Estrutura de Dados e Algoritmos 2: Lista 01

Maurício Serrano

Paulo Tada 11/0135431 Maria Luciene 12/0037742

Exercício 1

O algoritmo a seguir representa a uma busca sequencial com a utilização de indexação primária.

Listing 1: Busca sequencial com indice primario

```
// Com base no codigo da busca sequencial, evolua o
// codigo para uma busca sequencial com indice primario.

#include <iostream>
#include <sstream>
#include <string>

#define MAX 20
#define INDEX 5

using namespace std;

typedef struct index
{
     int position;
     int value;
}Index;


int search(int vector[], int element, int position){
  int result = -1;

  cout << endl << "Position " << position << endl;
  for(int i = position; i<MAX ; i++){
    if(vector[i] == element){
      cout << "entrou no for" << endl;
      result =  i;
    }
  }
  return result;
}

int search_table(int vector[], Index index[], int number) {
  int i = 0;
  int result = 0;
  while(number >= index[i].value && i<INDEX){
     i++;
  }

  cout << "I: " << i << endl;
  result = search(vector,number,index[i-1].position);
  return result;
}
void index_table(int vector[], Index index[]){
  int l = 0;
  for(int i = 0; i<INDEX ; i++) {
    index[i].value = vector[l];// Guarda a posicao do numero pertencente ao vetor
    index[i].position = l;// Popula o posicao dos valores no indice
```

```
49      l+=4;
50    }
51  }
52
53
54  void print(int vector[]){
55    for(int i = 0; i < MAX ; i++){
56      cout << vector[i] << " ";
57    }
58  }
59
60  void print_index(Index index[]) {
61    for (int i = 0; i < INDEX; ++i) {
62      cout << index[i].value << " ";
63    }
64  }
65
66
67  int main(){
68
69    Index index[INDEX];
70    int vector[MAX] = {1,3,12,23,46,67,87,99,120,150,160,177,182,203,230,
71      254,311,313,369,389};
72    int element = 0;
73
74    index_table(vector,index);
75    cout << "Vector " << endl;
76    print(vector);
77    cout << endl << "Index " << endl;
78    print_index(index);
79
80    cout << endl << "Insert the number" << endl;
81    cin >> element;
82
83    int result = search_table(vector,index,element);
84
85    if(result == -1){
86      cout << "Element not found" << endl;
87    }else{
88      cout << "Element found on position " << result << endl;
89    }
90
91    return 0;
92  }
```

A indexação é feita com a criação de a struct index contendo a posição e o valor do indice. O tamanho do vetor de indexação é definido pelas constantes INDEX. A definição dos valores dos indices ocorrem de 4 em 4 do vetor de busca (função index_table).

Exercício 2

O código abaixo foi desenvolvido apartir de uma estrutura de lista encadeada e modificado para lógica de lista circular com a implementação de busca binária.

Listing 2: Codigo de busca binaria em lista circular

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

/*
     Busca binaria em lista circular
*/

typedef struct node{
    int number;
    struct node *next;
}Node;

typedef struct list{
    Node *top;
    Node *tail;
}List;

void enqueue(List *list);
int empty(List *list);
void print_list(List *list);
void dequeue(List *list);
void clear(List *list);
Node *find_midle(Node *begin, Node* end);
Node *binary_search(List *list, int key);
void search(List *list);
void menu();



int main(){
    List *list = (List *)malloc(sizeof(List));
    list->top = list->tail = NULL;
    int option;

    do{
     menu();
        scanf("%d",&option);

         switch(option) {
            case 1:
                enqueue(list);
                break;
            case 2:
               if(!empty(list)){
                    dequeue(list);
               }
                break;
```

```
48              case 3:
49                 if(!empty(list)){
50                     clear(list);
51                  }
52                  break;
53              case 4:
54                 if(!empty(list)){
55                      print_list(list);
56                 }
57                  break;
58               case 5:
59                 search(list);
60                 break;
61           }
62
63      }while(option != 6);
64
65  return 0;
66  }
67
68  void menu() {
69       printf("\n----------------Menu----------------\n");
70       printf("1. Insert number\n");
71       printf("2. Remove first number\n");
72       printf("3. Empty line\n");
73       printf("4. Print elements\n");
74       printf("5. Search for a value\n");
75       printf("6. Exit\n");
76       printf("\n--------------------------------------\n");
77  }
78
79
80
81  Node *find_midle(Node *begin, Node *end) {
82
83       if(begin == NULL){
84            return NULL;
85       }
86
87       Node *slow = begin;
88       Node *fast = begin->next;
89
90       while(fast != end) {
91            fast =  fast->next;
92
93            if(fast != end) {
94
95                 fast =  fast->next;
96                 slow = slow->next;
97            }
98       }
99
100      return slow;
```

```
}


Node  *binary_search(List *list, int key) {
     Node *begin = list->top;
     Node *end = list->tail;

     while(begin != end) {
          Node *midle = find_midle(begin, end);

          if(midle == NULL) {
               return NULL;
          }

          if(midle->number ==  key){
               return midle;
          }else if(midle->number < key) {
               begin = midle->next;
          }else {
               end = midle;
          }
     }

     // Number not present
     return NULL;
}

void search(List *list) {
     Node *result;
     int key;

     printf("Enter the number you want to find:");
     scanf("%d", &key);

     result = binary_search(list, key);

     if(result == NULL) {
          printf("\nValue not found\n");
     }else{
          printf("\nElement %d found.\n", result->number);
     }
}

void enqueue(List *list)
{
    Node *temp = (Node *)malloc(sizeof(Node));
    temp->next = NULL;

    printf("Enter a value: \n");
    scanf("%d", &temp->number);

    if(list->top == NULL)
    {
```

```
154         list->top = list->tail = temp;
155         list->top->next = list->top;
156     }
157      else
158      {
159         list->tail->next = temp;
160         list->tail = temp;
161         temp->next = list->top;
162      }
163 
164 }
165 
166 int empty(List *list)
167 {
168     if(!list->top){
169         printf("a pilha esta vazia\n");
170         return 1;
171     }else{
172      return 0;
173     }
174 }
175 
176 void print_list(List *list)
177 {
178     Node *temp = list->top;
179 
180     do {
181 
182       printf("Value: %d\n", temp->number);
183       temp = temp->next;
184 
185     } while(temp != list->top);
186 
187 }
188 
189 void dequeue(List *list)
190 {
191     Node *temp = list->top;
192 
193     if(list->top == list->tail){
194      free(list);
195      list->top = list->tail = NULL;
196     }else{
197          list->top = list->top->next;
198          list->tail->next = list->top;
199          free(temp);
200     }
201 }
202 
203 void clear(List *list){
204 
205      Node *node =  list->top;
206      Node *removed_node = NULL;
```

```
207
208         do {
209               removed_node = node;
210               node = node->next;
211               free(removed_node);
212
213         }while(node != list->top);
214
215
216       printf("Deleted succeeded!\n");
217       free(list);
218       list->top = list->tail = NULL;
219   }
```

A busca pelo meio da lista é dada pela função "find_middle". A busca pelo meio é feito com a utilização de dois ponteiros: slow(inicia em um nó) e fast(inicia um nó a frente do slow). Ao percorrer a lista, a cada um passo do ponteiro slow, temos o ponteiro fast avançando duas casas. Quando o vetor é interiamente percorrido o ponteiro slow está na posição do meio.

Exercício 3

O código abaixo é uma versão modificada de um algoritmo de busca binária utilizando vetor. A alteração insere o método da interpolação (linha 13).

Listing 3: C++ Code

```
1  // Binary Search
2  #include <iostream>
3  #include <stdlib.h>
4  #include <ctime>
5
6  using namespace std;
7
8  // Binary search adding interpol method
9  int binary_search(int vector[], int element, int min, int max){
10   int result = -1;
11   while(min<=max){
12     // Modification to use interpol on binary search
13     int mid = min + (max - min)*((element-vector[min])/(vector[max]-vector[min]));
14     if(vector[mid] == element){
15       result = mid;
16       break;
17     } else
18     if(vector[mid] < element){
19       min = mid + 1;
20     } else {
21       max = mid - 1;
22     }
23   }
24   return result;
25 }
26
27 void preconfig(int vector[], int max_size){
28   for(int i = 0 ; i<max_size ; i++){
29     vector[i] = i*2;
30   }
31 }
32
33 int main(){
34   int max_size = 0; // Max size of the vector
35   while(1){
36     cin >> max_size;
37     if(max_size==-1) break;
38
39     int vector[max_size];
40     preconfig(vector, max_size); // Creates a vector with multiples of 2
41     int element = 0; // Element to find
42     cin >> element;
43
44     clock_t start, end;
45     start = clock();
46     binary_search(vector,element,0,max_size-1);
47     end = clock();
```

```
48
49      long double tmili = (end - start)/(double)(CLOCKS_PER_SEC/1000);
50      cout << "Time: " << tmili << " ms" << endl;
51
52    }
53    return 0;
54  }
```

Com a execução do código com os casos de teste com vetores de tamanho: 10, 25, 50, 100, 500, 1000, 10000, 100000.

Listing 4: Dados de entrata para os casos de teste

```
1  10
2  12
3  25
4  12
5  50
6  12
7  100
8  12
9  500
10 12
11 1000
12 12
13 10000
14 12
15 100000
16 12
17 1000000
18 12
19 -1
```

Após a execução do programa temos o tempo da busca em cada vetor, como demonstrado a seguir.

Listing 5: Dados de saída para os casos de teste

```
1  Time: 0.004 ms
2  Time: 0.003 ms
3  Time: 0.003 ms
4  Time: 0.003 ms
5  Time: 0.002 ms
6  Time: 0.003 ms
7  Time: 0.003 ms
8  Time: 0.003 ms
9  Time: 0.003 ms
```

Ao executar os casos de testes em varios momentos, não se obteve uma variação significativa entre o tempo de execução entre os diferentes tipos de tamanho de vetores.