تمرین سوم طراحی الگوریتم‌ها

اشکان شکیبا (۹۹۳۱۰۳۰)

سوال اول

اصلی‌ترین موضوع در این تغییر، افزایش مقدار اولیه exponential و کاهش آن به مرور است. روش اجرا بدین شکل است که ابتدا در اعداد ورودی، عدد با بیشترین تعداد رقم را یافته و تعداد ارقام آن را در متغیری نگه می‌داریم. سپس برای هر یک از اعداد با شروع از رقم پرارزش‌تر به رقم کم‌ارزش‌تر، عناصر را بر اساس مرتبه رقم کنونی مرتب کرده و سپس مرتبه رقم بعدی را به دست می‌آوریم و این مراحل را برای همه اعداد تکرار می‌کنیم. در انتهای این کار اعداد مرتب شده‌اند.

زمان اجرای این الگوریتم بسیار به تعداد ارقام بزرگترین عدد وابسته است. اگر تعداد اعداد را n و تعداد ارقام بزرگترین عدد را k در نظر بگیریم، پیچیدگی زمانی الگوریتم جدید از مرتبه O(nk) خواهد بود.

شبه کد:

```
func counting_sort(array, exp):
    n = array.length
    out = [0] * n
    num = [0] * 10
    for i in [0, n):
        num[(array[i] // exp) % 10] += 1
    for i in [1, 10):
        num[i] += num[i - 1]
    i = n – 1
    while i >= 0:
        j = array[j] // exp
        out[num[j % 10] – 1] = array[i]
        num[j % 10] -= 1
        i -= 1
    for i in [0, n):
        array[i] = out[i]

func radix_sort(array):
    exp = 1
    maximum = max(array)
    while maximum // exp > 0:
        counting_sort(array, exp)
        exp *= 10
```

سوال دوم

الف) بدترین حالت زمانی رخ می‌دهد که تمام داده‌ها در یک باکت قرار گیرند. در این صورت پیچیدگی زمانی اجرای الگوریتم به n^2 می‌رسد.

یک راه حل می‌تواند این باشد که باکت‌هایی کوچک‌تر تعریف شوند که احتمال پدید آمدن این حالت را کاهش دهند.

یک راه حل دیگر، استفاده از الگوریتم‌های بهینه‌تر همچون merge sort برای مرتب‌سازی باکت‌هاست.

ب) می‌دانیم که الگوریتم bucket sort برای مرتب‌سازی اعداد نامنفی طراحی شده است. از این رو باید به دنبال راه حلی برای تغییر صورت کلی این الگوریتم باشیم تا برای اعداد منفی نیز درستی خود را حفظ کند.

یک راه حل می‌تواند این باشد که در ابتدا آرایه اعداد ورودی را به دو آرایه اعداد نامنفی و اعداد منفی تقسیم کنیم. سپس به جای اعداد منفی قدرمطلق آنها را در نظر گرفته و هر دو آرایه را مرتب کنیم. سپس آرایه اعداد منفی را وارونه کرده (چون زمانی که قدرمطلق یک عدد منفی از عدد منفی دیگری بزرگتر باشد یعنی خودش کوچک‌تر است) و در انتها هر دو آرایه را با حفظ ترتیب ادغام کنیم.

سوال سوم

$d_0 = 15$

$d_1 = 10$

$d_2 = 20$

$d_3 = 15$

$d_4 = 25$

i \ j	1	2	3	4
1	0	3000 k=1	5250 k=1	10500 k=1
2	-	0	3000 k=2	6750 k=3
3	-	-	0	7500 k=3
4	-	-	-	0

$c_{1,\,2} = \min(c_{1,\,k} + c_{k+1,\,2} + d_0 d_k d_2);\ k \in [1,\ 2)$

⇨ $c_{1,\,2} = 0 + 0 + (15 * 10 * 20) = 3000$

$c_{2,\,3} = \min(c_{2,\,k} + c_{k+1,\,3} + d_1 d_k d_3);\ k \in [2,\ 3)$

⇨ $c_{2,\,3} = 0 + 0 + (10 * 20 * 15) = 3000$

$c_{3,\,4} = \min(c_{3,\,k} + c_{k+1,\,4} + d_2 d_k d_4);\ k \in [3,\ 4)$

⇨ $c_{3,\,4} = 0 + 0 + (20 * 15 * 25) = 7500$

$c_{1,3} = \min(c_{1,k} + c_{k+1,3} + d_0 d_k d_3);\ k \in [1, 3)$

⇨ $c_{1,3} = \min(0 + 3000 + (15 * 10 * 15), 3000 + 0 + (15 * 20 * 15))$
$= \min(5250, 7500) = 5250$

$c_{2,4} = \min(c_{2,k} + c_{k+1,4} + d_1 d_k d_4);\ k \in [2, 4)$

⇨ $c_{2,4} = \min(0 + 7500 + (10 * 20 * 25), 3000 + 0 + (10 * 15 * 25))$
$= \min(12500, 6750) = 6750$

$c_{1,4} = \min(c_{1,k} + c_{k+1,4} + d_0 d_k d_1);\ k \in [1, 4)$

⇨ $c_{1,4} = \min(0 + 6750 + (15 * 10 * 25), 3000 + 7500 + (15 * 20 * 25), 5250 + 0 + (15 * 15 * 25)) = \min(10500, 18000, 10875) = 10500$

پاسخ بهینه:

A((BxC)(D))

سوال چهارم

اگر $i=j$ و $s[i]=s[j]$:

$$p[i, j] = 1$$

در غیر این صورت:

$$p[i, j] = \min(p[i, k] + p[k+1, j]);\ k \in [i, j)$$

ادامه الگوریتم و جدول مورد نیاز در آن مشابه مسئله قبل است و به همان شکل ادامه می‌دهیم تا به سلول بالا-راست برسیم و به پاسخ مسئله دست یابیم.

برای تحلیل پیچیدگی زمانی الگوریتم نیز باید توجه داشته باشیم که نیاز به تکمیل نیمی از خانه‌های جدول (که در بالای قطر بین گوشه‌های بالا-چپ و پایین-راست قرار دارند) داریم که تعدادشان $n^2 / 2$ است. همچنین در هر یک از خانه‌ها، برای مقایسه هر زیررشته نیاز به زمان $n / 2$ خواهیم داشت. پیچیدگی زمانی کل از حاصل ضرب این دو به دست می‌آید:

پیچیدگی زمانی $= O((n^2 / 2) * (n / 2)) = O(n^3)$