دانشکده‌ی علوم ریاضی

داده‌ساختارها و الگوریتم‌ها ۳ آذر ۹۳

جلسه‌ی ۱۴: لیست‌های زنجیره‌ای، مرتب‌سازی سطلی

مدرّس: دکتر شهرام خزائی *نگارنده: امیر فروزنده‌مقدم و مهرداد خانی*

۱ مقدمه

اگر رابطه‌ی بین برنامه‌ی کامپیوتری و الگوریتم را در نظر بگیریم، می‌توان این رابطه را بین یک زبان برنامه‌نویسی و شبه‌کد، و همچنین بین کامپیوتر (ابزار محاسباتی واقعی) و یک مدل محاسباتی صادق دانست. یک مدل محاسباتی در واقع مشخص‌کننده‌ی مواردی است از جمله:

- الگوریتم اجازه‌ی استفاده از چه عملیاتی را دارد؟
- هزینه‌ی هر عملیات (زمان، حافظه، ...) چقدر است؟

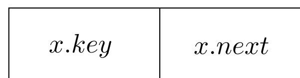
از بین مدل‌های محاسباتی می‌توان به دو مدل ماشین با دسترسی تصادفی[۱] یا رَم[۲] و همینطور مدل اشاره‌گر[۳] اشاره کرد. در مدل رَم اطلاعات در قالب آرایه‌ای ذخیره می‌شوند و امکان انجام عملیاتی مانند بازیابی، درج و حذف در زمان ثابت وجود دارد. در این مدل محاسباتی تخصیص حافظه‌ی پویا امکان‌پذیر نمی‌باشد. در مدل محاسباتی اشاره‌گر قابلیت تخصیص حافظه‌ی پویا در نظر گرفته شده‌است که مهم‌ترین تفاوت آن با مدل رَم می‌باشد.

۲ لیست زنجیره‌ای

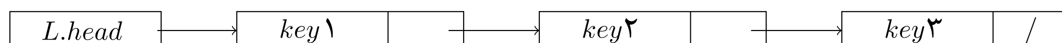
در این قسمت به توضیح و بررسی یکی از انواع ساختمان داده به نام لیست زنجیره‌ای می پردازیم. این نوع ساختمان‌داده را می توان به آرایه شبیه دانست، با این تفاوت که بر خلاف آرایه دسترسی به عناصر با استفاده از اشاره‌گر انجام می‌گیرد. در ادامه دو نوع یک‌سویه و دوسویه توضیح داده شده‌اند.

۱.۲ لیست زنجیره ای یک سویه

در این نوع لیست هر عنصر علاوه بر مقدار کلید، اشاره‌گری به عنصر بعدی را نیز در بر دارد. می توان ساختار یک عنصر را به شکل زیر نشان داد:



بدین ترتیب برای دسترسی به هر عنصر باید از اشاره‌گر به آن که در عنصر قبلی ذخیره شده است استفاده کرد که نهایتاً به پیچیدگی زمانی $O(n)$ منجر می‌شود. اشاره گر آخرین عنصر به $null$ یا nil اشاره می‌کند که مشخص‌کننده‌ی آخرین عضو می‌باشد. همینطور اشاره‌گری به نام $L.head$ وجود دارد که به اولین عنصر لیست اشاره می‌کند.به طور کلی می توان عناصر یک لیست یک‌سویه و روابط آنها را به شکل زیر نمایش‌داد:



[۱] Random Access Machine
[۲] RAM
[۳] Pointer

شبه کد اعمال درج، جستجو و حذف را در لیست یک‌سویه می‌توان به صورت زیر پیاده‌سازی کرد:

Algorithm 1 LinkedList Insert

```
function Insert(L, x)
    x.next ← L.head
    L.head ← x
```

Algorithm 2 LinkedList Search

```
function Search(L, k)
    x ← L.head
    while x ≠ nil & x.key ≠ k do
        x ← x.next

    return x
```

Algorithm 3 LinkedList Delete with key

```
function Delete(L, k)
    // assumes that k exists in L but it is not the first
    x ← L.head
    while x.next.key ≠ k do
        x ← x.next
    x.next ← x.next.next
```

بدیهیست که عملیات اول در $O(۱)$ و دو عملیات بعدی در $O(n)$ اجرا می‌شوند. با توجه به ساختار لیست یک‌طرفه، در الگوریتم‌های Search و Delete باید با استفاده از $L.head$ از ابتدای لیست شروع کنیم تا به عنصر مورد نظر برسیم. همین‌طور برای حذف یک عنصر، باید اشاره‌گر ذخیره شده در عنصر قبلی، به عنصر بعد از عنصر حذف‌شده اشاره کند. دقت شود که الگوریتم سوم با فرض اینکه می‌خواهیم با داشتن مقدار کلید یک عنصر آن را حذف کنیم کار می‌کند و حذف با داشتن آدرس به صورت زیر تغییر می‌کند:

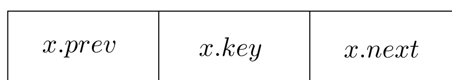
Algorithm 4 LinkedList Delete with pointer

```
function Delete(L, x)
    // assumes that k exists in L but it is not the first
    y ← L.head
    while y.next ≠ x do
        y ← y.next
    y.next ← y.next.next
```

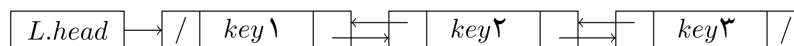
با توجه به اینکه لیست یک‌سویه است، نمی‌توان به‌طور مستقیم یک عنصر را با در دست داشتن آدرسش حذف کنیم (چرا؟) و در این حالت نیز به $O(n)$ زمان نیاز داریم. همان‌طور که در قسمت بعد خواهیم دید، لیست‌های زنجیره‌ای دوسویه در این زمینه سریع‌تر عمل می‌کنند.

۲.۲ لیست زنجیره‌ای دوسویه

عناصر لیست دوسویه علاوه بر مقدار کلید و اشاره‌گر به عنصر بعدی، اشاره‌گر به عنصر قبل از خود را نیز به عنوان مقدار سوم در بر دارد. بنابراین بر خلاف لیست یک سویه، در این جا می‌توان در دو جهت روی عناصر لیست حرکت کرد. چرا که از طریق هر یک از عناصر لیست می‌توان هم به عنصر قبل و هم به عنصر بعد دسترسی داشت. ساختار هر یک از عناصر لیست دوسویه به صورت زیر خواهد بود:



با در نظر گرفتن $L.head$ می‌توان یک لیست دوسویه را به شکل زیر نمایش داد:



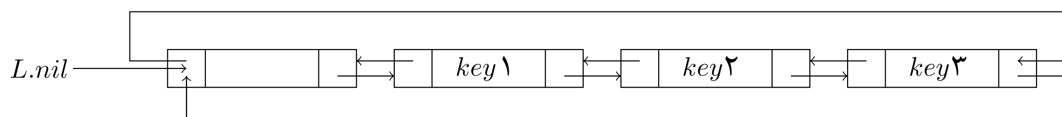
Algorithm 5 DOUBLY LINKEDLIST INSERT

function INSERT(L, x)
  $x.next \leftarrow L.head$
  if $L.head \neq nil$ **then**
    $L.head.prev \leftarrow x$
  $L.head \leftarrow x$
  $x.prev \leftarrow nil$

Algorithm 6 DOUBLY LINKEDLIST DELETE

function DELETE(L, x)
  if $x.prev \neq nil$ **then**
    $x.prev.next \leftarrow x.next$
  else
    $L.head \leftarrow x.next$
  if $x.next \neq nil$ **then**
    $x.next.prev \leftarrow x.prev$

سرلیست: با اضافه کردن یک عنصر جدید به ابتدای لیست به نام $L.nilL$ می توان بین ابتدا و انتهای لیست ارتباط ایجاد کرد. بدین شکل که مقدار $next$ آخرین عنصر و مقدار $prev$ اولین عنصر به $L.nil$ ، مقدار $L.nil.next$ به اولین عنصر و مقدار $L.nil.prev$ به آخرین عنصر لیست اشاره می کند. همینطور مقدار کلید این عنصر جدید برابر با nil می باشد، چرا که تنها نقش ارتباط دهنده ی ابتدا و انتهای لیست را بر عهده دارد. این آرایش در شکل زیر نمایش داده شده است.



اضافه شدن این عنصر جدید به لیست دو سویه پیاده سازی الگوریتم های جستجو ، درج و حذف را همان طور که در ادامه می بینیم بسیار ساده می کند:

Algorithm 7 DOUBLY LINKEDLIST DELETE WITH SENTINEL

function DELETE(L, x)
  $x.prev.next \leftarrow x.next$
  $x.next.prev \leftarrow x.prev$

Algorithm 8 Doubly LinkedList Search with Sentinel

```
function Search(L, x)
    x ← L.nil.next
    while  x ≠ L.nil & x.key ≠ k do
        x ← x.next

    return x
```

Algorithm 9 Doubly LinkedList Insert with Sentinel

```
function Insert(L, x)
    x.next ← L.nil.next
    L.nil.next.prev ← x
    L.nil.next ← x
    x.prev ← L.ni
```

در الگوریتم های درج و حذف مشابه لیست یک‌سویه عمل می کنیم، با این تفاوت که باید مقادیر $prev$ را نیز تغییر دهیم. در الگوریتم حذف باید مقدار $next$ در عنصر قبلی به عنصر بعدی، و مقدار $prev$ عنصر بعدی به عئصر قبلی اشاره کنند. همینطور در الگوریتم درج (در ابتدای لیست) باید مقدار $L.head$ و مقدار $prev$ عنصر بعدی به عنصر جدید و مقدار $next$ عنصر جدید به عنصر بعدی اشاره کنند. همینطور مقدار $prev$ عنصر جدید به nil اشاره می کند. همچنین همانطور که مشاهده می‌شود در حذف با داشتن اشاره‌گر در لیست دوسویه، $O(۱)$ زمان نیاز داریم.

۳.۲ مقایسه‌ی پیچیدگی‌های زمانی

در انتها به مقایسه‌ی پیچیدگی زمانی اجرای الگوریتم‌های گفته‌شده برای آرایه و لیست زنجیره‌ای می‌پردازیم. لازم به ذکر است که منظور از عملیات حذف، حذف یکی از عناصر به دلخواه (بدون نیاز به جستجو) و همینطور منظور از درج، اضافه کردن یک عنصر جدید به انتها (برای آرایه) یا ابتدا (برای لیست) می باشد.

	Insert	Delete	Search
Linked List	1	n	n
Doubly Linked List	1	1	n
Array	1	n	n

جدول ۱: مقایسه‌ی زمان‌های اجرا

۳ مرتب‌سازی سطلی

این روش مرتب‌سازی، همانند روش شمارشی یک ویژگی برای دنباله‌ی اعداد فرض می‌کند. در این روش فرض می‌شود که اعداد ورودی به صورت مستقل و یکنواخت روی بازه‌ی $[۰, ۱)$ قرار دارند.

۱.۳ الگوریتم مرتب‌سازی سطلی

روش مرتب سازی سطلی این بازه را به n زیربازه (یا سطل) با طول برابر تقسیم می‌کند و سپس هر عدد را به زیربازه‌ی مربوطه اضافه می‌کند. با توجه به فرض اولیه که اعداد به صورت یکنواخت پخش شده‌اند، احتمال اینکه تعداد زیادی از اعداد در هر سطل قرار گیرد بسیار پایین خواهد بود.

در این الگوریتم آرایه‌ای جداگانه از طول n شامل لیست‌های زنجیره‌ای درنظر گرفته می‌شود که هرکدام مشخص‌کننده‌ی یک سطل هستند که اعداد به آن اضافه می‌شوند. الگوریتم مربوط به مرتب‌سازی سطلی در ادامه آمده‌است:

Algorithm 10 BUCKET SORT

```
function BUCKETSORT(A[1..n])
    Let B[0..n − 1] be a new empty array
    for i = 1 → n do
        Insert A[i] into list B[⌊nA[i]⌋]
    for i = 0 → n − 1 do
        Sort list B[i] with insertion sort
    Concatenate the lists B[0], B[1], ..., B[n − 1] together in order
```

ابتدا اعداد با توجه مقدارشان در لیست متناظر اضافه می‌شوند. سپس هر کدام از لیست ها با استفاده از الگوریتم مرتب‌سازی درجی مرتب می‌شوند و نهایتاً همه‌ی لیست‌ها با هم الحاق می‌شوند که دنباله‌ی مرتب‌شده‌ی مورد نظر ما را تولید می‌کند. در ادامه پیچیدگی زمانی اجرای این الگوریتم در حالت متوسط محاسبه شده است.

۲.۳ پیچیدگی زمانی مرتب‌سازی سطلی

مشخص است که در بدترین حالت (یعنی حالتی که همه‌ی اعداد داخل یک لیست قرار بگیرند) پیچیدگی زمانی اجرای الگوریتم از $O(n^2)$ می‌باشد. حال به تحلیل حالت متوسط‌زمانی می‌پردازیم. با توجه به اینکه تا سر حلقه‌ی **for** دوم در الگوریتم مرتب‌سازی سطلی، در زمان $\Theta(n)$ انجام می‌پذیرد، با فرض اینکه تعداد عناصری که در سطل $B[i]$ قرار گرفته‌اند، متغیر تصادفی n_i باشند، داریم:

$$T(n) = \Theta(n) + \sum_{i=0}^{n-1} n_i^2$$

حال برای مقدار متوسط داریم:

$$\begin{aligned} E[T(n)] &= E\left[\Theta(n) + \sum_{i=0}^{n-1} n_i^2\right] \\ &= \Theta(n) + \sum_{i=0}^{n-1} E\left[n_i^2\right] \end{aligned}$$

$E\left[n_i^2\right] = 2 - \frac{1}{n}$ خواهد شد(چرا؟). بنابراین:

$$\begin{aligned} E[T(n)] &= \Theta(n) + \sum_{i=0}^{n-1} E\left[n_i^2\right] \\ &= \Theta(n) + n\left(2 - \frac{1}{n}\right) \\ &= \Theta(n) \end{aligned}$$

بنابراین مرتب‌سازی سطلی به طور متوسط $\Theta(n)$ زمان می‌برد.