Lista de Algoritmos 2ªva

Fábio Alves de Freitas 03/09/2017

Questão 1

```c
#include <stdio.h>
#include <stdlib.h>
#define tam 10

int inicio = 0, final = 0;

// checa se a fila est  vazia
int filavazia() {
        return inicio == final;
}

// checa se a fila est  cheia
int filacheia() {
        return (final+1)%tam == inicio;
}

// insere um elemento no final da fila
void colocanafila(int* fila, int conteudo) {
        if(!filacheia()) {
                fila[final++] = conteudo;
                if(final == tam) {
                        final = 0;
                }
        }
}

// tira um elemento do inicio da fila
int tiradafila(int* fila) {
        if(!filavazia()) {
                int n = fila[inicio++];
                if(inicio == tam) {
                        inicio = 0;
                }
                return n;
        }
```

```
}

// retorna o coprimento da fila (n mero de elementos)
int comprimentoDaFila () {
        int n;
        int count = 0;
        if(inicio > final ) {
                for( n = inicio ; n < tam  ; n++ )
                        count++;
                for( n = 0 ; n < final  ; n++ )
                        count++;
                return count;
        }
        for( n = inicio ; n < final ; n++ )
                count++;
        return count;
}
```

Questão 3

```
#include <stdio.h>
#define TAM 10

int fila[TAM];
int frente, tras;

// inicializa a fila com valores padroes
void inicializarFila () {
        frente = 5;
        tras = 5;
        int n;
        for(n=0;n<TAM;n++) {
                fila[n] = -1;
        }
}

// checa se a fila est  cheia
int filaCheia() {
        return (frente+1)%TAM == tras;
}

// checa se a fila est  vazia
int filaVazia() {
        return frente == tras;
}

// insere a frente da fila
```

```
28 void insereFrente(int num) {
29         if(!filaCheia()) {
30                 fila[frente++] = num;
31                 if(frente == TAM) {
32                         frente = 0;
33                 }
34         }
35         else {
36                 printf("fila cheia");
37         }
38 }
39
40 // insere atr s da fila
41 void insereTras(int num) {
42         if(!filaCheia()) {
43                 if(tras-1 == -1) {
44                         tras = TAM-1;
45                         fila[tras] = num;
46                 }
47                 else {
48                         fila[--tras] = num;
49                 }
50         }
51         else {
52                 printf("fila cheia");
53         }
54 }
55
56 // retorna o elemento na frente da fila
57 int removeFrente() {
58         if(!filaVazia()) {
59                 if(frente == 0) {
60                         frente = TAM-1;
61                         fila[frente] = -1;
62                         return fila[frente];
63                 }
64                 fila[frente-1] = -1;
65                 return fila[--frente];
66         }
67         else {
68                 printf("fila vazia");
69         }
70 }
71
72 // retorna o elemento atr s da fila
73 int removeTras() {
74         if(!filaVazia()) {
```

```
                int aux = fila[tras];
                fila[tras++] = -1;
                if(tras == TAM) {
                        tras = 0;
                }
        }
        else {
                printf("fila vazia");
        }
}

void imprimeFila () {
        int n;
        for(n=0;n<TAM;n++) {
                printf("(%i)",fila[n]);
        }
}

int main() {
        inicializarFila();
        imprimeFila();

        insereFrente(7);
        insereFrente(8);
        insereTras(6);
        insereTras(5);

        printf("\n");
        imprimeFila();

        removeFrente();
        removeTras();
        removeTras();

        printf("\n");
        imprimeFila();
}
```

Questão 4

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

typedef struct Cell {
        int value;
        struct Cell* prox;
}List;
```

```
8
9  void imprimir (List* a) {
10         List* n;
11         for( n = a->prox ; n != NULL ; n = n->prox ) {
12                 printf("(%i)",n->value);
13         }
14 }
15
16 // insere no fim de uma lista encadeada
17 void inserirFim(List *list, int value) {
18         List *novo= (List*) malloc(sizeof(List));
19         if(!novo){
20                 printf("\nSem memoria disponivel!\n");
21                 exit(1);
22         }
23         novo->value = value;
24         novo->prox = NULL;
25
26         if(list->prox == NULL)
27                 list->prox=novo;
28         else{
29                 List *tmp = list->prox;
30
31                 while(tmp->prox != NULL)
32                         tmp = tmp->prox;
33
34                 tmp->prox = novo;
35         }
36 }
37
38 // recebe duas listas (a e b) a serem intercaladas numa terceira (c)
39 void intercalarListas (List* a, List* b, List* nova) {
40         List *x = a->prox, *y = b->prox; // x compara a lista a, y compara
              a lista b
41         // passamos apenas a cabe a das listas, logo nao h  conteudo nas
              listas
42         if(x == NULL || y == NULL) {
43                 if(x == NULL && y == NULL) {
44                         printf("Listas Vazias\n");
45                         exit(1);
46                 }
47                 if(x == NULL && y != NULL) {
48                         while(y != NULL) {
49                                 inserirFim(nova, y->value);
50                                 y = y->prox;
51                         }
52                 }
```

```
                if(y == NULL && x != NULL) {
                        while(y != NULL) {
                                inserirFim(nova, y->value);
                                y = y->prox;
                        }
                }
        }
        else {
                // s  paramos o la o quando chegarmos no NULL de ambas as
                    listas
                while(x != NULL || y != NULL) {
                        // se a primeira lista apontar para nulo, copiamos
                           a segunda na nova lista
                        if(x == NULL && y != NULL) {
                                while(y != NULL) {
                                        inserirFim(nova, y->value);
                                        y = y->prox;
                                }
                                break;
                        }
                        // se a segunda lista apontar para nulo, copiamos a
                            primeira na nova lista
                        if(y == NULL && x != NULL) {
                                while(x != NULL) {
                                        inserirFim(nova, x->value);
                                        x = x->prox;
                                }
                                break;
                        }
                        // nunhuma das listas apontam para nulo
                        if(x != NULL && y != NULL) {
                                if(x->value <= y->value) {
                                        inserirFim(nova, x->value);
                                        x = x->prox;
                                }
                                else {
                                        inserirFim(nova, y->value);
                                        y = y->prox;
                                }
                        }
                }
        }
}
```

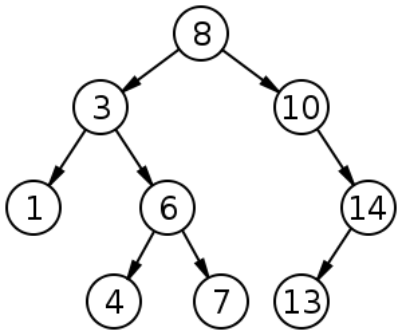
Questão 5

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

```c
#include <stdlib.h>

typedef struct List {
        int value;
        struct List* prox;
}List;

// recebe a lista completa, e duas celulas dessa lista
// troca as duas celular de posicao
void trocar_Lista_Simples (List* lista, List *a, List *b) {
        // buscar quais celular apontam para a e b
        // x aponta para a
        // y aponta para b
        List *antA = NULL, *antB = NULL, *n;
        int num = 0;
        for(n = lista ; n != NULL ; n = n->prox) {
                if(n->prox == a) {
                        antA = n;
                }
                if(n->prox == b) {
                        antB = n;
                }
                if(antA != NULL && antB != NULL) {
                        break;
                }
        }
        antA = b;
        antB = a;
        n = b->prox;
        b->prox = a->prox;
        a->prox = n;
}
```

Questão 7

A proposição não é verdade, pois pelo contra exemplo a seguir é possível demonstrar que a profundidade mais a altura de um nó qualquer de uma árvore binária nao necessáriamente resultará em sua altura.



Calculamos a altura e a profundidade do nó com o número 1.

- altura do nó 1 = 2
- produndidade do nó 1 = 0
- 2 + 0 = 2, mas a altura da ávore é 4

A altura mais a profundiade do nó 1 não resulta na altura da árvore, logo a propriedade não funciona.

Questão 8

```c
#include <stdio.h>

typedef struct Node {
        int valor;
        struct Node *dir;
        struct Node *esq;
}Arvore;

Arvore* primeiro (Arvore *tree) {
        if(tree->esq == NULL)
                return tree;
        return primeiro (tree->esq);
}
```

Questão 9

```c
#include <stdio.h>

typedef struct Node {
        int valor;
        struct Node *dir;
        struct Node *esq;
}Arvore;

Arvore* ultimo (Arvore *tree) {
        if(tree->dir == NULL)
                return tree;
        return ultimo(tree->dir);
}
```

Questão 19

A complexidade desta função é proporcional a linear, pois o primeiro laço efetua N (256) iterações e o segundo laço efetua o número de caracteres contidos no arquivo de iterações, que podemos chamar de M.

- Logo o a complexidade total do algoritmo é de N+M.

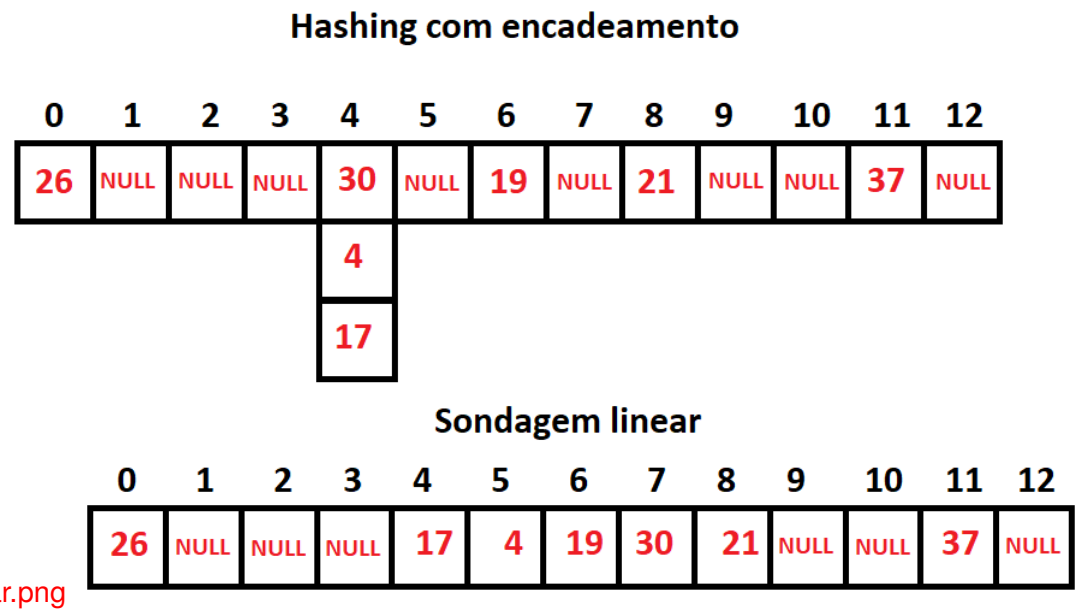
```c
#include <stdio.h>
#define ASCII 256

void tirarRepeticao (FILE* leitura, FILE* escrita) {
        if(escrita == NULL || leitura == NULL) {
                printf("Erro! Arquivos nulos");
                return;
        }
        char c;
        int tabela[ASCII], n;
        for(n = 0; n < ASCII; n++) {
                tabela[n] = -1;
        }
        while(1) {
                c = getc(leitura);
                if(feof(leitura)) {
                        break;
                }
                // sem colisao
                if(tabela[c] == -1) {
                        tabela[c] = c;
                        putc(c, escrita);
                }
        }
}



int main() {
        FILE* arq = fopen("Texto.txt","r");
        FILE* aux = fopen("SemRepeticao.txt","w");
        tirarRepeticao(arq, aux);

        fclose(arq);
        fclose(aux);
}
```

Questão 20

Neste exercício, eu utilizei os conceitos de Hashing e Linked list para o funcionamento do algoritmo. Criei um vetor (Tabela hash) de listas encadeadas, chamadas Conjunto. Casa célula da lista conjunto possui um contador e outra lista encadeada, chamada List, que representa uma palavra. Primeiro um laço lê um arquivo de texto, caracter a caracter, afim de achar uma

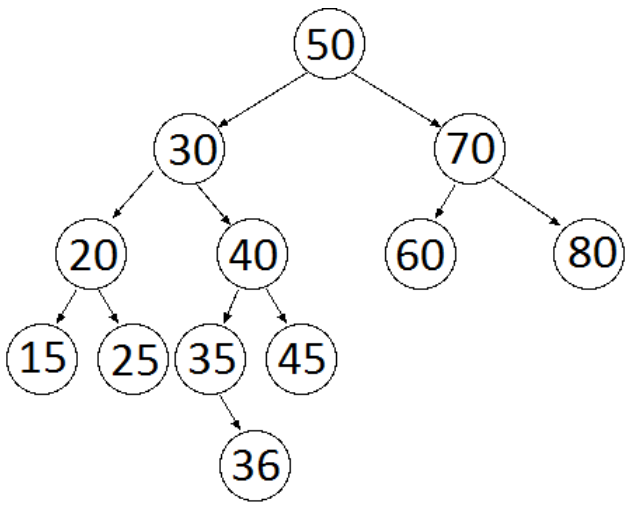
palavra. Quando é encontrada a palavra é armazenada numa lista encadeada auxiliar do tipo List. Calcula-se o Hash correspondente desta palavra e é feita a checagem de colisões. Caso nao ocorra uma colisao criamos uma nova célula do tipo Conjunto inserimos a palavra lida anteriormente na lista List presente nessa célula e inserimos a nova célula na tabela hash. Caso ocorra um colisão checamos se a palavra já foi armazenada nesta posição do hash. Se sim apenas incrementamos seu contador. Se não criamos outra célula do tipo Conjunto inserimos a palavra e colocamos esta nova célula no final da lista correspondente ao hash calculado.

Questão 18



linear.png

Questão 14



14.png