Laporan Tugas Eksperimen 2

Nama : Abdul Rafi

NPM : 2106630113

Kode Asdos : 3

Dengan ini saya menyatakan bahwa TE ini adalah hasil pekerjaan saya sendiri

1. Deskripsi Algoritma
   - Greedy Weighted Set Cover
     Pada algoritma ini akan digunakan approach greedy untuk mencari weighted set cover. Ide utama dari algoritma ini cukup sederhana. Pada setiap iterasi kita akan mencari subset yang paling menguntungkan kita untuk diambil. Untuk menentukan subset mana yang paling menguntungkan. Selanjutnya rumus berikut akan disebut dengan "nilai keuntungan":

     $$\frac{X-s}{costX}$$

     Untuk X merupakan subset ditinjau sekarang dan s subset yang sudah dimiliki. X – s merepresentasikan banyaknya element yang terdapat pada X namun, tidak terdapat pada s. Pada setiap iterasi akan dipilih subset dengan nilai keuntungan paling besar. Nilai keuntungan ini merepresentasikan ration antara satu buah elemen dengan harga. Maka nilai keuntungan lebih besar memiliki makna, harga yang dibayarkan untuk satu buah elemen lebih kecil.

     Berikut adalah contoh cara algoritma ini bekerja. Misal terdapat 5 buah element dengan subset berupa ([1,3], [2], [1,2,5], [3,5], [4], [2,3]) yang masing-masing subset memiliki cost [11, 4, 9, 12, 5, 9].

     1. Iterasi Pertama
        Pada iterasi pertama subset s = [] (masih kosong). Berikut untuk nilai keuntungan dari setiap subset.

Subset	Cost	Nilai Keuntungan
[1, 3]	11	0.18
[2]	4	0.25
[1, 2, 5]	9	0.33
[3, 5]	12	0.16
[4]	5	0.2
[2, 3]	9	0.22

        Berdasarkan table diatas akan dipilih subset ketiga. Sehingga s = [1, 2, 5]

     2. Iterasi Kedua
        Pada iterasi kedua subset s = [1, 2, 5]. Berikut untuk nilai keuntungan dari setiap subset.

Subset	Cost	Nilai Keuntungan
[1, 3]	11	0.09
[2]	4	0
[1, 2, 5]	9	0
[3, 5]	12	0.08
[4]	5	0.2
[2, 3]	9	0.11

        Berdasarkan table diatas akan dipilih subset keempat. Sehingga s = [1, 2, 4, 5]

     3. Iterasi Ketiga
        Pada iterasi kedua subset s = [1, 2, 4, 5]. Berikut untuk nilai keuntungan dari setiap subset.

Subset	Cost	Nilai Keuntungan
[1, 3]	11	0.09
[2]	4	0
[1, 2, 5]	9	0
[3, 5]	12	0.08
[4]	5	0
[2, 3]	9	0.11

Berdasarkan table diatas akan dipilih subset kelima. Sehingga s = [1, 2, 3, 4, 5]

Pada akhir iterasi ketiga kita sudah mendapatkan subset yang berisi semua element dari satu sampai lima. Dengan demikian harga minimum yang didapatkan adalah 9 + 5 + 9 = 23.

- Branch&Bound Weighted Set Cover

  Untuk menyelesaikan permasalahan weighted set cover menggunakan metode branch&bound umumnya memliki cara yang mirip dengan brute force. Namun, pada metode branch&bound kita akan melakukan pruning dengan cara melakukan pengecekan apakah jika kita mengambil subset ini harga yang didapatkan akan lebih murah dibanding harga yang sudah didapat sebelumnya, apabila tidak maka tidak akan dilakukan penelusuran lebih lanjut. Kita juga dapat melakukan pruning dengan memastikan jika kita terus mengambil subset maka akan didapat subset yang mengcover semua element. Jika, tidak kita juga tidak perlu melakukan penelusuran lebih lanjut.

  Berikut adalah contoh cara algoritma ini bekerja. Misal terdapat 5 buah element dengan subset berupa ([1,3],[2,3],[3,5],[4],[1,2,5]) yang masing-masing subset memiliki cost [35, 28, 12, 5, 9]. Pada contoh berikut akan didemonstrasikan dengan prioritas tidak mengembil subset terlebih dahulu baru didahulukan mengambil subset.

  Awalnya akan dicek apakah kita dapat tidak memilih subset pertama. Ternyata masih memungkinkan untuk mendapatkan subset yang mengcover semua elemen. Maka kita lanjut ke subset kedua. Untuk subset kedua masih memungkinkan untuk mendapatkan subset yang mengcover semua elemen apabila subset kedua tidak dipilih. Maka kita lanjut ke subset ketiga.

  Untuk subset ketiga tidak memungkinkan untuk mendapatkan subset yang mengcover semua elemen apabila subset ketiga tidak dipilih. Dengan demikian dipilih subset ketiga dengan harga 12. Dilanjutkan ke subset keempat.

  Untuk subset keempat tidak memungkinkan untuk mendapatkan subset yang mengcover semua elemen apabila subset keempat tidak dipilih. Dengan demikian dipilih subset keempat dengan total harga 17. Dilanjutkan ke subset kelima.

  Untuk subset kelima tidak memungkinkan untuk mendapatkan subset yang mengcover semua elemen apabila subset kelima tidak dipilih. Dengan demikian dipilih subset kelima dengan total harga 26. Setelah subset kelima diambil maka semua element sudah tercover dengan harga 26. Setelah itu akan dicari konfigurasi lain seperti mengambil subset kedua. Namun, tidak memberikan harga yang lebih murah sehingga akan diskip oleh branch&bound.

2. Hasil Eksperimen dan analisis
   Data berikut diambil dengan melakukan 5 kali perulangan pembuatan dataset:

Dataset	Greedy Time	Greedy Memory	Branch&Bound Time	Branch&Bound Memory
20 element dengan 20 subset	0.000001 ms	57.826 MiB	0.0098 ms	57.865 MiB
200 element dengan 60 subset	0.002 ms	57.882 MiB	0.3387 ms	58.673 MiB
2000 element dengan 250 subset	0.11 ms	58.462 MiB	0.8267 ms	78.930 MiB

   Berdasarkan data diatas dapat dilihat bahwa approach greedy menghasilkan run time dan memory yang lebih cepat dibandingkan menggunakan Branch&Bound. Hal ini dikarenakan algoritma greedy memiliki kompleksitas $O(N^2M)$ dan Branch&Bound memiliki kompleksitas $O(2^NM)$ (dimana N merupakan subset dan M ukuran dari subset). Namun, pada eksperimen didapatkan bahwa approach greedy tersebut tidak selalu mendapatkan solusi yang optimal. Sehingga kedua cara tersebut memiliki keuntungan dan kerugian masing-masing.

3. Kesimpulan
   Greedy approach memiliki running time yang lebih cepat dibandingan Branch&Bound. Terlihat dari kompleksitas Branch&Bound yaitu $O(2^NM)$ dan Greedy approach dengan kompleksitas $O(N^2M)$. Meskipun perlu dicatat bahwa Branch&Bound mungkin berjalan lebih cepat karena dilakukan pruning.

   Greedy approach memiliki kelemahan yang cukup mencolok yaitu, tidak mampu untuk selalu memberi solusi yang optimal dibanding Branch&Bound yang selalu mampu memberikan solusi optimal. Sehingga kedua approach diatas memerlukan pertimbangan masing-masin untuk digunakan.

4. Psudocode

   Link Repository : https://github.com/TakeMeGH/DAA-TE2

   - Branch&Bound Weighted set cover

```python
import time
from GenerateTC import generate_dataset
import sys
sys.setrecursionlimit(10**9)


def BB(nodeCount, nodeSets : list, setsPrice : list):
    minCost = sum(setsPrice)
    resultSubset =  [x for x in range(len(nodeSets))]


    def findMin(index : int, currentCost : int, choosenIndex : list, currentSets : set):
        nonlocal minCost, resultSubset
        if index == len(nodeSets):
            if(len(currentSets) == nodeCount):
                if(minCost > currentCost):
                    minCost = currentCost
                    resultSubset = choosenIndex
        else:
            if(currentCost > minCost):
                return

            tempSets = currentSets.copy()
            for i in range(index + 1, len(nodeSets)):
                tempSets = tempSets.union(set(nodeSets[i]))
            if len(tempSets) == nodeCount:
                findMin(index + 1, currentCost, choosenIndex.copy(), currentSets.copy())

            tempSets = currentSets.copy()
            for i in range(index, len(nodeSets)):
                tempSets = tempSets.union(set(nodeSets[i]))

            if(len(tempSets) == nodeCount):
                copyChoosenIndex = choosenIndex.copy()
                copyChoosenIndex.append(index)

                copyCurrentSets = currentSets.copy()
                copyCurrentSets = copyCurrentSets.union(nodeSets[index])

                findMin(index + 1, currentCost + setsPrice[index], copyChoosenIndex, copyCurrentSets)

    findMin(0, 0, [], set())
    return minCost, resultSubset

def main(nodeCount, nodeSets, setsPrice):
    start_time = time.time()
    minCost, resultSubset=(BB(nodeCount, nodeSets, setsPrice))
    end_time = time.time()
    cost= minCost
    cover= []
    for i in resultSubset:
        cover.append(nodeSets[i])
    print('covering sets: ',cover,'\n','total cost: ',cost,'$')
    print('time:',end_time-start_time)
```

- Greedy Weighted Set Cover

```python
import time

def Greedy(nodeCount, nodeSets : list, setsPrice : list):
    currentSets = set()
    result = []
    price = 0
    while len(currentSets) != nodeCount:
        choosenSubset = max(nodeSets, key=lambda s: len(set(s) -
currentSets)/setsPrice[nodeSets.index(s)])
        result.append(choosenSubset)
        price += setsPrice[nodeSets.index(choosenSubset)]
        currentSets = currentSets.union(set(choosenSubset))

    return price, result


def main(nodeCount, nodeSets, setsPrice):
    start_time = time.time()

    minCost, resultSubset=(Greedy(nodeCount, nodeSets, setsPrice))
    end_time = time.time()
    print('covering sets: ',resultSubset,'\n','total cost: ',minCost,'$')
    print('time:',end_time-start_time)
```