Ministério da Educação
Universidade Tecnológica Federal do Paraná
Câmpus Santa Helena
Disciplina: Algoritmos e Estruturas de Dados III

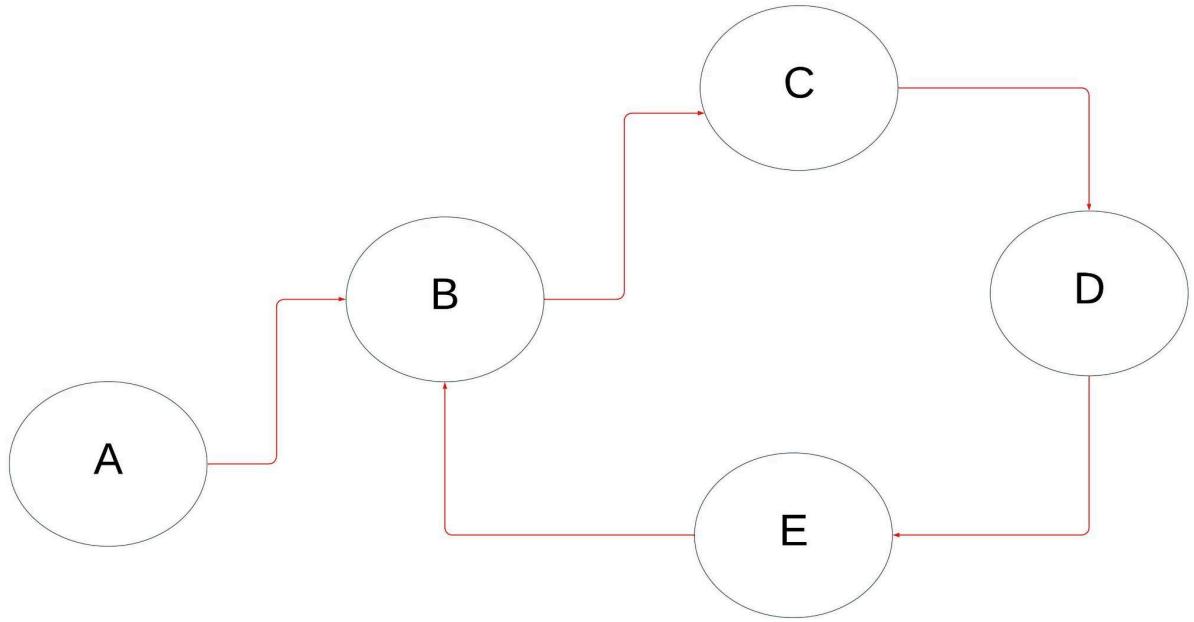
PDF Contendo as Respostas descritivas Sobre a Lista de Grafos

Aluno: José Carlos Seben de Souza Leite Ra:2651130
Aluno: Caio Macedo Lima da Cruz Ra:2651378

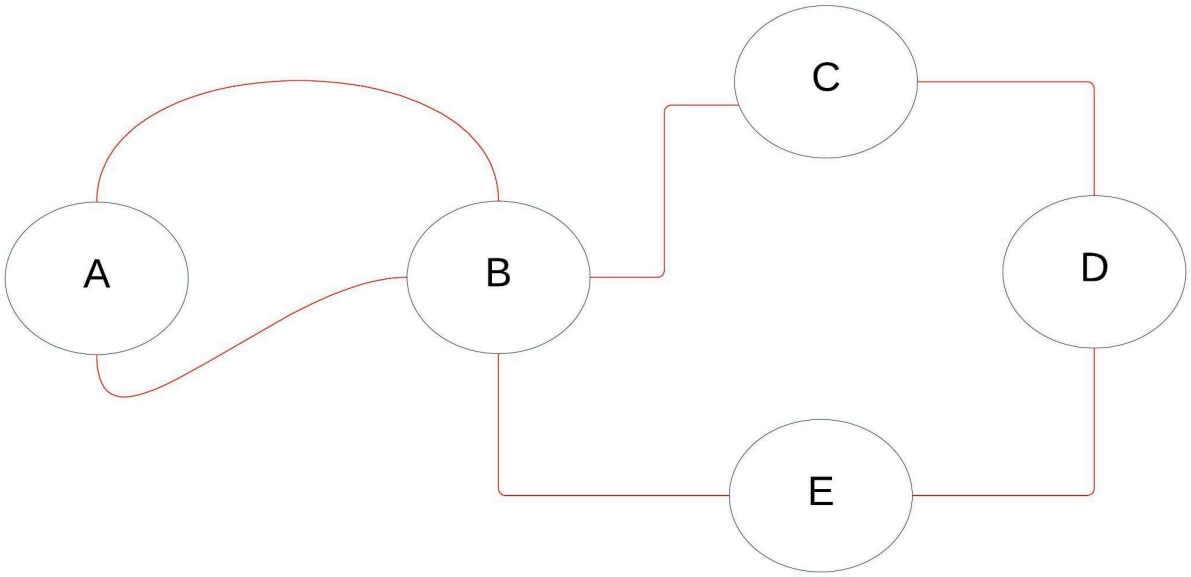
Atividade 1- Representações visuais de grafos:

1 - A:

Grafo Direcionado:

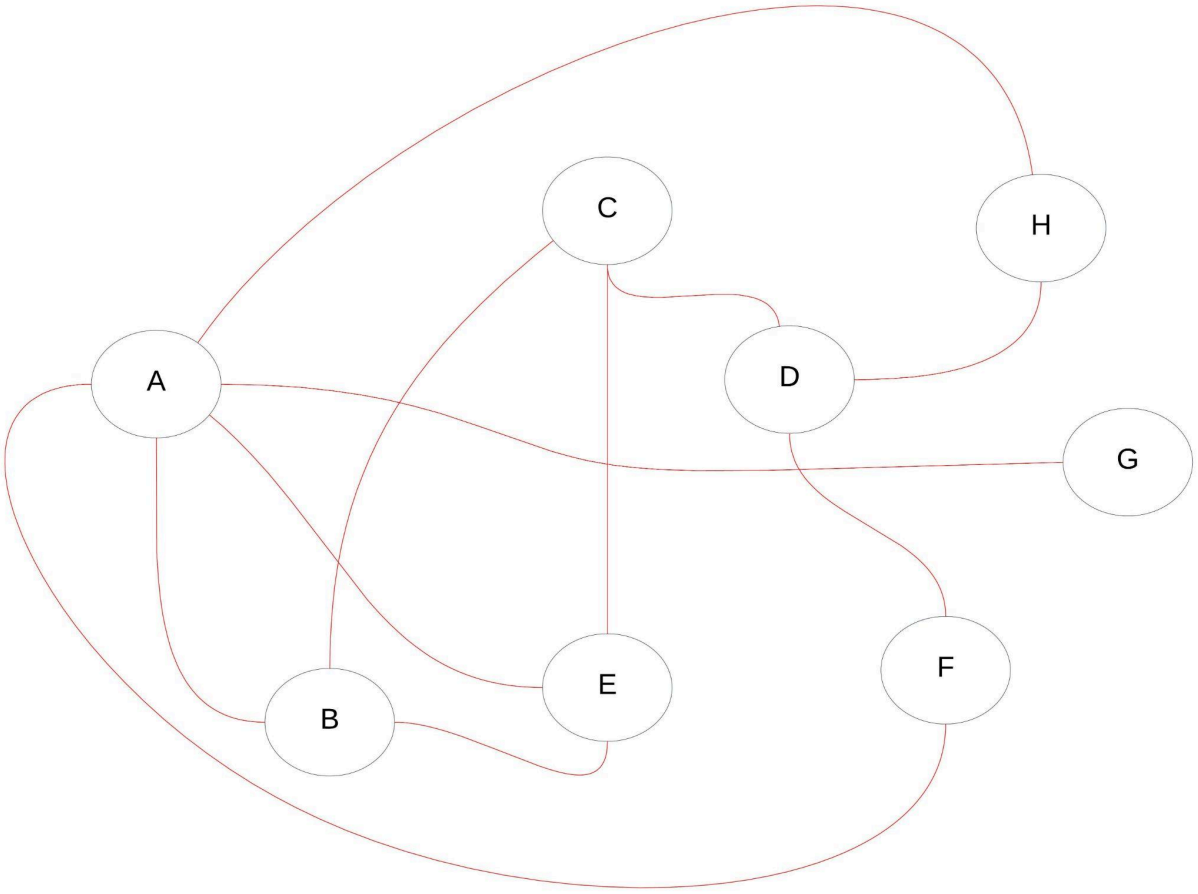


Grafo Não Direcionado:

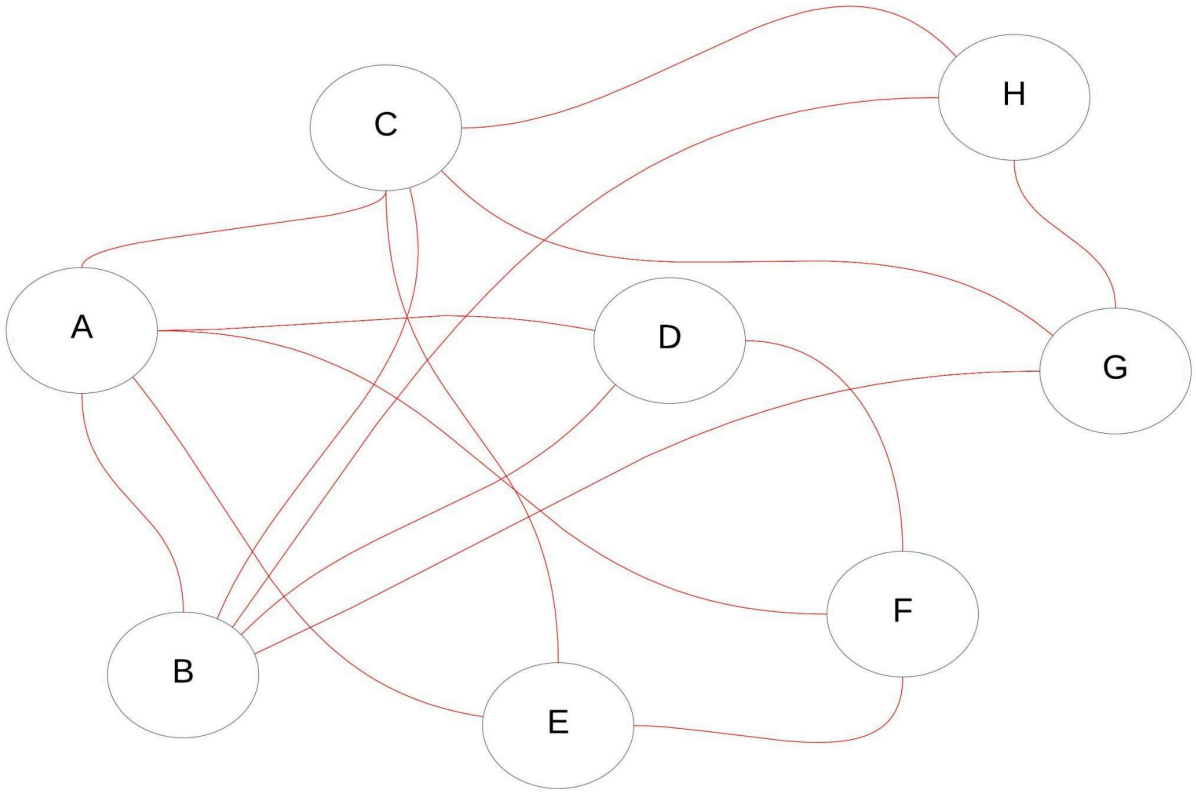


1 - B:

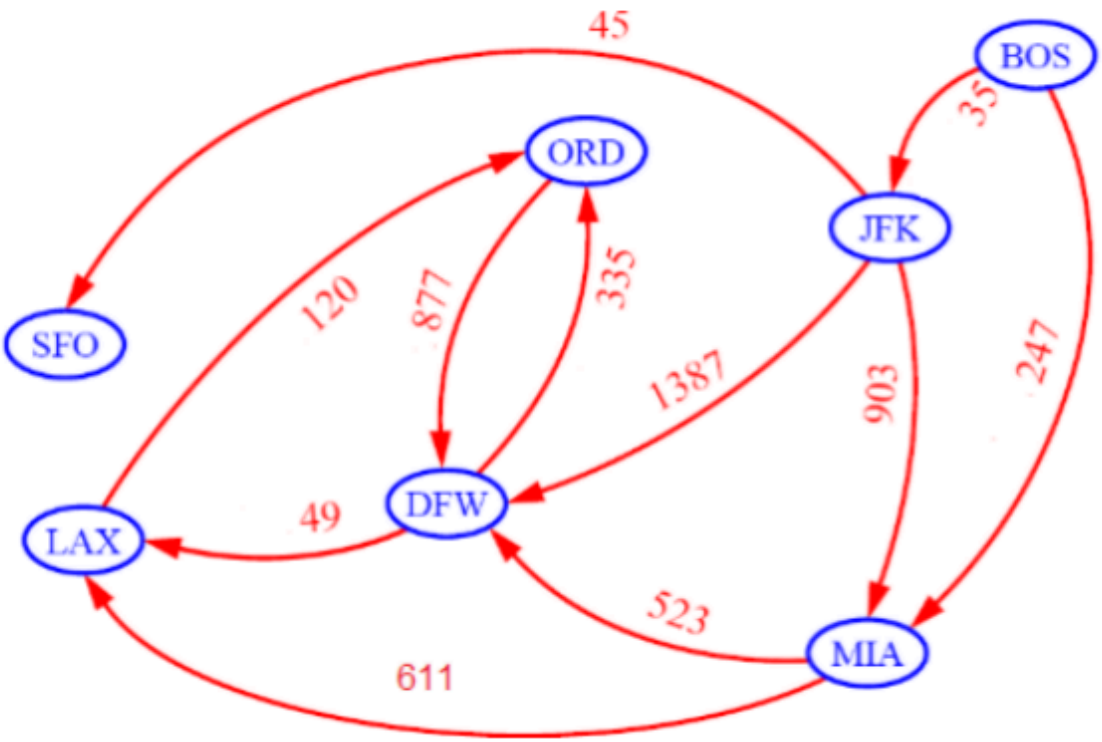
Grafo Simples Conexo Com Sequência de Graus (1, 1, 2, 3, 3, 4, 4, 6)



Grafo Simples Conexo Com Sequência de Graus (3, 3, 3, 3, 3, 5, 5, 5)



2: Interpretação do Grafo Direcionado



a) 12

b) 7

c) Não

d) MIA para DFW, DFW para LAX, com peso 523 + 43 = 566.

e)

	BOS	JFK	MIA	SFO	DFW	LAX	ORD
BOS	**0**	**35**	**247**	**0**	**0**	**0**	**0**
JFK	**0**	**0**	**903**	**45**	**1387**	**0**	**0**
MIA	**0**	**0**	**0**	**0**	**523**	**611**	**0**
SFO	**0**	**0**	**0**	**0**	**0**	**0**	**0**
DFW	**0**	**0**	**0**	**0**	**0**	**49**	**335**
LAX	**0**	**0**	**0**	**0**	**0**	**0**	**120**
ORD	**0**	**0**	**0**	**0**	**877**	**0**	**0**

f)

BOS -> (JFK | 35) -> (MIA | 247) ->
JFK -> (SFO | 45) -> (MIA | 903) -> (DFW | 1387) ->
MIA -> (DFW | 523) -> (LAX | 611) ->
SFO ->
DFW -> (LAX | 49) -> (ORD | 335) ->
LAX -> (ORD | 120) ->
ORD -> (DFW | 877) ->

g) O algoritmo PRIM deve ser utilizado em grafos não dirigidos, que não é o caso do grafo passado. Isso porque, o conceito de Árvore Geradora Mínima não se aplica a grafos dirigidos, já que, a mesma depende da existência de caminhos não direcionados (ida e volta) para que se tenha um caminho mínimo entre os vértices. Em grafos direcionados, pode existir casos em um vértice possui uma aresta vinda de outra apenas na ida, podendo causar problemas na execução do algoritmo, por exemplo, o vértice SFO, que ao chegar nele, o algoritmo iria quebrar, ou então, ORD, onde não é possível acessar todos os outros vértices a partir dele. Entretanto, ignorando as setas têm-se as seguintes iterações e Árvore Geradora Mínima:
Começando em SFO:

Arestas disponíveis :**(JFK | 45)**

Custo: 0 + 45 = 45
Árvore parcial: {SFO,JFK}

Em JFK:
Arestas disponíveis: **(MIA | 903), (DFW | 1387), (BOS | 35)**
custo: 45 + 35 = 80
Árvore Parcial: {SFO,JFK,BOS}

Em BOS:
Arestas disponíveis: **(MIA | 247)**
Custo: 247 + 80 = 327
Árvore Parcial {SFO,JFK,BOS,MIA}

Em MIA:
Arestas disponíveis: **(DFW | 523), (LAX | 611)**
Custo: 327 + 523 = 850
Árvore Parcial: {SFO,JFK,BOS,MIA,DFW}

Em DFW:
Arestas disponíveis: **(LAX | 49) , (ORD | 335), (ORD | 877)**
Custo: 850 + 49 = 899
Árvore Parcial: {SFO,JFK,BOS,MIA,DFW,LAX}

Em LAX:
Arestas disponíveis: **(ORD | 120)**
Custo: 899+120 = 1019
Árvore Parcial: {SFO,JFK,BOS,MIA,DFW,LAX,ORD}

Árvore Geradora Mínima {SFO,JFK,BOS,MIA,DFW,LAX,ORD}:
SFO - JFK (peso 45)
JFK - BOS (peso 35)
BOS - MIA (peso 247)
MIA - DFW (peso 523)
DFW - LAX (peso 49)
LAX - ORD (peso 120)
Custo Total: 1019

3: Descrição de Funções

A) **Grafo* cria_Grafo(int nro_vertices, int grau_max, int eh_ponderado):**

A função aloca memória para um novo grafo, recebendo por parâmetro o número de vértices deste grafo, o número máximo de graus que pode ter, também recebe eh_ponderado que vai ser um booleano de se pode receber mais uma aresta ou não, depois faz a estruturação e criação do novo grafo.

B) **libera_Grafo(Grafo* gr):**

A função recebe apenas um grafo de parâmetro e dentro da própria função inicia uma nova variável int para auxiliar nos loops para percorrer o grafo e dar free na memória alocada do grafo colocando a variável i < número de vértices vai percorrer o grafo dando free na memória alocada, e depois novamente colocando i < número de vértices quando he_ponderado é verdadeiro percorre o grafo e da fre na memória dos nós.

C) insereAresta(Grafo* gr, int orig, int dest, int eh_digrafo, float peso):

A função inicia recebendo cinco parâmetros um para o grafo de destino, uma para qual vai ser a origem para a nova aresta, outra para ser o destino da nova aresta, um int que vai funcionar como verdadeiro e falso 1 para verdadeiro e 0 para falso para identificar se o grafo é um grafo dirigido, e um float para ser o peso que vai ser mantido nesta aresta, e primeiro faz a verificação de se o grafo não é nulo e também se os vértices que vão ser ligados pela função não são inválidos após fazer estas verificações, e depois insire a nova aresta na posição indicada por origem e destino utilizando como base o grau para colocar no lugar correto, após a inserção realiza o ajuste do grau com grau++.

D) removeAresta(Grafo* gr, int orig, int dest, int eh_digrafo)

Inicia a função parecida com as outra recebendo o grafo como parâmetro a origem da aresta que vai ser removida e a aresta que vai ser removida como destino, e a verificação de verdadeiro ou falso de se o grafo é dígrafo ou não, após isso faz as verificações de se o grafo não é nulo, se o origem é menor que zero ou maior que a quantidade de vértices e também a verificação de se o dest é menor a zero ou maior que a quantidade de vértices, se passar por essas verificações contínua para a funcionalidade da função, ela funciona com um loop que percorre o grafo procurando onde o vértice dest está armazenado, quando encontra ela decrementa o grau dos vizinhos e copia a lista do último elemento para a posição da aresta que está sendo removida**.**