Universitatea Transilvania din Brașov
FACULTATEA DE MATEMATICĂ ȘI INFORMATICĂ

UNIVERSITATEA TRANSILVANIA DIN BRASOV
FACULTATEA DE MATEMATICA SI INFORMATICA

PROGRAMUL DE STUDII - INFORMATICA

Universitatea Transilvania din Brașov

ALGORITMI FUNDAMENTALI

Determinarea accelerației maxime a unui autovehicul utilizând tehnica Divide et Impera

-proiect-

STUDENT:
Opriș Andrei Ioan

BRAȘOV

2020

CUPRINS

1 Enunțul problemei

Unui inginer i se dau mai multe seturi de date constând in valori ale vitezei unu autovehicul in urma unor teste.

Inginerul trebuie sa sorteze crescător valorile vitezelor pentru a determina accelerația autovehiculului într-un interval de 2 valori succesive ale vitezelor măsurate, după care sa determine accelerația maxima.

Accelerația se va determina utilizând următoarea formula:

- accel = (vitezaActuală^2 - vitezaPrecedentă^2)/(2*Spațiu)

2 Explicație rezolvare

2.1 Contextul problemei

Având tabelul de mai jos realizat în urma unui test de măsurare a vitezelor unui autovehicul, unui inginer ii este data sarcina de a sorta crescător vitezele măsurate folosind un program bazat pe tehnica Divide et Impera.

tabelul 2.1

Nr. măsurătoare	Viteza [m/s]			
1	5	7	15	18
2	1	8	9	17
3	1	4	7	7

După sortarea crescătoare a vitezelor, acesta trebuie sa determine accelerația autovehiculului pentru 2 viteze măsurate succesive – utilizând formula:

$$a = \frac{v_{actual}^2 + v_{precedent}^2}{2 \cdot Spatiul} \qquad (2.1)$$

După care sa afle accelerația maxima a autovehiculului – utilizând un program de tip Divide et Impera.

2.2 Utilizarea unui algoritm pentru a transforma un set de date (vectori) într-un sigur set(vector)

Pentru tabelul 2.1 trebuie sa se realizeze un program ca pentru k seturi de măsurători fiecare de mărime n, acestea sa fie unite într-un singur set de date (vector).

- **Input:** viteze[ ][NrVitezePerSet] = {{5, 7, 15, 18}, {1, 8, 9, 17}, {1, 4, 7, 7}}
- **Output:** viteze[ ] = {5, 7, 15, 18, 1, 8, 9, 17, 1, 4, 7, 7}

```
int numarVitezePerSet = NrVitezePerSet;
	vector<float> vectorVitezeSortate;


	// Determinarea numarului de seturi de date pentru viteze
	int numarSeturiViteze = sizeof(vectorViteze) /
sizeof(vectorViteze[0]);

void MergeDate(vector<float>& vectorVitezeSortate, float
vectorViteze[][NrVitezePerSet], int numarVitezePerSet, int numarSeturiViteze)
{
	for (int indexLinii = 0; indexLinii < numarSeturiViteze; ++indexLinii)
	{
		for (int indexColoane = 0; indexColoane < numarVitezePerSet;
++indexColoane)
		{

	vectorVitezeSortate.push_back(vectorViteze[indexLinii][indexColoane]);
		}
	}
}
```

2.2.1 Analiza Complexității

- Timp de execuție: O(N * k * log(N * k)
  - Numărul total de viteze este dat de numarVitezePerSet * numarSeturiViteze de unde rezulta timpul de executie: O(NrVitezePerSet * numarSeturiViteze * log(NrVitezePerSet * numarSeturiViteze))

- Complexitate: O(N * k).
  - Pentru a retine toate seturile de date într-un singur vector complexitatea algoritmului este: O(NrVitezePerSet * numarSeturiViteze)

2.3 Utilizarea algoritmului Quicksort pentru sortarea vitezelor

QuickSort-ul este un algoritm de tip Divide et Impera care preia un element drept pivot si partiționează vectorul dat un funcție de acel pivot.

Rolul partiționării este ca pentru un vector dat si un element ”x” din acel vector luat drept pivot, toate elementele din vector care sunt mai mici decât x sa fie puse înaintea lui x, iar elementele mai mari decât x sa fie puse după elementul x, într-un timp liniar.

- Pseudocodul algoritmului recursiv QuickSort

```
procedure Partiționare   {Partiționează tabloul a[s..d]}
*selectează elementul x (de regulă de la mijlocul
                         intervalului de partiționat)
  repetă
    *caută primul element a[i]>x, parcurgând
       intervalul de la stânga la dreapta
    *caută primul element a[j]<x, parcurgând
       intervalul de la dreapta la stânga
    dacă i<=j  atunci
      *interschimbă pe a[i] cu a[j]

  până când  parcurgerile se întâlnesc (i>j)

procedure QuickSort(s,d);
  *partiționează intervalul s,d față de Mijloc
  dacă există partiție stânga atunci
    QuickSort(s,Mijloc-1)
  dacă există partiție dreapta atunci
    QuickSort(Mijloc+1,d);
```

Fig. 1 – Pseudocodul algoritmului QuickSort – sursa – Lect. Dr. A. Băicoianu - Curs Algoritmi Fundamentali Alte sort ˇ ari: Sortarea rapida (QuickSort)¸ si Bucket Sort Universitatea Transilvania din Brașov Facultatea de Matematica ¸ si Informatica 2020

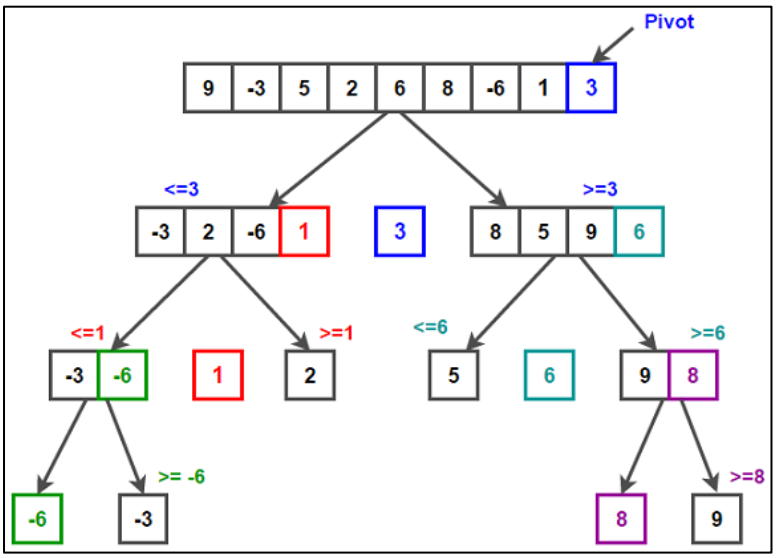

Fig. 2 – Schema logica de funcționare a algoritmului QuickSort – sursa - https://www.techiedelight.com/quicksort/

2.3.1 Analiza complexității algoritmului QuickSort

Timpul de execuție este dat de următoarea relație:

$$T(n) = T(k) + T(n-k-1) + \theta(n) \qquad (2.2)$$

,unde:

- $T(k),\ T(n-k-1)$ – sunt terminii pentru apelarea recursive
- $\theta(n)$ – termenul pentru partiționare (executat in timp liniar)

În funcție de partiționare pot exista trei cazuri:

- **Cazul cel mai defavorabil** – unde partiționarea alege ca si element de pivot cel mai mare sau cel mai mic element din vector. Considerând Fig. 2 drept strategie in care ultimul element este ales drept pivot, cel mai defavorabil caz ar fi cel in care vectorul este deja sortat crescător sau descrescător, astfel complexitatea algoritmului este:

$$\begin{gathered} T(n) = T(0) + T(n-1) + \theta(n) \\ T(n) = T(n-1) + \theta(n) \\ T(n) = \theta(n^2) \end{gathered} \qquad (2.3)$$

- **Cazul cel mai favorabil** – unde partiționarea alege ca si element de pivot elementul din mijloc al vectorului, iar astfel complexitatea algoritmului este:

$$T(n) = 2T\left(\frac{n}{2}\right) + \theta(n) \qquad (2.4)$$

Folosind cazul 2 din Teorema Master, complexitatea este de forma:

$$T(n) = \theta(n \cdot \log n) \qquad (2.5)$$

- **Cazul mediu** – unde trebuie luat in considerare toate permutările posibile dintr-un vector si calculat timpul necesar pentru fiecare permutare. Considerând o procedura de partiționare in care 90% din elemente sunt puse într-o parte, iar restul de 10% in cealaltă parte, avem următoarea complexitate

$$\begin{gathered} T(n) = T\left(\frac{9n}{10}\right) + T\left(\frac{n}{10}\right) + \theta(n) \\ T(n) = \theta(n \cdot \log n) \end{gathered} \qquad (2.6)$$

2.4 Determinarea accelerației

După ce vitezele au fost sortate crescător, rezultatul este un vector de forma:

- 1 4 5 7 7 7 8 9 15 17 18

Utilizând vectorul de mai sus, se aplica formula 2.1 pentru a determina accelerația vehiculului pentru 2 viteze succesive. De unde va rezulta un nou vector de accelerații având următoarea forma:

- Accelerațiile calculate sunt [m/s^2] :
- 0.75 0.45 1.2 0 0 0.75 0.85 7.2 3.2 1.75

2.5 Determinarea accelerației maxime

După ce au fost determinate accelerațiile pentru 2 viteze succesive datele de intrare pentru determinarea accelerației maxime utilizând un program de tip Divide et Impera sunt:

- **Input:** { 0.75, 0.45, 1.2, 0, 0, 0.75, 0.85, 7.2, 3.2, 1.75 } m/s^2
- **Output**: 7.2 m/s^2

```
float Maxim(vector<float> vector, int index, int l)
{
	float max;
	if (index >= l - 2)
	{
		if (vector[index] > vector [index + 1])
			return vector[index];
		else
			return vector [index + 1];
	}
	max = Maxim (vector, index + 1, l);
	if (vector[index] > max)
		return vector[index];
	else
		return max;
}
```

Pentru a determina maximul vom folosi o funcție recursiva in care vom verifica daca maximul se afla in partea stânga a vectorului – prin următoarea condiție: if (vector[index] > vector[index+1].) – după care se returnează valoarea.

După ce s-a terminat de verificat daca maximul se afla in partea stânga a vectorului urmează verificarea in partea dreapta a vectorului. Acest lucru se face prin apelarea recursiva a funcției pentru indexul curent al vectorului: max = Maxim (vector, index + 1, l). După care se va introduce o condiție de verificare daca elementul din partea dreapta a vectorului este cel maxim – if (vector[index] > max) – iar la sfârșit se va returna valoarea maxima.

3 Execuție

3.1 Programul

```
* D_I_viteze.cpp : Unui inginer i se dau mai multe seturi de date constând in valori ale
vitezei unu autovehicul in urma unor teste.
Inginerul trebuie sa sorteze crescător valorile vitezelor pentru a determina accelerația
autovehiculului într-un interval de 2 valori succesive ale vitezelor măsurate, după care
sa determine accelerația maxima.
Accelerația se va determina utilizând următoarea formula:
    • accel = (vitezaActuală^2 - vitezaPrecedentă^2)/(2*Spațiu)
*/

#include <iostream>
#include <math.h>
#include <cmath>
#include <bits.h>
#include <vector>
#include <iomanip>
using namespace std;
#define NrVitezePerSet 4
#define Spatiu 10


void MergeDate(vector<float>& vectorVitezeSortate, float vectorViteze[][NrVitezePerSet],
int numarVitezePerSet, int numarSeturiViteze)
{
	for (int indexLinii = 0; indexLinii < numarSeturiViteze; ++indexLinii)
	{
		for (int indexColoane = 0; indexColoane < numarVitezePerSet;
++indexColoane)
		{

	vectorVitezeSortate.push_back(vectorViteze[indexLinii][indexColoane]);
		}
	}
}


void Swap(float* a, float* b)
{
	float t = *a;
	*a = *b;
	*b = t;
}


int Partitionare(vector<float>& vector, int low, int high)
{
	int pivot = vector[high];
	int indexElementMic = (low - 1);

	for (int indexPart = low; indexPart <= high - 1; ++indexPart)
	{
		if (vector[indexPart] < pivot)
		{
			indexElementMic++;
			Swap(&vector[indexElementMic], &vector[indexPart]);
```

```
			}
		}
		Swap(&vector[indexElementMic + 1], &vector[high]);
		return (indexElementMic + 1);
}


void QuickSort(vector<float>& vector, int low, int high)
{
		if (low < high)
		{
			int indexPartitionare = Partitionare(vector, low, high);
			QuickSort(vector, low, indexPartitionare - 1);
			QuickSort(vector, indexPartitionare + 1, high);
		}
}


void AfisareVector(vector<float> vector)
{
		for (int indexViteze = 0; indexViteze < vector.size(); ++indexViteze)
		{
			cout << vector[indexViteze] << " ";
		}
		cout << "\n";
}


void CalculVectorAcceleratie(vector<float> vectorVitezeSortate, vector<float>&
vectorAcceleratii)
{
		cout << "Acceleratiile calculate sunt [m/s^2] : " << "\n";
		for (int indexViteze = 0; indexViteze < vectorAcceleratii.size() - 1;
++indexViteze)
		{
			vectorAcceleratii[indexViteze] = ((vectorVitezeSortate[indexViteze + 1] *
vectorVitezeSortate[indexViteze + 1]) - (vectorVitezeSortate[indexViteze] *
vectorVitezeSortate[indexViteze])) / (2 * Spatiu);
			cout << vectorAcceleratii[indexViteze] << " ";
		}
		cout << "\n";
}


float Maxim(vector<float> vector, int index, int l)
{
		float max;
		if (index >= l - 2)
		{
			if (vector[index] > vector[index + 1])
				return vector[index];
			else
				return vector[index + 1];
		}
		max = Maxim(vector, index + 1, l);

		if (vector[index] > max)
			return vector[index];
```

```
	else

		return max;
}


int main()
{
	// Datele de intrare
	float vectorViteze[][NrVitezePerSet] = { { 5, 7, 15, 18 },
	                                         { 1, 8, 9, 17 },
	                                        { 1, 4, 7, 7 } };
	int numarVitezePerSet = NrVitezePerSet;
	vector<float> vectorVitezeSortate;


	// Determinarea numarului de seturi de date pentru viteze
	int numarSeturiViteze = sizeof(vectorViteze) / sizeof(vectorViteze[0]);

	// procesare date viteza autovehicul
	MergeDate(vectorVitezeSortate, vectorViteze, numarVitezePerSet,
numarSeturiViteze);
	QuickSort(vectorVitezeSortate, 0, vectorVitezeSortate.size() - 1);
	cout << "Vitezele sortate crescator sunt [m/s] : " << "\n";
	AfisareVector(vectorVitezeSortate);

	// procesare date acceleratie autovehicul
	vector<float> vectorAcceleratii(vectorVitezeSortate.size());
	CalculVectorAcceleratie(vectorVitezeSortate, vectorAcceleratii);
	cout << "Acceleratia maxima este : "  << Maxim(vectorAcceleratii, 0,
vectorAcceleratii.size()) << " m/s^2" << "\n";

	return 0;
}
```