Fibra Ótica

Matheus Henrique de Castilho - 20222521

Resumo

Este trabalho investiga o uso de grafos na otimização do cabeamento de fibras ópticas, essencial para a transmissão de dados. A modelagem da infraestrutura permite aplicar algoritmos como Kruskal, para minimizar custos, e Dijkstra, para encontrar a rota mais curta. Uma versão modificada do Dijkstra prioriza rotas mais seguras, reduzindo falhas na rede. Além disso, no futuro, técnicas de Inteligência Artificial poderão aprimorar a manutenção e a otimização dinâmica da infraestrutura.

Introdução

A infraestrutura de fibra óptica é essencial para a transmissão rápida de dados, sendo sua expansão vital para assegurar uma comunicação eficiente tanto em áreas urbanas quanto rurais. Contudo, um dos principais desafios é determinar a melhor rota para o cabeamento, com o objetivo de reduzir os custos de implementação e garantir a confiabilidade da rede.

Esse desafio pode ser representado por meio de grafos, onde os nós representam os pontos de conexão e as arestas são as ligações entre as fibras ópticas. A utilização dessa abordagem permite aplicar algoritmos clássicos da teoria dos grafos para otimizar não apenas o custo de instalação, mas também a segurança e a eficiência na transmissão de dados.

Figura 1: *https://www.desktop.com.br/blog/internet-de-fibra-optica-o-caminho-ate-a-sua-casa/*

Modelagem do Problema como um Grafo

Para representar esse problema por meio de grafos, podemos definir:

- **Nós:** Representam pontos de conexão, como centrais de distribuição, bairros ou edifícios.
- **Arestas:** Representam os trechos de fibra óptica que ligam os nós.
- **Pesos:** O peso das arestas pode ser definido com base no custo de instalação, na distância entre os pontos ou na qualidade do sinal.
- **Fator de Risco:** Cada aresta também pode ter um peso baseado na probabilidade de falhas, como rompimentos por caminhões, tempestades ou desgaste da fibra.
- **Características do Grafo:**
  - **Grafo Conexo:** Para garantir que todos os pontos estejam interligados.

- **Grafo Ponderado:** Pois cada conexão possui um custo associado.
- **Grafo Possivelmente Cíclico:** Dependendo das redundâncias implementadas na rede.
- **Grafo Dirigido ou Não Dirigido:** Pode ser considerado não dirigido, pois o cabeamento de fibra geralmente funciona nos dois sentidos.

Algoritmos Aplicados

Algoritmos clássicos de grafos podem ser aplicados para resolver esse problema:

- **Algoritmo de Prim ou Kruskal:** Usado para encontrar a árvore geradora mínima, garantindo que todas as conexões sejam estabelecidas com o menor custo possível.
- **Algoritmo de Dijkstra:** Útil para encontrar o caminho mais curto entre dois pontos específicos, auxiliando na distribuição eficiente dos dados pela rede.
- **Caminho Mais Seguro:** Uma variação do Dijkstra que leva em conta o fator de risco, priorizando rotas que tenham menor probabilidade de falhas.

Árvore Geradora Mínima (Kruskal)

Para assegurar que a rede de fibra óptica seja montada com o custo mais baixo possível, podemos utilizar o Algoritmo de Kruskal para determinar a Árvore Geradora Mínima (AGM). Esse algoritmo organiza as arestas com base no custo e as adiciona sequencialmente, garantindo que todas as conexões sejam feitas de maneira a minimizar o custo total, sem a criação de ciclos.

- **Grafo:** Conexo, Ponderado, Não Dirigido.

- **Objetivo:** Minimizar o custo total da rede de cabeamento de fibras.

Exemplo de uso: Ao aplicar o algoritmo de Kruskal, podemos otimizar a distribuição de cabos entre os pontos de conexão, garantindo que a infraestrutura de fibra seja estabelecida com o menor custo, sem desperdício de recursos. Link GDB: https://onlinegdb.com/2Jsgrkf9y

```python
class Grafo:
    def __init__(self, vertices):
        self.V = vertices
        self.arestas = []  # Lista de arestas (custo, u, v)

    def adicionar_aresta(self, u, v, custo):
        self.arestas.append((custo, u, v))

    def encontrar_subconjunto(self, pai, i):
        if pai[i] == i:
            return i
        return self.encontrar_subconjunto(pai, pai[i])

    def unir_subconjuntos(self, pai, rank, x, y):
        raiz_x = self.encontrar_subconjunto(pai, x)
        raiz_y = self.encontrar_subconjunto(pai, y)
        if rank[raiz_x] < rank[raiz_y]:
            pai[raiz_x] = raiz_y
        elif rank[raiz_x] > rank[raiz_y]:
            pai[raiz_y] = raiz_x
        else:
            pai[raiz_y] = raiz_x
            rank[raiz_x] += 1

    def kruskal(self):
        self.arestas.sort()  # Ordena pelo custo
        pai = []
        rank = []
        arvore_minima = []

        for nodo in range(self.V):
            pai.append(nodo)
            rank.append(0)

        num_arestas = 0
```

Figura 2: *Imagem criada pelo próprio autor.*

```python
        i = 0
        while num_arestas < self.V - 1:
            custo, u, v = self.arestas[i]
            i += 1
            x = self.encontrar_subconjunto(pai, u)
            y = self.encontrar_subconjunto(pai, v)
            if x != y:
                arvore_minima.append((u, v, custo))
                self.unir_subconjuntos(pai, rank, x, y)
                num_arestas += 1

        return arvore_minima

# Exemplo de uso:
g = Grafo(6)  # 6 pontos de conexão
g.adicionar_aresta(0, 1, 2)
g.adicionar_aresta(0, 2, 3)
g.adicionar_aresta(1, 2, 1)
g.adicionar_aresta(1, 3, 4)
g.adicionar_aresta(2, 3, 5)
g.adicionar_aresta(3, 4, 2)
g.adicionar_aresta(4, 5, 3)

arvore_minima = g.kruskal()
print("Árvore Geradora Mínima (menor custo):", arvore_minima)
```

Figura 3: *Imagem criada pelo próprio autor.*

Caminho Mínimo (Dijkstra)

Outro desafio relevante é determinar a rota mais curta entre dois pontos específicos na rede de fibra óptica. O Algoritmo de Dijkstra é uma ferramenta eficaz para calcular o caminho mais eficiente, levando em consideração o custo ou a distância entre os nós.

- **Grafo:** Conexo, Ponderado, Não Dirigido.
- **Objetivo:** Encontrar o caminho mais curto entre dois pontos.

Exemplo de uso: Se quisermos otimizar a transmissão de dados entre dois pontos, podemos usar o Dijkstra para identificar a rota mais curta, garantindo que os dados percorrem o caminho mais eficiente possível, sem sobrecarregar a rede. Link GDB: https://onlinegdb.com/_x6GCyjr0

```python
import heapq

class Grafo:
    def __init__(self):
        self.grafo = {}

    def adicionar_aresta(self, u, v, custo):
        if u not in self.grafo:
            self.grafo[u] = []
        if v not in self.grafo:
            self.grafo[v] = []
        self.grafo[u].append((v, custo))
        self.grafo[v].append((u, custo))  # Grafo não-direcionado

    def dijkstra(self, inicio, destino):
        fila_prioridade = []
        heapq.heappush(fila_prioridade, (0, inicio))  # (custo, nó)
        distancias = {nodo: float("inf") for nodo in self.grafo}
        distancias[inicio] = 0
        caminho = {}

        while fila_prioridade:
            custo_atual, nodo_atual = heapq.heappop(fila_prioridade)

            if nodo_atual == destino:
                break  # Encontramos o destino

            for vizinho, peso in self.grafo[nodo_atual]:
                novo_custo = custo_atual + peso
                if novo_custo < distancias[vizinho]:
                    distancias[vizinho] = novo_custo
                    heapq.heappush(fila_prioridade, (novo_custo, vizinho))
                    caminho[vizinho] = nodo_atual  # Registrar o caminho
```

Figura 4: *Imagem criada pelo próprio autor.*

- **Grafo:** Conexo, Ponderado, Não Dirigido.

Exemplo de uso: Ao priorizar rotas com menor risco de falha, conseguimos garantir que a rede de fibra óptica seja mais confiável, evitando interrupções devido a fatores externos, como danos ao cabo ou condições climáticas adversas. Link GDB: https://onlinegdb.com/AaZ7UgmMC

```python
        # Reconstruindo o caminho mais curto
        rota = []
        atual = destino
        while atual in caminho:
            rota.append(atual)
            atual = caminho[atual]
        rota.append(inicio)
        rota.reverse()

        return rota, distancias[destino]

# Exemplo de uso:
g = Grafo()
g.adicionar_aresta("A", "B", 4)
g.adicionar_aresta("A", "C", 2)
g.adicionar_aresta("B", "C", 5)
g.adicionar_aresta("B", "D", 10)
g.adicionar_aresta("C", "D", 3)
g.adicionar_aresta("D", "E", 8)
g.adicionar_aresta("E", "F", 6)

inicio = "A"
destino = "E"
rota, custo = g.dijkstra(inicio, destino)
print(f"Menor rota de {inicio} para {destino}: {rota} com custo {custo}")
```

Figura 5: *Imagem criada pelo próprio autor.*

Caminho Mais Seguro (Dijkstra Modificado)

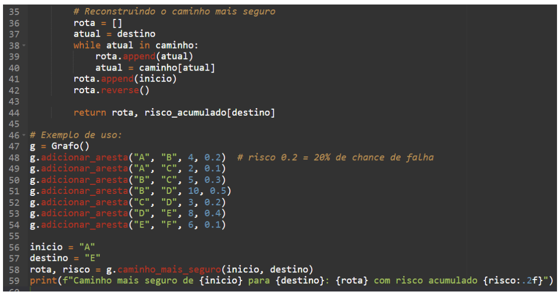
Além de otimizar o custo ou a distância, é fundamental considerar a segurança da rede, priorizando rotas com menores riscos de falhas. Fatores como danos aos cabos, condições climáticas adversas e tráfego intenso de veículos podem afetar as conexões de fibra. Esses riscos podem ser representados no grafo como pesos adicionais nas arestas.

Ao ajustar o algoritmo de Dijkstra para incluir esses riscos nas conexões, conseguimos identificar o caminho mais seguro, reduzindo as probabilidades de falhas na transmissão de dados.

- **Objetivo:** Minimizar o risco de falhas nas conexões de fibra óptica.

```python
import heapq

class Grafo:
    def __init__(self):
        self.grafo = {}

    def adicionar_aresta(self, u, v, custo, risco):
        if u not in self.grafo:
            self.grafo[u] = []
        if v not in self.grafo:
            self.grafo[v] = []
        self.grafo[u].append((v, custo, risco))
        self.grafo[v].append((u, custo, risco))  # Grafo não-direcionado

    def caminho_mais_seguro(self, inicio, destino):
        fila_prioridade = []
        heapq.heappush(fila_prioridade, (0, inicio))  # (risco acumulado, nó)
        risco_acumulado = {nodo: float("inf") for nodo in self.grafo}
        risco_acumulado[inicio] = 0
        caminho = {}

        while fila_prioridade:
            risco_atual, nodo_atual = heapq.heappop(fila_prioridade)

            if nodo_atual == destino:
                break  # Encontramos o destino

            for vizinho, _, risco in self.grafo[nodo_atual]:
                novo_risco = risco_atual + risco
                if novo_risco < risco_acumulado[vizinho]:
                    risco_acumulado[vizinho] = novo_risco
                    heapq.heappush(fila_prioridade, (novo_risco, vizinho))
                    caminho[vizinho] = nodo_atual  # Registrar o caminho
```

Figura 6: *Imagem criada pelo próprio autor.*



```
        # Reconstruindo o caminho mais seguro
        rota = []
        atual = destino
        while atual in caminho:
            rota.append(atual)
            atual = caminho[atual]
        rota.append(inicio)
        rota.reverse()

        return rota, risco_acumulado[destino]

# Exemplo de uso:
g = Grafo()
g.adicionar_aresta("A", "B", 4, 0.2)  # risco 0.2 = 20% de chance de falha
g.adicionar_aresta("A", "C", 2, 0.1)
g.adicionar_aresta("B", "C", 5, 0.3)
g.adicionar_aresta("B", "D", 10, 0.5)
g.adicionar_aresta("C", "D", 3, 0.2)
g.adicionar_aresta("D", "E", 8, 0.4)
g.adicionar_aresta("E", "F", 6, 0.1)

inicio = "A"
destino = "E"
rota, risco = g.caminho_mais_seguro(inicio, destino)
print(f"Caminho mais seguro de {inicio} para {destino}: {rota} com risco acumulado {risco:.2f}")
```

Figura 7: *Imagem criada pelo próprio autor.*

Conclusão

A representação de redes de fibra óptica por meio de grafos permite uma gestão mais eficiente da infraestrutura de comunicação, reduzindo custos e melhorando a performance na transmissão de dados. A aplicação de algoritmos como Prim, Kruskal e Dijkstra facilita o planejamento da distribuição dos cabos, evitando desperdícios e garantindo conexões mais robustas. Além disso, ao incluir o risco nas rotas, é possível aumentar a segurança e a confiabilidade da rede. No futuro, desafios como a manutenção preditiva e a otimização dinâmica da rede poderão ser abordados por meio de técnicas de Inteligência Artificial e aprendizado de máquina.

Referências

CORMEN, T. H.; LEISERSON, C. E.; RIVEST, R. L.; STEIN, C. *Algoritmos: Teoria e Prática*. 3. ed. Rio de Janeiro: Elsevier, 2012.

SEDGEWICK, R.; WAYNE, K. *Algorithms*. 4. ed. Addison-Wesley, 2011.

NETWORK optimization and graph theory. Disponível em: https://www.sciencedirect.com. Acesso em: 30 mar. 2025.

INTRODUCTION to graph algorithms for network routing. Disponível em: https://www.geeksforgeeks.org. Acesso em: 30 mar. 2025.

OTIMIZAÇÃO de redes de fibra óptica com algoritmos de grafos. Disponível em: https://ieeexplore.ieee.org. Acesso em: 30 mar. 2025.