Nama : Alvaro Austin
NPM : 2106752180
kelas : A
Kode Asisten : 3

Dengan ini saya menyatakan bahwa PR-1 ini adalah hasil pekerjaan saya sendiri

① Algoritma dalam python :

```
From math import floor
def two-array-xxx(X,Y,n):
    x = Find-xxx(X,Y,n,0,n)
    if(x == "Not Found"):
        x = Find-xxx(Y,X,n,0,n)
    return x
def Find-xxx(A,B,n,low,high):
    if(low > high):
        return "Not Found"
    else:
        k = floor((low+high)/2)
        if k == n and A[n] ≤ B[0]:
            return A[n]
        elif k<n-1 and B[n-k-1] ≤ A[k] ≤ B[n-k]:
            return A[k]
        elif A[k] > B[n-k]
            return Find-xxx(A,B,n,low,k-1)
        else:
            return Find-xxx(A,B,n,k+1,high)
X = [16,19,22,25,30,39]
Y = [11,20,29,20,36,40]
print(two-array-xxx(x,Y,len(x),1))
```

dijalankan dengan pemanggilan Two-ARRAY-xxx(x,Y,6)
x = Find-xxx(X,Y,6,1,6)
↳ 1≤6 maka
k = 3
line 5 : tidak masuk karena k ≠ 6
line 7 : tidak masuk karena B[3] ≤ A[5] ≤ B[4]
22 ≰ 22 ≤ 28
line 9 : A[3] > B[4] salah karena 22 ≤ 28
masuk ke Find-xxx(A,B,6,4,6)
↳ 4≤6 maka :
k = 5
line 6 tidak masuk karena 5≠6
line 7 tidak masuk karena B[1] ≤ A[5] ≤ B[2] tidak memenuhi
11 30 20
line 9 masuk karena A[5] > B[2] → 30 > 20
Find-xxx(A,B,6,4,4)
↳ line 1 tidak memenuhi karena low ≤ high, 4≤4 maka
k=4
line 5 tidak masuk karena A[6] > B[1]
30 > 11
line 7 tidak masuk, maka dikembalikan
A[4] = 25

Sehingga
X = 25
karena X ≠ Not-Found maka dikembalikan nilai 25

Algoritma ini digunakan untuk mencari median dari 2 array terurut X dan Y dengan ukuran yg sama
(median = midpoint)

Median ini pasti ada karena X dan Y memiliki ukuran yg sama dan kombinasi len(x) = n , len(Y) = n → len(x)+len(Y) = 2n pasti bisa dibagi 2 karena 2·n pasti bisa dibagi 2 untuk nilai n bilangan bulat positif.

Sehingga algoritma diatas mencari median (midpoint dari array X dan array Y dengan ukurannya sama dan terurut.

b) Kompleksitas waktu dari algoritma TWO-ARRAY-XXX adalah

Best case pada array : X = [1,3,5,7,9]
Y = [2,4,6,8,10]
dimana time complexity T(n) = O(1) → tidak ada operasi rekursif
karena pada pemanggilan FIND-XXX pertama, didapatkan nilai midpoint / mediannya

kondisi yg memenuhi :
IF k ≤ n and B[n-k] ≤ A[k] ≤ B[n-k+1]
dimana kita tahu bahwa n = 5 dan k = 3, sehingga,

k = 3 ≠ 5 - dalam
A[3] = 5
B[2] = 4
B[3] = 6

dimana diperoleh mediannya adalah A[3] = 5 , dimana benar bahwa ini adalah mediannya.

Worst case pada array : Karena pada setiap worst case algoritma diatas, akan melakukan evaluasi terhadap $\frac{n}{2}$ subarray. Hal ini dilihat dari pemanggilan fungsi kembali pada baris 10 dan 12, Sehingga pada setiap pemanggilan rekursif, subarray yg dievaluasi hanya $\frac{1}{2}$ dari array sebelumnya.

Fungsi rekurensi : $T(n) = T\left(\frac{n}{2}\right) + O(1)$

atau $T(n) = O(\log n)$

Alvaro Austin - 2106752180 - A-3

1c) IF then else :

- IF X = NOT-FOUND then
  X = FIND-XXX(Y, X, 0, 1, n) : IF condition ini akan mencari midpoint (median) pada array Y dan bukan array X. Hal ini akan dilakukan apabila median tidak ditemukan pada array X. Maka, apabila tidak ditemukan, median seharusnya berada pada array Y. Kondisi ini dapat dilakukan karena array X dan Y terurut ascending. Median diatas maksudnya median dari array X + Y

  - IF low > high : Apabila batasan bawah lebih besar dari batasan atas, maka median pada array A tidak ditemukan.

- IF k = n and A[n] ≤ B[1]:
  return A[n]
  : IF condition ini berfungsi untuk mendapatkan midpoint dari array A+B apabila semua elemen pada array A lebih kecil dari pada array B.
  Kita tahu bahwa kedua array ini terurut, sehingga apabila nilai terbesar pada array A lebih kecil dari array B. Tentu saja, nilai terbesar adalah midpoint / median dari array A+B. Hal ini karena, apabila diurutkan array A+B akan berbentuk seperti ini

  Ingat A, B merupakan array X/Y dimana keduanya terurut.

  [a, b, c, d, ... , f, g, h, ...]
  A : panjang = n B : panjang = n

  karena panjang A+B = 2n, maka midpoint / median pasti berada pada indeks n dari array A+B. Kita tahu bahwa indeks n dari array A+B adalah A[n].

- else IF k < n and B[n-k] ≤ A[k] ≤ B[n-k+1] then
  return A[k]
  : IF condition ini berfungsi untuk mengevaluasi apabila nilai dari A[k] yg merupakan midpoint dari sebuah sub array [low .... high]. kondisi ini bekerja dengan konsep bahwa apabila A[k] merupakan midpoint, maka kita akan melihat apabila B[n-k] lebih kecil dari pada A[k] maka terdapat (n-k) elemen dari array B. Kita juga tahu bahwa pastinya ada k elemen pada array A dimana elemen tersebut lebih kecil dari A[k]. Maka kita tahu bahwa terdapat (n-k)+k elemen dimana menyebutkan bahwa A[k] adalah elemen ke-n dari array A+B sorted.

  ↳ untuk k < n : mengecek apakah nilai k masih berada dalam batasan array B

- else if A[k] > B[n-k+1] then
  return FIND-XXX(A, B, n, low, k-1) : IF condition ini melakukan evaluasi median berdasarkan pencarian rekursif dari subarray [low .... k-1]. Ide dari kondisi ini karena pada elemen A[k] memiliki nilai dari B[n-k+1], Hal ini berarti terdapat n-k+1 elemen array B yg lebih kecil dari A[k]. Berarti, karena midpoint mencari elemen ke-n dari array sorted A+B, namun elemen A[k] saja sudah memiliki (n+1) elemen yg lebih kecil dari A[k].
  Hal ini berarti midpoint (median berada di sebelah kiri k. Oleh karena itu high boundary nya menjadi high, dan dilakukan kembali binary search.

- else (pada baris 4) : IF condition ini dapat dilakukan karena terdapat kurang dari (n-k) elemen yg berada lebih kecil dari A[k]. Maksudnya, pada kondisi ini kita tahu bahwa pada array A+B, terdapat kurang dari n elemen dimana hal ini berarti midpoint / median kedua kombinasi array berada di bagian sebelah kanan k. Oleh karena itu low digant menjadi k+1, lalu diulang step diatas.

Alvaro Austin - 2106732280 - KA-3

2a) Nilai maksimum total keramahan apabila x diundang :

X.Invited = Ada intuisi yg dapat kita gunakan untuk memperoleh nilai dari x ini. Apabila x diundang, maka kita dapat tentunya mengetahui bahwa inisial nilai total keramahan adalah nilai keramahan dari x. Selanjutnya, kita perlu untuk melihat, bahwa kita tidak dapat menambahkan keramahan dari bawahan langsung dari x tersebut. Sehingga dari intuisi tersebut, kita perlu membangun persamaan yg merepresentasikan sifat tersebut :

Berarti kita perlu mendapatkan nilai maksimum dari subtree child x dimana child x tersebut tidak diundang. Oleh karena itu, kita dapat memperoleh nilai X.invited, child yg dimaksud adalah bawahan.

$$X.Invited = \underbrace{X.keramahan}_{\text{initial value}} + \sum_{y \in child(x)}^{children(x)} y.not\_invited$$

dimana child(x) : bawahan langsung X

children(x) : banyaknya bawahan pada subtree X (termasuk juga : bawahan dan bawahan langsung
- bawahan dari bawahan langsung.
dari bawahan langsung
- dst.
)

Persamaan diatas konsisten dengan ide konstruksi diatas, dimana initial value adalah nilai keramahan X dan $\sum_{y \in child(x)}^{children(x)}$ y.not_invited adalah nilai dimana child tersebut tidak di undang karena merupakan bawahan langsung dari X

total keramahan subtree(x).

$X_{not\_invited}$: Intuisi yg digunakan adalah seharang x tidak diundang, maka bawahan dari X dapat di gunakan sebagai barangsawan yg diundang (bisa juga tidak). Maka kita dapat memperoleh Persamaan subproblem tersebut:

$$X_{not\_invited} = \sum_{y \in child(x)}^{children(x)} max\_value(y.invited, y.not\_invited)$$

untuk mengkalkulasi keramahan maksimal, dengan x tidak diundang, bisa saja bawahan child(x) yg perlu diundang (misalnya child(child(x))). Oleh karena itu, kita perlu mengambil nilai maksimumnya.

Nama: Alvaro Austin
Kelas: A
NRM: 2106752180
Kode asmen: 3

2b) MAXIMIZEKERAMAHAN(x):

```
if not hasChild(x):
    return (x.keramahan, 0) # tuple pair (x.invited, x.not_invited)
else:
    # untuk menyimpan, kita akan menggunakan memorization dengan list keramahan yg berupa dictionary memorization
    if keramahan.get(x) is not NULL:
        return keramahan[x] # Tidak akan adaerror karena sudah ada di ifcondition
        # Dijalankan karena agar tidak mengulang operasi evaluasi apabila sudah pernah dievaluasi.
    else: y ← x.left_child
          keramahan[x] ← (x.keramahan, 0)
          while y is not empty:
              (y.invited, y.not_invited) ← MAXIMIZEKERAMAHAN(y)
              keramahan[x][0] ← keramahan[x][0] + y.not_invited
              keramahan[x][1] ← keramahan[x][1] + max_value(y.invited, y.not_invited)
              y ← y.right_sibling
    return keramahan[x]
```

hasChild(x):
return x.left_child is not empty.

Pada algoritma diatas, setiap proses pencarian setiap node (x) membutuhkan sebanyak $O(1)$ operasi. Pada algoritma ini kita akan mencari setiap kombinasi subtree agar menghasilkan total keramahan yg maksimal. Pada normal, akan ditelusuri sebesar $O(2^n)$ hal, quantetap, sudah dilakukan memorization, sehingga tidak perlu untuk mengulang operasi lagi. Sekarang, secara rata2 operasi ini, akan dijalankan sebanyak $\Theta(n)$ operasi. Maka akibat memorization, kompleksitas diatas adalah $\Theta(n)$.

2c)

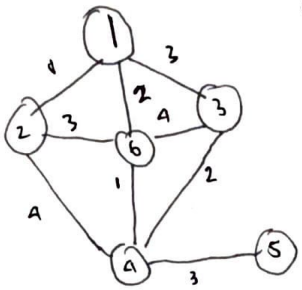
```
DAFTAR_TAMU = [] # list dari 1...n
OUTPUT UNDANGAN(P):
    (P.invited, P.not_invited) ← MAXIMIZE KERAMAHAN(P)
    child ← P.left_child
    # Apabila P diundang menghasilkan keramahan terbaik
    if P.invited ≥ P.not_invited
        DAFTAR_TAMU.append(P)
        while child is not empty:
            grand_child ← child.left_child
            while grand_child is not empty:
                OUTPUT UNDANGAN(grand_child) # mencari pada subtree bawahan langsung dari bawahan bagsung.
                grand_child ← grand_child.right_sibling # menelusuri bawahan lainnya
            child ← child.right_sibling
    else:
        while child is not empty:
            OUTPUT UNDANGAN(child) # mencari pada bawahan langsung
            child ← child.right_sibling
for i from 1 to DAFTAR_TAMU.length do
    Print(DAFTAR_TAMU[i].name)
```

Pada algoritma diatas kompleksitas waktu untuk setiap node (x), proses evaluasinya sebanyak $O(1)$ operasi. Quantetap, apabila kita melakukan evaluasi terhadap seluruh subtree maka secara rata2 keseluruhan membutuhkan kompleksitas waktu sebesar $\Theta(n)$ untuk tree dengan

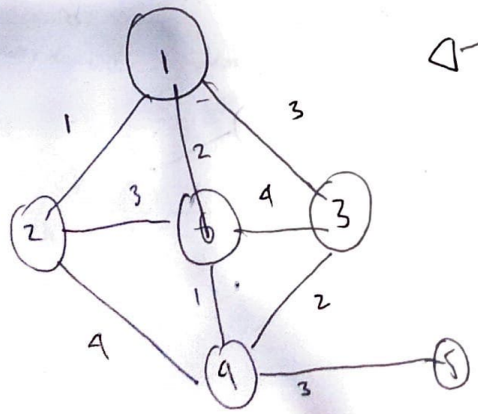
fitggi n //

Nama : Alvaro Pastin
Kelas : A
Npm : 2106752180
Kode Asisten : 3

3a) Hasil yg diperoleh : [1, 2, 3, 3, 1, 4, 2, 4, 3]



Step bystep :
- e : 9
- colors : [0,0,0,0,0,0,0,0,0]
- i : 0
- color : 1
- Iterasi 9 kali,
  - i : 0
    - colors[0] : 1
    - Flag : false
    - tidak pernah masuk ke kondisi baris 11, sehingga langsung masuk ke kondisi baris 15.
  - i : 1
    - colors[1] : 1
    - Flag : false
    - masuk ke kondisi 11, lalu color : 2, dan override colors[1] = 2, ubah color : 1, dan i : i+1 lagi
  - i : 2
    - colors[2] : 4
    - Flag : false
    - masuk ke kondisi 11 karena colors[0] = colors[2], dan ada intersection antara (1,2) dan (1,3), ubah color : 2
    - flag : true, lalu colors[2] : 2, flag : false, masuk lagi kondisi 11, color : 3, ubah colors[2] : 3,
    - Flag : false, lalu color : 1, i : i+1
  - i : 3 (step sama seperti sebelumnya, sehingga hanya beberapa hal yg diperhatikan)
    - Colors[3] : 1
    - Flag : false
    - colors[3] : 2
      - Flag : false
      - color : 3
      - Flag : true
      - colors[2] : 3
      - Flag : false
      - i : 4
      - Color : 1
  - i : 4
    - colors[4] : 1
    - Flag : false
    - tidak masuk ke baris 11 sama sekali karena intersect dengan (6) maka edge list yg intersect hanyalah [(1,6)(2,6)] namun nilai mereka ≠ 1, maka untuk [(3,6),(3,4)(2,4)(4,5)] yg bernilai masih 0, walaupun intersect, tidak berpengaruh!
    - Maka color : 1
    - i : 5
  - i : 5
    - colors[5] : 1
    - Flag : false
    - (untuk [(1,6), (1,3), (2,6)] (6,4) intersect dengan (3,6))
    - color : 2
    - Flag : True
    - colors[5] : 2
    - Flag : False
    - color : 3
    - Flag : True
    - colors[5] : 3
    - Flag : False
    - color : 4
    - Flag : True
    - colors[5] : 4
    - Flag : False
    - --- berhenti ---
  - i : 6
    - colors[6] : 1
    - Flag : False
    - color : 2
    - Flag : true
    - colors[6] : 2
    - Flag : false
    - masuk karena intersect dengan (6,4) yg memiliki edge : 1
    - namun untuk yg bernilai 1 adalah : [(1,3), (3,6), (2,4)(4,5)] → 3, 4, 0, 0
    - tidak ada yg bernilai 2
    - ..... Berhenti dengan color : 1, i : 7
  - i : 7 untuk [(1,2), (2,6), (6,4), (3,4), (4,5)] (1, 3, 1, 2, 0) dapat dihat bahwa yg belum adalah color : 4.
    - colors[7] : 1
    - Flag : False
    - color : 2
    - Flag : true
    - colors[7] : 2
    - Flag : False
    - color : 3
    - Flag : true
    - colors[7] : 3
    - Flag : False
    - color : 4
    - Flag : true
    - colors[7] : 4
    - Flag : False
    - --- Berhenti ---
    - Color : 1
    - i : 8
  - i : 8 untuk : [(1,6), (3,4), (2,4)] → 6, 6, 6 → 1, 2, 4 tidak ada color : 3
    - Colors[8] : 1
    - Flag : False
    - color : 2
    - Flag : True
    - colors[8] : 2
    - Flag : False
    - color : 3
    - Flag : True
    - colors[8] : 3
    - Flag : False
    - --- Loop Berhenti ---



List yang dihasilkan karena i : 9 Colors :

[1, 2, 3, 3, 1, 4, 2, 4, 3]

Nama: Bhara B, node asuransi: 3, kelas: B, NPM: 2106752180

5 b) Langkah-langkah algoritma Edge Coloring:

- Pertama, pada algoritma ini, dilakukan inisialisasi list yg memiliki panjang sebanyak edge yg ada.
  - ↳ List ini dinamakan sebagai colors
- Selanjutnya akan di pre-defined bahwa color = 1
- Selanjutnya iterasi untuk setiap elemen pada list colors:
  - Pada iterasi ini, anggap iterasi ini disebut i. Maka elemen dari colors ke-i akan diganti nilainya berdasarkan nilai color yg diperoleh dari iterasi sebelumnya (j)
  - Tidak hanya itu, akan juga dilakukan perubahan nilai flag menjadi false, Flag ini nantinya akan menandakan apakah iterasi sudah siap masuk ke iterasi selanjutnya
  - Selanjutnya, terdapat iterasi yg dijalankan sebanyak jumlah edge yg ada sebut saja j
    - ↳ Pada iterasi ini, akan dilakukan pengecekan untuk setiap edge, apabila ada node yg beririsan antara 4 kombinasi node
      misalnya (x, y) dan (y, z) maka irisannya (common-vertices) = {y}
      (x, y: node node; y, z: node node)

Apabila nilai i = j maka dilewatkan saja

↳ maksudnya, apabila kedua edge yg dievaluasi sama maka dilewatkan saja tahap ini.

↳ Lalu juga dicek, apabila colors[i] = colors[j], maksudnya dilakukan pengecekan apabila warna edge yg intersect tsb sama atau tidak. Apabila sama, maka harus dibedakan / color harus ditambahkan sampai tidak ada warna intersect element yg sama $(\forall x (x \in \ldots))$
$\ldots \cap x \neq color$ (colors_intersect)

↳ 1. Apabila ada yg sama maka color ditambah lalu diulang iterasi i sampai ditemukan color unik untuk node tsb. (tidak pernah diulang)
2. Apabila tidak ada sampai j iterasi maka karena Flag = False, maka tandai sebagai iterasi i selesai lalu lanjut pada iterasi i+1

- ulangi proses diatas sampai semua elemen pada colors (panjang = edgelist) terisi dengan angka bilangan bulat positif.

Algoritma diatas merupakan algoritma greedy. Algoritma greedy adalah algoritma dimana memilih keputusan optimal local yg berharap untuk mendapatkan keputusan optimal global. Pada algoritma ini, optimal lokal adalah pada pemilihan color terkecil yg belum pernah dilakukan pemetaan colornya pada node yg intersected. Hal ini dilakukan seperti pada cara diatas, melakukan pemetaan color = 1 pada edge sekarang lalu memastikan apakah edge yg berhubungan punya nilai yg sama atau tidak. Apabila ada yg sama maka color ditambah dan proses diulang. Hal ini adalah approach greedy karena kita selalu memilih color selanjutnya yg terkecil yg belum pernah dipakai / digunakan edge yg berhubungan.

5 c) Algoritma diatas tidak selalu memberikan hasil pewarnaan yg minimal. Hal ini pertama karena sifat dari algoritma greedy yg tidak selalu menghasilkan solusi yg optimal, karena algoritma ini hanya menggunakan color terkecil yg belum dipetakan pada suatu node. Namun Hal ini dapat dilihat dari sifat algoritma ini karena pendekatan ini tidak melakukan analisis pada seluruh graf. Hal ini dapat direpresentasikan urutan pewarnaan edge BERPENGARUH terhadap hasil akhir pewarnaan. Misalnya, nilai pada edgelist dapat di representasikan sebagai beberapa urutan berbeda seperti (3,6) terlebih dahulu dari (6,4). Hal ini akan mempengaruhi pewarnaan berdasarkan urutan edgelist

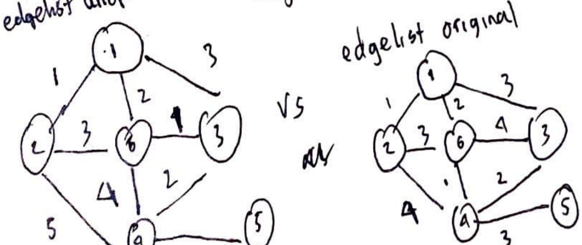
Sebagai contoh: apabila kita bandingkan edgelist awal yg menghasilkan: [1, 2, 2, 3, 1, 4, 2, 4, 3]
dengan edgelist: [(1,2), (1,6), (1,3), (2,6), (6,4), (3,6), (3,4), (2,4), (4,5)]
maka edgelist adalah

Namun apabila edgelist kita ditukar (6,4) dan (3,6)
= [(1,2), (1,6), (1,3), (2,6), (3,6), (6,4), (3,4), (2,4), (4,5)]

akan menghasilkan colors: [1, 2, 3, 3, 1, 4, 2, 5, 1]
banyak pewarnaan 5, padahal sebelumnya 4.

Sehingga, algoritma ini tidak selalu memberikan hasil pewarnaan yg minimal. Karena urutan pewarnaan penting dan tidak sama untuk setiap kemungkinan.

edgelist unoptimal vs edgelist original



Pewarnaan maks = 5 untuk edgelist baru namun pada edgelist original pewarnaan maks = 4