Cauã Borges Faria RA: 834437

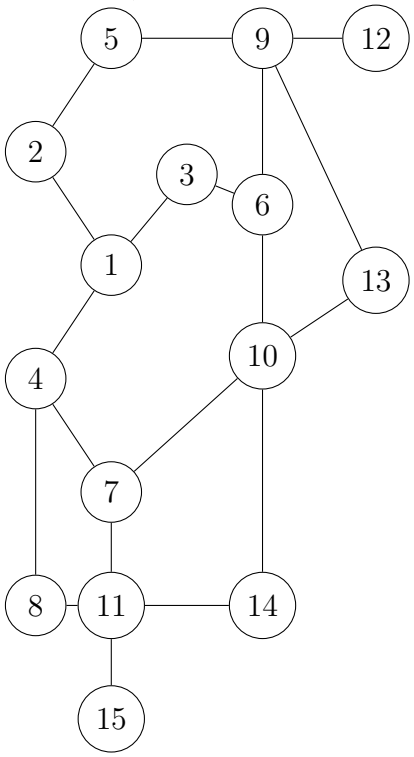
Atividade Avaliativa 4

1. Expedição à Caverna Perdida com Busca em Largura (BFS)

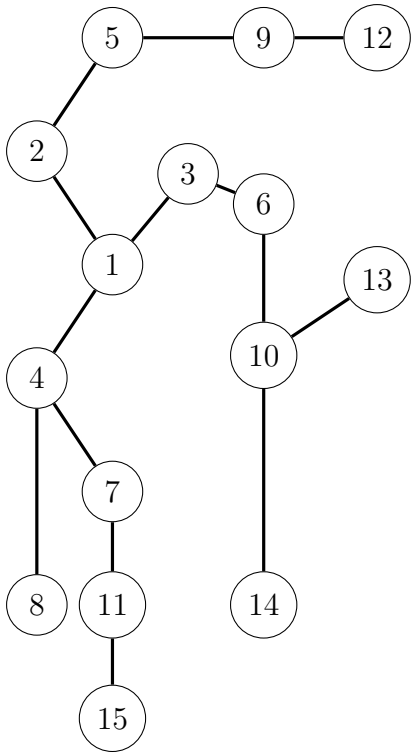
a) Estrutura da Caverna: grafo não-direcionado

A caverna é composta por 15 salas conectadas por túneis, de acordo com o seguinte mapeamento (*cada sala é um vértice, cada túnel é uma aresta não-direcionada*):

- Sala 1 (Entrada) conecta com as salas 2, 3 e 4; - Sala 2 conecta com a sala 5; - Sala 3 conecta com a sala 6; - Sala 4 conecta com as salas 7 e 8; - Sala 5 conecta com a sala 9; - Sala 6 conecta com as salas 9 e 10; - Sala 7 conecta com as salas 10 e 11; - Sala 8 conecta com a sala 11; - Sala 9 conecta com as salas 12 e 13; - Sala 10 conecta com as salas 13 e 14; - Sala 11 conecta com as salas 14 e 15 (Câmara Secreta).



i	$u = \text{pop}(Q)$	$V' = \{v \in N(u) \mid v.\text{color} = \text{WHITE}\}$	$\lambda(v)$	$\pi(v)$
0	1	$\{2, 3, 4\}$	$\lambda(2) = \lambda(1) + 1 = 1$ $\lambda(3) = \lambda(1) + 1 = 1$ $\lambda(4) = \lambda(1) + 1 = 1$	$\pi(2) = 1$ $\pi(3) = 1$ $\pi(4) = 1$
1	2	$\{5\}$	$\lambda(5) = \lambda(2) + 1 = 2$	$\pi(5) = 2$
2	3	$\{6\}$	$\lambda(6) = \lambda(3) + 1 = 2$	$\pi(6) = 3$
3	4	$\{7, 8\}$	$\lambda(7) = \lambda(4) + 1 = 2$ $\lambda(8) = \lambda(4) + 1 = 2$	$\pi(7) = 4$ $\pi(8) = 4$
4	5	$\{9\}$	$\lambda(9) = \lambda(5) + 1 = 3$	$\pi(9) = 5$
5	6	$\{10\}$	$\lambda(10) = \lambda(6) + 1 = 3$	$\pi(10) = 6$
6	7	$\{11\}$	$\lambda(11) = \lambda(7) + 1 = 3$	$\pi(11) = 7$
7	8	$\emptyset$	—	—
8	9	$\{12, 13\}$	$\lambda(12) = \lambda(9) + 1 = 4$ $\lambda(13) = \lambda(9) + 1 = 4$	$\pi(12) = 9$ $\pi(13) = 9$
9	10	$\{14\}$	$\lambda(14) = \lambda(10) + 1 = 4$	$\pi(14) = 10$
10	11	$\{15\}$	$\lambda(15) = \lambda(11) + 1 = 4$	$\pi(15) = 11$
11	12	$\emptyset$	—	—
12	13	$\emptyset$	—	—
13	14	$\emptyset$	—	—
14	15	$\emptyset$	—	—



BFS-TREE

Passo	**Fila**
$Q^{(0)}$	$[1]$
$Q^{(1)}$	$[2, 3, 4]$
$Q^{(2)}$	$[3, 4, 5]$
$Q^{(3)}$	$[4, 5, 6]$
$Q^{(4)}$	$[5, 6, 7, 8]$
$Q^{(5)}$	$[6, 7, 8, 9]$
$Q^{(6)}$	$[7, 8, 9, 10]$
$Q^{(7)}$	$[8, 9, 10, 11]$
$Q^{(8)}$	$[9, 10, 11]$
$Q^{(9)}$	$[10, 11, 12, 13]$
$Q^{(10)}$	$[11, 12, 13, 14]$
$Q^{(11)}$	$[12, 13, 14, 15]$
$Q^{(12)}$	$[13, 14, 15]$
$Q^{(13)}$	$[14, 15]$
$Q^{(14)}$	$[15]$
$Q^{(15)}$	$\emptyset$

b) Complexidade Computacional da Busca em Largura

A análise da complexidade da BFS clássica em um grafo com n vértices e m arestas segue abaixo.

- **Inicialização:** Para colorir e inicializar todos os n vértices, o custo é $O(n)$. - **Exploração das arestas:** Para cada vértice removido da fila, percorremos cada um dos seus vizinhos. Como o grafo é não-direcionado, cada aresta é examinada no máximo duas

vezes (uma por extremidade). Portanto, o custo total para esse processo é $O(m)$. - **Fila:** Cada vértice entra e sai da fila ao menos uma vez, custo $O(n)$.

Assim, o custo total da BFS pode ser expresso por:

$$T(n, m) = O(n) + O(m) = O(n + m)$$

ou seja, linear em relação à soma do número de vértices e arestas do grafo.

c) Prova de que BFS Calcula as Menores Distâncias em Grafos Não Ponderados

Teorema: Executando BFS a partir de um vértice s, ao terminar tem-se $\lambda(v) = d(s, v)$ para todo vértice v, onde $d(s, v)$ é a menor distância (mínimo número de arestas) entre s e v.

Demonstração por contradição:

1. Suponha que exista $v \in V$ tal que, ao ser removido da fila Q pela primeira vez, temos $\lambda(v) > d(s, v)$, e que v seja o primeiro nó para o qual essa desigualdade ocorre.
2. Como $d(s, v) < \infty$, existe caminho mínimo P de s a v. Seja u o vértice imediatamente anterior a v nesse caminho (ou seja, o predecessor de v em P).
3. Por definição de caminho mínimo, tem-se $d(s, v) = d(s, u) + 1$.
4. Quando u foi removido da fila, a aresta (u, v) foi considerada. Assim, apenas três possibilidades podem ocorrer quanto à cor de v:
   - **(i)** v **estava BRANCO:** $\lambda(v)$ seria atualizado para $\lambda(u) + 1 = d(s, u) + 1 = d(s, v)$, o que contradiz a hipótese de $\lambda(v) > d(s, v)$.
   - **(ii)** v **estava PRETO:** v já saiu da fila antes de u, então necessariamente $\lambda(v) \leq \lambda(u)$, logo $\lambda(v) < \lambda(u) + 1 = d(s, v)$, o que é absurdo.
   - **(iii)** v **estava CINZA:** v já foi descoberto anteriormente por algum w removido antes de u, logo $\lambda(v) \leq \lambda(w) + 1 \leq \lambda(u) + 1 = d(s, v)$.
5. Nenhuma dessas hipóteses resulta em $\lambda(v) > d(s, v)$.

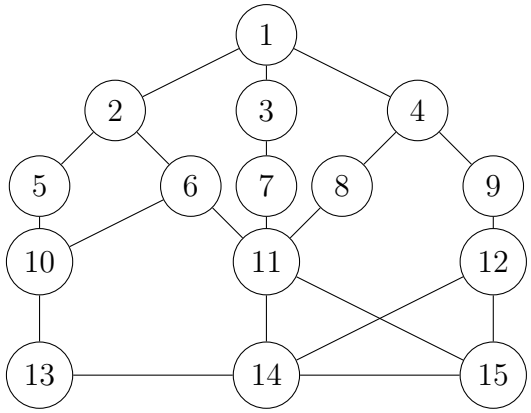
Portanto, por absurdo, sempre $\lambda(v) = d(s, v)$ ao retirar v da fila, mostrando que BFS constrói corretamente a árvore de caminhos mínimos em grafos não ponderados.

2. Resgate na Estação Espacial com Busca em Profundidade (DFS)

a) Estrutura da Estação Espacial

A estação é composta por 15 módulos interconectados por corredores. Cada módulo é representado por um vértice e cada ligação, por uma aresta não-direcionada. Os módulos e suas interligações são:

- Módulo 1 (Sala de Controle) conecta com os módulos 2, 3 e 4. - Módulo 2 conecta com os módulos 5 e 6. - Módulo 3 conecta com o módulo 7. - Módulo 4 conecta com os módulos 8 e 9. - Módulo 5 conecta com o módulo 10. - Módulo 6 conecta com os módulos 10 e 11. - Módulo 7 conecta com o módulo 11. - Módulo 8 conecta com o módulo 11. - Módulo 9 conecta com o módulo 12. - Módulo 10 conecta com o módulo 13. - Módulo 11 conecta com os módulos 14 e 15 (Sala de Suporte Vital). - Módulo 12 conecta com os módulos 14 e 15. - Módulo 13 conecta com o módulo 14. - Módulo 14 conecta com o módulo 15.

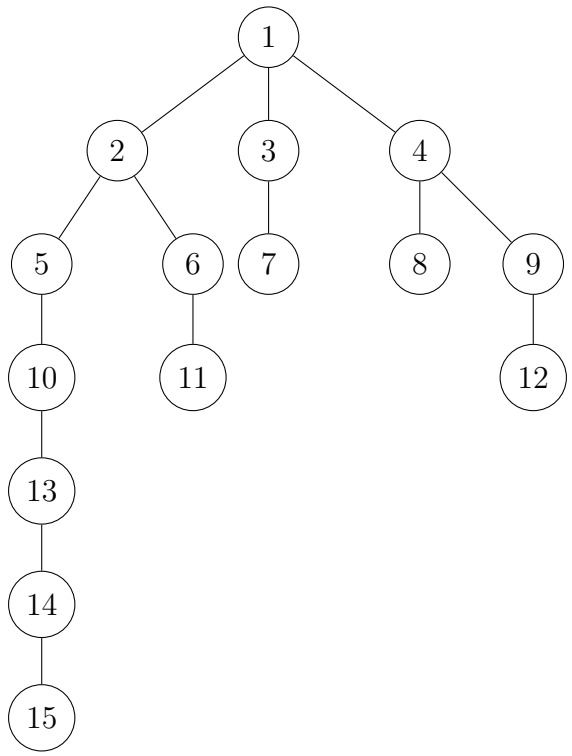


Ordem de acesso aos vértices (com detalhes de DFS)

u	u.color	u.d	V' = {v ∈ N(u) — v.color = WHITE}	π(v)	u.f
1	gray	1	{2, 3, 4}	—	30
2	gray	2	{5, 6}	1	17
5	gray	3	{10}	2	12
10	gray	4	{13}	5	11
13	gray	5	{14}	10	10
14	gray	6	{15}	13	9
15	gray	7	∅	14	8
6	gray	13	{11}	2	16
11	gray	14	∅	7	15
3	gray	18	{7}	1	21
7	gray	19	∅	3	20
4	gray	22	{9, 8}	1	29
8	gray	23	∅	4	24
9	gray	25	{12}	4	28
12	gray	26	∅	9	27

Pilha

Pilha
12
9
8
4
7
3
11
6
15
14
13
10
5
2
1



DFS-TREE

b) Análise Matemática da Complexidade do DFS

O algoritmo DFS em um grafo com n vértices e m arestas funciona de maneira recursiva, explorando completamente cada vértice antes de voltar ao anterior. Vamos detalhar o custo:

- **Inicialização:** Marcar todos os n vértices como não visitados, custo $O(n)$. - **Chamada Recursiva:** Para cada vértice, ao ser descoberto, percorrer a lista de vizinhos. - **Visitação de Arestas:** Como cada aresta (em grafo não direcionado) aparece

duas vezes (uma para cada extremidade), o total de iterações sobre todos os vizinhos dos vértices é $2m$.

Assim, durante toda execução, - cada vértice é visitado exatamente uma vez, - cada aresta é considerada no máximo duas vezes.

Logo, o tempo total é dominado pelo número de vértices mais o número de arestas:

$$T(n, m) = O(n) + O(m) = O(n + m)$$

Isto significa que, para grafos esparsos (quando m está próximo de n), o DFS é extremamente eficiente, escalando linearmente conforme cresce a entrada.

3. Resgate em uma Cidade Inundada com Dijkstra

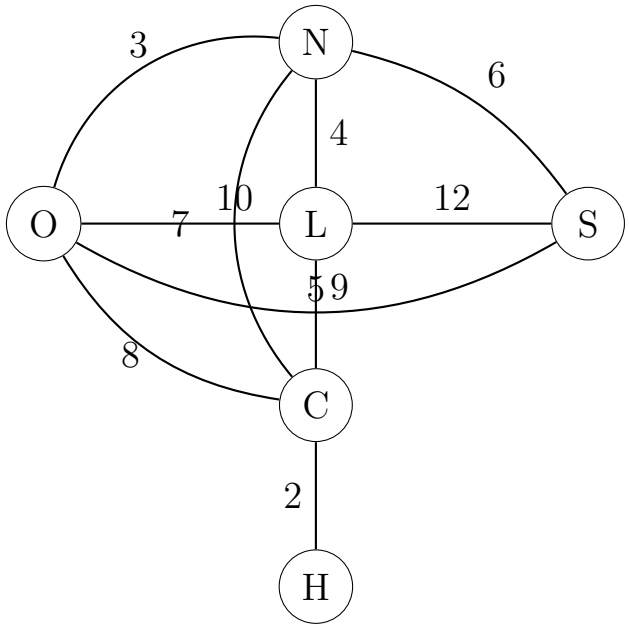
a) Estrutura do Problema e Descrição do Grafo

A cidade é composta por seis bairros e cada um é representado por um vértice. As ruas conectando os bairros são arestas ponderadas, cujo peso indica o tempo, em minutos, necessário para atravessar aquela rua. A conexão entre Bairro Sul (5) e o Hospital (6) encontra-se bloqueada e, portanto, não pode ser utilizada no cálculo dos caminhos.

As conexões possíveis e respectivos tempos são:

- Centro (1) se conecta a Bairro Norte (2) em 10 min; - Centro (1) se conecta a Bairro Leste (3) em 5 min; - Centro (1) se conecta a Bairro Oeste (4) em 8 min; - Bairro Norte (2) se conecta a Bairro Oeste (4) em 3 min; - Bairro Norte (2) se conecta a Bairro Sul (5) em 6 min; - Bairro Leste (3) se conecta a Bairro Oeste (4) em 7 min; - Bairro Leste (3) se conecta a Bairro Sul (5) em 12 min; - Bairro Leste (3) se conecta a Bairro Norte (2) em 4 min; - Bairro Oeste (4) se conecta a Bairro Sul (5) em 9 min; - Bairro Oeste (4) se conecta ao Hospital (6) em 2 min.

Destaca-se novamente: *a rua entre Bairro Sul (5) e o Hospital (6) está bloqueada, sendo a aresta desconsiderada.*

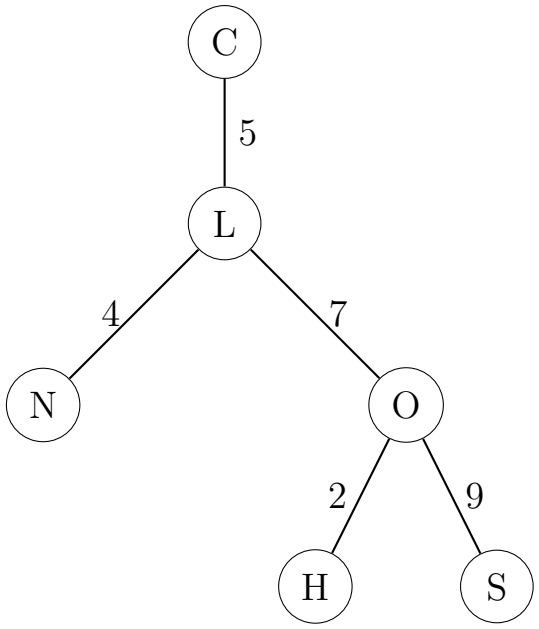


	C	N	L	O	S
$\lambda(0)(v)$	0	∞	∞	∞	∞
$\lambda(1)(v)$	0	10	5	8	∞
$\lambda(2)(v)$	0	10	5	8	16
$\lambda(3)(v)$	0	9	5	8	16
$\lambda(4)(v)$	0	9	5	8	16

U	V' = {v ∈ N(u) ∧ v ∈ Q}	λ(v), ∀ v ∈ V'	π(v)
C	{N, L, O}	λ(N)=min(λ(N), λ(C)+w(U,N))=min(∞,0+10)=10 λ(L)=min(λ(L), λ(C)+w(U,L))=min(∞,0+5)=5 λ(O)=min(λ(O), λ(C)+w(U,O))=min(∞,0+8)=8	π(N) = C π(L) = C π(O) = C
N	{O,S}	λ(O)=min(λ(O), λ(N)+w(N,O))=min(8,10+3)=8 λ(S)=min(λ(S), λ(N)+w(N,S))=min(∞,10+6)=16	π(O) = N π(S) = N
L	{O, S, N}	λ(O)=min(λ(O), λ(L)+w(L,O))=min(8,5+7)=8	π(O) = L

U	V' = {v ∈ N(u) ∧ v ∈ Q}	λ(v), ∀ v ∈ V'	π(v)
		λ(S)=min(λ(S), λ(L)+w(L,S))=min(16,5+12)=16 λ(N)=min(λ(N), λ(L)+w(L,N))=min(10,5+4)=9	π(S) = L π(N) = L
O	{S, H}	λ(S)=min(λ(S), λ(O)+w(O,S))=min(16,8+9)=16 λ(H)=min(λ(H), λ(O)+w(O,H))=min(∞,8+2)=10	π(S) = O π(H) = O
S H	- -	-	-

v H	C	N	L	O	S
π(v) O	—	L	C	C	O
λ(v) 10	0	9	5	8	16



b) Análise da Complexidade Computacional do Dijkstra

(i) Implementação com fila de prioridades simples (array):

- Inicialização de distâncias: $O(n)$. - Em cada iteração do while, busca pelo menor valor λ na fila, o que custa $O(n)$, repetido n vezes: $O(n^2)$. - As operações de relaxamento, atualização e inserção podem ser feitas em tempo constante ou linear na vizinhança. - No todo, temos $T(n) = O(n^2) + O(m) = O(n^2)$, pois $O(n^2)$ domina para grandes n.

(ii) Implementação com min-heap:

- Inicialização: $O(n)$. - Cada operação de inserção ou extração do menor elemento no heap custa $O(\log n)$. - As operações de `ExtractMin` e `DecreaseKey` sobre todos os vértices e arestas, respectivamente, custam $O(n \log n)$ e $O(m \log n)$. - Assim, totalizando $T(n) = O(n \log n) + O(m \log n) = O(m \log n)$.

Conclusão: Para grafos grandes, a implementação com min-heap é preferível, especialmente quando o número de arestas m é próximo de n (grafos esparsos). A versão com array só é prática para grafos muito pequenos.

c) Prova de Correção: Dijkstra Computa Caminhos Mínimos em Grafos com Pesos Não Negativos

Seja $G = (V, E)$ com pesos $w(u, v) \geq 0$ para quaisquer arestas. Queremos provar que, ao final da execução do Dijkstra a partir de s, temos, para todo $v \in V$, $\lambda(v) = d(s, v)$, onde $d(s, v)$ é a distância geodésica.

Base da indução: O primeiro vértice extraído é a própria fonte s, para o qual $\lambda(s) = 0 = d(s, s)$, portanto a base é satisfeita.

Hipótese da indução: Assuma que, após k extrações, para todo vértice já retirado, $\lambda(u) = d(s, u)$.

Passo da indução: Seja v o próximo vértice a sair da fila. Por absurdo, suponha que $\lambda(v) > d(s, v)$. Considere um caminho ótimo de s até v usando os vértices $s = v_0, v_1, \ldots, v_k = v$, e suponha que algum intermediário v_i está ainda na fila quando seu antecessor v_{i-1} foi retirado. Quando v_{i-1} foi removido, a aresta (v_{i-1}, v_i) foi relaxada,

então:

$$\lambda(v_i) \leq \lambda(v_{i-1}) + w(v_{i-1}, v_i)$$

Mas $\lambda(v_{i-1}) = d(s, v_{i-1})$ (hipótese), então

$$\lambda(v_i) \leq d(s, v_{i-1}) + w(v_{i-1}, v_i) = d(s, v_i) \ .$$

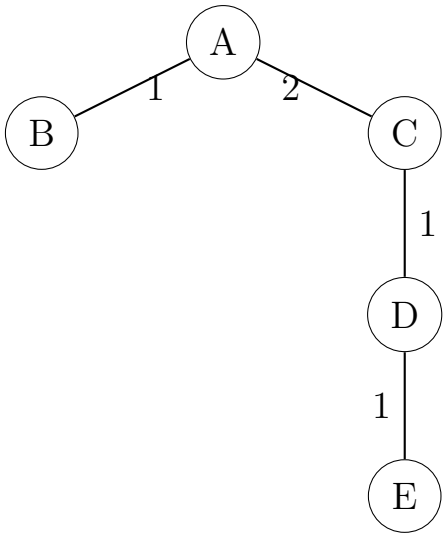
Seguindo este raciocínio pelo caminho mínimo, chegamos a $\lambda(v) \leq d(s, v)$, que contradiz a hipótese inicial. Portanto, remove-se da fila exatamente no momento em que $\lambda(v) = d(s, v)$.

Em conclusão, ao final da execução do algoritmo, todos os $\lambda(v)$ obtidos correspondem às menores distâncias de s até v, demonstrando a correção do Dijkstra em grafos com pesos não negativos.

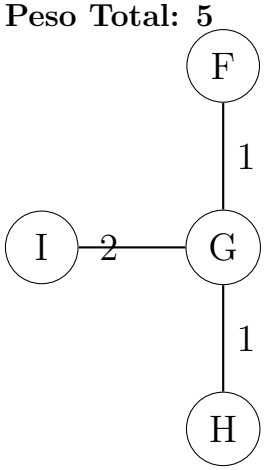
4. Sistema de Navegação de Drones com Dijkstra Multisource

	A	B	C	D	E	F	G	H	I
$\lambda(0)(v)$	0	∞	∞	∞	∞	∞	∞	0	∞
$\lambda(1)(v)$	0	1	2	∞	∞	∞	∞	0	∞
$\lambda(2)(v)$	0	1	2	∞	∞	5	1	0	4
$\lambda(3)(v)$	0	1	2	∞	∞	3	1	0	4
$\lambda(4)(v)$	0	1	2	∞	6	2	1	0	3
$\lambda(5)(v)$	0	1	2	3	5	2	1	0	3
$\lambda(6)(v)$	0	1	2	3	5	2	1	0	3
$\lambda(7)(v)$	0	1	2	3	4	2	1	0	3

U	V' = {v ∈ N(u) ∧ v ∈ Q}	λ(v), ∀ v ∈ V'	π(v)
A	{B, C}	λ(B)=min(λ(B), λ(A)+w(A,B))=min(∞,0+1)=1 λ(C)=min(λ(C), λ(A)+w(A,C))=min(∞,0+2)=2	π(B) = A π(C) = A
H	{I, F, G}	λ(I)=min(λ(I), λ(H)+w(H,I))=min(∞,0+4)=4 λ(F)=min(λ(F), λ(H)+w(H,F))=min(∞,0+5)=5 λ(G)=min(λ(G), λ(H)+w(H,G))=min(∞,0+1)=1	π(I) = H π(F) = H π(G) = H
B	{C, F}	λ(C)=min(λ(C), λ(B)+w(B,C))=min(2,1+4)=2 λ(F)=min(λ(F), λ(B)+w(B,F))=min(5,1+2)=3	π(C) = A π(F) = B
G	{I, F, E}	λ(I)=min(λ(I), λ(G)+w(G,I))=min(4,1+2)=3 λ(F)=min(λ(F), λ(G)+w(G,F))=min(3,1+1)=2 λ(E)=min(λ(E), λ(G)+w(G,E))=min(∞,1+5)=6	π(I) = G π(F) = G π(E) = G
C	{D, E}	λ(D)=min(λ(D), λ(C)+w(C,D))=min(∞,2+1)=3 λ(E)=min(λ(E), λ(C)+w(C,E))=min(6,2+3)=5	π(D) = C π(E) = C
F	{D}	λ(D)=min(λ(D), λ(F)+w(F,D))=min(3,2+3)=3	π(D) = C
D	{E}	λ(E)=min(λ(E), λ(D)+w(D,E))=min(5,3+1)=4	π(E) = D
E	∅	—	—
I	∅	—	—



Peso Total: 5



Peso Total: 4

Os tempos de entrega para os pontos F, G e I, anteriormente atendidos por H, seriam impactados da seguinte forma:

- **Ponto F:** O tempo de entrega aumentaria de 2 minutos para 3 minutos (rota A → B → F), representando um acréscimo de +1 minuto.
- **Ponto G:** O tempo de entrega passaria de 1 minuto para 4 minutos (rota A → B → F → G), resultando em um aumento de +3 minutos.
- **Ponto I:** O tempo de entrega subiria de 3 minutos para 6 minutos (rota A → B → F → G → I), com um acréscimo de +3 minutos.

Portanto, a falha do centro H resultaria em um aumento significativo nos tempos de voo para os pontos afetados, além de uma considerável elevação da carga logística sobre o centro A.