Algoritma Analizi Uygulama 3

Yıldız Teknik Üniversitesi

Problem 1: Dinamik Programlama

- A ve B takımları, içlerinden biri n galibiyet alana kadar maç yapacaklardır.
- A takımının bir maçı kazanma olasılığı her maç için p, kaybetme olasılığı ise 1-p'dir. Dolayısıyla beraberlik ihtimali mevcut değildir.
- A takımının seriyi kazanmak için i tane daha maç kazanması, B takımının da seriyi kazanmak için j tane maç kazanması gereken durumda A'nin seriyi kazanma olasılığı P(i,j)'dir.

Problem 1: Dinamik Programlama

- Eğer A takımı p ihtimalle bir maç kazanırsa, A takımının seriyi kazanması için gerekli maç sayısı i-1 iken B takımı hala j galibiyet elde etmelidir.
- Eğer A takımı 1-p ihtimalle bir maç kaybederse, seriyi kazanmak için A takımı hala i galibiyet, B takımı ise j-1 galibiyet elde etmelidir.
- $P(i,j) = p \times P(i-1,j) + (1-p) \times P(i,j-1) \; \forall \; i,j > 0$
- A takımının kazanacak maç sayısı sıfır olduğunda A takımı seriyi kazanmış demektir. Bu yüzden her i=0 ve j>0 durumunda olasılık 1 olacaktır.

  $P(0,j) = 1 \qquad \forall \; i = 0, j > 0$
- B takımının kazanacak maç sayısı sıfır olduğunda A takımı seriyi kaybetmiş demektir. Bu yüzden her j=0 ve i>0 durumunda olasılık 0 olacaktır.

  $P(i,0) = 0 \qquad \forall \; j = 0, i > 0$

Problem 1: Dinamik Programlama

- **Dinamik Programlama Tablosu**

i \ j	0	1	2	3	4
0		1	1	1	1
1	0	0,40	0,64	0,78	0,87
2	0	0,16	0,35	0,52	0,66
3	0	0,06	0,18	0,32	0,46
4	0	0,03	0,09	0,18	0,29

$$\boldsymbol{P(1,2) = 0,4 \times P(0,2) + 0,6 \times P(1,1)}$$

$$\boldsymbol{P(1,1) = 0,4 \times P(0,1) + 0,6 \times P(1,0)}$$

$$\boldsymbol{P(1,1) = 0,4 \times 1 + 0,6 \times 0 = 0,4}$$

$$\boldsymbol{P(1,2) = 0,4 \times 1 + 0,6 \times 0,4 = 0,64}$$

Problem 1: Dinamik Programlama

Algoritma WinSeries(n, p)
// n galibiyetin kazanılma olasılığını hesaplar
//Input: Kazanmak için gerekli n galibiyet sayısı
// ve takımın maçı kazanma olasılığı
//Output: Takımın seriyi kazanma olasılığı

```
q ← 1 − p
for j ← 1 to n do
        P[0, j] ← 1.0
for i ← 1 to n do
        P[i, 0] ← 0.0
        for j ← 1 to n do
                P[i, j] ← p ∗ P[i − 1, j] + q ∗ P[i, j − 1]
return P[n, n]
```

Problem 2:

- Sorgulanan bir cümlede yanlış yazılmış kelimeler varsa bu kelimelerin yerine doğru kelimeler öneren bir sistem tasarlanacaktır.
- Örnek : Kullanıcı: It is coold
  Bilgisayar: “coold” is not in the dictionary.
  Did you mean: “cool “ or “cold”
  Kullanıcı: cold
  Bilgisayar: It is cold

Problem 2:

1. Ödevde kullanılacak sözlüğü hazırlamak için smalldictionary.txt dosyasındaki sözlüğe ait kelimeler sözlük tablosuna (hash tablosu) yerleştirilir. Bu işlem bir defa yapılacaktır.

2. Verilen cümledeki her kelime sözlük tablosunda aranır.
   Eğer kelime sözlük tablosunda
   a. varsa kelime doğrudur. O kelime için işlem tamamlanır.
   b. yoksa, hatalı kelime tablosunda aranır.

Problem 2:

```
searchHashTable(hashTable,word,Mh1,Mh2 ):
        word←lowercase(word)
        key←horner(word)
        index←hashFunction1(key,Mh1)
        while hashTable [index]!=NULL and strcmp(hashTable [index].value,word)!=0 and rotation=Mh1:
                stepSize← hashFunction2(key,Mh2 )
                index←hashFunction(h1,stepSize,rotation,Mh1)
                rotation++
        if hashTable [index]==NULL: # bulunamadı!
                return 0
        else if rotation>=Mh1:
                return 0
        else if strcmp(hashTable [index].value,word)==0 #bulundu
                return 1
```

Problem 2:

```
suggestWord(dictionary_hashTable, wrong_hashTable, word, Mh1, Mh2):
        if(searchHashTable(dictionary_hashTable, word, Mh1, Mh2) == 1)
                kelime  sözlük tablosunda bulundu
                öneri işlemi yapma
                return
        else
                if(searchHashTable(wrong_hashTable, word, Mh1, Mh2) == 0 )
                        if(get_words_with_LevenshteinEditDistance(dictionary_hashTable, word, Mh1, Mh2, 1) != NULL)
                                kullanıcıya uzaklığı 1 olan kelimeleri göster
                                if (kullanıcı doğru kelimeyi uzaklığı 1 olan kelimelerden seçim yaparsa)
                                        hatalı tablosuna ekle
                                        return
                        if(get_words_with_LevenshteinEditDistance(dictionary_hashTable, word, Mh1, Mh2, 2) != NULL)
                                kullanıcıya uzaklığı 2 olan kelimeleri göster
                                if (kullanıcı doğru kelimeyi uzaklığı 2 olan kelimelerden seçim yaparsa)
                                        hatalı tablosuna ekle
                                        return
                                else
                                        doğru kelimeler tablosunda 2 ve 2 den küçük mesafede bir kelime bulunamadı.
                                        return
                else
                        kelime hatalı kelimler hash tablosunda mevcut.
                        Ekleme işlemi yapma
                        Doğru kelimeyi tablodan oku ve print et
                        return
```

Problem 2:

```
get_words_with_LevenshteinEditDistance(hashTable, word, Mh1, Mh2, k):
        kelimeler[] = NULL
        for index ← 0 to n do
                uzaklık = LevenshteinEditDistance(hashTable[index].value, word)
                if(uzaklık<=k)
                        kelimelerin tutulduğu diziye kelimeyi ekle
        return kelimeler[]
```

Problem 2:

```
LevenshteinEditDistance(word1, word2):
        n ← word1 in uzunluğu
        m ← word2 nin uzunluğu
        if n = 0
                return m
        if m = 0
                return n
        matrix[m][n] ← 0..m satırlık ve 0..n sütunluk bir matris oluştur
        for i←0 to n do
                matris[0][i] ←i
        for j←0 to m do
                matris[j][0] ←j
        Examine each character of word1 (i from 1 to n).
        Examine each character of word2 (j from 1 to m).
        if (word1[i] equals word2[j])
                the cost is 0
        else
                the cost is 1
        Set cell matris[i][j] of the matrix equal to the minimum of:
        a. The cell immediately above plus 1: d[i-1,j] + 1.
        b. The cell immediately to the left plus 1: d[i,j-1] + 1.
        c. The cell diagonally above and to the left plus the cost: d[i-1,j-1] + cost.
```

Problem 2:

2. Levenshtein Edit Distance hesaplama

		H	**E**	**L**	**L**	**O**
	0	1	2	3	4	5
H	1					
E	2					
L	3					
L	4					

Problem 2:

2. Levenshtein Edit Distance hesaplama

		H	**E**	**L**	**L**	**O**
	0	1	2	3	4	5
H	1	0	1	2	3	4
E	2	1	0	1	2	3
L	3	2	1	0	1	2
L	4	3	2	1	0	1

Problem 2:

Bonus

2. Levenshtein Edit Distance hesaplama

		F	R	I	E	D
	0	1	2	3	4	5
A	1					
R	2					
T	3					
I	4					
S	5					
T	6					

Problem 2:

Bonus

2. Levenshtein Edit Distance hesaplama

		F	R	I	E	D
	0	1	2	3	4	5
A	1	1	2	3	4	5
R	2	2	1	2	3	4
T	3	3	2	2	3	4
I	4	4	3	2	3	4
S	5	5	4	3	3	4
T	6					

Teşekkürler

Yıldız Teknik Üniversitesi