Problem Tanımı:

Elimizde kullanıcı tarafından 2 ifade verilir. İfadelerin uzunlukları farklı olabilir. Bu ifadeler içinde en uzun ortak ifade bulunmaya çalışılır.

Problem Çözümü:

Dinamik programlama mantığı ile ele alınır problem. Yani elimdeki problemi daha küçük alt problemlere bölüp öyle ele alınır. Sonra küçük problemlerden asıl büyük problemi çözmüş olunur. Bunu yapmak için de elimizde LCS matrisi dediğimiz matris (Şekil-1) var. Küçük problemlerden büyük probleme nasıl ulaşılırız onu inceleyelim. Birinci ifade “ABCDZB” ve ikinci ifade “BACEDAB” olsun. En alt problemden başlarayarak nasıl ele alınır inceleyelim:

- En alt problem elimizde hiç ifade olmaması yani "" ve "" ifadeleri olsun yani ifade olmasın o zaman olmayan bir ifadelerde ortak ifade olamaz. Bu en küçük problemimiz.
- Ondan sonra misal "" ve "A" ifadeleri olsun. "A" ifadesinden "" ifadesini elde etmek olamaz ondan dolayı yine ortak ifade olamaz. Elde edilen sonuç elimde ifade yoksa ortaklık olamaz. Ondan dolayı ilk satırın tamamı 0 olur. Aynısı ilk sütun için de geçerli.
- Başka bir Alt problem ise “B" ve “A" ifadeleride ortaklık var mı. Yok o zaman başka bir alt probleme geçilmesi gerek.
- “B" ve “AB" ifadeleri de alt problem ama öncekinin bir üst problemi. İki ifadenin son elemanları aynı olduğuna göre bir önceki çözüme bakarız. Ne kadar ortak ifade uzunluğu var diye. Bakılınca 0 tane ortak nokta vardı. O zaman bunun üzerine 1 eklenir. Yani toplam ortak nokta 1 oldu.
- Bir sonraki alt problem "A" ve “ABC" ifadeleri ele alınır. İki ifadenin son karakterlerine bakılır. Aynı olmadığı için alt problemlere bakılır. Hangi alt problem ele almamız gerek. Amaç en uzun ifade olduğu için hangi alt problemde daha uzun bulunmuşsa onu değerini ilgili indekse eklenmesi gerek. Şekil-1’deki tabloya bakınca “B” ve “AB” ifadelerindeki ortak ifade değeri 1 ve “” ve “AB” ifadelerindeki ortak değer sıfır. “B” ve “AB” ifadelerindeki ortak ifade değeri daha büyük olduğu için bir bilgisi yazılır.

Bu adımlar tablo doldurulana kadar bu mantıkla gidilir. Ek olarak matris satır satır doldurulur. Yani ikinci ifadenin alt ifadesi ile tüm birinci alt ifadeleri ile matris doldurulur. Sonra ikinci ifadenin bir sonraki alt ifadesi ile birinci harfin tüm alt ifadeleriyle hesaplanır. Matrisin sonunda ise en uzun ifadenin uzunluğu yazılır.

		A	B	C	D	Z	B
	0	0	0	0	0	0	0
B	0	0	1	1	1	1	1
A	0	1	1	1	1	1	1
C	0	1	1	2	2	2	2
E	0	1	1	2	2	2	2
D	0	1	1	2	3	3	3
A	0	1	1	2	3	3	3
B	0	1	2	2	3	3	4

Şekil-1

(Alt problemlerden istenilen probleme ulaşılınca matrisin son hali)

En uzun ifadenin uzunluğu var ancak ifade/ifadeler yok. Elimizdeki matris ile en uzun ifade/ifadeler bulunabilir. Matrisin en sağ altında uzunluk değeri vardı. Oradan başlayarak ifadeleri bulabiliriz. Elimizde bir de aynı karakter ise 1 olarak işaretlenmiş matris(**tableSelect**) de vardır.

1. İlk önce kontrol edilir. Eğer 1 ise demek ki aynı karakter o zaman değişken içinde saklanılır. Satır (**k**) ve sütun indeks (**l**) değerleri bir azaltılıp fonksiyon çağrısı(**Şekil-2**) yapılır.
2. Karakterler aynı değilse o zaman LCS matrisine bakılır. Üst ve sol indeksleri karşılaştırılır hangisi daha büyükse oraya gidilir. Eğer sol değer daha yüksek ise sütun indeksi(**l**) bir azaltılıp fonksiyon çağrısı yapılır. Eğer üst satırdaki değer daha yüksekse satır indeksi(**k**) bir azaltılıp tekrardan fonksiyon çağrısı yapılır.
3. Peki iki değer de aynı ise o zaman iki taraftan da gidilmeli. Potansiyel olarak birden fazla ifade olabilir. Hem sol ve üst taraf için fonksiyon çağrısı yapılır.

void findSecret(**char ***string2, **char ***result, int ****tableSelect , **int** **table , **int** k, int l, **int** index, **char** ****results)
k : LCS matrisi için satır indeksi, **l** : LCS matrisi için sütun indeksi
table : LCS matris, **tableSelect**: Harf sekansı
results : ifadelerin saklanıldığı matris, **result** : adım adım yapılırken karakterlerin saklanıldığı değişken
Şekil-2

Karşılaşılan Problemler:

Rekursif şekilde yapılırken bazen kesişen yollar oluyordu. Ondan dolayı aynı ifade birden fazla kere yazdırılıyordu. Ondan dolayı Şekil-2'deki **findSecret** fonksiyonunda **results** matrisine ifade ekleme yapıldı ve eğer daha önce o ifade varsa ekleme yapılmasına izin verilmedi. Ondan dolayı tekrar etme probleminden kurtulmuş olundu.
Bir de birden fazla ifade olabileceği için rekursif bir yapıda yapmam gerektiğini düşündüm. İki farklı yol olabileceği için iteratif çözüm daha zorlayıcı olur diye düşünüldü.

Algoritma Karmaşıklığı:

1. ifadenin uzunluğu: N
2. ifadenin uzunluğu: M
Ortak en uzun ifade/ifadelerin uzunluğu: L
LCS matrisini doldurmak için normalde Brute Force yaklaşım ile 2^N zaman karmaşıklığı olurdu. Dinamik programlama sayesinde zaman karmaşıklığımız O(N*M) olur. Alt problemlerden ana probleme ulaşıldığı için tüm ifadelerin alt ifadelerini gezmiş olunur.
En uzun ifadeyi bulurken rekursif şekilde ele alınmıştı. En iyi ihtimal elimdeki 2 ifade de eşit olursa karmaşıklığım O(L) oluyor.
En kötü ihtimalle ise aynı ifadenin tersten yazılmış olması. Misal "ABCD" ve "DCBA" ifadesinin zaman ikisinin de uzunluğu aynı olduğu için N diyelim o zaman karmaşıklığı O(N^2) olur. Tğm matrisi gezmesi gerekiyor ortak ifadeleri bulmak için.

Senaryolar:

(Not : Uzun olmasın diye sadece son çıktılar eklendi)

```
  0 0 0 0 0 0 0
B 0 0 1 1 1 1 1
A 0 1 1 1 1 1 1
C 0 1 1 2 2 2 2
E 0 1 1 2 2 2 2
D 0 1 1 2 3 3 3
A 0 1 1 2 3 3 3
B 0 1 2 2 3 3 4

S: SoldaN kalitim alinmis.
U: Ustten kalitim alinmis.
US: Ustten ya da soldan kalitim alinmis.
C: Caprazdan alinmis.
    • A  B  C  D  Z  B
    _  _  _  _  _  _  _
B   _  US C  S  S  S  C
A   _  C  US US US US US
C   _  U  US C  S  S  S
E   _  U  US U  US US US
D   _  U  US U  C  S  S
A   _  C  US U  U  US US
B   _  U  C  US U  US C

En uzun ifadenin boyutu: 4
BCDB
ACDB
```

Senaryo 2

```
13. satir sonu matrisler
    M A E R B P H C A P P B A
  0 0 0 0 0 0 0 0 0 0 0 0 0 0
A 0 0 1 1 1 1 1 1 1 1 1 1 1 1
M 0 1 1 1 1 1 1 1 1 1 1 1 1 1
R 0 1 1 1 2 2 2 2 2 2 2 2 2 2
R 0 1 1 1 2 2 2 2 2 2 2 2 2 2
E 0 1 1 2 2 2 2 2 2 2 2 2 2 2
R 0 1 1 2 3 3 3 3 3 3 3 3 3 3
C 0 1 1 2 3 3 3 3 4 4 4 4 4 4
H 0 1 1 2 3 3 3 4 4 4 4 4 4 4
A 0 1 2 2 3 3 3 4 4 5 5 5 5 5
Z 0 1 2 2 3 3 3 4 4 5 5 5 5 5
B 0 1 2 2 3 4 4 4 4 5 5 5 6 6
Z 0 1 2 2 3 4 4 4 4 5 5 5 6 6
A 0 1 2 2 3 4 4 4 4 5 5 5 6 7
```

```
S: SoldaN kalitim alinmis.
U: Ustten kalitim alinmis.
US: Ustten ya da soldan kalitim alinmis.
C: Caprazdan alinmis.
      M  A  E  R  B  P  H  C  A  P  P  B  A
   _  _  _  _  _  _  _  _  _  _  _  _  _  _
A  _  US C  S  S  S  S  S  S  C  S  S  S  C
M  _  C  US US US US US US US US US US US US
R  _  U  US US C  S  S  S  S  S  S  S  S  S
R  _  U  US US C  US US US US US US US US US
E  _  U  US C  US US US US US US US US US US
R  _  U  US U  C  S  S  S  S  S  S  S  S  S
C  _  U  US U  U  US US US C  S  S  S  S  S
H  _  U  US U  U  US US C  US US US US US US
A  _  U  C  US U  US US U  US C  S  S  S  C
Z  _  U  U  US U  US US U  US U  US US US US
B  _  U  U  US U  C  S  US US U  US US C  S
Z  _  U  U  US U  U  US US US U  US US U  US
A  _  U  C  US U  U  US US US C  US US U  C

En uzun ifadenin boyutu: 7
AERCABA
MERCABA
AERHABA
MERHABA
```