Algoritmos Lista 2-Q1

Goleiro Sistemas

Um sistema de segurança e controle de redes de computadores coleta informações sobre o tráfego na forma de uma **stream** de dados em que vários "pontos" na forma

```
(X0,W1), (X1,W1), (X2,W2), ...
```

são obtidos em fluxo contínuo. Cada ponto `(Xj, Wj)` representa um pacote transmitido, sendo `Xj` um valor inteiro num intervalo `[M]=0..M-1` que corresponde a um identficador único de um dispositivo, e `Wj` um inteiro positivo que corresponde ao no. de bytes transmitidos (payload) naquele pacote.

Uma atividade suspeita na rede é detectada quando um dispositivo transmite uma quantidade muito grande de informação. Assim, algumas consultas frequentes são:

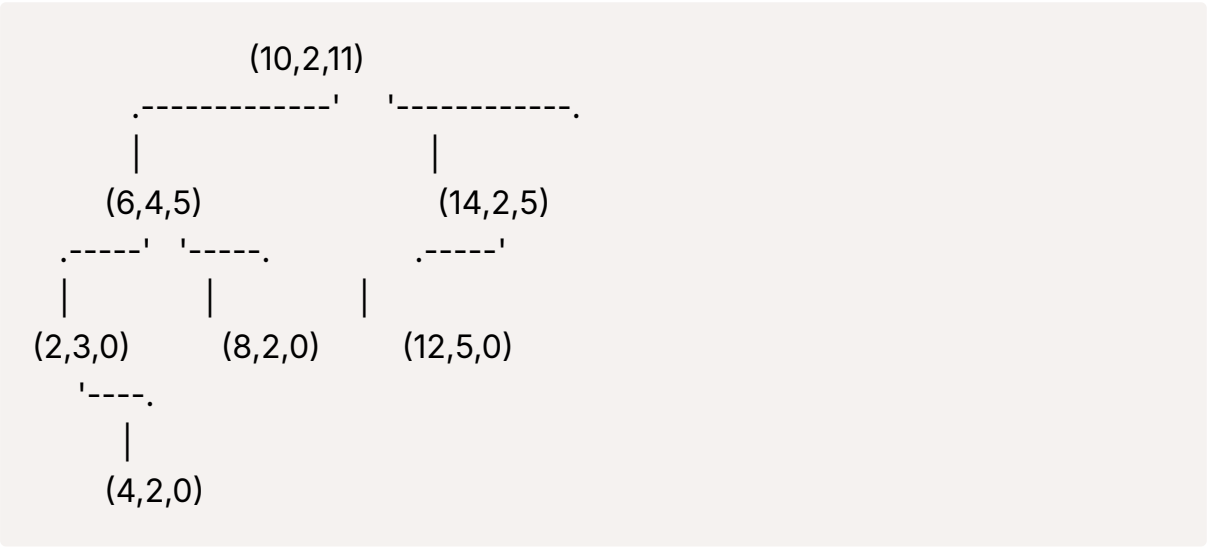
- `WEI(X)`: qual a quantidade total de bytes transmitidos pelo dispositivo com id `X`
- `RNK(X)`: qual a quantidade total de bytes transmitidos pelos dispositivos com id `< X`

A primeira consulta inspeciona a atividade de um dispositivo em particular. Já a segunda consulta pode servir para estimar a distribuição do tráfego entre os dispositivos. Por exemplo, num tráfego uniformemente distribuído, a metade do total de bytes transmitidos na rede deveria ter sido originada pelos dispositivos com id `< M/2`.

Steve foi contratado pela empresa **Goleiro Sistemas** para implementar essas funcionalidades. Imediatamente veio-lhe à memória as aulas sobre **Árvores**

AVL e uma maneira de usar essa estrutura para resolver o problema. A solução consiste em criar uma AVL ordenada pelos ids dos dispositivos. Além do campo `ID` com um certo valor `X`, cada nó `N` deve conter um campo `WEI` (peso) com a soma total dos bytes transmitidos pelo dispositivo `X` até o momento, ou seja, a resposta à consulta `WEI(X)`, e um outro campo `RNK` com a soma total dos bytes transmitidos pelos dispositivos com id estritamente inferior a `X` que estão na sub-árvore à esquerda de `N`, ou seja, a soma dos valores dos campos `WEI` nessa sub-árvore.

Por exemplo, a árvore a seguir ilustra o caso para 7 dispositivos. Os nós estão na forma `(ID,WEI,RNK)`.



Neste caso, a consulta por `RNK(10)` retorna o valor do campo `RNK` da raiz. A resposta para a consulta `RNK(5)` é obtida totalmente na sub-árvore à esquerda. Já a resposta a `RNK(14)` é obtida a partir da soma dos pesos dos valores menores que 14, isto é, a raiz e os elementos à sua esquerda, mais a soma dos pesos dos valores menores que 14 na sub-árvore direita.

Input Specification

A entrada é constituída por várias linhas, cada uma num dos formatos a seguir:

- `ADD X W`: sinaliza a transmissão e um novo pacote de `W` bytes para o dispositivo com id `X`.

- `WEI X`: consulta o total de pacotes transmitidos pelo dispositivo com id `X` até o momento.
- `RNK X`: consulta o total de pacotes transmitidos pelo dispositivo com id `< X` até o momento.

A entrada termina com uma linha

```
END
```

Output Specification

Para cada linha da entrada deve ser impressa uma saída correspondente como descrito a seguir.

- `ADD X W` ⇒ adiciona a informação à AVL e imprime uma linha com um inteiro correspondente ao total global de bytes transmitidos por todos os dispositivos na rede até o momento.
- `WEI X` ⇒ imprime uma linha com dois inteiros `W D` correspondentes, respectivamente, ao valor de `WEI(X)`, e à *profundidade* do nó com id `X` na AVL (a raiz tem profundidade 0, os seus filhos têm profundidade 1, e assim sucessivamente) **NOTA**: se a máquina com id `X` ainda não transmitiu dados, deve ser impressa a linha `0 -1`.
- `RNK X` ⇒ imprime uma linha com um inteiro correspondente ao total de bytes transmitidos por dispositivos com id `< X` até o momento. **NOTA**: não necessariamente o dispositivo com id `X` transmitiu dados até o momento. Ou seja, a AVL pode não conter um nó com id `X`. Mesmo assim a consulta tem uma resposta correta, ainda que seja `0`.

Exs.: Input

Exs.: Output

```
ADD 8 3641
ADD 43 3427
ADD 20 638
ADD 25 2197
ADD 42 3444
ADD 29 984
ADD 34 1589
ADD 42 1313
ADD 56 3203
ADD 37 300
ADD 35 1603
ADD 11 779
ADD 20 1517
ADD 40 2272
ADD 60 1809
ADD 3 2439
ADD 56 2059
ADD 42 209
ADD 55 2854
ADD 34 1086
ADD 51 1910
ADD 41 2773
ADD 33 1127
ADD 62 3043
ADD 48 3390
ADD 38 681
ADD 37 1868
ADD 54 1152
ADD 63 2893
ADD 62 708
ADD 42 1037
ADD 19 2769
WEI 19
WEI 25
ADD 32 153
ADD 6 2116
ADD 52 1564
```

```
3641
7068
7706
9903
13347
14331
15920
17233
20436
20736
22339
23118
24635
26907
28716
31155
33214
33423
36277
37363
39273
42046
43173
46216
49606
50287
52155
53307
56200
56908
57945
60714
2769 4
2197 2
60867
62983
64547
```

ADD 39 702	65249
WEI 37	2168 3
ADD 40 16	65265
WEI 35	1603 4
WEI 63	2893 4
ADD 17 3397	68662
WEI 37	2168 3
ADD 7 1247	69909
ADD 54 1067	70976
ADD 58 2287	73263
WEI 35	1603 4
WEI 43	3427 4
ADD 60 1951	75214
ADD 6 1594	76808
ADD 52 3709	80517
WEI 3	2439 3
ADD 12 76	80593
ADD 46 584	81177
ADD 53 576	81753
WEI 3	2439 3
ADD 30 2525	84278
ADD 18 929	85207
ADD 29 2333	87540
ADD 41 1093	88633
ADD 48 1270	89903
ADD 15 873	90776
WEI 32	153 3
ADD 38 651	91427
ADD 33 3621	95048
ADD 21 2634	97682
ADD 27 2111	99793
ADD 42 1043	100836
WEI 22	0 -1
WEI 6	3710 3
WEI 35	1603 5
ADD 5 584	101420
ADD 2 2961	104381
ADD 53 1737	106118

ADD 24 2627	108745
ADD 62 3041	111786
ADD 61 1570	113356
ADD 30 2924	116280
ADD 32 1092	117372
ADD 42 3281	120653
ADD 20 1606	122259
WEI 17	3397 1
ADD 35 1096	123355
ADD 44 2626	125981
WEI 44	2626 5
WEI 25	2197 2
ADD 54 990	126971
ADD 58 353	127324
ADD 18 1423	128747
WEI 20	3761 5
ADD 11 1062	129809
ADD 61 2262	132071
WEI 56	5262 2
ADD 24 2766	134837
WEI 25	2197 2
ADD 4 367	135204
ADD 48 2093	137297
WEI 8	3641 4
WEI 40	2288 4
WEI 6	3710 2
ADD 35 856	138153
WEI 32	1245 3
ADD 2 1792	139945
ADD 63 3223	143168
ADD 37 3778	146946
ADD 57 954	147900
ADD 61 3289	151189
ADD 13 3056	154245
WEI 30	5449 5
WEI 24	5393 5
WEI 3	2439 3
ADD 36 3853	158098

```
ADD 8 1577
WEI 55
ADD 2 3611
WEI 32
ADD 50 2888
ADD 13 3086
ADD 53 3767
ADD 6 3242
ADD 26 3451
WEI 54
WEI 44
ADD 30 1925
ADD 59 596
ADD 11 2011
ADD 6 1176
WEI 6
ADD 0 3349
WEI 40
ADD 5 672
ADD 47 519
WEI 63
ADD 40 4028
WEI 24
ADD 4 1614
WEI 3
WEI 38
WEI 26
ADD 28 3642
ADD 16 498
WEI 3
WEI 47
ADD 1 1704
ADD 20 273
ADD 44 3460
ADD 29 2984
WEI 37
ADD 9 474
ADD 5 184
```

```
159675
2854 4
163286
1245 3
166174
169260
173027
176269
179720
3209 3
2626 5
181645
182241
184252
185428
8128 2
188777
2288 4
189449
189968
6116 5
193996
5393 5
195610
2439 3
1332 3
3451 5
199252
199750
2439 3
519 5
201454
201727
205187
208171
5946 5
208645
208829
```

WEI 61	7121 5
ADD 57 1294	210123
ADD 27 677	210800
ADD 41 3988	214788
WEI 52	5273 4
ADD 5 2532	217320
ADD 45 673	217993
WEI 10	0 -1
ADD 60 3798	221791
ADD 41 2757	224548
ADD 52 1293	225841
WEI 61	7121 5
WEI 46	584 3
WEI 45	673 5
WEI 2	8364 5
WEI 28	3642 5
WEI 56	5262 2
ADD 22 1744	227585
WEI 21	2634 3
WEI 3	2439 3
ADD 17 3335	230920
ADD 46 1752	232672
ADD 18 301	232973
ADD 31 2660	235633
WEI 32	1245 4
ADD 48 464	236097
WEI 55	2854 4
ADD 44 1455	237552
ADD 2 2324	239876
ADD 49 2350	242226
WEI 1	1704 4
WEI 45	673 5
ADD 46 2389	244615
WEI 22	1744 5
WEI 3	2439 3
ADD 24 3017	247632
ADD 38 2050	249682
ADD 21 1671	251353

```
ADD 44 255
ADD 37 1855
WEI 60
WEI 50
WEI 40
ADD 19 452
ADD 38 1698
WEI 48
ADD 3 3049
ADD 27 1609
ADD 37 2791
END
```

```
251608
253463
7558 3
2888 5
6316 4
253915
255613
7217 4
258662
260271
263062
```

```
ADD 52 1934
ADD 44 3426
ADD 32 1479
ADD 48 657
ADD 58 3569
ADD 39 1566
ADD 57 1159
ADD 14 1628
ADD 50 75
ADD 50 2724
ADD 13 257
ADD 10 1982
ADD 60 1179
ADD 58 378
ADD 42 2422
ADD 51 3981
ADD 31 2772
ADD 23 2056
ADD 63 2090
ADD 34 2496
ADD 51 3660
ADD 28 281
ADD 0 1383
```

```
1934
5360
6839
7496
11065
12631
13790
15418
15493
18217
18474
20456
21635
22013
24435
28416
31188
33244
35334
37830
41490
41771
43154
```

```
ADD 56 1645
ADD 56 602
ADD 38 1136
ADD 9 2651
ADD 25 125
ADD 41 3875
ADD 46 2724
ADD 11 503
RNK 62
WEI 23
ADD 39 2536
ADD 44 2678
ADD 7 830
ADD 46 250
ADD 5 182
ADD 32 1771
ADD 20 1022
WEI 62
ADD 16 2596
WEI 34
WEI 41
ADD 14 1019
ADD 36 3229
ADD 44 3424
RNK 25
WEI 38
ADD 43 1183
WEI 20
ADD 29 2834
RNK 29
WEI 16
RNK 13
ADD 22 3331
WEI 25
ADD 38 208
ADD 56 1568
ADD 38 484
ADD 3 4063
```

```
44799
45401
46537
49188
49313
53188
55912
56415
54325
2056 1
58951
61629
62459
62709
62891
64662
65684
0 -1
68280
2496 4
3875 5
69299
72528
75952
16109
1136 5
77135
1022 5
79969
16515
2596 4
7531
83300
125 4
83508
85076
85560
89623
```

```
ADD 1 2930
ADD 40 2962
RNK 44
ADD 51 1002
ADD 16 1255
RNK 41
WEI 57
WEI 58
ADD 60 3198
WEI 20
WEI 29
ADD 49 1510
ADD 22 1233
WEI 1
RNK 1
ADD 51 3855
WEI 13
ADD 25 813
RNK 32
ADD 12 3594
WEI 43
ADD 35 1490
ADD 0 1897
ADD 6 3394
ADD 29 1810
RNK 11
RNK 1
ADD 41 2190
ADD 46 823
RNK 35
ADD 31 925
ADD 39 2632
WEI 62
RNK 16
WEI 57
ADD 57 1842
ADD 8 1397
ADD 15 1696
```

```
92553
95515
57792
96517
97772
51567
1159 4
3947 3
100970
1022 4
2834 4
102480
103713
2930 5
1383
107568
257 4
108381
35746
111975
1183 4
113465
115362
118756
120566
19312
3280
122756
123579
52187
124504
127136
0 -1
26313
1159 4
128978
130375
132071
```

```
RNK 59
ADD 26 2279
ADD 27 1386
ADD 41 4027
RNK 18
RNK 45
ADD 33 1418
ADD 42 526
RNK 10
ADD 26 585
ADD 53 74
RNK 33
RNK 56
WEI 20
RNK 15
WEI 60
ADD 21 1521
ADD 58 766
RNK 63
ADD 22 1166
ADD 10 4010
RNK 9
ADD 14 2882
ADD 30 3317
WEI 19
RNK 39
ADD 54 1834
WEI 52
ADD 23 1369
RNK 41
ADD 8 1833
ADD 62 3240
ADD 6 3987
ADD 13 719
WEI 40
ADD 13 2012
RNK 61
ADD 45 120
```

```
125604
134350
135736
139763
33257
99338
141181
141707
18727
142292
142366
57959
125136
1022 3
27710
4377 4
143887
144653
142563
145819
149829
16076
152711
156028
0 -1
81316
157862
1934 2
159231
92381
161064
164304
168291
169010
2962 5
171022
165692
171142
```

RNK 58	156722
ADD 12 2312	173454
WEI 16	3851 1
RNK 18	51012
ADD 44 231	173685
ADD 54 2384	176069
ADD 57 886	176955
ADD 19 625	177580
ADD 37 3582	181162
WEI 7	830 4
RNK 11	30539
ADD 8 904	182066
WEI 54	4218 6
WEI 6	7381 6
ADD 63 2809	184875
ADD 9 899	185774
RNK 21	54462
ADD 17 1230	187004
ADD 8 448	187452
RNK 28	72004
ADD 0 1472	188924
WEI 34	2496 4
WEI 46	3797 4
ADD 5 3153	192077
ADD 7 323	192400
WEI 50	2799 4
ADD 34 502	192902
WEI 1	2930 5
ADD 58 272	193174
ADD 60 1352	194526
WEI 8	4582 6
ADD 15 2142	196668
RNK 40	115562
RNK 7	22461
RNK 38	107000
WEI 41	10092 4
WEI 17	1230 5
RNK 44	132747

```
ADD 0 228
ADD 35 3512
RNK 3
ADD 14 2609
RNK 48
RNK 37
ADD 46 3099
WEI 27
ADD 40 182
RNK 20
WEI 58
RNK 30
ADD 52 2139
RNK 63
ADD 3 1718
ADD 28 293
WEI 5
RNK 59
ADD 39 2360
RNK 26
WEI 30
RNK 28
RNK 50
ADD 2 64
ADD 17 27
END
```

```
196896
200408
7910
203017
152772
109767
206116
1386 5
206298
65045
4985 3
86856
208437
203538
210155
210448
3335 5
196580
212808
79399
3317 4
83649
162591
212872
212899
```

```python
class NodoAVL:
    def __init__(self, chave, valor):
        self.chave = chave
        self.valor = valor
        self.altura = 1
        self.esquerda = None
        self.direita = None

class ArvoreAVL:
    def __init__(self):
```

```
        self.raiz = None

    # Função para obter a altura de um nodo
    def _obter_altura(self, nodo):
        if not nodo:
            return 0
        return nodo.altura

    # Função para calcular o fator de balanceamento
    def _obter_balanceamento(self, nodo):
        if not nodo:
            return 0
        return self._obter_altura(nodo.esquerda) - self._obter_altura(nodo.direita)

    # Rotação à direita
    def _rotacionar_direita(self, y):
        x = y.esquerda
        T2 = x.direita

        # Realiza a rotação
        x.direita = y
        y.esquerda = T2

        # Atualiza as alturas
        y.altura = 1 + max(self._obter_altura(y.esquerda), self._obter_altura(y.direi
        x.altura = 1 + max(self._obter_altura(x.esquerda), self._obter_altura(x.direi

        return x

    # Rotação à esquerda
    def _rotacionar_esquerda(self, x):
        y = x.direita
        T2 = y.esquerda

        # Realiza a rotação
        y.esquerda = x
        x.direita = T2
```

```
        # Atualiza as alturas
        x.altura = 1 + max(self._obter_altura(x.esquerda), self._obter_altura(x.direi
        y.altura = 1 + max(self._obter_altura(y.esquerda), self._obter_altura(y.direi

        return y

    # Inserção na árvore AVL
    def inserir(self, chave, valor):
        def _inserir(nodo, chave, valor):
            # Passo 1: Inserção normal de BST
            if not nodo:
                return NodoAVL(chave, valor)
            if chave < nodo.chave:
                nodo.esquerda = _inserir(nodo.esquerda, chave, valor)
            elif chave > nodo.chave:
                nodo.direita = _inserir(nodo.direita, chave, valor)
            else:  # Chaves iguais, atualiza o valor
                nodo.valor = valor
                return nodo

            # Passo 2: Atualiza a altura do nodo ancestral
            nodo.altura = 1 + max(self._obter_altura(nodo.esquerda), self._obter_alt

            # Passo 3: Obtém o fator de balanceamento
            balanceamento = self._obter_balanceamento(nodo)

            # Caso 1: Desbalanceamento esquerda-esquerda
            if balanceamento > 1 and chave < nodo.esquerda.chave:
                return self._rotacionar_direita(nodo)

            # Caso 2: Desbalanceamento direita-direita
            if balanceamento < -1 and chave > nodo.direita.chave:
                return self._rotacionar_esquerda(nodo)

            # Caso 3: Desbalanceamento esquerda-direita
            if balanceamento > 1 and chave > nodo.esquerda.chave:
                nodo.esquerda = self._rotacionar_esquerda(nodo.esquerda)
                return self._rotacionar_direita(nodo)
```

```
            # Caso 4: Desbalanceamento direita-esquerda
            if balanceamento < -1 and chave < nodo.direita.chave:
                nodo.direita = self._rotacionar_direita(nodo.direita)
                return self._rotacionar_esquerda(nodo)

            return nodo

        self.raiz = _inserir(self.raiz, chave, valor)

    # Busca na árvore AVL
    def buscar(self, chave):
        def _buscar(nodo, chave):
            if not nodo or nodo.chave == chave:
                return nodo
            if chave < nodo.chave:
                return _buscar(nodo.esquerda, chave)
            return _buscar(nodo.direita, chave)

        resultado = _buscar(self.raiz, chave)
        return resultado.valor if resultado else None

# Exemplo de uso
if __name__ == "__main__":
    avl = ArvoreAVL()
    avl.inserir(10, "Valor 10")
    avl.inserir(20, "Valor 20")
    avl.inserir(5, "Valor 5")
    avl.inserir(4, "Valor 4")
    avl.inserir(6, "Valor 6")

    print(avl.buscar(10))  # Retorna "Valor 10"
    print(avl.buscar(15))  # Retorna None
```

Na função atualizar no você insistiu em compactar as coisas mesmo eu falando muitas vezes para não fazer isso. Eu quero que você analise todo o código que ja temos e edite cada parte que você fez algo parecido refaça de uma maneira menos compacta e legivel

```
    subEsq = no.esq.subTotal if no.esq is not None else 0
    subDir = no.dir.subTotal if no.dir is not None else 0
```