

# **CENTRO UNIVERSITÁRIO UNIDOMBOSCO**

Gabriela Berdak Martins RA:2320765

Giuliano Varela RA:2320965

Guilherme Marcelo Gonçalves RA:2320880

## **Algoritmo Best Fit na Alocação de Memória em Sistemas Computacionais**

Curitiba,  
Junho de 2024

## Sumário

Introdução(1) .....	3
Gerenciamento de memória(2).....	3
Características do aplicativo(3).....	3
Tamanho do aplicativo(3.1) .....	3
Sistema operacional(3.2) .....	4
Tamanho de memória e disco e linguagem(3.3) .....	4
Compilador e biblioteca(3.4) .....	4
Telas do sistema(4) .....	4-5
Referências(5) .....	5
Listagem do código fonte(6) .....	6-10

## 1 Introdução

A gestão eficiente de memória é um aspecto crucial no desempenho de sistemas computacionais. Diversos algoritmos foram desenvolvidos para lidar com a alocação e desalocação de memória de maneira eficaz. Entre eles, se destaca bastante o modelo **Best Fit**, que tem como objetivo encontrar o bloco de memória livre mais adequado (menor bloco suficientemente grande) para uma nova alocação.

O algoritmo Best Fit tenta minimizar o desperdício de memória ao escolher o bloco que deixa o menor fragmento de espaço de memória não utilizado. No entanto, essa abordagem pode levar a uma fragmentação externa significativa, onde pequenos espaços de memória ficam inutilizados, aí que entramos com o sistema de realocação de memória que nos permite reorganizar os blocos de memória para criação de blocos maiores aproveitando todo o espaço disponível.

## 2 Gerenciamento de memória

O sistema utiliza uma estrutura de dados baseada em lista encadeada de blocos (**'Block'**), onde cada bloco representa uma área de memória com um tamanho específico. Esses blocos podem estar marcados como livres ou ocupados, indicando que estão disponíveis para alocação ou estão em algum outro processo

## 3 Características do aplicativo

O aplicativo a seguir recria o algoritmo que acontece por baixo dos panos sem o usuário comum saber de nada, isso foi criado e simulado utilizando a linguagem C.

### 3.1 Tamanho do aplicativo

O tamanho do aplicativo pode variar dependendo de sistema operacional e do compilador utilizado para compilação, neste caso foi usado o sistema operacional Windows 10 e o compilador MSVC (Microsoft Visual C/C++), nesse ambiente o aplicativo pesa 8KB

### 3.2 Sistema operacional

O aplicativo pode ser executado em muitos sistemas operacionais como: Windows, Linux e MacOS, o requisito base é que o sistema suporte a linguagem de programação C.

### 3.3 Tamanho de memória e disco e linguagem

**Memória Ram:** O aplicativo utiliza cerca de 0,3MB em sua execução.

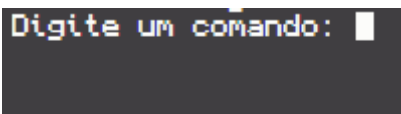
**Memória em disco:** O tamanho em disco do aplicativo é de 8KB.

Com certeza o aplicativo teve um grande bônus por usar a linguagem C, que acaba fazendo seu desempenho incrível, por ser uma linguagem de baixo nível.

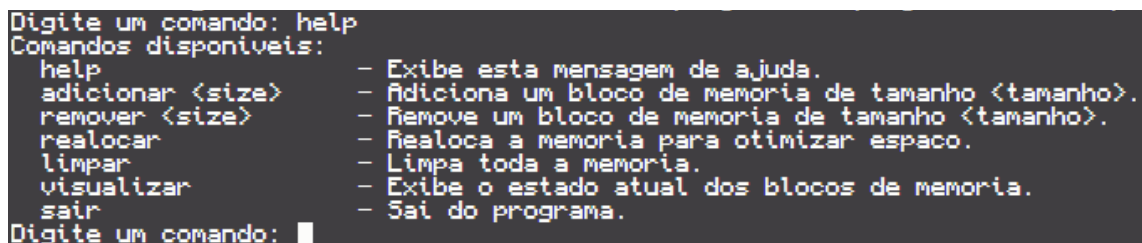
### 3.4 Compilador e biblioteca

Para compilar o aplicativo foi utilizado o compilador MSVC(Microsoft Visual C/C++), as bibliotecas utilizadas são <stdio.h>, <stdlib.h> e <string.h> que são bibliotecas padrão da linguagem e fazem respectivamente: entrada e saída, funções utilitárias e manipulação de dados e gerenciamento de memória.

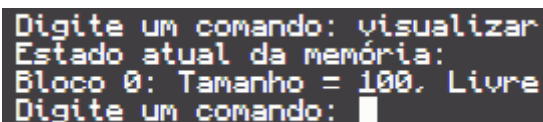
## 4 Telas do sistema



- Tela inicial que pede para o usuário digitar um comando



- Ao digitar o comando **help** o usuário recebe todos os comandos existentes no sistema



- O comando **visualizar** exibe todos os blocos de memória que existem no momento

```
Digite um comando: adicionar 10
Memoria de tamanho 10 adicionada.
Digite um comando: visualizar
Estado atual da memória:
Bloco 0: Tamanho = 10, Ocupado
Bloco 1: Tamanho = 90, Livre
Digite um comando: █
```

- O comando **adicionar** cria um bloco de memória se existir espaço suficiente, se utilizar adicionar<tamanho>.

```
Digite um comando: remover 10
Memoria de tamanho 10 removida.
Digite um comando: █
```

- O comando **remover** remove o primeiro bloco de memória do tamanho que foi descrito que achar.

```
Bloco 0: Tamanho = 10, Livre
Bloco 1: Tamanho = 90, Livre
Digite um comando: █
```

```
Digite um comando: realocar
Memoria realocada.
Digite um comando: █
```

- O comando **realocar** faz com que todos os blocos livres se unam e virem um só.

```
Digite um comando: limpar
Memoria limpa.
Digite um comando: visualizar
Estado atual da memória:
Bloco 0: Tamanho = 100, Livre
Digite um comando: █
```

- O comando **limpar** limpa toda a memória basicamente reiniciando o aplicativo.

```
Digite um comando: sair
Saindo do programa.
PS C:\Users\guilh\OneDrive\Área de Trabalho
```

- E finalmente, o comando **sair** faz com que o aplicativo pare de rodar na máquina.

## 5 Referências

<https://www.geeksforgeeks.org/program-best-fit-algorithm-memory-management/>

[https://en.wikipedia.org/wiki/Best-fit\\_bin\\_packing](https://en.wikipedia.org/wiki/Best-fit_bin_packing)

[https://en.wikipedia.org/wiki/C\\_\(programming\\_language\)#:~:text=C%20is%20an%20imperative%20procedural,all%20with%20minimal%20runtime%20support.](https://en.wikipedia.org/wiki/C_(programming_language)#:~:text=C%20is%20an%20imperative%20procedural,all%20with%20minimal%20runtime%20support.)

<https://www.youtube.com/@devilr3902>

## 6 Listagem do código fonte

```
7  #include <stdio.h> //Importa uma função padrão de entrada e saída
8  #include <stdlib.h> //Importa uma função padrão de funções utilitárias
9  #include <string.h> //Importa uma função padrão de manipulação
10
11 #define MEMORIA_MAXIMA 100 //Define tamanho máximo de memória
12 #define COMMAND_SIZE 50 //Define o tamanho máximo de um comando
13
14 //Definição de um bloco de memória
15 typedef struct Block {
16     int size; //Tamanho do bloco
17     int is_free; //Indica se o bloco está livre (1) ou ocupado (0)
18     struct Block *next; //Ponteiro para o próximo bloco
19 } Block;
20
21 Block *head = NULL; //Ponteiro global no início da lista
22
23 // Função para inicializar a memória com um bloco único de espaço MEMORIA_MAXIMA e
    livre
24 void initialize_memory() {
25     head = (Block *)malloc(sizeof(Block)); //Aloca memória para o bloco inicial
26     head->size = MEMORIA_MAXIMA; //Define o tamanho do bloco inicial
27     head->is_free = 1; //Marca o bloco inicial como livre
28     head->next = NULL; //Define que não há próximo bloco
29 }
30
31 //Função que exibe todas as funções existentes para o usuário
32 void help() {
33     printf("Comandos disponiveis:\n");
34     printf("  help          - Exibe esta mensagem de ajuda.\n");
35     printf("  adicionar <size> - Adiciona um bloco de memoria de tamanho <tamanho>.\n");
36     printf("  remover <size>   - Remove um bloco de memoria de tamanho <tamanho>.\n");
37     printf("  realocar        - Realoca a memoria para otimizar espaco.\n");
38     printf("  limpar          - Limpa toda a memoria.\n");
39     printf("  visualizar      - Exibe o estado atual dos blocos de memoria.\n");
40     printf("  sair           - Sai do programa.\n");
41 }
42
43 //Função que adiciona um bloco de memória
44 void adicionar(int size) {
45     Block *current = head; // Ponteiro que percorre a lista de blocos
46     Block *best_fit = NULL; // Ponteiro para o melhor bloco encontrado que pode acomodar o
        novo bloco
47
48     // Procura o bloco livre com Best Fit (menor tamanho suficiente)
49     while (current != NULL) {
50         if (current->is_free && current->size >= size) {
```

```

51     if (best_fit == NULL || current->size < best_fit->size) {
52         best_fit = current;
53     }
54 }
55 current = current->next;
56 }
57
58 // Se encontrou um bloco adequado, realiza a alocação
59 if (best_fit != NULL) {
60     if (best_fit->size > size) {
61
62         // Se o bloco é maior que o necessário, cria um novo bloco com o espaço restante
63         Block *new_block = (Block *)malloc(sizeof(Block));
64         new_block->size = best_fit->size - size;
65         new_block->is_free = 1;
66         new_block->next = best_fit->next;
67         best_fit->next = new_block;
68     }
69     best_fit->size = size; // Ajusta o tamanho do bloco para o tamanho solicitado
70     best_fit->is_free = 0; // Marca o bloco como ocupado (0)
71     printf("Memoria de tamanho %d adicionada.\n", size);
72 } else {
73     // Se não encontrou um bloco adequado, informa que não há espaço suficiente
74     printf("Nao foi possivel adicionar memoria de tamanho %d. Sem espaco suficiente.\n",
75 size);
76 }
77
78 // Função para remover um bloco de memória
79 void remover(int size) {
80     Block *current = head;
81     while (current != NULL) {
82         if (!current->is_free && current->size == size) {
83             current->is_free = 1; // Marca o bloco como livre(1)
84             printf("Memoria de tamanho %d removida.\n", size);
85             return;
86         }
87         current = current->next;
88     }
89     // Se não encontrou um bloco do tamanho especificado, informa ao usuário
90     printf("Nao foi encontrado bloco de memoria de tamanho %d.\n", size);
91 }
92
93 // Função de realocar memória, acha todos os blocos livres e os une em um só
94 void realocar() {
95     Block *current = head;
96     int total_free_size = 0;
97
98     // Calcula o tamanho total de todos os blocos livres e remove-os

```

```

99  Block *prev = NULL;
100 while (current != NULL) {
101     if (current->is_free) {
102         total_free_size += current->size;
103         if (prev != NULL) {
104             prev->next = current->next;
105             free(current);
106             current = prev->next;
107         } else {
108             head = current->next;
109             free(current);
110             current = head;
111         }
112     } else {
113         prev = current;
114         current = current->next;
115     }
116 }
117
118 // Se houve blocos livres, cria um novo bloco grande
119 if (total_free_size > 0) {
120     Block *new_block = (Block *)malloc(sizeof(Block));
121     new_block->size = total_free_size;
122     new_block->is_free = 1;
123     new_block->next = head;
124     head = new_block;
125 }
126
127 printf("Memoria realocada.\n");
128 }
129
130 //Função que limpa a memória
131 void limpar() {
132     Block *current = head;
133     while (current != NULL) {
134         Block *temp = current;
135         current = current->next;
136         free(temp);
137     }
138     head = NULL;
139     initialize_memory();//Reinicializa a memória após limpar
140     printf("Memoria limpa.\n");
141 }
142
143 //Função que visualiza os blocos de memória
144 void visualizar() {
145     Block *current = head;
146     int index = 0;
147     printf("Estado atual da memória:\n");

```



```

148 while (current != NULL) {
149     printf("Bloco %d: Tamanho = %d, %s\n", index, current->size, current->is_free ? "Livre"
150         : "Ocupado");
151     current = current->next;
152     index++;
153 }
154
155 // Função que processa os comandos do usuário
156 void process_command(char *command) {
157     char *token = strtok(command, " ");
158     if (strcmp(token, "help") == 0) {
159         help();
160     } else if (strcmp(token, "adicionar") == 0) {
161         token = strtok(NULL, " ");
162         if (token != NULL) {
163             int size = atoi(token);
164             adicionar(size);
165         } else {
166             printf("Uso: adicionar <tamanho>\n");
167         }
168     } else if (strcmp(token, "remover") == 0) {
169         token = strtok(NULL, " ");
170         if (token != NULL) {
171             int size = atoi(token);
172             remover(size);
173         } else {
174             printf("Uso: remover <tamanho>\n");
175         }
176     } else if (strcmp(token, "realocar") == 0) {
177         realocar();
178     } else if (strcmp(token, "limpar") == 0) {
179         limpar();
180     } else if (strcmp(token, "visualizar") == 0) {
181         visualizar();
182     } else if (strcmp(token, "sair") == 0) {
183         printf("Saindo do programa.\n");
184         exit(0);
185     } else {
186         printf("Comando desconhecido: %s\n", token);
187         help();
188     }
189 }
190
191 // Função que roda ao iniciar o programa
192 int main() {
193     initialize_memory(); // Inicializa a memória ao iniciar o programa
194
195     char command[COMMAND_SIZE];

```

```
196 while (1) {
197     printf("Digite um comando: ");
198     if (fgets(command, COMMAND_SIZE, stdin) != NULL) {
199         command[strcspn(command, "\n")] = '\0'; // Limpa a entrada de usuário para não
            existir caracteres de nova linha como \n
200         process_command(command);
201     }
202 }
203
204 return 0;
205 }
```