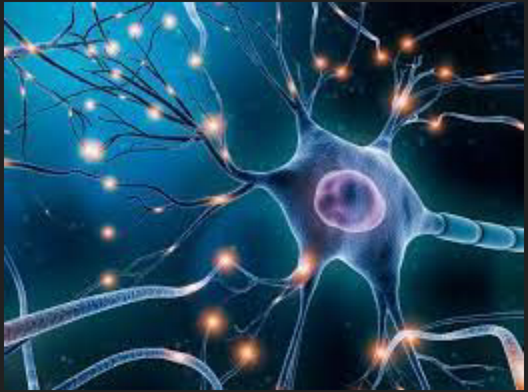
Árvores Geradoras de Tamanho Mínimo: Algoritmos de Kruskal e Prim

Thiago da Mota Souza

Árvores Geradoras Mínimas: motivação

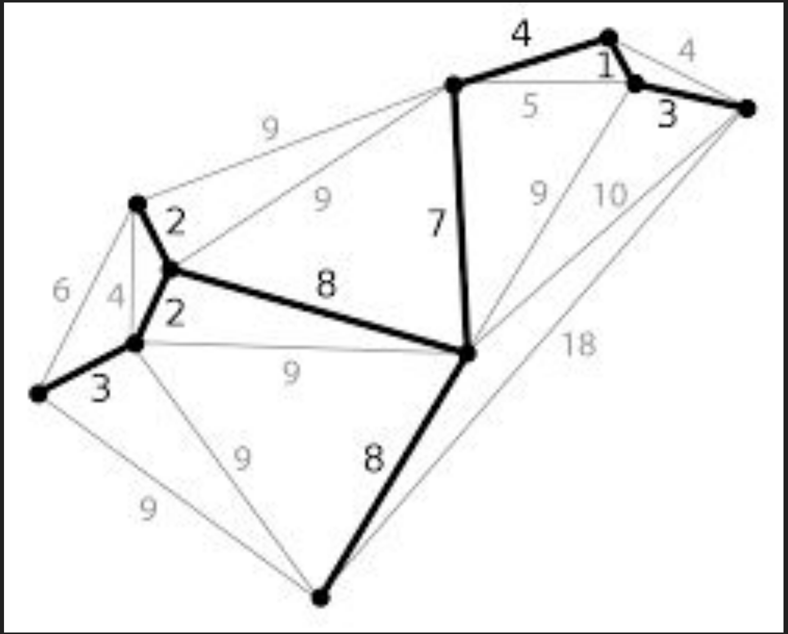


Fonte: google

Sumário

1. O que são Árvores Mínimas
2. Os pappers de Kruskal e Prim
3. Algoritmos
4. Experimentos para mediação da complexodade

Definições: Árvores Geradoras e Árvores Geradoras Mínimas

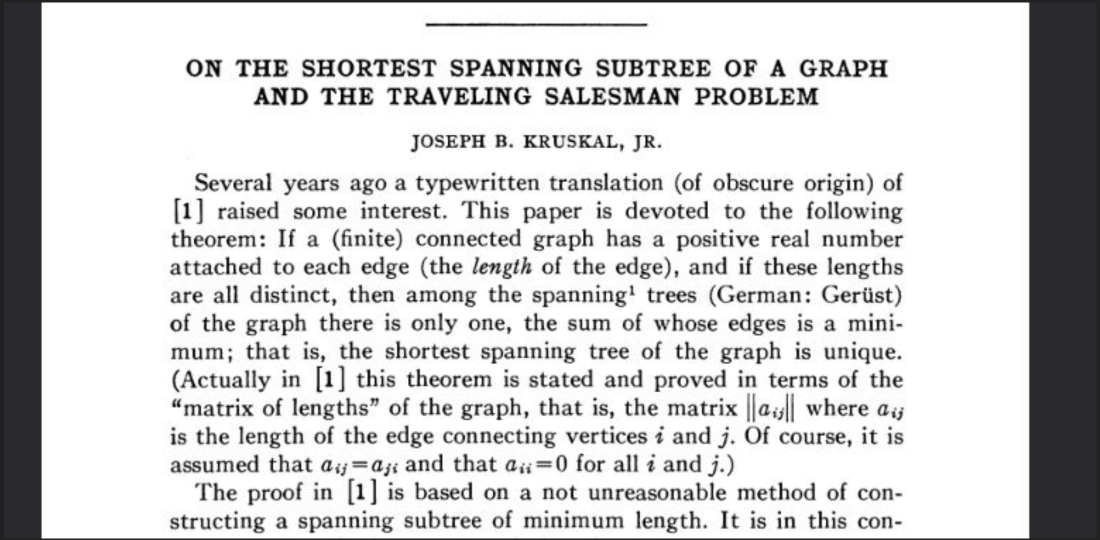


Wikipedia, the free encyclopedia. Minimum spanning 284 tree. Online: accessed December 2, 2023

T (V', E') é uma árvore geradora do grafo G (V, E) se, e somente se: **T é uma árvore e contém todos os vértices de G.**

T é uma árvore geradora de tamanho mínimo se, e somente se: dados os pesos w(E), **T é a árvore cuja soma dos pesos das arestas é mínima**

Algoritmo: Kruskal



ON THE SHORTEST SPANNING SUBTREE OF A GRAPH AND THE TRAVELING SALESMAN PROBLEM

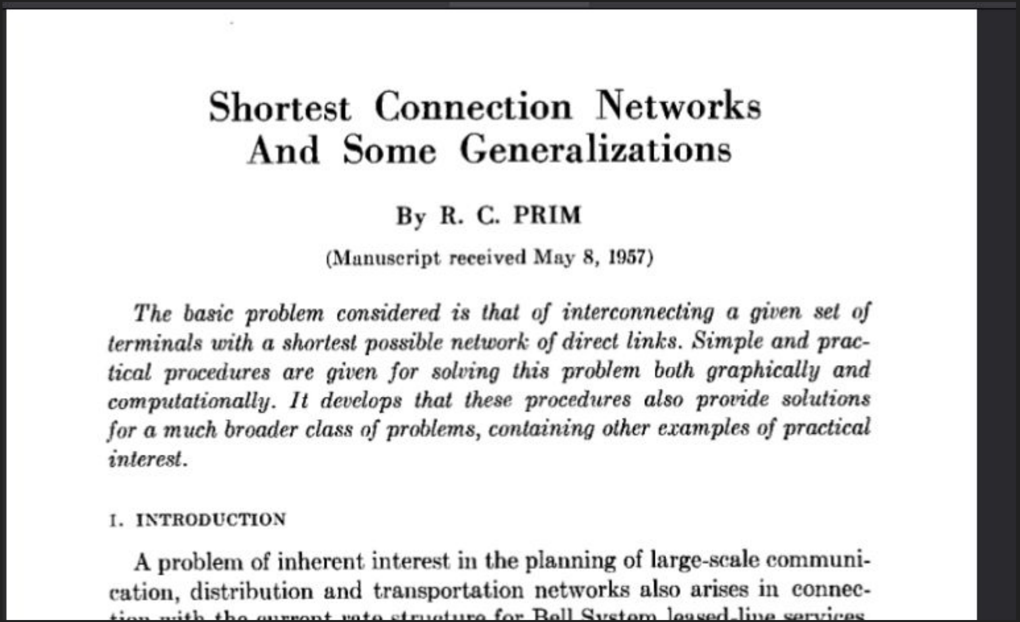
JOSEPH B. KRUSKAL, JR.

Several years ago a typewritten translation (of obscure origin) of [1] raised some interest. This paper is devoted to the following theorem: If a (finite) connected graph has a positive real number attached to each edge (the *length* of the edge), and if these lengths are all distinct, then among the spanning[1] trees (German: Gerüst) of the graph there is only one, the sum of whose edges is a minimum; that is, the shortest spanning tree of the graph is unique. (Actually in [1] this theorem is stated and proved in terms of the "matrix of lengths" of the graph, that is, the matrix $\|a_{ij}\|$ where a_{ij} is the length of the edge connecting vertices i and j. Of course, it is assumed that $a_{ij}=a_{ji}$ and that $a_{ii}=0$ for all i and j.)

The proof in [1] is based on a not unreasonable method of constructing a spanning subtree of minimum length. It is in this con-

Kruskal, Joseph B. "On the Shortest Spanning Subtree of a Graph and the Traveling Salesman Problem." *Proceedings of the American Mathematical Society* 7.1 (1956): 48-50. Web.

Algoritmo de Prim



Shortest Connection Networks And Some Generalizations

By R. C. PRIM

(Manuscript received May 8, 1957)

The basic problem considered is that of interconnecting a given set of terminals with a shortest possible network of direct links. Simple and practical procedures are given for solving this problem both graphically and computationally. It develops that these procedures also provide solutions for a much broader class of problems, containing other examples of practical interest.

I. INTRODUCTION

A problem of inherent interest in the planning of large-scale communication, distribution and transportation networks also arises in connection

Prim, R. C. "Shortest Connection Networks and Some Generalizations." *Bell System Technical Journal* 36.6 (1957): 1389-401. Web.

Meta Algoritmo

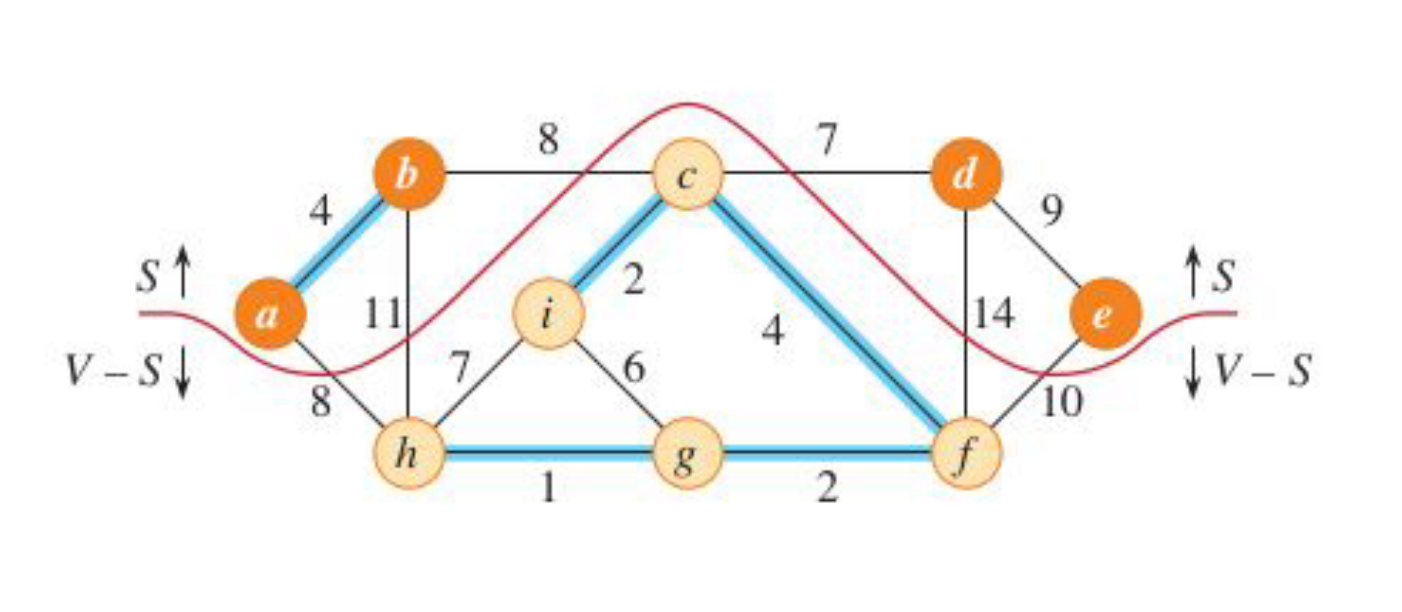
```
GENERIC-MST(G, w)
1  A = Ø
2  while A does not form a spanning tree
3      find an edge (u, v) that is safe for A
4      A = A ∪ {(u, v)}
5  return A
```

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2022). Introduction to Algorithms, fourth edition

Invariante:
Em cada execução do loop 2-4, **A é um subconjunto de uma árvore geradora** de peso mínimo e pela adição de uma **aresta segura** continua a ser

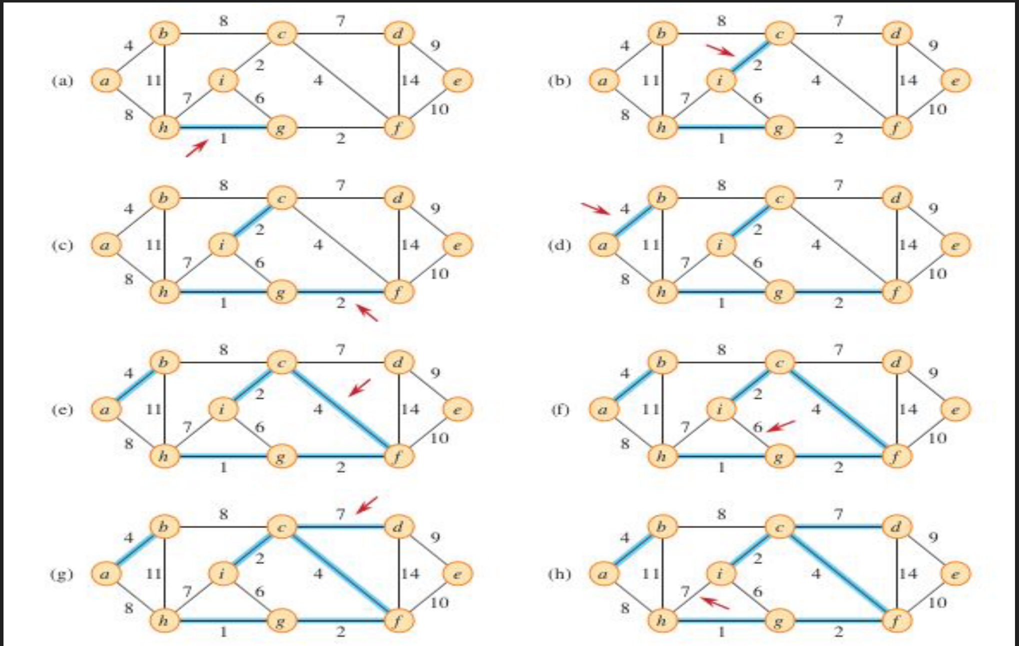
Coração dos algoritmos: Como achar uma aresta segura?

Teorema 21.1: Arestas Seguras de peso mínimo



Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2022). *Introduction to Algorithms, fourth edition*. p(588)

Algoritmo de Kruskal



Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2022). *Introduction to Algorithms, fourth edition*. p(593)

Inicializar o Meta-algoritmo com uma **floresta geradora em sem arestas.**

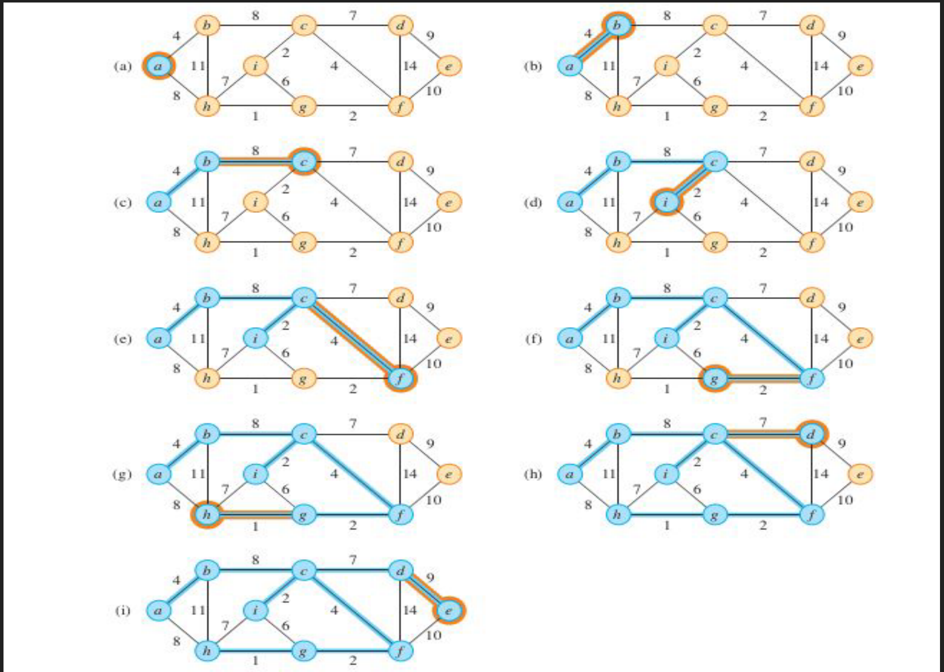
Aresta segura: a mais leve que conecta dois componentes da floresta

Esse algoritmo vai conectando os componentes da floresta a cada passo até que se tenha o menor número de componentes possível.

Implantação kruskal: Notas: detalhes da implantação

```
def kruskal_mst(self, roo_id: int = 0, weight_property = "weight") -> TGraph:

    mst = TGraph(num_vertices=self.get_num_vertices())
    mst.copy_nodes_properties(self)

    node_component_map = {v._id: i for i, v in enumerate(mst._V)}
    forest = [set([v._id]) for v in mst._V]

    edges = self.get_list_of_edges()

    sorted_edges = sorted(edges, key=lambda x: x[2][1][weight_property])

    for u_id, e_id, edge in sorted_edges:
        component_u = node_component_map[u_id]
        component_v = node_component_map[e_id]
        if component_u != component_v:
            mst.add_edge(u_id, e_id, edge[1])
            forest[component_u] = forest[component_u].union(forest[component_v])
            for v_id in forest[component_v]:
                node_component_map[v_id] = component_u
            forest[component_v] = set()

    return mst
```

Algoritmo de Prim



Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2022). *Introduction to Algorithms, fourth edition*. p(595)

Inicializar o Meta-algoritmo com uma **árvore sem arestas, isto é, escolher um nó.**

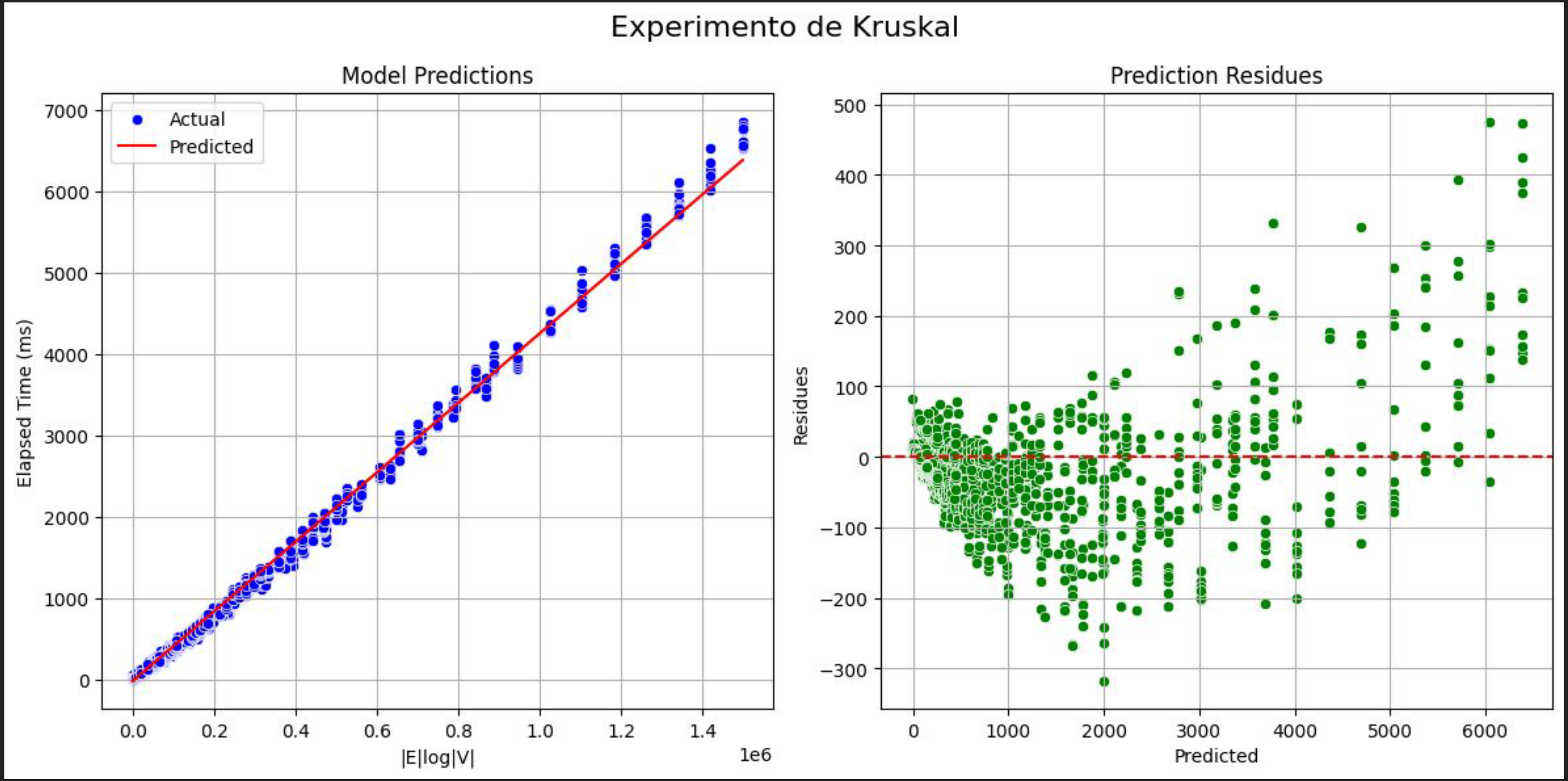
Aresta segura: a mais leve que **conecta a árvore a um novo nó**

Esse algoritmo vai conectando os nós formando uma única árvore.

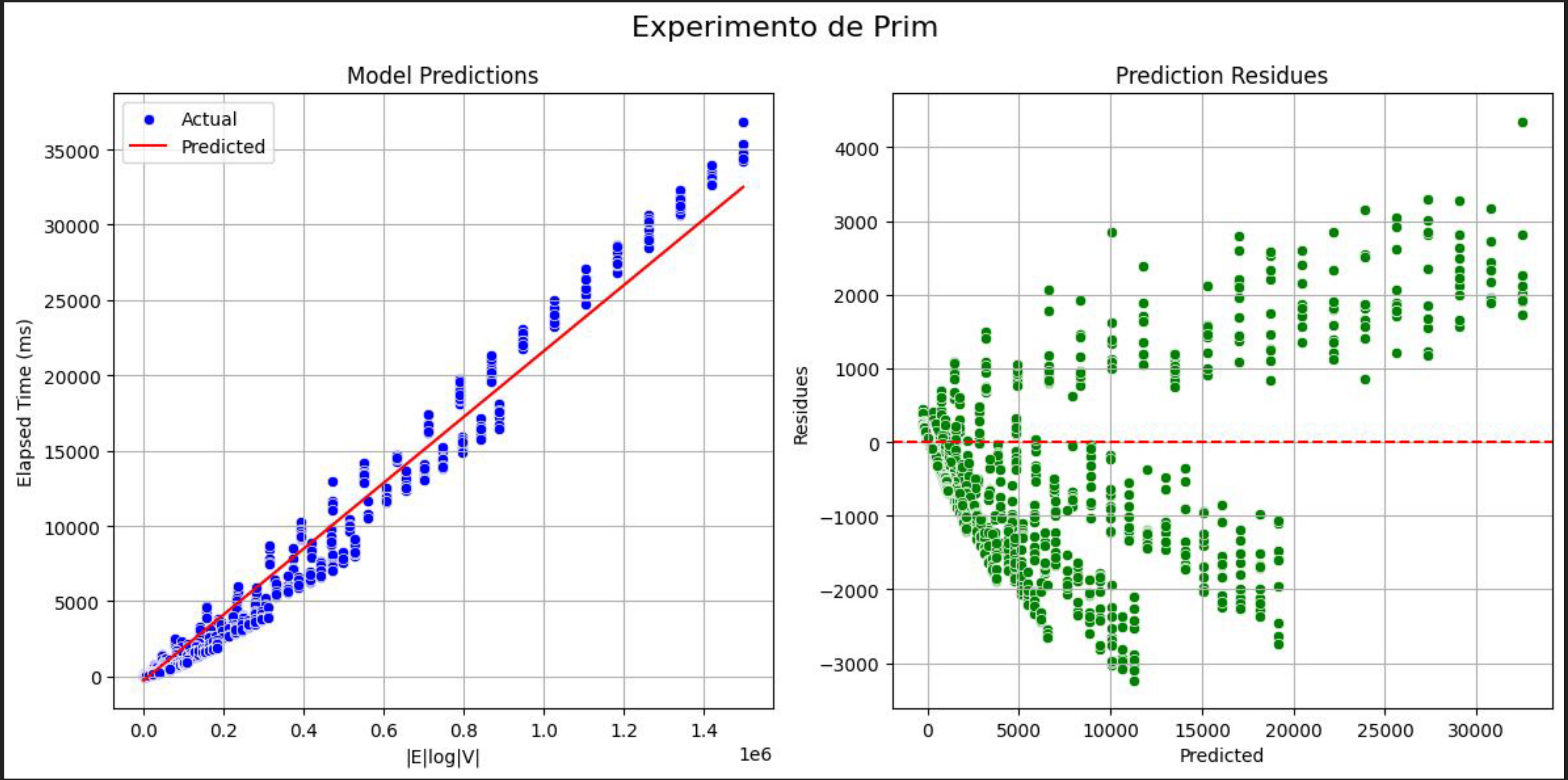
Implementação Prim: detalhes da implantação

```python
while len(Q) > 0:

    # get node with min edge weight crossing the cut
    u = min(Q, key=lambda x: x['key'])

    # add safe edge to tree, must guard against root on first loop
    if u['pi'] is not None:
        edge_propeties = self.get_edge(u['pi'], u._id)[1]
        mst.add_edge(u['pi'], u._id, edge_propeties)

    # update edges weights == keys crossing the cut via u
    for edge in self._E[u._id]:
        v_id = edge[0]._id
        v = mst.get_node(v_id)
        q_ids = [n._id for n in Q]

        if v._id in q_ids and edge.get_property(weight_property) < v['key']:
            v['pi'] = u._id
            v['key'] = edge.get_property(weight_property)

    # remove u from Q
    Q = [n for n in Q if n._id != u._id]

return mst
```

Experimentos



Experimentos



Relatório

Algoritmos de Kruskal e Prim para busca de árvores geradoras de tamanho mínimo: relatório final da disciplina GA-026

Thiago da Mota Souza
thiagoms@posgrad.lncc.br

Resumo

Grafos fornecem modelos úteis para abordagem de problemas em diversas áreas do conhecimento e da industria: de cadeias de logística à pesquisa de neurociência, de forma que a solução de problemas abstratos em grafos podem transladar em soluções para problemas importantes. Em particular, o problema de se encontrar AGTM (Árvores Geradoras de Tamanho Mínimo) em grafos era de fundamental importância para a indústria de telecomunicações que precisava fornecer telefonia fixa para consumidores dispersos geograficamente, pois a sua solução permitiria a minimização dos custos com o cabeamento. Neste contexto, Joseph Kruskal e Robert Prim, dois engenheiros trabalhando para a Bell Labs nos anos 50 publicaram artigos fundamentais para a ciência da computação [1][2] que são estudados em cursos de algoritmos até os dias de hoje. Neste artigo, apresentado como parte da avaliação da disciplina GA-026 do Laboratório Nacional de Computação Científica, serão conduzidos experimentos computacionais a fim de verificar as previsões teóricas quanto a ordem de crescimento do tempo de processamento dos algoritmos propostos por estes autores.

1 Introdução

Grafos são estruturas definidas por dois conjunto, nós (V) e arestas (E), em que as arestas representam relações entre os nós. Atribuindo-se atributos a arestas e nós, essas estruturas tornam-se poderosos mo-

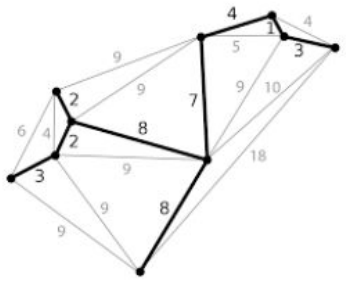


Figura 1: Exemplo de uma AGTM, em negrito, para um grafo e uma função de pesos, representados na arestas do grafo.

a rede deveria:

1. Conter todos os nós, lares e hubs.
2. Conter um conjunto de arestas, cuja soma dos pesos fosse mínima.

Os subgrafos que atendem a propriedade 1 listadas são chamados de Árvores Geradoras (AG) das quais as que atendem a propriedade 2 são chamadas Árvores Geradoras de Tamanho Mínimo(AGTM), ou, simples-

Conclusão

1. Verificamos que os algoritmos de Prim e Kruskal tem **tempo de execução com complexidade O(|E| log |V|)**, em grafos artificialmente criados por um processo uniforme.

2. Fornecemos uma implantação Python para ambos os algoritmos no github. Que contém os códigos e um sandbox para executar os experimentos
.

3. Discutimos detalhes relevantes a implementação desse algoritmos em Python

4. Para discussões mais aprofundadas, ver o relatório.

Obrigado