// LINEAR SEARCH (Doğrusal Arama)

// Küçük veri kümeleri için uygundur, sıralı olup olmaması önemli değildir.

// Baştan sona her elemanı kontrol eder. Aranan bulunursa indeksini döner, yoksa -1.

Dizi n elemanlı. En kötü durumda aranan eleman son sırada ya da hiç yok. Dolayısıyla tek tek tüm elemanlara bakmak gerekir. n kez kontrol. Bu nedenle O(n)

static int DogrusalArama(int[] dizi, int aranan)

{

for (int i = 0; i < dizi.Length; i++)

{

if (dizi[i] == aranan)

return i; // Aranan bulundu, indeks döndürülüyor

}

return -1; // Aranan eleman dizide yok

}

// BINARY SEARCH (İkili Arama)

// Sadece sıralı dizilerde çalışır. Dizinin ortasındaki elemana bakarak yön belirlenir.

// Zaman karmaşıklığı log(n). En hızlı arama yöntemlerinden biridir.

//Dizi sıralı. Her seferinde diziyi ikiye bölüyoruz. Aradığımız değer bu yarılardan birindedir.

Böl-böl-böl... ne zaman durur? Tek elemana kadar.

static int IkiliArama(int[] dizi, int aranan)

{

int sol = 0;

int sag = dizi.Length - 1;

while (sol <= sag)

{

int orta = (sol + sag) / 2;

if (dizi[orta] == aranan)

return orta; // Bulundu

else if (dizi[orta] < aranan)

sol = orta + 1; // Sağ yarıya geç

else

sag = orta - 1; // Sol yarıya geç

}

return -1; // Aranan bulunamadı

}

// JUMP SEARCH (Atlamalı Arama)

// Dizi sıralı olmalıdır. √n adım aralıklarla kontrol eder.

// Uygun bloğu bulunca, o blok içinde doğrusal arama yapar.

* Dizi sıralı ama binary search kadar güçlü bir bölme yapmıyoruz.
* Diziyi √n bloklara ayırıyoruz.
* İlk olarak √n adım aralıklarla zıplarız.
* Sonra uygun bloğun içinde max √n elemanlık bir lineer arama yaparız.

 Her elemana tek tek bakmak (linear search gibi) zaman alır.

 Ortayı bölmek (binary search gibi) donanımsal olarak zor olabilir.

 Bu yüzden belirli aralıklarla “zıplayarak” bakar.

* Diziye √n adım aralıklarla bakılır.
* Örn: 100 eleman varsa adım sayısı: √100 = 10
* Her zıplamada dizi[i] < aranan mı? diye kontrol edilir.
* Aranan değeri geçtiği blok bulunduğunda, geriye dönüp lineer arama yapılır.

static int AtlamaliArama(int[] dizi, int aranan)

{

int n = dizi.Length;

int adim = (int)Math.Floor(Math.Sqrt(n)); // √n

int onceki = 0;

while (dizi[Math.Min(adim, n) - 1] < aranan) //dizi sınırlarının patlamaması icin

{

onceki = adim;

adim += (int)Math.Floor(Math.Sqrt(n));

if (onceki >= n)

return -1;

}

for (int i = onceki; i < Math.Min(adim, n); i++)

{

if (dizi[i] == aranan)

return i; // Bulundu

}

return -1; // Bulunamadı

}

// EXPONENTIAL SEARCH (Üssel Arama)

// Çok büyük sıralı dizilerde kullanılır. 2^i adımlarla aralık bulunur.

// Sonrasında o aralıkta binary search yapılır.

static int UsselArama(int[] dizi, int aranan)

{

if (dizi[0] == aranan)

return 0;

int i = 1;

while (i < dizi.Length && dizi[i] <= aranan)

i \*= 2; // 2, 4, 8, 16 gibi sıçrayarak ilerle

return YardimciBinaryArama(dizi, i / 2, Math.Min(i, dizi.Length - 1), aranan);

}

static int YardimciBinaryArama(int[] dizi, int sol, int sag, int aranan)

{

while (sol <= sag)

{

int orta = (sol + sag) / 2;

if (dizi[orta] == aranan)

return orta; // Bulundu

else if (dizi[orta] < aranan)

sol = orta + 1;

else

sag = orta - 1;

}

return -1; // Bulunamadı

}

| **Algoritma** | **Zaman Karmaşıklığı (Big-O)** | **Dizi Sıralı mı Gerekir?** | **Ne Zaman Kullanılır?** |
| --- | --- | --- | --- |
| **Linear Search** | O(n) | Hayır | Sıralı değilse tek çare budur. |
| **Binary Search** | O(log n) | Evet | Dizi sıralıysa en hızlı ve en güvenli seçenektir. |
| **Jump Search** | O(√n) | Evet | Sıralı veride, binary yerine blok blok taramak için. |
| **Exponential Search** | O(log i) | Evet | Çok büyük sıralı dizilerde, aranan sayı başlardaysa süper hızlı. |

GRAFLAR

-DIJXTRA

0 numaralı şehirden (veya düğümden), diğer tüm şehirlere olan en kısa yolları bulmak istiyoruz.

çizgi, diyagram, daire içeren bir resim

Yapay zeka tarafından oluşturulan içerik yanlış olabilir.

0 1 2 3 4

0 0 0 10 30 10

1 0 0 0 0 0

2 10 0 0 20 60

3 30 0 20 0 10

4 10 0 60 10 0

{ 0, 0, 10, 0, 10 }, // 0. şehirden

{ 0, 0, 0, 0, 0 }, // 1. şehirden (bağlantı yok)

{ 10, 0, 0, 20, 60 }, // 2. şehirden

{ 30, 0, 20, 0, 10 }, // 3. şehirden

{ 10, 0, 60, 10, 0 } // 4. Şehirden

Önce veri yapısını kur. Matris şehir x şehir

mesafe[]: Her şehre olan en kısa mesafeyi tutar.  
ziyaretEdildi[]: Hangi şehirlerin işlenip bittiğini gösterir.

Bu vektörlere baslangic değerlerini ata çünkü yola çıktım, sadece bulunduğum yeri biliyorum. Diğer her yere olan mesafem sonsuz.

// Şehir sayısı (düğüm sayısı)

int N = 5;

// Yeni grafa göre şehirler arası mesafeler

int[,] graf = {

// 0 1 2 3 4

{ 0, 0, 10, 0, 10 }, // 0. şehirden

{ 0, 0, 0, 0, 0 }, // 1. şehirden (bağlantı yok)

{ 10, 0, 0, 20, 60 }, // 2. şehirden

{ 30, 0, 20, 0, 10 }, // 3. şehirden

{ 10, 0, 60, 10, 0 } // 4. Şehirden

};

// Mesafe dizisi (başta her yer sonsuz uzaklıkta)

int[] mesafe = new int[N];

// Hangi şehirler ziyaret edildi?

bool[] ziyaretEdildi = new bool[N];

// Başlangıç: 0 numaralı şehir

for (int i = 0; i < N; i++)

{

mesafe[i] = int.MaxValue;

ziyaretEdildi[i] = false;

}

mesafe[0] = 0; //baslangic dugumu

// Tüm şehirleri sırayla işle

//Dijkstra, her adımda en kısa yolu kesinleşmiş bir şehir seçip işleme sokar.  
 //Toplamda N-1 kere tekrar yapar çünkü zaten son şehir otomatik belirlenmiş olur.

for (int sayac = 0; sayac < N - 1; sayac++)

{

// En kısa mesafeli ve henüz ziyaret edilmemiş şehri bul

int enKisa = int.MaxValue;

int secilen = -1;

//Şu ana kadar en kısa yol bilgisi olan ama henüz uğramadığımız şehir hangisi? Onu seçeceğiz.

for (int i = 0; i < N; i++)

{

if (!ziyaretEdildi[i] && mesafe[i] < enKisa)

{

enKisa = mesafe[i];

secilen = i;

}

}

// Seçilen şehir artık ziyaret edildi

ziyaretEdildi[secilen] = true;

// Komşuların mesafelerini güncelle

for (int j = 0; j < N; j++)

{

if (graf[secilen, j] > 0 && !ziyaretEdildi[j])

{

//0🡪2

//0-4, 4-2

//0🡪2

int yeniMesafe = mesafe[secilen] + graf[secilen, j];

if (yeniMesafe < mesafe[j])

{

mesafe[j] = yeniMesafe;

}

}

}

}

// Sonuçları yazdır

Console.WriteLine("0 numaralı şehirden diğerlerine en kısa mesafeler:");

for (int i = 0; i < N; i++)

{

Console.WriteLine("Şehir {0} --Mesafe: {1}", i, mesafe[i]);

}

//4’ten 2’ye en kısa yol

using System;

class Program

{

static void Main()

{

int N = 5; // Şehir sayısı

// Şehirler arası mesafeler (0: yol yok)

int[,] graf = {

// 0 1 2 3 4

{ 0, 0, 10, 0, 10 }, // 0. şehir

{ 0, 0, 0, 0, 0 }, // 1. şehir

{ 10, 0, 0, 20, 0 }, // 2. şehir

{ 0, 0, 20, 0, 60 }, // 3. şehir

{ 10, 0, 0, 10, 0 } // 4. şehir

};

int[] mesafe = new int[N]; // Her şehre olan minimum mesafeleri tutar

bool[] ziyaretEdildi = new bool[N]; // Hangi şehirler ziyaret edildi?

// Başlangıç noktası: 4 numaralı şehir

for (int i = 0; i < N; i++)

{

mesafe[i] = int.MaxValue; // Başta tüm mesafeler sonsuz

ziyaretEdildi[i] = false; // Henüz hiçbir şehir ziyaret edilmedi

}

mesafe[4] = 0; // Başlangıç şehri kendine 0 uzaklıkta

// Dijkstra algoritması başlıyor

for (int sayac = 0; sayac < N - 1; sayac++)

{

// Ziyaret edilmemiş en yakın şehri seç

int enKisa = int.MaxValue;

int secilen = -1;

for (int i = 0; i < N; i++)

{

if (!ziyaretEdildi[i] && mesafe[i] < enKisa)

{

enKisa = mesafe[i];

secilen = i;

}

}

// Hiçbir şehir seçilememişse döngü kırılır (ulaşılamaz)

if (secilen == -1) break;

ziyaretEdildi[secilen] = true;

// Komşuları kontrol et ve mesafeleri güncelle

// Komşuların mesafelerini güncelle

//Bu Şehirden Komşulara Gidip Mesafeleri Güncelle

//komşudan diğer şehirlere mesafe

for (int j = 0; j < N; j++)

{

if (graf[secilen, j] > 0 && !ziyaretEdildi[j])

{

int yeniMesafe = mesafe[secilen] + graf[secilen, j];

if (yeniMesafe < mesafe[j])

{

mesafe[j] = yeniMesafe;

}

}

}

}

// Sadece 2 numaralı şehre olan mesafeyi yazdır

if (mesafe[2] == int.MaxValue)

Console.WriteLine("4 numaralı şehirden 2 numaralı şehre yol bulunamadı.");

else

Console.WriteLine("4 numaralı şehirden 2 numaralı şehre en kısa mesafe: " + mesafe[2]);

}

}