



دانشکده مهندسی
کامپیوتر و فناوری اطلاعات



دانشگاه صنعتی امیرکبیر
(پلی تکنیک تهران)

دانشگاه صنعتی امیرکبیر

(پلی تکنیک تهران)

دانشکده مهندسی کامپیوتر

طراحی الگوریتم

(پاییز ۱۴۰۱)

تمرین دوم

مهلت تمرین:

۱۴۰۱/۱۰/۰۲

ساعت ۱۱:۵۹

استاد درس : دکتر مهدی جوانمردی

سوال ۱ :

نشان دهید که چگونه می‌توان quicksort را در زمان $O(\log(n))$ در بدترین حالت اجرا کرد، با این فرض که همه عناصر متمایز هستند.

سوال ۲ :

در ارتباط با الگوریتم مرتب‌سازی Quicksort به سوالات زیر پاسخ دهید:

الف) نشان دهید چگونه می‌توان با فرض یکتا بودن همهٔ اعضای یک آرایه، Quicksort را به شکلی تغییر داد که آرایه را در بدترین حالت در زمان مرتب کند.

ب) احتمال آنکه الگوریتم مرتب‌سازی n عنصر را در زمان $\Omega(n^2)$ مرتب کند چقدر است؟

الگوریتم QUICKSORT زیر شامل دو فراخوانی بازگشتی با خودش است. پس از فراخوانی PARTITION، زیر آرایه سمت چپ و سپس زیر آرایه راست به صورت بازگشتی مرتب می‌شوند. به دومین فراخوانی بازگشتی در QUICKSORT در واقع نیازی نیست، زیرا می‌توان با استفاده از یک ساختار کنترل‌کنندهٔ تکراری آن را انجام داد. این تکنیک به نام tail recursion به طور خودکار توسط کامپایلرهای خوب ارائه می‌شود. نسخه زیر از quicksort را در نظر بگیرید که شبیه‌سازی tail recursion است:

QUICKSORT'(A, p, r)

```
1  while  $p < r$ 
2      do ▷ Partition and sort left subarray.
3           $q \leftarrow \text{PARTITION}(A, p, r)$ 
4          QUICKSORT'(A, p, q - 1)
5           $p \leftarrow q + 1$ 
```

ج) حالتی را توصیف کنید که عمق پشته QUICKSORT' برابر با $\Theta(n)$ است به ازای آرایه ورودی با n عضو.

بترن دیکھدی زمانی برای $O(n^2)$ است، که در اثر انتخاب غیرستین pivot می‌تواند بسیار بزرگ شود، یا اگر آرایه ورودی دقیقاً مرتب باشد سرتاسر داده‌ها را در صورتی که هدفی عناصر مقایسه باشند، آن‌گاه pivot ممکن است هم همواره یکتا خواهد بود، در نتیجه آرایه را به دو بخش غیرخالی تقسیم خواهد کرد، در نتیجه در نتیجه الگوریتم قادر هر مرحله حداقل یک کمتر سینه تولید خواهد کرد، که باعث می‌شود دیکھدی زمانی $O(n^2)$ کمتر شود.

دیکھدی زمانی ما، برای تعداد دفعات خواهد بود که فرآیند انتخاب pivot و تقسیم آرایه صورت می‌گیرد، اما توجه بین که ممکن است pivot های کل ممکن است خواهد بود، در نتیجه حل خطاهای سریع است.

د) کد QUICKSORT' را به گونه‌ای تغییر دهید که با حذف زمان اجرای الگوریتم در مرتبه‌ی $O(n \log n)$ بدترین عمق پشته $\Theta(\log n)$ شود.

سوال ۳ :

ثابت کنید هر الگوریتم مرتب سازی مقایسه‌ای در بدترین حالت نیاز به مقایسه $\Omega(n \lg n)$ دارد.

سوال ۴ :

قطعه کدی بنویسید که K امین عنصر کوچک آرایه را با استفاده از درخت max heap بدست آورد. مرتبه پیچیدگی زمانی و حافظه آن را بدست آورید.

سوال ۵ :

الگوریتم Radix Sort را طوری تغییر می‌دهیم که مرتب سازی رقم‌ها از رقم‌های پرارزش شروع شود و به رقم‌های کم ارزش خاتمه یابد. این الگوریتم تغییر یافته را توصیف کنید و زمان اجرای آن را تحلیل کنید.

سوال ۶ :

الگوریتم Counting Sort را به گونه‌ای تغییر دهید که مرتب سازی اعداد از اول آرایه به آخر انجام شود و خاصیت پایداری برقرار بماند. شبه کد مربوط به این الگوریتم تغییر یافته را بنویسید.

سوال ۷ :

توضیح دهید که چگونه می‌توان n عدد صحیح در بازه‌ی 0 تا $1 - n^k$ ($k \in \mathbb{N}$) را در زمان $O(kn)$ مرتب کرد.

سوال ۸ :

توضیح دهید که چگونه می‌توان الگوریتم counting sort را موازی‌سازی کرد و در p فرآیند به صورت موازی انجام داد. (پیچیدگی زمانی این روش را هم توضیح دهید)

تصویر کند و عدد داریم که عی خواهیم آن ها را با یک الگوریتم مرتب سازی می سیمایی، مرتب کنیم. این اعداد را با a_1, a_2, \dots, a_n نویسیم. زمان کنید مرتب سازی را با a_1, a_2, \dots, a_n شروع کنیم، $n-1$ عدد برای مقابله با آن داریم. سپس از اخبار عدد $n-2$ عدد برای مقابله با آن داریم، این آخر بعنوان "کل ترتیب". این حالت برای می سیمایی می آید (همان طور که اعداد درودی به a ترتیب مختلف ممکن است داده شده باشند).

حال، این حالات را در یک درخت تصویر کنید، که هر مسیری که تا برگ هایی می کند، کمی از حالاتیست که ممکن است برای مقابله باشند. در اینجا a برگ داریم، بعنوان a می باشد. کمی از ریت تا برگ، که هر یک در نهایت، ترتیب متفاوتی از نسبتی مرتب سازی می سیمایی ماست. کمی از این مسیرها، در نهایت به برگی می رسند که معادل ترتیب درست است، حداقل طول این مسیر برابر بعنوان درخت است. اگر بعنوان درخت را h برگ، یا سعی (h) خواهد بود، سپس باشد a را کاملاً کنیم. یک درخت دودویی بعنوان h ، حداقل 2^h برگ می تواند دارم و باست دوی داشت که درخت a با برگ دارد، سپس می توان (ستلال کرد):

$$n! \leq 2^h \xrightarrow{\log} (\log n!) \leq (\log 2^h) \rightarrow h \geq \log n! \Rightarrow h \in \Omega(n \log n)$$

با مکاری کوچکتر، Max Heap که در «برگتر» معنای را داشت را نیز با برگترین عضو می‌نامند. سپس اگر هر یکی از k اولین عضو بزرگ آن را در همین $n-k$ ایضاً k اولین عضو بزرگ را می‌توانیم.

```

class MaxHeap:
    def __init__(self, numbers):
        self.heap = []
        for i in range(len(numbers)):
            self.insert(numbers[i])

    def insert(self, data):
        self.heap.append(data)
        self.heapify_up(len(self.heap) - 1)

    def heapify_up(self, index):
        if index == 0:
            return
        parent = (index - 1) // 2
        if self.heap[parent] < self.heap[index]:
            self.swap(parent, index)
            self.heapify_up(parent)

    def swap(self, i, j):
        self.heap[i], self.heap[j] = self.heap[j], self.heap[i]

    def kth_largest(self, k):
        for i in range(k - 1):
            self.extract_max()
        return self.extract_max()

    def extract_max(self):
        if len(self.heap) == 0:
            return None
        self.swap(0, len(self.heap) - 1)
        max_value = self.heap.pop()
        self.heapify_down(0)
        return max_value

    def heapify_down(self, index):
        left = 2 * index + 1
        right = 2 * index + 2
        largest = index
        if left < len(self.heap) and self.heap[left] > self.heap[largest]:
            largest = left
        if right < len(self.heap) and self.heap[right] > self.heap[largest]:
            largest = right
        if largest != index:
            self.swap(index, largest)
            self.heapify_down(largest)

def main():
    numbers = [2, 4, 1, 3, 5, 6, 7, 3, 1, 2, 4]

    k = 3
    max_heap = MaxHeap(numbers)
    print(max_heap.kth_largest(k))

if __name__ == '__main__':
    main()

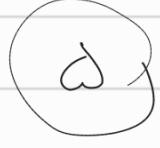
```

آنچه از این کار برای دخیره سازی می‌ستاده است که طول از $O(n \log n)$ تا $O(n^2)$ باشد.

مختصاً برای k اولین عضو، k از $\log k$ تا $\log n$ باشد.

نیازی heapify-down خواهد بود که به طور متوسط برابر با $\frac{1}{2}kn$ هست، بعنوان $O(n \log n)$ در

نتیجه-یکدیگر نیازی برابر $O(k \log n)$ خواهد بود.



در این الگوریتم (بدا) اعداد برآسان را از n تا 1 مرتب می‌سازند و سپس سایر ارقام کم اهمیت تر را می‌برند.
در این روش، در بترین حالت، را از n تا 1 مرتب می‌سازند و در متعاقبت است و در $O(n \log n)$ می‌توانند اعداد را مرتب کنند، چون خوبی مرحله اول، نیمه‌ی کوچکی خواهد بود. در بدترین حالت هم، اگر محصور به بررسی تمام ارقام شوند و متوسط طبل ارقام m باشند، الگوریتم $O(mn)$ خواهد بود.

حافظه‌ی سورر نیاز نیست، $O(n)$ که برای لیست اعداد استفاده می‌شود، $O(MB)$ که برای بکت‌ها یکال در هر مرحله احتساب داریم عدد B تعداد بکت‌های است (که در اعداد داده شده برابر ۱۰ است) و M عدد برابر طبل بکت‌ترین عدد است.

در سه من تابع، هر کدامیکی از عوامل را در ترتیبی که در آنها میباشد داشته باشند و محتوایات هر کدامیکی از عوامل را در ترتیبی که در آنها میباشد داشته باشند.



```
def main():
    numbers = [10, 1, 2, 4, 5, 9, 6, 7, 8, 3, 0, 3, 3, 4]
    max_number = 10
    print(counting_sort(numbers, max_number))
    print(stable_counting_sort(numbers, max_number))
    print(from_beginning_to_end_stable_counting_sort(numbers, max_number))

def counting_sort(numbers, max_number):
    counts = [0] * (max_number + 1)
    for number in numbers:
        counts[number] += 1
    result = []
    for i in range(len(counts)):
        for _ in range(counts[i]):
            result.append(i)
    return result

def stable_counting_sort(numbers, max_number):
    counts = [0] * (max_number + 1)
    for number in numbers:
        counts[number] += 1
    for i in range(1, len(counts)):
        counts[i] += counts[i - 1]
    result = [0] * len(numbers)
    for number in numbers:
        result[counts[number] - 1] = number
        counts[number] -= 1
    return result

def from_beginning_to_end_stable_counting_sort(numbers, max_number):
    counts = [[] for _ in range(max_number + 1)]
    for number in numbers:
        counts[number].append(number)
    result = []
    for i in range(len(counts)):
        for number in counts[i]:
            result.append(number)
    return result

if __name__ == '__main__':
    main()
```

✓

برای این کار از ترکیب Counting Sort و Radix sort استفاده می‌کنیم.

الگوریتم Counting Sort را که بر برای باقیانده اعداد بر n و سیلوس برای باقیانده اعداد
 n^k بر n^2 تقسیم می‌کند.

در مرحله اول، n بگات می‌سازیم، سپس باقیانده هر یک بر n را به دست می‌آوریم و آن را به می‌نامیم. لذا بتواند از 0 تا $n-1$ بینه بمتدار باقیانده عدد را در یکی از بگات‌های عالی که عضو π ام از رایجی بگات‌هاست می‌گذاریم.

حال اعداد بگات‌ها را به ترتیب بگات‌ها که هدیگر می‌گذاریم، نتیجه سرا بر آزادی ای است که براساس باقیانده بر n مرتب شده است. این مرحله $O(n)$ نیاز نداشت.

حالا همسن سرا حل را که این کسی و این بار اعداد را ابتدا بر n تقسیم می‌کنیم و مقدار صحیح آن را بر n باقیانده می‌گیریم. سپس بر n^2 تقسیم می‌کنیم و بعد اعداد را براساس باقیانده به n^2 مرتب می‌کنیم. همین کار را ترتیب‌سازی اعداد براساس باقیانده بر n^k (آنچه می‌رویم) می‌کنیم که مرحله هر مرحله $O(n)$ ، معنی نهایی دیده‌گشتنی الگوریتم ما سرا بر $O(kn)$ خواهد بود.

سوال ۹ :

به عنوان بخشی از یک بازی جدید، شرکت کنندگان به نوبت چندین عدد را بین ۰ تا ۱۰۰۰ حدس میزنند. در هر دور میزبان برنامه، باید بداند که کدام دو حدس به یکدیگر نزدیک‌تر هستند. یک الگوریتم بهینه از نظر زمانی ارائه کنید که به این پرسش پاسخ می‌دهد. استدلال کنید که این الگوریتم درست است و پیچیدگی زمانی آن را توضیح دهید.

بخش عملی :

برای دیدن تمرین پیاده سازی دوم به [این لینک](#) مراجعه فرمایید.

رمز ورود کلاس: ۰۰۱۰

ابدا آرایی ~ طول 1000 در نظر می‌گیرم. حدس مربوط به هر کاربر را در عضو خواهیم ذخیره کیم (نیز عددی سنت که توسط شرکت کسنه حدس زده شده است).

آن آرایی ~ O(n) طفظ احسان دارد.

حال کافیست تک دور آن آرایی را بسازیم (نیز طول می‌بگاهد) تا کاربر مascal
را بسازیم، مثلاً بین شنبه و یکشنبه:

$$\text{min_diff} = n$$

$$\text{last_guess} = \text{nill}$$

for i in [0, n]:

if guesses[i] ≠ nill:

if last_guess = nill:

last_guess = i

else:

cur_diff = i - last_guess

if cur_diff < min_diff:

min_diff = cur_diff

اگر منظور از «هر دور» در صورت سوال، «هر حدس» است، سه از دوین حدس،
نیز برابر مascalی دو حدس اول می‌گیرم. سپس بعداز هر حدس جدید که نیز باشد، کافیست خانه‌ها
را بررسی کنم، تا شنبه که روی رو:

for i in (0, min_diff):

if guesses[new_guess-i] ≠ nill or guesses[new_guess+i] ≠ nill :

min_diff = i

آن دوین هنگامی مناسب است که قرار است بازی بدرآید، در غیر این صورت مناسبتر است که
محبی، حدس‌ها را در لیست مرتب نگذارد و سه از هر حدس جدید، ابتدا خانه‌ی مناسب را بباید سپس مascalی
حدس جدید را با خانه‌ها قلچی و پیچی اس مقابله کند.

نکات تمرین:

- فایل تمرین های خود را صورت یک pdf با فرمت "Stunum_HWnum.pdf" نام گذاری کنید.
- به دلیل فشرده بودن زمان تا پایان ترم ها و شرایط پیش آمده در ترم جاری امکان تمدید تمارین وجود ندارد.
- برای تمرینات در مجموع ۵ روز زمان تاخیر وجود دارد. در صورت تاخیر بیشتر به ازای هر روز ۵ درصد از نمره‌ی کل تمرینات شما کسر می‌شود.
- در صورت شبیه بودن پاسخ تمارین دانشجویان، نمره تمرین بین دانشجویان با پاسخ مشابه تقسیم خواهد شد. (معیار برای شباهت تمرین های عملی کوئرا است)
- در صورت داشتن هرگونه ابهام و سوال با یکی از راه های زیر ارتباط برقرار کنید.

ایمیل: mohamadchoupan80@gmail.com

تلگرام: [lostago](#)