UNIVERSIDADE FEDERAL DO MARANHÃO
DEPARTAMENTO DE INFORMÁTICA
CURSO DE CIÊNCIA DA COMPUTAÇÃO
ESTRUTURA DE DADOS II (DEIN0083) 2024.1 P1

Professor(a): João Dallyson Sousa de Almeida

Data: 26/06/2024

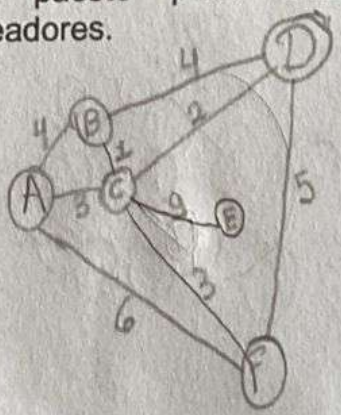
Matrícula: 2022029715 **Aluno:** Mario Vitor Vieira Cella

3ª Avaliação (70%)

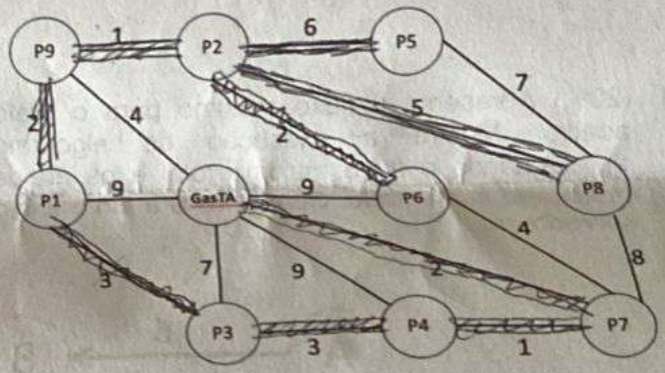
1) (20%) Protocolos de roteamento de estado de enlace utilizam difusão para propagar informações de estado de enlace que são usadas para calcular rotas individuais. Dessa forma, o algoritmo Dijkstra pode ser executado no local, com a finalidade de criar o caminho mais curto até todos os destinos possíveis. Os resultados desse algoritmo podem ser instalados nas tabelas de roteamento e a operação normal pode ser retomada. Assim, considere uma rede composta por 6 roteadores, designados pelas letras A, B, C, D, E e F, conectados conforme a seguinte tabela de custos de seus enlaces (tempo em ms):

Conexão	Enlace
A-B	4
A-C	3
A-F	6
B-C	1
B-D	4
C-D	2
C-E	9
C-F	3
D-F	5

Neste cenário, execute o algoritmo do Dijkstra passo a passo e apresente a tabela com os tempos mínimos **partindo do roteador "A".** Descreva a sua solução, mostre a fila de prioridade após cada iteração e o tempo mínimo total para enviar um pacote para todos os demais roteadores.



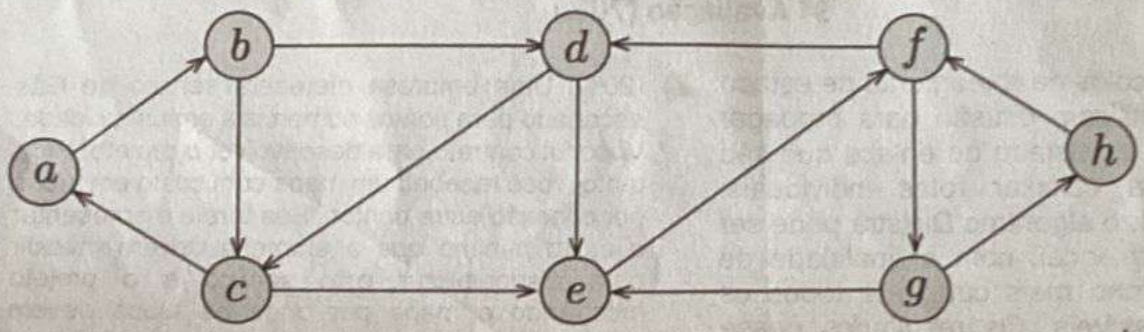
2) (20%) Uma empresa oferecerá serviço de Gás encanado para pontos comerciais em uma cidade. Você foi contrato para desenvolver o projeto. Para tanto, você recebeu um mapa com custo em tubos por conexão entre pontos. Sua tarefa é apresentar o custo mínimo que a empresa deverá investir para disponibilizar este serviço e o projeto mostrando o mapa por onde os tubos devem passar. Descreva a solução do problema e demostre a execução do algoritmo escolhido.



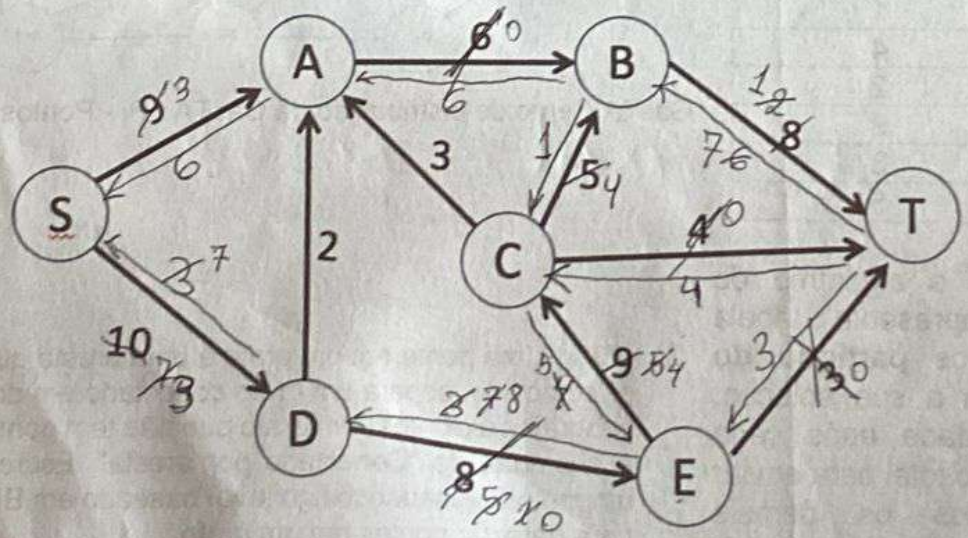
GasTA Cento de distribuição da GasTA ; Pi - Pontos

3) (20%) Uma ponte em um grafo é uma aresta que, se removida, separa um grafo conectado em dois subgrafos disjuntos. Um grafo que não tem pontes é chamado de "Conectado por aresta". Escreva um método (pseudocódigo/java) baseado em BFS para detectar pontes em um grafo.

4) (20%) Demonstre a execução do algoritmo de Busca em Profundidade no grafo abaixo, iniciando pelo vértice " b ". Apresente, também, a classificação das arestas e os ciclos.



5) (20%) Apresente o fluxo máximo para o grafo abaixo. Mostre a execução do algoritmo apresentando o fluxo máximo total e os grafos residual e aumentado. Considere S a fonte e T o sorvedouro.



Mario Vitor Vieira Cella

9,2

2,0

① Dijkstra (Começo: A)

1°) Q{A0, B∞, C∞, F∞, D∞, E∞}

. A sai da fila e atualiza os valores dos pesos.

2°) Q = {C3, B4, F6, D∞, E∞}

. C sai da fila e atualiza os pesos.

3°) Q = {D5, F6, E12}

. D sai da fila e atualiza os pesos.

4°) Q = {F6, E12}

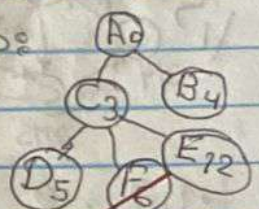
. F sai da fila.

5°) Q = {E,12}.

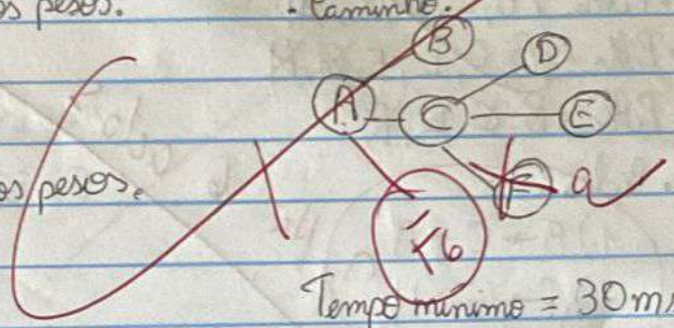
. E12 sai da fila de prioridade

. O algoritmo retorna a soma dos pesos: 0+3+4+5+6+12 = 30.

Heap:



. Caminho.



Tempo mínimo = 30ms

11,7

② Para encontrar o menor custo ao ligar todos pontos, dois algoritmos podem ser utilizados: PRIM e Kruskal.

. Vou escolher o Kruskal: ← Kruskal não usa Fila de Prioridade

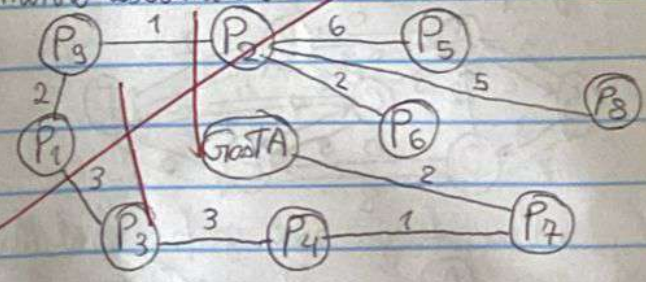
. Fila de prioridade = Q = {1,1,2,2,2,3,3,4,4,5,6,7,7,8,9,9}

. Arestas: Pesos.

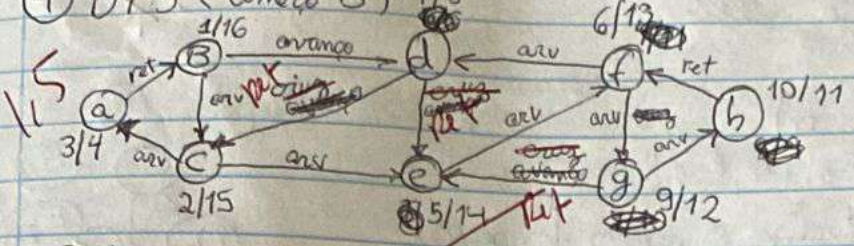
1) $P_9 \rightarrow P_2$: 1
2) $P_4 - P_7$: 1
3) $P_9 - P_1$: 2
4) $P_2 - P_6$: 2
5) GastA − P_7: 2
6) $P_3 - P_4$: 3
7) $P_3 - P_1$: 3
8) Arestas de peso 4 não podem ser incluídas (fechar ciclo)
9) $P_2 - P_8$: 5
10) $P_2 - P_5$: 6 +

25 ⟶ Menor Custo = 25

. Caminho escolhido:



④ DFS (começo B)



- Pilha: B, C, A
- Pilha: B, C, E, F, D
- Pilha: B, C, E, F, G, H
- Pilha: B, C, E, F
- Lidos:
  1) B → C → A
  2) f → G → H

Há 6 ciclos

- Classificação Arestas:
  A → B: retorno
  B → C: árvore
  C → A: árvore
  B → D: avanço
  F → d: árvore
  e → f: árvore
  f → g: arvore
  D → C: cruzada
  C → E → árvore
  g → h: árvore
  g → E: cruzada
  D → e: cruzada.
  H → F: retorno.

⑤. Escolhendo o 1º caminho como S → A → B → T

1º) $S \xrightarrow[6]{3} A \xrightarrow[6]{0} B \xrightarrow[6]{2} T$ f: 6

2º) $S \xrightarrow[3]{7} D \xrightarrow[3]{5} E \xrightarrow[3]{0} T$ f: 3

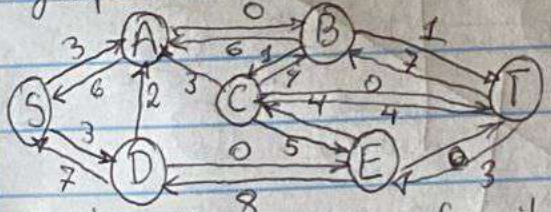
3º) $S \xrightarrow[7]{3} D \xrightarrow[7]{1} E \xrightarrow[4]{5} C \xrightarrow[4]{0} T$ f: 4

4º) $S \xrightarrow[8]{2} D \xrightarrow[8]{0} E \xrightarrow[5]{4} C \xrightarrow[1]{4} B \xrightarrow[7]{1} T$ f: 1

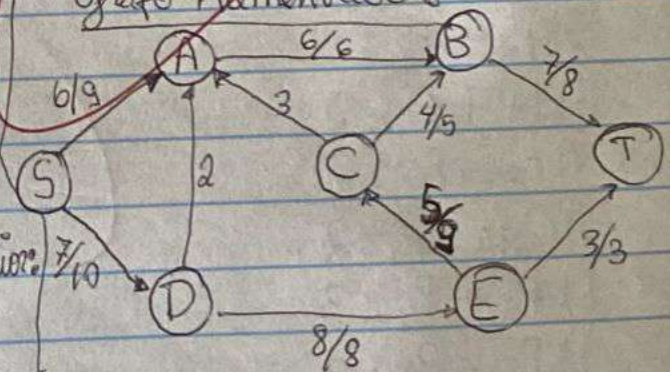
Fluxo Máx: 6 + 3 + 4 + 1 = 14

- Não existem mais caminhos possíveis saindo de S p/ t.

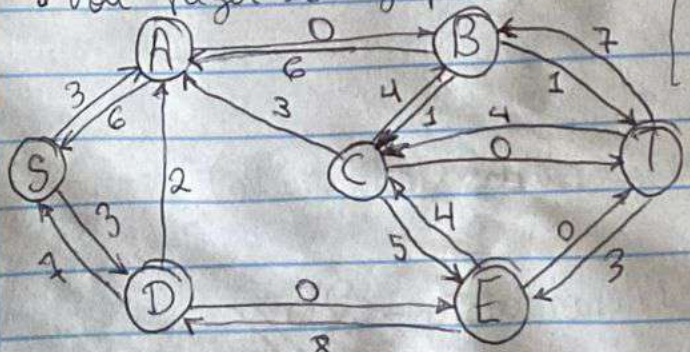
- Grafo Residual:



Grafo Aumentado:



- Vou fazer esse grafo residual Maior:



2,0

③ função detectar_pontes (grafo):
    pontes = lista_vazia()
    para cada aresta (u, v) no grafo:
        grafo_removido = remover_aresta (grafo, u, v)
        if not bfs (grafo_removido, u, v):
            pontes.adicionar ((u, v)))
    retornar pontes

função bfs (grafo, start, target):
visitados = conjunto_vazio()
    fila = fila_vazia()
    fila.enfileirar (start)
    enquanto fila não está vazia:
        nó_atual = fila.desenfileirar()
        se nó_atual == target:
            retornar verdadeiro
        para cada vizinho de nó_atual no grafo:
            se vizinho não está em visitado:
                visitado.adicionar (vizinho)
                fila.enfileirar (vizinho)
    retornar false;

função remover_aresta (grafo, u, v):
    grafo_copia = copiar (grafo)
    grafo.copia.remover_aresta (u, v)
    remover grafo_copia

2,0

③ função detectar_pontes (grafo):
    pontes = lista_vazia()
    para cada aresta (u, v) no grafo:
        grafo_removido = remover_aresta (grafo, u, v)
        if not bfs (grafo_removido, u, v):
            pontes.adicionar ((u, v)))
    retornar pontes

função bfs (grafo, start, target):
visitados = conjunto_vazio()
    fila = fila_vazia()
    fila.enfileirar (start)
    enquanto fila não está vazia:
        nó_atual = fila.desenfileirar()
        se nó_atual == target:
            retornar verdadeiro
        para cada vizinho de nó_atual no grafo:
            se vizinho não está em visitado:
                visitado.adicionar (vizinho)
                fila.enfileirar (vizinho)
    retornar false;

função remover_aresta (grafo, u, v):
    grafo_copia = copiar (grafo)
    grafo.copia.remover_aresta (u, v)
    remover grafo_copia