

Kodlama Söyleşisi

java

Program Deresi

1 Mayıs 2016

İçindekiler

1 Java'da Diziyi Döndür	13
2 Ters Lehçe Gösterimini Değerlendirin	17
3 İzomorf Dizi	21
4 Kelime Merdiveni	23
5 Kelime Merdiveni II	25
6 İki Sıralı Dizinin Medyanı	29
Bir Dizideki 7 Kinci En Büyük Eleman	31
8 Joker Karakter Eşleştirme	33
9 Java'da Normal İfade Eşleştirme	35
10 Birleştirme Aralığı	39
11 Aralık Ekle	41
12 İki Toplam	43
13 İki Toplam II Giriş dizisi sıralanır	45
14 Two Sum III Veri yapısı tasarımlı	47
15 3 Toplam	49
16 4 ben	53
17 3Toplam En Yakın	55
18 Dizeden Tamsayıya (atoi)	57
19 Sıralanmış Diziyi Birleştir	59
20 Geçerli Parantez	61

21 En Uzun Geçerli Parantez	63
22 strStr()'yi uygula	65
23 Minimum Boyutlu Alt Dizi Toplamı	69
24 Arama Ekle Konum	73
25 En Uzun Ardışık Dizi	75
26 Geçerli Palindrom	77
27 ZigZag Dönüştürme	81
28 İkili Ekle	83
29 Son Söz Uzunluğu	85
30 Üçgen	87
31 Kopya İçerir	89
32 Kopya II İçerir	91
33 Yinelenen III İçerir	93
34 Sıralanmış Diziden Kopyaları Kaldır	95
35 Sıralanmış Diziden Kopyaları Kaldır II	99
Karakterleri Tekrar Etmeden 36 En Uzun Alt Dizi	103
2 Benzersiz Karakter İçeren 37 En Uzun Alt Dizi	105
Tüm Sözcüklerin Birleştirilmesiyle 38 Alt Dizi	109
39 Minimum Pencere Alt Dizisi	111
Bir Dizide 40 Ters Sözcük	113
41 Döndürülmüş Sıralı Dizide Minimumu Bulma	115
42 Döndürülmüş Sıralı Dizide Minimum Bulma II	117
43 Döndürülmüş Sıralı Dizide Arama	119
44 Döndürülmüş Sıralı Dizide Arama II	121
45 Dakika Yiğin	123

İçindekiler

46 Çoğunluk Elemanı	125
47 Çoğunluk Elemanı II	127
48 Elemanı Kaldır	129
49 Histogramdaki En Büyük Dikdörtgen	131
50 En Uzun Ortak Önek	133
51 En Büyük Sayı	135
52 Yolu Basitleştir	137
53 Sürüm Numaralarını Karşılaştırın	139
54 Benzin İstasyonu	141
55 Pascal Üçgeni	143
56 Pascal Üçgeni II	145
57 En Fazla Su İçeren Kap	147
58 şeker	149
59 Yağmur Suyunun Tutulması	151
60 Say ve Söyle	153
61 Aralık Arama	155
62 Temel Hesap Makinesi	157
63 Grup Anagramı	159
64 En Kısa Palindrom	161
65 Dikdörtgen Alan	163
66 Özet Aralıkları	165
67 Artan Üçlü Dizi	167
68 Numara Listesini ve Aritmetik İşlemleri Kullanarak Hedef Numarayı Alın	169
Bir Dizinin 69 Ters Ünlüsü	171
70 Çevirme Oyunu	173

71 Döndürme Oyunu II	175
72 Sıfır Taşı	177
73 Geçerli Anagram	179
74 Grup Kaydırılmış Dizeler	181
75 En Sık Kullanılan K Öğe	183
76 Doruk Elemanı Bul	185
77 Kelime Kalıbü	187
78 Matris Sıfırlarını Ayarla	189
79 Spiral Matris	193
80 Spiral Anne II	197
81 2B Matris Arama	199
82 2D Matrix II Arama	201
83 Görüntüyü Döndür	205
84 Geçerli Sudoku	207
85 Minimum Yol Toplamı	209
86 Benzersiz Yol	211
87 Eşsiz Yol II	213
88 Ada Sayısı	215
89 Ada Sayısı II	217
90 Çevrelenmiş Bölge	219
91 Maksimal Dikdörtgen	223
92 Maksimum Araba	225
93 Kelime Arama	227
94 Kelime Arama II	229
95 Tamsayı Arası	233

İçindekiler

96 Aralık Toplamı Sorusu 2B Sabit	235
97 Matristeki En Uzun Artan Yol	237
98 Java'da Dizi Kullanarak Yiğin Gerçekleştirme	239
99 İki Numara Toplama	243
100 Yeniden Sıralama Listesi	245
101 Bağlantılı Liste Döngüsü	251
Rastgele İşaretçili 102 Kopya Listesi	253
103 Sıralanmış İki Listeyi Birleştirme	257
104 Tek Çift Bağlantılı Liste	259
105 Yinelenenleri Sıralı Listeden Kaldır	261
106 Yinelenenleri Sıralı Listeden Kaldır II	263
107 Bölüm Listesi	265
108 LRU Önbeliği	267
109 İki Bağlantılı Listenin Kesimi	271
110 Bağlantılı Liste Öğelerini Kaldır	273
Çiftler halinde 111 Düğüm Değiştirme	275
112 Ters Bağlantılı Liste	277
113 Ters Bağlantılı Liste II	279
114 N'inci Düğümü Listenin Sonundan Kaldır	281
115 Kuyrukları Kullanarak Yiğin Uygulama	283
116 Yiğinları Kullanarak Kuyruk Uygulama	285
117 Palindrome Bağlantılı Liste	287
118 Java'da Dizi Kullanarak Kuyruk Gerçekleştirme	289
119 Bağlantılı Listedeki Düğümü Sil	291
Veri Akışından 120 Hareketli Ortalama	293

1 Java'da Diziyi Döndür

Bir süredir Java kullanıyor olabilirsiniz. Basit bir Java dizisi sorusunun zor olabileceğini düşünüyor musunuz? Test etmek için aşağıdaki problemi kullanalım.

Problem: n elemanlı bir diziyi k adımda sağa döndürün. Örneğin, n = 7 ve k = 3 ile [1,2,3,4,5,6,7] dizisi [5,6,7,1,2,3,4]'e döndürülür . Bu sorunu çözmek için kaç farklı yol biliyorsunuz?

1.1 1. Çözüm - Ara Dizi

Basit bir şekilde, yeni bir dizi oluşturabilir ve ardından öğeleri yeni diziye kopyalayabiliriz. Ardından, orijinal diziyi System.arraycopy() kullanarak değiştiririn.

```
genel boşluk döndürme(int[] sayılar, int k)
{ if(k > sayı.uzunluk) k=k%sayı.uzunluk;

int[] sonuç = yeni int[sayı.uzunluk];

for(int i=0; i < k; i++)
    { sonuç[i] = sayılar[sayılar.uzunluk-k+i];
    }

int =
0; for(int i=k; i<nums.length; i++){ sonuç[i]
    = nums[j]; j++;
}

System.arraycopy( sonuç, 0, sayılar, 0, sayılar.uzunluk );
}
```

Uzay O(n) ve zaman O(n)'dir. Sys arasındaki farkı kontrol edebilirsiniz. tem.arraycopy() ve Arrays.copyOf().

1.2 2. Çözüm - Balon Döndürme

Bunu O(1) uzayında yapabilir miyiz ?
Bu çözüm bir kabarcık sıralama gibidir.

```
genel statik geçersiz döndürme(int[] dizi, int sıra) { if (dizi == null
| | sıra < 0) {
```

1 Java'da Diziyi Döndür

```
        throw new IllegalArgumentException("Geçersiz argüman!");
    }

    for (int i = 0; i < sıra; i++) { for (int j = dizi.uzunluk -
        1; j > 0; j--) { int temp = dizi[j]; dizi[j] = dizi[j - 1]; dizi[j - 1] = sıcaklık;
    }

}
}
```

Ancak, zaman $O(n*k)$ 'dir.

İşte bir örnek (uzunluk=7, sıra=3):

```
ben=0  
0 1 2 3 4 5 6  
0 1 2 3 4 6 5  
...  
6 0 1 2 3 4 5  
ben=1  
6 0 1 2 3 5 4  
...  
5 6 0 1 2 3 4  
ben=2  
5 6 0 1 2 4 3  
...  
4 5 6 0 1 2 3
```

1.3 3. Çözüm - Geri Alma

Bunu $O(1)$ uzayında ve $O(n)$ zamanında yapabilir miyiz ? Aşağıdaki çözüm yapar.

Bize 1,2,3,4,5,6 ve 2. sıra verildiğini varsayıyalım . Temel fikir şudur:

1. Diziyi iki parçaya bölün: 1,2,3,4 ve 5, 6 2. İlk kısmı ters çevirin: 4,3,2,1,5,6 3. İkinci kısmı ters çevirin: 4,3,2,1, 6,5 4. Tüm diziyi ters çevirin: 5,6,1,2,3,4
-
-

```
genel statik geçersiz döndürme (int[] dizi, int sıra)
{ if (arr == null || dizi.uzunluk==0 || sıra < 0)
    { throw new IllegalArgumentException("Geçersiz argüman!");
}

if(sıra > dizi.uzunluk){
    sıra = sıra %dizi.uzunluk;
}
```

[1 Java'da Diziyi Döndür](#)

```
//ilk parçanın uzunluğu int a
= dizi.uzunluk - sıra;

ters(dizi, 0, a-1); reverse(dizi,
a, dizi.uzunluk-1); ters(dizi, 0,
dizi.uzunluk-1);

}

public static void reverse(int[] dizi, int sol, int sağ)
{ if(arr == null || dizi.uzunluk == 1) dönüş;

    süre(sol < sağ)
    { int temp = dizi[sol]; dizi[sol]
    = dizi[sağ]; dizi[sağ] = sıcaklık;
    sol++; Sağ--;
    }

}
```

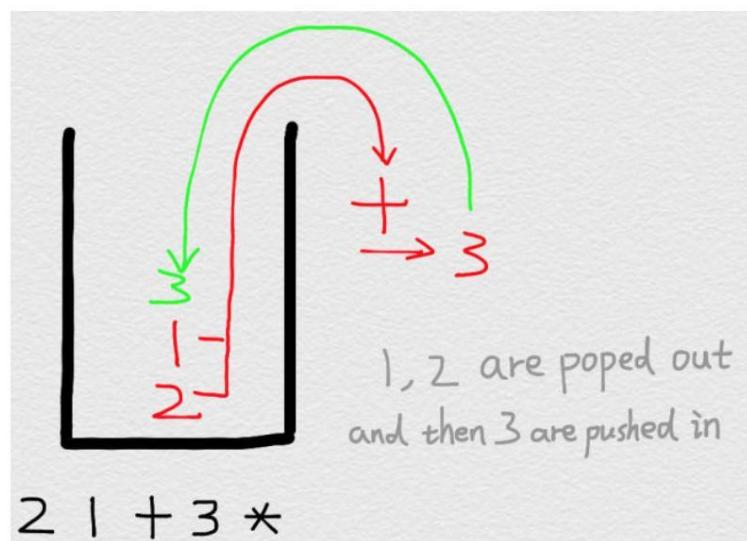
2 Ters Lehçe Gösterimini Değerlendirin

Bir aritmetik ifadenin değerini Ters Lehçe Notasyonunda değerlendirin. Geçerli işaretler +, -, *, /'dir. Her işlenen bir tamsayı veya başka bir ifade olabilir. Örneğin:

["2", "1", "+", "3", "*"] -> ((2 + 1) * 3) -> 9
 ["4", "13", "5", "/", "+"] -> (4 + (13 / 5)) -> 6

2.1 Naif Yaklaşım

Bu sorun bir yiğin kullanılarak çözülebilir. Verilen dizideki her bir eleman arasında döngü yapabiliriz. Bir sayı olduğunda, yiğine itin. Bir operatör olduğunda, yiğinden iki sayı çıkarın, hesaplamayı yapın ve sonucu geri alın.



Aşağıdaki koddur. Ancak bu kod leet kodunda derleme hataları içermektedir. Neden?

genel sınıf Testi {

```
public static void main(String[] args) IOException { atar.);  

Dize[] belirteçleri = yeni Dize[] { "2", "1", "+", "3",  

System.out.println(evalRPN(belirteçler));  

}
```

2 Ters Lehçe Gösterimini Değerlendirin

```
genel statik int evalRPN(String[] belirteçleri) { int dönüşDeğeri
    = 0;
    Dizi işleçleri = "+-*/";
    Yiğın<Dize> yiğın = yeni Yiğın<Dize>();

    for (String t : belirteçler) { if (!
        operators.contains(t)) { //eğer bir sayı ise stack'e bas stack.push(t); } else {//bir operatör ise
            yiğinden sayıları çıkar int a = Integer.valueOf(stack.pop()); int b =
            Integer.valueOf(stack.pop()); anahtar (t) {

                durum "+":
                    stack.push(String.valueOf(a + b)); kirmak;

                durum "-":
                    stack.push(String.valueOf(b - a)); kirmak; durum
                    "*": stack.push(String.valueOf(a * b)); kirmak;
                durum "/": stack.push(String.valueOf(b / a)); kirmak;

            }
        }
    }

   returnValue = Integer.valueOf(stack.pop());
    dönüşDeğeri ;
}
```

Sorun şu ki, switch string ifadesi yalnızca JDK 1.7'den edinilebilir . Eve git
görünüşe göre 1.7'nin altındaki bir JDK sürümünü kullanıyor .

2.2 Kabul Edilen Çözüm

switch deyimini kullanmak istiyorsanız, "+-*/" dizisinin dizinini kullanan aşağıdaki kodu kullanarak yukarıdakini dönüştürebilirsiniz.

```
genel sınıf Çözüm { genel int
evalRPN(String[] belirteçleri) {

    int dönüşDeğeri = 0;
```

Dizi İşleçleri = "+-*"/;

Yığın<Dize> yığın = yeni Yığın<Dize>();

```
for(String t : belirteçler){ if(!
    operatörler.contains(t)){ stack.push(t); }
    başka{ int bir =
        Integer.valueOf(stack.pop()); int b =
        Integer.valueOf(stack.pop()); int indeks =
        operatörler.indexOf(t); anahtar(dizin){

            durum 0:
            stack.push(String.valueOf(a+b)); kırmak;
            durum 1:

            stack.push(String.valueOf(ba)); kırmak; durum 2:
            stack.push(String.valueOf(a*b)); kırmak;

            durum 3:
            stack.push(String.valueOf(b/a)); kırmak;
        }
    }
}

returnValue = Integer.valueOf(stack.pop());

dönüşDeğeri ;
}
```

3 İzomorf Dizi

İki dizi s ve t verildiğinde, izomorfik olup olmadıklarını belirleyin. s'deki karakterler t'yi elde etmek için değiştirilebiliyorsa, iki dizi izomorfiktir.

Örneğin, "yumurta" ve "ekle" izomorfiktir, "foo" ve "bar" değildir.

3.1 Analiz

Bir haritayı ve değeri kabul eden ve haritada değerin anahtarını döndüren bir yöntem tanımlamamız gerekiyor.

3.2 Java Çözümü

```
public boolean isIsomorphic(String s, String t) { if(s==null || t==null) false
    döndürür;

    if(s.uzunluk() != t.uzunluk()) yanlış döndürür;

    if(s.uzunluk()==0 && t.uzunluk()==0)
        doğru dönüş;

    HashMap<Character, Character> map = new HashMap<Character,Character>(); for(int i=0; i<s.uzunluk(); i++) { char
    c1 = s.charAt(i); char c2 = t.charAt(i);

    Karakter c = getKey(harita, c2); if(c != boş &&
    c!=c1){ false döndürür;

    }else if(map.containsKey(c1)){ if(c2 !=
        map.get(c1)) false döndürür; }
        başka{ map.put(c1,c2);

    }

    doğrudan dönüş;
}
```

3 İzomorf Dizi

```
// bir hedef değerin anahtarını almak için bir yöntem public  
Character getKey(HashMap<Character,Character> map, Character target){ for (Map.Entry<Character,Character> giriş :  
    map.entrySet() { if (entry.getValue().equals(hedef)) { return entry.getKey();  
  
    }  
  }  
  
  boş dönüş;  
}
```

4 Kelime Merdiveni

İki kelime (başlangıç ve bitiş) ve bir sözlük verildiğinde, baştan sona en kısa dönüşüm dizisinin uzunluğunu bulun, öyle ki bir seferde yalnızca bir harf değiştirilebilir ve her bir ara kelime sözlükte bulunmalıdır. Örneğin, verilen:

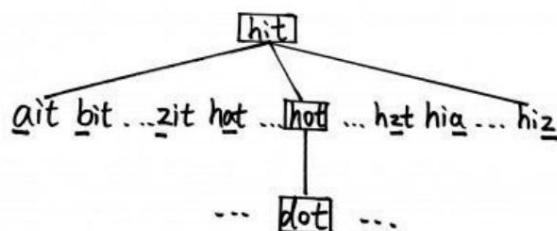
```
başlangıç \u200b\u200b="vurmak"
end = "cog"
dict = ["hot","dot","dog","lot","log"]
```

En kısa dönüşümlerden biri "hit" ->"hot" ->"dot" ->"dog" ->"cog" şeklidindedir, program uzunluğunu 5 döndürmelidir .

4.1 Analiz

06/07/2015 tarihinde GÜNCELLENMİŞTİR .

Böylece bunun bir arama sorunu olduğunu hemen anlarız ve önce nefes arama garanti eder. en uygun çözüm.



4.2 Java Çözümü

```
sınıf WordNode{
    Dize kelime; int
    adım sayısı;

    public WordNode(String word, int numSteps){ this.word =
        word; this.numSteps = sayıAdımları;

    }
}
```

4 Kelime Merdiveni

```
genel sınıf Çözümü
{ public int ladderLength(String beginWord, String endWord, Set<String>
    sözcükDict)
{ LinkedList<WordNode> kuyruğu = new LinkedList<WordNode>(); sıra.add(yeni
    WordNode(beginWord, 1));

    wordDict.add(endWord);

    while(!queue.isEmpty()){ WordNode üst =
        tail.remove(); Dize kelime = top.word;

        if(word.equals(endWord)){ dönüş
            top.numSteps;
        }

        char[] arr = kelime.toCharArray(); for(int i=0;
        i<arr.length; i++){ for(char c='a'; c<='z'; c++)
        { karakter sıcaklığı = dizi[i]; if(arr[i]!=c){ dizi[i]=c;

        }

        Dize yeniKelime = yeni Dize(arr);
        if(wordDict.contains(newWord)){ tail.add(new
            WordNode(newWord, top.numSteps+1)); wordDict.remove(yeniKelime);

        }

        dizi[i]=temp;
    }
}

0 dönüşü ;
}
}
```

5 Kelime Merdiveni II

İki kelime (başlangıç ve bitiş) ve bir sözlük verildiğinde, baştan sona tüm en kısa dönüşüm dizilerini bulun, öyle ki: 1) Bir seferde yalnızca bir harf değiştirilebilir, 2) Her bir ara kelime kelimedede bulunmalıdır. sözlük.

Örneğin, verilen: start = "hit", end = "cog" ve dict = ["hot", "dot", "dog", "lot", "log"], geri dönmek:

```
[  
  ["vur", "sıcak", "nokta", "köpek", "dişli"],  
  ["vur", "sıcak", "çok", "kütük", "dişli"]  
]
```

5.1 Analiz

Bu, Word Ladder'ın bir uzantısıdır .

Fikir aynı. Gerçek merdiveni izlemek için, WordNode sınıfındaki önceki düğüme işaret eden bir işaretçi eklememiz gereklidir.

Ayrıca kullanılan kelime doğrudan sözlükten çıkarılamaz. Kullanılmış kelime yalnızca adımlar değiştiğinde kaldırılır.

5.2 Java Çözümü

```
sınıf WordNode{  
    Dize kelime; int  
    adım sayısı;  
    WordNode öncesi;  
  
    genel WordNode(String word, int numSteps, WordNode pre){  
        this.kelime = kelime;  
        this.numSteps = sayıAdımları;  
        this.pre = ön;  
    }  
}  
  
genel sınıf Çözümü  
{ public List<List<String>> findLadders(Dize başlangıcı, Dize sonu,  
    Set<String> dict) {  
    List<List<String>> sonuç = new ArrayList<List<String>>();
```

5 Kelime Merdiveni II

```
LinkedList<WordNode> kuyruğu = new LinkedList<WordNode>(); sıra.add(yeni
WordNode(başlangıç, 1, boş));

dict.add(oğul);

int minAdım = 0;

HashSet<String> ziyaret edildi = new HashSet<String>();
HashSet<String> ziyaret edilmedi = new HashSet<String>(); ziyaret
edilmemiş.addAll(dict);

int preNumSteps = 0;

while(!queue.isEmpty()){ WordNode üst =
tail.remove(); Dize kelime = top.word; int
currNumSteps = top.numSteps;

if(word.equals(end))
{ if(minStep == 0)
{ minStep = top.numSteps;
}

if(top.numSteps == minStep && minStep !=0){ //
nothing ArrayList<String> t = new
ArrayList<String>(); t.add(üst.kelime); while(top.pre !=null){

t.add(0, top.pre.word); üst =
üst.pre;

} sonuç.add(t);
devam etmek;
}

}

if(preNumSteps < currNumSteps)
{ unvisited.removeAll(ziyaret edildi);
}

preNumSteps = currNumSteps;

char[] arr = kelime.toCharArray(); for(int i=0;
i<arr.length; i++){ for(char c='a'; c<='z'; c++)
{ karakter sıcaklığı = dizi[i]; if(arr[i]!=c){ dizi[i]=c;
}

}

}
```

```
Dize yeniKelime = yeni Dize(arr);
if(unvisited.contains(newWord)){ tail.add(new
    WordNode(newWord, top.numSteps+1, top)); ziyaret edildi.add(yeniKelime);

}

dizi[i]=temp;
}
}

dönüş sonucu;
}
}
```

6 İki Sıralı Dizinin Medyanı

Sırasıyla m ve n boyutunda iki sıralanmış A ve B dizisi vardır. Sıralanan iki dizinin medyanını bulun. Genel çalışma süresi karmaşıklığı $O(\log(m+n))$ olmalıdır.

6.1 Java Çözümü 1

$\log(n)$ görürsek, ikili bir şey kullanmayı düşünmeliyiz.

Bu problem, k'inci elemanı bulma problemine dönüştürülebilir, k (A'nın uzunluğu + B' Uzunluğu)/2'dir.

İki diziden herhangi biri boşsa, kth öğesi boş olmayan dizinin kth öğesidir. k == 0 ise, k'inci eleman A veya B'nin ilk elemanıdır.

Normal durumlar için (diğer tüm durumlar), işaretçiyi hızın yarısı kadar hareket ettirmemiz gereklidir. $\log(n)$ süresini elde etmek için bir dizi uzunluğu.

```
genel static çift findMedianSortedArrays(int A[], int B[]) { int m
    = A.uzunluk; int n
    = B.uzunluk;

    if ((m + n) % 2 != 0) // tek
        dönüş (çift) bulKth(A, B, (m + n) / 2, 0, m - 1, 0, n - 1); başka
    { // hatta dönüş (bulKth(A, B, (m + n) / 2, 0, m - 1, 0, n - 1)

        + findKth(A, B, (m + n) / 2 - 1, 0, m - 1, 0, n - 1)) * 0,5;
    }
}

public static int findKth(int A[], int B[], int k, int aStart, int aEnd, int bStart,
    int bEnd) {

    int aLen = aEnd - aStart + 1; int bLen =
    bEnd - bBaşlat + 1;

    // if (aLen == 0) B[bStart + k]
    döndürse özel durumları ele
    alin ; eğer (bLen == 0) A[aStart
    + k] döndürürse ; eğer (k == 0)
        A[aStart] < B[bStart] döndürür
    mü ? A[aBaşlat] : B[bBaşlat];

    int aMid = aLen * k / (aLen + bLen); // a'nın orta sayısı int bMid = k - aMid - 1; // b'nin
    orta sayısı
```

6 İki Sıralı Dizinin Medyani

```
// aMid ve bMid'i dizi indeksi yapın aMid = aMid + aStart;  
bMid = bMid + bStart;  
  
eğer (A[aOrta] > B[bOrta]) {  
    k = k - (bOrta - bBaşlat + 1); birSon =  
    birOrta; bBaşlat = bOrta + 1; } başka { k = k -  
    (aOrta - aStart + 1); bEnd = bOrta; aStart  
    = Orta + 1;  
  
}  
  
findKth (A, B, k, aStart, aEnd, bStart, bEnd);  
}
```

6.2 Java Çözümü 2

Çözüm 1, k'inci elemanı bulmak için genel bir çözümdür. Bu özel problem için yalnızca iki sıralı dizinin ortancasını bulan daha basit bir çözüm de bulabiliriz. Gunner86'ya teşekkürler. Algoritmanın açıklaması harika!

- 1) ar1[] ve ar2[] giriş dizilerinin medyanlarını m1 ve m2 hesaplayın sırasıyla.
 - 2) m1 ve m2 eşitse işimiz biter ve m1 (veya m2) değerini **döndürürüz** 3) .
m1, m2'den büyüğse, medyan aşağıdaki iki değerden birinde bulunur.
alt diziler. a)
ar1'in ilk elemanından m1'e (ar1[0..._n/2_-1]) b) m2'den ar2'nin son elemanına (ar2[_n/2_-1...n-1]) 4) m2, m1'den büyüğse, medyan aşağıdaki iki değerden birinde bulunur.
alt diziler. a)
 - 3) Her iki alt dizinin boyutu 2 olana kadar yukarıdaki işlemi tekrarlayın. 6) İki dizinin boyutu 2 ise, medyanı almak için aşağıdaki formülü kullanın.
Medyan = (maks(ar1[0], ar2[0]) + min(ar1[1], ar2[1]))/2
-

Bir Dizideki 7 Kinci En Büyük Eleman

Sıralanmamış bir dizideki k. en büyük öğeyi bulun. Sıralama düzeneinde k'inci en büyük öğe olduğuna dikkat edin, k'inci ayrı öğe değil.

Örneğin, [3,2,1,5,6,4] ve k = 2 verildiğinde, 5 döndürür.

Not: k'nin her zaman geçerli olduğunu varsayıbilirsiniz, 1 ≤ k dizinin uzunluğu.

7.1 Java Çözümü 1 - Sıralama

```
public int findKthLargest(int[] sayilar, int k) { Diziler.sort(sayilar);
    dönüş sayiları[sayilar.uzunluk-k];
}
```

Zaman O(nlog(n))

7.2 Java Çözümü 2 - Hızlı Sıralama

Bu sorun, hızlı sıralamaya benzer hızlı seçim yaklaşımı kullanılarak da çözülebilir .

```
public int KthLargest bul ( int [ ] boyut , int k ) { if ( k < 1 || boyut ==
boş ) { dönüş 0;
}
getKth (sayilar.uzunluk - k +1, sayilar, 0, sayı.uzunluk - 1);
}

public int getKth(int k, int[] sayilar, int başlangıç, int bitiş) {

    int pivot = sayilar[son];

    int sol = başlangıç; int
    sağ = bitiş;

    süre (doğru) {

        while (sayilar[sol] < pivot && sol < sağ) {
            güneş++;
        }
    }
}
```

Bir Dizideki 7 Kinci En Büyük Eleman

```
while (sayilar[sağ] >= pivot && sağ > sol) {  
    Sağ--;  
}  
  
eğer (sol == sağ) { ara;  
  
}  
  
takas(sayılar, sol, sağ);  
}  
  
takas(sayılar, sol, son);  
  
if (k == left + 1) { dönüş  
pivotu; } else if (k <  
left + 1) { dönüş  
getKth(k, sayılar, başlangıç, sol - 1); } else { getKth  
(k, sayıler, sol + 1, oğul);  
  
}  
}  
  
genel geçersiz takas(int[] sayılar, int n1, int n2) { int tmp =  
sayilar[n1]; sayılar[n1] = sayılar[n2]; sayılar[n2] = tmp;  
  
}  


---


```

Ortalama durum süresi $O(n)$, en kötü durum süresi $O(n^2)$ 'dir.

7.3 Java Çözümü 3 - Yığın

Bu sorunu çözmek için bir min yığını kullanabiliriz. Yığın, en üstteki k öğeyi depolar.

Boyut k'den büyük olduğunda min'i silin. Zaman karmaşıklığı $O(n \log(k))$.

Uzay karmaşıklığı, en yüksek k sayısını depolamak için $O(k)$ 'dir.

```
public int findKthLargest(int[] sayıları, int k)  
{ PriorityQueue<Tamsayı> q = new PriorityQueue<Tamsayı>(k); for(int i: nums)  
{ q.offer(i);  
  
if(q.size()>k){ q.poll();  
  
}  
}  
  
q.peek();  
}
```

8 Joker Karakter Eşleştirme

'?' desteğiyle joker desen eşleştirme uygulayın ve '*'.

8.1 Java Çözümü

Bu çözümü anlamak için `s="aab"` ve `p="*ab"` kullanabilirsiniz.

```
genel boolean isMatch(String s, String p) { int  
    ben = 0;  
    int i = 0; int  
    yıldızIndex = -1; int indeks =  
        -1;  
  
    while (i < s.uzunluk()) {  
        if (j < p.length() && (p.charAt(j) == '?' || p.charAt(j) == s.charAt(i)))  
  
            {++i;  
             ++j;  
         } else if (j < p.length() && p.charAt(j) == '*') { starIndex = j; dizin =  
             Ben; j++;  
  
         } else if (starIndex != -1) { j = starIndex + 1;  
             i = iIndex+1; indeks++; } else { yanlış  
             doldur;  
  
         }  
    }  
  
    while (j < p.uzunluk() && p.charAt(j) == '*') {  
        ++j;  
    }  
  
    j == p.uzunluk ();  
}
```

9 Java'da Normal İfade Eşleştirme

'.' desteğiyle eşleşen normal ifadeyi uygulayın. Ve '*'.

'.' Herhangi bir tek karakterle eşleşir. '*' Önceki ögenin sıfır veya daha fazlasıyla eşleştir.

Eşleştirme, tüm giriş dizesini kapsamalıdır (kısımlı değil).

İşlev prototipi şöyle olmalıdır: bool isMatch(const char *s, const char *p)

Bazı örnekler: isMatch("aa", "a") false döndürür isMatch("aa", "aa") true döndürür isMatch("aaa", "aa") false isMatch("aa", "a*") döndürme true isMatch("aa", ".") döndürme true isMatch("ab", ".") döndürme true isMatch("aab", "c*") a*b") true değerini döndürür

9.1 Analiz

Her şeyden önce, bu en zor problemlerden biridir. Tüm farklı durumları düşünmek zor. Sorun, 2 temel durumu ele alacak şekilde basitleştirilmelidir :

- kalının ikinci karakteri: • kalının ikinci karakteri değil

1. durum için , desenin ilk karakteri "." değilse, desenin ilk karakteri ve dize aynı olmalıdır. Ardından kalan kısmı eşitmeye devam edin. 2. durum için , desenin ilk karakterini eşitmeyen karakterin ilk i

9.2 Java Çözümü 1 (Kısa)

Aşağıdaki Java çözümü kabul edilir.

genel sınıf Çözümü

```
{ genel boolean isMatch(String s, String p) {
    if(p uzunluk() == 0) dönüş
        s uzunluk() == 0;

    //p'nin 1 uzunluğu özel bir durumdur
    if(p.length() == 1 || p.charAt(1) != '*') if(s.length() < 1 || (p.charAt(0) != '.' && s.charAt(0) != p.charAt(0))) yanlış dönüş;

    dönüş isMatch(s.substring(1), p.substring(1));

}başka{
```

9 Java'da Normal İfade Eşleştirme

```
int len = s.uzunluk();

int ben = -1;

while(i<len && (i < 0 || p.charAt(0) == '.' || p.charAt(0) ==
s.charAt(i)))
{ if(isMatch(s.substring(i+1), p.substring(2)))
    doğru dönüs; ben++;

    } yanlış döndür;
}
}
}
```

9.3 Java Çözümü 2 (Daha Okunabilir)

```
public boolean isMatch(String s, String p) { // temel durum if
(p.length() == 0) {

    dönüş s.uzunluk() == 0;
}

// özel durum if
(p.uzunluk() == 1) {

    // s'nin uzunluğu 0 ise, false döndür if (s.uzunluk() < 1)
    { false döndür;

    }

    // ilki eşleşmezse, false döndürün else if ((p.charAt(0) !=
s.charAt(0)) && (p.charAt(0) != '.')) { false döndür;

    }

    // aksi takdirde, s ve p dizisinin geri kalanını karşılaştırın. başka { dönüş
isMatch(s.substring(1), p.substring(1));

    }

}

// durum 1: p'nin ikinci karakteri '*' olmadığından if (p.charAt(1) != '*')
{ if (s.length() < 1) { false döndür;

    }

    } if ((p.charAt(0) != s.charAt(0)) && (p.charAt(0) != '.')) {
```

9 Java'da Normal İfade Eşleştirme

```
yanlış dönüş; } else
{ return
    isMatch(s.substring(1), p.substring(1));
}
}

// durum 2: p'nin ikinci karakteri '*' olduğunda, karmaşık durum. else { //durum 2.1: bir karakter
& '*', 0 eleman yerine geçebilir if (isMatch(s, p.substring(2))) { return true;

}

//durum 2.2: bir karakter & '*' önceki 1 veya daha fazla ögeyi temsil edebilir, //bu nedenle her alt dizгиyi deneyin
int i = 0; while (i<s.length() && (s.charAt(i)==p.charAt(0) || p.charAt(0)=='.')){

    if (isMatch(s.substring(i + 1), p.substring(2)))) {
        doğru dönüş;
    }
    ben++;
}

} yanlış döndür;
}
}
```

10 Birleştirme Aralığı

Sorun:

Bir aralık koleksiyonu verildiğinde, örtüsen tüm aralıkları birleştirin.

Örneğin, Verilen
[1,3],[2,6],[8,10],[15,18], [1,6],[8,10],[15,18]
döndür .

10.1 Bu Sorunla İlgili Düşünceler

Bu sorunu çözmenin anahtarı, Aralıkların dizi listesini sıralamak için önce bir Karşılaştırıcı tanımlamaktır. Ve sonra bazı aralıkları birleştirin.

Bu problemden çıkarılacak mesaj, sıralanmış liste/ar'in avantajını kullanmaktadır. İşin

10.2 Java Çözümü

```
sınıf Aralık { int başlangıç;
    int sonu;

    Aralık() { başlangıç
        = 0; bitiş = 0;

    }

    Aralık(int s, int e) { başlangıç = s; bitiş
        = ve;

    }

    genel sınıf Çözümü
    { public ArrayList<Interval> birleştirme(ArrayList<Interval> intervaller) {

        if (aralıklar == null || interval.size() <= 1)
            dönüş aralıkları;

        // kendi tanımlı Karşılaştırıcıyı kullanarak aralıkları sırala
```

10 Birleştirme Aralığı

```
Collections.sort(aralıklar, yeni IntervalComparator());  
  
ArrayList<Interval> sonuç = new ArrayList<Interval>();  
  
Aralık önceki = interval.get(0); for (int i = 1; i <  
interval.size(); i++) { Interval curr = interval.get(i);  
  
if (prev.end >= curr.start)  
{ // birleştirilmiş vaka  
    Aralık birleştirildi = new Interval(prev.start, Math.max(prev.end, curr.end)); önceki =  
    birleştirilmiş; } başka { sonuç.add(önceki);  
  
    önceki = geçerli;  
}  
}  
  
sonuç.add(önceki);  
  
dönüş sonuç;  
}  
}  
  
class IntervalComparator, Comparator<Interval> uygular { public int  
Compare(Interval i1, Interval i2) { return i1.start - i2.start;  
  
}  
}
```

11 Aralık Ekle

Sorun:

Bir dizi örtüşmeyen ve sıralanmış aralık verildiğinde, aralıklara yeni bir aralık ekleyin (gerekirse birleştirin).

Örnek 1:

Verilen aralıklar [1,3],[6,9], [2,5]'i [1,5],[6,9] olarak ekleyin ve birleştirin.

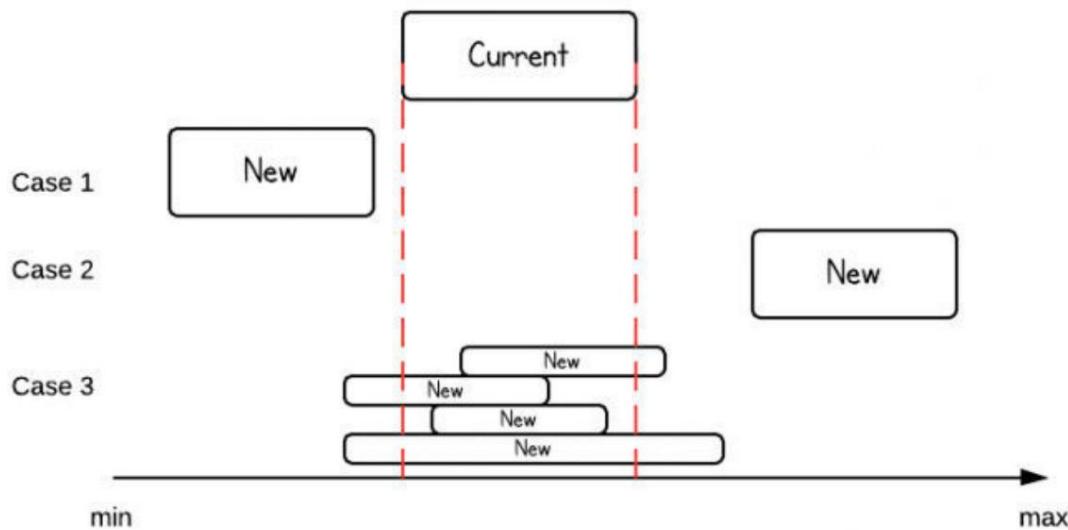
Örnek 2:

[1,2],[3,5],[6,7],[8,10],[12,16] verildiğinde, [4,9]'u şu şekilde ekleyin ve birleştirin: [1,2],[3,10], [12,16].

Bunun nedeni, [yeni](#) [4,9] aralığının [3,5],[6,7],[8,10] ile çakışmasıdır.

11.1 Bu Sorunla İlgili Düşünceler

3 vakayı hızlıca özetleyin . Kavşak olduğunda, yeni bir aralık oluşturulur.



11.2 Java Çözümü

11 Aralık Ekle

```
/**  
 * Aralık tanımı.  
 * genel sınıf Aralığı { int start; niyet  
 *      etmek;  
 *  
 *      Aralık() { başlangıç = 0; bitiş = 0; }  
 * Aralık(int s, int e) { başlangıç = s; bitiş = e; } */
```

genel sınıf Çözümü

```
{ public ArrayList<Interval> ekle(ArrayList<Interval> intervaller, Interval newInterval) {  
  
    ArrayList<Interval> sonuç = new ArrayList<Interval>();  
  
    for(Aralık aralığı: aralıklar){  
        if(interval.end < newInterval.start){ sonuç.add(interval);  
  
    }else if(interval.start > newInterval.end){ sonuç.add(newInterval);  
        yeni Aralık = aralık;  
  
    }else if(interval.end >= newInterval.start || interval.start <= newInterval.end){ newInterval =  
        new Interval(Math.min(interval.start,  
  
                           newInterval.start), Math.max(newInterval.end, interval.end));  
    }  
    }  
  
    sonuç.add(yeniAralık);  
  
    dönüş sonucu;  
}
```

12 İki Toplam

Bir tam sayı dizisi verildiğinde, toplamları belirli bir hedef sayı olacak şekilde iki sayı bulun.

twoSum işlevi, iki sayının dizinlerini, dizin1'in dizin2'den küçük olması gerektiği yerde, hedefi toplayacak şekilde döndürmelidir .Lütfen, döndürülen yanıtlarınızın (hem dizin1 hem de dizin2) sıfır tabanlı olmadığına dikkat edin. Örneğin:

Giriş: sayılar={2, 7, 11, 15}, hedef=9

Cıktı: indeks1=1, indeks2=2

12.1 Naif Yaklaşım

Bu sorun oldukça basittir. Bu tamsayı dizisindeki olası her sayı çiftini basitçe inceleyebiliriz.

En kötü durumda zaman karmaşıklığı: $O(n^2)$.

```
genel statik int[] ikiToplam(int[] sayılar, int hedef) { int[] ret = yeni int[2]; for (int
i = 0; i < sayılar.uzunluk; i++) {
    for (int j = i + 1; j < sayılar.uzunluk; j++) { if (sayılar[i] + sayılar[j] ==
        hedef) { ret[0] = i + 1; ret[1] = j + 1;
    }
}
} sağa dön;
```

Daha iyisini yapabilir miyiz?

12.2 Daha İyi Çözüm

Hedef değeri depolamak için HashMap'i kullanın.

```
genel sınıf Çözümü
{ public int[] twoSum(int[] sayılar, int hedef) {
    HashMap<Tamsayı, Tamsayı> map = new HashMap<Tamsayı, Tamsayı>(); int[] sonuç = yeni
    int[2];
```

12 İki Toplam

```
for (int i = 0; i < sayılar.uzunluk; i++) { if  
    (map.containsKey(sayılar[i])) { int dizin =  
        map.get(sayılar[i]); sonuç[0] = dizin+1 ; sonuç[1] =  
        i+1; kirmak; } başka { map.put(hedef - sayılar[i], i);  
  
    }  
}  
  
dönüş sonuç; }  
  
}
```

Zaman karmaşıklığı, normalde O(1) olan HashMap'in koy ve al işlemlerine bağlıdır .

Bu çözümün zaman karmaşıklığı O(n)'dir.

13 İki Toplam II Giriş dizisi sıralanır

Bu problem Two Sum'a benzer .

Bu sorunu çözmek için, diziyi her iki taraftan taramak için iki nokta kullanabiliriz. Görmek Aşağıdaki Java çözümü:

```
public int[] ikiToplam(int[] sayılar, int hedef) { if (sayılar == boş ||  
    sayılar.uzunluk == 0) null döndürür;  
  
    int ben = 0;  
    int j = sayılar.uzunluk - 1;  
  
    while (i < j) { int x =  
        sayılar[i] + sayılar[j]; eğer (x < hedef) { ++i; } else if (x  
        > hedef) { j--; } else { yeni dönüş int[]  
        { ben + 1, j + 1 };  
  
    }  
    }  
  
    boş dönüş;  
}
```

14 Two Sum III Veri yapısı tasarıımı

Bir TwoSum sınıfı tasarlayın ve uygulayın. Şu işlemleri desteklemelidir: ekle ve bul.

add - Sayıyı dahili bir veri yapısına ekleyin. find - Toplamı değere eşit olan herhangi bir sayı çifti olup olmadığını bulun.

Örneğin,

```
ekle(1);
ekle(3);
ekle(5);
bul(4) -> doğru bul(7) ->
yanlış
```

14.1 Java Çözümü

İstenilen sınıfın ekleme ve alma işlemlerine ihtiyacı olduğundan HashMap bu amaç için iyi bir seçenektedir.

```
genel sınıf TwoSum
{ private HashMap<Tamsayı, Tamsayı> öğeleri = yeni HashMap<Tamsayı,
  Tamsayı>();
  }

  public void add(int sayı) { if
    (elements.containsKey(sayı)) { elements.put(sayı,
      elements.get(sayı) + 1); } başka { elementler.put(sayı, 1);

    }
  }

  genel boole bul(int değeri) {
    for (Tamsayı i : elements.keySet()) { int hedef = değer - i; if
      (elements.containsKey(hedef)) {

        if (i == hedef && elements.get(target) < 2) { devam et;

        } true döndür;
      }
    }

    yanlış döndür;
  }
```

14 Two Sum III Veri yapısı tasarımı

```
    }  
}
```

15 3 Toplam

Sorun:

n tam sayıdan oluşan bir S dizisi verildiğinde, S'de $a + b + c = 0$ olacak şekilde a, b, c elemanları var mı?
Dizideki sıfırın toplamını veren tüm benzersiz üçlüleri bulun.

Not: (a,b,c) üçlüsünde bulunan elemanlar azalan sırada olmamalıdır. (yani a < b < c) Çözüm kümesi
yinelenen üçüzler içermemelidir.

Örneğin, S = {-1 0 1 2 -1 -4} dizisi verildiğinde,

Bir çözüm kümesi:

(-1, 0, 1) (-1, -1,
2)

15.1 Naif Çözüm

Naif çözüm 3 döngüdür ve bu, zaman karmaşıklığını $O(n^3)$ verir. Görünüşe göre bu kabul edilebilir
bir çözüm değil, ancak buradan bir tartışma başlayabilir.

genel sınıf Çözüm { genel

ArrayList<ArrayList<Integer>> threeSum(int[] num) { //sıralama dizisi Arrays.sort(num);

```
ArrayList<ArrayList<Tamsayı>> sonuç = yeni
    ArrayList<ArrayList<Tamsayı>>();
    ArrayList<Tamsayı> her biri = new ArrayList<Tamsayı>(); for(int i=0; i<num.length; i+
    +){ if(num[i] > 0) break;

    for(int j=i+1; j<num uzunluk; j++){ if(num[i] + num[j] > 0
        && num[j] > 0) ara;

        for(int k=j+1; k<num uzunluk; k++){
            if(sayı[i] + sayı[j] + sayı[k] == 0) {

                herbiri.add(sayı[i]); her
                biri.add(num[j]);
                herbiri.add(num[k]);
                sonuç.add(her); her
                biri.clear();

            }
        }
    }
}
```

15.3 Toplam

```
        }
    }
}

dönüş sonucu;
}
}
```

* Çözüm ayrıca kopyaları işlemez. Bu nedenle, sadece zaman açısından verimsiz değil, aynı zamanda yanlıştır.

Sonuç:

Gönderim Sonucu: Çıkış Limiti Aşıldı

15.2 Daha İyi Çözüm

Daha iyi bir çözüm, bir yerine iki işaretçi kullanmaktadır. Bu, $O(n^2)$ 'nin zaman karmaşıklığını sağlar.

Yinelemeyi önlemek için, sıralanmış dizilerden faydalananızıza, yani işaretçileri şu noktalara göre hareket ettirebiliriz: >1 , aynı öğeyi yalnızca bir kez kullanmak için.

```
public ArrayList<ArrayList<Tamsayı>> üçSum(int[] num) {
    ArrayList<ArrayList<Tamsayı>> sonuç = new ArrayList<ArrayList<Tamsayı>>();

    eğer (say.uzunluk < 3)
        sonuç döndürür;

    // sıralama dizisi
    Diziler.sort(num);

    for (int i = 0; i < sayı.uzunluk - 2; i++) { // //yinelenen
        çözümlerden kaçının if (i == 0 || sayı[i] > sayı[i - 1]) {

            int negatif = -num[i];

            int başlangıç = ben + 1;
            int bitiş = sayı.uzunluk - 1;

            while (start < end) { //case 1 if
                (num[start] + num[end] ==
                negate) { ArrayList<Tamsayı>
                    temp = new ArrayList<Tamsayı>(); temp.add(sayı[i]); temp.add(num[başlangıç]);
                    temp.add(sayı[son]);

                    sonuç.add(geçici);
                }
            }
        }
    }
}
```

```
+ başlat;  
oğul--; //  
tekrarlanan çözümlerden kaçının ( start < end  
&& num[end] == num[end + 1])  
oğul--;  
  
while (başlangıç < bitiş && sayı[başlangıç] == sayı[başlangıç - 1])  
+ başlat; //  
durum 2 } else if  
(sayı[başlangıç] + sayı[bitiş] < olumsuzla) { +  
başlat; // durum 3 } else  
  
{ son--;  
  
}  
}  
  
}  
}  
  
dönüş sonucu;  
}
```

16 4 ben

n tam sayıdan oluşan bir S dizisi verildiğinde, S'de $a + b + c + d = \text{hedef}$ olacak şekilde a, b, c ve d öğeleri var mı? Hedefin toplamını veren dizideki tüm benzersiz dördüzleri bulun.

Not: Dörtlü (a,b,c,d) içindeki elemanlar azalan olmayan sırada olmalıdır. (yani, bir d)

B c Çözüm kümesi yinelenen dördüzler içermemelidir.

Örneğin, verilen S = {1 0 -1 0 -2 2} dizisi ve hedef = 0.

Bir çözüm kümesi: (-1, 0, 0,
1) (-2, -1, 1, 2) (-2, 0, 0, 2)

16.1 Düşünceler

Tipik bir k-toplam problemi. Zaman N üzeri ($k-1$)'dir.

16.2 Java Çözümü

```
public ArrayList<ArrayList<Tamsayı>> fourSum(int[] sayı, int hedef) {
    Diziler.sort(sayı);

    HashSet<ArrayList<Tamsayı>> hashSet = new HashSet<ArrayList<Tamsayı>>();
    ArrayList<ArrayList<Tamsayı>> sonuç = new ArrayList<ArrayList<Tamsayı>>();

    for (int i = 0; i < sayı.length; i++) {
        for (int j = i + 1; j < sayı.length; j++) {
            int k = j + 1; int l =
            sayı.length - 1;

            iken (k < l) {
                int toplam = sayı[i] + sayı[j] + sayı[k] + sayı[l];

                if (toplam > hedef) { l--; } else
                    if (toplam < hedef) { k++; }

                } else if (toplam == hedef)
                    { ArrayList<Tamsayı> temp = new ArrayList<Tamsayı>(); temp.add(sayı[i]);

```

16 4 ben

```
temp.add(num[j]);
temp.add(num[k]);
temp.add(num[l]);

if (!hashSet.contains(temp))
    { hashSet.add(temp); sonuç.add(geçici);

}

k++;
ben--;
}
}
}

dönüş sonuç;
}
```

İşte ArrayList'in hashCode yöntemi. İki listenin tüm öğeleri aynıysa, iki listenin hash kodunun da aynı olmasını sağlar. ArrayList'teki her öğe Tamsayı olduğundan, aynı tamsayı aynı hash koduna sahiptir.

```
int hashCode = 1;
Yineleyici<E> i = list.iterator(); while
(i.hasNext()) { E nesne = i.next(); hashCode =
    31*hashCode + (obj==null ? 0 :
    obj.hashCode());
}
```

17 3Toplam En Yakın

n tam sayıdan oluşan bir S dizisi verildiğinde, S'de toplam belirli bir sayıya en yakın olacak şekilde üç tam sayı bulun, hedef. Üç tam sayının toplamını döndürür. Her girdinin tam olarak bir çözümü olacağını varsayıbilirsiniz.

Örneğin, verilen $S = \{-1 2 1 -4\}$ dizisi ve hedef = 1.
Hedefe en yakın toplam 2'dir ($-1 + 2 + 1 = 2$).

17.1 Analiz

Bu problem 2 Sum'a benzer . Bu tür bir problem, benzer bir yaklaşım kullanılarak, yani hem soldan hem de sağdan iki işaretçi kullanılarak çözülebilir.

17.2 Java Çözümü

```
public int ThreeSumClosest(int[] sayılar, int hedef) {
    int min = Tamsayı.MAX_VALUE; int sonuç
    = 0;

    Diziler.sort(sayılar);

    for (int i = 0; i < sayı.uzunluk; i++)
        { int j = ben + 1; int k =
        sayı.uzunluk - 1; while (j < k) { int
        toplam = sayılar[i] + sayılar[j] +
        sayılar[k]; int fark = Math.abs(toplam - hedef);

        if(diff == 0) toplamı döndürür ;

        eğer (fark < min) { min =
        fark; sonuç = toplam;

        } if (toplam <= hedef) { j++; }
        başka { k--;
        }

    }
```

17 3Toplam En Yakın

```
    }  
  
    dönüş sonucu;  
}
```

Zaman Karmaşıklığı $O(n^2)$.

18 Dizeden Tamsayıya (atoi)

Bir diziyi bir tamsayıya dönüştürmek için atoi'yi uygulayın.

İpucu: Olası tüm girdi durumlarını dikkatlice değerlendirin. Bir meydan okuma istiyorsanız, lütfen aşağıyı görmeyin ve kendinize olası girdi durumlarının neler olduğunu sorun.

18.1 Analiz

Bu sorun için aşağıdaki durumlar dikkate alınmalıdır:

1. `null` veya boş dize 2. beyaz boşluklar 3. +/- işaretini 4. gerçek değeri hesapla
5. min ve maks.

18.2 Java Çözümü

```
public int atoi(String str) { if (str == null ||  
    str.length() < 1) 0 döndürür ;  
  
    // beyaz boşlukları kırp str =  
    str.trim();  
  
    karakter bayrağı = '+';  
  
    // negatif veya pozitif kontrol et int i = 0;  
    if (str.charAt(0) == '-') { bayrak = '-'; ben++;  
  
    } else if (str.charAt(0) == '+') {  
        ben++;  
  
    } // sonucu saklamak için double kullan  
    double sonuç = 0;  
  
    // while değerini hesapla  
    (str.length() > i && str.charAt(i) >= '0' && str.charAt(i) <= '9') { *  
        10 + (str.charAt(i) - '0'); sonuç = sonuç
```

18 Dizeden Tamsayıya (atoi)

```
ben++;
}

eğer (bayrak == '-')
    sonuç = -sonuç;

// eğer (sonuç >
Tamsayı.MAX_VALUE) dönüş
Tamsayı.MAX_VALUE ise maks ve min'i işle ;

eğer (sonuç < Tamsayı.MIN_VALUE)
    Tamsayı.MIN_VALUE döndürürse ;

dönüş (int) sonucu;
}
```

19 Sıralanmış Diziyi Birleştir

İki sıralı tamsayı dizisi A ve B verildiğinde, B'yi tek bir sıralı dizi olarak A ile birleştirin.

Not: A'nın ek öğeleri tutmak için yeterli alana sahip olduğunu varsayıbilirsiniz.

B. A ve B'de başlatılan öğelerin sayısı sırasıyla m ve n'dir.

19.1 Analiz

Bu sorunu çözmenin anahtarı, A ve B öğelerini geriye doğru hareket ettirmektir. A yapıldıktan sonra B'de kalan bazı öğeler varsa, bu durumu da halletmeniz gereklidir.

Bu problemden çıkarılacak mesaj, döngü koşuludur. Bu tür Koşul, [iki sıralı bağıntılı listeyi birleştirmek](#) için de kullanılabilir .

19.2 Java Çözümü 1

```
genel sınıf Çözümü {
    genel geçersiz birleştirme(int A[], int m, int B[], int n) {

        while(m > 0 && n > 0){
            eğer(A[m-1] > B[n-1]){
                A[m+n-1] = A[m-1];
                M--;
            }başka{
                A[m+n-1] = B[n-1];
                N--;
            }
        }

        while(n > 0){
            A[m+n-1] = B[n-1];
            N--;
        }
    }
}
```

19.3 Java Çözümü 2

Döngü koşulu da aşağıdaki gibi $m+n$ kullanabilir.

19 Sıralanmış Diziyi Birleştir

```
genel geçersiz birleştirme(int A[], int m, int B[], int n) { int ben = m - 1; int  
j = n - 1; int k = m + n - 1;
```

```
    iken (k >= 0) { eğer (j < 0  
        || (i >= 0 && A[i] > B[j]))  
        A[k--] = A[i--]; başka  
        A[k--] = B[j--];  
    }  
}
```

20 Geçerli Parantez

Yalnızca '(', ')', "[", "]" ve "[" karakterlerini içeren bir dizî verildiğinde, giriş dizisinin geçerli olup olmadığını belirleyin. Köşeli parantezler doğru sırada kapanmalıdır, "(" ve "()" hepsi geçerlidir ancak "[]" ve "[]]" değildir.

20.1 Analiz

Yığın veri yapısı kullanılarak çözülebilten tipik bir problem.

20.2 Java Çözümü

```
public static boolean isValid(String s) { HashMap<Karakter,
    Karakter> harita = yeni HashMap<Karakter, Karakter>(); map.put('(', ')'); map.put('[', ']'); map.put('{', '}');

    Yığın<Karakter> yığın = yeni Yığın<Karakter>();

    for (int i = 0; i < s.length(); i++)
        { char curr = s.charAt(i);

            if (map.keySet().contains(curr)) {
                donanım.push(dur); } else
            if (map.values().contains(curr))
                { if (!stack.empty() && map.get(stack.peek()) == curr) { stack.pop(); } başka { yanlış
                    döndür;

                }
            }
        }

        yığın.bos ();
    }
```

21 En Uzun Geçerli Parantez

Yalnızca '(' ve ')' karakterlerini içeren bir dizi verildiğinde, en uzun geçerli (iyi biçimlendirilmiş) parantez alt dizisinin uzunluğunu bulun.

"()" için, geçerli en uzun parantez alt dizisi "()" olup uzunluğu = 2'dir. Başka bir örnek "()()" olup, burada en uzun geçerli parantez alt dizisi "()()" dir. , uzunluğu = 4 olan .

21.1 Analiz

Bu sorun , bir yığın kullanılarak çözülebilir Geçerli Parantezlere benzer .

21.2 Java Çözümü

```
genel statik int ve uzunValidParentheses(String s) {  
    Yığın<int[]> yığın = yeni Yığın<int[]>(); int sonuç = 0;  
  
    for(int i=0; i<=s.length()-1; i++){ char c = s.charAt(i);  
        if(c=='('){ int[] a = {i,0}; stack.push(a); }  
        else{ if(stack.empty() || stack.peek()[1]== 1){ int[] a  
            = {i,1}; stack.push(a); }else{ stack.pop(); int  
            currentLen=0; if(stack.empty()){  
  
                akımLen = i+1; }  
                else{ currentLen = i  
                    stack.peek()[0];  
  
                } sonuç = Math.max(sonuç, geçerliLen);  
            } } }  
    dönüş sonucu;
```

21 En Uzun Geçerli Parantez

}

22 strStr()'yi uygula

Sorun:

strStr()'yi uygulayın. Samanlıkta iğnenin ilk geçtiği yerin dizinini veya iğne samanlığının parçası değilse -1'i döndürür.

22.1 Java Çözümü 1 - Naif

```
public int strStr(String samanlık, String iğnesi)
    { if(samanlık==null || iğne==null) 0 dönüşü ;

        if(iğne.uzunluk() == 0) 0 döndürür ;

        for(int i=0; i<samanlık.uzunluk(); i++){
            if(i + iğne.uzunluk() > samanlık.uzunluk()) dönüş -1;

            int m = ben;
            for(int j=0; j<iğne.uzunluk(); j++){
                if(needle.charAt(j)==haystack.charAt(m)){ if(j==needle.length()-1)
                    dönüş i;

                m++;
            }

            } başka{ mola;

        }

        -1 döndürür ;
    }
```

22.2 Java Çözümü 2 - KMP

KMP algoritmasını anlamak için [bu makaleye](#) göz atın

```
public int strStr(String samanlık, String iğnesi) {
    if(samanlık==null || iğne==null)
```

22 strStr()'yi uygula

```
0 dönüşü ;  
  
int h = samanlık.uzunluk(); int n =  
iğne.uzunluk();  
  
eğer (n > h)  
-1 döndürür ;  
eğer (n == 0) 0  
döndürürse ;  
  
int[] sonraki = getNext(iğne); int ben = 0;  
  
while (i <= h - n) { int başarı =  
1; for (int j = 0; j < n; j++)  
{ if (needle.charAt(0) != samanlık.charAt(i))  
{ başarı = 0; ben++; kırmak;  
  
} else if (needle.charAt(j) != haystack.charAt(i + j)) { başarı = 0; ben = ben + j -  
sonraki[j - 1]; kırmak;  
  
}  
  
}  
  
if (başarı == 1) i  
döndürür ;  
}  
  
-1 döndürür ;  
}  
  
//KMP dizisini hesapla public  
int[] getNext(String iğnesi) {  
int[] sonraki = yeni int[iğne.uzunluk()]; sonraki[0] = 0;  
  
for (int i = 1; i < iğne.uzunluk(); i++) {  
int dizin = sonraki[i - 1]; while (dizin  
> 0 && iğne.charAt(dizin) != iğne.charAt(i))  
{ dizin = sonraki[dizin - 1];  
}  
  
if (iğne.charAt(dizin) == iğne.charAt(i))  
{ sonraki[i] = sonraki[i - 1] + 1; } başka  
{ sonraki[i] = 0;  
  
}  
}  
}
```

22 strStr()'yi uygula

```
    sonraki dönüş ;  
}
```

23 Minimum Boyutlu Alt Dizi Toplamı

n pozitif tam sayı dizisi ve s pozitif tam sayısı verildiğinde, toplamı s olan bir alt dizinin minimum uzunluğunu bulun. Bir tane yoksa, bunun yerine 0 döndürün .

Örneğin, [2,3,1,2,4,3] dizisi ve s = 7 verildiğinde, [4,3] alt dizisinin minimum problem kısıtlaması altında 2 uzunluğu .

23.1 Analiz

Kayan pencerenin sol ve sağ sınırlarını işaretlemek için 2 nokta kullanabiliriz . Toplam hedeften büyük olduğunda sol işaretçiyi kaydırın; toplam hedeften az olduğunda, sağ işaretçiyi kaydırın.

23.2 Java Çözümü 1

Basit bir sürgülü pencere çözümü.

```
public int minSubArrayLen(int s, int[] sayı) { if(sayı==null ||  
    sayı.uzunluk==1) 0 döndür ;  
  
    int sonuç = sayı.uzunluk;  
  
    int başlangıç=0;  
    int toplam=0; int  
    ben=0; boolean  
    var = yanlış;  
  
    while(i<=nums.length){ if(sum>=s)  
        { var=true; // böyle bir alt  
        dizi olup olmadığını işaretleyin if(start==i-1){ return 1;  
  
        }  
        başka{ if(i==nums.uzunluk)  
            break;  
        }  
        sonuç = Math.min(sonuç, i-başlangıç);  
        toplam=toplam-sayılar[başlangıç]; + başlat;  
    }  
}
```

23 Minimum Boyutlu Alt Dizi ToplAMI

```
        toplam = toplam+sayilar[i];
        ben++;
    }
}

eğer(varsa)
    dönüş sonucu; aksi
takdirde 0 döndürür;

}
```

23.3 Kullanımdan Kaldırılan Java Çözümü

Bu çözüm işe yarıyor ama daha az okunabilir.

```
public int minSubArrayLen(int s, int[] sayi) {
    if(nums == null || nums.length == 0){ return 0;

} if(sayı.uzunluk == 1 && sayılar[0] < s){ dönüş 0;

}

// giriş dizisi uzunluğu olarak minimum uzunluğu başlat int sonuç =
nums.uzunluk;

int ben =
0; int toplam = sayılar[0];

for(int j=0; j<sayı.uzunluk; ){ if(i==j)
{ if(nums[i]>=s){ return 1; //tek eleman
yeterince büyükse }else{ j++;



if(j<sayı.uzunluk){ toplam
    = toplam + sayılar[j]; }
else{ dönüş sonucu;

}

} }else{ //
toplasm yeterince büyükse, sol imleci hareket ettir if(toplam >=
s){ sonuç = Math.min(j-i+1, sonuç); toplam = toplam -
sayilar[i]; ben++;
```

23 Minimum Boyutlu Alt Dizi Toplamı

```
//toplam yeterince büyük değilse, sağ imleci hareket ettirin }else{ j+
+;

if(j<sayi.uzunluk){ toplam
    = toplam + sayilar[j]; }
else{ if(i==0){ 0 döndürür ; }
    else{ dönüş sonucu;

}

}

}

}

}

}

dönüş sonucu;
}
```

24 Arama Ekle Konum

Sıralanmış bir dizi ve bir hedef değer verildiğinde, hedef bulunursa dizini döndürür. Değilse, dizini sırayla eklenmiş olsaydı olacağı yere döndürün. Dizide kopya olmadığını varsayıbilirsiniz.

İşte birkaç örnek.

```
[1,3,5,6], 5 -> 2  
[1,3,5,6], 2 -> 1  
[1,3,5,6], 7 -> 4  
[1,3,5,6 ], 0 -> 0
```

24.1 Çözüm 1

Naif bir şekilde, diziyi yineleyebilir ve hedefi i . ve $(i+1)$ 'inci elemanla karşılaştırabiliriz.
Zaman karmaşıklığı $O(n)$

```
genel sınıf Çözümü  
{ public int searchInsert(int[] A, int hedef) {  
  
    if(A==null) 0 döndürür ;  
  
    if(hedef <= A[0]) 0 döndürür ;  
  
    for(int i=0; i<A.length-1; i++){ if(hedef > A[i]  
        && hedef <= A[i+1])  
        { i+1'i döndür ;  
        }  
    }  
  
    dönüş A.uzunluk;  
}  
}
```

24.2 Çözüm 2

Bu aynı zamanda bir ikili arama sorunu gibi görünüyor. Karmaşıklığı $O(\log(n))$ yapmaya çalışmalıyız.

```
genel sınıf Çözümü  
{ public int searchInsert(int[] A, int hedef) {
```

24 Arama Ekle Konum

```
if(A==null | |A.length==0) 0
    döndürür;

    return searchInsert(A,hedef,0,A.uzunluk-1);
}

public int searchInsert(int[] A, int hedef, int başlangıç, int bitiş){ int
orta=(başlangıç+bitiş)/2;

if(hedef==A[orta]) döñüş
    orta; else
if(hedef<A[orta])
    döñüş başlangıç<orta?searchInsert(A,hedef,başlangıç,orta-1):başlangıç; başka

    döñüş sonu>orta?searchInsert(A,target,mid+1,end):(end+1);
}
}
```

25 En Uzun Ardışık Dizi

Sıralanmamış bir tamsayı dizisi verildiğinde, ardışık en uzun eleman dizisinin uzunluğunu bulun.

Örneğin, [100, 4, 200, 1, 3, 2] verildiğinde, en uzun ardışık eleman dizisi [1, 2, 3, 4] olmalıdır ~~çalışma zamanı O(n^2)~~ Algoritmanız $O(n)$ karmaşıklığında

25.1 Analiz

$O(n)$ karmaşıklığı gerektirdiğinden, önce diziyi sıralayarak sorunu çözemeyiz. Sıralama en az $O(n \log n)$ zaman alır.

25.2 Java Çözümü

Öğe eklemek ve kaldırırmak için bir HashSet kullanabiliriz. HashSet, bir hash tablosu kullanılarak uygulanır. Öğeler sıralı değil. Ekleme, kaldırma ve içерme yöntemleri sabit zaman karmaşıklığına sahiptir $O(1)$.

```
public static int longArdışık(int[] num) { // eğer dizi boşsa, 0 döndür if
    (num.uzunluk == 0) { 0 döndür ;
}

Set<Tamsayı> set = new HashSet<Tamsayı>(); int maks = 1;

(int e : sayı) için set.add(e);

for (int e : sayı) { int sol = e
    - 1; int sağ = e + 1; int
    sayısı = 1;

while (küme.icerir(sol)) {
    sayı+
    +; set.remove(sol); sol--;
}

}
```

25 En Uzun Ardışık Dizi

```
while (set içerir(sağ))
    { sayı++;
        set.remove(sağ); sağ++;
    }

    maks = Math.max(sayı, maks);
}

maksimum dönüş;
}
```

Bir öğe kontrol edildikten sonra kümeden çıkarılmalıdır. Aksi takdirde, zaman karmaşıklığı, m^2 'nin tüm ardışık dizilerin ortalama uzunluğu olduğu $O(mn)$ olacaktır.

Zaman karmaşıklığını net bir şekilde görebilmeniz için bazı basit örnekler kullanmanızı ve programı manuel olarak çalıştırmanızı öneririm. Örneğin, 1,2,4,5,3 dizisi verildiğinde, program süresi m^2 dir. m , ardışık en uzun dizinin uzunluğudur.

26 Geçerli Palindrom

Bir dizi verildiğinde, yalnızca alfasayısal karakterleri göz önünde bulundurarak ve durumları göz ardı ederek bunun bir palindrom olup olmadığını belirleyin.

Örneğin, "Kırmızı rom, efendim, cinayettir" bir palindromdur, "Programcreek ise harika" değil.

Not: Dizenin boş olabileceğini düşündünüz mü? Bu, bir röportaj sırasında sorulacak iyi bir sorudur.

Bu problemin amacı için, boş diziyi geçerli palindrom olarak tanımlıyoruz.

26.1 Düşünceler

Dize, yani char dizisi olsa da başlangıç ve bitiş döngüsünden. Alfa veya sayı değilse, işaretçileri artırın veya azaltın. Alfa ve sayısal karakterleri karşılaştırın. Aşağıdaki çözüm oldukça basittir.

26.2 Java Çözümü 1 - Naif

genel sınıf Çözümü {

```
genel boolean isPalindrome(String s) {  
  
    if(s == null) false döndürür;  
    if(s.uzunluk() < 2) true döndürür;  
  
    char[] charArray = s.toCharArray(); int len =  
    s.uzunluk();  
  
    int ben=0;  
    int j=len-1;  
  
    while(i<j){ char  
        sol, sağ;  
  
        while(i<len-1 && !isAlpha(sol) && !isNum(sol)){  
            ben+  
            +; sol = charArray[i];  
        }  
  
        while(j>0 && !isAlpha(sağ) && !isNum(sağ)){  
            J--;  
        }  
    }  
}
```

26 Geçerli Palindrom

```
sağ = charArray[j];
}

if(i >= j) ara;

sol = charArray[i]; saag =
charArray[j];

if(!isSame(sol, sağ)){ false döndürür;
}

ben+
+; J--;

} true döndür;
}

public boolean isAlpha(char a){ if((a >= 'a'
&& a <= 'z') || (a >= 'A' && a <= 'Z'))
{ doğru dönüş; } else{ false
döndürür;

}

}

public boolean isNum(char a){ if(a >= '0'
&& a <= '9'){ return true; }else{ false
gönderir;

}

genel boole isSame(char a, char b){ if(isNum(a) &&
isNum(b)){ dönüş a == b;

}else if(Character.toLowerCase(a) == Character.toLowerCase(b))
{ doğru dönüş; } else{ false döndürür;

}

}
```

26.3 Java Çözümü 2 - Yığını Kullanma

Bu çözüm önce özel karakterleri kaldırır. (Tia'ya teşekkürler)

```
genel boolean isPalindrome(String s) {
    s = s.replaceAll("[^a-zA-Z0-9]", "").toLowerCase();

    int len = s.length(); eğer
    (uzunluk < 2)
        doğru dönüş;

    Yığın<Karakter> yığın = yeni Yığın<Karakter>();

    int dizini = 0; while
    (dizin < len / 2) {
        stack.push(s.charAt(dizin)); dizin++;

    }

    if (len % 2 == 1) index++;

    while (dizin < len) { if
        (stack.isEmpty()) false döndürür;

        karakter sıcaklığı = stack.pop(); if
        (s.charAt(dizin) != geçici) yanlış
            dönüş; aksi takdirde

        dizin++;
    }

    doğru dönüş;
}
```

26.4 Java Çözümü 3 - İki İşaretçi Kullanma

Aşağıdaki tartışmada, April ve Frank bu sorunu çözmek için iki işaretçi kullanıyor. Bu çözüm gerçekten basit görünüyor.

```
genel sınıf ValidPalindrome { genel static
    boolean isValidPalindrome(String s){ if(s==null| |s.length()==0) false döndürür;

    s = s.replaceAll("[^a-zA-Z0-9]", "").toLowerCase(); System.out.println(ler);
```

26 Geçerli Palindrom

```
for(int i = 0; i < s.uzunluk(); i++)
{ if(s.charAt(i) != s.charAt(s.length() - 1 - i)){ false döndürür;

}

    doğru dönüş;
}

genel statik geçersiz main(String[] args){
    String str = "Bir adam, bir plan, bir kanal: Panama";

    System.out.println(isValidPalindrome(str));
}
}
```

27 ZigZag Dönüşümü

"PAYPALISHIRING" dizesi, belirli sayıda satırda şu şekilde zikzak şeklinde yazılmıştır:
(daha iyi okunabilirlik için bu deseni sabit bir yazı tipinde görüntülemek isteyebilirsiniz)

PAHN
APLSIIG
YIR

Ve sonra satır satır okuyun: "PAHNAPLSIIGYIR" a yöntemini yazın convert("PAYPALISHIRING",
3) "PAHNAPLSIIGYIR" döndürür.

27.1 Java Çözümü

```
public String convert(String s, int numRows) {  
    eğer (numRows == 1) s  
    döndürürse ;  
  
    StringBuilder sb = yeni StringBuilder(); // adım int  
    adım = 2 * satır sayısı - 2;  
  
    for (int i = 0; i < numRows; i++) { //  
        ilk ve son satırlar if (i ==  
        0 || i == satır sayısı - 1)  
        { for (int j = i; j < s uzunluk(); j = j + adım)  
            { sb.append(s.charAt(j));  
  
        } //orta sıralar }  
        else { int j = i;  
            Boole bayrağı  
            = true; int adım1 = 2 *  
            (satır sayısı - 1 - i); int adım2 = adım - adım1;  
  
            while (j < s uzunluk())  
            { sb.append(s.charAt(j)); eğer  
            (bayrak) j = j + adım1; başka  
  
                j = j + adım2;  
                bayrak = !bayrak;  
            }  
    }  
}
```

27 ZigZag Dönüşümü

```
        }  
    }  
  
    sb.toString();  
}
```

28 İkili Ekle

İki ikili dizi verildiğinde, bunların toplamını (aynı zamanda bir ikili dizi) döndürün.

Örneğin, a = "11", b = "1", dönüş "100" dür.

28.1 Java Çözümü

Çok basit, özel bir şey yok. Bir karakterin int'e nasıl dönüştürüleceğine dikkat edin.

```
public String addBinary(String a, String b)
{
    if(a==null || a.length()==0) return b;
    if(b==null || b.length()==0)

        bir dönüş;

    int pa = a.length()-1; int pb =
    b.length()-1;

    int bayrağı = 0;
    StringBuilder sb = yeni StringBuilder(); while(pa >= 0 || pb
    >=0){ int gider = 0; int vb = 0;

    if(pa >= 0){ va =
        a.charAt(pa)=='0'? 0:1;
        Bu yüzden--;

    } if(pb >= 0){ vb =
        b.charAt(pb)=='0'? 0: 1; pb--;

    }

    int toplam = va + vb + bayrak; if(toplam  >=
    2){ sb.append(String.valueOf(toplam-2));
        bayrak = 1; }başka{ bayrak = 0;
        sb.append(String.valueOf(toplam));

    }

}

eğer(bayrak == 1){
```

28 İkili Ekle

```
        sb.append("1");
    }

Dize tersine çevrildi = sb.reverse().toString(); geri dönüş ;
```

}

29 Son Söz Uzunluğu

Büyük/küçük harflerden ve boşluk karakterlerinden '' oluşan bir dizge verildiğinde, dizgedeki son kelimenin uzunluğunu döndürür. Son kelime yoksa, 0 döndürün.

29.1 Java Çözümü

Çok basit bir soru. Sondan itibaren harflerin başlangıcını işaretlemek için bir bayrağa ihtiyacımız var. Bir harf başlıyorsa ve sonraki karakter bir harf değilse uzunluğu döndürür.

```
public int uzunluklarıOfLastWord(String s) { if(s==null ||  
    s.uzunluk() == 0) dönüş 0;  
  
    int sonuç = 0; int len  
    = s.uzunluk();  
  
    Boole bayrağı = yanlış; for(int  
    i=len-1; i>=0; i--){ char c = s.charAt(i);  
    if(((c>='a' && c<='z') || (c>='A' &&  
    c<='Z')){ flag = true; sonuç++; }else{ if(flag) sonucu döndürür ;  
    }  
    }  
    dönüş sonucu;  
}
```

30 Üçgen

Bir üçgen verildiğinde, yukarıdan aşağıya minimum yol toplamını bulun. Her adımda, aşağıdaki satırdaki bitişik sayılarla geçebilirsiniz.

Örneğin, aşağıdaki üçgen verildiğinde

[

[2],
[3,4],
[6,5,7],
[4,1,8,3]

]

Yukarıdan aşağıya minimum yol toplamı 11'dir (yani $2 + 3 + 5 + 1 = 11$).

Not: Bunu yalnızca O(n) fazladan boşluk kullanarak yapabiliyorsanız bonus puan, burada n üçgendeki toplam satır sayısı.

30.1 Yukarıdan Aşağıya Yaklaşım (Yanlış Cevap!)

Bu çözüm yanlış cevap alır! Daha sonra çalıştırılmaya çalışacağım.

genel sınıf Çözümü

```
{ public int minimumToplam(DiziList<DiziList<Tamsayı>> üçgen) {  
  
    int[] temp = new int[triangle.size()]; int minToplam  
    = Tamsayı.MAX_VALUE;  
  
    for(int i=0; i< temp.length; i++){ temp[i] =  
        Tamsayı.MAX_VALUE;  
    }  
  
    if(triangle.size() == 1) { dönüş  
        Math.min(minTotal, üçgen.get(0).get(0));  
    }  
  
    int birinci = üçgen.get(0).get(0);  
  
    for (int i = 0; i < üçgen.size() - 1; i++) { için  
        (int j = 0; j <= i; j++) {  
  
            int bir, b;  
  
            if(i==0 && j==0){
```

30 Üçgen

```

a = birinci + üçgen.get(i + 1).get(j); b = birinci +
üçgen.get(i + 1).get(j + 1);

}başka( bir
    = geçici[j] + üçgen.get(i + 1).get(j); b = geçici[j] + üçgen.get(i + 1).get(j +
1);

}

temp[j] = Math.min(a, temp[j]); sıcaklık[j + 1]
= Math.min(b, temp[j + 1]);
}
}

for (int e : temp) { if (e <
minToplam) minToplam = e;

}

minToplam'ı döndür ;
}
}

```

30.2 Aşağıdan Yukarıya (İyi Çözüm)

Aslında Üçgenin altından başlayabiliriz.

```

public int minimumToplam(DiziList<DiziList<Tamsayı>> üçgen) { int[] toplam
= yeni int[üçgen.boyut()]; int l = üçgen.size() - 1;

for (int i = 0; ben < üçgen.get(l).size(); i++) { toplam[i]
= üçgen.get(l).get(i);
}

// (int i = üçgen.size() - 2; i >= 0; i--) { için son ikinci
satırda itibaren yinele for (int j = 0; j < üçgen.get(i +
1).size() - 1; j++) {
    toplam[j] = üçgen.get(i).get(j) + Math.min(toplam[j], toplam[j + 1]);
}
}

dönüş toplam[0];
}

```

31 Kopya İçerir

Bir tamsayı dizisi verildiğinde, dizinin herhangi bir kopya içerip içermediğini bulun. Dizide herhangi bir değer en az iki kez görünecekseniz işleviniz true, her öğe farklıysa false döndürmelidir.

31.1 Java Çözümü

```
public boolean içerirDuplicate(int[] nums) { if(nums==null ||  
    nums.length==0) return false;  
  
    HashSet<Tamsayı> set = new HashSet<Tamsayı>(); for(int i: sayılar) {  
  
        if(!set.add(i)){ true  
            return true;  
        }  
    }  
  
    yanlış dönüş;  
}
```

32 Kopya II İçerir

Bir tamsayı dizisi ve bir tamsayı k verildiğinde, yalnızca ve yalnızca dizide $\text{nums}[i] = \text{nums}[j]$ olacak ve i ile j arasındaki fark en fazla k olacak şekilde iki farklı i ve j indeksi varsa true değerini döndürün.

32.1 Java Çözümü 1

```
public boolean içerirNearbyDuplicate(int[] sayılar, int k) {
    HashMap<Tamsayı, Tamsayı> map = new HashMap<Tamsayı, Tamsayı>(); int min =
    Tamsayı.MAX_VALUE;

    for(int i=0; i<sayı.uzunluk; i++)
        { if(map.containsKey(nums[i]))
            { int preIndex = map.get(nums[i]); int boşluk
                = i-preIndex; min = Math.min(min, boşluk);

            } map.put(sayılar[i], i);
        }

    if(min <= k){ true
        döndürür; }
    else{ false döndürür;

    }
}
```

32.2 Java Çözümü 2 - Basitleştirilmiş

```
public boolean içerirNearbyDuplicate(int[] sayılar, int k) {
    HashMap<Tamsayı, Tamsayı> map = new HashMap<Tamsayı, Tamsayı>();

    for(int i=0; i<sayı.uzunluk; i++)
        { if(map.containsKey(nums[i])){
            int pre = map.get(sayılar[i]); if(i
            pre<=k) true döndürür;

        }
    }
```

32 Kopya II İçerir

```
    map.put(sayılar[i], i);  
}  
  
yanlış dönüş;  
}
```

33 Yinelenen III İçerir

Bir tamsayı dizisi verildiğinde, dizide $\text{nums}[i]$ ve $\text{nums}[j]$ arasındaki fark en fazla t ve i ile j arasındaki fark en fazla k olacak şekilde iki farklı i ve j indeksi olup olmadığını bulun.

33.1 Java Çözümü 1

Bu çözüm bir ağaç kümesi kullanır.

$$\begin{array}{cccc} 1 & 2 & 4 & 5 \end{array}$$

$$\text{floor}(3) = 2$$

$$\text{ceiling}(3) = 4$$

Zaman karmaşıklığı $O(n \log(k))$ şeklindedir.

```
genel boolean içerirNearbyAlmostDuplicate(int[] nums, int k, int t) {
    (k < 1 || t < 0) ise false
        döndürür;

    TreeSet<Tamsayı> set = new TreeSet<Tamsayı>();

    for (int i = 0; i < sayı uzunluk; i++) { int c = sayı[i]; if
        ((set.floor(c) != null && c <= set.floor(c) + t) ||
        (set.ceiling(c) != null && c >= set.ceiling(c) - t)) gerçek dönüş;

    set.add(c);

    eğer (i >= k)
        set.remove(sayılar[i - k]);
    }

    yanlış dönüş;
}
```

33 Yinelenen III İçerir

33.2 Java Çözümü 2

Anlaşılması daha kolay olan başka bir çözüm.

```
genel boolean içerirNearbyAlmostDuplicate(int[] nums, int k, int t) {  
    (k < 1 || t < 0) ise false  
    döndürür;  
  
    SortedSet<Uzun> set = new TreeSet<Uzun>();  
  
    for (int j = 0; j < sayı.uzunluk; j++) {  
        uzun solSınır = (uzun) sayılar[j] - t; uzun sağSınır =  
        (uzun) sayılar[j] + t + 1; SortedSet<Uzun> altKüme =  
        küme.altKüme(leftSınır, sağSınır);  
  
        eğer (lsubSet.isEmpty()) true  
        döndürürse;  
  
        set.add((uzun) sayılar[j]);  
  
        if (j >= k)  
            { set.remove((uzun) nums[j - k]);  
            }  
    }  
  
    yanlış dönüş;  
}
```

34 Sıralanmış Diziden Kopyaları Kaldır

Sıralanmış bir dizi verildiğinde, her öğe yalnızca bir kez görünecek ve yeni uzunluğu döndürecek şekilde kopyaları yerinde kaldırın. Başka bir dizi için fazladan yer ayırmayınız, bunu sabit hafıza ile yerinde yapmalısınız. Örneğin, $A = [1,1,2]$ girdi dizisi verildiğinde, işleviniz uzunluk = 2 döndürmelidir, ve A şimdi $[1,2]$ 'dir.

34.1 Düşünceler

Sorun oldukça basit. Dizinin uzunluğunu benzersiz öğelerle döndürür, ancak orijinal dizinin de değiştirilmesi gereklidir. Bu sorun , [Sıralı Dizi II'den Kopyaları Kaldır](#) ile gözden geçirilmelidir .

34.2 Çözüm 1

```
// Orjinal diziyi işleyin public static int
removeDuplicatesNaive(int[] A) { eğer
(A.uzunluk < 2) dönüş A.uzunluk;

int = 0; int
ben = 1;

while (i < A.uzunluk) { if (A[i]
== A[j]) { i++; } başka { j++;
A[j] = A[i]; ben++;

}

}

dönüş j + 1;
}
```

Bu yöntem, benzersiz öğelerin sayısını döndürür, ancak orijinal diziyi doğru şekilde değiştirmez. Örneğin, giriş dizisi 1, 2, 2, 3, 3 ise dizi 1, 2, 3, 3, 3 olarak değiştirilir. Doğru sonuç 1, 2, 3 olmalıdır .

34 Sıralanmış Diziden Kopyaları Kaldır

oluşturulduktan sonra değiştirilemez, orijinal diziyi doğru sonuçlarla döndürmemizin hiçbir yolu yoktur.

34.3 Çözüm 2

```
// Tüm benzersiz öğeleri içeren bir dizi oluşturun public static int[]
removeDuplicates(int[] A) { (A.uzunluk < 2) ise A
    döndürür ;

    int = 0; int ben
    = 1;

    while (i < A.uzunluk) { if (A[i] ==
        A[j]) { i++; } başka { j++; A[j] =
        A[i]; ben++; }

    }
}

int[] B = Diziler.kopyaOf(A, j + 1);

dönüş B;
}

public static void main(String[] args) { int[] dizi = { 1, 2, 2, 3,
    3 }; dizi = kopyaları kaldır(arr); System.out.println(dizi.uzunluk);

}
```

Bu yöntemde yeni bir dizi oluşturulur ve döndürülür.

34.4 Çözüm 3

Yalnızca benzersiz öğelerin sayısını saymak istiyorsak, aşağıdaki yöntem yeterince iyidir.

```
// Benzersiz öğelerin sayısını sayın public static int
countBenzersiz(int[] A) { int sayısı = 0; for (int i = 0; i <
    A.length - 1; i++) { if (A[i] == A[i + 1]) { count++;
```

34 Sıralanmış Diziden Kopyaları Kaldır

```
    }  
  
    } dönüşüm(A.uzunluk - sayıım);  
  
}  
  


```
public static void main(String[] args) { int[] dizi = { 1,
2, 2, 3, 3 }; intboyut = sayıBenzersiz(arr);
System.out.println(boyut);
}
```



---


```

35 Sıralanmış Diziden Kopyaları Kaldır

III

"Kopyaları Kaldır" için takip : Yinelemelere en fazla iki kez izin verilirse ne olur?

Örneğin, A = [1,1,1,2,2,3] sıralanmış dizisi verildiğinde, işleviniz uzunluk döndürmelidir = 5 ve A şimdi [1,1,2,2,3].

35.1 Naif Yaklaşım

"public int removeDuplicates(int[] A)" yöntem imzası göz önüne alındığında, bir tamsayı döndüren bir yöntem yazmamız gerekiyor ve bu kadar. Aşağıdaki çözümü yazdıktan sonra:

```
genel sınıf Çözüm { public int
    removeDuplicates(int[] A)
    { if(A == boş || A.uzunluk == 0) 0
        döndürür;

        int ön = A[0]; Boolean
        bayrağı = yanlış; int sayısı =
        0;

        for(int i=1; i<A.length; i++){ int curr = A[i];

            if(curr == pre){ if(!flag)
                { flag = true; devam
                    etmek; }
                başka{ sayı+
                    +;

            } }
            başka{ ön =
                akım; bayrak = yanlış;
            }
        }

        dönüş A.uzunluk - sayım;
    }
}
```

35 Sıralanmış Diziden Kopyaları Kaldır II

Çevrimiçi Hakem geri döner:

Gönderim Sonucu: Yanlış Cevap

Giriş: [1,1,1,2]

Cıktı: [1,1,1]

Beklenen: [1,1,2]

Dolayısıyla bu problem aynı zamanda yerinde dizi manipülasyonu gerektirir.

35.2 Doğru Çözüm

Verilen dizinin boyutunu değiştiremeyiz, bu nedenle dizinin yalnızca kopyaları kaldırılmış ilk öğesini değiştiririz.

```
genel sınıf Çözüm { public int
    removeDuplicates(int[] A) { if (A == boş || A.length == 0)
        0 dönüşü ;
        int ön = A[0]; Boolean
        bayrağı = yanlış; int sayısı = 0;
        // güncelleme indeksi int o = 1;

        for (int i = 1; i < A.length; i++) { int curr = A[i];
            if (curr == ön) { if (!flag)
                { flag = true; A[o++] =
                    akım;
                    devam
                    etmek; } başka
                    { sayı++;
            } } başka {
                ön = akım;
                A[o++] = akım;
                bayrak = yanlış;
            }
        }
        dönüş A.length - sayı;
    }
}
```

35.3 Daha İyi Çözüm

```
genel sınıf Çözüm { public int
    silDuplicates(int[] A) {
        eğer (A.uzunluk <= 2) dönüş
            A.uzunluk;

        int önceki = 1; // önceki noktayı göster int curr = 2; //
        akımı işaret et

        while (curr < A.uzunluk) {
            if (A[curr] == A[prev] && A[curr] == A[prev - 1]) {
                kür++;
            } başka {
                önceki++;
                A[prev] = A[curr];
                kür++;
            }
        }
        önceki + 1'i döndür;
    }
}
```

36 Tekrarlanmayan En Uzun Alt Dizi Karakterler

Bir dizge verildiğinde, karakterleri tekrar etmeden en uzun alt dizginin uzunluğunu bulun. Örneğin, "abcabcb" için yinelenen harfler içermeyen en uzun alt dize, uzunluğu 3 olan "abc" dir. "bbbbbb" için en uzun alt dize, 1 uzunluğundaki "b" dir .

36.1 Java Çözümü 1

İlk çözüm, CC 150'deki "bir dizide tüm benzersiz karakterlerin olup olmadığını belirleme" sorunu gibidir. Karakterleri tekrar etmeden en uzun alt dizi için mevcut karakterleri izlemek için bir bayrak dizisi kullanabiliriz.

```
public int lengthOfLongestSubstring(String s) { if(s==null)
    dönüş 0; boolean[] bayrağı = yeni boolean[256];

    int sonuç = 0; int
    başlangıç = 0; char[]
    dizi = s.toCharArray();

    for (int i = 0; i < dizi.uzunluk; i++) { char akımı =
        dizi[i]; if (bayrak[geçerli]) {

            sonuç = Math.max(sonuç, i - başlangıç); // döngü
            yeni başlangıç noktasını günceller // ve bayrak
            dizisini sıfırlar // örneğin, abccab, 2. c'ye
            geldiğinde, // 0'dan 3'e kadar başlangıcı günceller, a,b için
            bayrakı sıfırlar ( int k = başlangıç; k < i; k++) { if (arr[k] == akım)
            { başlangıç = k + 1; kırmak;

        } bayrak[dizi[k]] = yanlış;

    } } başka
    { bayrak[geçerli] = doğru;
    }
}
```

Karakterleri Tekrar Etmeden 36 En Uzun Alt Dizi

```
sonuç = Math.max(dizi.uzunluk - başlangıç, sonuç);

dönüş sonuç;
}
```

36.2 Java Çözümü 2

Bu çözüm Tia'dan. Anlaşılması ilk çözümden daha kolaydır.

Temel fikir, mevcut karakterleri ve konumlarını izlemek için bir hash tablosu kullanmaktır. Tekrarlanan bir karakter oluştuğunda, daha önce tekrarlanan karakterden kontrol edin. Ancak, zaman karmaşıklığı daha yüksektir - $O(n^3)$.

```
public static int lengthOfLongestSubstring(String s) { if(s==null) dönüş 0;
    char[] dizi = s.toCharArray(); int = 0;
```

```
HashMap<Karakter, Tamsayı> map = new HashMap<Karakter, Tamsayı>();

for (int i = 0; i < dizi.uzunluk; i++) { if (!
    map.containsKey(dizi[i])) { map.put(dizi[i], i); } else
    { pre = Math.max(pre, map.size()); i =
    map.get(arr[i]); harita.clear(); }

}

return Math.max(pre, map.size());
}
```

Aşağıdaki basit örneği ele alalım.

abcda

Döngü ikinci "a"ya çarptığında, HashMap aşağıdakileri içerir:

bir
0 b 1
c 2
d 3

i indeksi 0'a ayarlanır ve 1 artırılır böylece döngü tekrar ikinci elemandan başlar.

2 İçeren 37 En Uzun Alt Dizi Eşsiz Karakterler

Bu, Google tarafından sorulan bir sorundur.

Bir dizi verildiğinde, yalnızca iki benzersiz karakter içeren en uzun alt diziyi bulun. Örneğin, "abcbbbbcccbdddadacb" verildiğinde, 2 benzersiz karakter içeren en uzun alt dize "bcbbbbccb" dir.

37.1 2 Benzersiz İçeren En Uzun Alt Dizi Karakterler

Bu çözümde, haritadaki benzersiz öğeleri izlemek için bir hashmap kullanılır. Haritaya üçüncü bir karakter eklendiğinde, sol işaretçinin sağa hareket etmesi gereklidir.

Bu çözümde gezinmek için "abac" kullanabilirsiniz.

```
public int lengthOfLongestSubstringTwoDistinct(String s) {  
    int maksimum=0;  
  
    HashMap<Character, Integer> map = new HashMap<Character, Integer>(); int başlangıç=0;  
  
    for(int i=0; i<s.length(); i++){ char c =  
        s.charAt(i); if(map.containsKey(c)){  
  
            map.put(c, map.get(c)+1); }  
        başka{ map.put(c,1);  
  
    }  
  
    if(map.size()>2){ maks =  
        Math.max(maks, i-başlangıç);  
  
        while(map.size()>2){ char t  
            = s.charAt(başlangıç); int sayısı =  
            map.get(t); if(sayı>1){ map.put(t, sayı-1); }  
            başka{ map.remove(t);  
  
        } başlat++;  
    }  
}
```

2 Benzersiz Karakter İçeren 37 En Uzun Alt Dizi

```
    }
}

maks = Math.max(maks, s.uzunluk()-başlangıç);

maksimum dönüş;
}
```

Şimdi, bu soru "k benzersiz karakter içeren en uzun alt dize" olacak şekilde genişletilirse, ne yapmalıyız?

37.2 K Benzersiz Karakter için Çözüm

Aşağıdaki çözüm düzeltildi. "abcadacacaca" ve 3 verildiğinde, "cadcaca" değerini döndürür kumar".

```
public int uzunluklarıOfLongestSubstringKDistinct(String s, int k) {
    int maksimum=0;
    HashMap<Character, Integer> map = new HashMap<Character, Integer>(); int başlangıç=0;

    for(int i=0; i<s.uzunluk(); i++){
        char c =
            s.charAt(i); if(map.containsKey(c)){
                map.put(c, map.get(c)+1);
                başka{ map.put(c,1);
            }

        if(map.size()>k){ max =
            Math.max(max, i-start);

            while(map.size()>k){ char t
                = s.charAt(başlangıç); int
                sayısı = map.get(t); if(sayı>1)
                { map.put(t, sayı-1); }başka{ map.remove(t);

                    } başlat++;
            }
        }
    }

    maks = Math.max(maks, s.uzunluk()-başlangıç);

    maksimum dönüş;
}
```

2 Benzersiz Karakter İçeren 37 En Uzun Alt Dizi

Zaman $O(n)$ 'dir.

Hepsinin Birleştirilmesiyle 38 Alt Dizi Kelimeler

Size bir dize (s) ve hepsi aynı uzunlukta olan bir sözcük listesi (sözcükler) verilir. Sözcüklerdeki her bir kelimenin tam olarak bir kez ve arada herhangi bir karakter olmadan birleştirilmesi olan s'deki alt dize(ler)in tüm başlangıç dizinlerini bulun. Örneğin, verilen: s="barfoothefoobarman" & kelimeler=["foo", "bar"], [0,9] döndürün.

38.1 Analiz

Bu sorun, 2 Benzersiz Karakter İçeren En Uzun Alt Dize'ye benzer (neredeyse aynı) .

Sözlükteki her kelimenin uzunluğu aynı olduğundan, her biri ele alınabilir. tek bir karakter olarak.

38.2 Java Çözümü

```
public List<Tamsayı> findSubstring(String s, String[] kelimeler) { ArrayList<Tamsayı>
    sonuç = new ArrayList<Tamsayı>(); if(s==null | |s.length()==0| |words==null| |
    words.length==0){ dönüş sonucu;
}

//kelimelerin sıklığı
HashMap<String, Integer> map = new HashMap<String, Integer>(); for(Dize w: kelimeler)
{ if(map.containsKey(w)){
    map.put(w, map.get(w)+1); }
    else{ map.put(w, 1);
}

}

int len = kelimeler[0].uzunluk();

for(int j=0; j<len; j++)
    HashMap<String, Integer> currentMap = new HashMap<String, Integer>(); int start = j;//başlangıç
    dizini int sayımı = 0;//şu ana kadar toplam nitelikli kelime sayısı
```

Tüm Sözcüklerin Birleştirilmesiyle 38 Alt Dizi

```
for(int i=j; i<=s.length()-len; i=i+len){ String sub =
    s.substring(i, i+len); if(map.containsKey(sub)){ //mevcut
        haritadaki hızı ayarlanır if(currentMap.containsKey(sub))
        { currentMap.put(sub, currentMap.get(sub)+1); } başka
        {currentMap.put(alternatif, 1);

    }

    sayı++;

    while(currentMap.get(sub)>map.get(sub))
        { Sol dize = s.substring(başlangıç, başlangıç+uzunluk);
            currentMap.put(sol, currentMap.get(sol)-1);

            saymak--;
            başlangıç = başlangıç + uzunluk;
        }

    if(count==words.length)
        { sonuç.add(başlangıç); //sonuca ekle

        //sağa kaydır ve currentMap, say ve başlangıç noktasını sıfırla String left =
        s.substring(start, start+len); currentMap.put(sol, currentMap.get(sol)-1);
        saymak--; başlangıç = başlangıç + uzunluk;

    }

}

else{ currentMap.clear();
    başlangıç \u003d i+len;
    sayı = 0;
}
}

}

dönüş sonuç;
}
```

39 Minimum Pencere Alt Dizisi

Bir S dizisi ve bir T dizisi verildiğinde, $O(n)$ karmaşıklığında T'deki tüm karakterleri içerecek olan S'deki minimum pencereyi bulun.

Örneğin, $S = "ADOBECODEBANC"$, $T = "ABC"$, Minimum pencere "BANC" dir.

39.1 Java Çözümü

```
public String minWindow(String s, String t)
{ if(t.uzunluk()>s.uzunluk()) dönüş
  "";
Dizi sonucu = "";

//t için karakter sayacı
HashMap<Karakter, Tamsayı> hedef = new HashMap<Karakter, Tamsayı>(); for(int i=0;
i<t.length(); i++){ char c = t.charAt(i); if(target.containsKey(c)){ target.put(c,target.get(c)+1); }
başka{ hedef.put(c,1);

}

}

// HashMap<Character, Integer>
için karakter sayacı map = new HashMap<Character, Integer>(); int kaldı = 0; int minLen =
s.uzunluk()+1;

int sayısı = 0; // eşlenen karakterlerin toplamı

for(int i=0; i<s.length(); i++){ char c =
s.charAt(i);

if(target.containsKey(c))
{ if(map.containsKey(c))
{ if(map.get(c)<target.get(c)){ count++;

}

} map.put(c,map.get(c)+1); }
başka{ map.put(c,1); sayı++;

}
```

39 Minimum Pencere Alt Dizisi

```
        }
    }

    if(sayı == t.uzunluk()){
        char sc = s.charAt(sol); iken (!
            map.containsKey(sc) || map.get(sc) > target.get(sc)) { if (map.containsKey(sc) &&
            map.get(sc) > target.get(sc))
                map.put(sc, map.get(sc) - 1); sol++; sc =
            s.charAt(sol);

        }

        if (i - sol + 1 < minLen) { sonuç =
            s.substring(sol, i + 1); minLen = ben - sol + 1;

        }
    }

    dönüş sonucu;
}
```

Bir Dizide 40 Ters Sözcük

Bir giriş dizesi verildiğinde, dizeyi sözcük sözcük ters çevirin.

Örneğin, verilen s = "gökyüzü mavidir", dönüş "mavi gökyüzündür".

40.1 Java Çözümü

Bu sorun oldukça basittir. Önce diziyi word dizisine ayırdık, sonra diziyi yineledik ve her bir öğeyi yeni bir dizgeye ekledik. Not: Çok fazla String oluşturmaktan kaçınmak için String Builder kullanılmalıdır. Dize çok uzunsa, **String değişmez** olduğundan ve çok fazla nesne oluşturulacağından ve çöp toplanacağından, String kullanmak ölçülebilir değildir .

```
sınıf Çözüm { genel
    String reverseWords(String s) { if (s == null || |
        s.length() == 0) { return "";
    }

    // kelimeye boşluk ile bölün
    String[] arr = s.split(" ");
    StringBuilder
    sb = yeni StringBuilder(); for (int i = dizi.uzunluk - 1;
    i >= 0; --i) { if (!arr[i].equals("")) { sb.append(dizi[i]).append(""
    });

    }
} dönüş sb.uzunluk() == 0 ? "" : sb.substring(0, sb.length() - 1);
}
```

41 Döndürülmüş Sıralamada Minimumu Bul

Sıralamak

Sıralanmış bir dizinin önceden bilmediğiniz bir eksende döndürüldüğünü varsayıalım. (yani, 0 1 2 4 5 6 7, 4 5 6 7 0 1 2 olabilir).

Minimum elemanı bulun. Dizide yinelenen bir öğe olmadığını varsayıbilirsiniz.

41.1 Analiz

Bu problem bir ikili aramadır ve anahtar diziyi iki parçaya bölmektir, böylece her seferinde dizinin sadece yarısı üzerinde çalışmamız gereklidir.

Ortadaki öğeyi seçersek, ortadaki öğeyi en soldaki (veya en sağdaki) öğeyle karşılaştırabiliriz. Ortadaki eleman en soldan küçükse, sol yarı seçilmelidir; ortadaki eleman en soldakinden (veya en sağdakinden) büyükse, sağ yarı seçilmelidir. Özyineleme veya yineleme kullanılarak, bu problem zaman günde(n) çözülebilir.

Ek olarak, herhangi bir döndürülmüş sıralanmış dizide, en sağdaki öğe en soldaki öğeden küçük olmalıdır, aksi takdirde sıralanan dizi döndürülmez ve en soldaki öğeyi minimum olarak seçebiliriz.

41.2 Java Çözüm 1 - Özyineleme

Bir yardımcı işlev tanımlayın, aksi takdirde, büyük diziler için pahalı olabilecek Arrays.copyOfRange() işlevini kullanmamız gerekecektir.

```
public int findMin(int[] sayı) { dönüş
    bulMin(sayı, 0, sayı uzunluk - 1);
}

public int findMin(int[] sayı, int sol, int sağ)
{ eğer (sol == sağ) dönüş sayısı[sol]; if ((sağ
    - sol) == 1) return Math.min(num[sol],
    num[sağ]);

int orta = sol + (sağ - sol) / 2;

// döndürülmez if
(sayı[sol] < sayı[sağ]) { dönüş sayı[sol];
```

41 Döndürülülmüş Sıralı Dizide Minimumu Bulma

```
// sağa git } else if
(num[orta] > sayı[sol]) { return findMin(sayı, orta,
    Sağ);

// sola git } else
{ findMin (sayı, sol,
    orta);
}
}
```

41.3 Java Çözümü 2 - Yineleme

```
public int findMin(int[] sayı) { if(sayı uzunluk == 1)
    döndürme sayı[0];

    int sol=0; int
    sağ=sayı uzunluk-1;

    //döndürülmedi
    if(sayılar[sol]<sayılar[sağ])
        dönüş sayıları[sol];

    while(sol <= sağ){ if(sağ-sol==1)
        { dönüş sayıları[sağ];
        }

        int m = güneş + (güneş-güneş)/2;

        if(sayılar[m] > sayılar[sağ]) sol = m;
        başka

        sağ = m;
    }

    dönüş sayıları[sol];
}
```

42 Döndürümüş Sıralı Minimum Bul

Dizi II

42.1 Sorun

"Döndürümüş Sıralı Dizide Minimum Bul" için takip: Yinelemelere izin verilirse ne olur?

Bu çalışma zamanı karmaşıklığını etkiler mi? Nasıl ve neden?

42.2 Java Çözümü

Bu, yinelenen öğeler olmadan döndürümüş sıralanmış dizide minimum öğeyi bulmanın devam eden bir sorunudur. Yalnızca en soldaki öğe ile en sağdaki öğenin eşit olup olmadığını kontrol eden bir koşul daha eklememiz gerekiyor. Eğer öyleyse, onlardan birini bırakabiliyoruz. Aşağıdaki çözümümde, en soldaki en sağa eşit olduğunda sol öğeyi bırakıyorum.

```
public int findMin(int[] sayı) { dönüş
    bulMin(sayı, 0, sayı.length-1);
}

public int findMin(int[] sayı, int güneş, int sağ){
    if(sağ==sol){ dönüş
        sayısı[sol];
    } if (sağ == sol + 1)
        { return Math.min ([so] deyin, [sağ] deyin);

    } // 3 3 1 3 3 3

    int orta = (sağ-sol)/2 + sol; // zaten sıralandı
    if(num[sağ] > sayı[sol])// dönüş num[sol];

    //bir sağa kaydır }else
    if(num[sağ] == num[sol]){
        findMin(sayı, sol+1, sağ) döndürür ; //sağa git } else
        if(num[middle] >= num[left]){

            findMin (sayı, orta, sağ); //
            sola git
```

42 Döndürülü Sıralı Dizide Minimum Bulma II

```
} else{ findMin (sayi, sol, orta); }  
}
```

43 Döndürülmüş Sıralı Dizide Arama

Sıralanmış bir dizinin önceden bilmediğiniz bir eksende döndürüldüğünü varsayıyalım. (yani, 0 1 2 4 5 6 7, 4 5 6 7 0 1 2 olabilir).

Aramanız için size bir hedef değer verilir. Dizide bulunursa, dizinini döndürür, aksi takdirde -1 döndürür. Dizide kopya olmadığını varsayıbilirsınız.

43.1 Java Çözümü 1- Özyinelemeli

```
genel int arama(int[] sayılar, int hedef) {
    ikili Arama döndür (sayılar, 0, sayılar.uzunluk-1, hedef);
}

public int binarySearch(int[] sayılar, int sol, int sağ, int hedef){ if(sol>sağ) dönüş -1;

    int orta = sol + (sağ-sol)/2;

    if(hedef == sayılar[orta]) dönüş
        ortalama;

    if(nums[left] <= nums[mid])
        { if(nums[left]<=hedef && hedef<nums[mid])
            { ikili Arama döndür (sayılar, sol, orta-1, hedef); }else{ dönüş
                ikiliArama(sayılar, orta+1, sağ, hedef);

            } }else
        { if(nums[mid]<target&& target<=nums[right]){
            ikili Aramayı döndür (sayılar, orta+1, sağ, hedef); }else{ dönüş
                ikiliArama(sayılar, sol, orta-1, hedef);

            }
        }
}
```

43.2 Java Çözümü 2 - Yinelemeli

```
genel int arama(int[] sayılar, int hedef) {
```

43 Döndürümüş Sıralı Dizide Arama

```
int kaldı = 0; int
sağ= sayı.uzunluk-1;

while(sol<=sağ){ int orta =
    sol + (sağ-sol)/2; if(target==nums[mid]) ortayı
    döndürür ;

    if(sayılar[sol]<=sayılar[orta])
        { if(nums[left]<=hedef&& hedef<nums[mid]){
            sağ=orta-1; }
        başka{ sol=orta+1;

    } }

    else{ if(nums[mid]<target&& target<=nums[right]){ left=mid+1; }
        başka{ sağ=orta-1;

    }
}

-1 döndürür ;
}
```

44 Döndürülmüş Sıralı Dizide Arama II

"Döndürülmüş Sıralı Dizide Ara" için takip : yinelemelere izin verilirse ne olur? Belirli bir hedefin dizide olup olmadığını belirleyen bir işlev yazın.

44.1 Java Çözümü

```
genel boolean araması(int[] sayılar, int hedef) { int left=0; int
sağ=sayılar.length-1;

while(sol<=sağ){ int orta =
    (sol+sağ)/2; if(sayılar[mid]==hedef) true
    döndürür;

    if(sayılar[sol]<sayılar[orta])
        { if(sayılar[sol]<=hedef&& hedef<sayılar[orta])
            { sağ=orta-1; } başka{ sol=orta+1;

        }

    }else if(sayılar[left]>sayılar[mid]){
        if(sayılar[mid]<target&&target<=sayılar[right]){ left=mid+1; }
        başka{ sağ=orta-1;

    }
}

    başka{ sol++;
}
}

yanlış dönüş;
}
```

45 Dakika Yiğin

Push, pop, top ve minimum öğeyi almayı destekleyen bir yiğin tasarlayın.
sürekli zaman

push(x) – x öğesini yiğinin üzerine itin. pop() – Yiğinin üstündeki öğeyi kaldırır. top() – En üstteki öğeyi alır. getMin() – Yiğindaki minimum öğeyi alır.

45.1 Analiz

17.06.2015 TARİHİNDE GÜNCELLENDİ

getMin()'in sabit zamanını yapmak için minimum öğeyi takip etmemiz gereklidir.
yiğindaki her öğe için.

45.2 Java Çözümü

Öğe değerini, minimum değeri ve aşağıdaki ögelere işaretçiyi tutan bir düğüm sınıfı tanımlayın
BT.

```
sınıf Düğüm { int  
    değer; int  
    dakika;  
    sonraki düğüm;  
  
    düğüm(int x)  
        { değer = x;  
         sonraki = boş;  
         dak = x;  
     }  
}
```

```
sınıf MinYiğin  
{ düğüm başı;  
  
genel geçersiz itme (int x) { if (head  
== null) { kafa = yeni Düğüm(x); }  
else { Düğüm sıcaklığı = yeni  
  
Düğüm(x); temp.min =  
Math.min(head.min, x); temp.sonraki = kafa;
```

45 Dakika Yıgin

```
        kafa = sıcaklık;
    }
}

genel geçersiz pop() { if
    (head == null)

        geri dönmek; baş = baş.sonraki;
    }

genel int top()
{ eğer (kafa == boş) dönüş
    Tamsayı.MAX_VALUE;

    baş değeri döndür;
}

public int getMin() { if (head
== null) dönüş
    Tamsayı.MAX_VALUE;

    dönüş kafası.dk;
}
```

46 Çoğunluk Elemanı

n boyutunda bir dizi verildiğinde, çoğunluk elemanını bulun. Çoğunluk öğesi, $n/2$ defadan fazla görünen öğedir . (dizinin boş olmadığını ve çoğunluk öğesinin her zaman dizide bulunduğu varsayıyalım.)

46.1 Java Çözümü 1 - Naif

Önce diziyi sıralayabiliriz, bu da $n \log(n)$ zamanını alır. Ardından en uzun ardışık alt dizileri bulmak için bir kez tarayın.

```
genel sınıf Çözüm { kamu int
    çoğunlukElement(int[] sayı) {
        if(sayı.uzunluk==1){ dönüş
            sayı[0];
        }

        Diziler.sort(sayı);

        int önceki=sayı[0]; int
        sayısı=1; for(int i=1;
        i<num.length; i++){ if(num[i] == önceki){ count+ +;
            if(sayı > sayı.uzunluk/2) dönüş sayı[i]; }
            başka{ sayı=1; önceki = sayı[i];

        }
    }

    0 dönüşü ;
}
}
```

46.2 Java Çözümü 2 - Çok Daha Basit

SK'ya teşekkürler. Onun çözümü çok daha verimli ve daha basit. Çoğunluk her zaman yarım boşluktan fazlasını kapladığından, ortadaki öğenin çoğunluk olması garanti edilir. Sıralama dizisi $n \log(n)$ alır. Yani bu çözümün zaman karmaşıklığı $n \log(n)$ 'dir. Şerefe!

46 Çoğunluk Elemanı

```
public int çoğulukElement(int[] sayı) { if (sayı.uzunluk  
== 1) { dönüş sayı[0];  
  
}  
  
Diziler.sort(sayı); dönüş  
sayısı[sayı.uzunluk / 2];  
}
```

46.3 Java Çözümü 3 - Doğrusal Zaman Çoğunluk Oylama Algoritması

```
public int çoğulukElement(int[] sayı) { int sonuç = 0, say  
= 0;  
  
for(int i = 0; i<sayı.uzunluk; i++ ) { if(sayı == 0){ sonuç  
= sayılar[ i ]; sayı = 1;  
  
}else if(result == nums[i])  
{ sayı++; } başka{ sayı--;  
  
}  
}  
  
dönüş sonucu;  
}
```

47 Çoğunluk Elemanı II

n boyutunda bir tamsayı dizisi verildiğinde, $n/3$ defadan fazla görünen tüm öğeleri bulun. Algoritma lineer zamanda ve $O(1)$ uzayında çalışmalıdır.

47.1 Java Çözümü 1 - Sayaç Kullanma

Zaman = $O(n)$ ve Uzay = $O(n)$

```
public List<Tamsayı> çoğunlukElement(int[] nums) { HashMap<Tamsayı,
    Tamsayı> harita = yeni HashMap<Tamsayı, Tamsayı>(); for(int i: nums){ if(map.containsKey(i)) {

        map.put(i, map.get(i)+1); }
    başka{ map.put(i, 1);

    }

}

List<Tamsayı> sonuç = new ArrayList<Tamsayı>();

for(Map.Entry<Integer, Integer> girişi: map.entrySet())
{ if(entry.getValue() > nums.length/3)
    { sonuç.add(entry.getKey());
    }
}

dönüş sonuç;
}
```

47.2 Java Çözümü 2

Zaman = $O(n)$ ve Uzay = $O(1)$

Çoğunluk Ögesi I'e bakın.

```
genel Liste<Tamsayı> çoğunlukElement(int[] sayılar) {
    List<Tamsayı> sonuç = new ArrayList<Tamsayı>();

    Tamsayı n1=null, n2=null; int c1=0,
    c2=0;

    for(int i: sayılar){
```

47 Çoğunluk Elemanı II

```
if(n1!=null && i==n1.intValue()){
    c1++;
}else if(n2!=null && i==n2.intValue()){
    c2++; }
başka if(c1==0){ c1=1;
    n1=i; }başka
    if(c2==0){ c2=1;
    n2=i; }başka{ c1--; c2--;
}
c1=c2=0;

for(int i: nums)
{ if(i==n1.intValue()){ c1++; }
    else if(i==n2.intValue())
{ c2++; }

}
}

if(c1>sayılar uzunluk/3)
    sonuç.topla(n1);
if(c2>sayılar uzunluk/3)
    sonuç.topla(n2);

dönüş sonuç;
}
```

48 Elemanı Kaldır

Bir dizi ve bir değer verildiğinde, bu değerin tüm örneklerini yerinde kaldırın ve yeni uzunluğu döndürün. (Not: Öğelerin sırası değiştirilebilir. Yeni uzunluğun ötesinde ne bıraktığınız önemli değildir.)

48.1 Java Çözümü

Bu problem iki indeks kullanılarak çözülebilir.

```
public int RemoveElement(int[] A, int eleman) { int i=0; int = 0;

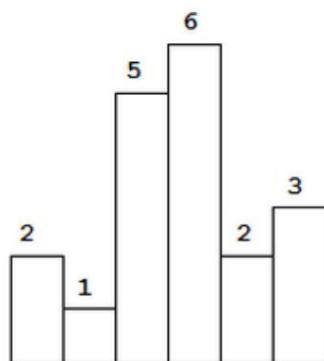
while(j < A.uzunluk){ if(A[j] != elem){ A[i] = A[j]; ben++; }

j++;
}

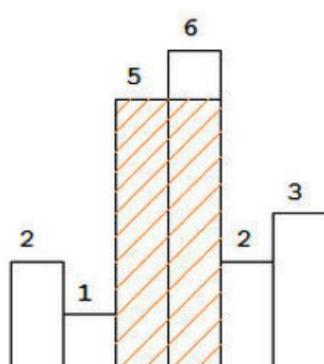
dönüş ben;
}
```

49 Histogramdaki En Büyük Dikdörtgen

Her bir çubuğun genişliğinin 1 olduğu histogramın çubuk yüksekliğini temsil eden negatif olmayan n tamsayı verildiğinde , histogramdaki en büyük dikdörtgenin alanını bulun.



Yukarıda, her çubuğun genişliğinin 1 olduğu, yükseklik = [2,1,5,6,2,3] verilen bir histogram var.



Örneğin, verilen yükseklik = [2,1,5,6,2,3], 10 döndürür .

49.1 Analiz

Bu sorunu çözmenin anahtarı, çubukların dizinlerini depolamak için bir yığın oluşturmaktır. Yığın yalnızca artan çubukları tutar.

49 Histogramdaki En Büyük Dikdörtgen

49.2 Java Çözümü

```
public int en büyük DikdörtgenAlan(int[] yükseklik) { if (yükseklik  
== boş || yükseklik.uzunluk == 0) { dönüş 0;  
  
}  
  
Yığın<Tamsayı> yığın = yeni Yığın<Tamsayı>();  
  
int maks = 0;  
int ben = 0;  
  
while (i < height.uzunluk) { //geçerli  
    yükseklik öncekinden daha büyük olduğunda dizini istiflemek için itin  
    bir  
    if (stack.isEmpty() || yükseklik[i] >= yükseklik[stack.peek()]) {  
        yığın.push(i); ben+  
        +; } else { //mevcut  
            yükseklik öncekinden  
            daha az olduğunda maksimum değeri hesapla  
            bir  
            int p = bileşenler.pop(); int  
            h = yükseklik[p]; int w =  
            bileşenler.isEmpty() ? i : i - stack.peek() - 1; max = Math.max(h * w, max);  
  
    }  
  
    }  
  
    while (!stack.isEmpty()) { int p =  
        yığın.pop(); int h = yükseklik[p]; int  
        w = bileşenler.isEmpty() ? i : i -  
        stack.peek() - 1; max = Math.max(h * w, max);  
  
    }  
  
    maksimum dönüş;  
}
```

50 En Uzun Ortak Önek

50.1 Sorun

Bir dizge dizisi arasında en uzun ortak önek dizisini bulan bir işlev yazın.

50.2 Analiz

Bu sorunu çözmek için iki döngü koşulunu bulmamız gerekiyor. Bir, en kısa dizenin uzunluğudur. Diğer, dize dizisinin her ögesi üzerinde yinelemedir.

50.3 Java Çözümü

```
public String longCommonPrefix(String[] strs) { if(strs == boş || strs.uzunluk  
== 0)  
    dönüş "";  
  
int minLen=Tamsayı.MAX_VALUE; for(String str:  
strs){ if(minLen > str.length() minLen  
= dizi.uzunluk();  
  
} if(minLen == 0) return "";  
  
for(int j=0; j<minLen; j++){ char prev='0';  
    for(int i=0; i<strs.length ;i++){ if(i==0)  
        { önceki = strs[i].charAt(j); devam etmek;  
  
    }  
  
    if(strs[i].charAt(j) != önceki)  
        { strs[i] .altdize (0, j);  
    }  
    }  
}  
  
dönüş strs[0].altdize(0,minLen);  
}
```

51 En Büyük Sayı

Negatif olmayan tam sayıların bir listesi verildiğinde, onları en büyük sayıyı oluşturacak şekilde düzenleyin. Örneğin [3, 30, 34, 5, 9] verildiğinde oluşan en büyük sayı 9534330'dur . (Not: Sonuç çok büyük olabilir, bu nedenle tamsayı yerine bir dizge döndürmeniz gereklidir.)

51.1 Analiz

Bu problem, tamsayıları sıralayarak değil, sadece dizgeleri sıralayarak çözülebilir. Dizeleri concat() sağdan sola veya soldan sağa göre karşılaştırmak için bir karşılaştırıcı tanımlayın.

51.2 Java Çözümü

```
public String en büyükSayı(int[] sayılar)
{ Dize[] strs = yeni Dize[nums.uzunluk]; for(int
i=0; i<nums.length; i++){ strs[i] =
    String.valueOf(nums[i]);
}

Diziler.sort(strs, yeni Karşılaştırıcı<Dize>()){
    public int karşılaştırma(Dize s1, Dizi s2){
        Dizi leftRight = s1+s2; string
        rightLeft = s2+s1; dönüş
        -leftRight.compareTo(rightLeft);

    } });

StringBuilder sb = yeni StringBuilder(); for(String s:
strs){ sb.append(s);

}

while(sb.charAt(0)=='0' && sb.length()>1)
    { sb.deleteCharAt(0);
}

    sb.toString ();
}
```

52 Yolu Basitleştir

Bir dosya için mutlak bir yol verildiğinde (Unix stil), onu basitleştirin.

Örneğin,

```
yol = "/home/", => "/home" yol = "/  
a./b/../../c/", => "/c" yol = "../", => "/" yol =  
"/home//foo/", => "/home/foo"
```

52.1 Java Çözümü

```
public String basitleştirmePath(Dize yolu) {  
    Yığın<Dize> yığın = yeni Yığın<Dize>();  
  
    //stack.push(yol.altdize(0,1));  
  
    while(yol.uzunluk() > 0 && yol.charAt(yol.uzunluk()-1) == '/')  
        { yol = yol.altdize(0, yol.uzunluk()-1);  
        }  
  
    int başlangıç =  
    0; for(int i=1; i<path.length(); i++){ if(path.charAt(i)  
        == '/') stack.push(path.substring(start, i));  
        başlangıç = ben; }else if(i==path.length()-1)  
        { stack.push(path.substring(start));  
  
        }  
    }  
  
    LinkedList<String> sonuç = new LinkedList<String>(); int geri = 0; while(!  
    stack.isEmpty()) {  
  
        String top = stack.pop();  
  
        if(top.equals(".")) || top.equals("/")){ //nothing }else  
            if(top.equals("../")){ back++; }başka{ if(geri > 0){
```

52 Yolu Basitleştir

```
        geri--; }

        başka{ sonuç.push(üst); }

    }

//eğer boşsa, "/" döndür
if(result.isEmpty()){ return "/";

}

StringBuilder sb = yeni StringBuilder(); while(!
sonuç.isEmpty()){ Dize s = sonuç.pop();
sb.append(ler);

}

sb.toString ());
}
```

53 Sürüm Numaralarını Karşılaştırın

53.1 Sorun

Sürüm1 ve sürüm2 olmak üzere iki sürüm numarasını karşılaştırın .
Sürüm1 >sürüm2
1 döndürse , sürüm1 <sürüm2 -1 döndürse , aksi takdirde 0 döndürür. Sürüm
dizelerinin boş olmadığını ve yalnızca rakam içerdigini ve . karakter.. karakter bir
ondalık noktayı temsil etmez ve sayı dizilerini ayırmak için kullanılır. Sürüm numaraları
sıralamasına bir örnek:

0,1 < 1,1 < 1,2 < 13,37

53.2 Java Çözümü

Problemin zor kısmı 1.0 ve 1 gibi durumları ele almaktır. Eşit olmaları gereklidir.

```
public int CompareVersion(String version1, String version2) { String[] arr1 = version1.split("\\.");  
String[] dizi2 = sürüm2.split("\\.");  
  
int ben=0;  
while(i<arr1.length || i<dizi2.length){ if(i<arr1.length &&  
i<dizi2.length){ if(Tamsayı.parseInt(dizi1[i])  
< Tamsayı.parseInt(dizi2[i])){ -1 dönüş ;  
  
}else if(Integer.parseInt(arr1[i]) > Integer.parseInt(arr2[i])){  
    dönüş 1;  
}  
}  
} else if(i<arr1.length)  
{ if(Integer.parseInt(dizi1[i]) != 0)  
{ return 1;  
  
} } else if(i<dizi2.length){  
if(Integer.parseInt(arr2[i]) != 0){ dönüş -1;  
  
}  
}  
ben++;  
}  
  
0 dönüşü ;
```

53 Sürüm Numaralarını Karşılaştırın

}

54 Benzin İstasyonu

Dairesel bir güzergâh üzerinde i istasyonundaki benzin miktarının $gaz[i]$ olduğu N benzin istasyonu vardır.

Sınırsız benzin deposu olan bir arabanız var ve i istasyonundan bir sonraki istasyona ($i+1$) gitmek için benzin maliyeti $[i]$ var. Yolculuğa benzin istasyonlarından birinde boş bir depo ile başlıyorsunuz.

Devreyi bir kez dolaşabilirseniz, başlangıç benzin istasyonunun indeksini döndürün, aksi halde -1 döndürür .

54.1 Analiz

Bu sorunu çözmek için aşağıdaki 2 olguya anlamamız ve kullanmamız gereklidir : 1) Gazların toplamı \geq maliyetlerin toplamı ise, daire tamamlanabilir. 2) $A \rightarrow B \rightarrow C$ sıralamasında A, C'ye ulaşamıyorsa, B de ulaşamaz.

Gerçeğin kanıtı 2:

Gaz[A] < maliyet[A] ise, A bile B'ye ulaşamaz.

Yani A'dan C'ye ulaşmak için gaz[A] \geq maliyet[A] olmalıdır.

A'nın C'ye ulaşamayacağı göz önüne alındığında, gaz[A] + gaz[B] < maliyet[A] + maliyet[B] ve gaz[A] \geq maliyet[A] var, bu nedenle, gaz[B] < maliyet [B], yani B, C'ye ulaşamaz.

index	0	1	2	3	4
gas	1	2	3	4	5
cost	1	3	2	4	5

54.2 Java Çözümü

```
public int canCompleteCircuit(int[] gaz, int[] maliyet) {
    int toplamKalan = 0; // mevcut kalan int toplamını izle = 0; // iz
    toplam kalan int start = 0;
```

```
    için (int i = 0; i < gaz uzunluk; i++) {
```

54 Benzin İstasyonu

```
int kalan = gaz[i] - maliyet[i];

//eğer (i-1)'in kalan toplamı >= 0 ise, (kalan toplam >= 0) ise
devam et { toplamKalan += kalan; //aksi

takdirde, başlangıç dizinini geçerli olacak şekilde sıfırlayın }
else { toplamKalan = kalan; başlangıç = ben;

} toplam += kalan;
}

if (toplam >= 0){ dönüş
başlangıcı; }
else{ dönüş -1;
}
```

55 Pascal Üçgeni

Verilen numRows'lar, Pascal üçgeninin ilk numRows'larını oluşturur. Örneğin, numRows = 5 verildiğinde, sonuç şöyle olmalıdır:

```
[  
    [1],  
    [1,1],  
    [1,2,1],  
    [1,3,3,1],  
    [1,4,6,4,1]]
```

55.1 Java Çözümü

```
public ArrayList<ArrayList<Tamsayı>> oluştur(int numRows) {  
    ArrayList<ArrayList<Tamsayı>> sonuç = new ArrayList<ArrayList<Tamsayı>>(); eğer (numRows <= 0) sonucu  
    döndürür;  
  
    ArrayList<Tamsayı> pre = new ArrayList<Tamsayı>(); pre.add(1);  
    sonuç.add(ön);  
  
    for (int i = 2; i <= numRows; i++) {  
        ArrayList<Tamsayı> cur = new ArrayList<Tamsayı>();  
  
        cur.add(1); //ilk için (int j = 0;  
        j < pre.size() - 1; j++) {  
            cur.add(pre.get(j) + pre.get(j + 1)); // orta  
  
        } cur.add(1); //son  
  
        sonuç.add(cur);  
        ön = kür;  
    }  
  
    dönüş sonucu;  
}
```

56 Pascal Üçgeni II

Bir k indeksi verildiğinde, Pascal üçgeninin k'inci satırını döndürün. Örneğin k = 3 olduğunda satır [1,3,3,1] olur.

56.1 Analiz

Bu problem, [Bir sayı dizisinin k'inci satırını bulmak](#) Pascal Üçgeni ile ilişkilidir.

$$\begin{array}{ccccccc} & & & 1 & & & \\ & & 1 & & 1 & & \\ & 1 & & 2 & & 1 & \\ 1 & & 3 & & 3 & & 1 \\ 1 & 4 & 6 & 4 & 1 & & \\ 1 & 5 & 10 & 10 & 5 & 1 & \end{array}$$

56.2 Java Çözümü

```
genel Liste<Tamsayı> getRow(int rowIndex) {
    ArrayList<Tamsayı> sonuç = new ArrayList<Tamsayı>();

    eğer (rowIndex < 0)
        sonuç döndürür;

    sonuç.ekle(1); for
    (int i = 1; i <= rowIndex; i++) { for (int j = sonuç.size() -
        2; j >= 0; j--) { sonuç.set(j + 1, sonuç.get(j) +
            sonuç.get(j + 1));

    } sonuç.add(1);

    } dönüş sonuç;
}
```

57 En Fazla Su İçeren Kap

57.1 Sorun

Verilen n negatif olmayan tamsayı a_1, a_2, \dots, a_n , burada her biri (i, a_i) koordinatındaki bir noktayı temsil eder. i çizgisinin iki uç noktası (i, a_i) ve $(i, 0)$ olacak şekilde n dikey çizgi çizilir . ~~x ve kaptılar fazla esbib kapanma hizmeti doğru bulun.~~

57.2 Analiz

Başlangıçta sonucun 0 olduğunu varsayıyoruz. Sonra her iki taraftan da tararız. $\text{leftHeight} < \text{rightHeight}$ ise, sağa hareket etin ve leftHeight 'tan büyük bir değer bulun. Benzer şekilde, $\text{leftHeight} > \text{rightHeight}$ ise, sola hareket ettirin ve rightHeight değerinden büyük bir değer bulun.

Ek olarak, maksimum değeri takip etmeye devam etin.



57.3 Java Çözümü

```
public int maxArea(int[] yükseklik) { if
    (yükseklik == boş || yükseklik.uzunluk < 2) { dönüş 0;
    }

    int maks = 0;
    int kaldı = 0; int
    sağ = yükseklik.uzunluk - 1;

    while (sol < sağ) { max =
        Math.max(max, (sağ - sol) * Math.min(yükseklik[sol], yükseklik[sağ]));
    }
}
```

57 En Fazla Su İçeren Kap

```
if (yükseklik[sol] < yükseklik[sağ]) left++;
başka

Sağ--;
}

maksimum dönüş;
}
```

58 şeker

Bir sırada duran N çocuk var. Her çocuğa bir derecelendirme değeri atanır. Aşağıdaki gereksinimlere tabi olan bu çocuklara şeker veriyorsunuz:

1. Her çocuğun en az bir şekeri olmalıdır.
2. Puanı daha yüksek olan çocukların komşularından daha fazla şeker.

Vermeniz gereken minimum şeker miktarı nedir?

58.1 Analiz

Bu problem $O(n)$ zamanda çözülebilir.

Komşunun derecelendirme değeri daha yüksekse, komşuya her zaman 1 tane daha atayabiliriz . Ancak, minimum toplam sayıyı elde etmek için her zaman artan sırada 1'ler eklemeye başlamalıyız . Diziyi her iki taraftan da tarayarak bu sorunu çözebiliriz. Önce diziyi soldan sağa tarayın ve artan tüm çiftler için değerler atayın. Ardından sağdan sola tarayın ve azalan çiftlere değerler atayın.

Bu sorun , Yağmur Suyunu Yakalamak sorununa benzer .

58.2 Java Çözümü

```
public int candy(int[] dereceler) { if
(derecelendirmeler == boş || derecelendirmeler.uzunluk
== 0) { dönüş 0;
}

int[] şekerler = yeni int[derecelendirme.uzunluk];
şekerler[0] = 1;

//itten sağa for (int i =
1; i < derecelendirmeler.uzunluk; i++) { if
(derecelendirmeler[i] > derecelendirmeler[i - 1])
{ şekerler[i] = şekerler[i - 1] + 1; } else { // artan
değilse 1 şeker ata[i] = 1;

}

int sonuç = şekerler[derecelendirme.uzunluk - 1];
```

58 şeker

```
//sağdan sola for (int i =  
    derecelendirmeler.uzunluk - 2; i >= 0; i--) { int cur = 1; if  
        (derecelendirme[i] > derecelendirme[i + 1]) { cur = şekerler[i + 1] + 1;  
  
    }  
  
    sonuç += Math.max(cur, candies[i]); şekerler[i] =  
        koymak;  
}  
  
dönüş sonucu;  
}
```

59 Yağmur Suyunun Tutulması

Her bir çubuğun genişliğinin 1 olduğu bir yükseklik haritasını temsil eden negatif olmayan n tamsayı verildiğinde , yağmurdan sonra ne kadar su tutabileceğini hesaplayın. Örneğin

[0,1,0,2,1,0,1,3,2,1,2,1] verildiğinde 6 döndürür

59.1 Analiz

Bu problem Candy'ye benzer . Her iki taraftan tarayarak çözülebilir ve ardından toplamı alınabilir.

59.2 Java Çözümü

```
genel int trap(int[] yükseklik)
{ int sonuç = 0;

    if(height==null || height.length<=2)
        dönüş sonucu;

    int sol[] = yeni int[yükseklik.uzunluk]; int sağ[]=
    yeni int[yükseklik.uzunluk];

    //soldan sağa tara int max =
    yükseklik[0]; sol[0] = yükseklik[0];
    for(int i=1; i<height.length; i++){

        if(yükseklik[i]<maks)
            { sol[i]=maks; }

        başka{ sol[i]=yükseklik[i];
            maks = yükseklik[i];

        }

    }

    //sağdan sola tara max =
    yükseklik[yükseklik.uzunluk-1];
    sağ[yükseklik.uzunluk-1]=yükseklik[yükseklik.uzunluk-1]; for(int
    i=height.length-2; i>=0; i--){ if(yükseklik[i]<maks) { sağ[i]=maks; }

    başka{
```

59 Yağmur Suyunun Tutulması

```
sağ[i]=yükseklik[i]; maks =
yükseklik[i];
}
}

//toplamı hesapla for(int
ben=0; i<yükseklik.uzunluk; i++){ sonuç+=
Math.min(sol[i],sağ[i])-yükseklik[i];
}

dönüş sonucu;
}
```

60 Say ve Söyle

60.1 Sorun

Say ve söyle dizisi, aşağıdaki şekilde başlayan tamsayı dizisidir: 1, 11, 21, 1211, 111221, ...

1, "bir 1" veya 11 olarak okunur
11, "iki 1" veya 21 olarak okunur
21, "bir 2, sonra bir 1" veya 1211 olarak okunur

Bir n tamsayısı verildiğinde, n'inci diziyi oluşturun.

60.2 Java Çözümü

Sorun basit bir yineleme kullanılarak çözülebilir. Aşağıdaki Java çözümüne bakın:

```
public String countAndSay(int n)
{ eğer (n <= 0) boş
    döndürse;
    Dizi sonuc = "1"; int ben = 1;

    while (i < n) { StringBuilder
        sb = new StringBuilder(); int sayısı = 1; for (int j = 1; j <
        sonuç.uzunluk(); j++) {

            if (sonuç.charAt(j) == sonuç.charAt(j - 1)) {
                sayı++; }
            başka
            { sb.append(sayı);
            sb.append(result.charAt(j - 1)); sayı = 1;
            }

        }
    }

    sonuç.append(sayı);
    sonuç.append(sonuç.charAt(sonuç.uzunluk() - 1)); sonuç = sb.toString(); ben++;
}

}
```

60 Say ve Söyle

```
dönüş sonucu;  
}
```

61 Aralık Arama

Sıralanmış bir tamsayı dizisi verildiğinde, belirli bir hedef değerin başlangıç ve bitiş konumunu bulun. Algoritmanızın çalışma zamanı karmaşıklığı $O(\log n)$ sırasında olmalıdır. Hedef dizide bulunmazsa, $[-1, -1]$ döndürün. Örneğin, $[5, 7, 7, 8, 8, 10]$ ve hedef değer 8 verildiğinde $[3, 4]$ döndürün .

61.1 Analiz

$O(\log n)$ gerekliliğine bağlı olarak, bu görünüşe göre bir ikili arama problemidir.

61.2 Java Çözümü

```
public int[] searchRange(int[] sayilar, int hedef) { if(sayilar == null ||  
sayilar.length == 0){ null döndürür;  
  
}  
  
int[] dizi= yeni int[2]; dizi[0]=-1;  
dizi[1]=-1;  
  
binarySearch(sayılar, 0, sayılar.length-1, hedef, dizi);  
  
dönüş dizisi;  
}  
  
genel geçersiz ikili Arama(int[] sayılar, int sol, int sağ, int hedef, int[]  
varış)  
{ if(sağ<sol) dönüş;  
  
if(nums[left]==nums[sağ] && nums[left]==hedef){ arr[0]=left; dizi[1]=sağ; geri  
dönmek;  
  
}  
  
int orta = sol+(sağ-sol)/2;
```

61 Aralık Arama

```
if(sayılar[orta]<hedef){  
    binarySearch(sayılar, orta+1, sağ, hedef, dizi); }else  
if(nums[mid]>hedef){ binarySearch(nums, left, mid-1, target, arr); }  
başka{ dizi[0]=orta; dizi[1]=orta;  
  
//kopyaları işle - left int t1 = mid;  
while(t1 >sol && sayılar[t1]==sayılar[t1-1]) {  
  
    t1--;  
    dizi[0]=t1;  
}  
  
//kopyaları işle - right int t2 = mid;  
while(t2 < sağ&& sayılar[t2]==  
sayılar[t2+1]){ t2++; dizi[1]=t2;  
  
} dönüş;  
}  
}
```

62 Temel Hesap Makinesi

Basit bir ifade dizesini değerlendirmek için temel bir hesap makinesi uygulayın.

Ifade dizesi açık (ve kapanış parantezleri), artı + veya eksı işaret - , negatif olmayan tamsayılar ve boşluklar içerebilir. Verilen ifadenin her zaman geçerli olduğunu varsayıbilsiniz.

Bazı örnekler: "1 + 1" = 2, "(1)" = 1, "(1-(4-5))" = 2

62.1 Analiz

Bu sorun bir yiğin kullanılarak çözülebilir. Elemanı yiğine itmeye devam ediyoruz, ")" karşılandığında ilk "("'ye kadar ifadeyi hesaplıyoruz.

62.2 Java Çözümü

```
public int hesapla(String s) { // boşlukları
    sil s = s.replaceAll(" ", "");
    Yiğin<Dize> yiğin = yeni Yiğin<Dize>(); char[] dizi =
    s.toCharArray();

    StringBuilder sb = yeni StringBuilder(); for (int i = 0;
    i < dizi.uzunluk; i++) { (dizi[i] == ')') devam ederse ;

    if (arr[i] >= '0' && arr[i] <= '9')
        { sb.append(dizi[i]);

    if (i == dizi.uzunluk - 1)
        stack.push(sb.toString());
        stack.push(sb.toString()); sb =
        yeni StringBuilder();
    }

    if (arr[i] != ')') {
        stack.push(new String(new char[] { dizi[i] }));
    }
}
```

```
    } başka {//  
        '}' ile karşılaşıldığında, açılır ve hesaplanır  
        ArrayList<String> t = new ArrayList<String>(); while (!  
        stack.isEmpty()) { String top = stack.pop(); if (top.equals("("))  
            { ara; } else { t.add(0, top);  
  
        }  
    }  
  
    int sıcaklık = 0; if  
    (t.size() == 1) { temp =  
        Integer.valueOf(t.get(0)); } else { for (int j = t.size() -  
        1; j > 0; j = j - 2) {  
  
        if (t.get(j - 1).equals("-")) { temp += 0 -  
            Integer.valueOf(t.get(j)); } else { temp +=  
            Integer.valueOf(t.get(j));  
  
        }  
  
        } temp += Integer.valueOf(t.get(0));  
  
        } stack.push(String.valueOf(temp));  
    }  
}  
}  
  
ArrayList<String> t = new ArrayList<String>(); while (!  
stack.isEmpty()) { String öğeler = stack.pop(); t.add(0, eleman);  
  
}  
  
int sıcaklık = 0;  
for (int i = t.size() - 1; i > 0; i = i - 2) {  
    if (t.get(i - 1).equals("-")) { temp += 0 -  
        Integer.valueOf(t.get(i)); } else { temp +=  
        Integer.valueOf(t.get(i));  
  
    }  
  
} temp += Integer.valueOf(t.get(0));  
  
dönüş sıcaklığı;  
}
```

63 Grup Anagramı

Bir dizi dizi verildiğinde, anagram olan tüm dizi gruplarını döndürür.

63.1 Analiz

Anagram, orijinal harflerin tamamını tam olarak bir kez kullanarak yeni bir sözcük veya tümcecik oluşturmak için bir sözcük veya deyimin harflerini yeniden düzenlemenin sonucu olan bir tür sözcük oyunudur; örneğin Torchwood, Doctor Who olarak yeniden düzenlenebilir.

İki dizi birbirine anagram ise, sıralanma sırası aynıdır.

5/1/2016 tarihinde güncellendi .

63.2 Java Çözümü

```
public List<List<String>> groupAnagrams(String[] strs) {  
    List<List<String>> sonuç = new ArrayList<List<String>>();  
  
    HashMap<String, ArrayList<String>> map = new HashMap<String,  
        ArrayList<String>>();  
    for(String str: strs){ char[] dizi  
        = str.toCharArray(); Diziler.sort(dizi); Dize ns  
        = yeni Dize(arr);  
  
        if(map.containsKey(ns))  
            { map.get(ns).add(str); }  
        else{ ArrayList<String> al = new  
            ArrayList<String>(); al.add(str); map.put(ns, al);  
  
            }  
    }  
  
    for(Map.Entry<String, ArrayList<String>> girişi: map.entrySet()){  
        Collections.sort(entry.getValue());  
    }  
  
    sonuç.addAll(map.values());  
  
    dönüş sonuçu;  
}
```

63 Grup Anagramı

63.3 Zaman Karmaşıklığı

Fıillerin ortalama uzunluğu m ve kelime dizisi uzunluğu n ise, zaman $O(n*m*log(m))$ olur.

64 En Kısa Palindrom

Bir S dizesi verildiğinde, önüne karakterler ekleyerek onu bir palindroma dönüştürmenize izin verilir. Bu dönüşümü gerçekleştirerek bulabileceğiniz en kısa palindromu bulun ve döndürün.

Örneğin, "aacecaaa" verildiğinde "aaacecaaa" döndürür; "abcd" verildiğinde "dcbabcd" döndürür.

64.1 Analiz

Bu sorunu [en uzun palindrom alt dizi](#) problemini çözmek için kullanılan yöntemlerden birini kullanarak çözebiliriz .

Spesifik olarak, merkezden başlayıp iki tarafı tarayabiliriz. Sol sınır okunursa en kısa palindrom belirlenir.

64.2 Java Çözümü

```
public String shortPalindrome(String s) { if(s == boş ||  
    s.uzunluk() <= 1) dönüş s;  
  
    Dize sonucu = null;  
  
    int len = s.uzunluk(); int orta  
    = uzunluk / 2;  
  
    İçin (int i = orta; i >= 1; i--)  
    { if (s.charAt(i) == s.charAt(i - 1)) { if ((sonuç  
        = scanFromCenter(s, i - 1, i)) != null) sonucu döndür; } else { if ((sonuç =  
            scanFromCenter(s, i - 1, i - 1)) != null)  
  
                dönüş sonucu;  
            }  
        }  
    dönüş sonucu;  
}  
  
private String scanFromCenter(String s, int l, int r) {  
    int ben = 1;
```

64 En Kısa Palindrom

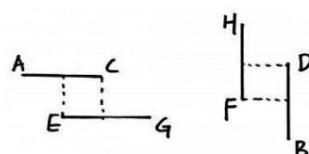
```
//merkezden her iki tarafa tara for (l - i >= 0; i++) { if (s.charAt(l - i) != s.charAt(r + i)) kirmak; } //eğer s'nin başında bitmiyorsa, null döndür, eğer (l - i >= 0) null döndür; StringBuilder sb = new StringBuilder(s.substring(r + i)); sb.ters(); sb.append(s.toString());' ben gidiyorum . }
```

65 Dikdörtgen Alan

2B düzlemde iki doğrusal dikdörtgenin kapsadığı toplam alanı bulun `sol`. Her dikdörtgen, alt köşesi ve sağ üst köşesi koordinatlarıyla tanımlanır.

65.1 Analiz

Bu sorun, örtüsen bir iç soruna dönüştürülebilir. x ekseninde (A,C) ve (E,G) vardır; y ekseninde (F,H) ve (B,D) vardır. Çakışmaları yoksa, toplam alan 2 dikdörtgen alanın toplamıdır. Çakışmaları varsa, toplam alandan örtüsen alan çıkarılmalıdır.



65.2 Java Çözümü

```
public int computeArea(int A, int B, int C, int D, int E, int F, int G, int
H) { if(C<E || G<A ) dönüş (GE)*(HF) + (CA)*(DB);
if(D<F || H<B)
    dönüş (GE)*(HF) + (CA)*(DB);
int sağ = Math.min(C,G); int kaldi
= Math.max(A,E); int top =
Math.min(H,D); int alt =
Math.max(F,B);
dönüş (GE)*(HF) + (CA)*(DB) - (sağ-sol)*(üst-alt);
}
```

66 Özet Aralıkları

Yinelemeler olmadan sıralanmış bir tamsayı dizisi verildiğinde, ardışık sayılar için aralıklarının özetini döndürün.

Örneğin, [0,1,2,4,5,7] verildiğinde, ["0->2","4->5","7"] döndürün .

66.1 Analiz

66.2 Java Çözümü

```
genel Liste<Dize> özetAralıklar(int[] sayılar) {
    List<String> sonuç = new ArrayList<String>();

    if(nums == null || nums.length==0) sonuç
        döndürür;

    if(nums.length==1)
        { sonuç.add(nums[0]+"");

    }

    int ön = sayılar[0]; // önceki eleman int first = pre; //
    her aralığın ilk elemani

    for(int i=1; i<nums.length; i++){ if(nums[i]==ön+1)
        { if(i==nums.uzunluk-1){ sonuç.add(ilk+"-"
            >" "+sayilar[i]);

        }

        else{ if(ilk == ön)
            { sonuç.add(ilk+"");
            else{ sonuç.add(ilk + "-"+pre);

            }

            if(i==nums.uzunluk-1)
                { sonuç.add(sayılar[i]+"");

            }

            birinci = sayılar[i];
        }

    }

birinci = sayılar[i];
}
```

66 Özet Aralıkları

```
    pre = sayilar[i];  
}  
  
dönüş sonucu;  
}
```

67 Artan Üçlü Dizi

Sıralanmamış bir dizi verildiğinde, dizide 3 uzunluğunda artan bir alt dizinin var olup olmadığı döndürülür.

Örnekler: [1, 2, 3, 4, 5] verildiğinde, true değerini döndürün.

[5, 4, 3, 2, 1] verildiğinde, false döndürün.

67.1 Analiz

Bu problem, the_smallest_so_far <the_second_smallest_so_far <current.js gibi bir dizi olup olmadığını bulmaya dönüştürülebilir. Sırasıyla 3 sayısını belirtmek için x, y ve z'yi kullanırız.

67.2 Java Çözümü

```
genel boolean artanTriplet(int[] sayılar)
{ int x = Tamsayı.MAX_VALUE; int y
= Tamsayı.MAX_VALUE;

for (int i = 0; i < sayı.uzunluk; i++) { int z = sayı[i];

if (x >= z) { x =
z;// x'i daha küçük bir değer olacak şekilde güncelleinyin } else
if (y >= z) { y = z; // y'i daha küçük bir değer olacak şekilde
güncelleinyin } else { true;

}

yanlış dönüş;
}
```

68 Numara Listesini ve Aritmetik İşlemleri Kullanarak Hedef Numarayı Alın

Bir sayı listesi ve bir hedef sayı verildiğinde, sayı listesine "+-*/" işlemleri uygulanarak hedef sayının hesaplanması hesaplanamayacağını bulan bir program yazınız? () gerekçinde otomatik olarak eklendiğini varsayıbilirsiniz.

Örneğin, 1,2,3,4 ve 21 verildiğinde, true değerini döndürür. Çünkü $(1+2)*(3+4)=21$

68.1 Analiz

Bu, önce derinlik araması kullanılarak çözülebilen bir bölümleme sorunudur.

68.2 Java Çözümü

```
genel statik boolean isReachable(ArrayList<Tamsayı> listesi, int hedef) {
    eğer (list == null || list.size() == 0) false döndürse;

    int ben = 0;
    int j = liste.size() - 1;

    ArrayList<Tamsayı> sonuçlar = getResults(list, i, j, hedef);

    for (int sayı : sonuçlar) { if (num ==
        hedef) { true döndür;

    }
}

yanlış dönüş;
}

genel statik ArrayList<Tamsayı> getResults(ArrayList<Tamsayı> listesi,
    int sol, int sağ, int hedef) {
    ArrayList<Tamsayı> sonuç = new ArrayList<Tamsayı>();

    if (sol > sağ) { sonucu
        döndür;
    } else if (sol == sağ)
        { sonuç.add(liste.get(sol));
    }
}
```

68 Numara Listesini ve Aritmetik İşlemleri Kullanarak Hedef Numarayı Alın

```
dönüş sonucu;
}

for (int i = sol; i < sağ; i++) {

    ArrayList<Tamsayı> sonuç1 = getResults(liste, sol, i, hedef);
    ArrayList<Tamsayı> sonuç2 = getResults(liste, i + 1, sağ, hedef);

    (int x : sonuç1) için
        { for (int y : sonuç2) { sonuç.topla(x
            + y); sonuç.topla(x - y);
            sonuç.topla(x * y); sonuç.topla(x /
            y);

        }
    }
}

dönüş sonucu;
}
```

Bir Dizinin 69 Ters Ünlüsü

Bir dizgeyi girdi olarak alan ve bir dizgenin sadece sesli harflerini ters çeviren bir fonksiyon yazın.

69.1 Java Çözümü

bu, dizinin başından ve sonundan tarama yapan iki işaretçi kullanılarak çözülebilten basit bir sorundur.

```
public String reverseVowels(String s) {  
    ArrayList<Karakter> yeminList = new ArrayList<Karakter>(); yeminList.add('a');  
    yeminList.add('e'); yeminList.add('i'); yeminList.add('o'); yeminList.add('u');  
    yeminList.add('A'); yeminList.add('E'); yeminList.add('I'); yeminList.add('O');  
    yeminList.add('U');  
  
    char[] dizi = s.toCharArray();  
  
    int ben=0;  
    int j=s.uzunluk()-1;  
  
    while(i<j){ if(!  
        yeminList.contains(arr[i])){  
            ben+  
            +; devam etmek;  
        }  
  
        if(!vowList.contains(arr[j]))  
            { J--; devam etmek;  
        }  
  
        karakter t = dizi[i];  
        dizi[i]=dizi[j]; dizi[j]=t;  
  
        ben+  
        +; J--;
```

Bir Dizinin 69 Ters Ünlüsü

```
    }  
  
    yeni String(arr) döndür;  
}
```

70 Çevirme Oyunu

Arkadaşınızla şu Çevirme Oyununu oynuyorsunuz: Bir dizi verildiğinde - -, siz ve karakterden iki tanesini çevirirsiniz: + ve ardışık arkadaşınız "+", "- "ye. dönüşümlü Oyun, olarak bir kişi sadece rtik hamle bu iki yapamadığı zaman sona erer ve bu nedenle diğer kişi kazanan olur.

Geçerli bir hareketten sonra dizenin tüm olası durumlarını hesaplayan bir işlev yazın.

70.1 Java Çözümü

```
genel Liste<String> createPossibleNextMoves(String s) {
    List<String> sonuç = new ArrayList<String>();

    if(s==null)
        dönüş sonucu;

    char[] dizi = s.toCharArray(); for(int i=0;
    i<arr.length-1; i++){ if(dizi[i]==dizi[i+1] &&
    dizi[i]=='+'){ dizi[i]='-';

        dizi[i+1]='-';
        sonuç.add(yeni Dizge(arr)); dizi[i]='+';
        dizi[i+1]='+';

    }
}

dönüş sonucu;
}
```

71 Döndürme Oyunu II

Arkadaşınızla şu Çevirme Oyununu oynuyorsunuz: Yalnızca şu iki karakteri içeren bir dizi verildiğinde: siz ve arkadaşınız kişi + veya - iki ardışık "+"yi yapamadığı "- "ye çevirirsiniz. zaman Oyun, sonaerer bir kişi ve artık bu nedenle hamle diğer kazanan olur.

Başlangıç oyuncusunun kazanmayı garanti edip edemeyeğini belirleyen bir fonksiyon yazın. Örneğin, s = "+++" verildiğinde, true değerini döndürür. Başlangıç oyuncusu ortadaki "+"yi "+-+" olacak şekilde çevirerek bir galibiyeti garanti edebilir.

71.1 Java Çözümü

Bu sorun geriye dönük olarak çözülür.

```
genel boolean canWin(String s) { if(s==null || s.length()==0){ false döndürür;
}

dönüş canWinHelper(s.toCharArray());
}

genel boolean canWinHelper(char[] arr){ for(int i=0; i<arr.length-1;i++)
{ if(arr[i]=='+&&arr[i+1]=='+')
{ dizi[i]='-'; dizi[i+1]='-';

boolean kazancı = canWinHelper(arr);

dizi[i]='+';
dizi[i+1]='+';

//eğer diğer oyuncunun kaybetmesine neden olan bir atış varsa, ilk oyna ,
eğer!
win){ true
    döndürürse kazanır;
}
}
}

yanlış dönüş;
```

71.2 Zaman Karmaşıklığı

Kabaca, zaman $n^*n^*...n$ 'dir, bu da $O(nn)$ 'dir. Bunun nedeni, her özyinelemenin $O(n)$ alması ve toplam n özyineleme olmasıdır.

72 Sıfır Taşı

Bir dizi numarası verildiğinde, sıfır olmayan öğelerin göreli sırasını korurken tüm 0'ları dizinin sonuna taşıyan bir işlev yazın .

Örneğin, verilen nums = [0, 1, 0, 3, 12], işlevinizi çağrırdıktan sonra, nums [1, 3, 12, 0, 0] olsun .

72.1 Java Çözümü

```
public void moveZeroes(int[] sayi) { int m=-1;

    for(int i=0; i<nums.length; i++)
        { if(sayılar[i]==0) { if(m==-1
            || sayıler[m]!=0){ m=i;

        } }
        else{ if(m!=-1)
            { int temp = nums[i];
                sayıler[i]=sayılar[m];
                nums[m]=temp; m++;

            }
        }
    }
}
```

73 Geçerli Anagram

İki dizi s ve t verildiğinde, t'nin s'nin bir anagramı olup olmadığını belirleyen bir işlev yazın.

73.1 Java Çözümü 1

Dize yalnızca küçük harf içeriyorsa, işte basit bir çözüm.

```
public boolean isAnagram(String s, String t) { if(s.uzunluk()!=t.uzunluk())
    false döndürür;

    int[] dizi = yeni int[26]; for(int i=0;
    i<s.uzunluk(); i++){ char c1 = s.charAt(i); dizi[c1-'a']+
        +;

    }

    for(int i=0; i<s.uzunluk(); i++){ char c2 = t.charAt(i);
        if(arr[c2-'a'] == 0){ false döndürür; }
        başka{ dizi[c2-'a']--;
            }

        }

    for(int i=0; i<26; i++)
        { if(arr[i]%2==1){ false
            döndürür;
            }
        }

    doğru dönüş;
}
```

73.2 Java Çözümü 2

Girişler unicode karakterler içeriyorsa, 26 uzunluğunda bir dizi yeterli değildir.

```
public boolean isAnagram(String s, String t) { if(s.uzunluk()!=t.uzunluk())
```

73 Geçerli Anagram

yanlış dönüş;

```
HashMap<Karakter, Tamsayı> map = new HashMap<Karakter, Tamsayı>();  
  
for(int i=0; i<s.length(); i++){ char c1 = s.charAt(i);  
if(map.containsKey(c1)){  
  
    map.put(c1, map.get(c1)+1); }  
başka{ map.put(c1,1);  
  
}  
}  
  
for(int i=0; i<s.uzunluk(); i++){ char c2 = t.charAt(i);  
if(map.containsKey(c2)){ if(map.get(c2)==1)  
{ harita.kaldır(c2); }başka{ map.put(c2,  
map.get(c2)-1);  
  
} }  
else{ false döndürür;  
}  
}  
  
if(map.size()>0) yanlış  
döndürür;  
}

---


```

74 Grup Kaydırılmış Dizeler

Bir dizi verildiğinde, onun harfinin her birini ardışık harfine "kaydırabiliriz", örneğin:
"abc" ->"bcd". "abc" ->"bcd" ->... ->"xyz" dizisini oluşturan "kaydırılmaya" devam edebiliriz.

Yalnızca küçük harf içeren dizilerin bir listesi verildiğinde, aynı kaydırma dizisine ait olan tüm dizileri gruplandırın, şunu döndürün:

```
[  
    ["abc", "bcd", "xyz"],  
    ["az", "ba"], ["acef"], [" a", "z"]]
```

```
]
```

74.1 Java Çözümü

```
public List<List<String>> groupStrings(String[] strings) {  
    List<List<String>> sonuç = new ArrayList<List<String>>();  
    HashMap<String, ArrayList<String>> haritası  
        = new HashMap<String, ArrayList<String>>();  
  
    for(String s: strings){ char[] arr  
        = s.toCharArray(); if(dizi.uzunluk>0)  
        { int fark = dizi[0]-'a'; for(int i=0;  
            i<dizi.uzunluk; i++){ if(arr[i]-diff<'a'  
                { dizi[i] = (char) (dizi[i]-diff+26); }else{ arr[i]  
                    = (karakter) (dizi[i]-fark);  
  
                }  
            }  
        }  
  
    Dize ns = yeni Dize(arr);  
    if(map.containsKey(ns))  
        { map.get(ns).add(s); }  
    else{ ArrayList<String> al = yeni  
        ArrayList<Dize>(); al.add(s); map.put(ns, al);
```

74 Grup Kaydırılmış Dizeler

```
    }
}

for(Map.Entry<String, ArrayList<String>> giriş: map.entrySet()){
    Collections.sort(entry.getValue());
}

sonuç.addAll(map.values());

dönüş sonuç;
}
```

75 En Sık Kullanılan K Öğe

Boş olmayan bir tamsayı dizisi verildiğinde, en sık kullanılan k elemanı döndürür.

75.1 Java Çözümü

Bu sorunu bir sayaç kullanarak çözebilir ve ardından **sayıçı değere göre sıralayabiliriz**.

genel sınıf Çözüm { genel

```
Liste<Tamsayı> topKFrequent(int[] nums, int k) {
    List<Tamsayı> sonuç = new ArrayList<Tamsayı>();

    HashMap<Tamsayı, Tamsayı> sayıçı = new HashMap<Tamsayı, Tamsayı>();

    for(int i: sayılar)
        { if(counter.containsKey(i))
            { counter.put(i, counter.get(i)+1);
            başka{ counter.put(i, 1);

        }
    }

    TreeMap<Tamsayı, Tamsayı> sortedMap = new TreeMap<Tamsayı, Tamsayı>(yeni
        ValueComparator(sayıçı));
    sortedMap.putAll(sayıçı);

    int ben=0;
    for(Map.Entry<Tamsayı, Tamsayı> giriş: sortedMap.entrySet()){ sonuç.add(entry.getKey());
        ben++; if(i==k) mola;

    }

    dönüş sonucu;
}
}
```

class ValueComparator, Comparator<Integer>{ öğesini **uggular**.

```
HashMap<Tamsayı, Tamsayı> map = new HashMap<Tamsayı, Tamsayı>();

public ValueComparator(HashMap<Tamsayı, Tamsayı> m){ map.putAll(m);

}
```

75 En Sık Kullanılan K Öğe

```
public int Compare(Tamsayı i1, Tamsayı i2){ int diff =
    map.get(i2)-map.get(i1);

    eğer(fark==0){ 1
        döndürür ; }
    else{ dönüş farkı;

    }
}
}
```

76 Doruk Elemanı Bul

Tepe elemanı, komşularından daha büyük olan bir elementtir. num[i] = num[i+1] olan bir girdi dizisi verildiğinde, tepe elemanı num[i] ve tepe noktası döndürmek iyidir. num[-1] = num[n] = - olduğunu düşünübilirsiniz. Örneğin, [1, 2, 3, 1] dizisinde 3 bir tepe elemanı ve işleviniz 2 numaralı indeksi döndürmelidir.

76.1 Düşünceler

Bu çok basit bir problem. Diziyi tarayabilir ve önceki ve sonrakinden daha büyük olan herhangi bir öğeyi bulabiliyoruz. İlk ve son eleman ayrı ayrı ele alınır.

76.2 Java Çözümü

```
genel sınıf Çözüm { genel int
    findPeakElement(int[] sayı) { int max = sayı[0]; int dizini =
        0; for(int i=1; i<=num.length-2; i++){ int önceki =
            num[i-1]; int akım = sayı[i]; int sonraki = sayı[i+1];

            if(curr > önceki && curr > sonraki && curr > max){
                indeks = ben;
                maks = akım;
            }
        }

        if(sayı[num.uzunluk-1] > maks)
            { sayı.uzunluk-1 döndürür ;
        }

        dönüş indeksi;
    }
}
```

77 Kelime Kalıbü

Bir örüntü ve str dizesi verildiğinde, str'nin aynı örüntüyü izleyip izlemediğini bulun. Burada takip, tam eşleşme anlamına gelir, öyle ki kalıptaki bir harf ile str'deki boş olmayan bir kelime arasında bir eşleştirme vardır.

77.1 Java Çözümü

```
public boolean wordPattern(String pattern, String str) {  
    String[] dizi = str.split(" ");  
  
    if(pattern.length()!=dizi.length) return false;  
  
    HashMap<Karakter, Dize> map = new HashMap<Karakter, Dize>();  
  
    for(int i=0; i<dizi.length; i++){ char c =  
        pattern.charAt(i); String s = dizi[i];  
  
        if(map.containsKey(c)){  
            if(!map.get(c).equals(s)) return false;  
  
        }  
        else{ if(map.containsValue(s)) return false;  
            map.put(c, s);  
        }  
    }  
  
    return true;  
}
```

78 Matris Sıfırlarını Ayarla

Yerim * n matrisinde, bir eleman 0 ise , tüm satırını ve sütununu 0 olarak ayarlayın.
verildi.

78.1 Analiz

Bu problem yerinde çözülmeli yani başka bir dizi kullanılmamalıdır. Bir satırın/sütun 0 olarak ayarlanması gerekip gerekmediğini izlemek için ilk sütunu ve ilk satırı kullanabiliriz. Sıfır satırını/sütununu işaretlemek için ilk satırı ve ilk sütunu kullandığımız için, orijinal değerler değiştirilir.

1	1	1	0
1	1	1	0
1	1	0	0
1	0	0	0

Adım 1: İlk satır sıfır içerir = true; İlk sütun sıfır içerir = yanlış;

1	0	0	0
0	1	1	0
0	1	0	0
0	0	0	0

78 Matris Sıfırlarını Ayarla

1	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

78.2 Java Çözümü

```
genel sınıf Çözüm { genel geçersiz
    setZeroes(int[][] matris) { boolean firstRowZero = false;
        boolean firstColumnZero = false;

        //ilk satırı ve sütunu sıfır'a ayarla veya yapma for(int
        i=0; i<matrix.length; i++){ if(matrix[i][0] == 0){

            firstColumnZero = true; kırmak;

        }
    }

    for(int i=0; i<matris[0].uzunluk; i++){ if(matris[0][i] == 0)
        {
            firstRowZero = doğru; kırmak;

        }
    }
}
```

```
//ilk satır ve sütunda sıfırları işaretle for(int i=1;
i<matrix.length; i++){ for(int j=1; j<matrix[0].uzunluk;
j++){ if(matrix[i][j]
== 0){ matrix[i][0] = 0; matrix[0][j] = 0;

}

}

}

//elemanları ayarlamak için işaret
kullanın for(int i=1; i<matrix.length; i++)
{ for(int j=1; j<matrix[0].uzunluk; j++)
{ if(matrix[i][0] == 0 || matrix[0][j] == 0){ matrix[i][j] = 0;

}

}

}

//ilk sütunu ve satırı ayarla
if(firstColumnZero){ for(int i=0;
i<matrix.uzunluk; i+
+) matrix[i][0] = 0;
}

if(firstRowZero){ for(int
i=0; i<matrix[0].length; i++)
matrix[0][i] = 0;
}

}

}
```

79 Spiral Matris

$m \times n$ elemanlı bir matris verildiğinde (m satır, n sütun), matrisin tüm elemanlarını spiral sırayla döndürür.

Örneğin, aşağıdaki matris verildiğinde:

```
[[1, 2, 3], [4,  
5, 6], [7, 8, 9]]
```

[1,2,3,6,9,8,7,4,5] döndürmelisiniz .

79.1 Java Çözümü 1

Birden fazla satır ve sütun kaldıysa daire oluşturabilir ve daireyi işliyoruz.

Aksi takdirde, yalnızca bir satır veya sütun kaldıysa, YALNIZCA o sütun veya satırı işleriz.

```
genel sınıf Çözüm { genel  
    ArrayList<Tamsayı> spiralOrder(int[][] matris) {  
        ArrayList<Tamsayı> sonuç = new ArrayList<Tamsayı>();  
  
        if(matrix == null || matrix.length == 0) sonuç döndürür;  
  
        int m = matris.uzunluk; int n =  
            matris[0].uzunluk;  
  
        int x=0;  
        int y=0;  
  
        while(m>0 && n>0){  
  
            //bir satır/sütun kaldıysa çember oluşturulamaz if(m==1){ for(int i=0; i<n;  
                i++){ result.add(matrix[x][y++]);  
  
            } mola; }  
            else if(n==1){ for(int  
                ben=0; ben i++){ sonuç.topla(matris[x+  
                    +][y]);  
            }  
        }  
    }  
}
```

79 Spiral Matris

```

        kirmak;
    }

    //aşağıda bir daire işleyin

    //top - sağa hareket ettir for(int
    i=0;i<n-1;i++)
    {
        sonuç.add(matriş[x][y++]);
    }

    //sağ - aşağı kaydır for(int
    i=0;i<m-1;i++){ result.add(matriş[x+
        +][ve]);
    }

    //bottom - sola kaydır for(int
    i=0;i<n-1;i++){ result.add(matriş[x]
        [Ve--]);
    }

    //sola - yukarı taşı
    for(int i=0;i<m-1;i++)
    {
        sonuç.topla(matriş[x--][y]);
    }

    x++;
    y++;
    m=m-2;
    n=n-2;
}

dönüş sonucu;
}
}

```

79.2 Java Çözümü 2

Bu sorunu özyinelemeli olarak da çözebiliriz. Çözümün performansı Çözümden daha iyi veya Çözüm 1 kadar net değil. Bu nedenle Çözüm 1 tercih edilmelidir.

```

genel sınıf Çözüm { genel
    ArrayList<Tamsayı> spiralOrder(int[] matris) { if(matrix==null
        || matrix.length==0) yeni DiziListesi<Tamsayı>()
        döndürür;

    dönüş spiralOrder(matris,0,0,matris.uzunluk,matris[0].uzunluk);
}
}

```

```
public ArrayList<Tamsayı> spiralOrder(int [][] matris, int x, int y, int
m, int n){
ArrayList<Tamsayı> sonuç = new ArrayList<Tamsayı>();

if(m<=0 | |n<=0)
    sonuç döndürür ;

//sadece bir eleman kaldı
if(m==1&&n==1)
    { sonuç.topla(matris[x][y]); dönüş
sonuç;
}

//top - sağa hareket ettir
for(int i=0;i<n-1;i++)
    { sonuç.add(matris[x][y++]);
}

//sağ - aşağı kaydır for(int
i=0;i<m-1;i++){ result.add(matrix[x+
+][[ve]);
}

//bottom - sola git if(m>1)
{ for(int i=0;i<n-1;i++)
    { sonuç.add(matris[x][y--]);

}

//sola - yukarı git if(n>1)
{ for(int i=0;i<m-1;i++)
    { sonuç.add(matrix[x--][y]);
}

if(m==1 | |n==1)
    sonuç.addAll(spiralOrder(matris, x, y, 1, 1)); başka
    sonuç.addAll(spiralOrder(matris, x+1, y+1, m-2, n-2));

    dönüş sonucu;
}
```

80 Spiral Anne II

Bir n tamsayısı verildiğinde, 1'den n^2 'ye kadar olan elemanlarla spiral sırayla doldurulmuş bir kare matris oluşturun. Örneğin, verilen n = 4,

```
[ [1, 2, 3, 4], [12, 13,  
14, 5], [11, 16, 15, 6],  
[10, 9, 8, 7] ]
```

80.1 Java Çözümü

```
public int[][] createMatrix(int n) { int toplam = n*n; int[][]  
sonuç= yeni int[n][n];  
  
int = 0; Sen  
y=0; int  
adım = 0;  
  
for(int i=0;i<total;){ while(y+adım< n)  
{ i++; sonuç[x][y]=i;  
  
ve++;  
}  
ve--;  
x++;  
  
while(x+adım< n){ i++;  
sonuç[x][y]=i;  
  
x++;  
}  
X-;  
ve--;  
  
while(y>=0+adım){ i++;
```

80 Spiral Anne II

```
    sonuç[x][y]=i;
    ve--;
}
ve++;
X--;
adım++;
while(x>=0+adım){ i++;
    sonuç[x][y]=i;

    X--;
}
x++;
ve++;
{
    dönüş sonucu;
}
```

81 2B Matris Arama

Bir $m \times n$ matrisinde bir değer arayan verimli bir algoritma yazın. Bu matrisin özellikleri vardır: 1) Her satırdaki tamsayılar soldan sağa doğru sıralanır. 2) Her satırın ilk tam sayısı önceki satırın son tamsayısından büyüktür.

Örneğin, aşağıdaki matrisi göz önünde bulundurun:

```
[  
 [1, 3, 5, 7], [10, 11,  
 16, 20], [23, 30, 34, 50]  
 ]
```

Verilen hedef = 3, true döndür.

81.1 Java Çözümü

Bu, ikili aramanın tipik bir sorunudur.

Önce satırı sonra sütunu bularak bu sorunu çözmeye çalışabilirsiniz.

Bunu yapmaya gerek yok. Matrisin kendine has özellikleri nedeniyle, matris sıralanmış bir dizi olarak düşünülebilir. Amacınız, ikili aramayı kullanarak bu sıralanmış dizideki bir öğeyi bulmaktır.

```
genel sınıf Çözüm { genel boolean  
 searchMatrix(int[][] matrisi, int hedefi) {  
     if(matrix==null || matrix.length==0 || matrix[0].length==0) false döndür;  
  
     int m = matris.uzunluk; int n =  
         matris[0].uzunluk;  
  
     int başlangıç = 0;  
     int bitiş = m*n-1;  
  
     while(başlangıç<=bitiş)  
     { int orta=(başlangıç+bitiş)/2;  
         int ortaX=orta/n; int  
         ortaY=orta%n;  
  
         if(matrix[midX][midY]==hedef) true  
             döndür;
```

81 2B Matris Arama

```
if(matrix[midX][midY]<hedef){ start=mid+1; }
    başka{ bitiş=orta-1;

}
yanlış dönüş;
}
}
```

82 2D Matrix II Arama

Bir $m \times n$ matrisinde bir değer arayan verimli bir algoritma yazın. Bu matris aşağıdaki özelliklere sahiptir: Her satırındaki tamsayılar soldan sağa doğru artan şekilde sıralanmıştır. Her birinde tamsayılar sütunları yukarıdan aşağıya doğru sıralanır.

Örneğin, aşağıdaki matrisi göz önünde bulundurun:

```
[  
    [1, 4, 7, 11, 15], [2, 5, 8,  
        12, 19], [3, 6, 9, 16, 22],  
        [10, 13, 14, 17, 24], [18,  
            21, 23, 26, 30]  
]
```

Verilen hedef = 5, true döndür.

82.1 Java Çözümü 1

Naif bir yaklaşımıla, arama alanını azaltmak için matris sınırını kullanabiliriz. İşte basit bir özyinelemeli uygulama.

```
jenerik boolean aramaMatrix(int[][] matris, int hedef)  
{ int i1=0; int  
    i2=matris.uzunluk-1; int j1=0;  
    int j2=matris[0].uzunluk-1;  
  
    dönüş yarımıcı(matris, i1, i2, j1, j2, hedef);  
}  
  
genel boole yarımıcı(int[][] matris, int i1, int i2, int j1, int j2, int  
hedef){  
  
    if(i1>i2 | j1>j2) false  
        döndür;  
  
    for(int j=j1;j<=j2;j++){ if(target <  
        matrix[i1][j]){ dönüş yarımıcı(matris, i1,  
            i2, j1, j-1, hedef); }else if(target ==  
            matrix[i1][j)){ true döndür;  
}
```

82 2D Matrix II Arama

```
        }
    }

    for(int i=i1;i<=i2;i++){
        if(target <
            matrix[i][j1]) {
                döndürür;
            }
        }

        for(int j=j1;j<=j2;j++){
            if(hedef > matris[i2][j]) {
                döndürür;
            }
            else if(target == matris[i2][j]) {
                döndürür;
            }
        }

        for(int i=i1;i<=i2;i++){
            if(hedef > matris[i][j2]) {
                döndürür;
            }
            else if(target == matris[i][j2]) {
                döndürür;
            }
        }

        yanlış döndürür;
    }
}
```

82.2 Java Çözümü 2

Zaman Karmaşıklığı: O(m + n)

```
genel boole aramaMatrix(int[][] matris, int hedef) { int m=matrix.uzunluk-1; int
n=matris[0].uzunluk-1;

        int ben=m;
        int = 0;

        while(i>=0 && j<=n){ if(hedef <
            matris[i][j]) {
                i--;
            }
            else if(hedef > matris[i][j]) {
                j++;
            }
            else{ döndürür;
        }
    }
}
```

```
    }  
}  
  
yanlış dönüş;  
}
```

83 Görüntüyü Döndür

Size bir görüntüyü temsil eden bir nxn 2B matris verilir .

Görüntüyü 90 derece döndürün (saat yönünde).

Takip: Bunu yerinde yapabilir misiniz?

83.1 Naif Çözüm

Aşağıdaki çözümde döndürülen matrisi depolamak için 2 boyutlu yeni bir dizi oluşturulur ve sonuç matrise ucta atanır. Bu yanlış! Neden?

```
genel sınıf Çözüm { genel geçersiz
    döndürme(int[][]) matris {
        if(matris == boş || matris.uzunluk==0)
            dönüş;

        int m = matris.uzunluk;

        int[][] sonuç = yeni int[m][m];

        for(int i=0; i<m; i++){
            for(int j=0; j<m; j++){ sonuç[j][m-1-i]
                = matris[i][j];
            }
        }

        matris = sonuç;
    }
}
```

Sorun şu ki, Java referansa göre değil değere göre geçiyor! "matris", 2 boyutlu bir diziye yalnızca bir referanstır . Yöntemde yeni bir 2 boyutlu diziye "matris" atanırsa , orijinal dizi değişmez. Bu nedenle, her ögeyi "matris" tarafından başvurulan diziye atamak için başka bir döngü olmalıdır. ["Değerine göre Java geçisi"](#) konusuna bakın .

```
genel sınıf Çözüm { genel geçersiz
    döndürme(int[][]) matris {
        if(matris == boş || matris.uzunluk==0)
            dönüş;

        int m = matris.uzunluk;
```

83 Görüntüyü Döndür

```
int[][] sonuç = yeni int[m][m];  
  
for(int i=0; i<m; i++)  
    { for(int j=0; j<m; j++){ sonuç[j]  
        [m-1-i] = matris[i][j];  
    }  
}  
  
for(int i=0; i<m; i++){  
    for(int j=0; j<m; j++){ matris[i][j]  
        = sonuç[i][j];  
    }  
}  
}
```

83.2 Yerinde Çözüm

"matris[i][j] = matris[n-1-j][i]" bağıntısını kullanarak , matris boyunca döngü yapabiliriz.

```
genel boşluk döndürme(int[][] matris) { int n =  
    matris.uzunluk; for (int i = 0; i < n / 2; i++) { for  
(int j = 0; j < Math.ceil(((double) n) / 2); j++) { int  
    temp = matrix[i][j]; matris[i][j] = matris[n-1-j][i]; matris[n-1-j][i] = matris[n-1-i][n-1-  
    j]; matris[n-1-i][n-1-j] = matris[j][n-1-i]; matris[j][n-1-i] = sıcaklık;  
    }  
}  
}
```

84 Geçerli Sudoku

Bir Sudoku'nun geçerli olup olmadığını belirleyin. Sudoku tahtası, boş hücrelerin '.' karakteriyle doldurulduğu kısmen doldurulabilir.

5	3			7				
6			1	9	5			
	9	8				6		
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8		7	9	

84.1 Java Çözümü

```
public boolean isValidSudoku(char[][] board) { if (board == null) ||
    board.length != 9 || board[0].length != 9) yanlış dönüş; // (int
    i = 0; i < 9; i++) için her sütunu kontrol et {

    boolean[] m = yeni boolean[9]; for (int j
    = 0; j < 9; j++) { if (board[i][j] != '.') {

        if (m[(int) (tahta[i][j] - '1')]) { false döndürür;

        } m[(int) (board[i][j] - '1')] = true;
    }
}
}

//her satırı kontrol et
(int j = 0; j < 9; j++) {
    boolean[] m = yeni boolean[9]; için (int
    ben = 0; ben < 9; i++) {
```

84 Geçerli Sudoku

```
if (tahta[i][j] != '.') {
    if (m[(int)(tahta[i][j] - '1')]) { false döndürür;

} m[(int)(board[i][j] - '1')] = true;
}

}

}

//her 3*3 matris için (int blok = 0;
blok < 9; blok++) { boolean[] m = yeni boolean[9]; için (int) ben =
blok / 3 * 3; ben < blok / 3 * 3 + 3; ben++) {

for (int j = blok % 3 * 3; j < blok % 3 * 3 + 3; j++) { if (board[i][j] != '.') {

if (m[(int)(tahta[i][j] - '1')]) { false döndürür;

} m[(int)(board[i][j] - '1')] = true;
}

}

}

}

}

doğru dönüş;
}
```

85 Minimum Yol Toplami

Negatif olmayan sayılarla dolu amxn ızgarası verildiğinde, sol üstten sağ alta doğru yolu boyunca tüm sayıların toplamını en aza indiren bir yol bulun.

85.1 Java Çözüm 1: Önce Derinlik Araması

Yerel bir çözüm, derinlemesine arama olacaktır. Zaman çok pahalı ve çevrimiçi yargıda başarısız oluyor.

```
public int minPathSum(int[][] grid) { dönüş dfs(0,0,grid);  
}  
  
public int dfs(int i, int j, int[][] grid)  
{ if(i==ızgara uzunluk-1 && j==kılavuz[0].uzunluk-1){  
    dönüş ızgarası[i][j];  
}  
  
if(i<ızgara uzunluk-1 && j<ızgara[0].uzunluk-1){ int  
    r1 = ızgara[i][j] + dfs(i+1, j, ızgara); int r2 = ızgara[i][j] + dfs(i,  
    j+1, ızgara); Math.min(r1,r2) döndürür ;  
}  
  
}  
  
if(i<ızgara uzunluk-1)  
{ dönüş ızgarası[i][j] + dfs(i+1, j, ızgara);  
}  
  
if(j<ızgara[0].uzunluk-1)  
{ dönüş ızgarası[i][j] + dfs(i, j+1, ızgara);  
}  
  
0 dönüşü ;  
}
```

85.2 Java Çözümü 2: Dinamik Programlama

```
public int minPathSum(int[][] grid) { if(grid == null ||  
grid.length==0)  
    0 dönüşü ;
```

85 Minimum Yol Toplami

```
int m = izgara.uzunluk; int
n = izgara[0].uzunluk;

int[][] dp = yeni int[m][n]; dp[0][0] =
izgara[0][0];

// en üst satırı for(int i=1;
i<n; i++){ dp[0][i] = dp[0][i-1] +
izgara[0][i];
}

// sol sütunu for(int j=1; j<m; j++)
{ dp[j][0] = dp[j-1][0] + izgara[j][0];

}

// dp tablosunu şunun için
doldurun :(int i=1; i<m; i++){
    for(int j=1; j<n; j++){
        if(dp[i-1][j]
        > dp[i][j-1]){
            dp[i][j] = dp[i][j-1] +
izgara[i][j];
        }başka{ dp[i][j] = dp[i-1][j] + izgara[i][j];
    }
}
}

dönüş dp[m-1][n-1];
}
```

86 Benzersiz Yol

amxn izgarasının sol üst köşesinde bir robot bulunur. Herhangi bir zamanda yalnızca aşağı veya sağa hareket edebilir. Robot, ızgaranın sağ alt köşesine ulaşmaya çalışıyor.

Kaç tane olası benzersiz yol vardır?

86.1 Java Çözümü 1 - DFS

Bir derinlik öncelikli arama çözümü oldukça basittir. Ancak, bu çözümün süresi çok pahalı ve çevrimiçi jüriden geçmedi.

```
public int uniquePaths(int m, int n)
    { dönüş dfs(0,0,m,n);
    }

genel int dfs(int i, int j, int m, int n)
    { if(i==m-1 && j==n-1){ 1
        yönlendirici ;
    }

    eğer(i<m-1 && j<n-1){
        dönüş dfs(i+1,j,m,n) + dfs(i,j+1,m,n);
    }

    eğer(i<m-1)
        { dönüş dfs(i+1,j,m,n);
    }

    if(j<n-1){ dönüş
        dfs(i,j+1,m,n);
    }

    0 dönüşü ;
}
```

86.2 Java Çözümü 2 - Dinamik Programlama

```
public int uniquePaths(int m, int n) { if(m==0 || n==0)
    dönüş 0; if(m==1 || n==1) 1 döndürür;
```

86 Benzersiz Yol

```
int[][] dp = yeni int[m][n];  
  
//sol sütun for(int  
i=0; i<m; i++){ dp[i][0] = 1;  
  
}  
  
//üst sıra  
for(int j=0; j<n; j++){ dp[0][j] = 1;  
  
}  
  
//dp tablosunu şunun için  
doldur :(int i=1; i<m; i++)  
{ for(int j=1; j<n; j++) { dp[i][j] =  
    dp[i-1][j] + dp[i][j-1];  
}  
}  
  
dönüş dp[m-1][n-1];  
}
```

87 Eşsiz Yol II

"Benzersiz Yollar" için takip : Şimdi

İzgaralara bazı engellerin eklenip eklenmediğini düşünün. Kaç benzersiz yol olur muydu?

Izgarada bir engel ve boş alan sırasıyla 1 ve 0 olarak işaretlenir . İçin Örneğin, aşağıda gösterildiği gibi 3x3'lük bir izgaranın ortasında bir engel vardır ,

[

[0,0,0],

[0,1,0],

[0,0,0]

]

benzersiz yolların toplam sayısı 2'dir.

87.1 Java Çözümü

```
public int uniquePathsWithObstacles(int[][] engelGrid) { if(engelliGrid==null || engelliGrid.uzunluk==0) dönüş 0;

int m = engelGrid.uzunluk; int n =
engelGrid[0].uzunluk;

if(engelGrid[0][0]==1 | engelGrid[m-1][n-1]==1) 0
dönüş ;

int[] dp = yeni int[m][n]; dp[0][0]=1;

//sol sütun for(int
ben=1; ben i++){ if(engelliGrid[i]
[0]==1){ dp[i][0] = 0; }başka{ dp[i][0] =
dp[i-1][0];

}

}

// üst sıra
```

87 Eşsiz Yol II

```
for(int i=1; i<n; i++)
{ if(engelliGrid[0][i]==1){ dp[0][i] =
0; }başka{ dp[0][i] = dp[0][i-1];

}

//içindeki hücreleri doldur
for(int i=1; i<m; i++){ for(int
j=1; j<n; j++)
{ if(engelliGrid[i][j]==1)
{ dp[i][j]=0; }
başka{ dp[i] [j]=dp[i-1][j]
+dp[i][j-1];
}

}

dönüş dp[m-1][n-1];
}
```

88 Ada Sayısı

'1'lerin (kara) ve '0'ların (su) 2 boyutlu bir izgara haritası verildiğinde , adaların sayısını sayın. Bir ada su ile çevrilidir ve bitişik karaların yatay veya dikey olarak bağlanmasıyla oluşur. Izgaranın dört kenarının da suyla çevrili olduğunu varsayıbilirsiniz.

Örnek 1:

```
11110
11010
11000
00000
```

Cevap:

1 Örnek 2:

```
11000
11000
00100
00011
```

Cevap: 3

88.1 Java Çözümü

Aşağıdaki çözümün temel fikri bitişik arazileri birleştirmektir ve birleştirme yinelemeli olarak yapılmalıdır.

```
public int numIslands(char[][] grid) {
    if(grid==null || grid.length==0 || grid[0].length==0) return 0;

    int m=kılavuz.uzunluk;
    int n=kılavuz[0].uzunluk;

    int sayısı=0;
    for(int i=0;i<m; i++){
        for(int j=0;j<n; j++){
            if(grid[i][j]=='1'){
                say+
                +; birleştir(izgara, i, j);
            }
        }
    }
}
```

88 Ada Sayısı

```
döñüş sayısı;
}

genel geçersiz birleştirme(char[][] grid, int i, int j){
    if(i<0 | |j<0 | |j>=ızgara.uzunluk | |j>=ızgara[0].uzunluk)
        geri dönmek;

    if(grid[i][j]=='1'){ grid[i][j]='0';

        birleştirme(ızgara, i-1,j);
        birleştirme(ızgara, i+1,j);
        birleştirme(ızgara, i,j-1);
        birleştirme(ızgara, i,j+1);
    }
}
```

Ada Sayısı II'ye göz atın .

89 Ada Sayısı II

m sıra ve n sütundan oluşan 2 boyutlu bir ızgara haritası başlangıçta suyla doldurulur. (satır, sütun) konumundaki suyu karaya çeviren bir addLand işlemi gerçekleştirilebiliriz. Çalıştırılacak konumların bir listesi verildiğinde, her bir addLand işleminden sonra adaların sayısını sayın. Bir ada su ile çevrilidir ve bitişik karaların yatay veya dikey olarak bağlanmasıyla oluşur. Izgaranın dört kenarının da çevrelenmiş olduğunu varsayıbilsiniz.

vardır.

89.1 Java Çözümü

Her hücre için üst düğümü izlemek üzere bir dizi kullanın.

```
genel Liste<Tamsayı> numIslands2(int m, int n, int[][] konumlar) { int[] rootArray = new
int[m*n]; Arrays.fill(rootArray,-1);

ArrayList<Tamsayı> sonuç = new ArrayList<Tamsayı>();

int[][] yönler = {{-1,0},{0,1},{1,0},{0,-1}}; int sayısı=0;

for(int k=0; k<konumlar.uzunluk; k++){
    sayı++;
    int[] p = konumlar[k]; int dizini =
    p[0]*n+p[1];
    rootArray[index]=index;// kökü her düğüm için kendisi olacak şekilde ayarla

    for(int r=0;r<4;r++){
        int
        i=p[0]+yönler[r][0]; int j=p[1]+yönler[r]
        [1];

        if(i>=0&&j>=0&&i<m&&j<n&&rootArray[i*n+j]!=-1){ //komşunun
            kökünü al int thisRoot = getRoot(rootArray, i*n+ j); if ( thisRoot != tasarırm ){
                rootArray[thisRoot]=index;// önceki kökün kök sayısını ayarla--;
            }
        }
    }
}
```

89 Ada Sayısı II

```
    sonuç.add(sayı);
}

dönüş sonucu;
}

public int getRoot(int[] dizi, int i){
    while(i!=arr[i]){
        i=arr[arr[i]];
    }
} dönüş i;
}
```

90 Çevrelenmiş Bölge

'X' ve 'O' içeren bir 2B pano verildiğinde, 'X' ile çevrili tüm bölgeleri yakalayın. Bir bölge, çevrelenmiş bölgelerdeki tüm 'O'lari' X'lere çevirerek yakalanır.

Örneğin,

XXXX
HAYVAN
XXOX
XXXX

İşlevinizi çalıştırıldıktan sonra pano şöyle olmalıdır:

XXXX
XXXX
XXXX
XOXX

90.1 Analizi

Bu sorun [Ada Sayısına](#) benzer . Bu problemde sadece yatılılar üzerindeki hücreler çevrelenemez. Bu yüzden önce [Ada Sayısı'ndaki](#) gibi bordürlerdeki O'lari birleştirip O'lari '#' ile değiştirebiliriz, sonra tahtayı tarayıp (varsayı) kalan tüm O'lari değiştirebiliriz.

90.2 Derinlik Öncelikli Arama

```
public void çöz(char[][] board)
{ if(tahta == boş || tahta.uzunluk==0) dönüş;

    int m = tahta.uzunluk; int n
    = pano[0].uzunluk;

    //O'lari sol ve sağ kenarda birleştir for(int
    i=0;i<m;i++){ if(tahta[i][0] == 'O')
        { birleştirme(tahta, i, 0);

    }

    if(tahta[i][n-1] == 'O'){


```

90 Çevrelenmiş Bölge

```
        birleştirme(tahta, i,n-1);
    }
}

//O'ları üst ve alt sınırda birleştir for(int j=0; j<n;
j++){ if(tahta[0][j] == 'O') { birleştirme(tahta, 0,j);

}

if(tahta[m-1][j] == 'O')
{ birleştirme(tahta, m-1,j);
}
}

//panoyu for(int i=0;i<m;j+
+){ for(int j=0; j<n; j++)}{ için
işle if(tahta[i][j] == 'O')
{ tahta[i][j] =
'X'; }else if(board[i][j] == '#')
{ board[i][j] = 'O';

}
}
}
}

genel geçersiz birleştirme(char[][] kartı, int i, int j){
if(i<0 || i>=tahta.uzunluk || j<0 || j>=tahta[0].uzunluk)
geri dönmek;

if(board[i][j] != 'O') dönüşü;

panel[i][j] = '#';

birleştirme(tahta, i-1, j);
birleştirme(tahta, i+1, j);
birleştirme(tahta, i, j-1);
birleştirme(tahta, i, j+1);
}
```

Bu çözüm, java.lang.StackOverflowError'a neden olur, çünkü büyük bir pano için de birçok yöntem çağrısına yiğina itilir ve taşmaya neden olur.

90.3 Önce Nefes Araması

Bunun yerine, önce nefes araması yapmak için bir kuyruk kullanınız.

```
genel sınıf Çözümü {
    // BFS yapmak için bir sıra
    kullanın özel Kuyruk<Tamsayı> sıra = new LinkedList<Tamsayı>();

    public void çöz(char[][] board)
    { eğer (tahta == boş || tahta.uzunluk == 0)
        geri dönmek;

        int m = tahta.uzunluk; int n
        = pano[0].uzunluk;

        // O'ları sol ve sağ kenarda birleştir for (int i = 0; i
        < m; i++) { if (board[i][0] == 'O') { bfs(board, i, 0);

        }

        if (tahta[i][n - 1] == 'O') { bfs(tahta, i, n -
            1);
        }
    }

    // O'ları üst ve alt sınırda birleştir for (int j = 0; j <
    n; j++) { if (board[0][j] == 'O') { bfs(board, 0, j);

        }

        if (tahta[m - 1][j] == 'O') { bfs(tahta, m -
            1,j);
        }
    }

    // tahtayı şu şekilde işle :
    (int i = 0; i < m; i++) {
        for (int j = 0; j < n; j++) { if (board[i][j] ==
            'O') { tahta[i][j] = 'X'; } else if (board[i]
            [j] == '#') { board[i][j] = 'O';
            }

            }
        }
    }

    özel geçersiz bfs(char[][] tahta, int i, int j)
    { int n = pano[0].uzunluk;

        // önce akımı sonra komşularını doldur fillCell(board, i, j);
    }
}
```

90 Çevrelenmiş Bölge

```
while (!queue.isEmpty()) { int cur
    =kuyruk.anket(); int x = kur / n; int
    y = kür %n;

    fillCell (değer, x -- 1, y); fillCell(değer,
    x+1, y); fillCell (değer, x, y -- 1); fillCell
    (değer , x , y + 1 );

}

{

    özel geçersiz fillCell(char[][] tahta, int i, int j) { int m = tahta.uzunluk; int n =
    pano[0].uzunluk; if (i < 0 || i >= m || j < 0 || j >= n || board[i][j] !
    = 'O')

        geri dönmek;

    // geçerli hücreyi ekle ve ardından komşularını bfs'de işle kuyruğu.offer(i * n + j); pano[i]
    [j] = '#';

}
}
```

91 Maksimal Dikdörtgen

0'lar ve 1'lerle dolu bir 2B ikili matris verildiğinde , hepsini içeren en büyük dikdörtgeni bulun ve alanını bulun.

91.1 Analizi

Bu problem "Histogramdaki En Büyük Dikdörtgen" problemine dönüştürülebilir .

91.2 Java Çözümü

```
public int maximalRectangle(char[][] matrix) {
    int m = matrix.length; int n =
    m == 0 ? 0 : matrix[0].length; int[] yükseklik = yeni
    int[m][n + 1];

    int maxAlan =
    0; için (int ben = 0; ben < m; i++)
    { for (int j = 0; j < n; j++) { if (matrix[i][j] ==
        '0') { yükseklik[i][j] = 0; } başka
        { yükseklik[i][j] = ben == 0 ? 1 :
        yükseklik[i - 1][j] + 1;

    }
    }
}

for (int i = 0; i < m; i++) { int alan =
    maxAreaInHist(height[i]); eger (alan >
    maksimumAlan) {
    maksimumAlan = alan;
}
}

maxArea'yi döndür ;
}

özel int maxAreaInHist(int[] yükseklik) {
    Yığın<Tamsayı> yığın = yeni Yığın<Tamsayı>();

    int ben = 0;
    int maksimum = 0;
```

91 Maksimal Dikdörtgen

```
while (i < yükseklik.uzunluk)
    { if (stack.isEmpty() || yükseklik[stack.peek()] <= yükseklik[i]) { stack.push(i+
    +); } else { int t = stack.pop(); max = Math.max(max, height[t] *
    (stack.isEmpty() ? i : i - stack.peek() - 1));
    }
}
maksimum dönüş;
}
```

92 Maksimum Araba

0'lar ve 1'lerle dolu bir 2B ikili matris verildiğinde , tüm 1'leri içeren en büyük kareyi bulun ve alanını bulun.

Örneğin, aşağıdaki matris verildiğinde:

```
1101  
1101  
1111
```

4. dönüş

92.1 Analiz

Bu problem dinamik programlama ile çözülebilir. Değişen koşul: $t[i][j] = \min(t[i][j-1], t[i-1][j], t[i-1][j-1]) + 1$. Bu noktadan önce oluşan kare demektir.

92.2 Java Çözümü

```
public int maximalSquare(char[][] matrix) {  
    if (matrix == null || matrix.length == 0 || matrix[0].length == 0)  
        return 0;  
  
    int m = matrix.length; int n =  
    matrix[0].length;  
  
    int[][] t = new int[m][n];  
  
    //üst satır for  
    for (int i = 0; i < m; i++) { t[i][0] =  
        Character.getNumericValue(matrix[i][0]);  
    }  
  
    //soldaki sütun for  
    for (int j = 0; j < n; j++) { t[0][j] =  
        Character.getNumericValue(matrix[0][j]);  
    }  
  
    // içindeki hücreler (int  
    for (int i = 1; i < m; i++) {  
        for (int j = 1; j < n; j++) {  
            if (matrix[i][j] == '0')  
                t[i][j] = 0;  
            else  
                t[i][j] = Math.min(t[i-1][j],  
                    Math.min(t[i][j-1], t[i-1][j-1])) + 1;  
        }  
    }  
}
```

92 Maksimum Araba

```
for (int j = 1; j < n; j++) { if (matris[i][j]
== '1') { int min = Math.min(t[i - 1]
[j], t[i - 1][j - 1]); min = Math.min(min,t[i] [j - 1]); t[i][j] = min + 1; }
başka { t[i][j] = 0;

}

}

}

int maksimum = 0; //
(int i = 0; i < m; i++) için
maksimum uzunluk elde et
{ for (int j = 0; j < n; j++) { if (t[i][j] > max)
{ max = t[i][j];

}
}
}

maksimum dönüş * maksimum;
}
```

93 Kelime Arama

2B bir tahta ve bir kelime verildiğinde, kelimenin izgarada olup olmadığını bulun.

Sözcük, "bitişik" hücrelerin yatay veya dikey olarak komşu olduğu sıralı olarak bitişik hücrenin harflerinden oluşturulabilir. Aynı harf hücresi birden fazla kullanılamaz.

Örneğin verilen tahta =

[

["ABCE"],

["SFC"],

["ADEE"]

]

word = "ABCED", -> true döndürür, word = "SEE", -> true döndürür, word = "ABCB",
->yanlış döndürür.

93.1 Analiz

Bu sorun, tipik bir DFS yöntemi kullanılarak çözülebilir.

93.2 Java Çözümü

```
public boolean var(char[] board, String word) { int m = board.length; int  
n = pano[0].uzunluk;  
  
mantıksal sonuç = yanlış; for(int  
i=0; i<m; i++){  
    for(int j=0; j<n; j++){  
        if(dfs(board,word,i,j,0)){ sonuç = true;  
    }  
}  
}  
  
dönüş sonucu;  
}  
  
public boolean dfs(char[][] board, String word, int i, int j, int k){ int  
m = tahta.uzunluk;
```

93 Kelime Arama

```
int n = pano[0].uzunluk;

if(i<0 || j<0 || i>=m || j>=n){ false döndürür;

}

if(board[i][j] == word.charAt(k)){ char temp = board[i][j];
board[i][j]='#'; if(k==word.length()-1){ true
döndürür;

}else if(dfs(tahta, kelime, i-1, j, k+1) | |dfs(tahta, kelime,
i+1, j, k+1) | |dfs(tahta, kelime, i, j- 1, k+1) | |dfs(tahta,
sözcük, i, j+1, k+1)){

    doğru dönüş;

} pano[i][j]=temp;
}

yanlış dönüş;
}
```

94 Kelime Arama II

2B bir tahta ve sözlükten bir kelime listesi verildiğinde, tahtadaki tüm kelimeleri bulun.

Her kelime, "bitişik" hücrelerin yatay veya dikey olarak komşu olduğu sıralı olarak bitişik hücrenin harflerinden oluşturulmalıdır. Aynı harf hücresi bir kelimedede birden fazla kullanılamaz.

Örneğin, verilen kelimeler = ["yemin", "bezelye", "ye", "yağmur"] ve tahta =

```
[  
  ['o','a','a','n'], ['e','t','a','e'],  
  ['i','h','k','r'], ['i','f','l','v']  
]
```

["ye", "yemin"] iade et.

94.1 Java Çözümü 1

Kelime Aramaya benzer şekilde, bu sorun DFS tarafından çözülebilir. Ancak bu çözüm zaman sınırlını aşıyor.

```
genel Liste<String> findWords(char[][] board, String[] kelimeler) {  
    ArrayList<String> sonuç = new ArrayList<String>();  
  
    int m = tahta.uzunluk; int n  
    = pano[0].uzunluk;  
  
    for (Dize kelime : kelimeler) { boolean  
        bayrağı = false; için (int) ben = 0;  
        ben < m; i++) { for (int j = 0; j < n;  
            j++) { char[][] yeniBoard = yeni  
            char[m][n]; için (int) x = 0; x < m; x++)  
  
                için (int) y = 0; y < n; y++)  
                    yeniTahta[x][y] = tahta[x][y];  
  
                if (dfs(newBoard, word, i, j, 0)) { flag = true;  
                }  
            }  
        } if (bayrak) {
```

94 Kelime Arama II

```
        sonuç.add(kelime);
    }
}

dönüş sonuç;
}

public boolean dfs(char[][] board, String word, int i, int j, int k) { int m = board.length; int n =
pano[0].uzunluk;

if (i < 0 || j < 0 || i >= m || j >= n || k > kelime.uzunluk() - 1) { yanlış
    dönüş;
}

if (board[i][j] == word.charAt(k)) { char temp =
    board[i][j]; pano[i][j] = '#';

    eğer (k == kelime.uzunluk() - 1) { true
        döndürür; } else if (dfs(tahta, kelime,
        i - 1, j, k + 1) || dfs(tahta, kelime, i + 1, j, k + 1) || dfs(tahta,
        kelime, i, j - 1, k + 1) || dfs(tahta, kelime, i, j + 1, k + 1)) {

        pano[i][j] = sıcaklık; doğru
        dönüş;
    }

} else
    { false döndür;
}

yanlış dönüş;
}
```

94.2 Java Çözümü 2 - Trie

Mevcut aday tüm kelimelerin ön ekinde yoksa, geri izlemeyi hemen durdurabiliriz. Bu bir trie yapısı kullanılarak yapılabilir.

```
genel sınıf Çözümü {
    Set<String> sonuç = new HashSet<String>();

    genel Liste<String> findWords(char[][] board, String[] kelimeleri) {
        //HashSet<String> sonuç = yeni HashSet<String>();
        Trie Trie = new Trie();
```

```

for(Dize kelime: kelimeler)
    { trie.insert(kelime);
}

int m=tahta.uzunluk; int
n=tahta[0].uzunluk;

boolean[][] ziyaret edildi = yeni boolean[m][n];

for(int i=0; i<m; i++)
{
    for(int j=0; j<n; j++){
        dfs(pano,
            ziyaret edildi, "", i, j, trie );
    }
}

yeni ArrayList<String>(sonuç) döndürür ;
}

public void dfs(char[][] board, boolean[][] ziyaret edildi, String str, int i, int j, Trie trie){ int
m=board.length; int n=tahta[0].uzunluk;

if(i<0 || j<0 || i>=m || j>=n){ dönüş;
}

if(ziyaret edildi[i][j])
    dönüş;

dizi = dizi + pano[i][j];

if(!trie.startsWith(str))
    geri dönmek;

if(trie.search(str))
    { sonuç.add(str);
}

ziyaret edildi[i][j]=true;
dfs(pano, ziyaret edildi, str, i-1, j, trie); dfs(pano, ziyaret
edildi, str, i+1, j, trie); dfs(pano, ziyaret edildi, str, i, j-1, trie);
dfs(tahta, ziyaret edildi, str, i, j+1, trie); ziyaret edildi[i]
[j]=yanlış;
}

```

//Üçlü Düğüm

94 Kelime Arama II

```
sınıf TrieNode { genel
    TrieNode[] çocukları = yeni TrieNode[26]; genel Dize ögesi = "";
}

//Üçlü
sınıfı Trie{ public
    TrieNode kökü = yeni TrieNode();

    genel geçersiz ekleme(String word){ TrieNode
        düğümü = kök; for(char c: word.toCharArray())
        { if(node.children[c-'a']==null)
            { node.children[c-'a']= new TrieNode();

                } düğüm = düğüm.çocuklar[c-'a'];

            } düğüm.öge = sözcük;
        }

        genel boole araması(String word){ TrieNode
            düğümü = kök; for(char c: word.toCharArray()) {

                if(node.children[c-'a']==boş)
                    yanlış dönüş; düğüm =
                    düğüm.çocuklar[c-'a'];

                } if(node.item.equals(word)){
                    doğru dönüş; }
                else{ false döndürür;

                }
            }

        public booleanstartsWith(String öneki)
            { TrieNode düğümü = kök; for(char
                c:prefix.toCharArray())
                    { if(node.children[c-'a']==null) false döndürür;
                        düğüm = düğüm.çocuklar[c-'a'];

                    } true döndür;
            }
        }
```

95 Tamsayı Arası

Pozitif bir n tam sayısı verildiğinde, onu en az iki pozitif tam sayının toplamına bölün ve bu tam sayıların çarpımını maksimize edin. Alabileceğiniz maksimum ürünü iade edin.

Örneğin, n = 2 verildiğinde , 1 (2 = 1 + 1) döndürür ; n = 10 verildiğinde, 36 döndürür (10 = 3 + 3 + 4).

95.1 Java Çözümü 1 - Dinamik Programlama

i sayısını kırmak için maksimum üretim değeri dp[i] olsun. dp[i+j], i*j olabileceğinden, $dp[i+j] = \max(\max(dp[i], i) * \max(dp[j], j)))$, $dp[i+j]]$.

```
genel int tamsayıBreak(int n)
{ int[] dp = yeni int[n+1];

    for(int i=1; i<n; i++)
        { for(int j=1; j<i+1; j++){ if(i+j<=n)

            { dp[i+j]=Math.max(Math.max(dp[i],i)*Math .max(dp[j],j), dp[i+j]); }

        }
    }

    dönüş dp[n];
}
```

95.2 Java Çözümü 2 - Kuralları Kullanma

Bazı sayılar için kırılma sonucunu görürsek, aşağıdaki gibi tekrarlanan formasyon görebiliriz:

```
2 -> 1*1
3 -> 1*2
4 -> 2*2
5 -> 3*2
6 -> 3*3
7 -> 3*4 8
-> 3*3*2
9 -> 3*3*3
10 -> 3*3*4
11 -> 3*3*3*2
```

95 Tamsayı Arası

Sadece $n > 4$ olduğunda kaç tane 3 alabileceğimizi bulmamız gerekiyor . eğer n

```
genel int tamsayıBreak(int n) {  
  
    if(n==2) 1 döndürür ;  
    if(n==3) 2 döndürür ;  
    if(n==4) 4 döndürür ;  
  
    int sonuç=1;  
    if(n%3==0){ int m  
        = n/3; sonuç =  
        (int) Math.pow(3, m); }  
    else if(n%3==2){ int m=n/3;  
        sonuç = (int)  
        Math.pow(3, m) } * 2;  
    else if(n%3==1){ int  
        m=(n-4)/3; sonuç =  
        (int) Math.pow(3, m) *4;  
    }  
  
    dönüş sonucu;  
}
```

96 Aralık Toplami Sorgusu 2B Sabit

Bir 2B matris matrisi verildiğinde , sol üst köşesi (sıra1, sütun1) ve sağ alt köşesi (sıra2, sütun2) tarafından tanımlanan dikdörtgenin içindeki elemanların toplamını bulun .

96.1 Analiz

Varsayımdır, sumRegion yöntemine yapılan birçok çağrı olduğu için, ara sonuçları depolamak için biraz fazladan boşluk kullanmalıyız. Burada ,~~(Toplamda geri dönmek için klasördeki kütüphaneleri kullanmayı biliyoruz)~~ tanımlarız .

96.2 Java Çözümü

```
genel sınıf NumMatrix { int [][]  
    toplam;  
  
    public NumMatrix(int[][] matrix) { if(matrix==null  
        || matrix.length==0 || matrix[0].length==0)  
        geri dömek;  
  
        int m = matrix.uzunluk; int n  
        = matrix[0].uzunluk; toplam =  
        yeni int[m][n];  
  
        for(int i=0; i<m; i++){ int  
            sumRow=0; for(int j=0; j<n;  
            j++){ if(i==0){ sumRow +=  
                matrix[i][j]; toplam[i]  
                [j]=sumRow; }else{ sumRow  
                += matrix[i][j]; toplam[i]  
                [j]=toplamSatır+toplam[i-1][j];  
  
            }  
        }  
    }  
    }  
  
    public int toplamBölge(int satır1, int sütun1, int satır2, int sütun2) { if(this.sum==null)  
        return 0;  
    else  
        return this.sum[satır1][sütun1] - this.sum[satır1][sütun2] - this.sum[satır2][sütun1] + this.sum[satır2][sütun2];  
    }  
}
```

96 Aralık Toplamı Sorgusu 2B Sabit

0 dönüşü ;

```
int topRightX = satır1; int  
üstSağY = col2;  
  
int altSolX=satır2; int altSolY=  
col1;  
  
int sonuç=0;  
  
if(satır1==0 && sütun1==0){  
    sonuç = toplam[satır2][sütun2]; }  
else if(row1==0){ sonuç = toplam[row2]  
[col2] -toplam[altSolX]  
[altSolY-1];  
  
}else if(col1==0){ sonuç =  
toplam[row2][col2] -sum[topRightX-1]  
[üstSağY];  
  
} else{ sonuç = toplam[row2][col2]  
-toplam[topRightX-1][topRightY]  
-toplam[bottomLeftX][bottomLeftY-1]  
+toplam[row1-1][col1-1];  
}  
  
dönüş sonucu;  
}  
}
```

97 Matristeki En Uzun Artan Yol

Bir tamsayı matrisi verildiğinde, en uzun artan yolun uzunluğunu bulun.

Her hücreden dört yöne gidebilirsiniz: sola, sağa, yukarı veya aşağı. Sen
çapraz olarak hareket edemez veya sınırın dışına çıkamaz

97.1 Java Çözümü 1 - DFS

Bu çözüm zaman sınırını aşıyor.

genel sınıf Çözüm { int en uzun=0;

```
public int longIncreasingPath(int[][] matrisi) {
    if(matrix==null || matrix.length==0 || matrix[0].length==0)
        dönüş ;
    for(int i=0; i<matrix.length; i++)
        { for(int j=0; j<matrix[0].uzunluk; j++){
            helper(matris, i, j,
            1);
        }
    }
    en uzun dönüş ;
}

genel geçersiz yardımcı(int[][]) matris, int i, int j, int len){

    if(i-1>=0 && matrix[i-1][j]>matris[i][j]){
        en uzun = Math.max(en
        uzun, len+1); yardımcı(matris, i-1, j, len+1);

    }

    if(i+1<matrix.length && matrix[i+1][j]>matris[i][j]){
        en uzun = Math.max(en uzun, boy+1);
        yardımcı(matris, i+1, j, len+1);
    }

    if(j-1>=0 && matrix[i][j-1]>matris[i][j]){
        en uzun = Math.max(en
        uzun, len+1);itational(matris, i, j-1, len+1);

    }
}
```

97 Matristeki En Uzun Artan Yol

```
        if(j+1<matris[0].uzunluk && matris[i][j+1]>matris[i][j])
            { en uzun = Math.max(en uzun, boy+1);
              yardımcı(matris, i, j+1, len+1);
            }

        }
    }
```

97.2 Java Çözümü - Optimize Edilmiş

```
genel sınıf Çözümü
{ int[] dx = {-1, 1, 0, 0}; int[] dy = {0,
0, -1, 1};

public int longIncreasingPath(int[][] matrisi) {
    if(matrix==null | matrix.length==0 | matrix[0].length==0)
        dönüş;

    int[][] mem = new int[matrix.uzunluk][matris[0].uzunluk]; int en uzun=0;

    for(int i=0; i<matrix.length; i++)
        { for(int j=0; j<matrix[0].length; j++){ en uzun =
            Math.max(en uzun, dfs(matrix, i, j, mem));
        }
    }

    en uzun dönüş;
}

public int dfs(int[][] matrix, int i, int j, int[][] mem){ if(mem[i][j]!=0) return mem[i][j];

for(int m=0; m<4; m++){ int x =
    i+dx[m]; int y = j+dy[m];

    if(x>=0&y>=0&&x<matrix.length&&y<matrix[0].length&&matrix[x][y]>matris[i][j]){
        mem[i][j]=Math.max(mem[i][j], dfs(matrix, x, y, mem));
    }
}

++ mem [i][j];
}
}
```

98 Dizi Kullanarak Yığın Gerçekleştirme

java

Bu gönderi, bir dizi kullanarak bir yığının nasıl uygulanacağını gösterir.

Yığın gereksinimleri şunlardır: 1) yığının, boyutunu başlatmak için bir sayı kabul eden bir yapıcısı vardır, 2) yığın her türden öğeyi tutabilir, 3) yığının bir push() ve bir pop() yöntemi vardır.

Joshua Bloch tarafından yazılan "Etkin Java" kitabında benzer bir örnek olduğunu hatırlıyorum, ancak örneğin nasıl kullanıldığından emin değilim. Bu yüzden sadece bir tane yazıyorum ve sonra kitabı okuyorum ve bir şey kaçırıp kaçırmadığımı görüyorum.

98.1 Basit Yığın Uygulaması

```
genel sınıf Yığın<E>
{ özel E[] dizi = boş; özel int CAP;
  özel int üst = -1; özel int boyutu
  = 0;

  @SuppressWarnings("işaretlenmemiş")
  genel Yığın(int büyük harf) { this.CAP =
    büyük harf; this.arr = (E[]) yeni
    Nesne[cap];
  }

  public E pop()
  { if(this.size == 0){ null
    yönlendirir;
  }

    this.size--; E
    sonuç = this.arr[top]; this.arr[top]
    = null;//belleğin sizmasını öle this.top--;
    dönüş sonucu;
  }

  public boolean push(E e) { if (!isFull())
    yanlış dönüş;
```

98 Java'da Dizi Kullanarak Yığın Gerçekleştirme

```
this.size++;
this.arr[++üst] = e; yanlış
dönüş;
}

genel boole isFull()
{ if (this.size == this.CAP) false
    döndürür; doğru dönüş;

}

public String toString() { if(this.size==0)
{ null yürüter;

}

StringBuilder sb = yeni StringBuilder(); for(int i=0;
i<this.size; i++){
    sb.append(this.arr[i] + ", ");
}

sb.setLength(sb.length()-2); sb.toString ();

}

genel statik geçersiz main(String[] args) {

Yığın<Dize> yığın = yeni Yığın<Dize>(11); stack.push("merhaba");
stack.push("dünya");

System.out.println(yığın);

yığın.pop();
System.out.println(yığın);

yığın.pop();
System.out.println(yığın);
}
}
```

Çıktı:

Selam Dünya
Merhaba
hükümsüz

98.2 "Etkin Java"dan Bilgi

Görünüşe göre hiçbir şeyi iyileştirmeme gerek yok. Bazı adlandırma farklılıklarları var ama genel olarak yöntemim tamam.

Bu örnek, "Etkili Java"da iki kez oluşur. İlk olarak, bellek sizıntısını göstermek için yiğin örneği kullanılır. İkinci olarak, örnek, denetlenmeyen uyarıları ne zaman bastırabileceğimizi göstermek için kullanılır.

Bir dizi kullanarak kuyruğu nasıl uygulayacağınızı merak ediyor musunuz?

99 İki Numara Toplama

Size negatif olmayan iki sayıyı temsil eden iki bağlantılı liste verilir. Basamaklar ters sırada saklanır ve düğümlerinin her biri tek bir basamak içerir. İki numarayı ekleyin ve bağlantılı bir liste olarak döndürün.

Giriş: (2 ->4 ->3) + (5 ->6 ->4) Çıkış: 7 ->0 ->8

99.1 Java Çözümü

```
genel sınıf Çözüm { genel
    ListNode addTwoNumbers(ListNode l1, ListNode l2) { int taşıma =0;

        ListNode newHead = yeni ListNode(0);
        ListNode p1 = l1, p2 = l2, p3=yeniBaşlat;

        while(p1 != boş || p2 != boş)
            { if(p1 != null){ taşıma +=
                p1.val; p1 =
                p1.sonraki;
            }

            if(p2 != null){ taşıma
                += p2.val; p2 =
                p2.sonraki;
            }

            p3.next = yeni ListNode(taşıma %10); p3 =
            p3.sonraki; taşımak /= 10;

        }

        if(carry==1)
            p3.next=new ListNode(1);

        newHead.next'i döndür ;
    }
}
```

Basamaklar ters sıra yerine normal sırada saklanırsa ne olur?

Cevap: Listeyi tersine çevirebilir, sonucu hesaplayabilir ve sonucu tersine çevirebiliriz.

100 Yeniden Sıralama Listesi

Tek bağıntılı bir liste verildiğinde L: L0L1 ... Ln-1Ln, yeniden sıralayın: L0LnL1Ln
1L2Ln-2...

Örneğin, 1,2,3,4 verildiğinde, 1,4,2,3 olarak yeniden sıralayın . Bunu yerinde olmadan yapmalısınız. düğümlerin değerlerini değiştirmek.

100.1 Analiz

"Yerinde" işlemler gerektirdiğinden, bu sorun basit değildir. Bu, yeni bir liste oluşturmadan yalnızca işaretçilerini değiştirebileceğimiz anlamına gelir.

100.2 Java Çözümü

Bu sorun aşağıdakileri yaparak çözelebilir:

- Listeyi ortadan iki listeye ayırin (hızlı ve yavaş işaretçiler kullanın) • İkinci listenin sırasını tersine çevirin
- İki listeyi tekrar birleştirin

Aşağıdaki kod, test içeren tam bir çalıştırılabilir sınıftır.

```
//ListNode'un sınıf tanımı class ListNode
{
    int değer;
    ListNode sonraki;

    public ListNode(int x) { dalga =
        x; sonraki = boş;
    }

    genel sınıf ReorderList {

        public static void main(String[] args) { ListNode n1 = new
            ListNode(1); ListNode n2 = yeni ListNode(2); ListNode
            n3 = yeni ListNode(3); ListNode n4 = yeni ListNode(4);
            n1.sonraki = n2; n2.sonraki = n3;
```

100 Yeniden Sıralama Listesi

```
n3.sonraki = n4;

listeyi yazdır(n1);

reorderList(n1);

listeyi yazdır(n1);
}

genel statik geçersiz yeniden sıralama Listesi (ListNode kafası) {

if (head != null && head.next != null) {

    ListNode slow = kahve;
    ListNode hızlı = baş;

    //bağlantıyı iki parçaya ayırmak için hızlı ve yavaş bir işaretçi kullanın. while (hızlı != boş && hızlı.next != boş) {
        //neden üçüncü/ikinci koşula ihtiyaç var?
        System.out.println("ön " + slow.val + yavaş = yavaş.next; hızlı =
                           + hızlı.val);
        hızlı.next.next; System.out.println("sonra "
                           + yavaş.val +
                           + hızlı.val);
    }

    ListNode saniye = yavaş.sonraki; slow.next =
    null;// ilk kısmı kapatmanız gerekiyor

    // şimdi iki liste olmalı: baş ve hızlı

    // ikinci kısım ters sıra saniye = reverseOrder(saniye);

    ListNode p1 = baş;
    ListNode p2 = saniye;

    // iki listeyi burada birleştir while (p2 !=
    = null) { ListNode temp1 = p1.next;
        ListNode temp2 = p2.sonraki;

        p1.sonraki = p2;
        p2.sonraki = temp1;

        p1 = geçici1; p2 =
        geçici2;
    }
}
}
```

```
genel statik ListNode reverseOrder(ListNode kafası) {  
  
    eğer (head == null || head.next == null) { dönüş  
        kafası;  
    }  
  
    ListNode açık = kahve; ListNode  
    curr = baş.sonraki;  
  
    while (curr != null) { ListNode  
        sıcaklık = akım.sonraki; curr.next = siz;  
  
        ön = akım;  
        akım = sıcaklık;  
    }  
  
    // baş düğümün bir sonraki  
    başlığını ayarla.next = null;  
  
    dönüş öncesi;  
}  
  
genel statik geçersiz printList(ListNode n)  
{ System.out.println("----"); while (n != boş)  
{ System.out.print(n.val); n = n.sonraki;  
  
}  
System.out.println();  
}  
}
```

100.3 Paket Mesajlar

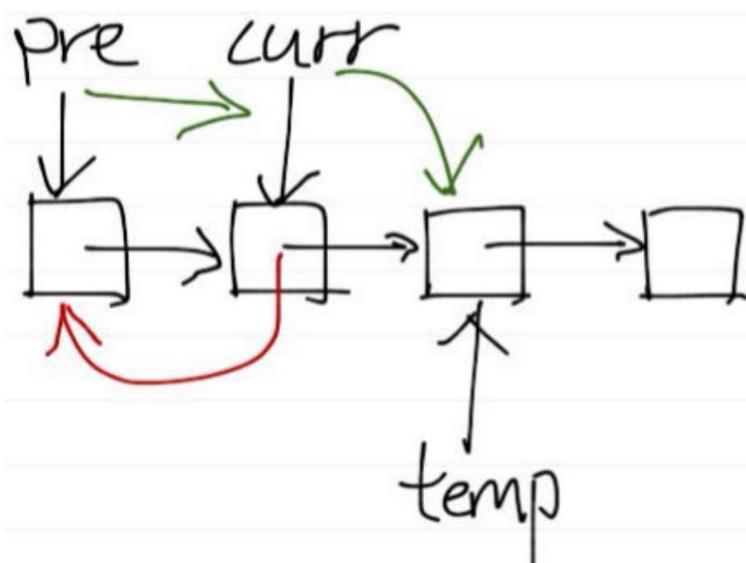
Üç adım, bağıntılı listenin diğer sorunlarını çözmek için kullanılabilir. Küçük bir şema onları daha iyi anlamana yardımcı olabilir.

Ters Liste:

```
ListNode pre = head;
ListNode curr = head.next;

while (curr != null) {
    ListNode temp = curr.next;
    curr.next = pre;
    pre = curr;
    curr = temp;
}

head.next = null;
```



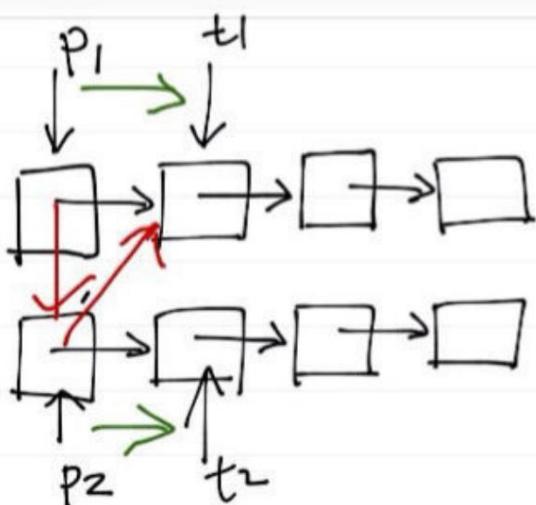
Listeyi Birleştir:

```
ListNode p1 = head;
ListNode p2 = second;

//merge two lists here
while (p2 != null) {
    ListNode temp1 = p1.next;
    ListNode temp2 = p2.next;

    p1.next = p2;
    p2.next = temp1;

    p1 = temp1;
    p2 = temp2;
}
```

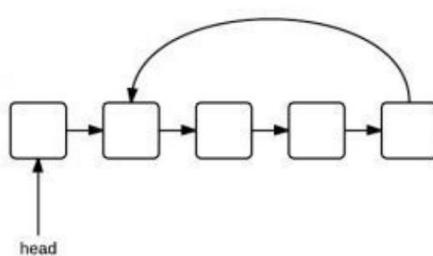


101 Bağlantılı Liste Döngüsü

Bağlantılı bir liste verildiğinde, içinde bir döngü olup olmadığını belirleyin.

101.1 Analiz

2 işaretçimiz varsa - hızlı ve yavaş. Bir daire varsa, hızlı olanın yavaş olanla buluşması garanti edilir.



101.2 Java Çözümü

```
genel sınıf Çözüm { genel boolean
    hasCycle(ListNode head) {
        ListNode hızlı = kafa;
        ListNode yavaş = kahve;

        while(hızlı != boş && hızlı.sonraki != boş){ yavaş = yavaş.sonraki;
            hızlı = hızlı.sonraki.sonraki;

            if(yavaş == hızlı)
                doğru dönüş;
        }

        yanlış dönüş;
    }
}
```

Rastgele İşaretçili 102 Kopya Listesi

Bağlantılı bir liste, her düğüm, listedeki herhangi bir düğümü veya null'u işaret edebilecek ek bir rasgele işaretçi içerecek şekilde verilir.

Listenin derin bir kopyasını döndürün.

102.1 Java Çözümü 1

Aşağıdaki adımları uygulayarak bu sorunu çözebiliriz:

- her düğümü kopyala, yani her düğümü çoğalt ve listeye ekle yeni
- oluşturulan tüm düğümler için rastgele işaretçileri kopyala
- listeyi ikiye böl

```
public RandomListNode copyRandomList(RandomListNode head) {  
  
    eğer (head == null) null  
        değerini döndürürse;  
  
    RandomListNode p = önbellek;  
  
    // her düğümü kopyala ve listeye ekle while (p != null)  
    { RandomListNode kopyası = yeni RandomListNode(p.label);  
        kopyala.sonraki = p.sonraki; p.next = kopyala; p = kopyala.sonraki;  
  
    }  
  
    // her yeni düğüm için rasgele işaretçiyi kopyala p = kafa; while  
(p != boş) {  
  
        if (p.random != null)  
            p.next.random = p.random.next; p =  
            p.sesli.sesli;  
    }  
  
    // listeyi ikiye böl p = baş;  
    RandomListNode newHead =  
    head.next; while (p != null) { RandomListNode  
        sıcaklık = p.sonraki;
```

Rastgele İşaretçili 102 Kopya Listesi

```
p.sonraki = geçici.sonraki;
eğer (temp.sonraki != boş)
    geçici.sonraki = geçici.sonraki.sonraki;
    p = p.sonraki;
}

newHead'i döndür ;
}
```

İşaretçiyi her seferinde 2 adım hareket ettiren yukarıdaki parça listesi kısmı , aşağıdaki gibi daha basit olan bir seferde bir tane de hareket ettirebilirsiniz:

```
while(p != null && p.next != null){ RandomListNode
    temp = p.next; p.sonraki = geçici.sonraki; p =
    sıcaklık;
}

}
```

102.2 Java Çözümü 2 - HashMap Kullanımı

Xiaomeng'in aşağıdaki yorumundan, onu daha basit hale getiren bir HashMap kullanabiliriz.

```
public RandomListNode copyRandomList(RandomListNode head) { if (head == null)
    null döndürür;

    HashMap<RandomListNode, RandomListNode> haritası = yeni HashMap<RandomListNode,
        RandomListNode>();
    RandomListNode newHead = yeni RandomListNode(head.label);

    RandomListNode p = kafa;
    RandomListNode q = yeniBaşlık;
    map.put(kafa, yeniKafa);

    p = p.sonraki;
    while (p != boş)
        { RandomListNode temp = new RandomListNode(p.label); map.put(p,
            geçici); q.sonraki = sıcaklık; q = sıcaklık; p = p.sonraki;

    }

    p = baş; q
    = yeni Başlık;
    while (p != boş)
        { eğer (p.rastgele != boş)
            q.random = map.get(p.random); başka
```

q.sağ = boş;

```
p = p.ses; q =  
q.ses;  
}  
  
newHead'i döndür ;  
}
```

103 Sıralanmış İki Listeyi Birleştirme

Sıralanmış iki bağlantılı listeyi birleştirin ve yeni bir liste olarak döndürün. Yeni liste, ilk iki listenin düğümlerini birleştirerek yapılmalıdır.

103.1 Analiz

Sorunu çözmenin anahtarı, sahte bir kafa tanımlamaktır. Ardından her listedeki ilk öğeleri karşılaşırın. Küçük olanı birleştirilmiş listeye ekleyin. Son olarak, bunlardan biri boş olduğunda, zaten sıralanmış olduğu için onu birleştirilmiş listeye eklemeniz yeterlidir.

103.2 Java Çözümü

```
/**  
 * Tek bağlantılı liste tanımı. * genel sınıf ListNode  
 * { int val; * Sıradaki ListNode; * ListNode(int x) { val =  
 *     x; sonraki = boş;  
  
 *     }  
 * }  
 */  
genel sınıf Çözüm { genel  
    ListNode selectTwoLists(ListNode l1, ListNode l2) {  
  
        Liste Düğümü p1 = l1;  
        Liste Düğümü p2 = l2;  
  
        ListNode fakeHead = yeni ListNode(0);  
        ListNode p = sahteKafa;  
  
        while(p1 != null && p2 != null)  
            { if(p1.val <= p2.val){ p.next  
                = p1; p1 = p1.sonraki; }  
                başka{ p.sonraki = p2; p2  
                = p2.sonraki;  
            }  
    }
```

103 Sıralanmış İki Listeyi Birleştirme

```
p = p.sonraki; }

if(p1 != boş)
    p.sonraki = p1;
if(p2 != boş) p.sonraki
    = p2;

fakeHead.next'i döndür ;
}
}
```

104 Tek Çift Bağlı Liste

104.1 Sorun

Tek bağlı bir liste verildiğinde, tüm tek düğümleri ve ardından çift düğümleri birlikte grupperdir.

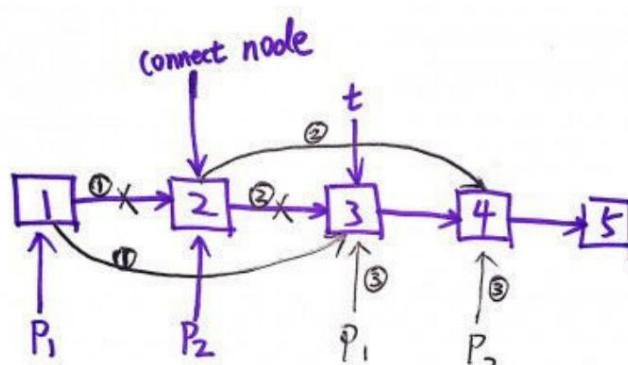
Lütfen burada düğümlerdeki değerden değil düğüm numarasından bahsettiğimize dikkat edin.

Program, $O(1)$ uzay karmaşıklığında ve $O(\text{düğümler})$ zaman karmaşıklığında çalışmalıdır. Örnek:

1->2->3->4->5->NULL verildiğinde,
1->3->5->2->4->BOŞ döndür . . .

104.2 Analiz

Bu sorun iki işaretçi kullanılarak çözülebilir. Bağlantıyı yineliyoruz ve iki işaretçiyi hareket ettiriyoruz.



104.3 Java Çözümü

```
public ListNode tekEvenList(ListNode baş) { if(kafa == boş) dönüş kafası;
```

```
ListNode sonucu = kahve; Liste Düğümlü p1 =
kafa; ListNode p2 = baş.sonraki;
ListNode connectNode = baş.sonraki;
```

104 Tek Çift Bağlantılı Liste

```
while(p1 != null && p2 != null){ ListNode t =  
    p2.next; if(t == null) break;  
  
    p1.sonraki = p2.sonraki;  
    p1 = p1.sonraki;  
  
    p2.sonraki = p1.sonraki;  
    p2 = p2.sonraki;  
}  
  
p1.next = bağlantıNode;  
  
dönüş sonucu;  
}
```

105 Yinelenenleri Sıralı Listededen Kaldır

Sıralanmış bir bağıntılı liste verildiğinde, her öğe yalnızca görünecek şekilde tüm kopyaları silin bir kere.

Örneğin,

1->1->2 verildiğinde, 1->2'yi **döndürün** .

1->1->2->3->3 verildiğinde, 1->2->3'ü **döndürün** .

105.1 Düşünceler

Bu sorunun anahtarı, doğru döngü koşulunu kullanmaktır. Ve her döngüde gerekli olanı değiştirin. Aşağıdaki 2 çözüm gibi farklı yineleme koşullarını kullanabilirsiniz .

105.2 Çözüm 1

```
/*
 * Tek bağıntılı liste tanımı. * genel sınıf ListNode { * int val; *
 * Sıradaki ListNode; * ListNode(int x) { val = x; sonraki = boş;
 *
 * }
 */
genel sınıf Çözümü
{ public ListNode deleteDuplicates(ListNode head) { if(head == null || head.next == null) dönüş kafası;

    ListNode önceki = kafa;
    ListNode p = baş.sonraki;

    while(p != null){ if(p.val
        == önceki.val){ önceki.sonraki =
        p.sonraki; p = p.sonraki; //
        değişiklik yok önceki
```

105 Yinelenenleri Sıralı Listededen Kaldır

```
        }
        başka{ önceki = s;
        p = p.sonraki;
    }
}

dönüş kafası;
}
}
```

105.3 Çözüm 2

genel sınıf Çözümü

```
{ public ListNode deleteDuplicates(ListNode head) { if(head == null ||

head.next == null) dönüş kafası;

ListNode p = kahve;

while( p!= null && p.next != null){ if(p.val ==
p.next.val){ p.next = p.next.next; }başka{ p =
p.sonraki;

}

dönüş kafası;
}
}
```

106 Yinelenenleri Sıralı Listeden Kaldır

III

Sıralanmış bir bağlantılı liste verildiğinde, yinelenen numaralara sahip tüm düğümleri silin ve orijinal listeden yalnızca farklı numaralar bırakın.

Örneğin, 1->1->1->2->3 verildiğinde, 2->3 döndürün .

106.1 Java Çözümü

```
public ListNode deleteDuplicates(ListNode head) { ListNode t = yeni
    ListNode(0); t.sonraki = kahve;

    Liste Düğümü p = t;
    while(p.next!=null&&p.next.next!=null){ if(p.next.val ==
        p.next.next.val){ int dup = p.next.val; süre(p.sonraki!
        =null&&p.next.val==dup){ p.next = p.next.next;

    }
    başka{ p=p.next;
    }

}
    dönüş t.sonraki;
}
```

107 Bölüm Listesi

Bağlantılı bir liste ve x değeri verildiğinde, x'ten küçük tüm düğümler x'ten büyük veya ona eşit düğümlerden önce gelecek şekilde bölümleyin. İki bölümün her birindeki düğümlerin orijinal göreli sırasını korumanız gereklidir.

Örneğin, 1->4->3->2->5->2 ve x = 3 verildiğinde, 1->2->2->4->3->5 döndürün .

107.1 Java Çözümü

[genel sınıf](#) Çözümü

```
{ genel ListNode bölümü(ListNode kafası, int x) { if(head == null) null
    döndürür;

    ListNode fakeHead1 = yeni ListNode(0); ListNode
    fakeHead2 = yeni ListNode(0); fakeHead1.next = kahve;

    ListNode p = kahve;
    Önceki ListNode = fakeHead1;
    ListNode p2 = fakeHead2;

    while(p != null){ if(p.val < x)
        { p = p.next; önceki =
            önceki.sonraki; }
        başka{

            p2.sonraki = p;
            önceki.sonraki = p.sonraki;

            p = önceki.sonraki;
            p2 = p2.sonraki;
        }
    }

    // listeyi kapat p2.next =
    null;

    önceki.next = fakeHead2.next;

    fakeHead1.next'i döndür ;
```

107 Bölüm Listesi

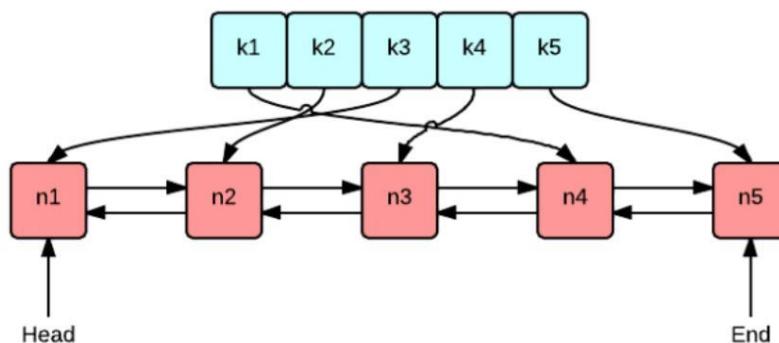
```
    }  
}
```

108 LRU Önbelleği

En Son Kullanılan (LRU) önbelleği için bir veri yapısı tasarlayın ve uygulayın. Şu işlemleri desteklemelidir: al ve ayarla. get(key) - Anahtar önbellekte varsa anahtarın değerini alın (her zaman pozitif olacaktır), aksi takdirde -1 döndürür. set(key, value) - Anahtar zaten mevcut değilse değeri ayarlayın veya girin. Önbellek kapasitesine ulaştığında, yeni bir öğe eklenmeden önce en son kullanılan öğeyi geçersiz kılmalıdır.

108.1 Analiz

Bu sorunu çözmek için anahtarı, düğümleri hızlı bir şekilde taşımamızı sağlayan çift bağlantılı bir liste kullanmakta.



LRU önbelleği, anahtarların ve çift bağlantılı düğümlerin bir karma tablosudur. Hash tablosu, get()'in zamanını O(1) yapar. Çift bağlantılı düğümlerin listesi, düğümleri ekleme/çıkarma işlemlerini O(1) yapar.

108.2 Java Çözümü

Çift bağlantılı bir liste düğümü tanımlayın.

```
sınıf Düğümü{ int
    anahtari; int
    değeri;
    düğüm öncesi;
    sonraki düğüm;
```

108 LRU Ön belleği

```
genel Düğüm(int anahtarı, int değer){ this.key
    = anahtar; this.değer = değer;

}

genel sınıf LRUCache { int
    kapasite;
    HashMap<Tamsayı, Düğüm> map = new HashMap<Tamsayı, Düğüm>();
    Düğüm başı=null;
    Düğüm sonu=null;

    genel LRUCache(int kapasite) { this.capacity
        = kapasite;
    }

    genel int get(int anahtarı)
    { if(map.containsKey(key)){ Düğüm n =
        map.get(key); kaldır(n); setHead(n);
        dönüş n.değer;

    }

    -1 döndürür ;
}

public void kaldır(Düğüm n){ if(n.pre!=null)
{ n.pre.next = n.next; } başka{ kafa =
    n.sonraki;

}

if(n.next!=null){ n.next.pre =
    n.pre; } başka{ bitiş =
    is. için;

}

}

genel geçersiz setHead(Düğüm n)
{ n.sonraki = kafa;
    n.pre = boş;

if(kafa!=boş)
    kafa.pre = n;
```

```
bacak = n;

if(end ==null) end =
    baş;
}

genel geçersiz küme (int anahtarı, int değeri) { if
    (map.containsKey (key)) { eski
        düğüm = map.get(key);
        eski.değer = değer; kaldır(eski);
        setHead(eski); }else{ Düğüm
            oluşturuldu = yeni Düğüm(anahtar,
            değer); if(map.size()>=kapasite)
                { map.remove(end.key); kaldır(son);
                setHead(olşturuldu);

        } else{ setHead(olşturuldu); }

        map.put(anahtar, oluşturuldu);
    }
}
}
```

109 İki Bağlantılı Listenin Kesişimi

109.1 Sorun

Birbirine bağlı iki listenin kesiştiği noktayı bulan programı yazınız.

Örneğin, aşağıdaki iki bağlantılı liste:

A: a1 -> a2
 ->
 c1 -> c2 -> c3
 ->
B: b1 -> b2 -> b3

c1 düğümünde kesişmeye başlar .

109.2 Java Çözümü

Once iki listenin uzunluğunu hesaplayın ve farkı bulun. Ardından fark uzaklığında daha uzun listeden başlayın, 2 listeyi yineleyin ve düğümü bulun.

```
/**  
 * Tek bağlantılı liste tanımı. * genel sınıf ListNode  
 { * int val; * Snext; * isNode; * bfirstNode(int x)  
  
 *  
 *  
 * }  
 */  
  
genel klasz Çözümü  
{ public ListNode getIntersectionNode(ListNode headA, ListNode headB) { int len1 = 0; int  
len2 = 0;  
  
ListNode p1=headA, p2=headB; eğer (p1 ==  
null || p2 == null) boş döndürürse;  
  
while(p1 != null){ len1++; p1  
= p1.sonraki;
```

109 İki Bağlantılı Listenin Kesişimi

```
 } while(p2 != null){ len2+=  
     +; p2 = p2.sonraki;  
  
 }  
  
 int fark = 0;  
 p1=kapaA;  
 p2=kapaB;  
  
 if(len1 > len2){ fark =  
     len1-len2; int ben=0;  
     while(i<diff){ p1 =  
         p1.sonraki; ben++;  
  
     } }  
 else{ fark = len2-len1; int  
     ben=0; while(i<diff){ p2  
         = p2.sonraki; ben++;  
  
     } }  
  
 while(p1 != boş && p2 != boş){ if(p1.val ==  
     p2.val){ dönüş p1; }başka{  
  
     } p1 = p1.sonraki;  
     p2 = p2.sonraki;  
 }  
  
 boş dönüş;  
 }  
 }
```

110 Bağlantılı Liste Öğelerini Kaldır

val değerine sahip bağlantılı bir tamsayılar listesinden tüm öğeleri kaldırın.

Örnek

Verilen: 1 --> 2 --> 6 --> 3 --> 4 --> 5 --> 6, val = 6

Dönüş: 1 --> 2 --> 3 --> 4 --> 5

110.1 Java Çözümü

Bu sorunu çözmenin anahtarı, listenin başını izlemek için bir yardımcı düğüm kullanmaktır.

```
public ListNode removeElements(ListNode üzerinde, int val) { ListNode
    yardımcı = yeni ListNode(0); helper.next = baş; ListNode p = yardımcı;

    while(p.sonraki != boş)
        { if(p.next.val == val)
            { Sonraki ListNode = p.sonraki;
              p.sonraki = sonraki.sonraki; }
            başka{ p = p.sonraki;

        }
    }

    dönüş = yardımcı.sonraki;
}
```

Çiftler halinde 111 Düğüm Değiştirme

Bağlantılı bir liste verildiğinde, her iki bitişik düğümü değiştirin ve başını döndürün.

Örneğin, 1->2->3->4 verildiğinde, listeyi 2->1->4->3 olarak döndürmelisiniz .

Algoritmanız yalnızca sabit alan kullanmalıdır. içindeki değerleri değiştiremezsiniz. liste, yalnızca düğümlerin kendisi değiştirilebilir.

111.1 Java Çözümü

Her çiftin önceki ve sonraki düğümünü izlemek için iki şablon değişkeni kullanın.

```
herkese açık ListNode swapPairs(ListNode head) { if(head == null
    || head.next == null) dönüş kafası;

    ListNode h = yeni ListNode(0); h.sonraki =
    baş; Liste Düğümü p = h;

    while(p.next != null && p.next.next != null){ //ilk
        //düğümü izlemek için t1'i kullanın
        Liste Düğümü t1 = p; p = p.sonraki; t1.sonraki
        = p.sonraki;

        // t2yi çiftin bir sonraki düğümünü izlemek için kullanın
        ListNode t2 = p.next.next; p.sonraki.sonraki = p; p.sonraki
        = t2;

        }
        dönüş h.sonraki;
    }
```

112 Ters Bağlantılı Liste

Tek bağlantılı bir listeyi tersine çevirin.

112.1 Java Çözümü 1 - Yinelemeli

```
genel ListNode reverseList(ListNode head) { if(head==null ||  
    head.next == null) dönüş kafası;  
  
    ListNode p1 = kahve;  
    ListNode p2 = baş.sonraki;  
  
    baş.sonraki = boş;  
    while (p1 != null && p2 != null) { ListNode t =  
        p2.next; p2.sonraki = p1; p1 = p2; eğer (t!=  
        =null){ p2 = t; }başka{ mola;  
  
    }  
   }  
    dönüş p2;  
}
```

112.2 Java Çözümü 2 - Özyinelemeli

```
genel ListNode reverseList(ListNode head) { if(head==null ||  
    head.next == null) dönüş kafası;  
  
    //ikinci düğümü al  
    ListNode saniye = head.next; //ilk  
    sonrakini boş olarak ayarla head.next =  
    null;  
  
    ListNode rest = reverseList(saniye);
```

112 Ters Bağlantılı Liste

ikinci.sonraki = kafa;

dinlenme [dönüşü](#) ;
}

113 Ters Bağlantılı Liste II

Bağlantılı bir listeyi m konumundan n konumuna ters çevirin. Yerinde ve tek geçişte yapın.

Örneğin: 1->2->3->4->5->NULL, m = 2 ve n = 4 verildiğinde, 1->4->3->2->5- döndür
>BOŞ.

113.1 Analiz

113.2 Java Çözümü

```
public ListNode reverseBetween(ListNode kafası, int m, int n)
{
    if(m==n) dönüş başlığı;

    ListNode prev = null;//track (m-1)inci düğüm
    Önce ListNode = new ListNode(0); //birincinin sonraki noktaları mth
    ListNode saniye = new ListNode(0); //saniyenin sonraki noktaları (n+1)th

    int ben=0;
    ListNode p = kafa; while(p!=
    =null){ i++; eğer(i==m-1){

        önceki = s;
    }

    eğer(i==m)
    { ilk.sonraki = p;
    }

    eğer(i==n)
    { saniye.sonraki = p.sonraki;
    p.sonraki = boş;
    }

    p= p.sonraki;

} if(ilk.sonraki == boş)
    dönüş kafası;

// ters listesi [m, n]
ListNode p1 = ilk.sonraki; ListNode
p2 = p1.sonraki;
```

113 Ters Bağlantılı Liste II

```
p1.sonraki = saniye.sonraki;

while(p1!=null && p2!=null){ ListNode t =
    p2.next; p2.sonraki = p1; p1 = p2; p2
    = t;

}

//önceki kısma bağlan if(prev!=null)
prev.next = p1; başka

dönüş p1;

dönüş kafası;
}
```

114 N'inci Düğümü Listenin Sonundan Kaldır

Bağlantılı bir liste verildiğinde, n'inci düğümü listenin sonundan kaldırın ve başını döndürün.

Örneğin, 1->2->3->4->5 ve n = 2 bağlantılı liste verildiğinde, sonuç 1->2->3->5'tir.

114.1 Java Çözümü 1 - Naif İki Geçiş

Once uzunluğu hesaplayın ve ardından n'inciyi baştan kaldırın.

```
public ListNode removeNthFromEnd(ListNode kafası, int n) { if(kafa == boş) boş
    döndürür;

    //listenin uzunluğunu al
    ListNode p = baş; int uzunluk
    = 0; while(p != null){ len++; p =
    p.sonraki;

    }

    //başlangıçtan ilk düğüm int'i
    kaldırırsanız = len-n+1;
    if(Başlangıçtan==1) dönüş
        baş.sonraki;

    //ilk olmayan düğümü kaldır p =
    baş; int ben=0; while(p!=null){ i+
    +; if(i==Başlangıç-1'den){ p.next
    = p.sonraki.sonraki;

    } p=p.sonraki;
}

dönüş kafası;
```

114.2 Java Çözümü 2 - Tek Geçiş

Hızlı ve yavaş işaretçiler kullanın. Hızlı işaretçi, yavaş işaretçinin n adım ilerisindedir. Hızlı sona ulaştığında, yavaş işaretçi hedef ögenin bir önceki öğesini gösterir.

```
public ListNode removeNthFromEnd(ListNode başında, int n) {
    if(head == null) null
        döndürür;

    ListNode hızlı = kafa;
    ListNode     = kahve;

    for(int i=0; i<n; i++){ hızlı =
        hızlı.sonraki;
    }

    //eğer ilk düğümü kaldır if(fast == null) {

        baş = baş.sonraki; dönüş
        kafası;
    }

    while(hızlı.sonraki != boş)
        { hızlı = hızlı.sonraki;
        yavaş = yavaş.sonraki;
    }

    yavaş.sonraki = yavaş.sonraki.sonraki;

    dönüş kafası;
}
```

115 Kuyrukları Kullanarak Yiğin Uygulama

Kuyrukları kullanarak bir yiğinin aşağıdaki işlemlerini gerçekleştirin. push(x) – x öğesini yiğinin üzerine itin. pop() – Yiğinin üstündeki öğeyi kaldırır. top() – En üstteki öğeyi alır. empty() – Yiğinin boş olup olmadığını döndürür.

Not: Java'da yalnızca standart kuyruk işlemlerine izin verilir, yani anket(), teklif(), peek(), size() ve isEmpty().

115.1 Analiz

Bu sorun iki sıra kullanılarak çözülebilir.

115.2 Java Çözümü

```
sınıf MyStack {  
    LinkedList<Tamsayı> sıra1 = new LinkedList<Tamsayı>();  
    LinkedList<Tamsayı> sıra2 = new LinkedList<Tamsayı>();  
  
    // x öğesini yiğine itin. public void  
    push(int x) { if(empty()){ tail1.offer(x); }  
        else{ if(queue1.size()>0)  
            { tail2.offer(x); int boyut =  
                kuyruk1.size(); while(size>0)  
                    { tail2.offer(queue1.poll());  
                        boyut--;  
                }  
            }  
        }  
    }  
}  
}
```

115 Kuyrukları Kullanarak Yiğin Uygulama

```
// Yiğinin en üstündeki elemanı kaldırır. genel boşluk
pop() { if(queue1.size()>0){queue1.poll(); }else
    if(queue2.size()>0){ tail2.poll(); }

}

// En üstteki elemanı al. public
int top() { if(queue1.size()>0)
{ dönüş kuyruğu1.peek(); }
    başka if(queue2.size()>0)
{ dönüş kuyruğu2.peek(); }

} 0 döndürür ;
}

// Yiğinin boş olup olmadığını döndürür. public
boolean empty() { dönüş kuyruğu1.isEmpty() &
    tail2.isEmpty();
}
```

116 Yiğinları Kullanarak Kuyruk Uygulama

Yiğinları kullanarak bir kuyruğun aşağıdaki işlemleri gerçekleştirebilirsiniz.

- push(x) - x öğesini kuyruğun arkasına itin. pop() - Öğeyi kuyruğun önünden kaldırır.
- peek() - Ön elemanı alın. empty() - Kuyruğun boş olup olmadığını döndürür.

116.1 Java Çözümü

```
sınıf MyQueue {  
  
    Yiğin<Tamsayı> temp = yeni Yiğin<Tamsayı>();  
    Yiğin<Tamsayı> değer = yeni Yiğin<Tamsayı>();  
  
    // x öğesini kuyruğun arkasına itin. genel geçersiz itme(int  
    x) { if(değer.isEmpty())( değer.push(x); }else{ while(!  
        value.isEmpty()){  
  
            temp.push(değer.pop());  
        }  
  
        değer.push(x);  
  
        while(!temp.isEmpty())  
            { değer.push(temp.pop());  
        }  
    }  
}  
  
// Elemani kuyruğun önünden kaldırır. genel boşluk pop() { value.pop();  
  
}  
  
// Ön elemani al. genel int gözetleme()  
{  
    dönüş değer.peek();  
}  
  
// Kuyruğun boş olup olmadığını döndürür.
```

116 Yiğinları Kullanarak Kuyruk Uygulama

```
genel boolean boş()
    { dönüş değeri.isEmpty();
}
}
```

117 Palindrome Bağlantılı Liste

Tek bağlantılı bir liste verildiğinde, bunun bir palindrom olup olmadığını belirleyin.

117.1 Java Çözümü 1

Ters sırada yeni bir liste oluşturabilir ve ardından her düğümü karşılaştırabiliriz. Zaman ve mekan $O(n)$ 'dır.

```
genel boolean isPalindrome(ListNode head) { if(head == null) true
    döndürür;

    ListNode p = kahve; ListNode
    önceki = yeni ListNode(head.val);

    while(p.sonraki != boş)
        { ListNode geçici = yeni ListNode(p.next.val); temp.sonraki =
        önceki; önceki = sıcaklık; p = p.sonraki;

    }

    ListNode p1 = kahve;
    ListNode p2 = önceki;

    while(p1!=null){ if(p1.val !=
        = p2.val) false döndürür;

    p1 = p1.sonraki; p2
        = p2.sonraki;
    }

    doğru dönüş;
}
```

117.2 Java Çözümü 2

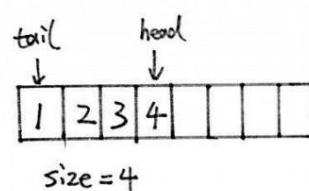
Listenin merkezini bulmak için hızlı ve yavaş bir işaretçi kullanabilir, ardından ikinci listeyi tersine çevirebilir ve iki alt listeyi karşılaştırabiliriz. Zaman $O(n)$ ve uzay $O(1)$ 'dir.

117 Palindrome Bağlantılı Liste

```
genel boolean isPalindrome(ListNode head) {  
  
    if(head == null || head.next==null) true yürütür;  
  
    // liste merkezini bul  
    ListNode hızlı = kafa;  
    ListNode     = kahve;  
  
    while(fast.next!=null && fast.next.next!=null){  
        hızlı = hızlı.sonraki.sonraki; yavaş =  
        yavaş.sonraki;  
    }  
  
    ListNode ikinci Başlık = yavaş.sonraki;  
    yavaş.sonraki = boş;  
  
    //ListNode p1 = SecondHead; ListNode p2 =  
    p1.sonraki;  
  
    while(p1!=null && p2!=null){ ListNode temp =  
        p2.next; p2.sonraki = p1; p1 = p2; p2  
        = sıcaklık;  
  
    }  
  
    SecondHead.next = boş;  
  
    //şimdi iki alt listeyi karşılaştırın ListNode p =  
    (p2==null?p1:p2); ListNode q = kafa; while(p!  
=null){ if(p.val != q.val) false döndürür;  
  
    p = p.ses; Q  
    = q.sonraki;  
  
    }  
  
    doğru dönüş;  
}
```

118 Java'da Dizi Kullanarak Kuyruk Gerçekleştirme

Aşağıdaki Java kodu, Java'da herhangi bir ekstra veri yapısı kullanmadan bir kuyruğun nasıl uygulanacağını gösterir. Bir dizi kullanarak bir kuyruk uygulayabiliriz.



```
java.lang.reflect.Array'i içe aktarın;
java.util.Arrays'i içe aktarın;

genel sınıf Kuyruk<E> {

    E[] dizi; int
    kafa = -1; int kuyruk
    = -1; int boyutu;

    public Queue(Class<E> c, int size) { E[] newInstance =
        (E[]) Array.newInstance(c, boyut); this.arr = yeniÖrnek; bu.size = 0;

    }

    boolean itme(E e) { if (boyut
        == dizi.uzunluk) false döndürür;

        kafa = (kafa + 1) % dizi.uzunluk; dizi[kafa] = e;
        boyut++;

        if(kuyruk == -1){ kuyruk =
            baş;
        }
    }
}
```

118 Java'da Dizi Kullanarak Kuyruk Gerçekleştirme

```
    doğru dönüş;
}

boole pop() { if (boyut
== 0) { false döndürür;

}

E sonuç = dizi[kuyruk];
dizi[kuyruk] = boş; boyut--;
kuyruk = (kuyruk+1)%diz.uzunluk;

eğer (boyut == 0) { kafa
= -1; kuyruk = -1;

}

doğru dönüş;
}

E peek()
{ if(size==0) null
    döndürür;

    dönüş dizisi[kuyruk];
}

public int size() { dönüş
this.size;
}

public String toString() { dönüş
Diziler.toString(this.arr);
}

public static void main(String[] args) { Queue<Integer> q =
new Queue<Integer>(Integer.class, 5); q.push(1); q.push(2); q.push(3); q.push(4);
q.push(5); q.pop(); q.push(6); System.out.println(q);

}

}
```

119 Bağlantılı Listedeki Düğümü Sil

Yalnızca o düğüme erişim verildiğinde, tek bağlantılı bir listedeki bir düğümü (kuyruk hariç) silmek için bir işlev yazın.

Bağlantılı listenin 1 ->2 ->3 ->4 olduğunu ve size değeri olan üçüncü düğümün verildiğini varsayırsak 3, işlevinizi çağırıldıkten sonra bağlantılı liste 1 ->2 ->4 olmalıdır .

119.1 Java Çözümü

```
genel geçersiz silmeNode(ListNode düğümü)
{ node.val = node.next.val; düğüm.sonraki =
  düğüm.sonraki.sonraki;
}
```

Bu problem çok mu kolay? Ya da ben yanlış yapıyorum.

Veri Akışından 120 Hareketli Ortalama

Bir tamsayı akışı ve bir pencere boyutu verildiğinde, kayan penceredeki tüm tamsayıların hareketli ortalamasını hesaplayın.

120.1 Java Çözümü

Bu sorun bir kuyruk kullanılarak çözülür.

```
genel sınıf HareketliOrtalama {  
  
    LinkedList<Tamsayı> kuyruğu; int  
    boyutu;  
  
    /** Veri yapınızı burada başlatıyoruz. */  
    public MovingAverage(int size) { this.queue =  
        new LinkedList<Integer>(); this.size =boyutu;  
  
    }  
  
    public double next(int val) { tail.offer(val);  
        if(queue.size()>this.size){ tail.poll();  
  
    }  
  
    int toplam=0; for(int i:  
        sıra) { toplam=toplam+i;  
    }  
  
    return (double)sum/queue.size();  
}  
}
```
