

Introdução (Algoritmos e Programação III) – Aula 1

Após a implementação de um array e uma lista encadeada, foi possível realizar a comparação de tempo, para a inclusão de 1 milhão de registros, entre as duas estruturas.

Array

Com a implementação realizada, baseada em um array, temos a certeza, mesmo sem realizar o teste, de que a inclusão neste tipo de estrutura é muito mais rápida em relação a uma lista encadeada.

A questão é que para este tipo de estrutura, o sistema deixa reservada a quantidade necessária de memória para todos os elementos em um bloco linear, ou seja, lado a lado, tornando assim o acesso muito mais rápido.

Listas encadeadas

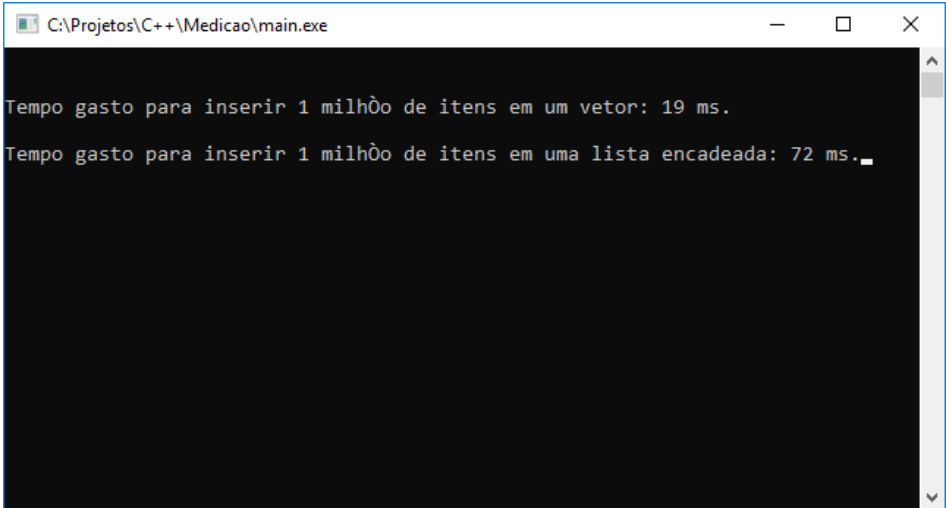
Já na estrutura de listas encadeadas, o sistema não precisa necessariamente, separar um bloco na memória para os itens que queremos incluir. As posições vão sendo ocupadas conforme a necessidade e os itens pode ficar espalhados na memória, ou seja, fragmentados melhor dizendo.

Com este tipo de estrutura existe uma menor utilização da memória, já que a lista vai crescendo conforme a necessidade, ou seja, não é necessário saber o tamanho da lista para então reservar seu espaço na memória.

Conclusão

Devemos analisar a real necessidade que temos diante a problemas que devemos resolver para então escolher tipo de estrutura que mais se adequa à nossa solução.

Abaixo temos uma imagem que exemplifica os testes realizados e o tempo que foi gasto para cada uma das estruturas descritas nesta atividade.



```
C:\Projetos\C++\Medicao\main.exe
Tempo gasto para inserir 1 milhão de itens em um vetor: 19 ms.
Tempo gasto para inserir 1 milhão de itens em uma lista encadeada: 72 ms.
```

Código em C++

```
#include<stdio.h>
#include<stdlib.h>
#include<time.h>

//Para o array
struct _nodo {
    int data;
    int index;
};

//Para a lista encadeada
typedef struct elementList {
    int data;
    struct elementList *next;
} element;

const int width = 1000000;

_nodo vetor[width];

void run_vetor();
void run_list();

int main()
{
    run_vetor();
    run_list();
}

void run_vetor()
{
    clock_t time_elapsed[2];
    time_elapsed[0] = clock();

    for(int i = 0; i < width; i++){
        vetor[i].data = rand() % width + 1;
        vetor[i].index = i;
    }
    time_elapsed[1] = clock();

    //show_items();

    double final_time = (time_elapsed[1] - time_elapsed[0]) * 1000.0 / CLOCKS_PER_SEC;

    printf("\n\nTempo gasto para inserir 1 milhão de itens em um vetor: %g ms.",
final_time);
}
```

```

void show_items()
{
    for(int i = 0; i < width; i++)
        printf("Valor : %d\n", vetor[i].data);
}

void run_list()
{
    element *pList;
    clock_t time_elapsed[2];

    void init_list(element **pElement);
    void insert_element(element **pRecebido);
    void show_elements(element **pRecebido);

    time_elapsed[0] = clock();

    pList = (element *)malloc(sizeof(struct elementList));
    init_list(&pList);

    //Inserindo elementos
    for(int i = 0; i < width; i++)
        insert_element(&pList);

    time_elapsed[1] = clock();

    //show_elements(&pList);

    double total = (time_elapsed[1] - time_elapsed[0]) * 1000.0 / CLOCKS_PER_SEC;

    printf("\n\nTempo gasto para inserir 1 milhão de itens em uma lista encadeada: %g
ms.", total);

    getchar();
}

void init_list(element **pElement)
{
    (*pElement)->next = NULL;
}

void insert_element(element **pElement)
{
    element *temp = (element *)malloc(sizeof(element));
    temp->data = rand() % width + 1;
    temp->next = (*pElement)->next;
    (*pElement)->next = temp;
}

void show_elements(element **pElement)
{

```

```
    element *temp;
    if ((*pElement)->next == NULL)
        printf("A lista está vazia");
    else
    {
        temp = (element *)malloc(sizeof(element));
        temp = (*pElement)->next;
    while(temp != NULL)
        {
            printf("Valor : %d\n", temp->data);
            temp = temp->next;
        }
    }
}
```