Seyfullah Bozerikli
121064028

CSE321 Homework 05

1) a) Greedy algoritmalar basit ve açıktır. Yaklaşımları, ileride bu yaklaşımların yaratacağı etkiyi düşünmeksizin eldeki bilgilere dayanarak kararlar almalarına bağlı olarak kısa mesafelidirler. Bu algoritmaları oluşturmak kolaydır ve oldukça efektiftirler. Çoğu problem greedy algoritmalar ile doğru çözülemez. Greedy algoritmalar optimizasyon problemlerinin çözümünde kullanılır.

b) Birçok problem için Greedy algoritmalar çoğunlukla (ancak her zaman değil) küresel olarak en uygun çözümü bulamaz çünkü genelde tüm veriler üzerinde kapsamlı bir çalışma yapmıyorlar. Belirli seçimleri çok erken vakitte bulabilirler bu da en iyi genel çözümü daha sonra bulmalarını engeller. Örneğin, grafik renklendirme problemi ve tüm NP-complete problemleri için bilinen tüm Greedy algoritmalar en uygun çözümleri bulamazlar. Bunun yanında kullanışlılardır. çünkü düşünmesi kolay ve çoğu zaman optimuma yakın sonuçlar verirler.

2) a) Doğru. (Aksi durumda, Kruskal's algoritması geçersiz olur)

b) Yanlış. Üç köşeli ve üç kenarda aynı ağırlıktaki tam bir graph buna karşı bir örnek olabilir.

c) Doğru

d) Yanlış

3) a) En basit ve mantıklı yöntem bütün edge'lerin ağırlıklarını 1'e atamaktır.

b) Depth first search tree elde etmek için depth first search yada breadth first search uygulamak basittir ve bunların bitişik listeleriyle daha hızlı temsil edilen seyrek graflar için daha hızlıdır.

8) Algorithm myChange(D[1...m], n)

```
for i=1 to m do
    C[i] = ⌊n / D[i]⌋
    n = n mod D[i]
if n=0
    return C
else
    return "nothing"
```

Algoritmanın efficiency si $\Theta(m)$ dir. Burada integer bölmenin ne kadar büyük olursa olsun constant time sürdüğünü varsayıyoruz. Eğer algoritmayı durdurursak kalan miktar 0 olur ve time efficiency $O(m)$ olur.

7) a) Bütün matris tipleri için şu operasyonlar n kez uygulanır. İşaretlenmiş satırlardaki ve cost matrisinin sütunlarındaki en küçük öğe seçilir ve ardında satırı ve sütun işaretlenir.

Satır sıra versiyonu için şu operasyonlar yapılır. İlk satırdan başlayıp son satır ile biten satırda daha önce işaretli bir sütunda bulunmayan en küçük öge seçilir. Bu öge seçildikten sonra, aynı sütundan başka bir öge seçimini engellemek için o sütun işaretlenir.

b) Yukarıdaki her iki çözümde optimum çözümü getirmez.

$C = \begin{bmatrix} 1 & 2 \\ 2 & >> \end{bmatrix}$ buna zıt bir problemdir.

6) İşleri yapılma sürelerine bakarak artan sıraya göre sıralanır ve bu sıraya göre uygulanır.

Bu greedy algoritması herzaman optimum çözümü getirir. Aslında $i_1, i_2 \dots i_n$ işlerinin herhangi bir sırası ile sistemdeki toplam süre şu formül ile hesaplanır.

$$t_{i_1} + (t_{i_1} + t_{i_2}) + \dots + (t_{i_1} + t_{i_2} + \dots + t_{i_n}) = n \cdot t_{i_1} + (n-1) t_{i_2} + \dots + t_{i_n}$$

Böylece ağırlıklar $t_1, t_2, \dots t_n$ olan sayılar ile elimizdeki $n, n-1, \dots 1$ sayılarının toplamı bir sırada sıralanır. Böyle bir toplamı en aza indirmek için daha küçük sayılar atamak zorundayız. Yani işler yapılma sıralarına bakarak artan sırada sıralanarak ele alınmalıdır.

Eğer işler $t_{i_k} > t_{i_{k+1}}$ için $i_1, i_2, \ldots i_n$ sırasında yapılırsa sistem toplam zamanı azalan sırada olabilir. Bu nedenle hiçbir sıralama optimal çözüm olamaz. Diğer bir sıralama düşünürsek bu sıralama k ve k+1 işlerinin yerini değiştirsin. Burada sistemdeki zaman bu iki iş hariç aynı kalacaktır. Bu nedenle yeni sıralama için sistemdeki toplam süre ile yer değiştirme öncesi toplam süre arasındaki fark olur.

Bu fark;

$$\left[\left(\sum_{j=1}^{k-1} t_{i_j} + t_{i_{k+1}}\right) + \left(\sum_{j=1}^{k-1} t_{i_j} + t_{i_{k+1}} + t_{i_k}\right)\right] - \left[\left(\sum_{j=1}^{k-1} t_{i_j} + t_{i_k}\right) + \left(\sum_{j=1}^{k-1} t_{i_j} + t_{i_k} + t_{i_{k+1}}\right)\right]$$

$$= t_{i_{k+1}} - t_{i_k} < 0.$$

3) Değer ve ağırlıkların sıralı olduğunu varsayarsak

```
def Knapsack(v, w, W)
    load = 0
    i = 1
    while load < W && i <= n:
        if w[i] <= W - load
            # took all item:
        else
            # took (W - load) / w[i] item:
        # add weight of what taken to load.
        i = i + 1
    return load.
```

Time complexity $= O(n) = \Theta(n) = \Omega(n) = n$

4)

```
algorithm Greedytsp(V, E, hom)
    x[1] = hom
    I = 1
    while I < N - 1
        #select (u,v) min weight
        I = I + 1
    return A.
```

5) Algorithm greedycolor():

```
result = []
u = 1
for u in rang (1, v)
    result[u] = -1
available = []
for i in rang (0, v)
    available[i] = false
for u in rage (1, v)
    for i in rage (begin, end)
        if result[i] != -1
            available [result[i]] = true

    for i in rag(0, v)
        if available[i] == false
            break
    for k in range(begin, end)
        if result[k] != -1
            available [result[k]] = false.
```