دانشکده علوم ریاضی و آمار

مدرس: دکتر مجتبی رفیعی نیمسال اول ۱۴۰۰-۱۴۰۱

ساختمان داده‌ها و الگوریتم‌ها

جلسه ۸

نگارنده: منصوره حقانی منش

۳۰ مهر ۱۴۰۰

فهرست مطالب

۱ اثبات درستی الگوریتم به روش ثابت حلقه

از خصیصه‌ای ثابت در هر یک از حلقه‌های موجود در الگوریتم برای اثبات درستی الگوریتم استفاده می‌نماییم. لازم به ذکر است که این روش برمبنای اثبات استقرایی بنانهاده شده است.

یادآوری اثبات استقرایی: فرض کنید P(n) حکمی درمورد اعداد طبیعی باشد، برای اثبات صحت حکم برای تمام مقادیر طبیعی n، یک اثبات استقرایی صحت گزاره‌های زیر را تایید می کند:

- **پایه استقراء:** **P(۱)** درست است (در گام اول باید اثبات کنیم که گزاره P(n) به زای n=۱ درست می‌باشد).
- **گذار (گام) استقراء** : به ازای عدد طبیعی مانند k، اگرP(k) درست باشد آنگاه P(k+۱) نیز درست است.

درنهایت با تایید مراحل پایه و گام استقراء صحت حکم به استقراء ثابت می‌شود.

ثابت حلقه (Loop-Invariant): همانطور که گفتیم ثابت حلقه، نوعی اثبات استقرایی است وبه طور خلاصه برای اثبات درستی یک حلقه داریم که:

۱): خصیصه‌ی ثابت در آغاز حلقه درست باشد،

۲): خصیصه‌ی ثابت درهر تکرار حلقه درست باشد،

۳): خصیصه‌ی ثابت در انتهای حلقه درست باشد.

توجه: قبل از اثبات درستی الگوریتم به این روش باید ابتدا ثابت حلقه (همان خصیصه‌ی ناوردا) را تشخیص دهیم. لازم به ذکر است که برای حلقه‌های مختلف در یک الگوریتم، ثابت حلقه می‌تواند متفاوت باشد و چنانچه بخواهیم به طور کاملا دقیق درستی یک الگوریتم را بررسی کنیم باید درستی هریک از حلقه‌های موجود درالگوریتم را جداگانه برسی کنیم.
با اینحال گام‌های ثابت حلقه را می‌توان به طور کلی در سه گام زیر خلاصه کرد:

- **گام۱، آغاز(Initialization):** همانند بررسی پایه برای اثبات استقرایی است. درواقع باید نشان دهیم که خصیصه ناوردای مورد نظر (ثابت حلقه) قبل ازاجرای حلقه (قبل از شروع حلقه) برقرار است.

- **گام۲، نگهداری(Maintenance):** همانند بررسی گام گذار برای اثبات استقرایی است. در واقع باید نشان دهیم که خصیصه ناوردای موردنظر (ثابت حلقه) قبل از اجرای هر تکرار برقرار است؛ به عبارت دیگر، قبل از تکرار حلقه درست است و قبل از تکرار حلقه‌ی بعد هم درست است.

- **گام۳، پایانی(Termination):** در پایان نشان می‌دهیم که پس از خارج شدن الگوریتم از حلقه (پس از به اتمام رسیدن حلقه)، خصیصه ناوردایی دارای ویژگی است که می‌توان ازآن صحت الگوریتم را استنتاج کرد.

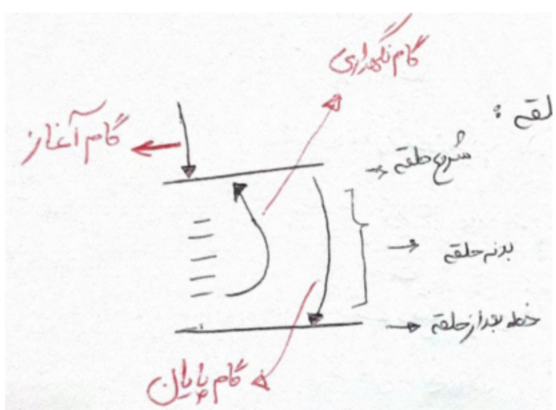
دریک الگوریتم ابتدا قسمت‌های بدیهی و غیربدیهی شبه کد را از هم تفکیک می‌کنیم و در ادامه به حلقه‌ها توجه می‌کنیم و برای هریک از حلقه‌ها خصیصه‌ای ثابت را درنظر می‌گیریم ودرستی الگوریتم را اثبات می‌کنیم.

شکل۱، گام‌های ثابت حلقه را با جزئیات بیشتری به تصویر کشیده است. همانطور که درشکل مشاهده می‌شود:

- **گام آغاز:** قبل از شروع حلقه، قبل از اینکه وارد بدنه حلقه شویم، زمانی که شمارنده‌ی حلقه مقداردهی اولیه می‌شودگام آغاز طی می‌شود.

- **گام نگهداری:** هر بار که حلقه اجرا شد و دوباره به ابتدای حلقه باز می‌گردیم(قبل از اجرای گام بعدی) گام نگهداری طی می‌شود.

- **گام پایان:** شمارنده‌ی حلقه به مقدار مشخصی که رسید، عملیات حلقه دیگر انجام نمی‌شود و از ابتدای حلقه به خط بعد از حلقه هدایت می‌شویم و در اینجا گام پایانی اثبات صورت می‌گیرد.



شکل ۱: گام‌های مصور ثابت حلقه

۲ اثبات درستی الگوریتم مرتب سازی درجی

شبه کد مربوط به الگوریتم مرتب سازی درجی در ادامه آورده شده است.

Algorithm 1 Insertion-Sort(A[1..n])

```
1: for k = 2 to n do
2:     key = A[k]
3:     i = k
4:     while (i > 1 and A[i − 1] > key) do
5:         A[i] = A[i − 1]
6:         i = i − 1
7:     A[i] = key
```

با توجه به مطالب بیان شده در بخش‌های قبلی، برای اثبات درستی الگوریتم مرتب‌سازی درجی به ترتیب گام‌های زیر را دنبال می‌کنیم:

- **تعیین خصیصه ناوردایی حلقه :** درست پس از مقداردهی شمارنده حلقه، اغاز مرحله k-ام، زیر آرایه $A[1..k-1]$ مرتب است، بنابراین از همین خصیصه‌ی ثابت برای اثبات درستی استفاده می‌نماییم.

- **طی کردن گام‌های ناوردایی حلقه :**

  - **گام آغاز:** باید نشان دهیم که خصیصه ناوردایی حلقه، درست بعد از اولین مقداردهی به k برقرار است . واضح است که: در مرحله k=۲، (منظور قبل از شروع حلقه است) داریم $A=[1..k-1]=A[1..1]$ و درنتیجه چون آرایه A یک عنصر دارد مرتب است.
  - **گام نگهداری:** باید نشان دهیم که درآغاز هر تکرار، خصیصه ناوردایی برقرار است. به عبارت دیگر اگر تکرار i-امانجام شده باشد و در آغاز تکرار (i+۱) -ام باشیم؛ یعنی k=i+۱، باید نشان دهیم که زیرآرایه $A[1..i]=A[1..k]$ مرتب است. از آنجاییکه خطوط ۲ تا ۷ مقدار $A[i]$ را در جای صحیح(موقت) خود قرار می‌دهد پس $A[1..i]$ مرتب است. لازم به ذکر است که دراینجا دراثبات درستی حلقه‌ی for، از درستی حلقه‌ی درونی while بهره گرفته شده است .
  - **گام پایان :** در پایان بررسی می‌کنیم که وقتی حلقه به پایان رسید، چه روی می‌دهد. شرط پایان حلقه را بررسی می‌کنیم؛ در این الگوریتم شرط پایان یافتن حلقه زمانی است که $k=n+1$ باشد که باتوجه به خصیصه‌ی ناوردایی تعیین شده بدین معنی است که آرایه A[۱..n]، n عنصر آرایه مرتب شده می‌باشند. به وضوح دیده شد که خصیصه ناوردایی حلقه، صحت الگوریتم را نتیجه می‌دهد.

نکته: اثبات دقیق‌تر آن است که خصیصه ناوردایی برای حلقه‌ی داخلی (while) نیز لحاظ شود.
نکته: درصورتی که حلقه‌ها تودرتو باشند از داخلی‌ترین حلقه برای اثبات درستی شروع می‌کنیم.
توجه: در مثال الگوریتم مرتب‌سازی به روش درجی، محتوای خانه‌های آرایه در هیچ مرحله‌ای تغییر نمی‌کنند و صرفا جایگشتی از عناصر را داریم، اما دربرخی مسائل لازم است که درحین کار محتوای آرایه نیز تغییر کند و در این صورت این موضوع دراثبات تاثیر دارد و باید در نظر گرفته شود.

در ادامه اثبات حلقه درونی الگوریتم مرتب‌سازی درجی آورده شده است.

- **تعیین خصیصه ناوردایی :** درست پس از مقداردهی متغیر حلقه‌ی while یعنی i=k،i، همه‌ی زیر آرایه‌های سمت راست i عنصر اول بزرگتر از عنصر k-ام هستند. به عبارت دیگر جای صحیح عنصر k-ام درحلقه while یافت می‌شود.

- **دنبال کردن گام‌های ناوردایی حلقه:**

  - **گام آغاز :** دراین مرحله $i=k$ است و واضح است که زیر آرایه‌های سمت راست k عنصر اول بزرگتر از عنصر k-ام هستند.
  - **گام نگهداری :** در آغاز تکرار $i+1-$امی با توجه به خصیصه ناوردایی زیر آرایه‌های سمت راست i عنصر اول بزرگتر از عنصر k-ام هستند.

- **گام پایان :** دراین گام به وضوح مشخص است وقتی از حلقه while خارج می‌شود که یا عنصر موردنظر در خانه‌ی اول آرایه قرار گرفته باشد (i<=۱) یا اینکه هیچ زیر آرایه‌ای از عنصر مورد نظر بزرگتر نباشد ($A[i-1] <= key$) که در هر دو صورت در صورت پیدا شدن جایگاه صحیح عنصر مورد نظر دو مورد گفته شده رخ می‌دهد (خصیصه ناوردایی صحت الگوریتم مورد نظر را نتیجه می‌دهد) ولذا الگوریتم حلقه‌ی while صحیح می‌باشد.

۳ تحلیل پیچیدگی زمانی الگوریتم مرتب سازی درجی

دراین بخش سعی بر آن است تا تحلیل واقعی الگوریتم مرتب سازی درجی را بررسی کنیم. دراین راستا، شبه کد مربوط به این الگوریتم را که در ادامه آورده شده است، در نظر می‌گیریم. برای تحلیل این الگوریتم، فرض کرده‌ایم خط i–ام دارای هزینه اجرای C_i است.

Algorithm 2 Insertion-Sort(A[1..n])

```
1: for  k=2 to n do     C1
2:     key=A[k]     C2
3:     i=k     C3
4:     while i>1 and A[i-1]>key do     C4
5:         A[i]=A[i-1]     C5
6:         i=i-1     C6
7:     A[i]=key     C7
```

یادآوری: تحلیل پیچیدگی زمانی الگوریتم:

۱) واقعی

I) بهترین حالت ،

II) بدترین حالت ،

III) حالت میانگین(متوسط) ،

۲) مجانبی

تحلیل پیچیدگی (خط به خط) زمانی (درحالت واقعی) الگوریتم مرتب سازی درجی:

- **خط ۱:** درخط اول، به تعداد۱+۲-n بار خودحلقه‌ی for تکرار می‌شود و۱ بارهم وقتی که حلقه به اتمام می‌رسد (خود حلقه به خاطر شرط خاتمه یکباربیشتر از بدنه حلقه تکرار می‌شود). به عبارتی: n=۱+۱+۲-n. هزینه اجرا: (n)C۱

- **خط ۲ و۳:** خط دوم وسوم از بدنه‌ی حلقه‌ی for می‌باشند و تعداد تکرار دستورات این خطوط به تعداد تکرار بدنه‌ی حلقه‌ی for می‌باشد به عبارتی :۱-n .

  هزینه اجرا : $(n-1)(C2+C3)$

- **خط ۴ :** در خط چهارم، در حلقه‌ی while ، شرطی بررسی می‌شود که درست بودن یا نبودن آن به نوع آرایه ورودی وابسته است، به عبارتی تعداد دفعاتی که خط چهارم اجرا می‌شود به نوع آرایه ورودی بستگی دارد، لذا یک مقدار متغیر همانند tk را که درواقع متناسب با نوع ورودی مقدار می‌گیرد را به عنوان تعداد تکرار های خط ۴ درنظر می‌گیریم وبدین معنی است که در هر تکرار حلقه‌ی for باتوجه به نوع آرایه ورودی، تعداد دفعات لازم برای تکرار حلقه‌ی while متفاوت می‌باشد و لذا این مقادیر را به ازای هرشمارنده با یکدیگر جمع می‌کنیم، به عبارتی هزینه اجرای خط ۴ برابر است با :

$$(\sum_{k=2}^{n} t_k)C4$$

- **خط ۵ و ۶ :** خط پنجم و خط ششم هم دقیقا مشابه خط ۴ هستند (تعداد تکرارشان متناسب با نوع ورودی می‌باشد). با این تفاوت که درهرحالت (مهم نیست ورودی از چه نوعی باشد). یکبار کمتراز خط ۴ اجرا می‌شوند، به عبارتی مقدار متغیری که متناسب باورودی مقدارمی‌گیرد را tk-۱ در نظرمی‌گیریم و هزینه‌ی اجرای خط ۵و ۶ برابر است با:

$$(\sum_{k=2}^{n}(t_k-1))(C5+C6)$$

- **خط ۷ :**خط هفتم جزئی از بدنه‌ی حلقه‌ی for است وتعداد تکرارش برابر با تعداد تکرار بدنه حلقه است به عبارتی:n-۱.
هزینه اجرا : (n-۱)C۷

درنهایت هزینه‌ی اجرای هریک از خطوط را با هم جمع کرده و تابع T(n)رابه عنوان تابع محاسبه پیچیدگی زمانی (برحسبn، اندازه آرایه)درنظرمی‌گیریم و داریم:

$$T(n)=(n)C_1+(n-1)(C_2+C_3)+(\sum_{k=2}^{n}t_k)C_4+(\sum_{k=2}^{n}(t_k-1))(C5+C6)+(n-1)C_7$$

چناچه عبارتT(n)راساده کنیم وبرحسب n ساده تر بنویسیم می‌توانیم فرم کلی زیر را داشته باشیم:

$$T(n)=An+B+C\sum_{k=2}^{n}t_k$$

توجه : برای محاسبه دقیق پیچیدگی زمانی باید مقدار دقیق t_k را بدانیم، اما چطور می‌توان این مقدار دقیق را محاسبه کرد؟
برای این کار تحلیل پیچیدگی زمانی الگوریتم را درسه حالت (طبق آنچه که در یادآوری بیان شد): بهترین حالت، بدترین حالت وحالت میانگین بررسی می‌کنیم. در این جلسه به بررسی دو حالت اول می‌پردازیم. لازم به ذکر است که در الگوریتم مرتب سازی درجی گفته شده در بالا هدف مرتب کردن آرایه به صورت صعودی بود.
درادامه برای سهولت درنوشتن هزینه هایی که ثابت هستند و به نوع ورودی بستگی ندارند را با*Aنمایش می‌دهیم:

$$A^*=(n)C_1+(n-1)(C_2+C_3)+(n-1)C_7$$

I) **بهترین حالت :** زمانی است که آرایه به صورت صعودی مرتب شده باشد.

دراین صورت خط ۴ تنها یکبار بررسی می‌شود، tk=۱ ، به ازای هر بار اجرای بدنه‌ی حلقه‌ی for یعنی n-۱بار، همچنین از آنجا که آرایه مرتب است خطوط ۵ و۶ اجرا نمی‌شوند، لذا داریم:

$$T^{'}(n)=A^*+(n-1)C_4+0(C_5+C_6)$$

فرم کلی (که به صورت خطی می‌باشد):

$$T^{'}(n)=An+B$$

II) **بدترین حالت :** زمانی است که آرایه به صورت نزولی مرتب شده باشد.

خط ۴ :
دراین صورت درخط۴ به ازای هرشمارنده‌ی k به تعداد k-۱ بارشیفت صورت می‌گیرد، هم چنین آخرین باری که حلقه‌ی while بررسی می‌کند نیز یک عمل به حساب می‌آید لذا: tk=(k-۱)+۱=k و هزینه محاسباتی خط ۴ برابر است با:

$$\sum_{k=2}^{n}t_k=\frac{n(n+1)}{2}-1$$

خط ۵و ۶ : خط پنجم وششم هم همانطور که گفتیم درهرحالت تعداد تکرارشان کمترازخط ۴ است ، ۱−tk ،شمارنده:k و tk=k ، لذا برای هزینه محاسباتی خط ۵و ۶ باهم داریم:

$$\sum_{k=2}^{n}(t_k - 1) = \frac{n(n-1)}{2}$$

درنهایت خواهیم داشت :

$$T^{''}(n) = A^{*} + (\frac{n(n+1)}{2} - 1)C_4 + \frac{n(n-1)}{2}(C_5 + C_6)$$

فرم کلی (به صورت چند جمله‌ای درجه ۲ است):

$$T^{''}(n) = An + B + Cn^2$$

جمع بندی: در تحلیل یک الگوریتم باید توجه داشته باشیم که ما الگوریتم را برای مجموعه‌ای از نمونه‌ها ارائه می‌دهیم لذا حالتی که از اهمیت برخوردار است بدترین حالت است زیرا بدترین حالت یک کران بالا تعیین می‌کند بدین معنی است که هر ورودی بدهیم بیشتر از کران محاسبه شده، زمان لازم نیست و از طرفی نیز محاسبه‌ی حالت متوسط دشوار می‌باشد، بدین خاطر در تحلیل الگوریتم‌ها بیشتر، بدترین حالت را مد نظر قرار می‌دهند.