طراحی الگوریتم ها

مدرس: یکماز

منبع درسی

- **عنوان:**

  طراحی و تحلیل الگوریتم ها

- **نویسنده:**

  دکتر بهروز قلی زاده

- **انتشارات:**

  دانشگاه صنعتی شریف

- **منبع اصلی:**

Fundamentals of computer algorithm

E.Horowitz, S.Sahni

Computer science Press 1978

بارم بندی

- پایان ترم: ۱۵ نمره
- حل تمرینات و مشارکت در مباحث کلاسی: ۵ نمره
- حضور در کلاس: ۲– تا ۲ نمره

سرفصل آموزشی

- مفاهیم اولیه الگوریتم ها، پیچیدگی زمانی، مرتبه اجرایی
- روابط بازگشتی
- روش تقسیم و غلبه
- برنامه نویسی پویا
- روش حریصانه
- روش عقب گرد

مفاهیم اولیه

- **تعاریف**
- **مساله: پرسشی است که به دنبال پاسخ آن هستیم.**
- **مثال:**
  - ***لیست S شامل n عدد را به ترتیب صعودی مرتب کنید.***

پاسخ مساله $S=\{12,17,11,5,3,8\} \Rightarrow S=\{3,5,8,11,12,17\}$

  - ***تعیین کنید آیا عدد*** x ***در لیست*** S ***شامل*** n ***عدد وجود دارد یا خیر. اگر*** x ***در*** S ***باشد پاسخ مثبت است و اگر*** x ***در*** S ***نباشد، پاسخ منفی است.***

مفاهیم اولیه

- **پارامترهای مساله:** متغیرهایی در صورت مساله که مقادیر مشخصی به آنها نسبت داده نشده است.
- **مثال:**
  - ***در مثال اول دو پارامتر S(لیست اعداد) و n(تعداد اعداد لیست)***
  - ***در مثال دوم سه پارامتر S، n و x(عدد مورد نظر جهت جستجو)***
- **نکته: با هربار نسبت دادن مقادیر مشخص به پارامترها، نمونه ای از مساله به دست می آید.**

مفاهیم اولیه

- **تعاریف**
- **الگوریتم: مجموعه ای از دستور العمل ها که اگر به ترتیب دنبال شوند موجب حل مساله می گردد.**
- **ویژگی های الگوریتم:**
  - ***دقیق***
  - ***ترتیب***
  - ***پایان پذیر***

مفاهیم اولیه

- تحلیل الگوریتم: محاسبه زمان اجرا و میزان حافظه مورد نیاز
- مشخصه های الگوریتم خوب:
  - سادگی
  - واضح بودن
  - زمان اجرا
  - میزان حافظه مصرفی
  - ... ؟؟؟

مفاهیم اولیه

- **یک الگوریتم برای مثال دوم (جستجوی عنصر)**

با شروع از نخستین عضو موجود در S، x را با هر یک از اعضای S مقایسه می کنیم تا x پیدا شود یا به انتهای لیست برسیم. حال اگر x یافت شد، پاسخ مثبت و اگر x پیدا نشد، پاسخ منفی است.

- **نوشتن الگوریتم به زبان فارسی دو ایراد دارد:**
  - ***نوشتن الگوریتم های پیچیده به این شیوه دشوار است.***
  - ***مشخص نیست از توصیف فارسی الگوریتم چگونه می توان یک برنامه کامپیوتری ایجاد کرد.***

الگوریتم ۱–۱: جستجوی ترتیبی

```
Void  seqsearch ( int n
                  const keytype  S[ ]
                  keytype  x,
                  index& location)
{
     location = 1;
     while (location <= n && S[location]  ! = x)
           location++;
     if  (location  >  n )
           location  = 0 ;
}
```

الگوریتم ۲–۱:محاسبه مجموع عناصر آرایه

```
number sum (int n , const number s[ ])
{
    index i;
    number  result;

    result = 0;
    for (i = 1; i <= n; i++)
        result = result + s[i];
    return  result;
}
```

الگوریتم ۳–۱: مرتب سازی تعویضی

```
void  exchangesort (int n , keytype S[ ])
 {
      index i,j;
      for (i = 1 ; i<= n -1; i++)
           for (j = i +1; j <= n ; j++)
                if ( S[j] < S[i])
                     exchange S[i]  and S[j];
}
```

الگوریتم ۴–۱: ضرب ماتریس ها

```
void matrixmult (int n
                 const number  A [ ] [ ],
                 const number  B [ ] [ ],
                       number  C [ ] [ ],
{
    index i , j, k;
    for  ( i = 1; I <= n ; i++)
        for (j = 1; j <= n ; j++)
        {
            C [i] [j] = 0;
        for (k = 1 ; k <= n ; k++)
                        C [i][j] = C[i] [j] + A [i][k] * B [k][j]    }
}
```

جست و جوی دودویی–مقدمه

- ***مساله:*** تعیین کنید آیا x در آرایه مرتب شده S با n کلید وجود دارد یا خیر.
- ***ورودی:*** عدد صحیح مثبت n، آرایه مرتب شده S، کلید x
- ***خروجی:*** Location، موقعیت x در S(صفر در صورتی که x در S نباشد)
- ***روش کار:*** ابتدا x را با عنصر میانی آرایه مقایسه می کند. اگر مساوی بودند الگوریتم به پایان می رسد. اگر x کوچکتر از عنصر میانی بود، باید در نیمه نخست آرایه باشد و الگوریتم در نیمه نخست تکرار می شود. اگر x بزرگتر از عنصر میانی بود، جستجودر نیمه دوم آرایه تکرار می شود. این رویه تکرار می شود تا x پیداشود یا مشخص گردد که x در آرایه وجود ندارد.

الگوریتم ۵–۱: جست و جوی دودویی

```
Void  binsearch (int n,
                 const keytype  S[ ],
                 keytype  x,
                 index& location)
{
     index low, high, mid;
     low = 1 ; high = n;
     location = 0;
     while (low <= high && location = = 0) {
          mid = ⌊(low + high)/2⌋;
 if ( x = = S [mid])
          location = mid;
     else if (x < S [mid])
          high = mid – 1;
     else
          low = mid + 1;
     }
   }
```

جست وجوی دودویی–تحلیل

- در داخل حلقه با یک پیاده سازی مناسب یک مقایسه داریم.
- بیشترین تعداد مقایسه: x از همه کوچک و یا از همه بزرگتر
- اگر n=32 و x از همه بزرگتر:
  - *مقایسه با اندیس های ۱۶، ۲۴، ۲۸، ۳۰، ۳۱، ۳۲*
- تعداد کل مقایسه: lg n + 1

اهمیت ساخت الگوریتم های کارآمد

- **جست و جوی دودویی معمولا بسیار سریع تر ازجست و جوی ترتیبی است.**
- **تعداد مقایسه های انجام شده توسط جست و جوی دودویی برابر با $\lg n + 1$ است .**

اندازه آرایه	تعداد مقایسه های انجام شده توسط جستجوی ترتیبی	تعداد مقایسه های انجام شده توسط جستجوی دودویی
۱۲۸	۱۲۸	۸
۱۰۲۴	۱۰۲۴	۱۱
۱۰۴۸۵۷۶	۱۰۴۸۵۷۶	۲۱
۴۲۹۴۹۶۷۲۹۴	۴۲۹۴۹۶۷۲۹۴	۳۳

دنباله فیبوناچی

- سری فیبوناچی به طریق بازگشتی زیر تعریف می شود:

$f_0=0$

$f_1=1$

$f_n=f_{n-1}+f_{n-2}$ For $n\geq 2$

- محاسبه چند جمله ابتدایی:

$f_2=f_1+f_0=1+0=1$

$f_3=f_2+f_1=1+1=2$

$f_4=f_3+f_2=2+1=3$

$f_5=f_4+f_3=3+2=5$

.

.

.

الگوریتم ۶–۱: جمله nام فیبوناچی (بازگشتی)

- **مسئله : جمله nام از دنباله فیبوناچی را تعیین کنید.**

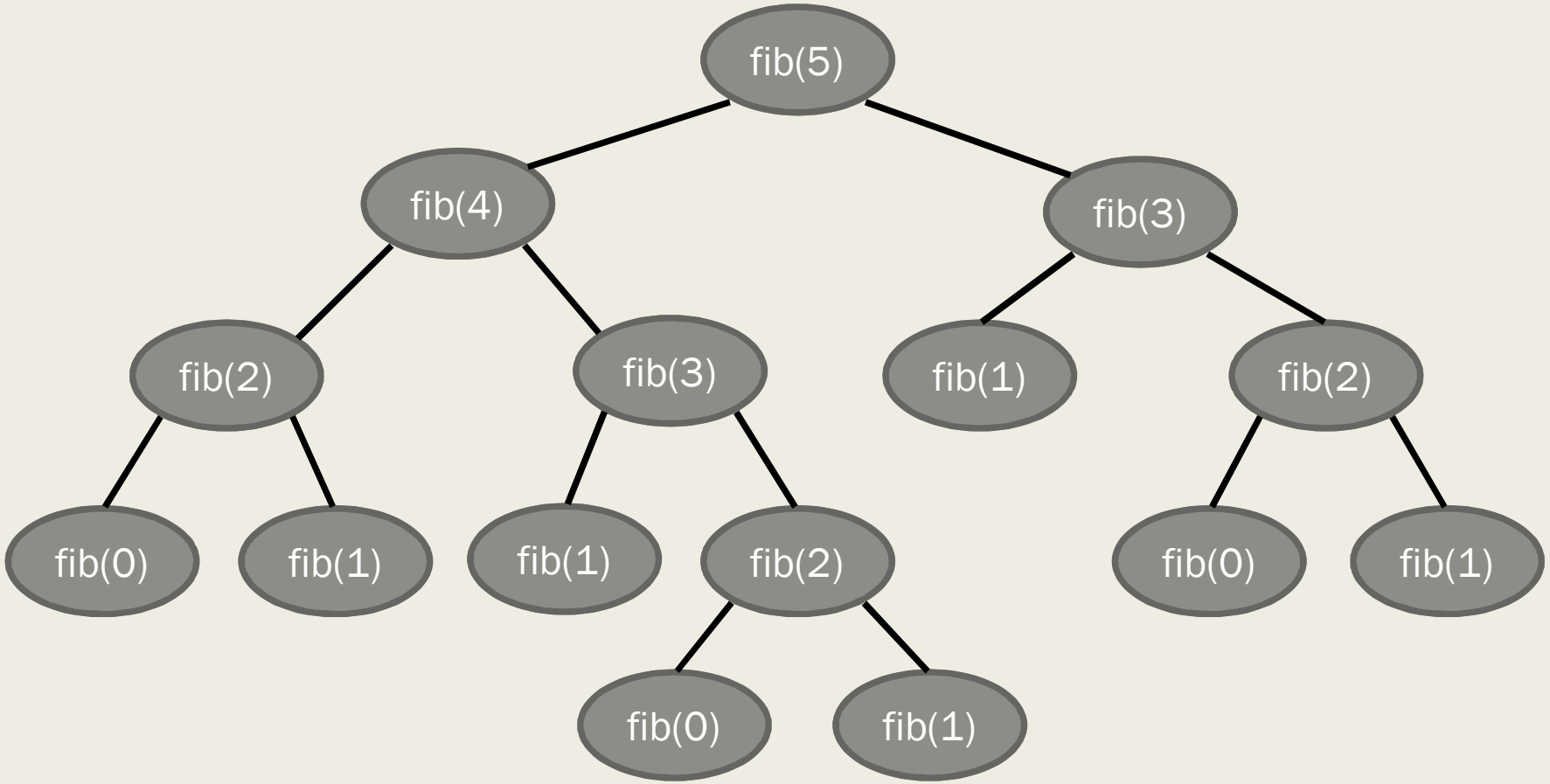
```
int fib  (int n)
{
    if ( n <= 1)
        return n;
    else
        return fib (n – 1)  + fib (n – 2);
}
```

الگوریتم ۷–۱: جمله n ام فیبوناچی (تکراری)

```
int fib2 (int n)
 {
    index i;
    int f [0..n];
    f[0] = 0;
    if (n > 0)  {
          f[1] = 1;
          for  (i = 2 ; I <= n; i++)
              f[i] = f [i -1] + f [i -2];
    }
    return  f[n];
}
```

جمله n ام فیبوناچی (بازگشتی)–تحلیل

- تعداد جملات محاسبه شده برای fib(n)
- بطور مثال برای محاسبه fib(5) تعداد ۱۵ فراخوانی انجام می شود.



جمله n ام فیبوناچی (بازگشتی)–تحلیل

n	0	1	2	3	4	5	6
تعداد جملات محاسبه شده	۱	۱	۳	۵	۹	۱۵	۲۵

- **اگر T(n) تعداد جملات در درخت بازگشتی باشد آنگاه**

$$T(n) > 2 \times T(n-2)$$

$$T(n) > 2 \times 2 \times T(n-4)$$

...

$$T(n) > 2^{n/2} \times T(0) = 2^{n/2}$$

مقایسه دو الگوریتم فیبوناچی

زمان اجرا الگوریتم ۶-۱	زمان اجرا الگوریتم ۷-۱	$2^{n/2}$	n+1	اندازه آرایه n
1048 ms	41 ns	۱۰۴۸۵۷۶	۴۱	۴۰
1 s	61 ns	1.1×10^{9}	۶۱	۶۰
18 min	81 ns	1.1×10^{12}	۸۱	۸۰
13 day	101 ns	1.1×10^{15}	۱۰۱	۱۰۰
36 year	121 ns	1.2×10^{18}	۱۲۱	۱۲۰
3.8×10^{7} year	161 ns	1.2×10^{24}	۱۶۱	۱۶۰

تحلیل الگوریتم ها

- **برای تعیین میزان کارایی، یک الگوریتم را باید تحلیل کرد.**
- **انواع تحلیل الگوریتم**
  - ***تحلیل زمانی* یعنی بدانیم که یک الگوریتم برای اجرا به چه میزان زمان و فضا نیاز دارد.**
  - ***تحلیل صحت و درستی* با پی گیری الگوریتم و امتحان کردن آن با داده های مختلف یا با استفاده از روش های ریاضی و اثبات صحیح بودن آن انجام می شود.**
- **نکته:** نوع دیگر تحلیل مربوط به ساده بودن الگوریتم است. البته برخی مواقع ساده ترین و مستقیم ترین روش مساله، بهترین راه حل نیست.

تحلیل الگوریتم ها– پیچیدگی زمانی

- **تعیین دقیق زمان اجرای یک الگوریتم غیرممکن است؟؟؟**
  - ***وابسته به سخت افزار***
  - ***وابسته به اندازه ورودی***
- **زمان تقریبی مورد درخواست یک الگوریتم به صورت تابعی از اندازه های ورودی بیان می شود.**
- **اگر n اندازه ورودی باشد:**
  - ***زمان مورد نیاز $T(n)$***
  - ***فضای مورد نیاز $S(n)$***

تحلیل الگوریتم ها– پیچیدگی زمانی

- **مراحل محاسبه تحلیل زمانی:**
  - ***به دست آوردن اندازه ورودی:***

    با توجه به مسئله برابر اندازه آرایه، تعداد سطرها، تعداد یال و گره و ....

  - ***تعیین عمل اصلی الگوریتم***

    دستور(ها) که کل کار انجام شده توسط الگوریتم تقریبا متناسب با تعداد دفعات اجرای این دستور(ها) است

- **تحلیل پیچیدگی زمانی الگوریتم، تعیین تعداد دفعاتی است که عمل اصلی به ازای هر مقدار از ورودی انجام می شود.**

تحلیل الگوریتم ها– پیچیدگی زمانی

- **تحلیل پیچیدگی زمانی برای حالت معمول برای الگوریتم جمع کردن عناصر آرایه**

عمل اصلی: افزودن یک عنصر از آرایه به sum

اندازه ورودی: n، تعداد عناصر آرایه.

$$T(n) = n$$

تحلیل الگوریتم ها– پیچیدگی زمانی

- **تحلیل پیچیدگی زمانی برای حالت معمول برای الگوریتم مرتب سازی تعویضی**

عمل اصلی: مقایسه S[j] با S[i]

اندازه ورودی: n، تعداد عناصری که باید مرتب شوند.

$$T(n) = \frac{n(n-1)}{2}$$

تحلیل الگوریتم ها– پیچیدگی زمانی

- **تحلیل پیچیدگی زمانی برای حالت معمول برای الگوریتم ضرب ماتریس ها**

عمل اصلی: دستور ضرب در داخلی ترین حلقه تکرار(for)

اندازه ورودی: n، تعداد سطرها و ستون ها.

$$T(n) = n^2$$

تحلیل الگوریتم ها– پیچیدگی زمانی

- تحلیل های فوق برای همه موارد الگوریتم یکسان بود
- الگوریتم هایی وجود دارند که همیشه پیچیدگی زمانی یکسانی ندارند.
- در این صورت سه تکنیک حل وجود دارد:
  - *W(n): تحلیل پیچیدگی زمانی در بدترین حالت*
  - *A(n): پیچیدگی زمانی در حالت میانگین*
  - *B(n): پیچیدگی زمانی در بهترین میانگین*

تحلیل الگوریتم ها– پیچیدگی زمانی

- **تحلیل پیچیدگی زمانی در بدترین حالت برای الگوریتم جست و جوی ترتیبی**

عمل اصلی: مقایسه یک عنصر آرایه با x

اندازه ورودی: n، تعداد عناصر موجود در آرایه

$$W(n) = n$$

تحلیل الگوریتم ها– پیچیدگی زمانی

- تحلیل پیچیدگی زمانی در حالت میانگین برای الگوریتم جست و جوی ترتیبی

عمل اصلی: مقایسه یک عنصر آرایه با x

اندازه ورودی: n، تعداد عناصر موجود در آرایه

$$A(n) = \frac{n+1}{2}$$ جستجوی موفق

$$A(n) = n\left(1 - \frac{p}{2}\right) + \frac{p}{2}$$ جستجوی ناموفق

تحلیل الگوریتم ها– پیچیدگی زمانی

- **تحلیل پیچیدگی زمانی در بهترین حالت برای الگوریتم جست و جوی ترتیبی**

عمل اصلی: مقایسه یک عنصر آرایه با x

اندازه ورودی: n، تعداد عناصر موجود در آرایه

$$B(n) = 1$$

مرتبه الگوریتم

- الگوریتم هایی با پیچیدگی زمانی ازقبیل n و 100n را الگوریتم های زمانی خطی می گویند.
- الگوریتم های از مرتبه n^2 را الگوریتم های زمانی درجه دوم می گویند
- مجموعه کامل توابع پیچیدگی را که با توابع درجه دوم محض قابل دسته بندی باشند، $\theta(n^2)$ می گویند.
- اگر تابعی عضو این مجموعه باشد، گویند این تابع در مرتبه n^2 است
- مثال: اگر داشته باشیم:

$$g(n) = 5n^2 + 100n + 20 \in \theta(n^2)$$

یعنی g(n) از مرتبه n^2 است

مرتبه الگوریتم

- مجموعه ای ازتوابع پیچیدگی که با توابع درجه سوم محض قابل دسته بندی باشند، $\theta(n^3)$ نامیده می شوند.
- به این مجموعه ها، گروه های پیچیدگی می گویند.
- برخی از گروه های پیچیدگی متداول در زیر داده شده است:

$$\theta(\lg n) < \theta(n) < \theta(n \lg n) < \theta(n^2) < \theta(n^3) < \theta(2^n)$$

- نمادهای O، θ، Ω به دلیل اینکه حد توابع را برای مقادیر بزرگتر پارامترها تعیین می کنند، نمادهای مجانبی نامیده می شوند.

آشنایی بیشتر با مرتبه الگوریتم ها

- **نماد O**

برای یک تابع پیچیدگی مفروض f(n)، O (f (n)) "O بزرگ" مجموعه ای از توابع پیچیدگی g (n) است که برای آن ها یک ثابت حقیقی مثبت c و یک عدد صحیح غیر منفی N وجود دارد به قسمی که به ازای همه ی n >= N داریم:

$$g(n) >= c \times f(n)$$

- **مثال:**

$4n^2 \in O(n^2)$

$3log^2 n \in O(n^2)$

$n \in O(n^2)$

$n\ log\ n \in O(n^2)$

آشنایی بیشتر با مرتبه الگوریتم ها

- **نماد Ω**

برای یک تابع پیچیدگی مفروض f(n)، Ω (f (n)) مجموعه ای از توابع پیچیدگی g (n) است که برای آن ها یک ثابت حقیقی مثبت c و یک عدد صحیح غیر منفی N وجود دارد به قسمی که به ازای همه ی n >= N داریم:

$$g(n) <= c \times f(n)$$

- **مثال:**

$4n^2 \in \Omega(n^2)$

$(6n^2 + 9) \in \Omega(n^2)$

$4n^3 + 3n^2 \in \Omega(n^2)$

آشنایی بیشتر با مرتبه الگوریتم ها

نماد θ

برای یک تابع پیچیدگی مفروض f(n)، داریم:

$$\theta\,(f(n)) = O\,(f(n)) \cap \Omega\,(f(n))$$

یعنی θ(f(n)) مجموعه ای از توابع پیچیدگی g (n) است که برای آن ها ثابت های حقیقی مثبت c وd و عدد صحیح غیر منفی N وجود دارد به قسمی که :

$$c \times f\,(n) <= d \times f(n)$$

- **مثال:**

$$(n+1)^2 \in \theta(3n^2)$$

$$\frac{(n^2+5n+7)}{(5n^3+7n+2)} \in \theta(\frac{1}{n})$$

$$(1+\frac{3}{n})^n \in \theta(1)$$