دانشگاه اصفهان
دانشکده علوم ریاضی و کامپیوتر

Design and Analysis of Algorithms

طراحی و تحلیل و الگوریتم ها

محمد ملائی

عنوان:

تمرینات ۵

نیم‌سال دوم ۱۴۰۲-۱۴۰۳

نام استاد درس

جعفر الماسی زاده

تمرین ۱

این دو مسأله را در نظر بگیرید:

- **مساله ۱**: فرض کنید $G =< V, E >$ یک گراف جهتدار بی‌دور باشد. طولانی‌ترین مسیر را در گراف G بیابید.
- **مساله ۲**: فرض کنید $G =< V, E >$ یک گراف جهتدار وزندار باشد. (وزن یالها ممکن است عددی مثبت یا عددی منفی باشد.) دوری را (در صورت وجود) در گراف G بیابید که مجموع وزن یالهای آن صفر باشد.

الف یا با ارائه الگوریتمی کارا ثابت کنید که مسأله ۱ در رده P است؛ یا با تبدیل مسأله‌ای در رده NPC به آن، ثابت کنید که در رده NPC است.

ب یا با ارائه الگوریتمی کارا ثابت کنید که مسأله ۲ در رده P است؛ یا با تبدیل مسأله‌ای در رده NPC به آن، ثابت کنید که در رده NPC است.

جواب

الف

این مسئله در رده P است. برای حل این مسئله ابتدا اگر گراف وزن دار باشد، وزن همه‌ی یال‌ها را قرینه می‌کنیم و اگر بدون وزن باشند همه را -1 در نظر می‌گیریم، سپس کافیست به ازای هر راس، کوتاه‌ترین مسیر گراف جدید را پیدا کنیم. کوتاه‌ترین مسیر این مجموعه، همان جواب مسئله‌است چون مقدار آن از بقیه بیشتر است.

برای پیدا کردن کوتاه ترین مسیر از الگوریتم $dijkstra$ نمی‌توانیم استفاده کنیم اما می‌توانیم ابتدا آنها را با الگوریتم $toplogical$ $sort$ مرتب کنیم سپس با استفاده از الگوریتمی تعمیم یافته از پیمایش سطحی، هربار با رسیدن به یک راس، چون کوتاه‌ترین مسیر تا خود راس را داریم با بررسی راس های متصل، اگر مسیری که از راس کنونی می‌گذرد کوتاه‌تر باشد، آن را در نظر می‌گیریم. پس از اتمام الگوریتم کوتاه ترین مسیر را بین راس ابتدایی و هر راس دیگر داریم. بنابراین کافیست الگوریتم را یکبار برای هر راس اجرا کنیم تا طولانی‌تری مسیر گراف G را بیابیم

از آنجایی که می‌توانیم الگوریتم $sort$ $toplogical$ را در زمان $O(V + E)$ اجرا کنیم و الگوریتم تعریف شده نیز کارایی زمانش $O(|V + E|)$ است که آن را به تعداد راس‌ها اجرا می‌کنیم پس کارایی الگوریتم بصورت زیر خواهد بود:

$$O((V + E|) + V(V + E)) = O((V + 1)(V + E)) \in O(V^3)$$

ب

مساله ۲ در رده NPC است. برای اثبات این موضوع، مسئله مسیر همیلتونی را که یک مسئله سخت است به این مسئله تبدیل می‌کنیم. فرض می‌کنیم گراف G و دو راس t و s را داشته باشیم می‌خواهیم یک مسیر همیلتونی از راس s به راس t بیابیم. حالا از روی گراف G گراف G' را می‌سازیم و وزن همه‌ی یال‌ها را ۱ قرار می‌دهیم سپس یال جدیدی از t به s اضافه می‌کنیم و وزن آن‌را $n - 1$ قرار می‌دهیم. واضح است که یک مسیر همیلتونی وجود دارد اگر و تنها اگر یک دور با مجموع صفر در گراف G' وجود داشته باشد.

مراجع

–YouTube– –Wikipedia–

تمرین ۲

مدیر یکی از انجمن‌های دانشجویی بزرگِ دانشگاه، با مسأله‌ای به سراغ شما آمده است. او مسئول نظارت بر کار گروهی n نفره از دانشجویان است که هر یک از آنها طبق یک زمانبندی، باید یک نوبت در هفته، در انجمن کار کند. کارهای مربوط به نوبت‌های کاری دانشجویان متفاوت است (مانند حضور در پشت میز، کمک در تحویل بسته‌هایی و غیره) ، اما می‌توانیم هر نوبت را به شکل یک بازه زمانی پیوسته ببینیم. ممکن است چند نوبت کاری در یک زمان باشند.

مدیر می‌خواهد تعدادی از دانشجوی انجمن را انتخاب کند تا با آنها یک هیأت ناظر تشکیل دهد و با آنها جلسه‌های هفتگی داشته باشد. از نظر او، چنین هیأتی وقتی کامل خواهد بود که نوبت کاری هر دانشجویی که در هیأت نیست، با نوبت کاری یکی از دانشجویانی که در هیأت است، تداخل (گرچه جزئی) داشته باشد. بدین طریق، کارایی هر دانشجویی توسط حداقل یکی از افرادی که در هیأت حضور دارند، قابل مشاهده خواهد بود.

الف الگوریتم کارایی را با شبه‌کد توصیف کنید که زمانبندی n نوبت کاری دانشجویان را بگیرد و یک هیأت ناظرِ کامل تشکیل دهد که شاملِ کمترین تعداد دانشجو باشد.

مثلاً اگر باشد و نوبتهای کاری دانشجویان :

- دوشنبه ۴ بعد از ظهر تا ۸ بعد از ظهر
- و دوشنبه ۶ بعد از ظهر تا ۱۰ بعد از ظهر
- و دوشنبه ۹ بعد از ظهر تا ۱۱ بعد از ظهر

باشند، از آنجا که زمان نوبت کاری دوم، با زمان نوبتهای کاری اول و سوم، تداخل دارد، کوچکترین هیأت ناظرِ کامل، شامل تنها دومین دانشجو خواهد بود.

ب کارایی زمانی الگوریتم خود را اندازه بگیرید و درستی آن را نیز ثابت کنید؛ یعنی ثابت کنید که الگوریتم همیشه جواب بهینه مسأله را برمیگرداند.

جواب

الف

مسئله را با این فرض حل می‌کنیم که اگر دو بازه p_1, p_2 داشته باشیم که باهم تداخل دارند، و P_1 مجموعه تمام بازه‌هایی باشد که p_1 با آنها تداخل دارد، اگر p_2 با همه‌ی اعضای این مجموعه و احتمالا تعدادی بازه دیگر که در این مجموعه نیستند تداخل داشته باشد، تنها p_2 می‌تواند در جواب حضور داشته باشد. (البته با فرض اینکه بازه‌ی دیگری که همین وضعیت را با p_2 داشته باشد، نداشته باشیم، در غیر اینصورت آن بازه انتخاب خواهد شد). چرا که اگر آن را به عنوان بازه‌ی ناظر در نظر بگیریم همه‌ی مجموعه P_1 را پوشش خواهد داد.

حال با دانستن این موضوع الگوریتم را ارائه می‌دهیم:

```
OptimalCommittee(R = {r_1, ..., r_n})
Input: A set of shift periods R
Output: Optimal set of shifts to create a committee
  Copy R to S
  for i = 1 to n do
      R_i = {r ∈ R | r intersest r_i}
      remove = true
      for s ∈ S − {r_i} do
          for r ∈ R_i do
              if s does not intersect with r then
                  remove = false
                  break
      if remove then
          remove r_i from S
  return S
```

ب

کارایی زمانی الگوریتم در بدترین حالت که همه‌ی شیفت‌ها باهم تداخل داشته باشند $O(n^3)$ خواهد بود.

فرض می‌کنیم الگوریتم جواب بهینه را به ما ندهد بنابراین در مجوعه S عضوی وجود دارد که عضو دیگری می‌تواند مجموعه بازه‌هایی که این عضو پوشش می دهد را پوشش دهد که این با روند الگوریتم تناقض دارد چرا که این عضو در الگوریتم حذف می‌شود پس به سادگی اثبات شد که الگوریتم جواب بهینه را با می‌دهد.

تمرین ۳

تصور کنید که شما برای یک شرکت باربری بزرگ کار میکنید و یکی از وظایف شما این است که با استفاده از تعدادی کامیون، مجموعه‌ای از n جعبه را از یک شهر بندری به شهرهای دور از آن منتقل کنید. شما میدانید که بار این کامیون‌ها در نقاط مختلفی در طول مسیر، وزن خواهند شد و در صورتی که هر یک از کامیون‌ها بیش از حد مجاز بار شده باشد، شرکت باربری باید جریمه‌ای بپردازد. بنابراین، شما میخواهید که جعبه‌ها را به گونه‌ای در کامیون‌ها بگذارید که «وزن بارِ پر بارترین کامیون» به حداقل برسد.

الف با این فرض که تعداد کامیون‌ها و وزن هر یک از جعبه (که اعدادی صحیح هستند) معلوم باشند، یک الگوریتم حریصانه تقریبی برای تخصیص جعبه‌ها به کامیون‌ها طراحی کنید که نسبت تقریب آن ۲ باشد.

ب ثابت کنید که الگوریتم حریصانه‌ای که برای مسأله «کمینه‌سازی وزن بار پر بارترین کامیون» طراحی کرده‌اید، یک الگوریتم ۲ — تقریبی است. کارایی زمانی الگوریتم‌تان را هم اندازه بگیرید.

جواب

الف

الگوریتم بدین شرح خواهد بود که ابتدا جعبه‌ها را بصورت کاهشی مرتب می‌کند سپس از اولین جعبه که بزرگترین جعبه است شروع می‌کند و در هر مرحله هر جعبه را به کم‌بارترین کامیون اختصاص می‌دهد. درصورتی که چند کامیون این شرط را داشته باشند جعبه را به یکی از آن‌ها می‌دهیم چرا که تفاوتی در روند الگوریتم ایجاد نمی‌شود.

ب

فرض می‌کنیم S مجموع وزن همه‌ی جعبه‌ها باشد و s^* جواب بهینه باشد بطوری‌که وزن آن $W(s^*)$ باشد. داریم :

$$W(s^*) \geq max(b_i); b_i \in Boxes \qquad (۱)$$

این موضوع واضح است چرا که بزرگترین حتی اگر بزرگترین جعبه را به تنهایی در یک کامیون بگذاریم، یک حد پایین برای جواب بهینه خواهد بود.

از طرفی به سادگی می‌توانیم نتیجه بگیریم که:

$$W(s_a) \leq W(s^*) + max(b_i); bi \in Boxes \qquad (۲)$$

چرا که اگر جز این بود یعنی یکی از کامیون‌ها حداقل دو جعبه بیشتر از دیگر کامیون‌ها داشت که با روند الگوریتم در تضاد بود چون حداقل یکی از جعبه‌ها را به کامیون دیگری می‌داد که از $W(s^*)$ کمتر بود. و از (۱) نتیجه می‌گیریم که :

$$W(s_a) \leq W(s^*) + W(s^*) = 2W(s^*) \qquad (۳)$$

بنابراین ثابت شد که الگوریتم، یک الگوریتم ۲ ـ تقریبی است.

تمرین ۴

این مسأله را در نظر بگیرید: تابع پیوسته صعودی f و مقدار y و بازه باز (a, b) مشخص شده‌اند. مقدار x ای را در بازه باز (a, b) بیابید که $f(x) = y$ باشد. (تابعی مانند $f(x) = x^3 + x - 100$ در کل دامنه‌اش یعنی در کل بازه $(-\infty, \infty)$ صعودی است. اما اگر تابعی در کل دامنه‌اش صعودی نباشد، کافی است که در بازه مورد نظر صعودی باشد. مثلاً تابع $f(x) = sin(x)$ در بازه صعودی است و در بازه $(0, \frac{\pi}{2})$ نزولی است.)

الف الگوریتمی کارا را برای حل این مسأله توصیف کنید.

ب برنامه‌ای برای حل تقریبی مسأله بنویسید. آستانه‌ای برای حداکثر میزان خطا تعیین کنید و درستی برنامه‌تان را با چند تابع پیوسته (که در بازه‌های مورد نظر صعودی باشند) بیازمایید.

جواب

الف

برای حل این مسئله کافی‌است آن را به مسئله پیدا کردن ریشه تبدیل کنیم سپس با الگوریتم‌هایی مانند دوبخشی یا نابجایی حل کنیم که هر دو کارا هستند. تابع g را تعریف می‌کنیم به طوری که :

$$g(x) = f(x) - y$$

پس از تبدیل مسئله خواهیم داشت:

$$g(a) = f(a) - y, \; g(b) = f(b) - y \tag{۴}$$

اگر f در بازه مورد نظر شرط زیر را داشته باشده آنگاه طبق قضیه میانی مسئله اول دارای جواب است:

$$f(a) < y < f(b) \tag{۵}$$

پس داریم:

$$\rightarrow f(a) - y < y - y < f(b) - y \tag{۶}$$

$$(1) \longrightarrow g(a) < 0 < g(b) \tag{۷}$$

بنابراین مسئله دوم نیز جواب خواهد داشت و جواب آن با مسئله اول برابر خواهد بود.

ب

ابتدا تابع محاسبه ریشه را پیاده‌سازی می‌کنیم:

```python
def bisection_root(fn, eps, period, max_iters=math.inf):
    a, b = period
    iterations = 0
    while iterations <= max_iters:
        iterations += 1
        x = (a + b) / 2
        if abs(x - a) <= eps:
            return x
        if (fn(x) * fn(a)) < 0:
            b = x
        else:
            a = x
```

```python
def false_position_root(fn, eps, period, max_iters=math.inf):
    a, b = period
    iterations = 0
    while iterations <= max_iters:
        iterations += 1
        x = (a * fn(b) - b * fn(a)) / (fn(b) - fn(a))
        if abs(fn(x)) <= eps:
            return x
        if fn(x) * fn(a) < 0:
            b = x
        else:
            a = x
    return x
```

هر دو تابع از روشی مشابه استفاده می‌کنند که در هر مرحله بازه را کوچکتر می‌کنند تا به دقت مورد نظر برسند. نحوه کار الگوریتم در کلاس توضیح داده شده است پس به سراغ پیدا کردن جواب مسئله اصلی می‌رویم. برای این کار کافی است تنها تابع را تغییر دهیم و به توابع بالا بدهیم بدین‌صورت که :

```python
def find_point_bisection(fn, y, eps, period):
    return bisection_root(lambda x: fn(x) - y, eps, period)

def find_point_false_p(fn, y, eps, period, max_iters=math.inf):
    return false_position_root(lambda x: fn(x) - y, eps, period, max_iters)
```

حالا برنامه را با دقت 10^{-14} روی چند تابع پیوسته که در بازه مورد نظر صعودی باشند بررسی می‌کنیم:

$$x^2 + \ln(x) \tag{۸}$$

$$e^{x+2} + \sin(x) \tag{۹}$$

$$x^3 + 2x^2 + 2x + 4 \tag{۱۰}$$

$$x^3 - x - 1 \tag{۱۱}$$

خروجی برنامه بصورت زیر خواهد بود:

```
    0 -- y: 3 , period: (0.4, 4)
    bisection: 1.5921429370580948
    false position: 1.5921429370580926
    answer : 1.5921429370581

    1 -- y: 2 , period: (-3, -0.5)
    bisection: -0.9629509544247856
    false position: -0.9629509544247973
    answer : -0.9629509545352

    2 -- y: 7 , period: (-1, 3)
    bisection: 0.7429592021663112
    false position: 0.7429592021663136
    answer : 0.7429592021663

    3 -- y: 1 , period: (-1, 2)
    bisection: 1.5213797068045647
    false position: 1.5213797068045662
    answer : 1.5213797068046
```

تمرین ۵

الف برنامه‌ای برای پیاده‌سازی الگوریتم «نزدیک‌ترین همسایه» برای حل مسأله فروشنده دوره‌گرد بنویسید. حداقل ۱۰ نمونه با اندازه‌های مختلف از مسأله فروشنده دوره‌گرد را تولید کنید و برنامه خود را روی آن ورودی‌ها اجرا کنید و زمان‌های اجرای برنامه و خروجی‌های برنامه (مقدار تقریبی گشت فروشنده‌گرد) را در یک جدول ثبت کنید.

ب برنامه‌ای برای پیاده‌سازی الگوریتم $2OPT$ برای حل مسأله فروشنده دوره‌گرد بنویسید. به عنوان گشت اولیه، از خروجی برنامه پیاده‌ساز الگوریتم «نزدیک‌ترین همسایه» استفاده کنید. برنامه خود را روی همان داده‌های ورودی‌ای که در قسمت (الف) تولید کرده‌اید اجرا کنید. (پیشاپیش مشخص کنید که برنامه خود را روی هر ورودی به چه مدت زمانی می‌خواهید اجرا کنید.) زمان‌های اجرا و خروجی‌های این برنامه و برنامه اول را در یک جدول ثبت کنید تا کارایی زمانی و میزان دقت خروجی‌های دو برنامه را با هم مقایسه کنید.

جواب

الف

```python
def nearest_neighbor(G, v=0):
    visited = [v]
    for _ in range(len(G) - 1)::
        row = G[visited[-1]]
        min = (-1, math.inf)
        for j in range(len(G)):
            if 0 < row[j] < min[1] and j not in visited:
                min = (j, row[j])
        if min[0] == -1:
            return None
        visited.append(min[0])

    if G[visited[-1]][v] == 0:
        return None

    return visited + [v]
```

الگوریتم ابتدا از راس v شروع می‌کند و نزدیک‌ترین همسایه‌اش را به لیست $visited$ اضافه می‌کند. سپس الگوریتم این کار را برای بقیه راس‌ها نیز انجام می‌دهد بدین صورت که نزدیک‌ترین راسی که در لیست $visited$ نباشد را به عنوان نزدیک‌ترین همسایه به این لیست اضافه می‌کند اگر در بررسی یکی از راس‌ها هیچ راس کناری‌ای چنین شرطی را نداشت، الگوریتم به بن‌بست می‌رسد که می‌تواند نشان دهنده‌ی نبودن دور باشد البته ممکن است دور وجود داشته باشد اما الگوریتم نتوانسته باشد آن‌را پیدا کند. این موضوع به راس اولیه نیز مرتبط است اما کاملا تابع این موضوع نیست.

ب

```python
def two_opt(G, T):
    last_tour_length = get_tour_length(G, T)
    start = time()
    while True:
        tour_length = last_tour_length
        for i in range(len(T) - 2):
            for j in range(i + 2, len(T) - 1):
                new_tour = T[: i + 1] + [T[j]] + T[i + 2 : j] + [T[i + 1]] + T[j + 1 :]
                new_tour_length = get_tour_length(G, new_tour)
                if new_tour_length < tour_length and validate_tour(G, new_tour):
                    T = new_tour
                    tour_length = new_tour_length
                if time() - start > MAX_TIME:
                    return T
        if tour_length == last_tour_length:
            return T
        last_tour_length = tour_length
```

الگوریتم، گراف G و گشت اولیه T را می‌گیرید و سپس گشت جدیدی را طبق الگوریتم $2OPT$ تولید می‌کند و اگر طول گشت ایجاد شده از طول کمترین گشتی که الگوریتم تا این لحظه پیدا کرده‌است کمتر باشد، آنگاه آن را انتخاب می‌کند البته با توجه به اینکه ممکن است گراف کامل نباشد و گشت جدید، گشت قابل قبولی نباشد پس معتبر بودن آن را نیز بررسی می‌کنیم. این روند تا زمانی که زمان به پایان برسد یا در یک مرحله پیشرفتی حاصل نشود ادامه پیدا می‌کند. حداکثر زمان در این برنامه ۳۰ ثانیه در نظر گرفته شده است.

الگوریتم‌ها روی داده‌های تصادفی بصورت زیر عمل خواهند کرد:

```
N = 5
Nearest Neighbor: 181 -- time: 8e-06
TWO_OPT: 180 -- time: 2.4e-05

N = 10
Nearest Neighbor: 366 -- time: 1.4e-05
TWO_OPT: 366 -- time: 7.2e-05

N = 20
Nearest Neighbor: 466 -- time: 4.4e-05
TWO_OPT: 450 -- time: 0.000935

N = 30
Nearest Neighbor: 605 -- time: 0.000102
TWO_OPT: 541 -- time: 0.005498

N = 50
Nearest Neighbor: 823 -- time: 0.00024
TWO_OPT: 793 -- time: 0.010689

N = 100
Nearest Neighbor: 1412 -- time: 0.000878
TWO_OPT: 1377 -- time: 0.102146

N = 200
Nearest Neighbor: 2401 -- time: 0.00365
TWO_OPT: 2346 -- time: 0.78668

N = 300
Nearest Neighbor: 3405 -- time: 0.010927
TWO_OPT: 3339 -- time: 2.698994

N = 500
Nearest Neighbor: 5446 -- time: 0.040208
TWO_OPT: 5394 -- time: 8.617581

N = 1000
Nearest Neighbor: 10414 -- time: 0.434724
TWO_OPT: 10377 -- time: 30.000081
```