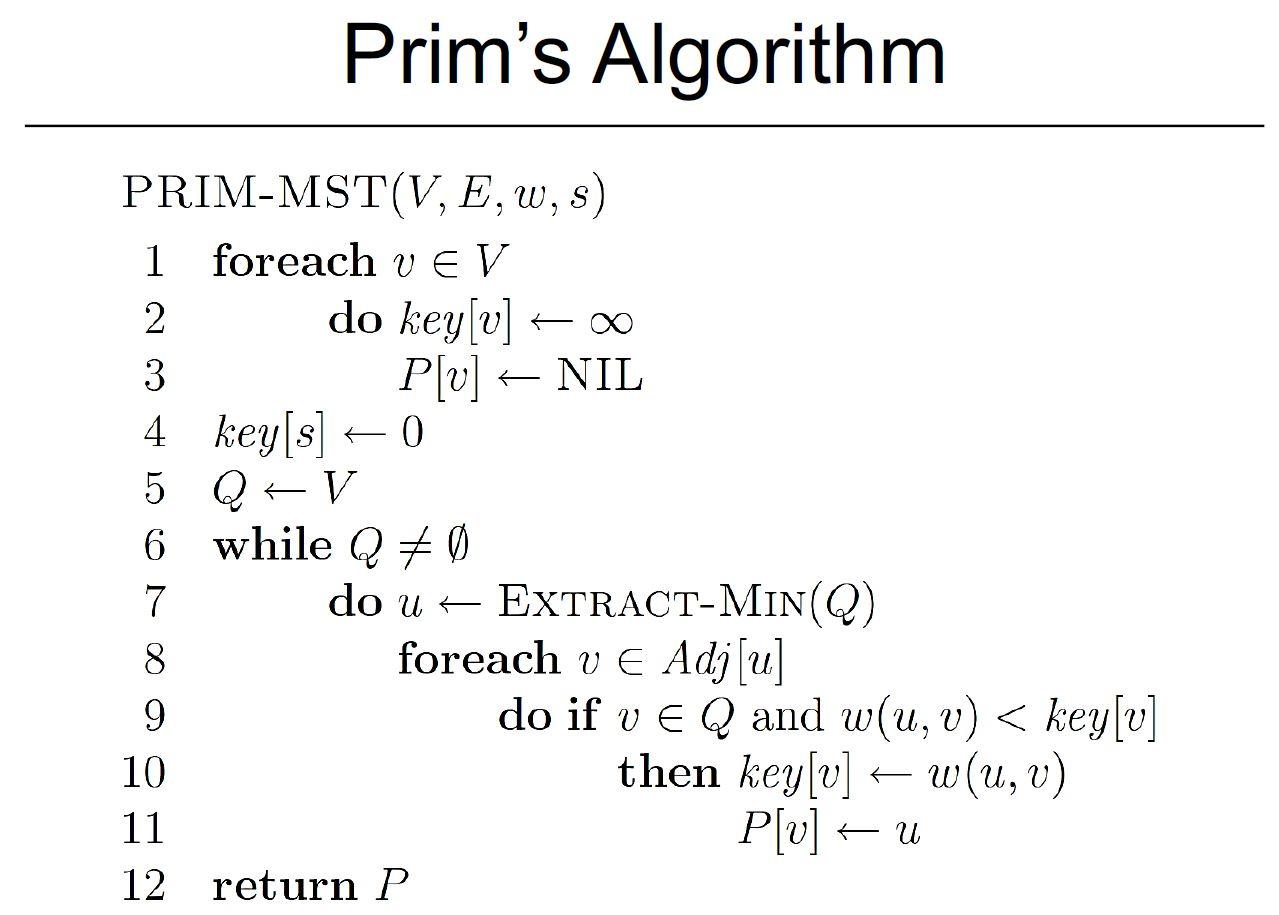
## شرح للخوارزمية برايم وكيفية عملها

خوارزمية برايم (Prim Algorithm) هي خوارزمية تستخدم لإيجاد أقل مسار عظمى في الشبكة. يعتمد عمل هذه الخوارزمية على المبدأ البسيط للبحث عن الروابط ذات التكلفة الأدنى في كل مرحلة وإضافتها إلى الشبكة العظمى الأدنى.



وفيما يلي شرح لكيفية عمل خوارزمية برايم:

البداية:

يتم اختيار عقد بداية في الشبكة ليكون نقطة بداية للبحث. يمكن اختيار أي عقد كنقطة بداية.

المرحلة الأولى:

يتم اختيار أقل رابط يربط العقد البداية بأحد العقد المتاحة الأخرى.

يتم إضافة هذا الرابط إلى الشبكة العظمى الأدنى.

المرحلة التالية:

يتم البحث في جميع الروابط المتاحة التي تربط العقد المضاف حديثًا إلى الشبكة بالعقد غير المضافة.

يتم اختيار أقل رابط من بين هذه الروابط وإضافته إلى الشبكة العظمى الأدنى.

المراحل التالية:

يتم تكرار الخطوات السابقة حتى يتم تضمين جميع العقد في الشبكة العظمى الأدنى.

الانتهاء:

يتم الانتهاء عندما يتم تضمين جميع العقد في الشبكة العظمى الأدنى، ويتم الحصول بذلك على أقل مسار عظمى يربط جميع العقد بأقل تكلفة ممكنة.

بهذه الطريقة، تقوم خوارزمية برايم ببناء الشبكة العظمى الأدنى خطوة بخطوة، حيث تضيف الروابط بشكل تدريجي حتى يتم تضمين جميع العقد في الشبكة بأقل تكلفة. وبفضل هذه الخواص، تعتبر خوارزمية برايم أداة قيمة في تصميم الشبكات وتوفير البنية التحتية بأقل تكلفة ممكنة.

## الكود البرمجي

from collections import defaultdict

class Graph:

    def \_\_init\_\_(self):

        self.graph = defaultdict(list)

    def add\_edge(self, u, v, w):

        self.graph[u].append((v, w))

        self.graph[v].append((u, w))

    def prim\_mst(self):

        visited = set()

        mst = []

        start\_vertex = next(iter(self.graph))

        visited.add(start\_vertex)

        while len(visited) < len(self.graph):

            min\_edge = None

            for vertex in visited:

                for neighbor, weight in self.graph[vertex]:

                    if neighbor not in visited and (min\_edge is None or weight < min\_edge[1]):

                        min\_edge = (neighbor, weight, vertex)

            mst.append(min\_edge)

            visited.add(min\_edge[0])

        return mst

    def total\_cost(self, mst):

        return sum(edge[1] for edge in mst)

*# Define the graph*

g = Graph()

g.add\_edge('San Francisco', 'New York', 2000)

g.add\_edge('San Francisco', 'Chicago', 1200)

g.add\_edge('San Francisco', 'Denver', 900)

g.add\_edge('San Francisco', 'Atlanta', 2200)

g.add\_edge('Chicago', 'Denver', 1300)

g.add\_edge('Chicago', 'New York', 1000)

g.add\_edge('Chicago', 'Atlanta', 700)

g.add\_edge('New York', 'Denver', 1600)

g.add\_edge('New York', 'Atlanta', 800)

g.add\_edge('Denver', 'Atlanta', 1400)

*# Compute Minimum Spanning Tree using Prim's Algorithm*

mst = g.prim\_mst()

*# Print the Minimum Spanning Tree edges and total cost*

print("Minimum Spanning Tree:")

for edge in mst:

    print(edge[2], '->', edge[0], '=', edge[1], '$')

total\_cost = g.total\_cost(mst)

print("Total cost of the Minimum Spanning Tree:", total\_cost, "$")

## النتائج

