Lütfullah Türker
141044050

CSE – 321 HW 4

Q1)

Örnek liste ilk elemanı 0 olacak şekilde oluşturuldu. Boş bir optimal listesi oluşturulup fillOptimalList fonksiyonuna oluşturulmuş 2 liste gönderildi.

fillOptimalList fonksiyonunun başında opt listesinin ilk elemanı soruda belirtildiği gibi 0 olarak atandı. Bu base case'dir. Fonksiyon bittiğinde opt[i] , i. otele ulaşmak için olabilecek minimum toplam penalty'yi verir. J'den i'ye giderkenki günlük cost (200 - (distanceArr[i] - distanceArr[j]))^2 dir.

Bu formüllerden yola çıkarak i'ye ulaşmak için oluşacak en düşük penalty aşağıdaki gibi elde edilebilir.

opt[j] + (200 - (distanceArr[i] - distanceArr[j]))^2 formülü i değerine kadar olan tüm j'ler için uygulandığında sonuçlar arasından minimum olan sonuç bize opt[i] değerini verir. Bu şekilde tüm i leri gezersek opt listesini doldurmuş oluruz.

N – hotel listesinin size I olmak üzere,

N adet subproblem'imimiz var. ve her biri i kere dönen bir döngüye sahip yani O(i)

Yani i=1'den N'ye kadar olan i'lerin toplamı = (n*(n+1))/2 olduğundan dolayı Complexity **O(n^2)** olur.

Lütfullah Türker
141044050

Q2)

Sorudan anladığım kadarıyla dict fonksiyonunun yazılı olduğunu ve O(1) de çalıştığını varsayarak yapmamız isteniyor. Fakat test edebilmek için bir dict ve dict fonksiyonu yazdım. Test stringi olarak "therewasbestoftheturkey" belirledim ve bu kelimeler dahil birkaç fazla kelimeyi de dictionary'me ekledim.

Bu test stringinde the ve there kelimelerini özellikle test ettim çünkü the kelimesi there kelimesi içinde yer alır. Algoritmamı bu ayrımı, sonraki kelimenin anlamlı olup olmamasına göre algılayacak şekilde tasarladım. The'dan sonra dictionary de olan bir kelime bulunmazsa geri dönüp daha uzun bir kelime arar yani there kelimesini alır ve ardından anlamlı bir kelime daha gelirse sonuca doğru kelimeyi ekler. Geri dönme işlemini yapabilmesi için bir flag tuttum ve gerekli durumda döngüyü flag e geri alıp tekrar çalıştırdım.

Cümlenin tüm kelimelerinin dictionary de olması ve verdiğim örnekteki gibi içiçe kelimelerin olmaması durumunda complexity **O(n)** olur. En kötü durumda ise **O(n^2)** olur. Fakat bu durumun olma ihtimali çok çok düşüktür.

Q3)

Bu problem için yazılacak algoritmanın merge sort algoritmasının 2d arrayler için olan versiyonu gibi düşündüm. K adet N boyutunda array olması demek [k][n] boyutunda bir 2D matrix array olması demektir. Merge sort da olduğu gibi bir merge isimli helper fonksiyon yazdım. Ve Recursive olarak arrayleri merge yapacak bir mergeAndSort fonksiyonu yazdım. mergeAndSort içinde merge çağrılarak arrayler sıralanarak birleştiriliyor ve mergeAndSort, matrix tamamlanana kadar kendini recursive olarak çağırıyor. En sonunda tamamen sıralanmış bir 1D array return edilir ve ekrana basılır. Complexity ise **O(k*n)** olarak elde edilir.

Alternatif algoritma : bir fonksiyonda k*n boyutunda 1D array oluşturulur ve for döngüleriyle matrix olan 2D array in tüm elemanları bu array e aktarılır. Ve oluşan 1D array mergeSort (divide and conquer) fonksiyonuna gönderilerek sort yapılır. **O(k*n)**

Lütfullah Türker
141044050

Q4)

Q5)

Bu problemin çözümünde graf kullanmayı uygun gördüm. Constraint lerin input unun tipi konusunda bir şey belirtilmediği için string array i olarak yapmaya uygun gördüm. Örneğin ['a=b','b=c','c!a'] şeklinde. Eşitsizlik durumu için = yerine ! konmalı.

Eşitlik olan denklemlerde eşitliğin iki tarafı arasında bir undirected edge tanımladım. Yani grafda connected olan vertex'ler birbirine eşit olacaktır.

Eşitsizlik durumunda ise eşitsizliğin 2 tarafından biri vertex ler arasında yok ise zaten eşitsizlik sağlanabilir. Çünkü vertex lerde olmayan bir variable herhangi bir değer alabilir.

Her ikisi de vertexlerde var ise de 1. Yi root olarak Kabul edecek şekilde DFS yaparak 2. Yi arar. Yani 1. Nin bağlı olduğu edge leri gezerek 2. Yi arar. Bulur ise birbirine bağlı olduklarından dolayı eşittirler ve bu sistem sağlanmaz. Bulunmaz ise 1. Ye eşit olan vertex ler arasında 2. Olmadığından eşitsizlik doğrudur.

Bu şekilde tüm sistem sağlanırsa constraints can be satisfied printi ekrana basılır. Aksi halde ise sağlanmadığı belirtilir.

DFS'in complexity'si **O(V+E)**'dir. V=Vertex sayısı yani n E=Edge sayısı yani m

constraintsControl fonksiyonunda bir döngü içinde tüm constraintler geziliyor. Yani bu gezme işlemi **O(m)**. DFS bu döngünün içinde çağrıldığı için complexity'lerinin çarpılması gerekir.

DFS sadece eşitsizlik olduğu ve eşitsizliğin 2 tarafının da vertexlerde olduğu durumda yapılıyor. Bu yüzden çok az sayıda yapılsa da en kötü durum **O(m*(m+n)) = O(m^2 + m*n)** olur. Fakat bu durum çok sık olmaz.