IFC-Campus Videira
Thiago Trzcinski
Algorítimos

Relatório

O projeto foi construído possuindo em mente o objetivo de encontrar 4 caracteres coincidentes em sequência dentro de uma matriz de tamanho variável. Uma matriz kernel chamada dna, foi realizada para garantir que as verificações seriam realizadas dentro de uma matriz controlada e não no ambiente volátil da matriz completa, assim o código a seguir foi utilizada para percorrer a matriz principal chamada dna_C.

```
for(k=0; k < TAM_C-3; k++)
{
    for(i=0; i<TAM; i++){
        for(j=0; j<TAM; j++){
            dna[i][j] = dna_C[l+i][k+j];
        }
    }
```

Este código então gerava a seguinte matriz:

```
a       t       c       t       t
c       t       a       c       a
c       t       t       a       c
c       a       c       t       a
a       t       t       a       a

a       t       c       t
c       t       a       c
c       t       t       a
c       a       c       t

t       c       t       t
t       a       c       a
t       t       a       c
a       c       t       a

c       t       a       c
c       t       t       a
c       a       c       t
a       t       t       a

t       a       c       a
t       t       a       c
a       c       t       a
t       t       a       a
```

Com a matriz dna funcionando e garantindo que as verificações ocorreriam em uma matriz 4x4, era necessário realizar as verificações horizontais, verticais e em ambas as diagonais. Para a criação de uma verificação horizontal o código a seguir foi utilizado:

```
int verifica_horizontal(char dna[][TAM])//Linha horizotal
{
    int macaco = 0;
    int conce_letra= 0;
    int i=0;
    int j=0;
    for(i=0; i<TAM; i++)//Percore matriz
    {
        for(j=0; j<TAM; j++)
        {
            if(conce_letra+1 ==TAM)
            {
                break;
            }
            if(dna[i][j] == dna[i][j+1])//Verifica DNA
            {
                conce_letra += 1;
            }else
            {
                conce_letra= 0;
                break;
            }
        }
    }
    if(conce_letra+1 == TAM)
    {
        macaco = 1;
    }
    return(macaco);
}
```

Aqui é realizada a verificação das letras(dna[i][j] == dna[i][j+1]), confirmando se a posição atual é idêntica à próxima posição na horizontal. Para a criação de uma verificação na vertical o código a seguir foi utilizado:

```c
int verifica_vertical(char dna[][TAM])//Linha vertical
{
    int macaco = 0;
    int conce_letra= 0;
    int i=0;
    int j=0;
    for(i=0; i<TAM; i++)//Percore matriz
    {
        for(j=0; j<TAM; j++)
        {

            if(conce_letra+1 ==TAM)
            {
                break;
            }
            if(dna[j][i] == dna[j+1][i])//Verifica DNA
            {
                conce_letra += 1;

            }else
            {
                conce_letra= 0;
                break;
            }
        }
    }
    if(conce_letra+1 ==TAM)
        {
            macaco = 1;
        }
    return(macaco);
}
```

Aqui é realizada a verificação das letras (dna[j][i] == dna[j+1][i]), confirmando se a posição atual é idêntica à próxima posição na vertical. Um processo semelhante foi utilizado para verificar a diagonal principal.

```c
int princDiag(char dna[][TAM])//Diagonais
{
    int macaco = 0;
    int conce_letra= 0;
    int i=0;
    for (i=0; i<TAM ; i++)
    {
        if(conce_letra+1 ==TAM)
        {
            break;
        }
            if(dna[i][i] == dna[i+1][i+1])//Verifica DNA
            {
                conce_letra += 1;
            }else
            {
                conce_letra= 0;
                break;
            }
        }
         if(conce_letra+1 ==TAM)
    {
            macaco = 1;
    }

        return(macaco);
}
```

Aqui apenas um valor i é utilizado para ambas as posições, realizando assim uma verificação na vertical com o código (dna[i][i] == dna[i+1][i+1]). A realização da segunda diagonal possui um processo um pouco diferente.

```
int segunDiag(char dna[][TAM])//Diagonais-2
{
    int macaco = 0;
    int conce_letra= 0;
    int i=0;
    int j=0;
    for (i=TAM-1; i>=0 ; i--)//Percore matriz
    {
        if(conce_letra+1 ==TAM)
        {
            break;
        }
        if(dna[i][j] == dna[i-1][j+1])//Verifica DNA
        {
            conce_letra += 1;
        }else
        {
            conce_letra= 0;
            break;
        }

        j+=1;
        }

    if(conce_letra+1 ==TAM)
    {
            macaco = 1;
    }
        return(macaco);
}
```

Aqui o valor i começa em seu máximo, enquanto j começa em seu mínimo e então os valores se amínguam e incrementam respectivamente, realizando assim a verificação na segunda diagonal com o código (dna[i][j] == dna[i-1][j+1]).

Todas as verificações possuem um retorno em comum onde se o número de letras consecutivas é igual ao tamanho desejado, no caso quatro, o valor retornado sera um, caso contrario o retorno sera zero. Durante o processo da matriz kernel a cada atualização dos valores de dna às quatro verificações eram realizadas, caso uma delas retornasse um então o valor final seria um, resultando em um paciente símio e terminando o loop, caso contrario o valor final seria 0, resultando em um paciente humano.

```
//verificações
if(verifica_horizontal(dna) == 1 || verifica_vertical(dna) == 1 || princDiag(dna) == 1|| segunDiag(dna) == 1)
{
    macaco = 1;
    break;
}
else
{
    macaco = 0;
}
```

Então um loop era criado com o tamanho de pacientes analisados, onde o usuário é confrontado com a primeira opção onde escolhe entre adicionar outro paciente ou encerar o programa.

```c
printf("Adicionar paciente[Digite 1 para sim e 0 para nao]: ");
scanf("%i", &rodar);
if(rodar == 0)
{
    exit(0);
}
```

Apos isso o usuário escolhe o modo de preencher a matriz dna_C, possuindo um modo manual, onde cada letra deve ser digitada pelo usuário.

```c
void carregaMatrizCompleta(char dna_C[][TAM_C])//Declara matriz manual
{
    char l, k;
    char a = 'a';
    char t = 't';
    char c = 'c';
    char g = 'g';

    for(l=0; l<TAM_C; l++){
        for(k=0; k<TAM_C; k++){
            printf("Informe o valor para dna[%i][%i]: ", l, k);
            scanf("%s", &dna_C[l][k]);

            if(dna_C[l][k] != a && dna_C[l][k] != t && dna_C[l][k] != c && dna_C[l][k] != g) //Verifica DNA
            {
                printf("\n Valor informado invalido");
                exit(0);
            }
        }
    }
}
```

Esta opção também possuem uma verificação, confirmando assim que os valores enviados pelo usuário estão aptos a verificação. A outra opção seria um preenchimento com valores aleatórios.

```c
void carregaMatrizAleatoria(char dna_C[][TAM_C])//Declara matriz aleatorio
{
    char l, k;
    srand(time(NULL));
    char sequencia[4]="atcg";
    for(l=0; l<TAM_C; l++){
        for(k=0; k<TAM_C; k++){
            dna_C[l][k] = sequencia[rand() % (sizeof(sequencia) - 1)];
        }
    }
}
```

Nesta opção o programa escolhe entre às quatro letras possíveis e preenche toda a matriz. Apos isso a matriz dna_C é escrita para o usuário, podendo assim verificar os valores recebidos.

```c
void escreveMatriz(char dna_C[][TAM_C])//Escreve Matriz
{
    char l, k;
    for(l=0; l<TAM_C; l++){
        for(k=0; k<TAM_C; k++){
            printf("%c\t", dna_C[l][k]);
        }
        printf("\n");
    }
}
```

Apos isto o registro criado para o paciente é chamado, salvando o resultado de cada dna.

```c
typedef struct{ //Registro Paciente
    int resultado;
} Paciente;
```

Neste mesmo momento o programa retorna o resultado ao usuário.

```c
pacientes[i].resultado = percorreMatriz(dna_C, dna);
printf("\n");
if( percorreMatriz(dna_C, dna) == 1 )
{
    printf("Resultado paciente[%i]:  Simio", i);
}
else
{
    printf("Resultado paciente[%i]:  Humano", i);
}
printf("\n");
```

Assim o programa final ficou do seguinte modo:

```
Adicionar paciente[Digite 1 para sim e 0 para nao]: 1

Informe o modo de receber o dna[Digite 0 para manual e 1 para aleatorio]: 1

t       c       t       a       c
c       c       a       a       a
a       a       a       a       a
c       a       c       c       c
c       a       c       t       t

Resultado paciente[0]:  Simio
Adicionar paciente[Digite 1 para sim e 0 para nao]:
```