João Pedro Serezuelo Amancio

João Vitor Fagundes de lima

1) Faça as seguintes definições

a. Defina o que e um subgrafo.

> G1 = (V1, E1) é subgrafo de G2 = (V2, E2) se e somente se: – V1 é subconjunto de V2; e, – E1 é subconjunto de E2. Subgrafos podem ser obtidos através da remoção de arestas e vértices.

b. Defina o que e um grafo bipartido.

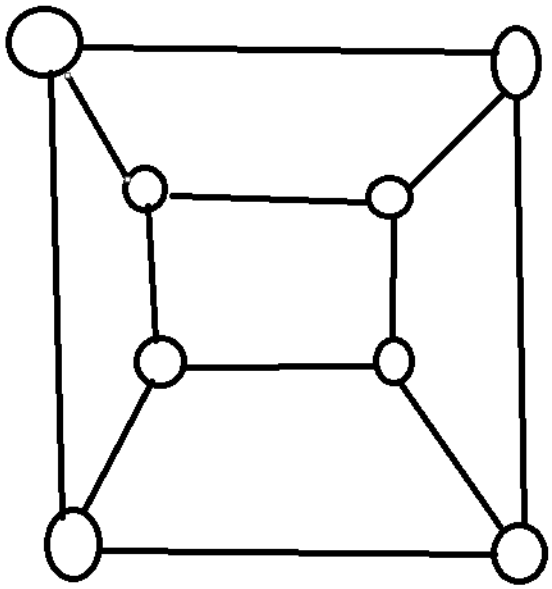
> Um grafo é dito ser bipartido quando seu conjunto de vértices V puder ser particionado em dois subconjuntos V1 e V2, tais que toda aresta de G une um vértice de V1 a outro de V2.

c. Defina o que e um grafo conexo. E um desconexo?

> Um grafo conexo é aquele que para todo par de vértices i e j de G existe pelo menos um caminho entre i e j .

d. O que são grafos isomorfos? Desenhe um exemplo.

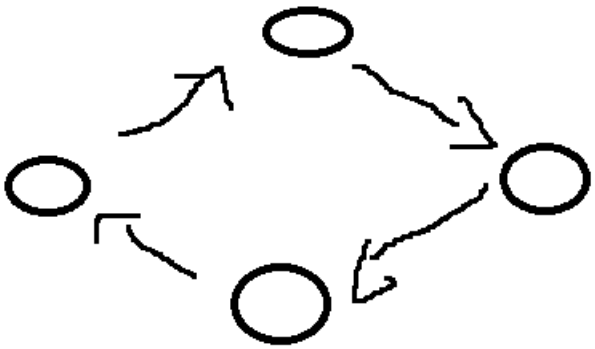
> Dois grafos G1(V1,E1) e G2(V2,E2) são ditos isomorfos entre si se: existe uma correspondência entre os seus vértices e arestas de tal maneira que a relação de incidência seja preservada.



e. Defina o que e um grafo Hamiltoniano. Desenhe um exemplo

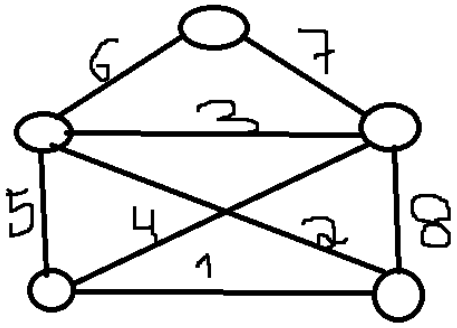
> Um grafo G é dito ser hamiltoniano se existe um
>
> ciclo em G que contenha todos os seus vértices,

sendo que cada vértice só aparece uma vez no ciclo.



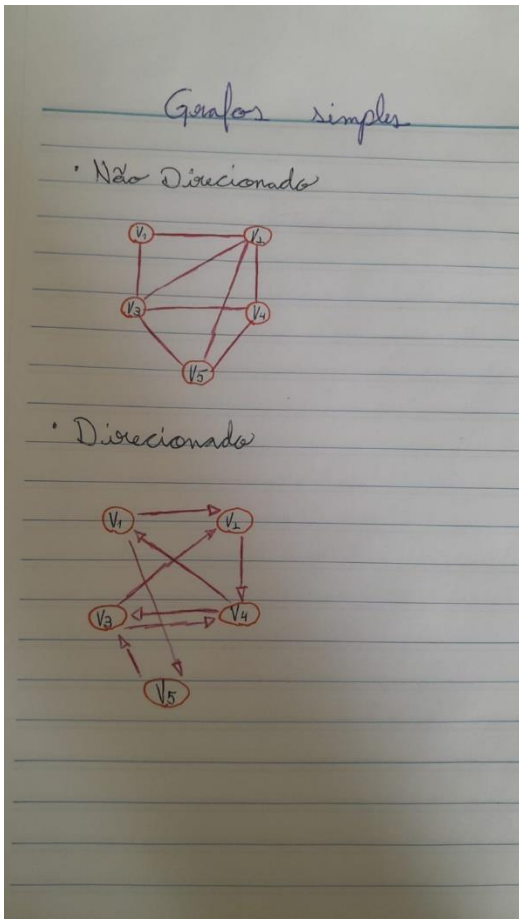
f. Defina o que e um grafo Euleriano. Desenhe um exemplo

Um grafo G é dito ser euleriano se existe um ciclo em G que contenha todos as suas arestas, sendo que cada aresta só aparece uma vez no ciclo. Este ciclo é chamado de ciclo euleriano. O ciclo visita cada aresta apenas uma vez.




2) Faça o que se pede:

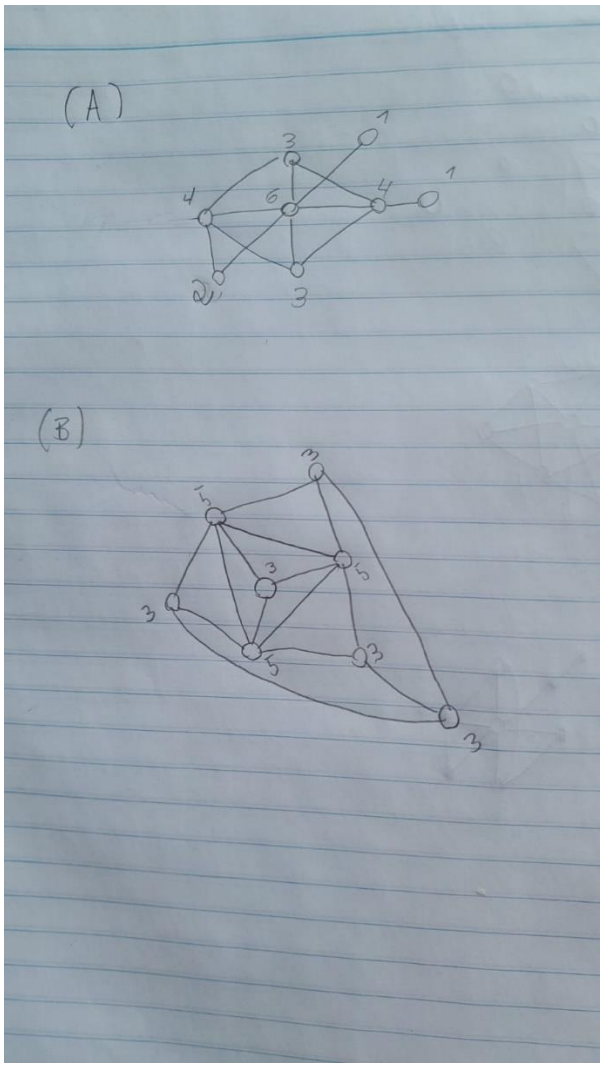
a. Construa um exemplo de grafo simples direcionado e um não direcionado.



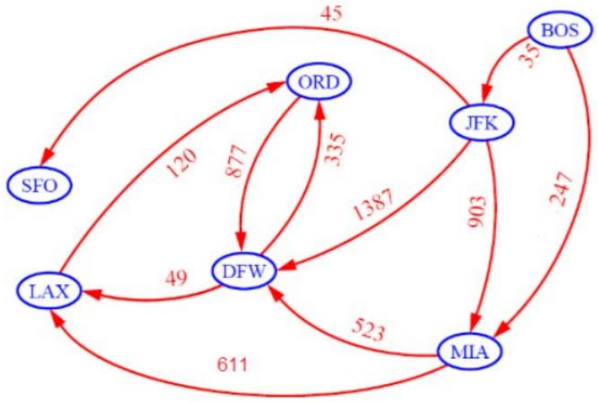

b. Construa um grafo simples conexo, com as seguintes sequências de graus.

(a) (1, 1, 2, 3, 3, 4, 4, 6)

(b) (3, 3, 3, 3, 3, 5, 5, 5)




3) Dado o seguinte grafo:




a. Quantas arestas esse grafo possui?

11

b. Quantos vértices esse grafo possui?

7

c. É possível ir da posição “DFW” para a “JFK”?

não

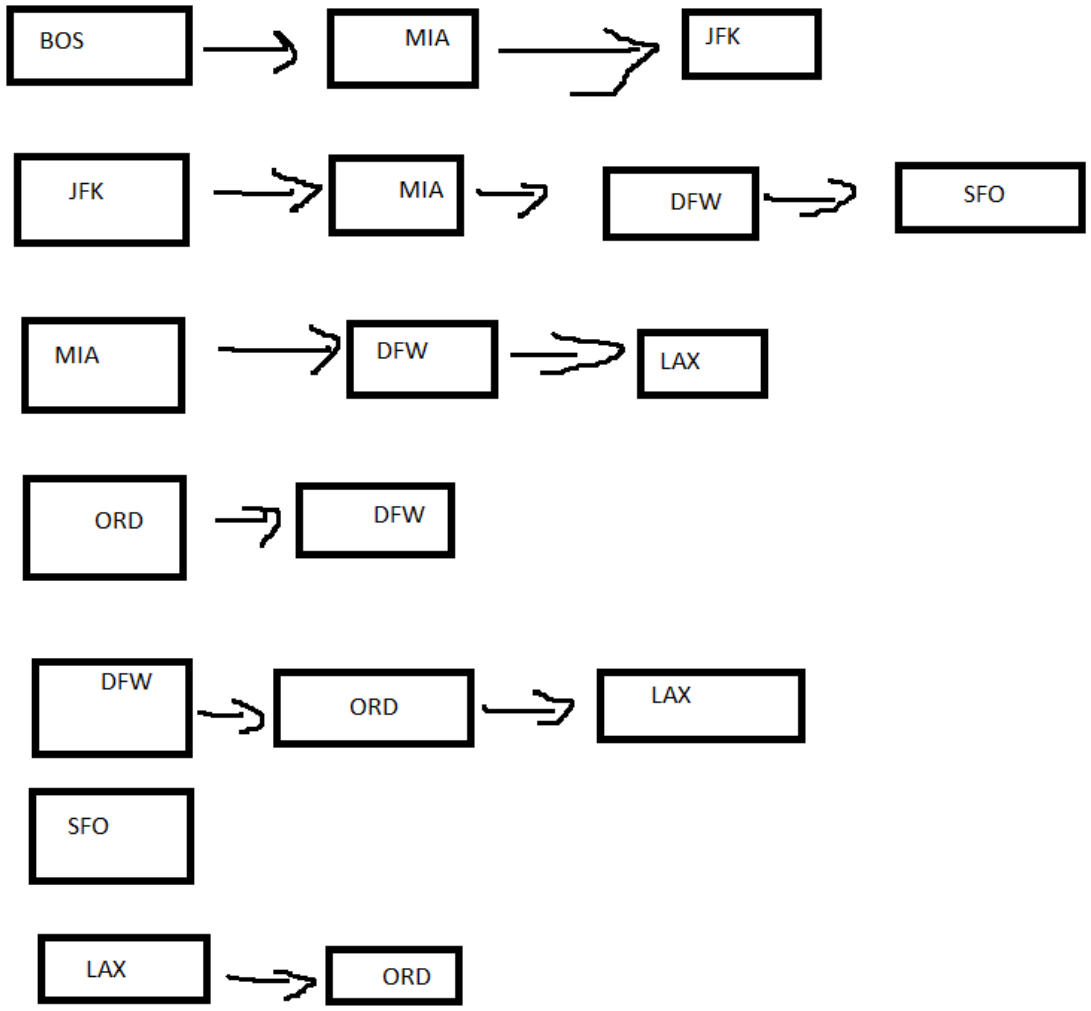
d. Qual é o caminho mais curto para ir de uma posição “MIA” para “LAX”?

o caminho 523+49

e. Ilustre como é a representação desse grafo através de uma matriz de adjacências.

| | BOS | JFK | MIA | ORD | DFW | SFO | LAX |
|---|---|---|---|---|---|---|---|
| BOS | 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| JFK | 0 | 0 | 1 | 0 | 1 | 1 | 0 |
| MIA | 0 | 0 | 0 | 0 | 1 | 0 | 1 |
| ORD | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| DFW | 0 | 0 | 0 | 1 | 0 | 0 | 1 |
| SFO | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| LAX | 0 | 0 | 0 | 1 | 0 | 0 | 0 |

f. Ilustre como é a representação desse grafo através de uma lista de adjacências.




g. Aplique o algoritmo de PRIM e descubra a Árvore Geradora Mínima

4) Utilizando o arquivo “ProjGrafo” e o seu respectivo código:

a. Detalhe o funcionamento da função Grafo* cria_Grafo(int nro_vertices, int grau_max, int eh_ponderado). Utilize trechos do código para ilustrar sua resposta.

```c
Grafo* cria_Grafo(int nro_vertices, int grau_max, int eh_ponderado){
    Grafo *gr;//cria um ponteiro para o tipo Grafo
    gr = (Grafo*) malloc(sizeof(struct grafo));//aloca memoria para o tipo Grafo
    if(gr != NULL){//se a alocacaoo deu certo
        int i;//variavel auxiliar
        gr->nro_vertices = nro_vertices;//atribui o valor de nro_vertices para o campo nro_vertices do tipo Grafo
        gr->grau_max = grau_max;//atribui o valor de grau_max para o campo grau_max do tipo Grafo
        gr->eh_ponderado = (eh_ponderado != 0)?1:0;//atribui o valor de eh_ponderado para o campo eh_ponderado do tipo Grafo
        gr->grau = (int*) calloc(nro_vertices,sizeof(int));//aloca memoria para o campo grau do tipo Grafo

        gr->arestas = (int**) malloc(nro_vertices * sizeof(int*));//aloca memoria para o campo arestas do tipo Grafo
        for(i=0; i<nro_vertices; i++)//percorre o vetor de ponteiros
            gr->arestas[i] = (int*) malloc(grau_max * sizeof(int));//aloca memoria para cada ponteiro

        if(gr->eh_ponderado){//se o grafo for ponderado
            gr->pesos = (float**) malloc(nro_vertices * sizeof(float*));//aloca memoria para o campo pesos do tipo Grafo
            for(i=0; i<nro_vertices; i++)//percorre o vetor de ponteiros
                gr->pesos[i] = (float*) malloc(grau_max * sizeof(float));//aloca memoria para cada ponteiro
        }

    }
    return gr;//retorna o ponteiro para o tipo Grafo
}
```

b. Detalhe o funcionamento da função libera_Grafo(Grafo* gr). Utilize trechos do código para ilustrar sua resposta.

```c
void libera_Grafo(Grafo* gr){
    if(gr != NULL){//se o ponteiro para o tipo Grafo for diferente de NULL
        int i;//variavel auxiliar
        for(i=0; i<gr->nro_vertices; i++)//percorre o vetor de ponteiros
            free(gr->arestas[i]);//libera a memoria de cada ponteiro
        free(gr->arestas);//libera a memoria do vetor de ponteiros

        if(gr->eh_ponderado){//se o grafo for ponderado
            for(i=0; i<gr->nro_vertices; i++)//percorre o vetor de ponteiros
                free(gr->pesos[i]);//libera a memoria de cada ponteiro
            free(gr->pesos);//libera a memoria do vetor de ponteiros
        }
        free(gr->grau);//libera a memoria do campo grau do tipo Grafo
        free(gr);//libera a memoria do tipo Grafo
    }
}
```

c. Detalhe o funcionamento da função insereAresta(Grafo* gr, int orig, int dest, int eh_digrafo, float peso). Utilize trechos do código para ilustrar sua resposta.

```
}
    return 1;//retorna 1
    insereArestaGrafo(gr,dest,orig,1,peso);//insere a aresta no vertice de destino
    if(eh_digrafo == 0)//se o grafo nao for direcionado
    gr->grau[orig]++;//incrementa o grau do vertice de origem
    gr->pesos[orig][gr->grau[orig]] = peso;//insere o peso no vetor de pesos
    if(gr->eh_ponderado)//se o grafo for ponderado
    gr->arestas[orig][gr->grau[orig]] = dest;//insere o vertice de destino no vetor de arestas
        return 0;//retorna 0
    if(dest < 0 || dest >= gr->nro_vertices)//se o vertice de destino for menor que 0 ou maior ou igual ao numero de vertices
        return 0;//retorna 0
    if(orig < 0 || orig >= gr->nro_vertices)//se o vertice de origem for menor que 0 ou maior ou igual ao numero de vertices
        return 0;//retorna 0
    if(gr == NULL)//se o ponteiro para o tipo Grafo for igual a NULL
int insereArestaGrafo(Grafo* gr, int orig, int dest, int eh_digrafo, float peso){
```

d. Detalhe o funcionamento da função removeAresta(Grafo* gr, int orig, int dest, int eh_digrafo). Utilize trechos do código para ilustrar sua resposta.

```
int removeAresta(Grafo* gr, int orig, int dest, int eh_digrafo){
    if(gr == NULL)//se o ponteiro para o tipo Grafo for igual a NULL
        return 0;//retorna 0
    if(orig < 0 || orig >= gr->nro_vertices)//se o vertice de origem for menor que 0 ou maior ou igual ao numero de vertices
        return 0;//retorna 0
    if(dest < 0 || dest >= gr->nro_vertices)//se o vertice de destino for menor que 0 ou maior ou igual ao numero de vertices
        return 0;//retorna 0

    int i = 0;//variavel auxiliar
    while(i<gr->grau[orig] && gr->arestas[orig][i] != dest)//percorre o vetor de arestas
        i++;//incrementa a variavel auxiliar
    if(i == gr->grau[orig])//elemento nao encontrado
        return 0;//retorna 0
    gr->grau[orig]--;//decrementa o grau do vertice de origem
    gr->arestas[orig][i] = gr->arestas[orig][gr->grau[orig]];//remove o vertice de destino do vetor de arestas
    if(gr->eh_ponderado)//se o grafo for ponderado
        gr->pesos[orig][i] = gr->pesos[orig][gr->grau[orig]];////remove o peso do vetor de pesos
    if(eh_digrafo == 0)//se o grafo nao for direcionado
        removeAresta(gr,dest,orig,1);//remove a aresta no vertice de destino
    return 1;//retorna 1
}
```