دانشگاه اصفهان
دانشکده علوم ریاضی و کامپیوتر

Design and Analysis of Algorithms

طراحی و تحلیل و الگوریتم ها

محمد ملائی

عنوان:

تمرینات ۲

نیم‌سال دوم ۱۴۰۲-۱۴۰۳

نام استاد درس

جعفر الماسی زاده

تمرین ۱

الف) برنامهای برای پیادهسازی دو الگوریتم سادهاندیشانه (تعریف - مبنا) و الگوریتم کاراتسوبا برای محاسبه حاصلضرب دو عدد صحیح رقمی بنویسید.

ب) الگوریتم کاراتسوبا از نظر مجانبی، کاراتر از الگوریتم سادهاندیشانه (تعریف - مبنا) است، اما در عمل، طبق انتظار تا نقطهای (مقداری از n) اجرای الگوریتم سادهاندیشانه سریعتر از اجرای الگوریتم کاراتسوبا است و از آن نقطه به بعد، اجرای الگوریتم کاراتسوبا سریعتر از اجرای الگوریتم سادهاندیشانه است. با انجام آزمایشی، «نقطه عبور» از اجرای سریعتر الگوریتم سادهاندیشانه به اجرای سریعتر الگوریتم کاراتسوبا را تعیین کنید.

جواب

الف

برای پیادهسازی الگوریتم سادهاندیشانه حاصل ضرب دو عدد صحیح A و B، طبق تعریف باید هر رقم B را در هر رقم A ضرب کنیم و سپس طبق اندازهشان یا به تعبیر دیگر جایگاهشان با هم جمع میکنیم تا جواب حاصلضرب را پیدا کنیم. پیادهسازی این الگوریتم به شکل زیر خواهد بود :

```python
def brute_force_multiply(x, y):
    strx = list(reversed(str(x)))
    stry = list(reversed(str(y)))
    y_length = len(stry)
    value = 0
    for n in range(len(strx)):
        xn = int(strx[n])
        for m in range(y_length):
            ym = int(stry[m])
            power = 10 ** (n + m)
            value += xn * ym * power
    return value
```

در اینجا ابتدا اعداد را به رشته تبدیل میکنیم تا استفاده از ارقام آن راحتتر باشد. برعکس کردن رشتهها به این دلیل است که اندیس هر رقم و ارزش جایگاهش یکسان باشند تا از انجام محاسبات اضافی جلوگیری شود.

برای پیاده سازی الگوریتم کاراتسوبا نیز کافیاست مقادیر c_0، c_1 و c_2 را محاسبه کرده و آنهارا طبق ارزششان باهم جمع میکنیم

```python
def karatsuba(a, b):
    if a < 10 and b < 10:
        return a * b
    else:
        a_str = str(a)
        b_str = str(b)
        n = max(len(a_str), len(b_str))
        a_str = "0" * (n - len(a_str)) + a_str
        b_str = "0" * (n - len(b_str)) + b_str
        m = n // 2
        a0 = int(a_str[m:])
        a1 = int(a_str[:m])

        b0 = int(b_str[m:])
        b1 = int(b_str[:m])

        c2 = karatsuba(a1, b1)
        c0 = karatsuba(a0, b0)
        c1 = karatsuba(a1 + a0, b1 + b0) - c2 - c0

        return c2 * 10 ** (2 * (n - m)) + c1 * 10 ** (n - m) + c0
```

در اینجا نیز مانند قسمت قبل ابتدا اعداد را به رشته تبدیل می‌کنیم تا کار با آنها راحت‌تر باشد سپس آنها را از نظر تعداد ارقام یکسان می‌کنیم تا هنکام تقسیم کردن آنها به دوقسمت، اعداد را با ترتیب درست در هم ضرب کنیم. پس محاسبه c_0، c_1 و c_2 کافیست حاصل را همانطور که در الگوریتم آمده است بدست آوریم. به جای اینکه در توان ها از m استفاده کنیم، به دلیل اینکه m می‌تواند عددی فرد باشد، از $n - m$ استفاده می‌کنیم زیرا تعداد صفرهای قسمت بزرگتر برابر این عدد است.

ب

با بررسی زمان اجرای الگوریتم‌ها می‌توان دید که در اعداد 32 رقمی و بزرگتر، الگوریتم کاراتسوبا از الگوریتم ساده‌اندیشانه پیشی می‌گیرد. در زیر نمونه‌ای از خروجی برنامه است که نقطه عبور را نشان می‌دهد:

```
25 False - BruteForce: 0.00023828506469726564 - Karatsuba: 0.0002685451507568359
26 False - BruteForce: 0.0002584576606750488 - Karatsuba: 0.00028472185134887696
27 False - BruteForce: 0.00028037309646606446 - Karatsuba: 0.0003039193153381347 7
28 False - BruteForce: 0.000302121639251709 - Karatsuba: 0.00032413721084594726
29 False - BruteForce: 0.000326075553894043 - Karatsuba: 0.000344121456146240 2
30 False - BruteForce: 0.0003538823127746582 - Karatsuba: 0.0003652334213256836
31 False - BruteForce: 0.00037674427032470705 - Karatsuba: 0.00038497209548950197
32 True - BruteForce: 0.00040317773818969725 - Karatsuba: 0.0003983688354492188
33 True - BruteForce: 0.0004382157325744629 - Karatsuba: 0.00043043136596679685
34 True - BruteForce: 0.00046774864196777346 - Karatsuba: 0.00044938564300537107
35 True - BruteForce: 0.0005058121681213379 - Karatsuba: 0.00047495365142822263
36 True - BruteForce: 0.0005348515510559082 - Karatsuba: 0.0004929041862487793
37 True - BruteForce: 0.0005624246597290039 - Karatsuba: 0.0005095911026000977
38 True - BruteForce: 0.0006023669242858887 - Karatsuba: 0.0005268192291259765
39 True - BruteForce: 0.0006373739242553711 - Karatsuba: 0.0005588960647583007
40 True - BruteForce: 0.0006713724136352539 - Karatsuba: 0.0005788040161132812
```

تمرین ۲

رستم (بازیکن ۱) و اسفندیار (بازیکن ۲) قرار است به یک بازی راهبردی دو نفره با این قوانین بپردازند: عددی زوج است و سکه با ارزشهای یکسان یا متفاوت در یک ردیف گذاشته شده‌اند هر دو بازیکن، به نوبت و هر بار یکی از سکهها را از یکی از دو انتهای (راست یا چپ) ردیف باقیمانده سکهها برمیدارند. بازیکنی که مجموع ارزش سکههایی که برداشته باشد بیشتر باشد، در نهایت برنده بازی خواهد بود. بازی را از چشمانداز بازیکن ۱ ببینید و فرض کنید راهبرد هر دو بازیکن در انتخاب سکه بهینه باشد؛ بهینه به این معنا که رستم در هر حرکت خود، سکهای را انتخاب می‌کند تا در نهایت بیشترین مبلغ ممکن از سکهها نصیب او شود؛ و اسفندیار در هر حرکت خود، سکهای را انتخاب می‌کند تا در نهایت کمترین مبلغ ممکن از سکهها نصیب رستم شود.

الف الگوریتمی برای تعیین راهبرد بهینه رستم (بازیکن ۱) برای بازی ارائه کنید.

ب اجرای گامبهگام الگوریتم را با در نظر گرفتن این ردیف سکهها نشان دهید:

$$2, 5, 8, 6, 9, 2, 10, 5, 7, 4$$

جواب

الف

برای محاسبه اینکه بازیکن ۱ چه سکه‌هایی را در هر مرحله بردارد تا در نهایت بیشترین جایزه را برنده شود می‌توانیم از راهبرد برنامه‌ریزی پویا (*dynamic programming*) استفاده‌کنیم. ابتدا باید تابع بازگشتی‌ای را تعریف کنیم که با گرفتن اندیس ابتدایی و انتهایی سکه‌ها به ما حداکثر جایزه‌ای را که بازیکن ۱ می‌تواند در یافت کند را برگرداند. پیش از ارائه این تابع، باید به این نکته توجه کنیم که بازیکن دوم نیز در تلاش برای بردن این بازی است :

$$f(m,n) = s - min\{f(m+1,n), f(m,n-1)\}$$

در اینجا s مجموع ارزش همه‌ی سکه‌هایی‌است که شماره آنها از m تا n است. تابع بدین شکل تعریف شده‌است که اگر بازیکن دوم کم‌ارزش ترین مجموع سکه‌ها را بردار آنگاه باقیمانده‌ی سکه‌ها با ارزش‌ترین مجموع را دارند که آنها را بازیکن اول برمیدارد. پیاده‌سازی این تابع بازگشتی به شکل زیر خواهد بود:

```
def max_value(list, start, end):
    if start == end:
        return list[start]
    coin_values = sum(list[start : end + 1])
    return coin_values - min(
        max_value(list, start + 1, end),
        max_value(list, start, end - 1),
    )
```

این تابع برای مثال قسمت ب مقدار 36 را برمی‌گرداند که درست است. می‌توانیم یا استفاده‌از راهبرد برنامه‌ریزی پویا و کاهش زمان اجرا در ازای مصرف حافظه علاوه بر سریع‌تر پیدا کردن مقدار جواب، خود جواب را نیز پیدا کنیم.

از آنجایی که در تابع به مقدار سطر بعد احتیاج داریم، ساخت جدول را از سطر آخر شروع می‌کنیم و دو شرط برای ساختن این جدول تعیین می‌کنیم تا به درستی ساخته شود: شرط اول اینست که اگر اندیس ابتدا از اندیس انتها بزرگتر باشد، مقدار این خانه از جدول صفر خواهد بود و شرط دوم اینست که اگر درایه، یک درایه قطری باشد یعنی تنها یک سکه برای برداشتن وجود دارد و مقدار بهینه همان ارزش این سکه خواهد بود و اگر شرطها برقرار نبودند، آنگاه مقادیر را با استفاده از شرط تابع پیدا می‌کنیم:

```python
def coin_values_table(A: list):
    D = [[0] * len(A) for i in range(len(A))]
    for i in range(len(A) - 1, -1, -1):
        for j in range(len(A)):
            if i > j:
                D[i][j] = 0
            elif i == j:
                D[i][j] = A[i]
            else:
                s = sum(A[i : j + 1])
                D[i][j] = s - min(D[i + 1][j], D[i][j - 1])
    return D

```

حال کافیست با استفاده از این جدول حرکت‌های هر دو بازیکن را برای گرفتن بیشترین جایزه پیدا کنیم. بدین صورت که اگر با برداشتن سکه اول، حریف جایزه کمتری می گیرد آنرا برمی‌داریم در غیر اینصور سکه آخر را برمی‌داریم و دامنه سکه‌ها را بروز کرده و دوباره این مقایسه را انجام می‌دهیم حرکت‌هایی با اندیس زوج (اگر از صفر شروع کنیم)، حرکت‌های بازیکن اول و با اندیس فرد حرکت‌های بازیکن دوم خواهند بود:

```python
def get_moves(A, D):
    moves = []
    domain = (0, len(A) - 1)
    for _ in range(len(A)):
        first_coin = D[domain[0] + 1][domain[1]]
        last_coin = D[domain[0]][domain[1] - 1]
        if first_coin < last_coin:
            moves.append(A[domain[0]])
            domain = (domain[0] + 1, domain[1])
        else:
            moves.append(A[domain[1]])
            domain = (domain[0], domain[1] - 1)
    first_player = [moves[i] for i in range(0, len(moves), 2)]
    second_player = [moves[i] for i in range(1, len(moves), 2)]
    return sum(first_player), first_player, second_player

```

ب

با اجرای برنامه‌ای که روند ساخت آن در قسمت قبل توضیح داده‌شد. حرکات این دو بازیکن به شکل زیر خواهند بود:

$$first\_player = [2, 7, 10, 9, 8]$$

$$second\_player = [4, 5, 2, 6, 5]$$

تمرین ۳

این مسأله کاربردی در حوزه تحلیل دادهها را در نظر بگیرید: مجموعه S از n نقطه در صفحه را به قسمی به دو مجموعه A و B افراز کنید که فاصله اقلیدسی هر دو نقطهای در مجموعه A، کمتر یا مساوی با فاصله اقلیدسی هر نقطهای در مجموعه A و هر نقطهای در مجموعه B باشد (و بالعکس، فاصله اقلیدسی هر دو نقطهای در مجموعه B، کمتر یا مساوی با فاصله اقلیدسی هر نقطهای در مجموعه A و هر نقطهای در مجموعه B باشد).

الف الگوریتمی کارا برای این مسأله طراحی کنید.

ب الگوریتم را با شبهکد توصیف کنید و کارایی زمانی آن را تعیین کنید.

جواب

الف

برای حل این مسئله می‌توانیم از الگوریتم‌هایی استفاده کنیم که درخت پوشای کمینه متناظر با گراف را که در اینجا یک گراف کامل است پیدا می‌کنند. روند حل سوال بدین صورت است که یالی را که بیشترین وزن را دارد (با فرض اینکه وزن یال نشان دهنده فاصله بین دو راس است) از درخت حذف می‌کنیم و جنگل به‌دست آمده را که متشکل از دو درخت است به عنوان افرازی از این نقاط که شرایط مسئله را دارد معرفی می‌کنیم. به تعبیری دیگری برای اینکه این n نقطه را به k مجموعه افراز کنیم، کافیست در زمان اجرای الگوریتم $kruskal$ و زمانی که تعداد درخت‌های همبند به k رسید، اجرای الگوریتم را متوقف کنیم زیرا مولفه‌های همبندی ساخته شده توسط این الگوریتم شرط مسئله را دارند. شبه‌کد الگوریتم بصورت زیر خواهد بود :

2-Partition($S = \{s_0, s_1, \ldots, s_n\}$)

Input: A non-emtpy set S of n points
Output: 2 clusters of points in A
  Set points and their distances as graph G $= < V, E >$
  Create the minimum spanning tree of G as T $= < V', E' >$
  Remove the most weighted edge of T
  Put the vertices of this two tree into sets A and B
  return A, B

اگر درخت پوشای کمینه را با الگوریتم $kruskal$ محاسبه کنیم، زمان اجرای الگوریتم $O(|E|log(|E|)) = O(n^2log(n^2))$ خواهد بود.

تمرین ۴

فرض کنید که n رایانه در یک شبکه محلی سیمی، به شکل درخت ریشهدار T چیده شده باشند؛ یعنی گره‌های درخت T ، نشان رایانههای شبکه هستند و یالهای درخت زوج رایانههایی را مشخص میکنند که مستقیماً با کابل به هم وصل شدهاند. ریشه درخت T ، نشان رایانهای است که باید به اینترنت وصل باشد. مدیر شبکه دغدغه ارتباطات امن در شبکه را دارد و میخواهد با خرید تعدادی نرمافزار ناظر، پیوسته کیفیت ارتباطات در شبکه را بررسی کند. اگر یک نرمافزار ناظر روی رایانه x نصب شود، کاربر x میتواند همه ارتباطات مستقیم خود با دیگر رایانهها را نظارت کند. مدیر برای صرفهجویی بیشتر در هزینههای خرید و ارتقای نرمافزار، از شما خواسته است که کمترین تعداد نرمافزار لازم را برای آنکه حداقل یک نرمافزار ناظر بر هر خط ارتباطی در شبکه نظارت کند، تعیین کنید. برای مثال، اگر هر رایانه در T (جز ریشه) فرزند ریشه باشد، پس مدیر فقط به یک ناظر نیاز خواهد داشت که باید روی رایانه ریشه نصب شود.

الف الگوریتمی کارا برای مسأله مدیر شبکه طراحی کنید که با آن بتوان از روی ساختار درختی یک شبکه رایانهای، کمترین تعداد نرمافزار لازم را محاسبه کرد و رایانههایی را که نرمافزار باید روی آنها نصب شود، تعیین کرد

ب بر مبنای الگوریتم، برنامه‌ای برای حل رایانهای مسأله بنویسید. برای آنکه بتوانید درستی برنامه خود را به طور دستی بیازمایید، سه درخت ۲۰ گرهای تولید کنید و برنامه خود را روی آن ورودیها اجرا کنید. نهایتاً آن سه درخت را به طور دستی نیز بکشید و گرههای جواب را علامت بزنید.

جواب

برای اینکه تشخیص دهیم که روی یک کامپیوتر در این شبکه نرم افزار نظارتی را نصب کنیم یا نه می‌توانیم از این نکات استفاده کنیم که اگر تمام فرزندان یک گره نظارت شوند و حداقل روی یکی از آنها این نرم افزار نصب شده باشد، احتیاجی به نصب نرم افزار روی این کامپیوتر نیست یا به عبارتی اگر حداقل یکی از فرزندان این گره تحت نظارت نباشد باید روی خود گره این نرم افزار را نصب کنیم تا این فرزند یا فرزندان را بتوانیم نظارت کنیم یا اگر همه‌ی فرزندان تحت نظارت باشند اما نرم‌افزار روی هیچ‌یک از آنها نصب نشده باشد، در اینصورت برای اینکه خود گره را تحت نظارت داشته باشیم، باید نرم افزار را روی خودش نصب کنیم.

با توجه به این دو شرط تعداد رایانه‌هایی که لازم است روی آنها نرم افزار را نصب کنیم به صورت زیر است:

$$f(T) = f(T_0) + f(T_1) \ldots + f(T_n) + max\{x, y\} \quad (۱)$$

$$x = 1 \; if \; \exists a \in children(T) : monitor(a) = 0 \; else \; 0 \quad (۲)$$

$$y = 1 \; if \; \forall a \in children(T) : software(a) = 0 \; else \; 0 \quad (۳)$$

$$(۴)$$

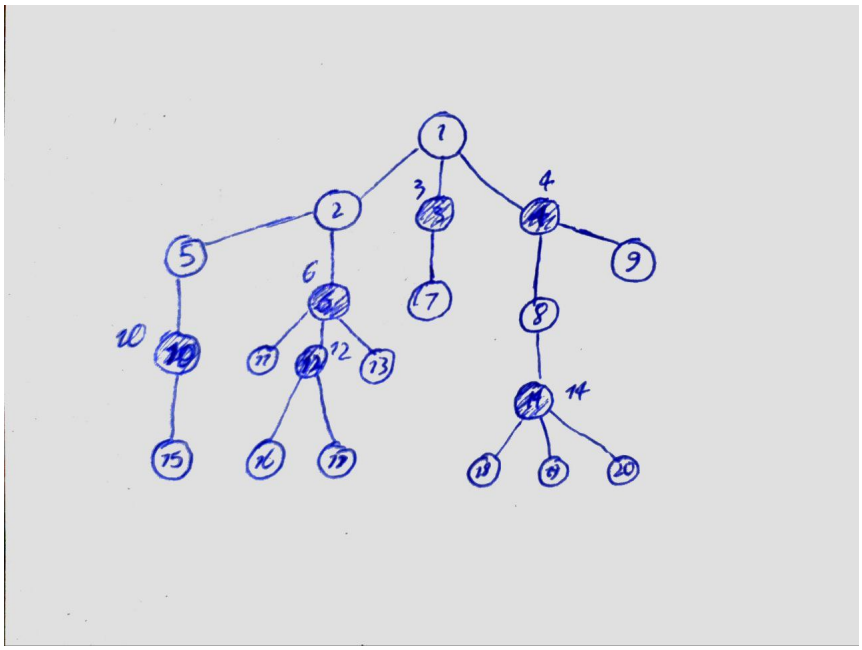
البته باید به این نکته توجه داشته باشیم که روی هیچ‌یک از برگ‌ها این نرم افزار نصب نخواهد شد زیرا می‌توان آن‌را روی گره پدرشان نصب کرد. می‌توانیم با برچسب زدن هر گره و فرزندانش در هر مرحله از پیمایش پس‌ترتیب درخت، اطلاعاتی که احتیاج داریم را بدست آوریم. ابتدا هر گره را به صورت زیر تعریف می‌کنیم تا اطلاعات موردنیاز را ذخیره کند:

```python
class Node:
    def __init__(self, name=None, parent=None) -> None:
        self.parent = parent
        self.name = name
        self.children: list["Node"] = []
        self.is_installed = None
        self.is_monitored = None

    def add_child(self, *nodes: "Node") -> None:
        for node in nodes:
            node.parent = self
        self.children.extend(nodes)

    def remove_child(self, node: "Node") -> None:
        self.children.remove(node)
```

حال با اجرای تابع زیر می‌توانیم حداقل تعداد نرم‌افزار های موردنیاز و گره‌هایی که این نرم افزار باید روی آنها نصب شود را پیدا می‌کنیم، تابعی که از روی تابع بازگشتی بالا درست شده است و شرط‌ها را برای هر گره بررسی می‌کند.

```python
def process_network_tree(root: Node):
    if len(root.children) == 0:
        if root.parent is None:
            return 1, [root.name]
        else:
            root.is_installed = False
            root.is_monitored = False
            return 0, []

    sofware_nodes = []
    softwares = 0

    for child in root.children:
        count, nodes = process_network_tree(child)
        sofware_nodes += nodes
        softwares += count

    x = any([not child.is_monitored for child in root.children])
    y = all([not child.is_installed for child in root.children])
    if x or y:
        root.is_installed = True
        root.is_monitored = True
        sofware_nodes = [root.name] + sofware_nodes
        softwares += 1
        root.parent.is_monitored = True
        for child in root.children:
            child.is_monitored = True

    return softwares, sofware_nodes
```

ب



تمرین ۵

فرض کنید که ساختار یک شبکه تلفن را بتوان به شکل گراف $G =< V, E >$ تصور کرد که هر رأس آن، نشان یک مرکز هادی و هر یال آن، نشان یک خط ارتباطی موجود بین دو مرکز هادی باشد؛ و هر یال گراف با پهنای باند خط ارتباطی متناظر با آن (که حداکثر سرعتی است برحسب بیت بر ثانیه، که دادهها را می‌توان در امتداد آن خط ارتباطی انتقال داد) برچسب خورده باشد. پهنای باند هر مسیری در گراف G را پهنای باند یالی از آن مسیر که کمترین پهنای باند را داشته باشد، تعریف میکنیم.

الف الگوریتمی کارا برای این مسأله طراحی کنید که با آن بتوان از روی ساختار گرافی یک شبکه تلفن، حداکثر پهنای باند مسیرهای بین هر دو مرکز هادی را محاسبه کرد و به شکل یک ماتریس نمایش داد.

ب بر مبنای الگوریتم، برنامه‌ای برای حل رایانهای مسأله بنویسید. برای آنکه بتوانید درستی برنامه خود را به طور دستی بیازمایید، سه گراف وزن‌دار خلوت ۲۰ رأسی تولید کنید و برنامه خود را روی آن ورودیها اجرا کنید. نهایتاً آن سه گراف را به طور دستی نیز بکشید و سه ماتریس خروجی برنامه را نیز در کنار آنها بگذارید.

جواب

الف

این مساله را می‌توان به مسئله کوتاه‌ترین مسیر بین هر دو نقطه گراف تشبیه کرد که آن را با استفاده از الگوریتم فلوید حل می‌کنیم. می‌توانیم با تغییر شرط آن به الگوریتمی دست‌یابیم که این مسئله را برای ما حل کند.

فرض می‌کنیم در روند حل مسئله و در مرحله k ام الگوریتم فلوید هستیم، اگر از راس i به راس j بدون گذر از راس k مسیری باشد، پهنای باند آن‌را $b_{ij}^{(k-1)}$ می‌نامیم. حال اگر با استفاده از راس k به عنوان راس میانی، مسیری از راس i به j پیدا کنیم، پهنای باند آن برابر است با مسیری که پهنای باند آن کمتر است یا $min\{b_{ik}^{(k-1)}, b_{kj}^{(k-1)}\}$ بنابراین حداکثر پهنای باند میان این دو راس در مرحله k ام بصورت زیر است:

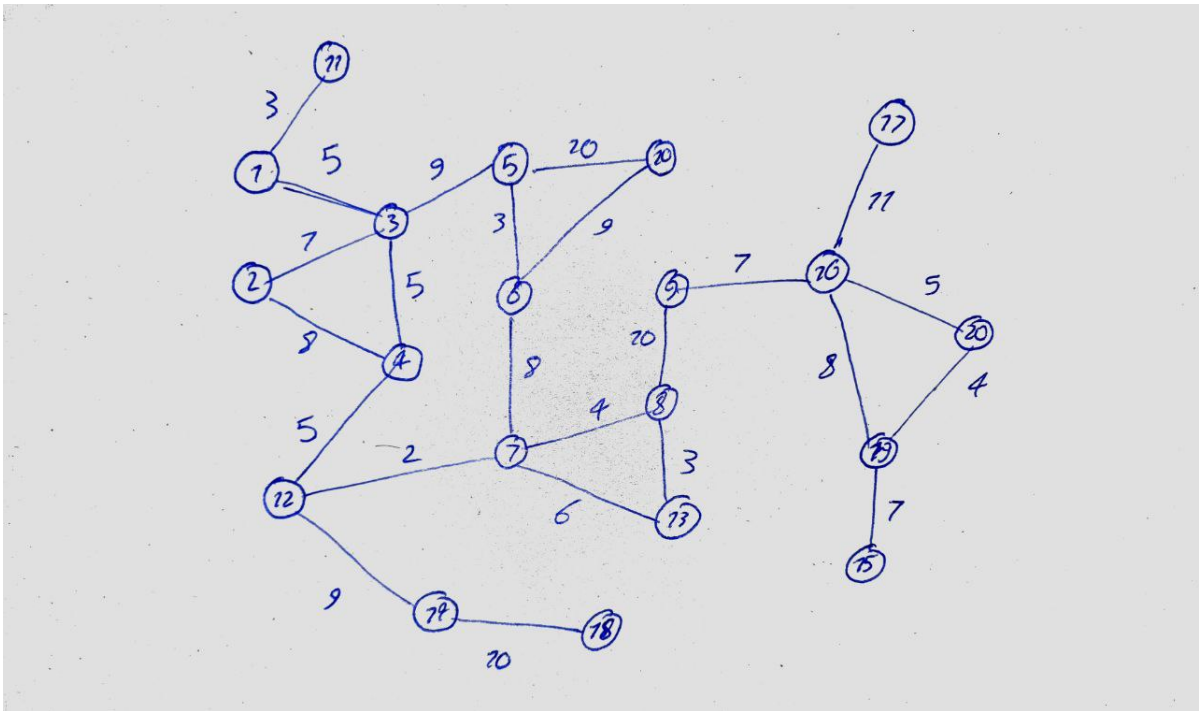
$$b_{ij}^{(k)} = max\{\ b_{ij}^{(k-1)},\ min\{b_{ik}^{(k-1)}, b_{kj}^{(k-1)}\}\ \}$$

حال با توجه به این شرط و الگوریتم فلوید، پیاده سازی این الگوریتم به شکل زیر خواهد بود:

```
def max_bandwith(table):
    for k in range(n):
        for i in range(n):
            for j in range(n):
                table[i][j] = max(D[i][j], min(table[i][k], table[k][j]))
```

برای بدست آوردن مقادیر صحیح، پهنای باند هر راس و خودش را بی‌نهایت و پهنای باند دو راس که مسیری بینشان نیست را صفر در نظر می‌گیریم. از طرفی می‌توانیم با استفاده از الگوریتم دایکسترا، این مقادیر را با تغییر شرط الگوریتم و برای هر گره جداگانه محاسبه و در ماتریس ذخیره کنیم.

ب



```
[inf, 5, 5, 5, 5, 5, 5, 4, 4, 5, 3, 5, 5, 5, 4, 4, 4, 5, 4, 4]
[5, inf, 7, 8, 7, 7, 7, 4, 4, 7, 3, 5, 6, 5, 4, 4, 4, 5, 4, 4]
[5, 7, inf, 7, 9, 9, 8, 4, 4, 9, 3, 5, 6, 5, 4, 4, 4, 5, 4, 4]
[5, 8, 7, inf, 7, 7, 7, 4, 4, 7, 3, 5, 6, 5, 4, 4, 4, 5, 4, 4]
[5, 7, 9, 7, inf, 9, 8, 4, 4, 10, 3, 5, 6, 5, 4, 4, 4, 5, 4, 4]
[5, 7, 9, 7, 9, inf, 8, 4, 4, 9, 3, 5, 6, 5, 4, 4, 4, 5, 4, 4]
[5, 7, 8, 7, 8, 8, inf, 4, 4, 8, 3, 5, 6, 5, 4, 4, 4, 5, 4, 4]
[4, 4, 4, 4, 4, 4, 4, inf, 10, 4, 3, 4, 4, 4, 7, 7, 7, 4, 7, 5]
[4, 4, 4, 4, 4, 4, 4, 10, inf, 4, 3, 4, 4, 4, 7, 7, 7, 4, 7, 5]
[5, 7, 9, 7, 10, 9, 8, 4, 4, inf, 3, 5, 6, 5, 4, 4, 4, 5, 4, 4]
[3, 3, 3, 3, 3, 3, 3, 3, 3, 3, inf, 3, 3, 3, 3, 3, 3, 3, 3, 3]
[5, 5, 5, 5, 5, 5, 5, 4, 4, 5, 3, inf, 5, 9, 4, 4, 4, 9, 4, 4]
[5, 6, 6, 6, 6, 6, 6, 4, 4, 6, 3, 5, inf, 5, 4, 4, 4, 5, 4, 4]
[5, 5, 5, 5, 5, 5, 5, 4, 4, 5, 3, 9, 5, inf, 4, 4, 4, 10, 4, 4]
[4, 4, 4, 4, 4, 4, 4, 7, 7, 4, 3, 4, 4, 4, inf, 7, 7, 4, 7, 5]
[4, 4, 4, 4, 4, 4, 4, 7, 7, 4, 3, 4, 4, 4, 7, inf, 11, 4, 8, 5]
[4, 4, 4, 4, 4, 4, 4, 7, 7, 4, 3, 4, 4, 4, 7, 11, inf, 4, 8, 5]
[5, 5, 5, 5, 5, 5, 5, 4, 4, 5, 3, 9, 5, 10, 4, 4, 4, inf, 4, 4]
[4, 4, 4, 4, 4, 4, 4, 7, 7, 4, 3, 4, 4, 4, 7, 8, 8, 4, inf, 5]
[4, 4, 4, 4, 4, 4, 4, 5, 5, 4, 3, 4, 4, 4, 5, 5, 5, 4, 5, inf]
```