TP1 partie1-1

trouver les problèmes et les modifier.(plus de 5)

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

#define INFINI 9999

/\*

Retourne la valeur de a si elle est plus petite que la valeur de b.

Retourne la valeur de b sinon.

\*/

int min(int number1, int number2)

{

if(number1 >= number2)

{

return number2;

}

else

{

return number1;

}

}

int max(int number1, int number2)

{

if(number1 >= number2)

{

return number1;

}

else

{

return number2;

}

}

1.Ici,pour les deux fonctions,ce n’est pas bien utilisant a et b pour comparer les deux paramètres.

C’est mieux utilisant ‘number1’,’number2’ remplaçant ‘a’et’b’.

/\*

LibËre l'espace mÈmoire utilisÈ par un tableau 2D dynamique allouÈ sur le tas.

EntrÈe:

\* Le tableau 2d dynamique d'entiers (au moins une ligne et une colonne).

\* Le nombre de lignes du tableau.

\*/

void freeTab2D(int \*\*tab, int tailleX)

{

int ligne;

for( ligne= 0; ligne < tailleX; ligne++)

{

free(tab[ligne]);

}

free(tab);

}

2.pour cette fonction, on ne sait pas ‘i’c’est quoi. parce que il y a deux coordonnées d’un tableau.

alors, selon l’intention de cette fonction, il faut utilisant’ligne’ remplaçant ‘i’

/\*

Affiche les valeurs du tableau passÈ en paramËtre, avec un espace entre chaque valeur.

EntrÈe:

\* Le tableau d'entiers.

\* La taille du tableau (>= 0).

\*/

void afficherTabEntier(int \*tab, int tailleTab)

{

int Position;

for(Position = 0; Position < tailleTab; Position++)

{

printf("%d ", tab[Position]);

}

}

3.pour cette fonction, on ne sait pas ‘i’c’est quoi.Selon l’intention de cette fonction, il faut utilisant’Position’ remplaçant ‘i’

/\*

Affiche les valeurs du tableau 2D passÈ en paramËtre sous forme de matrice.

EntrÈe:

\* Le tableau d'entiers 2D.

\* La taille du tableau en X (>= 0).

\* La taille du tableau en Y (>= 0).

\* Le nombre de chiffres maximum que peuvent contenir les valeurs du tableau.

--> Si les valeurs peuvent Ítre nÈgatives, ajouter 1 ‡ ce paramËtre.

--> Valeurs autorisÈes pour ce paramËtre: 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

--> Ce paramËtre sera utilisÈ pour formater correctement l'affichage du tableau en matrice.

\*/

void afficherTab2DEntier(int \*\*tab, int tailleX, int tailleY, int nbChiffreMax)

{

int x, y;

/\* char format[4] = {'%', (char) (nbChiffreMax + '0'), 'd', ' '}; \*/

for(x = 0; x < tailleX; x++)

{

for(y = 0; y <= tailleY; y++)

{

printf("%4d ", tab[x][y]);

}

printf("\n");

}

}

4. pour cette fonction, il faut utilisant’x’ remplaçant ‘i’,’y’ remplaçant ‘j’. C’est justement les coordonnées d’un tableau. C’est plus clair.

/\*

Calcul de la matrice des diffÈrences pour l'algorithme MVM.

EntrÈe:

\* Tableau d'entiers reprÈsentant la sÈquence A (le "modËle" ‡ retrouver dans la sÈquence B).

\* Tableau d'entiers reprÈsentant la sÈquence B (la sÈquence source dans laquelle on procÈdera ‡ l'analyse).

\* Taille de la sÈquence A (nombre de cases dans la tableau).

\* Taille de la sÈquence B (nombre de cases dans la tableau).

Sortie:

\* Tableau 2D reprÈsentant la matrice des diffÈrences, allouÈ sur le tas.

\*/

int \*\*matriceDiff(int \*sequenceA, int \*sequenceB, int tailleSeqA, int tailleSeqB)

{

int \*\*matrice = (int \*\*) malloc(sizeof(int\*) \* tailleSeqA);

int x, y;

for(x = 0; x < tailleSeqA; x++)

{

matrice[x] = (int \*) malloc(sizeof(int) \* tailleSeqB);

for(y = 0; y < tailleSeqB; y++)

{

matrice[x][y] = sequenceB[y] - sequenceA[x];

}

}

return matrice;

}

5.pour cette fonction, C’est la même chose qu’avant. il faut utilisant’x’ remplaçant ‘i’,’y’ remplaçant ‘j’. C’est justement les coordonnées d’un tableau. C’est plus clair.

/\*

Affiche les sÈquences B et A l'une en dessous de l'autre en mettant en Èvidence les ÈlÈments liÈs entre les 2 (dÈterminÈs par l'algorithme MVM).

EntrÈe:

\* Tableau d'entiers reprÈsentant la sÈquence A (le "modËle" ‡ retrouver dans la sÈquence B).

\* Tableau d'entiers reprÈsentant la sÈquence B (la sÈquence source dans laquelle on procÈdera ‡ l'analyse).

\* Taille de la sÈquence A (nombre de cases dans la tableau).

\* Taille de la sÈquence B (nombre de cases dans la tableau).

\* Le tableau de correspondance sequenceA <=> sÈquenceB de la taille de sÈquenceA (appelÈ indiceBpourBprime).

--> Correspondance seqA <=> seqB, pour chaque case i de ce tableau: sequenceA[i] <=> sequenceB[indiceBpourBprime[i]].

\*/

void afficherAlignementSeqAB(int \*sequenceA, int \*sequenceB, int tailleSeqA, int tailleSeqB, int \*indiceBpourBprime)

{

int i;

int j = 0;

printf("Sequence B: ");

/\* affichage de toute la sÈquence B \*/

for(i = 0; i < tailleSeqB; i++)

{

printf("%4d", sequenceB[i]);

}

printf("\nSequence A: ");

/\* affichage de toute la sÈquence A, morceau par morceau, de faÁon ‡ aligner chaque ÈlÈment de la sÈquence A sur son ÈlÈment correspondant dans la sÈquence B \*/

for(i = 0; i < tailleSeqB; i++)

{

if(j < tailleSeqA)

{

if(i == indiceBpourBprime[j]) /\* alors l'ÈlÈment courant de la sÈquence B a une correspondance dans la sÈquence A \*/

{

printf("%4d", sequenceA[j]);

j++;

}

else

{

printf("%4c", (char) ' ');

}

}

}

printf("\n");

}

6.dans cette fonction, le paramètre ‘indiceBpourBprime’est un peu unclair. et pour le nommage d’un paramètre, il faut écrire comme ça: indicePourBPrime. (pour tous les mots,il faut commencer avec des caractères majuscules) je pense que ça signifique ‘indice à la place de séquenceB’. Alors, peut-être on peut utiliser ‘indiceToB’.C’est plus simple je trouve.

/\*

Applique l'algorithme MVM.

Affiche la matrice des diffÈrences, path et pathcost.

Et enfin affiche la meilleure sous-sÈquence B' de B, la plus proche de la sÈquence A.

EntrÈe:

\* La sÈquence A dans un tableau d'entiers.

\* La sÈquence B dans un tableau d'entiers.

\* La taille de la sÈquence A.

\* La taille de la sÈquence B.

Postconditions:

La taille de la sÈquence A doit Ítre infÈrieure ou Ègale ‡ celle de la sÈquence B.

\*/

void MVM(int \*sequenceA, int \*sequenceB, int tailleSeqA, int tailleSeqB)

7.je pense que le nommage n’est pas bien utilisant trois caractères Majuscules .

{

int \*\*r = matriceDiff(sequenceA, sequenceB, tailleSeqA, tailleSeqB);

8.ici, il faut donner un nom exact pour ‘r’. par exemple: ‘mdiff’ ou ‘diff’.

int elasticity = max(tailleSeqB - tailleSeqA,1+round(tailleSeqA/10)); /\*round n'est pas dans le std C89\*/

printf("elasticity=%d\n",elasticity); /\* = nombre díÈlÈments de B que l'on peut sauter pour dÈterminer Bí \*/

int pathcost[tailleSeqA][tailleSeqB]; /\* co˚t de la sÈquence trouvÈe \*/

int path[tailleSeqA][tailleSeqB]; /\* chemin de la meilleure sous-sÈquence \*/

int i, j, k;

9.ici,c’est la même chose qu’avant, on utilise ‘a’,’b’,’c’ à remplacer ‘i’’j’’k’,c’est mieux.

int stopk, stopj, debk;/\* debj;\*/

int indiceBpourBprime[tailleSeqA]; /\* contient les indices de la sÈquence B retenu pour la meilleure sous-sÈquence B' \*/

int meilleureSousSeq[tailleSeqA]; /\* meilleure sous-sÈquence B' de B la plus proche de la sÈquence A \*/

/\* initialisation des tableaux de sortie path et pathcost \*/

for(i = 1; i <= tailleSeqA; i++)

{

for(j = 1; j <= tailleSeqB; j++)

{

pathcost[i - 1][j - 1] = INFINI;

path[i - 1][j - 1] = 0;

}

}

/\* cout du premier ÈlÈment de la sÈquence \*/

for(j = 1; j <= elasticity + 1; j++)

{

pathcost[1 - 1][j - 1] = (r[1 - 1][j -1]) \* (r[1 - 1][j -1]);

}

for(i = 2; i <= tailleSeqA; i++)

stopk = min(i - 1 + elasticity, tailleSeqB);

/\*for(k = i-1; k <= stopk; k++) \*/

debk=max(1,i-2);/\*elasticity+1);\*/

for(k = debk; k <= stopk; k++)

{

stopj = min(k + 1 + elasticity, tailleSeqB);

/\*debj=max(1,k-elasticity);

for(j = debj ; j <= stopj; j++)\*/

for(j = k ; j <= stopj; j++)

/\*for(j = k + 1; j <= stopj; j++)\*/

{

if( pathcost[i -1][j - 1] > pathcost[i - 1 - 1][k - 1] + ((r[i - 1][j - 1]) \* (r[i - 1][j - 1])) )

{

pathcost[i -1][j - 1] = pathcost[i - 1 - 1][k - 1] + ((r[i - 1][j - 1]) \* (r[i - 1][j - 1]));

path[i -1][j - 1] = k;

}

}

}

}

/\* On dÈtermine la meilleure sous sÈquence B' de B ‡ partir des tableaux path et pathcost (qui sont maintenant peuplÈs) : \*/

/\* recherche de l'indice de la case qui contient la plus petite valeur dans la derniËre ligne de pathcost \*/

indiceBpourBprime[tailleSeqA - 1] = 0;/\*l'indice est dans le repère tableau C (de 0 à tailleSeq)\*/

for(i = 1; i < tailleSeqB; i++)

{

/\*printf("%d : pt[i]=%d ; pt[i-1]=%d; \n",i,pathcost[tailleSeqA - 1][i],pathcost[tailleSeqA - 1][i-1]);\*/

/\*if(pathcost[tailleSeqA - 1][i] < pathcost[tailleSeqA - 1][i - 1])\*/

if(pathcost[tailleSeqA - 1][i] <= pathcost[tailleSeqA - 1][indiceBpourBprime[tailleSeqA - 1]])

{

indiceBpourBprime[tailleSeqA - 1] = i;

}

}

/\*printf("meilleureSousSeq[%d]=%d\n", tailleSeqA - 1, sequenceB[indiceBpourBprime[tailleSeqA - 1]]);\*/

meilleureSousSeq[tailleSeqA - 1] = sequenceB[indiceBpourBprime[tailleSeqA - 1]];

/\* on remplit meilleureSousSeq de l'avant derniËre case ‡ la premiËre case grace ‡ path qui nous donne la correspondance B <=> B' \*/

for(i = 1; i < tailleSeqA; i++)

{

indiceBpourBprime[tailleSeqA - i - 1] = path[tailleSeqA - i][indiceBpourBprime[tailleSeqA - i]] - 1;

meilleureSousSeq[tailleSeqA - i - 1] = sequenceB[indiceBpourBprime[tailleSeqA - i - 1]];

/\*printf("meilleureSousSeq[%d]=%d\n", tailleSeqA -i- 1, sequenceB[indiceBpourBprime[tailleSeqA -i - 1]]);\*/

}

/\* Affichage de r \*/

printf("\nMatrice des differences :\n");

afficherTab2DEntier(r, tailleSeqA, tailleSeqB, 2);

/\* Affichage de pathcost \*/

printf("\nResultat pathcost :\n");

for(i = 0; i < tailleSeqA; i++)

{

for(j = 0; j < tailleSeqB; j++)

{

printf("%4d ", pathcost[i][j]);

}

printf("\n");

}

/\* Affichage de path \*/

printf("\nResultat path :\n");

for(i = 0; i < tailleSeqA; i++)

{

for(j = 0; j < tailleSeqB; j++)

{

printf("%4d ", path[i][j]);

}

printf("\n");

}

printf("\nMeilleure sous-sequence de B: ");

afficherTabEntier(meilleureSousSeq, tailleSeqA);

/\*printf("\n\nAlignement visuel :\n");\*/

/\*afficherAlignementSeqAB(sequenceA, sequenceB, tailleSeqA, tailleSeqB, indiceBpourBprime);\*/

freeTab2D(r, tailleSeqA); /\* libÈration de l'espace mÈmoire occupÈ par la matrice des diffÈrences \*/

}

10.dans cette fonction, il faut changer ‘i’ à ‘a’; ‘j’à’b’. pour correspondre à séquenceA et séquenceB.

/\*

Demande ‡ l'utilisateur de saisir une sÈquence d'entiers et retourne la sÈquence dans un tableau d'entiers.

EntrÈe:

\* Pointeur vers un entier qui sera utilisÈ comme variable de sortie.

Sortie:

\* Le tableau d'entiers naturels contenant la sÈquence saisie par l'utilisateur, allouÈ sur le tas.

\* La taille de la sÈquence saisie par l'utilisateur et contenue dans le tableau de sortie.

\*/

int \*saisieSequence(int \*tailleSequence)

{

int c; /\* Contiendra le dernier ÈlÈment de la sÈquence tapÈ par l'utilisateur \*/

int i = 0;

int \*sequence = (int \*) malloc(sizeof(int)\*10);

printf("Veuiller entrer la sequence.\n");

while(c != -1) /\* boucle jusqu'‡ ce que l'utilisateur tape -1 \*/

{

printf("Element %d ?: ", i+1);

scanf("%d",&c);

printf("\n");

if(c != -1)

{

if(c >= 0)

{

sequence[i]=c;

i++;

if(i%10 == 0) /\* ajout de 10 cases supplÈmentaires au tableau de sortie si la limite est atteinte \*/

{

sequence = realloc(sequence, sizeof(int)\*(i+10));

}

}

else

{

printf("Erreur. Vous devez saisir un entier positif.\n");

}

printf("Sequence saisie: ");

afficherTabEntier(sequence, i);

printf("\nTaper -1 si la saisie de votre sequence est terminee.\n");

}

}

\*tailleSequence = i;

return sequence;

11.dans cette fonction, paramètre C est inexact.il faut changer à’int end’.

/\*

Demande ‡ l'utilisateur la saisie de deux sÈquences A et B et faÁon ‡ ce que ces sÈquences soient exploitables par l'algorithme MVM.

(==> sÈquences d'entiers positifs et taille de la sÈquence B >= taille de la sÈquences A).

O˘ la sÈquence B est la sÈquence de dÈpart.

Et o˘ la sÈquence A est le "modËle" ‡ retrouver dans la sÈquence B.

EntrÈe:

\* Pointeurs vers des variables qui seront utilisÈes comme variables de sortie.

Sortie:

\* (Pointeur sur) la sÈquence A saisie par l'utilisateur (tableau allouÈ sur le tas).

\* (Pointeur sur) la sÈquence B saisie par l'utilisateur (tableau allouÈ sur le tas).

\* La taille de la sÈquence A.

\* La taille de la sÈquence B.

\*/

void saisieSequenceABmvm(int \*\*sequenceA, int \*\*sequenceB, int \*tailleSeqA, int \*tailleSeqB)

{

printf("SEQUENCE SOURCE DE DEPART B :\n");

\*sequenceB = saisieSequence(tailleSeqB);

printf("\nVotre sequence source B vaut: ");

afficherTabEntier(\*sequenceB, \*tailleSeqB);

do /\* On demande la saisie de la sequence A ‡ l'utilisateur tant que celle-ci est plus grande que la sÈquence B \*/

{

printf("\n\nSEQUENCE A (A PARTIR DE LAQUELLE LA SOUS-SEQUENCE DE B LA PLUS PROCHE SERA DETERMINE) :\n");

\*sequenceA = saisieSequence(tailleSeqA);

if((\*tailleSeqA) > (\*tailleSeqB))

{

printf("\n /!\\ VOTRE SEQUENCE A EST TROP GRANDE. CELLE-CI NE DOIT PAS DEPASSER LA TAILLE DE LA SEQUENCE B (%d ELEMENT(S)). VEUILLEZ RE-SAISIR LA SEQUENCE A.", \*tailleSeqB);

}

}

while((\*tailleSeqA) > (\*tailleSeqB));

printf("\nVotre sequence A vaut: ");

afficherTabEntier(\*sequenceA, \*tailleSeqA);

printf("\nVotre sequence source B vaut: ");

afficherTabEntier(\*sequenceB, \*tailleSeqB);

printf("\n");

}

int main()

{

int \*sequenceA ;

int tailleSeqA = 0;

int \*sequenceB = NULL;

int tailleSeqB = 0;

saisieSequenceABmvm(&sequenceA, &sequenceB, &tailleSeqA, &tailleSeqB);

MVM(sequenceA, sequenceB, tailleSeqA, tailleSeqB);

free(sequenceA);

free(sequenceB);

return 0;

}