دانشگاه اصفهان
دانشکده علوم ریاضی و کامپیوتر

Design and Analysis of Algorithms

طراحی و تحلیل و الگوریتم ها

محمد ملائی

عنوان:

تمرینات ۳

نیم‌سال دوم ۱۴۰۲-۱۴۰۳

نام استاد درس

جعفر الماسی زاده

تمرین ۱

الف) برنامه‌ای برای پیاده‌سازی الگوریتم گاوس برای حل دستگاه‌های معادلات خطی n معادله در n مجهول بنویسید. مجری برنامه باید بتواند با اجرای برنامه، یا جواب یکتای یک دستگاه معادلات خطی را بیابد یا آنکه پی ببرد که دستگاه جواب ندارد یا آنکه پی ببرد که دستگاه بیشمار جواب دارد.

ب) با اجرای برنامه روی ماتریس‌های مختلف و تحلیل داده‌های خروجی، این را تحقیق کنید که رده کارایی زمانی الگوریتم $O(n^3)$ است.

جواب

الف

برای حل این سوال و پیاده‌سازی الگوریتم گاوس، از شبه کد ارائه شده در کتاب استفاده می‌کنیم با این تفاوت که به جای محور گیری جزئی، از محور گیری کلی استفاده می‌کنیم. دلیل این کار با توجه به اینکه هزینه محاسباتی آن بیشتر است اینست که مدیریت صفرهای ماتریس را آسان‌تر می‌کند و خطاهای برنامه را کاهش می‌دهد.

```python
def better_forward_eliminate(matrix):
    n = len(matrix)
    for i in range(n):
        full_pivot(matrix, i)
        for j in range(i + 1, n):
            if matrix[i][i] == 0:
                return
            co_effiecent = matrix[j][i] / matrix[i][i]
            for k in range(i, n + 1):
                matrix[j][k] -= co_effiecent * matrix[i][k]
```

در ادامه پیاده‌سازی محورگیری کلی آمده‌است که با پیدا کردن بزرگترین درایه زیرماتریس، سطر و ستون‌ها را جابجا می‌کند

```python
def full_pivot(matrix, k):
    n = len(matrix)
    max_position = (k, k)
    max_value = matrix[k][k]
    for i in range(k, n):
        for j in range(k, n):
            if abs(matrix[i][j]) > max_value:
                max_position = (i, j)
                max_value = abs(matrix[i][j])
    row, column = max_position
    for p in range(k, n + 1):
        matrix[k][p], matrix[row][p] = matrix[row][p], matrix[k][p]
    for p in range(k, n):
        matrix[p][k], matrix[p][column] = matrix[p][column], matrix[p][k]
```

در انتها با جایگذاری‌های پسرو جواب یا جواب‌های دستگاه معادلات را پیدا می‌کنیم. ذکر این نکته ضروری است که برای فهمیدن اینکه دستگاه جواب ندارد یا بی‌نهایت جواب دارد، کافی است دترمینان ماتریس ضرایب را بررسی کنیم، این کار را پس از اجرای الگوریتم گاوس و بررسی سطرهایی که همه‌ی درایه‌های آنها صفر باشد انجام می‌دهیم. اگر همه‌ی درایه‌های یک سطر صفر و درایه آخر ماتریس افزوده در همان سطر نیز صفر باشد بی نهایت جواب داریم و اگر صفر نباشد جوابی نداریم زیرا این بدین معنی است که صفر برابر با عددی دیگر است که تناقض است. در الگوریتم جایگذاری‌های پسرو ابتدا وجود تناقض را بررسی و سپس شرط بی‌نهایت بودن جواب‌ها را بررسی می‌کنیم:

```python
def backward_substitute(matrix):
    n = len(matrix)
    infinit_solutions = False
    for i in range(n):
        if all(matrix[i][j] == 0 for j in range(0, n - 1)):
            if matrix[i][n] == 0:
                return "No Solution"
            else:
                infinit_solutions = True
    if infinit_solutions:
        return "Infinite Solutions"
    x = [0] * len(matrix)
    for i in range(n - 1, -1, -1):
        m = 0
        for j in range(i + 1, n):
            m += matrix[i][j] * x[j]
        x[i] = (matrix[i][n] - m) / matrix[i][i]
    return x
```

ب

با توجه به زمان اجرای الگوریتم رو ماتریس‌هایی با اندازه‌های مختلف که درایه‌های آن‌ها بصورت تصادفی انتخاب شده است، کارایی $O(n^3)$ برای این الگوریتم درست است.

```
 n = 20, execution: 0.00131221
 n = 40, execution: 0.0058578
 n = 60, execution: 0.011902
 n = 80, execution: 0.02445478
 n = 100, execution: 0.04363322
 n = 120, execution: 0.07302938
 n = 140, execution: 0.11416078
 n = 160, execution: 0.16694379
 n = 180, execution: 0.23424001
 n = 200, execution: 0.3173264
 n = 220, execution: 0.41943684
 n = 240, execution: 0.54057956
 n = 260, execution: 0.6872694
 n = 280, execution: 0.86881361
```

تمرین ۲

جاده‌ای طولانی و مستقیم را در منطقه‌ای سرسبز تصور کنید که از نزدیکی شهری بزرگ میگذرد و در کناره‌های آن، خانه‌های مسکونی و فروشگاه‌هایی، دور از هم یا نزدیک به هم، قرار گرفته‌اند. موضوع این است که افراد بومی ساکن در خانه‌ها و شاغل در فروشگاه‌ها، جمعیت نسبتاً زیادی را تشکیل میدهند و روزانه و شبانه مسافران زیادی نیز از جاده گذر میکنند، ولی سطل زباله‌ای در کناره‌های جاده وجود ندارد و افراد بومی و مسافران حواشی جاده را پر از زباله کرده‌اند! شهرداری برای رفع این معضل، تصمیم گرفته است که تعدادی سطل زباله بزرگ را با فاصله‌های مناسب در کناره‌های جاده نصب کند. از آنجا که رهگذران میتوانند زباله‌های خود را در هر سطل زباله‌ای خالی کنند، شهرداری برای تشویق افراد بومی به تمیز نگه داشتن محل زندگی خود، میخواهد سطلهای زباله را در نقاطی نصب کند که افراد ساکن در هر ساختمان کناره جاده، حداکثر ۱ کیلومتر راه را برای رسیدن به نزدیک‌ترین سطل زباله به ساختمان خود بپیمایند. از طرف دیگر، شهرداری بودجهای آنچنانی برای خرید و نصب سطل‌های زباله ندارد و میخواهد از کمترین تعداد سطل زباله ممکن استفاده کند.

الف الگوریتمی را توصیف کنید که به عنوان ورودی، مجموعه نقاطی را که مکان‌های همه ساختمان‌های مستقر در کناره‌های جاده را مشخص میکنند، بگیرد و به عنوان خروجی، کمترین تعداد سطل‌های زباله لازم و نقطه نصب هر یک از آنها را بدهد. برای حل مسأله در حالت کلی، تعداد ساختمان‌ها را n بگیرید. درستی الگوریتم خود را ثابت کنید.

ب برنامه‌ای برای پیاده‌سازی الگوریتم خود بنویسید و با اجرای آن روی چند نمونه ورودی مختلف، درستی آن را تحقیق کنید.

جواب

الف

با فرض اینکه نقاط نشان‌دهنده ساختمان‌ها روی یک خط قرار دارند و می‌توان مختصات آنها را بصورت یک عدد حقیقی نشان داد، الگوریتم حریصانه‌ای را برای حل این مسئله به شکل زیر توصیف می‌کنیم:

ابتدا نقاط را مرتب می‌کنیم و سپس با شروع از اولین نقطه، نقاطی را که فاصله آنها تا این نقطه حداکثر ۲ است را یک مجموعه در نظر می‌گیریم که نشان دهنده این است که تنها یک سطل زباله می‌تواند شرط‌های مسئله را برای این مجموعه نقاط تامین کنید. البته با ذکر این نکته که مختصات آن دقیقا در وسط این مجموعه باشد. سپس مجموعه‌ای جدید در نظر می‌گیریم و همین روند برای برای نقاط باقیمانده ادامه می‌دهیم تا همه‌ی ساختمان‌ها به یک سطل زباله دسترسی داشته باشند.

```python
def min_trash_bins(points: list):
    points.sort()
    n = len(points)
    i = 0
    bins = []
    while i < n:
        p = points[i]
        j = 1
        while i + j < n and abs(p - points[i + j]) <= 2:
            j += 1
        bins.append((p + points[i + j - 1]) / 2)
        i = i + j
    return bins
```

برای اثبات درستی الگوریتم از استقرای ریاضی استفاده می‌کنیم. اگر فقط یک ساختمان داشته باشیم، الگوریتم مختصات همان ساختمان را به عنوان جایی که باید سطل زباله را نصب کنیم برمی‌گرداند که درست است. بنابر استقرای ریاضی فرض می‌کنیم الگوریتم برای n نقطه اول جواب درستی می‌دهد، نشان می‌دهیم با در نظر گرفتن نقطه $n+1$ نیز خروجی الگوریتم همچنان درست است.

اگر برای n نقطه اول k سطل زباله نصب کنیم که از ۱ شماره گذاری شده‌اند، P_{k0} را اولین نقطه در مجموعه k ام تعریف می‌کنیم و P_t را مختصات سطل زباله‌ای که به این مجموعه تعلق می‌گیرد تعریف می‌کنیم.

اگر $P_{n+1} - P_{k0} \leq 2$، الگوریتم این نقطه را به این مجموعه اضافه می‌کند و تغییری در تعداد سطل‌های زباله ایجاد نمی‌کند چرا که:

$$max\{P_t - P_{k0}\} = 1,\ max\{P_{n+1} - P_t\} = 1 \to max\{P_{n+1} - P_{k0}\} = 2$$

بنابراین در بدترین حالت که فاصله نقطه جدید و اولین نقطه ۲ است، کافی است سطل زباله را وسط این دو نقطه قرار دهیم. این کار روی سایر نقاط مجموعه تاثیری ندارد زیرا به دلیل مرتب بودن آنها فاصله‌شان از نقطه ابتدایی از نقطه جدید کمتر است و چون سطل زباله نقاط ابتدایی و انتهایی مجموعه را پوشش می‌دهد، آنهارا نیز پوشش خواهد داد.

اگر $P_{n+1} - P_{k0} > 2$، الگوریتم مجموعه جدیدی ایجاد خواهد کرد و این نقطه اولین نقطه آن خواهد بود. نمی‌توانیم این نقطه را به مجموعه قبلی اضافه کنیم زیرا نمی‌توانیم جایی برای سطل زباله بیابیم که شرط‌های مسئله را داشته باشد. نتیجه گیری بالا برای این قسمت نیز صادق است و احتیاجی به اثبات دوباره نیست چرا که اگر $max\{P_{n+1} - P_{k0}\} = 2$ با شرط این قسمت در تضاد است. بنابراین ثابت شد که الگوریتم جواب بهینه را به ما می‌دهد زیرا در صورت امکان نقاط را به مجموعه‌های قبلی اضافه می‌کند و تنها زمانی که ممکن نیست، مجموعه‌ای جدید ایجاد می‌کند.

ب

الگوریتم را روی ورودی‌های زیر بررسی می‌کنیم:

```
    [1, 1.2, 1.3, 2, 2.1, 3, 3.01, 5, 4.5, 3, 8]
    [2.01, 8, 6, 0.7, 4, 9, 0.3]
```

و داریم:

```
    [2.0, 4.005, 8.0]
    [1.1549999999999998, 5.0, 8.5]
```

که جواب‌های درست مسئله هستند.

تمرین ۳

مسأله ازدواج پایدار را به این شکل تعمیم میدهیم که تشکیل زوج‌های خاصی از مردها - زنها صریحاً ممنوع باشد. (در مورد تطابق کارفرماها و کارجوها، میتوانیم این گونه تصور کنیم که بعضی از کارجوها فاقد صلاحیت‌ها یا گواهی‌های لازم باشند و بنابراین، با وجود آنکه موجه به نظر میرسند، نتوانند در شرکتهای خاصی استخدام شوند.) پس ما یک مجموعه M شامل n مرد داریم و یک مجموعه W شامل n زن. و یک مجموعه $F \subseteq M \times W$ شامل زوج‌هایی که مجاز به ازدواج با یکدیگر نیستند. هر مرد m ، تمام زنهای w را با شرط $(m,w) \notin F$ رتبه‌بندی میکند و هر زن w' ، تمام مردهای m' را با شرط $(m',w') \notin F$ رتبه‌بندی میکند.

در این قالب کلیتر از مسأله ازدواج پایدار، ما میگوییم که یک تطابق ازدواج S پایدار است، اگر هیچ یک از این نوع ناپایداریها را نداشته باشد:

- دو زوج (m,w) و (m',w') در S وجود داشته باشند و با شرط $(m',w') \notin F$ ، مرد m ترجیح دهد زن w' را به w ، و زن w' ترجیح دهد مرد m را به m' . (این حالت، همان نوعِ عادی ناپایداری است.)

- زوج $(m,w) \in S$ باشد، اما یک مرد m' وجود داشته باشد که در هیچ زوجی از تطابق قرار نگرفته باشد، و با شرط $(m',w) \notin F$ ، w ترجیح دهد m' را به m . (در این حالت، زنی با مردی زوج شده است، ولی مردی مجرد را که ازدواج با او ممنوع نیست، به آن مرد ترجیح میدهد.)

- زوج $(m,w) \in S$ باشد، اما یک زن w' وجود داشته باشد که در هیچ زوجی از تطابق قرار نگرفته باشد، و با شرط $(m,w') \notin F$ ، m ترجیح دهد w' را به w . (در این حالت، مردی با زنی زوج شده است، اما زنی مجرد را که ازدواج با او ممنوع نیست، به آن زن ترجیح میدهد.)

- یک مرد m و یک زن w وجود داشته باشند که با شرط $(m,w) \notin F$ ، هیچ یک از آن دو در هیچ زوجی از تطابق قرار نگرفته باشند. (در این حالت، یک مرد مجرد و یک زن مجرد وجود دارند که مانعی برای ازدواج آنها با یکدیگر وجود ندارد.)

الف ثابت کنید که با این تعریف از ناپایداری یک تطابق ازدواج، میتوان با همان الگوریتمی که مسأله پایه‌ای ازدواج پایدار را حل میکند، این مسأله را نیز حل کرد. الگوریتم باید همیشه برای هر مجموعه‌ای از لیستهای ترجیحات مردان و زنان و هر مجموعه‌ای از زوجهای ممنوع، یک تطابق ازدواج پایدار تولید کند.

ب برنامه‌ای برای پیاده‌سازی الگوریتم بنویسید به نحوی که رده کارایی زمانی آن $O(n^2)$ باشد. با اجرای برنامه روی چند نمونه ورودی مختلف، درستی آن را تحقیق کنید.

جواب

الف

ابتدا با این تعریف از ناپایداری نشان می‌دهیم که خروجی الگوریتم پایدار خواهد بود. با اولین قسمت از تعرق شروع می‌کنیم و با استفاده از برهان خلف فرض می‌کنیم این دو زوج وجود دارند یعنی (تابع $P_x(y)$ را بدین صورت تعریف می‌کنیم که اندیس y در لیست ترجیحات x را برمی‌گرداند) :

$$\exists (m,w),(m',w') \in S : P_m(w') < P_m(w) \wedge P_{w'}(m) < P_{w'}(m') \quad (1)$$

و از آنجایی که $P_m(w') < P_m(w)$ می‌توانیم این نتیجه را بگیریم که طبق الگوریتم m از w' زودتر از w خواستگاری می‌کند اما طبق فرض می‌دانیم که $(m,w') \notin S$ بنابراین یا w' آزاد نبوده است و پارتنر خود را به m ترجیح داده است یا آزاد بوده است اما با پیدا کردن پارتنر بهتر m را آزاد کرده است. در هر صورت می‌توانیم نتیجه بگیریم که w' مرد m' را به m ترجیح می‌دهد و این تناقض است که نشان می‌دهد چنین اتفاقی رخ نخواهد داد.

در بخش دوم این تعریف نیز با برهان خلف فرض می‌کنیم که چنین مردی وجود دارد، یعنی:

$$(m, w) \in S \quad (۲)$$
$$\exists m' \in M \ \ \forall w' \in W : (m', w') \notin S \quad (۳)$$
$$P_w(m') < P_w(m) \quad (۴)$$

طبق الگوریتم اگر m' آزاد است یعنی m' به همه‌ی زن‌هایی که در لیست ترجیحاتش هستند پیشنهاد داده است و طبق فرض ۳ می‌دانیم که $(m', w) \notin S$، مانند قسمت قبل و از آنجایی که می‌دانیم حتما m' به w پیشنهاد داده است، می‌توانیم نتیجه بگیریم که w یا درخواست m' را رد کرده است یا او را آزاد کرده‌است که هردو به این معنا هستند که $P_w(m) < P_w(m')$ که تناقض است و نشان می‌دهد که این اتفاق نیز رخ نخواهد داد.

بخش سوم نیز به سادگی رد می‌شود چرا که اگر m، w' را ترجیح می‌داد به او زودتر پیشنهاد می‌داد و w' چون آزاد است، پیشنهاد او را قبول می‌کرد.

قسمت آخر نیز به سادگی رد می‌شود زیرا الگوریتم تا جایی ادامه پیدا می‌کند که هر مرد یا پارتنر خود را پیدا کرده باشد یا به تمام زن‌هایی که در لیست ترجیحاتش هستند پیشنهاد داده باشد و این بدین معنی است که اگر در بدترین حالت w آخرین نفر این لیست باشد، به دلیل آزاد بودن پیشنهاد m را خواهد پذیرفت و تنها نخواهد ماند.

ب

پیاده‌سازی این الگوریتم بسیار شبیه مسئله اصلی است با این تفاوت که ممکن است برای یکی از مردها یا زن‌ها زوجی پیدا نشود که تنها شرط توقف را تغییر می‌دهد.

```python
def extended_stable_matche(W, M):
    men_last_propose = {}
    w_partners = {}

    free_mens = list(M.keys())
    for man in free_mens:
        men_last_propose[man] = -1

    while len(free_mens) > 0:
        man = free_mens[0]
        i = men_last_propose[man]
        while i < len(M[man]) - 1:
            i += 1
            woman = M[man][i]
            woman_partner = w_partners.get(woman, None)
            if woman_partner is None:
                w_partners[woman] = man
                free_mens.remove(man)
                break
            elif prefers(W, woman, man, woman_partner):
                free_mens.append(woman_partner)
                free_mens.remove(man)
                w_partners[woman] = man
                break
        if i == len(M[man]) - 1:
            free_mens.remove(man)
        men_last_propose[man] = i
    return w_partners
```

در اینجا پس از حلقه $while$ درونی، اگر یک مرد به همه‌ی زن‌هایی که در لیست ترجیحاتش قرار دارند درخواست داده‌باشد، چون این موضوع که همه‌ی مردها با یک نفر ازدواج کنند تضمین شده نیست، او را از لیست مردان آزاد حذف می‌کنیم تا شرط توقف برقرار شود. در غیر اینصورت او هربار لیست مردان خالی نیست و برنامه توقف ناپذیر خواهد شد.

-stable matching problem-

تمرین ۴

در گونه‌ای از مسائل جریان شبکه، مقدار جریانی که باید از مبدأ به مقصد بفرستیم، از قبل مشخص شده است اما موضوع این است که ارسال جریان از هر رأس به رأسی دیگر هزینه‌های خواهد داشت. بنابراین، ما به دنبال «هدایت بهینه» جریانی با مقدار معلوم هستیم؛ به این معنا که هزینه ارسال آن از مبدأ به مقصد، حداقل مقدار ممکن باشد.
در اینجا نیز می‌توان شبکه انتقال مورد نظر را با گراف $G = < V, E >$ که گرافی جهتدار، همبند و وزندار است، نمایش داد. این گراف، n رأس دارد و رئوس آن از ۱ تا n شماره‌گذاری شده‌اند؛ دقیقاً یک رأسِ بدون یال ورودی دارد؛این رأس، مبدأ نامیده میشود و شماره آن ۱ است؛ دقیقاً یک رأس بدون یال خروجی دارد؛ این رأس، مقصد نامیده می‌شود و شماره آن n است.

هر یال جهتدار (i, j) گراف دو برچسب دارد:

- برچسب u_{ij} ، که یک عدد صحیح مثبت است و ظرفیت یال را مشخص میکند.
- برچسب c_{ij} ، که یک عدد حقیقی مثبت است و هزینه ارسال یک واحد جریان را از طریق آن یال (خط) مشخص می‌کند.

راه‌حلی الگوریتمی برای این مسأله بیان کنید که با آن بتوان در یک شبکه انتقال که ساختار آن و ظرفیت هر یک از خطوط آن و هزینه ارسال جریان از هر یک از خطوط آن معلوم باشد، جریانی با مقدار مشخص f را (در صورت امکان) با کمترین هزینه ممکن، از مبدأ به مقصد فرستاد.

جواب

این مسئله را می‌توانیم با تبدیل کردن آن به یک مسئله جریان بیشینه و سپس با استفاده از تکنیک تکرار و بهبود حل کنیم بدین صورت که ابتدا جریان بیشینه‌ای که میتوانیم از راس ۱ به راس n بفرستیم پیدا می‌کنیم، اگر مقدار این جریان کمتر از f باشد، این سوال جواب نخواهد داشت اما اگر برابر f باشد، یعنی حداقل یک جواب احتمالی برای این سوال داریم. پیدا کردن جواب مسئله جواب بیشینه را می‌توانیم با الگوریتم $ford - fulkerson$ پیدا کنیم که قبلا درباره آن بحث شده است.
پس از پیدا کردن این جواب باید با استفاده از تکنیک تکرار و بهبود جواب را تا جایی که امکان دارد بهبود ببخشیم تا جواب بهینه را پیدا کنیم. این روند بدین صورت خواهد بود که ما باید با استفاده از مسیر فعلی مسیری را پیدا کنیم که همین مقدار جریان را اما با هزینه‌ای کمتر به مقصد می‌رساند.
الگوریتم بدین صورت خواهد بود که باید به جای مسیرهای جریان افزا، دورهای جریان افزایی را پیدا کنیم که ارسال جریان از آن‌ها هزینه کمتری دارد یعنی نه تنها باید دور جریان افزا پیدا کنیم که به جای ارسال جریان از مسیر فعلی، از طرف دیگر این جریان را ارسال کنیم، باید این دور جریان افزا هزینه‌ی ارسال را کاهش نیز دهد. دلیل بهینه بودن این الگوریتم قضیه‌ای است که به همین موضوع اشاره می‌کند و اثبات آن خارج از موضوع این سوال خواهد بود:
با در نظر گرفتن گراف $G =< V, E >$ با u_{ij} که ظرفیت یال را مشخص می‌کند و cij که هزینه ارسال از این یال را مشخص می‌کند، جریان ارسال کمترین هزینه را دارد اگر و تنها اگر دور جریان افزایی که وزن آن منفی است وجود نداشته باشد.
شبه کد الگوریتم به صورت زیر خواهد بود:

MinCostFlow($G =< V, E >$, $U = \{u_{ij}\}$, $C = \{c_ij\}$)

Input: A graph G, its edge capacities and their cost
Output: Minimum Cost Flow

```
Find the maximum flow of graph G
Cost = cost of transferring flow through maximum flow answer
while there is an f-augmenting cycle C with negative total weight do
    Update Cost and flow of nodes and edges with new values
```

-YouTube-

تمرین ۵

فرض کنید میخواهیم در شبکه‌ای رایانه‌ای، از رایانه‌ای خاص به عنوان رایانه مبدأ، جریانی پیوسته از داده‌ها را با حداکثر سرعت ممکن به رایانه‌ای دیگر به عنوان رایانه مقصد منتقل کنیم. برای آنکه بتوانیم میزان بیشتری از داده‌ها را در واحد زمان ارسال کنیم، میتوانیم داده‌ها را به بسته‌هایی تقسیم کنیم و آن بسته‌ها را از طریق مسیرهای مختلف از مبدأ به مقصد بفرستیم. از طرف دیگر، برای ارسال داده‌ها هم با این قید مواجهیم که نمیتوان داده‌ها را با سرعتی بیشتر از یک مقدار مشخص از هر خط ارتباطی بین دو رایانه عبور داد (هر مسیریاب) عبور داد. این شبکه رایانه‌ای را میتوان با یک گراف‌جهتدارِ وزندار نمایش داد: هر رایانه عضو شبکه را میتوان رأسی از رئوس گراف در نظر گرفت و هر خط ارتباطیِ یک طرفه بین دو رایانه را میتوان یالی از یالهای جهتدار گراف دانست. پهنای باند هر خط ارتباطی (که حداکثر تعداد بایتهایی است که میتوان در یک ثانیه از آن خط عبور داد) در شبکه، وزن یک یال جهتدار در گراف را مشخص میکند و پهنای باند هر رایانه در شبکه، وزن یک رأس در گراف را مشخص میکند.

الف برنامه‌ای برای حل حالت کلی این مسأله بنویسید. این برنامه باید گراف جهتدارِ وزندار $G =< V, E >$ و دو رأس مبدأ s و مقصد t را بگیرد و حداکثر میزان داده‌هایی را که میتوان در واحد زمان از رایانه s به رایانه t منتقل کرد، و همچنین میزان بسته‌های ارسالی از هر یک از خطوط ارتباطی و از هر یک از رایانه‌های میانی شبکه را تعیین کند.

ب برای آنکه بتوانید درستی برنامه خود را به طور دستی بیازمایید، سه شبکه (گراف‌جهتدارِ وزندار) تولید کنید برنامه خود را روی آن ورودی‌ها اجرا کنید. نهایتاً آن سه شبکه را به طور دستی نیز بکشید و میزان بسته‌های ارسالی از هر یک از خطوط ارتباطی و از هر یک از رایانه‌های میانی شبکه را نیز روی آنها مشخص کنید.

جواب

الف

این مسئله بسیار به مسئله جریان بیشینه شبیه است و با استفاده از تکنیک تبدیل مسئله می‌توان آنرا حل کرد. روند تبدیل این مسئله بدین صورت است که به جای هر راس این گراف دو راس در گراف جدید قرار می‌دهیم به طوری که تمام یال‌هایی که به راس v وارد می‌شوند را به v_{in} وارد می‌کنیم و تمام راس‌هایی که از v خارج می‌شوند را از v_{out} خارج می‌کنیم و یک یال بین این دو راس قرار می‌دهیم که ظرفیت آن برابر با ظرفیت v است و با حل مسئله جدید، جواب مسئله اصلی را پیدا می‌کنیم.

ابتدا نحوه‌ی نمایش گراف را بصورتی انتخاب می‌کنیم که روند حل مسئله را برای ما راحت‌تر کند. اگر برای نمایش گراف از ماتریس استفاده کنیم، پس از تبدیل درایه‌های بسیار زیادی از ماتریس صفر خواهند بود که سرعت برنامه را کاهش خواهد و تبدیل مسئله و تبدیل نتیجه الگوریتم به جواب را سخت‌تر خواهد کرد بنابراین استفاده از لیست مجاورت و ذخیره اطلاعتی اضافی در آن بسیار آسان‌تر خواهد بود البته در اینجا برای سادگی کار و ذخیره این اطلاعت اضافی از $dictionary$ استفاده شده است. ساختار نمونه‌ای از یک گراف در $python$ به صورت زیر خواهد بود:

```python
g = {
    1: {"n": {2: (2, 0), 4: (3, 0)}, "f": (4, 0)},
    2: {"n": {3: (5, 0), 5: (3, 0)}, "f": (10, 0)},
    3: {"n": {6: (2, 0)}, "f": (3, 0)},
    4: {"n": {3: (1, 0)}, "f": (2, 0)},
    5: {"n": {6: (4, 0)}, "f": (3, 0)},
    6: {"n": {}, "f": (12, 0)},
}
```

در اینجا به هر راس همسایه‌های آن (n) و جریانی که از آن راس می‌گذرد (f) را نسبت می‌دهیم. هر یک از این دوتایی‌ها به صورت $(capacity, flow)$ هستند.

برای پیدا کردن جریان بیشینه یک گراف از الگوریتم $ford - fulkerson$ استفاده می‌کنیم. پیاده‌سازی این الگوریتم با در نظر گرفتن ساختار گراف به صورت زیر خواهد بود:

```python
def ford_fulkerson(graph, source, destination):
    while (path := bfs_path(graph, source, destination)) is not None:
        path_flow = math.inf
```

```
        for i in range(0, len(path) - 1):
            capacity, flow = graph[path[i]]["n"][path[i + 1]]
            path_flow = min(path_flow, capacity - flow)
        for i in range(0, len(path) - 1):
            u, v = path[i], path[i + 1]
            capacity, flow = graph[u]["n"][v]
            graph[u]["n"][v] = (capacity, flow + path_flow)
            capacity, flow = graph[v]["n"].get(u, (0, 0))
            graph[v]["n"][u] = (capacity, flow - path_flow)
    for node in graph.keys():
        for key, value in list(graph[node]["n"].items()):
            if value[1] < 0:
                del graph[node]["n"][key]
```

الگوریتم بدین‌صورت عمل خواهد کرد که در صورت وجود مسیری جریان‌افزا که آن را از bfs_path می‌گیرد مقدرا جریانی که می‌تواند از این مسیر عبور کند را که برابر است با کم ظرفیت‌ترین یال یا خط ارتباطی (با در نظر گرفتن جریانی که در حال حاظر دارند) را پیدا می‌کند و جریان گذرنده از این یال را بروز می‌کند. ذکر این نکته ضروری است که در این الگوریتم اگر از یک یال جهتدار جریانی عبور دهیم، می‌توانیم به همین مقدار از آن کم‌کنیم و جریان را برگردانیم به همین دلیل مقدار جریان یال برگشتی‌را از مقدار جریان کم‌میکنیم و اگر وجود نداشته باشد آن را ایجاد می‌کنیم. در انتها یال‌هایی که جریان گذرنده از آن‌ها منفی است را حذف می‌کنیم تا خروجی الگوریتم درست باشد.

در اینجا به الگوریتم bfs_path احتیاج داشتیم که پیاده سازی آن بدین صورت خواهد بود:

```
def bfs_path(graph, source, destination):
    queue = deque([source])
    predecessors = {source: None}
    visited = set([source])

    while queue:
        node = queue.popleft()
        if node == destination:
            path = []
            while node is not None:
                path.append(node)
                node = predecessors[node]
            path.reverse()
            return path
        for neighbor in graph[node]["n"]:
            capacity, flow = graph[node]["n"][neighbor]
            if neighbor not in visited and flow < capacity:
                visited.add(neighbor)
                predecessors[neighbor] = node
                queue.append(neighbor)

    return None
```

در اینجا الگوریتم همانند پیمایش سطحی عمل می‌کند با این تفاوت که علاوه بر شرط دیده نشدن راس، این که جریان گذرنده از یال کمتر از ظرفیت آن باشد را نیز بررسی می‌کند. برای اینکه مسیر پیدا شده توسط این الگوریتم را بیابیم، پدر یا راسي که از آن به راس دیگر رفته‌ایم را ذخیره می‌کنیم تا بتوانیم از روی آن‌ها مسیر را برگردانیم.

حال پس از پیاده‌سازی الگوریتم‌های اصلی، الگوریتم‌های تبدیل را طراحی می‌کنیم:

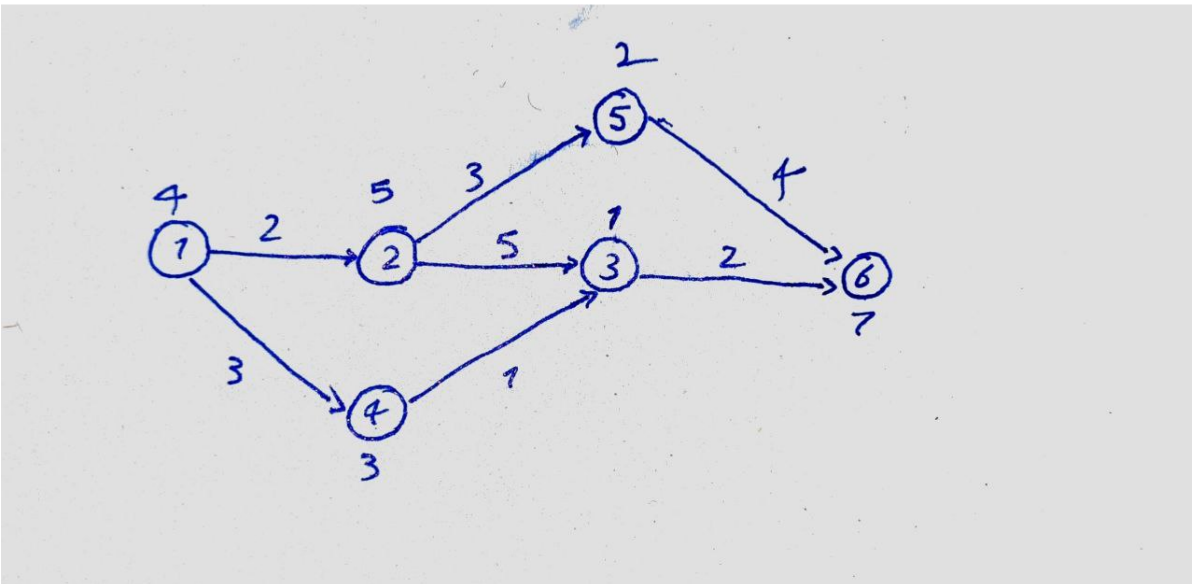
```
def transfer_graph(graph):
    i = 0
    new_graph = {}
    for node in graph:
        i -= 1
        in_node = {"n": {i: graph[node]["f"]}, "out": i, "in": True}
        out_node = {"n": graph[node]["n"]}
        new_graph[node] = in_node
        new_graph[i] = out_node
    return new_graph, i
```

برای تبدیل گراف اصلی به گراف دلخواه به ازای هر راس دوراس جدید ایجاد می‌کنیم. نام یکی از آنها را که راس v_{in} است با نام راس اصلی یکسان قرار می‌دهیم تا مشکلی برای پیمایش گراف به وجود نیاید چرا که در صورت ایجاد تغییر یا قراردادن نامی دلخواه باید تمام همسایه‌ها را در همه‌ی راس‌ها تغییر دهیم. این راس تنها به یک راس متصل است که آن راس v_{out} خواهد بود که تمام همسایه‌های راس اصلی را به آن نسبت می‌دهیم. برای برگرداندن این گراف به گراف اصلی، یکی از این راس‌ها را علامتگذاری می‌کنیم و نام راس دیگر را به این راس می‌دهیم. البته باید توجه کنیم که نام راس‌های v_{out} از منفی ۱ شروع می‌شود و نباید گراف اصلی از اعداد منفی برای نام گذاری راس‌های اصلی استفاده کرده باشد. در انتها گراف جدید و نام آخرین راس آن‌را برمی‌گردانیم.

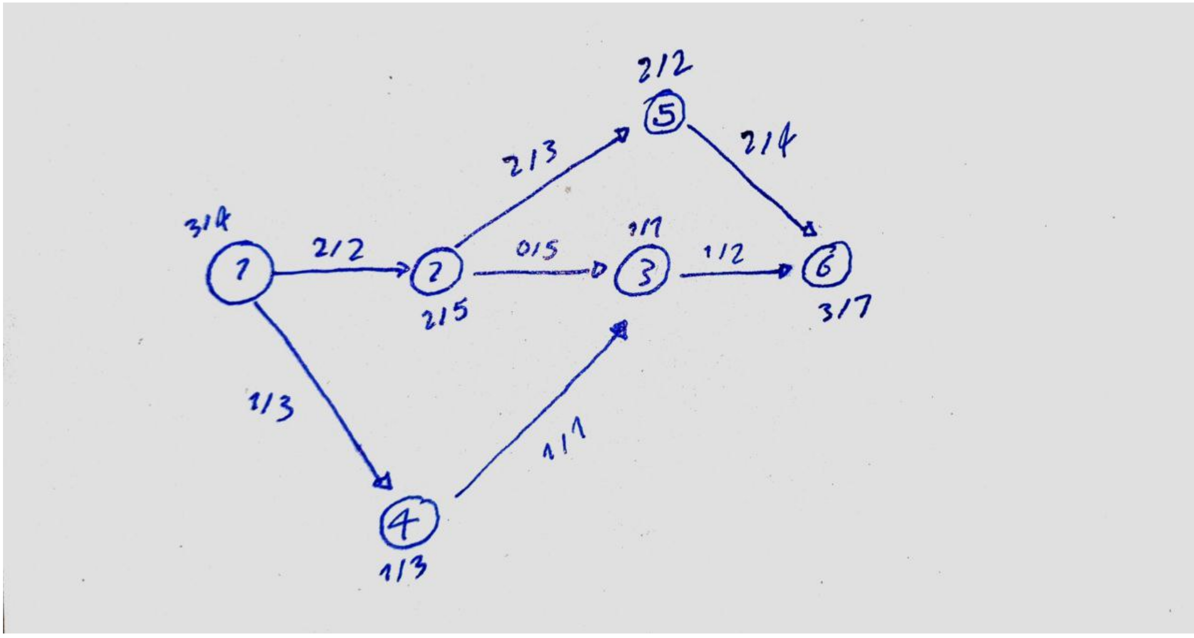
```python
def revert_graph_transform(graph):
    og_graph = {}
    for node in graph:
        if graph[node].get("in", False):
            out_node = graph[node]["out"]
            og_graph[node] = {
                "n": graph[out_node]["n"],
                "f": graph[node]["n"][out_node],
            }
    return og_graph
```

پس از اجرای الگوریتم، با استفاده از این تابع جواب نهایی مسئله اصلی را پیدا می‌کنیم.

ب



```
    1 {'n': {2: (2, 2), 4: (3, 1)}, 'f': (4, 3)}
    2 {'n': {3: (5, 0), 5: (3, 2)}, 'f': (5, 2)}
    3 {'n': {6: (2, 1)}, 'f': (1, 1)}
    4 {'n': {3: (1, 1)}, 'f': (3, 1)}
    5 {'n': {6: (4, 2)}, 'f': (2, 2)}
    6 {'n': {}, 'f': (7, 3)}
```



مراجع

Ford-Fulkerson github