

Lista II

Alunos: Felipe Ferreira Marra e Fabrício Dinali Adão

Exercícios dos Slides:

Exercício 1 (1,0): Procure na internet e dê mais dois exemplos de aplicações de grafos. Explique as aplicações, destacando como o grafo é usado nas aplicações, qual tipo ou quais tipos de grafos são usados e quais algoritmos de grafos são utilizados.

Síntese Ativ 3

/ /

1-) Os grafos apresentam diversas aplicações e podem ser utilizados em representações de diferentes temas, como exemplo temos o uso de grafos em redes de transporte aéreo. Neste caso, os grafos representam as diferentes rotas possíveis, de modo que cada vértice do grafo representa uma cidade específica e as conexões entre vértices representam as rotas possíveis. Um algoritmo utilizado em redes de transporte aéreo com grafos é o caminho mínimo, no qual mapeia-se o menor caminho a ser percorrido em uma determinada viagem, tendo em vista uma chegada e saída específicas. Outro exemplo de aplicação de grafos se dá em distribuição de energia elétrica, nela temos equipamentos interligados por elementos elétricos (linhas) assim na representação por grafos os equipamentos serão representados pelos vértices e as linhas pelas arestas.

Exercício 2 (1,0): Considerando um grafo já criado com um determinado número de vértices, como podemos implementar as operações de criação e remoção de vértices (usando a matriz de adjacência)? Pense numa solução e discuta as vantagens e desvantagens da sua solução. Discuta quais os problemas em criar e remover vértices dinamicamente

2-) Para que haja a remoção ou adição de um vértice, precisaremos nos preocupar com a matriz adjacente, visto que caso desejarmos remover um vértice que se encontra no meio da matriz, desejamos liberar o espaço na memória e em seguida rearranjar os vértices que possuem posições maiores na matriz, para adicionar desejamos reservar mais espaço na matriz de adjacência, isso implica também na necessidade de alterar as arestas, ou seja, modificar os valores referentes a estas arestas dentro da matriz de adjacência. Portanto essa solução seria eficiente na criação de novos vértices apenas adicionando uma nova posição na matriz, porém o processo de remoção seria complexo, com a necessidade de serem realizados rearranjos.

tilibra

Exercício 3 (1,0): Utilizando como base os códigos da matriz de adjacência e da lista de adjacência, implemente uma função chamada adjacentes que recebe um vértice e retorna em um vetor todos os adjacentes daquele vértice.

- Lista adjacentes

Código em anexo no arquivo: 3_lista_adj.c

```
ListaAdj* adjacentes(Grafo* g, int v) {
    //primeiro item da lista de adjacentes de v
    ListaAdj* cabeca = NULL;
    //pega lista de adjacentes de v
    ListaAdj* adjacentes_v = g->lista_adj[v];
    //copia lista adj v
    while (adjacentes_v != NULL) {
        //copia no atual
        ListaAdj* novo_no = (ListaAdj*) malloc(sizeof(ListaAdj));
        novo_no->id = adjacentes_v->id;
        novo_no->peso = adjacentes_v->peso;
        novo_no->prox = NULL;
```

```

    if(cabeca == NULL)
        cabeca = novo_no;
    //anda pro proximo
    adjacentes_v = adjacentes_v->prox;
}
return cabeca;
}

```

- Matriz adjacentes

Código em anexo no arquivo: 3_matriz_adj.c

```

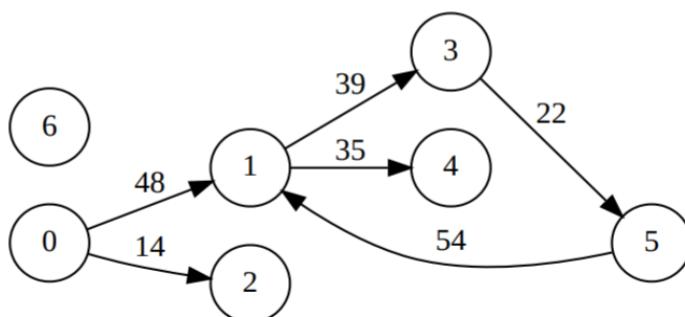
int* adjacentes(Grafo* g, int v){
    //inicializa lista que varre a linha
    int* adj = (int*) malloc(g->qv * sizeof(int));
    int count = 0;
    int count_back = 0;
    //varre a linha v procurando por pesos maiores que 0
    for (int i = 0; i < g->qv; i++){
        int peso_no_adj = g->matrizadj[v][i];
        if(peso_no_adj){
            //printf("%d eh adjacente de %d \n", i, v);
            adj[count] = i;
            count++;
        }else{
            //coloca -1 de trás pra frente pra cada posição que
            //não é adjacente
            adj[g->qv-1-count_back] = -1;
            count_back++;
        }
    }
    return adj;
}

```

Exercício 4 (1,0): Explique como podemos obter todos os adjacentes de um vértice utilizando a representação de matrizes esparsas.

Para obter os adjacentes do vértice v buscaremos na tabela todos os locais em que v é a origem, pois nestes locais os destinos serão vértices adjacentes ao mesmo.

Exemplo:



origem	0	0	1	1	3	5
destino	1	2	3	4	5	1
peso	48	14	39	35	22	54

Vamos obter os adjacentes de 0. Observamos que onde 0 é origem os destinos são 1 e 2, e, portanto 1 e 2 são adjacentes de 0. Do mesmo modo, se buscássemos os adjacentes de 1, teríamos como resultado 3 e 4.

Exercício 5 (2,0): Implemente o algoritmo acima (busca em largura) em alguma linguagem de programação de sua escolha. Entregue junto com o código o resultado de cinco testes diferentes.

Código em anexo na pasta 5_busca_largura

Após entrar na pasta para compilar e abrir o programa execute:

```
gcc -c busca.c queue.c && gcc busca.o queue.o -o program && ./program
```

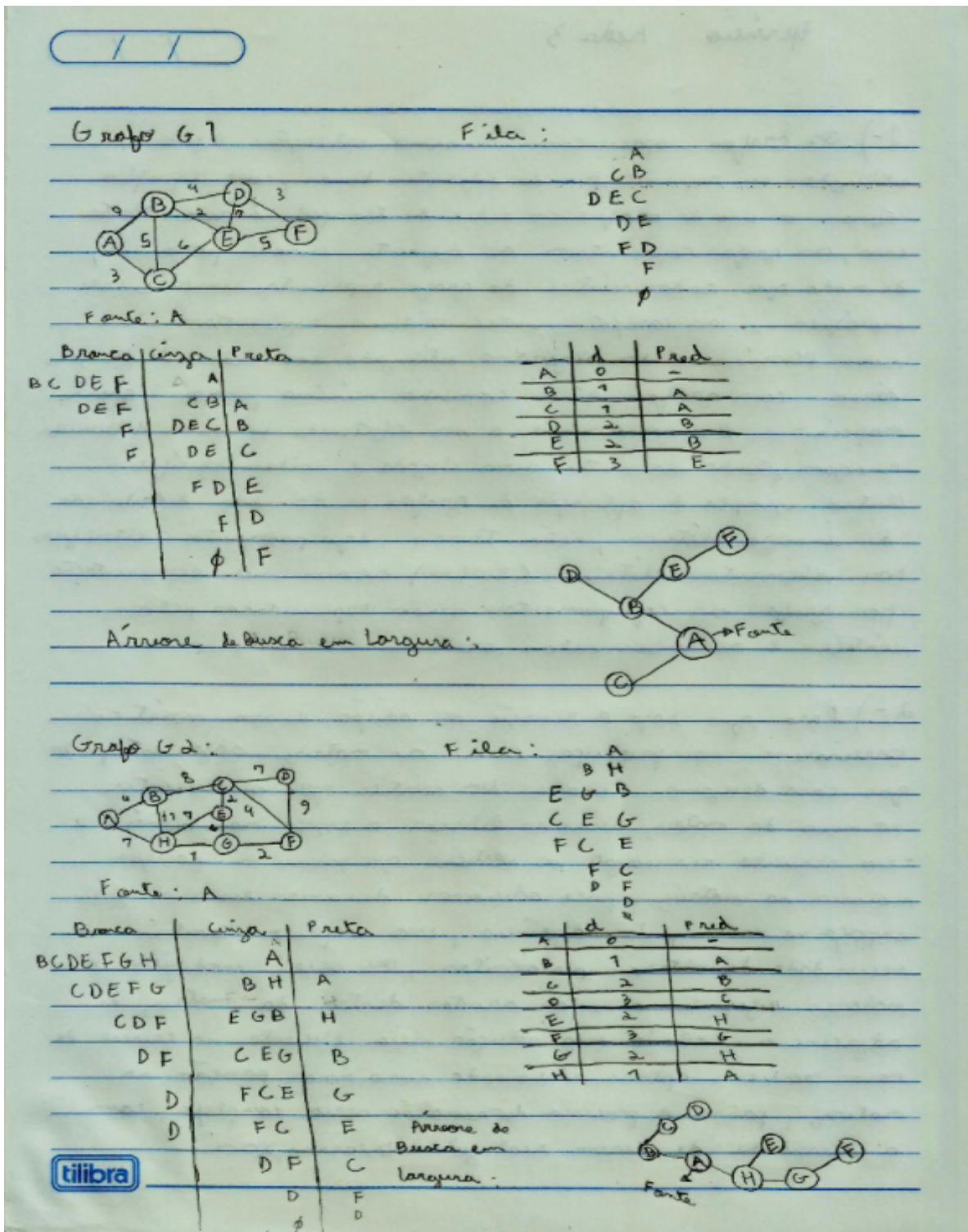
```
void buscaLargura(Grafo* g, int s){  
    //nós do grafo  
    NoBusca* nos = (NoBusca*) malloc(g->qv * sizeof(NoBusca));  
  
    //inicializa outros nós como branco e s como cinza  
    for(int i = 0; i < g->qv; i++){  
        nos[i].id = i;  
        if(i != s){  
            nos[i].cor = 0;  
            nos[i].d = INT_MAX;  
            nos[i].pred = -1;  
        }else{  
            nos[i].cor = 1;  
            nos[i].d = 0;  
            nos[i].pred = i;  
        }  
    }  
    //Inicia fila com s
```

```
Queue Q = createQueue();
insertValue(Q,s);

//Enquanto a fila não está vazia
while (!vazio(Q)){
    NoBusca u = nos[removeValue(Q)]; //Desenfileira
    int *adj = adjacentes(g,s);
    for(int i = 0; i < g->qv; i++){
        if(adj[i] != -1){
            NoBusca v = nos[adj[i]];
            v.cor = 1;
            v.d = u.d + 1;
            v.pred = u.id;
            insertValue(Q,v.id);
        }
    }
    u.cor = 2;
    printf("Vertice %d: cor = %d, d = %d e pred = %d\n", u.id,
u.cor, u.d, u.pred);
}
}
```

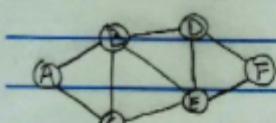
Lista

1 - a.(2.0) Mostre o passo a passo da busca em largura mostrando todas as informações utilizadas durante a busca e calculando a distância (em número de arestas) de cada vértice e os predecessores na árvore de busca em largura. Mostre a árvore de busca em largura obtida.



1 - b.(2,0) Mostre o passo a passo da busca em profundidade mostrando todas as informações utilizadas durante a busca e calculando os tempos de descoberta e término de cada vértice.

Gráfico G₁:

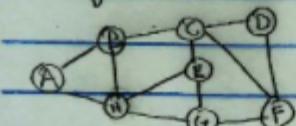


Fonte:A:

Pilha

			mãos visitadas	visitadas	Pilha
			BCDEF	A	A
A	1	12	CDEF	A,B	AB
B	2	11	DEF	A,B,C	ABC
C	3	10	DF	A,B,C,E	ABCDE
D	5	8	F	A,B,C,E,D	ABCDE
E	4	9	∅	A,B,C,E,D,F	ABCED
F	6	7			ABC

Gráfico G₂:



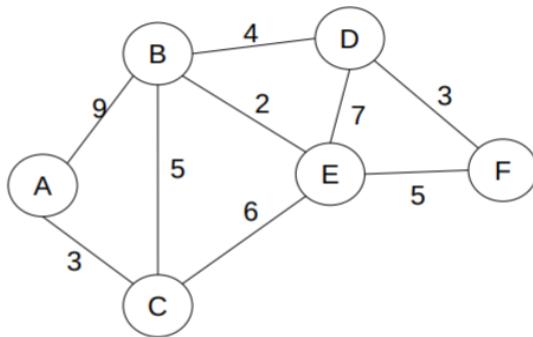
Fonte : A

Pilha

			mãos visitadas	visitadas	Pilha
			BCDEFGH	A	A
A	1	16	CDEFGH	A,B	AB
B	2	15	CDEFG	A,B,H	ABH
C	6	11	CDEF	A,B,H,G	ABHG
D	8	9	CDF	A,B,H,G,E	ABHGEC
E	5	12	DF	A,B,H,G,E,C	ABHGECF
F	7	10	D	A,B,H,G,E,C,F	ABHGECFD
G	4	13	∅	A,B,H,G,E,C,F,D	ABHGECF
H	3	14			A

1 - c.(3,0) Mostre o passo a passo do algoritmo de Dijkstra (mostrando todas as informações utilizadas) calculando as distâncias mínimas e os predecessores no caminho de custo mínimo. Mostre o caminho ou os caminhos mínimos obtidos e as escolhas gulosas realizadas.

Grafo G1:



Fonte: A

Matriz de adjacência (Pesos)

	A	B	C	D	E	F
A		9	3			
B	9		5	4	2	
C	3	5			6	
D		4			7	3
E		2	6	7		5
F				3	5	

Matriz de distâncias

	B	C	D	E	F
0	∞	∞	∞	∞	∞
1 ^a	9	3	∞	∞	∞
2 ^a	8	3	∞	9	∞
3 ^a	8	3	12	9	∞
4 ^a	8	3	12	9	14
5 ^a	8	3	12	9	14

Matriz de Predecessores

	B	C	D	E	F
0	-	-	-	-	-
1 ^a	A	A	-	-	-
2 ^a	C	A	-	C	-
3 ^a	C	A	B	C	-
4 ^a	C	A	B	C	E
5 ^a	C	A	B	C	E

Escolhas gulosas: A => C => B => E => D

- | | |
|-----------------------------------|--|
| 1^a u = A, v = B | $d[A][A] + P[A][B] < d[A][B] \Rightarrow 0 + 9 < \infty \Rightarrow 9 < \infty \Rightarrow d[A][B] = 9$, pred[A][B] = A |
| 1^a u = A, v = C | $d[A][A] + P[A][C] < d[A][C] \Rightarrow 0 + 3 < \infty \Rightarrow 3 < \infty \Rightarrow d[A][C] = 3$, pred[A][C] = A |
| 2^a u = C, v = B | $d[A][C] + P[C][B] < d[A][B] \Rightarrow 3 + 5 < 9 \Rightarrow 8 < 9 \Rightarrow d[A][B] = 8$, pred[A][B] = C |
| 2^a u = C, v = E | $d[A][C] + P[C][E] < d[A][E] \Rightarrow 3 + 6 < \infty \Rightarrow 9 < \infty \Rightarrow d[A][E] = 9$, pred[A][E] = C |
| 3^a u = B, v = D | $d[A][B] + P[B][D] < d[A][D] \Rightarrow 8 + 4 < \infty \Rightarrow 12 < \infty \Rightarrow d[A][D] = 12$, pred[A][D] = B |
| 3^a u = B, v = E | $d[A][B] + P[B][E] < d[A][E] \Rightarrow 8 + 2 < 9 \Rightarrow 10 < 9 \Rightarrow \text{Falso}$ |
| 4^a u = E, v = D | $d[A][E] + P[E][D] < d[A][D] \Rightarrow 9 + 7 < 12 \Rightarrow 16 < 12 \Rightarrow \text{Falso}$ |
| 4^a u = E, v = F | $d[A][E] + P[E][F] < d[A][F] \Rightarrow 9 + 5 < \infty \Rightarrow 14 < \infty \Rightarrow d[A][F] = 14$, pred[A][F] = E |
| 5^a u = D, v = F | $d[A][D] + P[D][F] < d[A][F] \Rightarrow 12 + 3 < 14 \Rightarrow 15 < 14 \Rightarrow \text{Falso}$ |

Caminhos:

$$C(A,B) = A,C,B$$

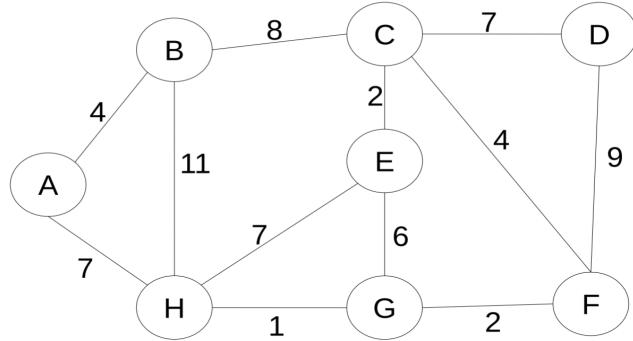
$$C(A,C) = A,C$$

$$C(A,D) = A,C,B,D$$

$$C(A,E) = A,C,E$$

$$C(A,F) = A,C,E,F$$

Grafo G2:



Fonte: A

Matriz de adjacência

	A	B	C	D	E	F	G	H
A	4							7
B	4	8						11
C		8	7	2	4			
D			7			9		
E				2			6	7
F				4	9			2
G					6	2		1
H	7	11			7		1	

Matriz de distâncias

	B	C	D	E	F	G	H
0	∞						
1 ^a	4	∞	∞	∞	∞	∞	7
2 ^a	4	12	∞	∞	∞	∞	7
3 ^a	4	12	∞	14	∞	8	7
4 ^a	4	12	∞	14	10	8	7
5 ^a	4	12	19	14	10	8	7
6 ^a	4	12	19	14	10	8	7
7 ^a	4	12	19	14	10	8	7

Matriz de Predecessores

	B	C	D	E	F	G	H
0	-	-	-	-	-	-	-
1 ^a	A	-	-	-	-	-	A
2 ^a	A	B	-	-	-	-	A
3 ^a	A	B	-	H	-	H	A
4 ^a	A	B	-	H	G	H	A
5 ^a	A	B	F	H	G	H	A
6 ^a	A	B	F	H	G	H	A
7 ^a	A	B	F	H	G	H	A

Escolhas gulosas: A => B => H => G => F => C => E

$$1^a \mathbf{u} = A, \mathbf{v} = B$$

$$d[A][A] + P[A][B] < d[A][B] \Rightarrow 0 + 4 < \infty \Rightarrow 4 < \infty \Rightarrow d[A][B] = 4, \text{pred}[A][B] = A$$

$$1^a \mathbf{u} = A, \mathbf{v} = H$$

$$d[A][A] + P[A][H] < d[A][H] \Rightarrow 0 + 7 < \infty \Rightarrow 7 < \infty \Rightarrow d[A][H] = 7, \text{pred}[A][H] = A$$

$$2^a \mathbf{u} = B, \mathbf{v} = H$$

$$d[A][B] + P[B][H] < d[A][H] \Rightarrow 4 + 11 < \infty \Rightarrow 7 < 15 \Rightarrow \text{FALSO}$$

$$2^a \mathbf{u} = B, \mathbf{v} = C$$

$$d[A][B] + P[B][C] < d[A][C] \Rightarrow 4 + 8 < \infty \Rightarrow 12 < \infty \Rightarrow d[A][C] = 12, \text{pred}[A][C] = B$$

$$3^a \mathbf{u} = H, \mathbf{v} = E$$

$$d[A][H] + P[H][E] < d[A][E] \Rightarrow 7 + 7 < \infty \Rightarrow 14 < \infty \Rightarrow d[A][E] = 14, \text{pred}[A][E] = H$$

$$3^a \mathbf{u} = H, \mathbf{v} = G$$

$$d[A][H] + P[H][G] < d[A][G] \Rightarrow 7 + 1 < \infty \Rightarrow 8 < \infty \Rightarrow d[A][G] = 8, \text{pred}[A][G] = H$$

$$4^a \mathbf{u} = G, \mathbf{v} = E$$

$$d[A][G] + P[G][E] < d[A][E] \Rightarrow 8 + 6 < 14 \Rightarrow 14 < 14 \Rightarrow \text{FALSO}$$

$$4^a \mathbf{u} = G, \mathbf{v} = F$$

$$d[A][G] + P[G][F] < d[A][F] \Rightarrow 8 + 2 < \infty \Rightarrow 10 < \infty \Rightarrow d[A][F] = 10, \text{pred}[A][F] = G$$

$$5^a \mathbf{u} = F, \mathbf{v} = C$$

$$d[A][F] + P[F][C] < d[A][C] \Rightarrow 10 + 4 < 12 \Rightarrow 14 < 12 \Rightarrow \text{FALSO}$$

$$5^a \mathbf{u} = F, \mathbf{v} = D$$

$$d[A][F] + P[F][D] < d[A][D] \Rightarrow 10 + 9 < \infty \Rightarrow 19 < \infty \Rightarrow d[A][D] = 19, \text{pred}[A][D] = F$$

$$6^a \mathbf{u} = C, \mathbf{v} = D$$

$$d[A][C] + P[C][D] < d[A][D] \Rightarrow 12 + 7 < 19 \Rightarrow 19 < 19 \Rightarrow \text{FALSO}$$

$$6^a \mathbf{u} = C, \mathbf{v} = E$$

$$d[A][C] + P[C][E] < d[A][E] \Rightarrow 12 + 2 < 14 \Rightarrow 14 < 14 \Rightarrow \text{FALSO}$$

$$7^a \mathbf{u} = E --$$

Caminhos:

$$C(A,B) = A,B$$

$$C(A,D) = A,H,G,F,D$$

$$C(A,C) = A,B,C$$

$$C(A,F) = A,H,G,F$$

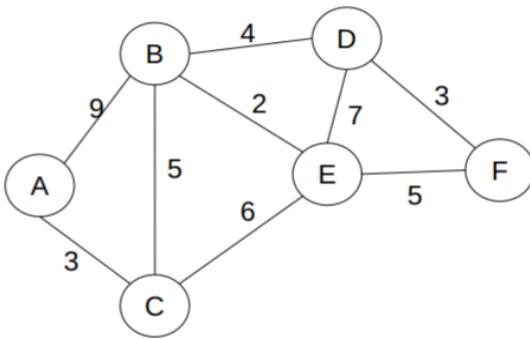
$$C(A,H) = A,H$$

$$C(A,E) = A,H,E$$

$$C(A,G) = A,H,G$$

1 - d.(3,0) Mostre o passo a passo do algoritmo de *Prim* para construir a árvore geradora mínima (AGM). Mostre a AGM obtida e todas as escolhas gulosas realizadas.

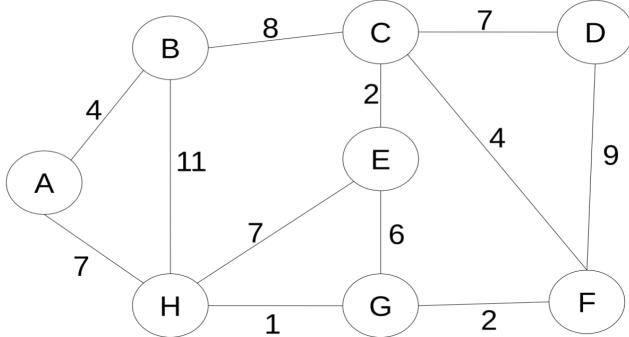
Grafo G1:



Fonte: A

CHAVE	B	C	D	E	F	Escolha Gulosa	AGM
Ini	∞	∞	∞	∞	∞	A	-
1 ^a	9	3	∞	∞	∞	C	(A,C)
2 ^a	5	3	∞	6	∞	B	(A,C); (C,B)
3 ^a	5	3	4	2	∞	E	(A,C); (C,B); (B,E)
4 ^a	5	3	4	2	5	D	(A,C); (C,B); (B,E); (B,D)
5 ^a	5	3	4	2	3	F	(A,C); (C,B); (B,E); (B,D); (D,F)

Grafo G2:

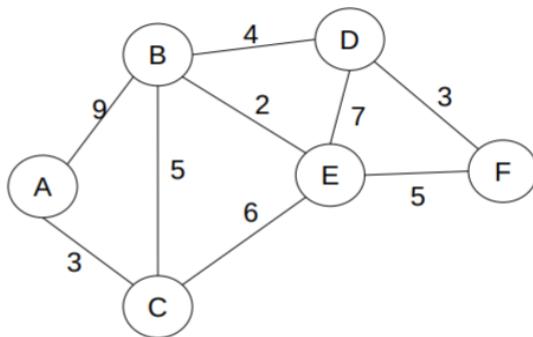


Fonte: A

CHAVE	B	C	D	E	F	G	H	Escolha Gulosa	AGM
Ini	∞	A	-						
1 ^a	4	∞	∞	∞	∞	∞	7	B	(A,B)
2 ^a	4	8	∞	∞	∞	∞	7	C	(A,B); (B,C)
3 ^a	4	8	7	2	4	∞	7	E	(A,B); (B,C); (C,E)
4 ^a	4	8	7	2	4	6	7	F	(A,B); (B,C); (C,E); (C,F)
5 ^a	4	8	7	2	4	2	7	G	(A,B); (B,C); (C,E); (C,F); (F,G)
6 ^a	4	8	7	2	4	2	1	H	(A,B); (B,C); (C,E); (C,F); (F,G); (G,H)
7 ^a	4	8	7	2	4	2	1	D	(A,B); (B,C); (C,E); (C,F); (F,G); (G,H); (C,D)

1 - e.(3,0) Mostre o passo a passo do algoritmo de Kruskal para construir a árvore geradora mínima (AGM). Mostre a AGM obtida e todas as escolhas gulosas realizadas.

Grafo G1:

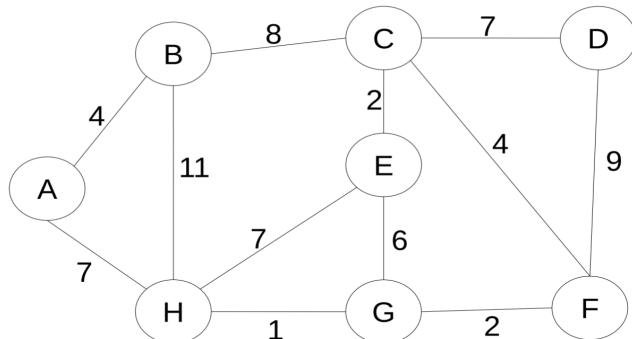


Fonte: A

$$A = (B,E); (A,C); (D,F); (B,D); (B,C); (E,F); (C,E); (E,D); (A,B)$$

Floresta	ESCOLHA GULOSA	AGM
{A}, {B}, {C}, {D}, {E}, {F}	(B,E)	(B,E)
{A,C}, {B,E}, {D}, {F}	(A,C)	(B,E); (A,C)
{A,C}, {B,E}, {D,F}	(D,F)	(B,E); (A,C); (D,F)
{A,C}, {B,E,D,F}	(B,D)	(B,E); (A,C); (D,F); (B,D)
{A,C,B,E,D,F}	(B,C)	(B,E); (A,C); (D,F); (B,D); (B,C)
FINDSET FALSE P/ TODOS OS PRÓXIMOS	-	AGM EM ORDEM ALFABÉTICA: (A,C); (B,C); (B,D); (B,E); (D,F)

Grafo G2:



Fonte: A

$$A = (G,H); (C,E); (F,G); (A,B); (C,F); (E,G); (A,H); (C,D); (E,H); (B,C); (D,F); (B,H)$$

Floresta	ESCOLHA GULOSA	AGM
{A}, {B}, {C}, {D}, {E}, {F}, {G}, {H}	(G,H)	(G,H)
{A}, {B}, {C,E}, {D}, {F}, {G,H}	(C,E)	(G,H); (C,E)
{A}, {B}, {C,E}, {D}, {F,G,H}	(F,G)	(G,H); (C,E); (F,G)

{A,B}, {C,E}, {D}, {F,G,H}	(A,B)	(G,H); (C,E); (F,G); (A,B)
{A,B}, {C,E,F,G,H}, {D}	(C,F)	(G,H); (C,E); (F,G); (A,B); (C,F)
FINDSET FALSE	(E,G)	(G,H); (C,E); (F,G); (A,B); (C,F)
{A,B,C,E,F,G,H}, {D}	(A,H)	(G,H); (C,E); (F,G); (A,B); (C,F); (A,H)
{A,B,C,E,F,G,H,D}	(C,D)	(G,H); (C,E); (F,G); (A,B); (C,F); (A,H); (C,D)
FINDSET FALSE P/ TODOS OS PRÓXIMOS	-	AGM EM ORDEM ALFABÉTICA: (A,B); (A,H); (C,D); (C,E); (C,F); (F,G); (G,H);

2.(1,0) Explique como funcionam as heurísticas dos algoritmos BM, BMH e BMHS. Faça uma comparação entre as heurísticas destacando as diferenças.

O algoritmo BM realiza a busca por padrões verificando o padrão de trás para frente e buscando conflitos entre o texto buscado e o padrão, ao ser encontrado o conflito o algoritmo avança posições referentes a um cálculo baseado na posição anterior, na posição do conflito e na última ocorrência do caractere em conflito. O algoritmo realiza um pré-processamento, determinando dentro do padrão as últimas ocorrências de cada caractere e o último elemento recebe o valor de -1, caso não tenha outra ocorrência anterior. O algoritmo BMH realiza esta mesma busca por padrões, tendo também no pré-processamento, o último elemento recebendo como valor da última ocorrência -1, caso não haja outra ocorrência anterior. Porém, diferentemente do BM, o algoritmo BMH avança posições com base no valor de M menos 1, permitindo assim avançar mais posições que o algoritmo BM. O algoritmo BMHS, em comparação aos dois algoritmos anteriores, atribui, no momento do pré-processamento, o valor real do último elemento nos valores de última ocorrência. Outra diferença se refere ao avanço de posições, sendo similar com o algoritmo BMH, contudo avançando com base em M, permitindo avançar mais posições e um total de M + 1 posições.

3.(1,0) Mostre o passo a passo do algoritmo de Boyer-Moore (BM) para o texto e padrão dados abaixo.

a) T:

X C B A B X C B A A X B C B A B X

P:

B C B A

Bölger - Moore. (BM)

a) $XCB A BX CBA AXB CBABX =$

$\begin{array}{c} B \\ C \\ B \\ A \end{array}$

$\begin{array}{c} B \\ C \\ B \\ A \end{array}$

$\begin{array}{c} B \\ C \\ BA \end{array}$

$\begin{array}{c} B \\ C \\ B \\ A \end{array}$

$\boxed{B \\ C \\ B \\ A}$

$\begin{array}{c} B \\ C \\ B \\ A \end{array}$

$\begin{array}{c} B \\ C \\ BA \end{array}$

$\begin{array}{c} B \\ C \\ BA \end{array}$

Pós-processamento:

$P = BCB A (m=4)$

$\text{lastOcc}[B] = 2; \quad \text{lastOcc}[A] = -1;$

$\text{lastOcc}[C] = 1;$

BM: Atualização de IO:

$$IO = 0$$

$$IO = 0 + 0 - (-1) = 1$$

$$IO = 1 + 3 - 2 = 2$$

$$IO = 2 + 3 - (-1) = 6$$

$$IO = 6 + 2 - (-1) = 9$$

$$IO = 9 + 3 - 1 = 11$$

→ Achou o padrão na posição 11

$$IO = 12$$

$$IO = 12 + 3 - 2 = 13$$

$$IO = 13 + 3 - (-1) = 17$$

$$IO = (17) + 1 = 18$$

$$IO = 18 + P + PR = 18 + 0 = 18$$

$$IO = 18 + P + PR = 18 + 0 = 18$$

b)

T:

GCATCGCAGAGAGTATAACGTACG

P:

GCAGAGAG

b) GCATCGCAGAGAGTATAACGTACG

GCAGAGAG

GCAGAGAG

GCAGAGAG

GCAGAGAG

GCAGAGAG

GCAGAGAG

GCAGAG...

Pre'processamento: $P = \text{GCAGAGAG}$ ($m=8$)

lastOcc[G] = 5; lastOcc[C] = 1; lastOcc[A] = 6

B_M: Atualização de ID:

$$i_0 = 0$$

$$i_0 = 0 + 7 - 6 = 1$$

$$i_0 = 1 + 5 - 7 = 5$$

→ Padrão encontrado na posição 5.

$$i_0 = 6$$

$$i_0 = 6 + 7 - (-1) = 14$$

$$i_0 = 14 + 7 - 6 = 15$$

$$i_0 = 15 + 7 - 1 = 21$$

4.(1.0) Mostre o passo a passo do algoritmo de Boyer-Moore-Horspool (BMH) para o texto e padrão dados abaixo.

a)

T:

X C B A B X C B A A X B C B A B X

P:

B C B A

a)

X C B A B X C B A A X B C B A B X

B C B A

B C B A

B C B A

B C B A

B C B A

B C B A

Pré-processamento:

P = B C B A (m = 4)

LastOcc[B] = 2 ; LastOcc[C] = 1 ; LastOcc[A] = -1

BMH : Atualização de i0 :

i0 = 0

i0 = i0 + m - 1 - LastOcc [A] = 0 + 3 - (-1) = 4

i0 = i0 + m - 1 - LastOcc [B] = 4 + 3 - 2 = 5

i0 = i0 + m - 1 - LastOcc [A] = 5 + 3 - (-1) = 9

i0 = i0 + m - 1 - LastOcc [C] = 9 + 3 - 1 = 11

→ Padrão encontrado posição 11.

i0 = i0 + m - 1 - LastOcc [A] = 11 + 3 - (-1) = 15

b)

T:

GCATCGCAGAGAGTATAACGTACG

P:

GCAGAGAG

b)

GCATCGCAGAGAGTATAACGTACG

GCAGAGAG

GCAGAGAG

GCAGAGAG

GCAGAGAG

GCAGAGAG

GCAGAGAG

GCAGAGAG

GCAGAGAG

Pré-processamento:

P = GCAGAGAG ($m = 8$)

LastOcc[G] = 5 ; LastOcc[C] = 1 ; LastOcc[A] = 6

BMH : Atualização de i0 :

i0 = 0

i0 = i0 + m - 1 - LastOcc [A] = 0 + 7 - 6 = 1

i0 = i0 + m - 1 - LastOcc [G] = 1 + 7 - 5 = 3

i0 = i0 + m - 1 - LastOcc [G] = 3 + 7 - 5 = 5

→ Padrão encontrado na posição 5

i0 = i0 + m - 1 - LastOcc [G] = 5 + 7 - 5 = 7

i0 = i0 + m - 1 - LastOcc [A] = 7 + 7 - 6 = 8

i0 = i0 + m - 1 - LastOcc [T] = 8 + 7 - (-1) = 16

i0 = i0 + m - 1 - LastOcc [G] = 16 + 7 - 5 = 18

5.(1.0) Mostre o passo a passo do algoritmo de Boyer-Moore-Horspool-Sunday (BMHS) para o texto e padrão dados abaixo.

a)

T:

X C B A B X C B A A X B C B A B X

P:

B C B A

a)

X C B A B X C B A A X B C B A B X

B C B A

B C B A

B C B A

B C B A

B C B A

Pré-processamento:

$P = B C B A \ (m = 4)$

$\text{LastOcc}[B] = 2 ; \text{LastOcc}[C] = 1 ; \text{LastOcc}[A] = 3$

BMHS : Atualização de i_0 :

$i_0 = 0$

$i_0 = i_0 + m - \text{LastOcc}[B] = 0 + 4 - 2 = 2$

$i_0 = i_0 + m - \text{LastOcc}[C] = 2 + 4 - 1 = 5$

$i_0 = i_0 + m - \text{LastOcc}[A] = 5 + 4 - 3 = 6$

$i_0 = i_0 + m - \text{LastOcc}[X] = 6 + 4 - (-1) = 11$

→ Padrão encontrado na posição 11

$i_0 = i_0 + m - \text{LastOcc}[B] = 11 + 4 - 2 = 13$

b)

T:

G C A T C G C A G A G A G T A T A C A G T A C G

P:

G C A G A G A G

b)

G C A T C G C A G A G A G T A T A C A G T A C G

G C A G A G A G

Pré-processamento:

$P = G C A G A G A G \quad (m = 8)$

$\text{LastOcc}[G] = 7 ; \text{LastOcc}[C] = 1 ; \text{LastOcc}[A] = 6$

BMHS : Atualização de i_0 :

$i_0 = 0$

$i_0 = i_0 + m - \text{LastOcc}[G] = 0 + 8 - 7 = 1$

$i_0 = i_0 + m - \text{LastOcc}[A] = 1 + 8 - 6 = 3$

$i_0 = i_0 + m - \text{LastOcc}[A] = 3 + 8 - 6 = 5$

→ Padrão encontrado na posição 5

$i_0 = i_0 + m - \text{LastOcc}[T] = 5 + 8 - (-1) = 14$

$i_0 = i_0 + m - \text{LastOcc}[C] = 14 + 8 - 1 = 21$

6.(1.0) Mostre o passo a passo do algoritmo de Huffman para criar a árvore de Huffman e a tabela de codificação para o texto abaixo. "Acreditamos que a situação do Brasil pode melhorar"

1 / 1

Acreditamos que a situação do Brasil pode melhorar

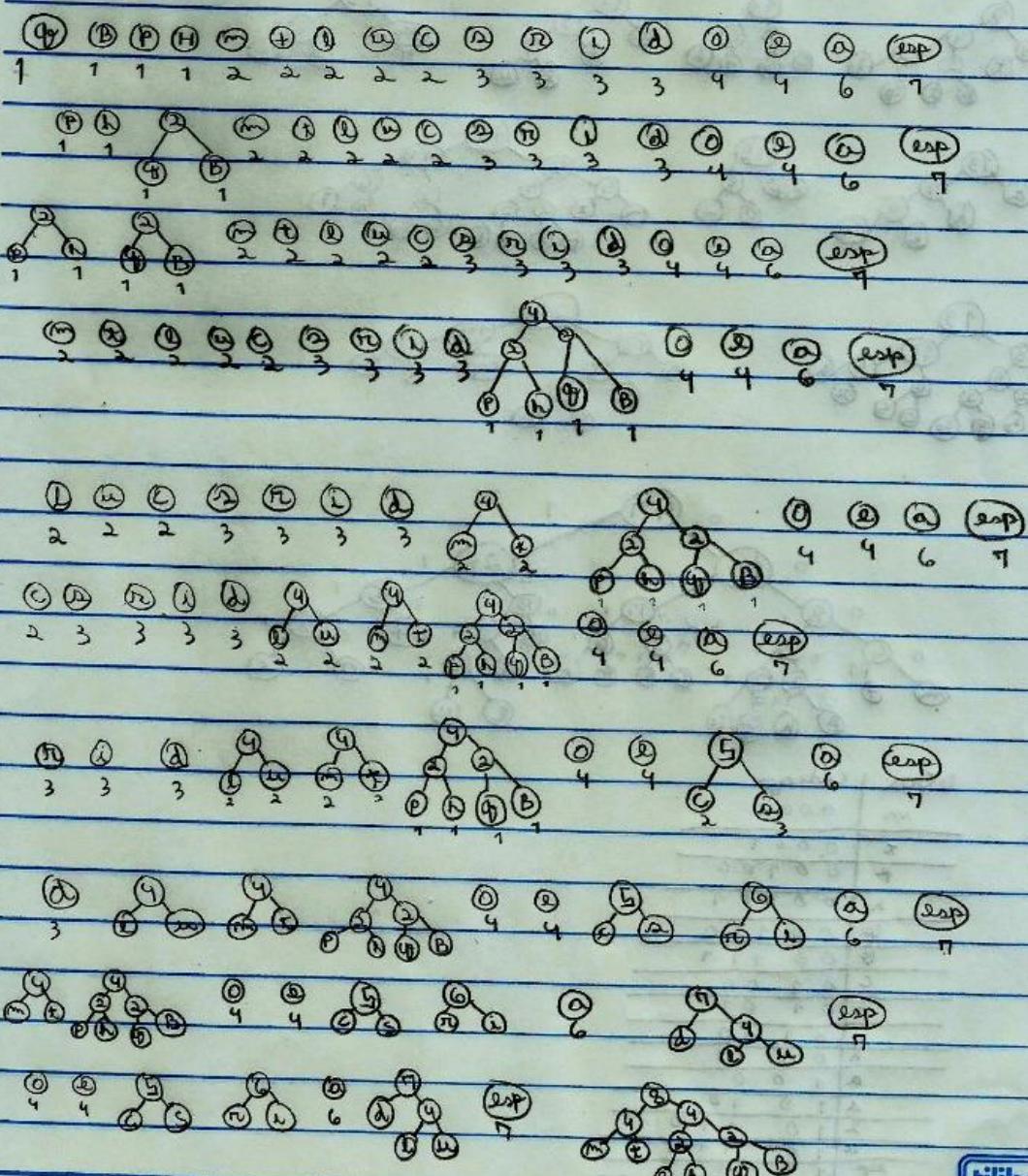
$$\text{freq}(a) = 1;$$

$$\text{freq}(B) = 1; \text{ freq}(P) = 1; \text{ freq}(H) = 1; \text{ freq}(m) = 2; \text{ freq}(+) = 2$$

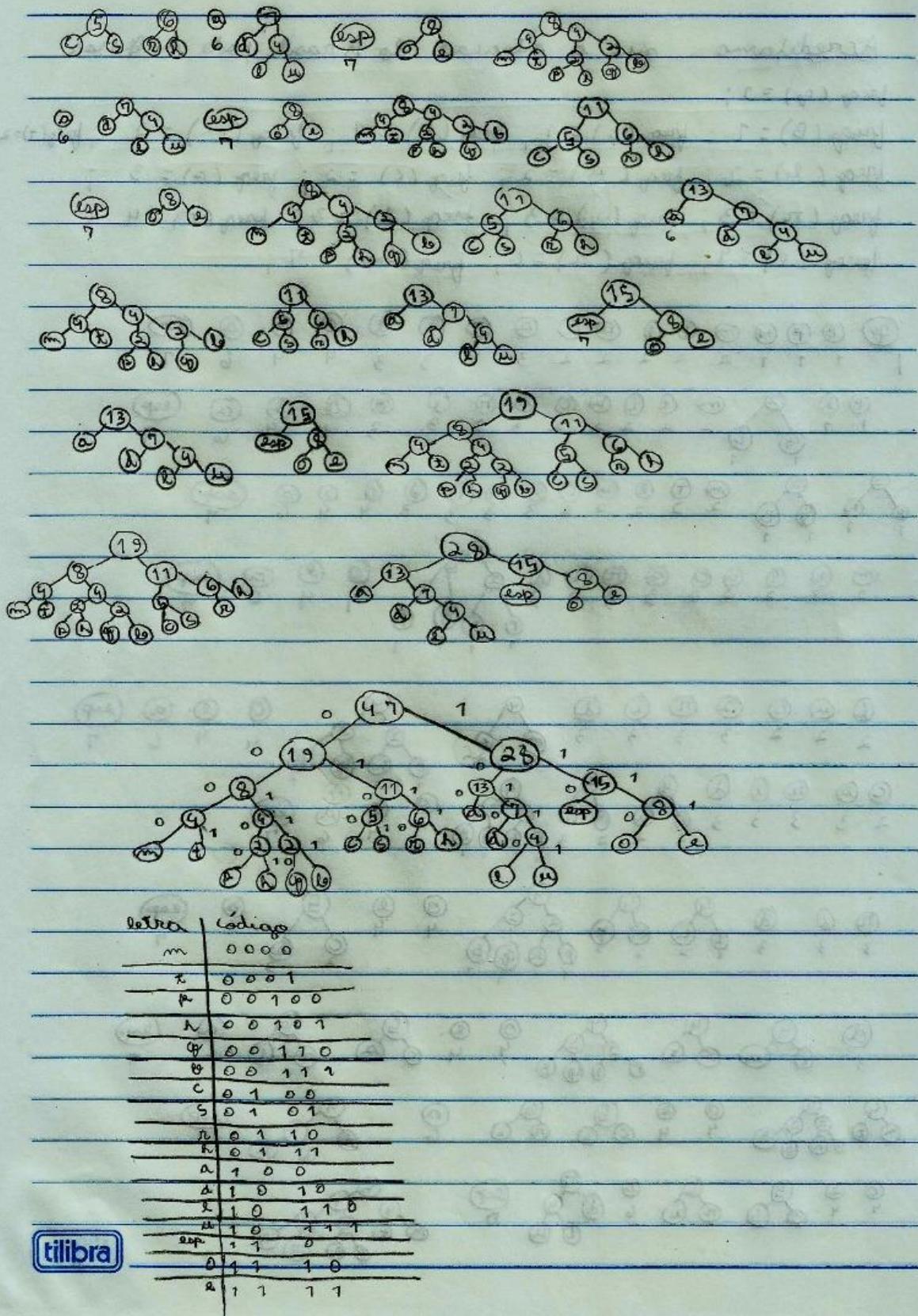
$$\text{freq}(l) = 2; \text{ freq}(u) = 2; \text{ freq}(c) = 2; \text{ freq}(z) = 3;$$

$$\text{freq}(n) = 3; \text{ freq}(i) = 3; \text{ freq}(d) = 3; \text{ freq}(o) = 4;$$

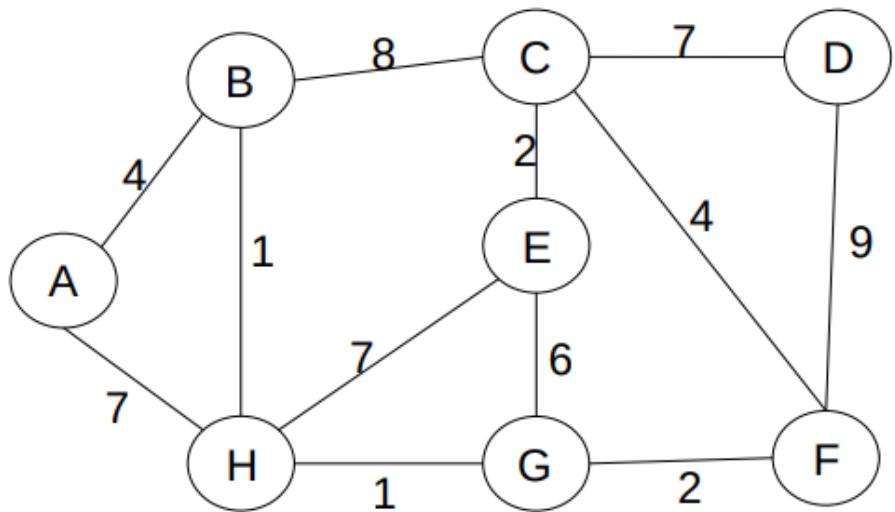
$$\text{freq}(e) = 9; \text{ freq}(a) = 6; \text{ freq}(esp) = 7;$$



X X



7.(1,0) Com base no grafo abaixo, faça o que se pede:



a) Represente o grafo utilizando listas de adjacência.

A	-> (B,4) -> (H,7)
B	-> (A,4) -> (H,1) -> (C, 8)
C	-> (B,8) -> (E,2) -> (D, 7) -> (F,4)
D	-> (F,9) -> (C,7)
E	-> (C,2) -> (H,7) -> (G, 6)
F	-> (C,4) -> (D,9) -> (G, 2)
G	-> (H,1) -> (E,6) -> (F, 2)
H	-> (B,1) -> (E,7) -> (G, 1) -> (A,7)

b) Represente o grafo utilizando a representação por matrizes esparsas.

Origem	A	A	B	B	C	C	C	D	E	E	F	G
Destino	B	H	C	H	D	E	F	F	H	G	G	H
Peso	4	7	8	1	7	2	4	9	7	6	2	1