CD

Tema 0 – Proiectarea algoritmilor

1. Tehnica Divide et Impera

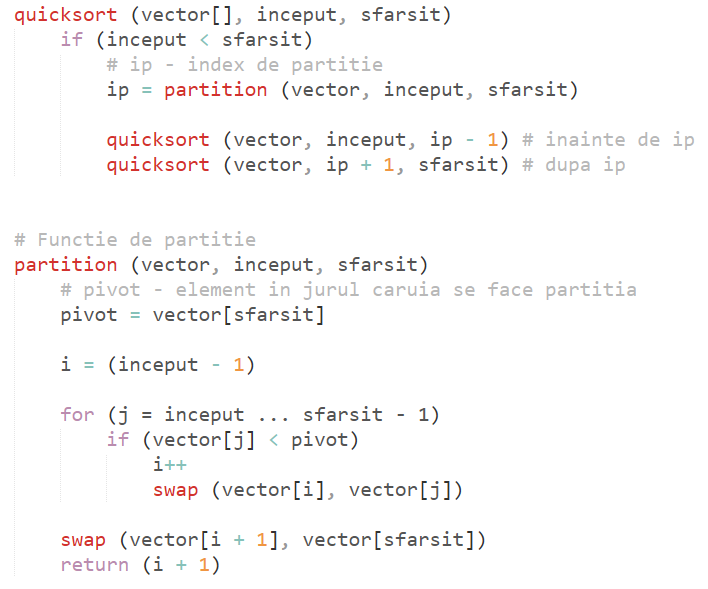
* Sortarea consta in rearanjarea unei structuri de date, astfel incat structura rezultata in urma aplicarii algoritmului de sortare este ordonata crescator (sau descrescator).

**Enunt**: Sa se sorteze crescator un vector.

* Algoritmul de sortare ales pentru rezolvarea problemei sortarii\ este quicksort.

**Quicksort** este un algoritm de sortare implementat folosind tehnica divide et impera. In cadrul acestuia se alege un element (pivotul) si se realizeaza o partitie a vectorului in jurul elementului ales.

* **Pseudocod** (implementare cu pivot ca ultim element):



* In general, **complexitatea** algoritmului poate fi exprimata astfel:

T(n) = T(k) + T(n – k – 1) + Θ(n),

unde primii doi termeni reprezinta cele 2 apeluri recursive, ultimul reprezinta operatia de partitie, 'k' reprezinta numarul de elemente mai mici decat pivotul.

Timpul de executie depinde de vectorul primit si de implementarea functiei de partitie.

Cazuri:

* In cel mai bun caz: T(n) = 2T(n / 2) + Θ(n)

=> Θ(n log(n)),

cand operatia de partitie alege intotdeauna elementul mijlociu (numar egal de elemente mai mici si mai mari decat acesta) ca pivot.

* In cel mai rau caz: T(n) = T(n – 1) + Θ(n)

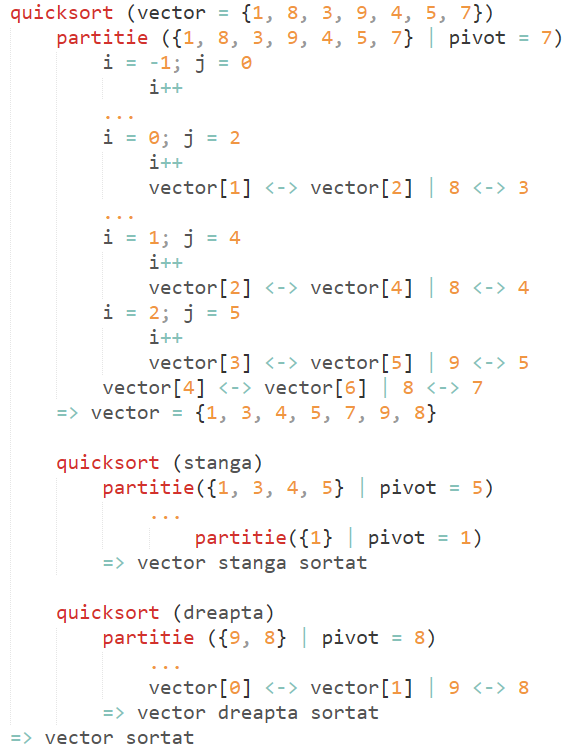
=> Θ(n2),

cand operatia de partitie alege intotdeauna cel mai mic sau cel mai mare element ca pivot. Pentru implementarea de mai sus (ultimul element este ales ca pivot), cel mai rau caz va fi atunci cand vectorul este deja sortat.

* Quicksort este unul dintre cei mai eficienti algoritmi de sortare, de asemenea fiind unul dintre cei mai folositi. Mai multe variante ale acetui algoritm exista, in functie de pivotul selectat (pivotul sa fie primul element intotdeauna, ultimul element intotdeauna, un element random sau elementul aflat in mijlocul vectorului), acestea fiind similare in general, dar in cazuri specifice o implementare poate fi superioara celorlalte.

Algoritmul poate fi optimizat prin impunerea unor conditii atunci cand se efectueaza apelurile recursive: in loc sa se faca call mai intai pe portiunea stanga (de la inceput pana la pivot – 1), se face call mai intai pe portiunea mai mica. De asemenea, implemetarea standard mai poate fi optimizata implementand o stiva proprie.

* **Exemplificare** pe un vector = {1, 8, 3, 9, 4, 5, 7}:

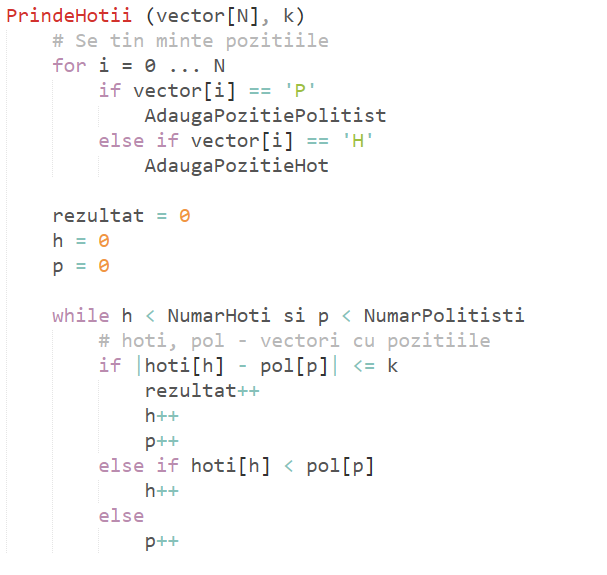


1. Tehnica Greedy

* **Enuntul Problemei** (Politistii si Hotii): Se da un vector format din elemete ce pot fi de 2 tipuri: politisti sau hoti, fiecare politist poate prinde maxim 1 hot la maxim o distanta k de acesta. Sa se gaseasca numarul maxim de hoti ce pot fi prinsi.

(sursa: <https://www.chegg.com/homework-help/questions-and-answers/given-array-size-n-following-specifications-element-array-contains-either-policeman-thief--q52795788>)

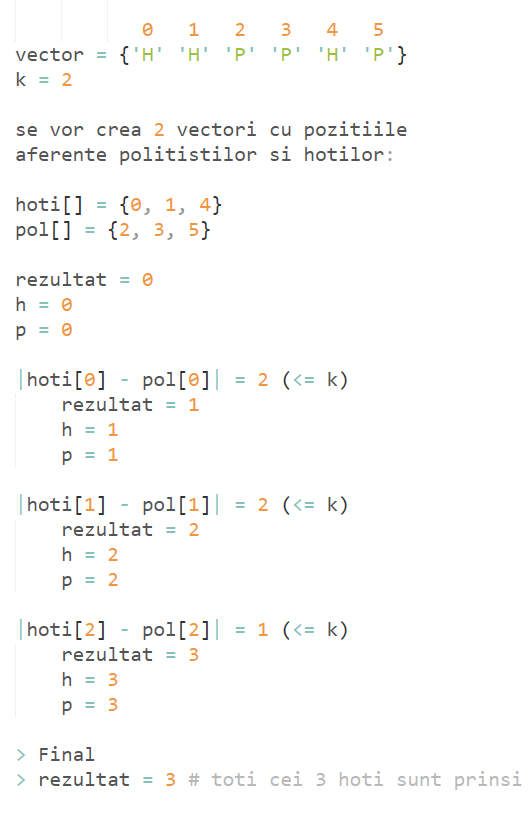
* Problema data **poate fi rezolvata** folosind un algoritm greedy astfel: se gaseste pozitia cea mai mica a unui politist si a unui hot, daca diferenta dintre acestea este mai mica decat k, s-a gasit o solutie; daca nu, se incrementeaza minimul dintre pozitii si se revine, se vor repeta acesti pasi pana cand toate solutiile vor fi gasite.
* **Pseudocod**:



* **Complexitate**: Θ(n) (timp liniar, deoarece, in cel mai rau caz, incare niciun hot nu poate fi prins, se va trece prin fiecare element al vectorilor de pozitii)
* Alegerea din cadrul algoritmului este de tip greedy deoarece solutia este construita iterativ si, de asemenea, nu se revine asupra deciziilor luate.Proprietatea de substructura optimala este indeplinita de algoritm, deoarece solutiile sub-problemelor sunt optime.

Asadar, un optim global poate fi obtinut de algoritmul prezentat.

* **Exemplificare**:



1. Tehnica Programarii dinamice

* **Enuntul Problemei** (mina de aur): Se da o mina de dimensiune n \* m. Fiecare spatiu din mina contine un anumit numar de grame de aur (valorile vor fi numere intregei, pozitive). Minerul incepe de pe prima coloana, el se poate deplasa doar spre iesirea din mina, care este spre dreapta (are 3 posibilitati de miscare: dreapta, dreapta-sus si dreapta-jos; "->", "/", "\"). Sa se gaseasca cantitatea maxima de aur ce poate fi extrasa de catre miner.

(sursa: <https://www.geeksforgeeks.org/gold-mine-problem/>)

* Problema data **poate fi rezolvata** folosing tehnica de dynamic programming. Se observa faptul ca valorile sunt doar pozitive, asadar trebuie efectuate toate mutarile posibile pentru a maximiza rezultatul si, de asemenea, faptul ca se va ajunge intotdeauna pe ultima coloana a minei, de unde orice miscare este imposibila. Se va crea o matrice de aceeasi dimensiune cu cea a minei. Incepand cu ultima coloana a minei, se va determina maximul in fiecare pozitie, tinand cont de toate posibilitatile de deplasare (pe ultima coloana nu mai este nicio deplasare posibila, pe randul de sus, deplsarea in dreapta-sus este imposibila, iar pe randul de jos, deplasarea in dreapta-jos este imposibila). Maximul global se va gasi pe prima coloana a matricei create.
* **Pseudocod**:

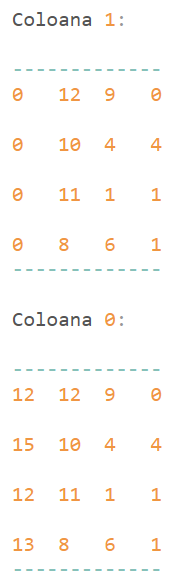


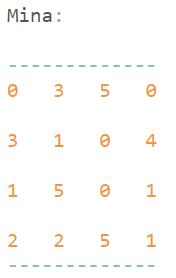
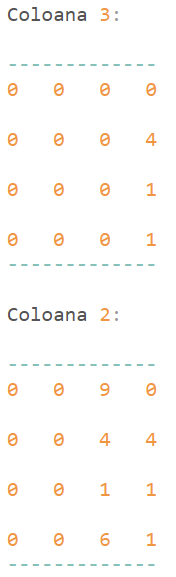
* **Complexitate:**
* Temporala:Θ (m \* n) (Se va verifica fiecare element al matricei Mina o singura data) pentru a genera matricea D a rezultatelor.
* Memorie: Θ (m \* n) (Se va crea o singura matrice cu dimensiunea egala cu cea a matricei data ca input)
* **Relatia de recurenta:**

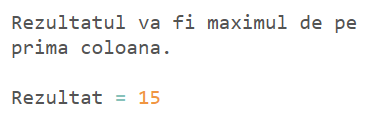
D[linie][coloana] = Mina[linie][coloana] + maxim(dreapta, dreapta-sus, dreapta-jos)

Aceasta reprezinta valoarea maxima de aur ce poate fi extrasa de pe pozitia [linie][coloana]. Se ajunge la aceasta valoare adunand la valoarea din matricea Mina, data ca input, maximul dintre toate posibilitatile de deplasare din pozitia [linie][coloana].

* **Exemplificare:**





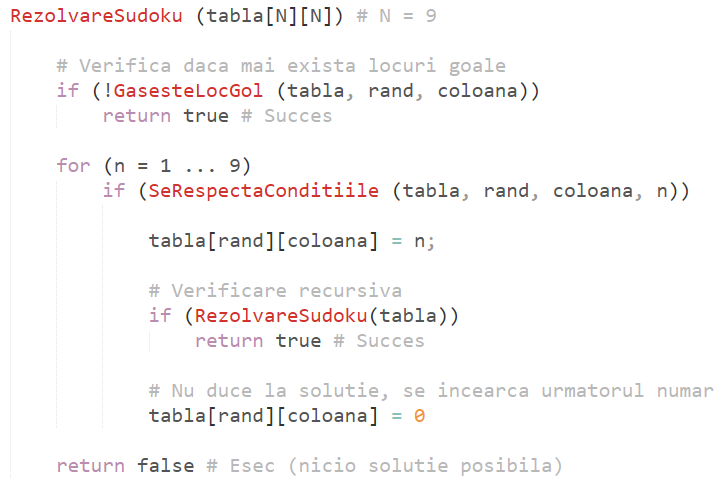


1. Tehnica Backtracking

* **Enuntul Problemei** (sudoku): Se da o tabla de 9 x 9 (o matrice 9 x 9), aceasta trebuie completata cu numere de la 1 la 9, astfel incat pe fiecare linie, coloana sau sub-diviziune de 3 x 3 (vor fi 9 astfel de sub-diviziuni) sa se regaseasca exact cate o instanta a fiecarei cifre.

(sursa: <https://leetcode.com/problems/sudoku-solver/>)

* Problema sudoku **poate fi rezolvata** folosind tehnica de backtracking. Completam locurile goale unul cate unul, verificand de fiecare data respectarea conditiilor. Dupa completarea unui loc cu un numar, se verifica recursiv daca aceasta completare a dus la gasirea unei solutii, daca nu, atunci se incearca urmatorul numar, iar daca niciun numar (de la 1 la 9) nu duce la solutie, atunci nicio solutie nu este posibila si returnam false.
* **Pseudocod**:



* **Complexitate**

Pentru fiecare loc gol sunt 9 optiuni posibile, asadar complexitatea va fi **Θ(9 n \* n )**, aceeasi cu a algoritmului naiv (in care se incearca toate posibilitatile), dar, in general, rezolvarea folosind metoda backtracking va fi mai rapida.

* Pentru optimizarea algoritmului folosit in rezolvarea problemei tablei de sudoku, se poate tine cont de numerele care sunt deja pe tabla, pentru nu a mai trece prin toate posibilitatile pentru fiecare loc gol gasit.

(Implementare optimizata: <http://byteauthor.com/2010/08/sudoku-solver/>)

* Din cauza marimii unei table de sudoku am considerat ca o exemplificare pas cu pas ar fi mult prea lunga. (vizualizare a rezolvarii unei table de sudoku prin metoda backtracking: <https://en.wikipedia.org/wiki/Sudoku_solving_algorithms#/media/File:Sudoku_solved_by_bactracking.gif>)