IPAR 4.0 .NET C# ALAPOKON

2. forduló

SEMILAB

A kategória támogatója: Semilab Zrt.

Ismertető a feladathoz

Útmutató:

- A **radio button-os kérdésekre** egy helyes válasz van.
- **Ha lejár a feladatlap ideje, a rendszer AUTOMATIKUSAN** beküldi azt az addig megjelölt válaszokkal.
- Az **adatbekérős feladatokra NEM jár részpontszám**, csak a feleletválasztósakra.
- **Badge-ket** a 4.forduló után kapsz majd először.
- Az **adatbekérős kérdéseknél** igyekeztünk minden variációt megadni (kisbetű, nagybetű, szóköz), de ha mégis eltérést tapasztalsz a megoldásokban, kérjük, jelezd felénk!

+1: Azért szólunk, hogy senkit ne a végén érjen meglepetés: a játék nem tipp-mix és csapatkategória sincs! Természetesen akinek nem inge...

Jó versenyzést kívánunk!

A feladatlap több csatolmányt is tartalmaz, ezért a megoldását asztali gépen javasoljuk!

Fontos!

Fordulónként javasoljuk az összes részfeladat végigolvasását a kidolgozás megkezdése előtt, mivel a feladatok sokszor egymásra épülnek. Előfordul, hogy egy részfeladat nehézségét az input mérete adja, így érdemes hatékony megoldásokra törekedni.

Felhasznált idő: 00:00/40:00 **Elért pontszám:** 0/12

Indítás előtti csatolmányok

1. feladat 0/3 pont

Egy eredetileg rendezett mintahalmazt összekeverés után vizsgálunk. Az egyes mintákat egy 8-bites jellemzővektorral írjuk le, ez lesz az összehasonlításunk alapja.

Az inputunk egy 8-bites bináris jelsorozatokat tartalmazó tömb (jellemzővektorok).

Rendezzük az input tömbünket a „01011101” referencia jelsorozattól számított Hamming-távolság alapján! **A kimeneten adjuk meg ezt a rendezést az eredeti tömbindexek felhasználásával, távolság szerint növekvő sorrendben, szóközzel elválasztva**. A sorrendben távolságegyezés esetén mindig az eredeti tömbben kisebb index értékkel rendelkező elemet vegyük előre.

Példa

Input:

00000111

01011111

11111111

Output:

1 2 0

(Az 1-es indexű jelsorozat távolsága a legkisebb a referenciától, utána a 2-es indexű, majd 0-s)

A 2_1_test.txt-re adott kimenet: 0 3 4 2 1

Adjuk meg a 2_1.txt-re a kimenetet!

Válaszok

A helyes válasz:

6 5 7 0 1 4 2 3

65701423

6 5 7 0 1 4 2 3

Magyarázat

Lásd 3. feladat magyarázat.

2. feladat 0/3 pont

A kevert mintahalmazunk valójában a minták egy sorozata volt, amiben az egymás mellett lévők nagyon hasonlóak voltak. Szeretnénk ezt a hasonlóságot felhasználni extra információként egy méréshez, ezért megkíséreljük helyreállítani az összekeverés előtti sorrendet. Egy tömbön belüli sorrend jóságát definiáljuk úgy, hogy a sorrend alapján végighaladva a mintákon, az egymást követő minták jellemzővektorai közötti Hamming-távolságokat összegezzük. Minél kisebb ez az össztávolság, annál jobb a sorozat.
Adjuk meg az 1. feladatban kapott sorrendre az össztávolságot!

Példa

Input:

00000111

01011111

11111111

Output:

7

(Az 1-es és 2-es indexű jelsorozat távolsága 2, a 2-es és 0-s indexű távolsága 5, így az összeg 7.)

A 2_1_test.txt-re adott kimenet: 13

Adjuk meg a 2_1.txt-re a kimenetet!

Válasz

A helyes válasz:

26

Magyarázat

Lásd 3. feladat magyarázat.

3. feladat 0/6 pont

Találjuk meg a 2. feladatban definiált jóság alapján legjobb sorozatot **az input tömbre** és **írjuk a kimenetre az össztávolságát**! A bemeneti tömb kisméretű, így a problémát belátható időn belül meg lehet oldani.

Példa:

Input:

00000111

01011111

11111111

Output:

5

(Az 0 1 2 sorrend össztávolsága minimális, melynek értéke 5.)

A 2_1_test.txt-re adott kimenet: 11

Adjuk meg a 2_1.txt-re a kimenetet!

Válasz

A helyes válasz:

18

Magyarázat

```
public static class HammingDistance
{
    public static readonly string _refSample = "01011101";


    public static void Solve(string fileName, out int[] hammingOrder, out int totalDistance, out int shor
    {
        var lines = File.ReadAllLines(fileName);

        // minden mintához kiszámoljuk a referenciától mért Hamming távolságát
        var samples = lines.Select((x, i) => new Sample(x, i, CalcHammingDist(_refSample, x))).ToList();
```

```
        // a mért Hamming távolság (és utána az inputban szereplő helyük) alapján sorbarakjuk a mintákat
        var samplesOrdered = samples.OrderBy(s => s.HammingDistFromRef).ThenBy(s => s.Idx).ToArray();
        hammingOrder = samplesOrdered.Select(s => s.Idx).ToArray();


        // kiszámoljuk és összegezzük a rendezett minták sorában az egymás után lévő tagok között levő Ha
        totalDistance = Enumerable.Range(0, samples.Count - 1)
            .Select(i => CalcHammingDist(samplesOrdered[i].Value, samplesOrdered[i + 1].Value)).Sum();


        // előállítjuk a minták összes lehetséges permutációját és megkeressük a legkisebb össztávolságot
        shortestTotalDistance = int.MaxValue;
        foreach (var permutation in GetPermutations(Enumerable.Range(0, samples.Count), samples.Count))
        {
            var p = permutation.ToArray();
            var distance = Enumerable.Range(0, samples.Count - 1)
                .Select(i => CalcHammingDist(samples[p[i]].Value, samples[p[i + 1]].Value)).Sum();


            if (distance < shortestTotalDistance)
            {
                shortestTotalDistance = distance;
            }
        }
    }


    static int CalcHammingDist(string sample1, string sample2)
    {
        return sample1.Zip(sample2, (c1, c2) => c1 != c2 ? 1 : 0).Count(x => x == 1);
    }


    static IEnumerable<IEnumerable<T>> GetPermutations<T>(IEnumerable<T> list, int length)
    {
        if (length == 1) return list.Select(t => new T[] { t });


        return GetPermutations(list, length - 1)
            .SelectMany(t => list.Where(e => !t.Contains(e)),
                (t1, t2) => t1.Concat(new T[] { t2 }));
    }
}


public class Sample
{
    public string Value { get; set; }
    public int Idx { get; set; }
    public int HammingDistFromRef { get; set; }


    public Sample(string value, int idx, int hammingDistFromRef)
    {
        Value = value;
        Idx = idx;
        HammingDistFromRef = hammingDistFromRef;
    }
}
```

Megjelenés

Világos