YILDIZ TEKNİK ÜNİVERSİTESİ
BİLGİSAYAR MÜHENDİSLİĞİ

Ad: Emir

Soyad: Oğuz

Öğrenci No: 20011059

Öğrenci E-Postası: emir.oguz1@std.yildiz.edu.tr

BLM1012 - YAPISAL PROGRAMLAMAYA GİRİŞ FİNAL PROJESİ

PRİM ALGORİTMASI

Ders Yürütücüsü

Doç. Dr. Mehmet Fatih Amasyalı

Haziran, 2022

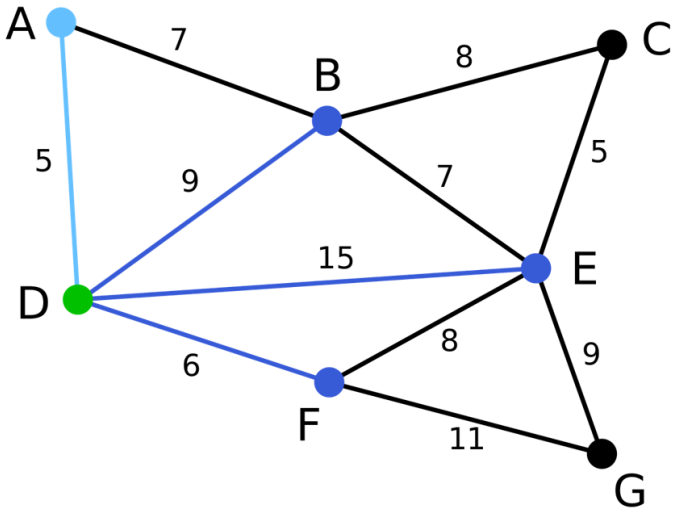
- İÇERİK

- Prim Algoritması nedir? Ne işe yarar? Nasıl çalışır?
- Kullanım Yerleri
- Karmaşıklığı
- Avantajları - Dezavantajları
- Rakipleri ve Karşılaştırmalar
- C Dilindeki Kodu
- Kodun Ekran Çıktıları
- Kaynaklar

- VİDEO ADRESİ

- https://youtu.be/pvLsWHIlL4E

PRIM'S
ALGORITHM



1. Prim Algoritması nedir? Ne işe yarar? Nasıl çalışır?

- ## Prim Algoritması nedir? Ne işe yarar?

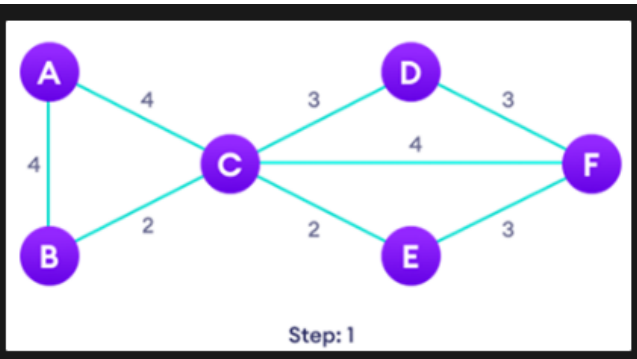
- Bilgisayar bilimlerinde, Prim Algoritması, Greedy Algoritmaları sınıfına girer ve ağırlıklandırılmış, bağlı bir çizge üzerinde minimum kapsayan ağaç (minimum spanning tree) problemine çözüm bulma algoritmalarından birisidir.
- Prim Algoritması, bulunduğu düğümle bağlantısı olan komşu düğümlerden en yakın düğümü bünyesine katarak ilerler. Bünyesine kattığı düğüm ile bir sistem oluşturur ve bu sisteme en yakın düğümü bünyesine katarak minimum kapsayan ağacı işlemeye devam eder. Bu sayede toplam yol maliyetini en aza indirerek tüm düğümleri gezmiş olur.

- ## Prim Algoritması nasıl çalışır?

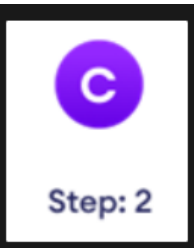
  - *Algoritmanın İşleyişi*

1. Minimum kapsayan ağacın işleyişini başlatmak için düğümler arasından rastgele bir tepe noktası seçilir
2. Seçilen tepe noktasına bağlantılı olan çevre düğümler kontrol edilir. En kısa bağlantıya sahip düğüm seçilir ve bu düğüm ağaç sistemine katılır.
3. Bünyeye katılan yeni düğümler ile oluşan sistem üzerinden tüm düğümlere ulaşılıp minimum kapsayan ağaç elde edilene kadar ikinci adım tekrarlanır.

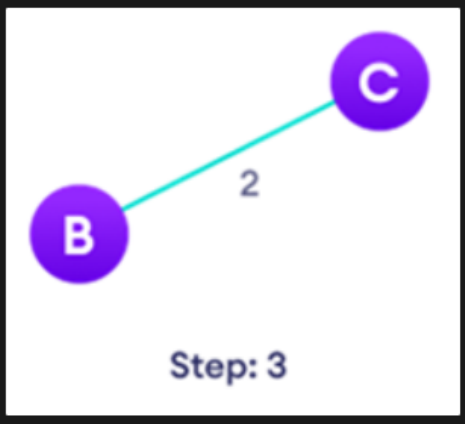
- *Adımların Görselleştirilmiş Hali*



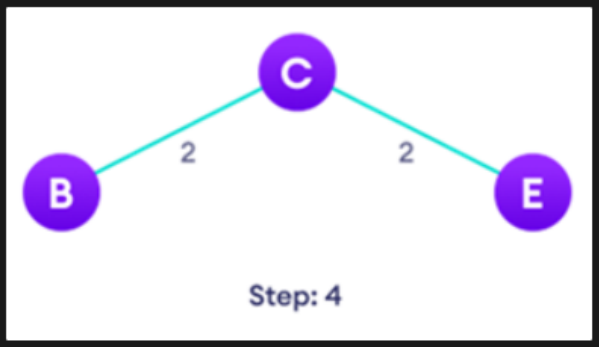
Bir çizge (graf) veri yapısı ile işleme başlanır.



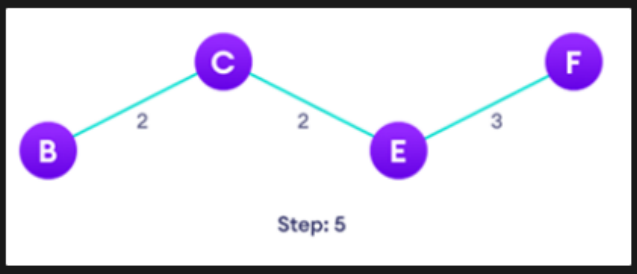
Rastgele bir tepe noktası seçilir.



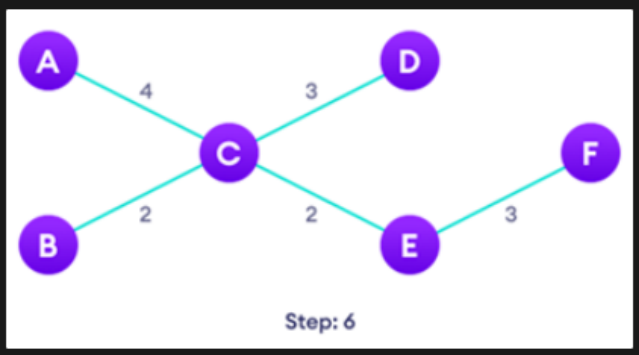
Bu tepe noktasına en kısa yoldan bağlantılı olan düğüm seçilir, sisteme eklenir.



Sisteme bağlantılı ve henüz sistemde bulunmayan en yakın düğüm seçilir.



Dördüncü adım tekrarlanır. En yakın düğüm için aynı mesafeye sahip birden fazla seçenek varsa rastgele biri seçilir.



Tüm düğümlere ulaşılıp minimum bir tarama ağacı oluşturulana kadar bu adımlar tekrar edilir ve sonuca ulaşılır.

- *Bir Graf Veri Yapısının Matris ile Gösterimi ve Prim Algoritması ile Çözümü (Örnek Çıktı)*

```
Girdiginiz graf yapisinin matris hali:
       (0)     (1)     (2)     (3)     (4)

(0)      0       2       0       5       0

(1)      2       0       3       8       6

(2)      0       3       0       0       7

(3)      5       8       0       0       9

(4)      0       6       7       9       0

 Dugumler       Mesafe
 0 ---> 1         2

 1 ---> 2         3

 0 ---> 3         5

 1 ---> 4         6
Toplam mesafe = 16
```

2. Kullanım Yerleri

- Prim Algoritması, yukarıda bahsedildiği gibi minimum kapsayan ağaç problemine çözüm sunmaktadır. Minimum kapsayan ağaçları içeren algoritmalar; bilgisayar ağları, telekomünikasyon ağları, ulaşım ağları, su tedarik ağları ve elektrik şebekeleri gibi ağ yapılarında kullanılır.
- Prim algoritmasının genel olarak kullanıldığı uygulamalar:
  - Tüm şehirleri birbirine bağlayan yol ve demiryolu hatları için oluşturulan ağlar
  - Yapay zeka uygulamalarında kullanılan yol bulma çözümleri
  - Google Maps, Google Search vb. uygulamalar
  - Gezgin Satıcı Problemi
  - Oyun Geliştirme
  - Bilişsel Bilim

3. Karmaşıklığı

Minimum edge weight data structure	Time complexity (total)
adjacency matrix, searching	$O(\|V\|^2)$
binary heap and adjacency list	$O((\|V\| + \|E\|) \log \|V\|) = O(\|E\| \log \|V\|)$
Fibonacci heap and adjacency list	$O(\|E\| + \|V\| \log \|V\|)$

- Prim Algoritması'nın karmaşıklığı, çözüm yollarına göre değişiklik göstermektedir. Yukarıdaki tabloda bu durum gösterilmiştir. *"V", graf üzerindeki tepe noktası (düğüm) sayısını, "E" ise graf üzerindeki kenar sayısını, yani tepe noktaları arasındaki yol sayısını ifade eder.*

- Komşuluk Matrisi (Adjacency Matrix) yolu uygulanmış algoritmanın karmaşıklığı, $O(|V|^2)$ olarak hesaplanır. *[Proje içeriğindeki kodda bu yol kullanılmıştır.]*

- İkili Yığın (Binary Heap) yolu uygulanmış algoritmanın karmaşıklığı, $O\big((|V| + |E|)log\ |V|\big) = O(|E|log\ |V|)$ olarak hesaplanır.

- Fibonacci Yığını (Fibonacci Heap) yolu uygulanmış algoritmanın karmaşıklığı, $O(|E| + |V|log\ |V|)$ olarak hesaplanır.

4. Avantajları – Dezavantajları

• Avantajları

- Çok sayıda kenarı olan yoğun graf yapılarının çözümünde etkilidir.
- Komşuluk Matrisi, İkili Yığın ve Fibonacci Yığını olmak üzere üç farklı yol ile uygulanabilir.
- Fibonacci Yığını ile kodlanmış halinin zaman karmaşıklığı rakiplerine göre daha iyidir.

• Dezavantajları

- Minimum kapsayan ağaç üzerinde güçlü bir kontrole sahip değildir.
- Üzerinde işlem yapılan graf yapısının içeriğine yeni kenar eklendiği zaman, kenar listesi için tekrar baştan bir arama yapar.
- Bir graf veri yapısı üzerinde aynı değere sahip birden fazla kenar varsa, aynı değerli minimum değere sahip kenarlarda fazladan kontrol yapılmasına izin vermez.

5. Rakipleri ve Karşılaştırmalar

- Kruskal ve Dijkstra algoritmaları, Prim Algoritması'nın rakipleridir. Şimdi sırasıyla bu algoritmaları Prim Algoritması ile karşılaştıralım.

Prim Algoritması	Kruskal Algoritması
Minimum kapsayan ağacı, graf üzerindeki herhangi bir tepe noktasından başlayarak oluşturmaya başlar.	Minimum kapsayan ağacı, graf üzerindeki minimum yola bağlı olan bir tepe noktasından oluşturmaya başlar.
Minimum mesafeyi elde etmek için bir düğüm üzerinde birden fazla kez işlem yapar.	Bir düğüm üzerinde yalnız bir kez işlem yapar.
En kötü senaryoda $O(V^2)$ zaman karmaşıklığına sahiptir, Fibonacci Yığını kullanılarak $O(\|E\| + \|V\| log \|V\|)$ değerine kadar geliştirilebilir.	En kötü senaryoda $O(E + \|E\| log \|E\|)$ zaman karmaşıklığına sahiptir, $O(\|E\| log \|V\|)$ değerine kadar geliştirilebilir.
Bağlı bileşenler üretir ve yalnızca bağlı graf yapısı üzerinde çalışır.	Herhangi bir anda orman yapısı (bağlantısız bileşenler) üretebilir veya bunlar üzerinde çalışabilir.
Yoğun graf yapılarında (dense graphs) daha hızlı çalışır.	Seyrek graf yapılarında (sparse graphs) daha hızlı çalışır.
İlk tepe noktasından başlayarak minimum yayılan ağacı oluşturur.	En az ağırlıklı kenardan başlayarak minimum yayılan ağacı oluşturur.
Uygulama alanları: Gezgin Satıcı Problemi, Karayolları ağı ve tüm şehirleri birbirine bağlayan Demiryolu hatları vb.	***Uygulama alanları:*** LAN bağlantısı, TV Ağı vb.

Prim Algoritması	Dijkstra Algoritması
Minimum kapsayan ağacı bulur.	En kısa yolu bulur.
Sadece yönsüz graf yapıları üzerinde çalışır.	Hem yönlendirilmiş hem de yönlendirilmemiş graf yapıları üzerinde çalışabilir.
Negatif kenar ağırlıklarını işleyebilir.	Yapı içerisinde bir veya daha fazla negatif kenar ağırlığı varsa mesafeleri doğru bir şekilde hesaplayamayabilir.
Birden çok noktayı birbirine bağlayan yolların yapımında malzeme maliyetlerini en aza indirmek için kullanılır.	Bir noktadan diğerine seyahat ederken zamandan ve yakıttan tasarruf etmek için kullanılır.

6. C Dilindeki Kod

```c
#include <stdio.h>
#include <stdbool.h>
#include <limits.h>
#include <sys/time.h>
#define N 20

//Graf yapisini kullanicidan matris seklinde alan fonksiyon
void readMatrix(int graph[N][N], int size)
{
  int i, j;

  printf("\nBir matris olusturacak sekilde graf uzerindeki mesafeleri giriniz...\n");
  for(i=0; i<size; i++){

            for(j=0; j<size; j++){
                      if(i < j){
                                printf("- Eleman[%d][%d] = ", i, j);
                                scanf("%d", &graph[i][j]);
                      }
                      else if(i == j){
                                graph[i][j] = 0; //Kosegen degerleri sifira esitlenir.
                                printf("- Eleman[%d][%d] = %d\n", i, j, graph[i][j]);
                      }
                      else{
                                graph[i][j] = graph[j][i]; //Kullanicinin girdigi elemanlarin degerleri, matris kosegenine gore simetrik olan elemanlara esitlenir.
                                printf("- Eleman[%d][%d] = %d\n", i, j, graph[i][j]);
                      }
            }

            printf("\n");
  }
}

//Kullanicinin girdigi matrisi yazdiran fonksiyon
void printMatrix(int graph[N][N], int size)
{
  int i, j;

  for(j=0; j<size; j++){
            printf("\t(%d)", j); //Sutun numaralari yazdirilir.
  }

  for(i=0; i<size; i++){
            printf("\n\n\n(%d)", i); //Satir numaralari yazdirilir.

            for(j=0; j<size; j++){
                      printf("\t %d", graph[i][j]); //Matrisin elemanlari yazdirilir.
            }
  }
}

//En kisa mesafeyi ve bu mesafenin bitis noktasi olan dizini bulan fonksiyon
int valueMST(int pathValue[], bool visited[], int size)
{
  int i, minIndex;
  int min = INT_MAX;

  for(i=0; i<size; i++){

            if(visited[i] == false && pathValue[i] < min){
                      min = pathValue[i]; //"min" degiskenine guncel olan en kisa mesafe degeri atanir.
                      minIndex = i; //"minIndex" degiskenine en kısa mesafeye sahip olan dizin degeri atanir.
            }
  }

  return minIndex;
}

//Gidilen yolda, baslangic-bitis dugumlerini ve mesafeyi yazdiran fonksiyon
void printMST(int parent[], int pathValue[], int size)
{
  int i, sum=0;

  printf("\n\n\n Dugumler\tMesafe");
  for(i=1; i<size; i++){
            printf("\n\n %d ---> %d\t  %d\n", parent[i], i, pathValue[i]); //Sirasiyla baslangic, bitis ve mesafe degerleri yazdirilir.
  }

```

```
79.    for(i=1; i<size; i++){
80.                sum += pathValue[i]; //Mesafe degerleri toplanir, toplam mesafe bulunur.
81.    }
82.
83.    printf("\nToplam mesafe = %d\n", sum);
84. }
85.
86. //Prim Algoritmasi'nin islendigi fonksiyon
87. int primsAlgorithm(int graph[N][N], int size)
88. {
89.    int i, j, edge, path, counter=0;
90.    int pathValue[N], parent[N]; //"pathValue" dizisi, yol degerlerini tutar, "parent" dizisi, guncel dizindeki dugumun
    ata dugumunu tutar.
91.    bool visited[N]; //"visited" dizisi, dugumlerin ziyaret edilip edilmedigi bilgisini tutar.
92.
93.    for(i=0; i<size; i++){
94.                visited[i] = false; //Baslangic icin tum dugumlerin ziyaret bilgisine '0' degerini atanir.
95.                pathValue[i] = INT_MAX; //Baslangic icin tum yol degerlerine en buyuk tam sayi degeri atanir.
96.    }
97.
98.    pathValue[0] = 0; //Tepe noktasi olan '0' dugumunun yol degerine '0' atanir. Cunku burasi baslangic noktasidir.
99.    parent[0] = -1; //Tepe noktasi olan dugumun ata dugumu olamaz. Bu nedenle bu noktanin ata dugum degerine '-1'
    atanir.
100.
101.   for(path=0; path<size; path++){
102.               edge = valueMST(pathValue, visited, size); //"valueMST" fonksiyonu cagirilir. "edge" degiskenine
    fonksiyonun dondurdugu "minIndex" degeri atanir.
103.               visited[edge] = true; //"edge" degerine atanan dugum numarasinin ziyaret bilgisine '1' degeri atanir.
104.
105.               for(j=0; j<size; j++){
106.                          counter++; //Zaman karmasikligi icin ic ice dongulere giris sayisi hesaplanir.
107.
108.                          if(graph[edge][j]!=0 && visited[j] == false && graph[edge][j] < pathValue[j]){
109.                                     pathValue[j] = graph[edge][j];
110.                                     parent[j] = edge;
111.                          }
112.               }
113.   }
114.
115.   printMST(parent, pathValue, size);
116.
117.   return counter;
118. }
119.
120. int main()
121. {
122.   int i, j, choice=1, process=0, size, graph[N][N], star[N], complexity[N];
123.   double time[N];
124.   struct timeval start, stop;
125.
126.   while(choice == 1){
127.               system("cls");
128.
129.               printf("--> PRIM ALGORITMASI");
130.
131.               printf("\n\n--> %d. ISLEM\n\n", process+1);
132.               printf("Graf uzerinde bulunan dugum (nokta) sayisini giriniz: ");
133.               scanf("%d", &size);
134.
135.               readMatrix(graph, size);
136.
137.               printf("\nGirdiginiz graf yapisinin matris hali:\n\n");
138.               printMatrix(graph, size);
139.
140.               gettimeofday(&start, NULL);
141.               complexity[process] = primsAlgorithm(graph, size);
142.               gettimeofday(&stop, NULL);
143.
144.               time[process] = (double)(stop.tv_usec - start.tv_usec) / 1000000 + (double)(stop.tv_sec - start.tv_sec);
145.               printf("\n%d. Islem icin Prim Algoritmasi'nin baslatilmasindan sonlanmasina kadar gecen sure: %lf
    saniye\n", process+1, time[process]);
146.               star[process] = (int)(time[process] * 15000);
147.
148.               printf("\n1) ISLEME BASTAN BASLA");
149.               printf("\n2) CIKIS");
150.               printf("\nSeciminiz: ");
151.               scanf("%d", &choice);
152.
153.               process++;
154.   }
155.
156.   for(i=0; i<process; i++){
157.               printf("\n%d. COZUM", i+1);
158.               printf("\n- Sure: %lf saniye", time[i]);
159.
160.               printf("\n- Sayaca Bagli Yildiz Gostergesi (N^2): ");
```

```
161.             for(j=0; j<complexity[i]; j++){
162.                       printf("*");
163.             }
164.             printf(" (%d Yildiz)", complexity[i]);
165.
166.             printf("\n- 'gettimeofday()' Fonksiyonuna Bagli Sapka Gostergesi: ");
167.             for(j=0; j<star[i]; j++){
168.                       printf("^");
169.             }
170.             printf(" (%d Sapka)", star[i]);
171.
172.             printf("\n");
173.   }
174.
175.   getch();
176.
177.   return 0;
178. }
```

7. Kodun Ekran Çıktıları

- Çıktı 1

```
--> PRIM ALGORITMASI

--> 1. ISLEM

Graf uzerinde bulunan dugum (nokta) sayisini giriniz: 4

Bir matris olusturacak sekilde graf uzerindeki mesafeleri giriniz...
- Eleman[0][0] = 0
- Eleman[0][1] = 4
- Eleman[0][2] = 7
- Eleman[0][3] = 0

- Eleman[1][0] = 4
- Eleman[1][1] = 0
- Eleman[1][2] = 3
- Eleman[1][3] = 6

- Eleman[2][0] = 7
- Eleman[2][1] = 3
- Eleman[2][2] = 0
- Eleman[2][3] = 2

- Eleman[3][0] = 0
- Eleman[3][1] = 6
- Eleman[3][2] = 2
- Eleman[3][3] = 0


Girdiginiz graf yapisinin matris hali:

        (0)     (1)     (2)     (3)


(0)      0       4       7       0


(1)      4       0       3       6


(2)      7       3       0       2


(3)      0       6       2       0
```

```
 Dugumler       Mesafe

 0 ---> 1         4


 1 ---> 2         3


 2 ---> 3         2

Toplam mesafe = 9

1. Islem icin Prim Algoritmasi'nin baslatilmasindan sonlanmasina kadar gecen sure: 0.000496 saniye
```

- Çıktı 2

```
--> PRIM ALGORITMASI

--> 2. ISLEM

Graf uzerinde bulunan dugum (nokta) sayisini giriniz: 5

Bir matris olusturacak sekilde graf uzerindeki mesafeleri giriniz...
- Eleman[0][0] = 0
- Eleman[0][1] = 2
- Eleman[0][2] = 0
- Eleman[0][3] = 6
- Eleman[0][4] = 0

- Eleman[1][0] = 2
- Eleman[1][1] = 0
- Eleman[1][2] = 3
- Eleman[1][3] = 8
- Eleman[1][4] = 5

- Eleman[2][0] = 0
- Eleman[2][1] = 3
- Eleman[2][2] = 0
- Eleman[2][3] = 0
- Eleman[2][4] = 7

- Eleman[3][0] = 6
- Eleman[3][1] = 8
- Eleman[3][2] = 0
- Eleman[3][3] = 0
- Eleman[3][4] = 9

- Eleman[4][0] = 0
- Eleman[4][1] = 5
- Eleman[4][2] = 7
- Eleman[4][3] = 9
- Eleman[4][4] = 0
```

```
Girdiginiz graf yapisinin matris hali:

       (0)     (1)     (2)     (3)     (4)

(0)      0       2       0       6       0

(1)      2       0       3       8       5

(2)      0       3       0       0       7

(3)      6       8       0       0       9

(4)      0       5       7       9       0

 Dugumler       Mesafe
 0 ---> 1         2

 1 ---> 2         3

 0 ---> 3         6

 1 ---> 4         5
Toplam mesafe = 16

2. Islem icin Prim Algoritmasi'nin baslatilmasindan sonlanmasina kadar gecen sure: 0.001487 saniye
```

- Çıktı 3 (İlk İki Çıktının Zaman Karmaşıklığı Karşılaştırması)

```
1. COZUM
- Sure: 0.000496 saniye
- Sayaca Bagli Yildiz Gostergesi (N^2): ********** (16 Yildiz)
- 'gettimeofday()' Fonksiyonuna Bagli Sapka Gostergesi: ^^^^^^^ (7 Sapka)

2. COZUM
- Sure: 0.001487 saniye
- Sayaca Bagli Yildiz Gostergesi (N^2): ************************* (25 Yildiz)
- 'gettimeofday()' Fonksiyonuna Bagli Sapka Gostergesi: ^^^^^^^^^^^^^^^^^^^^^^^^^ (22 Sapka)
```

- Kod içerisinde bulunan sayaç, algoritmanın bulunduğu fonksiyondaki döngülerde kullanılmıştır. Döngüye giriş miktarını hesaplar. Kıyaslama için tüm düğüm sayılarında net bir göstergedir.
- İşlemlerdeki süreler, Prim Algoritması'nın bulunduğu fonksiyona yönelik yerleştirilmiş "gettimeofday()" fonksiyonu ile hesaplanmıştır. Az sayıda düğüm sayısı için net bir karşılaştırma yolu değildir ancak bazı sonuçlar için referans noktası olabilir.

8. Kaynaklar

- https://tr.wikipedia.org/wiki/Prim_algoritmas%C4%B1
- https://en.wikipedia.org/wiki/Prim%27s_algorithm
- https://www.programiz.com/dsa/prim-algorithm
- https://www.geeksforgeeks.org/prims-minimum-spanning-tree-mst-greedy-algo-5/
- https://www.geeksforgeeks.org/difference-between-prims-and-kruskals-algorithm-for-mst/
- https://www.baeldung.com/cs/kruskals-vs-prims-algorithm
- https://www.baeldung.com/cs/prim-dijkstra-difference
- https://medium.com/@mitanshupbhoot/comparative-applications-of-prims-and-kruskal-s-algorithm-in-real-life-scenarios-4aa0f92c7abc