طراحی الگوریتم

موسسه آموزش عالی پاسارگاد شیراز

الگوریتم ضرب ماتریس ها

- روش ساده ضرب ماتریسها

```
void mul_mat(int a[][n], int b[][n], int c[][n])
{
        int i, j, k;
        for (i = 0; i < n; i++)
        {
                for (j = 0; j < n; j++)
                {
                        c[i][j] = 0;
                        for (k = 0; k < n; k++)
                                c[i][j] = c[i][j] + a[i][k] * b[k][j];
                }
        }
}
```

مرتبه اجرایی $O(n^3)$

الگوریتم ضرب ماتریس ها

- روش ساده ضرب ماتریسها به کمک روش تقسیم و غلبه
- در این روش فرض می کنیم اندازه ماتریس ها n×n باشد که در آن n توانی از۲ باشد. در این حالت برای انجام ضرب، ماتریس ها را به چهار قسمت تقسیم می کنیم که اندازه هر ماتریس n/2 خواهد بود بنابراین ضرب به صورت زیر می شود

$$\underbrace{\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}}_{n\times n} \times \underbrace{\begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}}_{n\times n} = \underbrace{\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix}}_{n\times n}$$

$\frac{n}{2}$ (بالای ستون اول)

$\frac{n}{2}$ (کنار سطر اول)

الگوریتم ضرب ماتریس ها

- روش ساده ضرب ماتریسها به کمک روش تقسیم و غلبه

$$\underbrace{\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}}_{\frac{n}{2}\times\frac{n}{2}} \times \underbrace{\begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}}_{n\times n} = \underbrace{\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix}}_{n\times n}$$

$$C_{11}=A_{11}\times B_{11}+A_{12}\times B_{21} \qquad C_{12}=A_{11}\times B_{12}+A_{12}\times B_{22}$$

$$C_{21}=A_{21}\times B_{11}+A_{22}\times B_{21} \qquad C_{22}=A_{21}\times B_{12}+A_{22}\times B_{22}$$

الگوریتم ضرب ماتریس ها

- روش ساده ضرب ماتریسها به کمک روش تقسیم و غلبه

$$C_{11}=A_{11} \times B_{11}+A_{12}\times B_{21} \qquad C_{12}=A_{11} \times B_{12}+A_{12}\times B_{22}$$

$$C_{21}=A_{21} \times B_{11}+A_{22}\times B_{21} \qquad C_{22}=A_{21} \times B_{12}+A_{22}\times B_{22}$$

$$T(n)=\begin{cases} 1 & n=1 \\ 8T\left(\frac{n}{2}\right)+4\left(\frac{n^2}{4}\right) & n>1 \end{cases}$$

a=8
b=2
k=2

$$\theta\left(n^{log_2^8}\right) \qquad log_2^8 > 2 \rightarrow \theta(n^3)$$

الگوریتم ضرب ماتریس ها

- ضرب ماتریس به روش استراسن
- در این روش نیز فرض می کنیم اندازه ماتریس n×n باشد که n توانی از ۲ است.و سپس ماتریس را به ۴ ماتریس n/2 تقسیم می کنیم و از فرمول زیر ضرب ماتریس را انجام می دهیم

$$\underbrace{\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}}_{n\times n} \times \underbrace{\begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}}_{n\times n} = \underbrace{\begin{bmatrix} m_1+m_4-m_5+m_7 & m_3+m_5 \\ m_2+m_4 & m_1+m_3-m_2+m_6 \end{bmatrix}}_{n\times n}$$

الگوریتم ضرب ماتریس ها

- ضرب ماتریس به روش استراسن

$$\underset{n\times n}{\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}} \times \underset{n\times n}{\begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}} = \underset{n\times n}{\begin{bmatrix} m_1+m_4-m_5+m_7 & m_3+m_5 \\ m_2+m_4 & m_1+m_3-m_2+m_6 \end{bmatrix}}$$

$$m_1 = (A11 + A22) * (B11 + B22)$$

$$m_2 = (A21 + A22) * B11$$

$$m_3 = A11 * (B12 - B22)$$

$$m_4 = A22 * (B21 - B11)$$

$$m_5 = (A11 + A12) * B22$$

$$m_6 = (A21 - A11) * (B11 + B12)$$

$$m_7 = (A12 - A22) * (B21 + B22)$$

الگوریتم ضرب ماتریس ها

- ضرب ماتریس به روش استراسن

$$m_1 = (A11 + A22) * (B11 + B22)$$

$$m_2 = (A21 + A22) * B11$$

$$m_3 = A11 * (B12 - B22)$$

$$m_4 = A22 * (B21 - B11)$$

$$m_5 = (A11 + A12) * B22$$

$$m_6 = (A21 - A11) * (B11 + B12)$$

$$m_7 = (A12 - A22) * (B21 + B22)$$

الگوریتم ضرب ماتریس ها

- ضرب ماتریس به روش استراسن

$$m_1 = (A11 + A22) * (B11 + B22)$$

$$m_2 = (A21 + A22) * B11$$

$$m_3 = A11 * (B12 - B22)$$

$$m_4 = A22 * (B21 - B11)$$

$$m_5 = (A11 + A12) * B22$$

$$m_6 = (A21 - A11) * (B11 + B12)$$

$$m_7 = (A12 - A22) * (B21 + B22)$$

مرتبه اجرایی جمع

$$T(n) = 7T\left(\frac{n}{2}\right) + 18\left(\frac{n}{2}\right)^2 \rightarrow \theta\left(n^{log_2^7}\right) = \theta(n^{2.81})$$

مرتبه اجرایی ضرب

$$T(n) = 7T\left(\frac{n}{2}\right) \rightarrow \theta\left(n^{log_2^7}\right) = \theta(n^{2.81})$$

پیدا کردن کوچکترین و بزرگترین عنصر در یک لیست

- الگوریتم معمولی برای انجام این کار

```
void min_max_1(int list[n], int &max, int &min)
{
        max = min = list[0];
        for (int i = 1; i < n; i++)
        {
                if (list[i] > max)
                        max = list[i];
                else if (list[i] < min)
                        min = list[i];
        }
}
```

پیدا کردن کوچکترین و بزرگترین عنصر در یک لیست

- الگوریتم معمولی برای انجام این کار

```
void min_max_1(int list[n], int &max, int &min)
{
        max = min = list[0];
        for (int i = 1; i < n; i++)
        {
                if (list[i] > max)
                        max = list[i];
                else if (list[i] < min)
                        min = list[i];
        }
}
```

بهترین حالت آرایه صعودی مرتب است پس شرط ها n-1 بار اجرا می شود چون if دوم اصلا اجرا نمی شود. (n-1)

بدترین حالت آرایه نزولی مرتب باشد پس هر دو شرط n-1 بار اجرا می شود.2(n-1)

پیدا کردن کوچکترین و بزرگترین عنصر در یک لیست

- روش دوم(فرض تعداد خانه ها زوج)

```
void min_max_2(int list[n], int &max, int &min)
{
        if (list[0] > list[1])
        {
                max = list[0];
                min = list[1];
        }
        else
        {
                max = list[1];
                min = list[0];
        }
        for (int i = 2; i < n-1; i+=2)
        {
                if (list[i] < list[i + 1])
                        swap(list[i], list[i + 1]);
                if (list[i]> max)
                        max = list[i];
                if (list[i+1]< min)
                        min = list[i+1];
        }
}
```

پیدا کردن کوچکترین و بزرگترین عنصر در یک لیست

- روش دوم(فرض تعداد خانه ها زوج)

```
void min_max_2(int list[n], int &max, int &min)
{
        if (list[0] > list[1])
        {
                max = list[0];
                min = list[1];
        }
        else
        {
                max = list[1];
                min = list[0];
        }
        for (int i = 2; i < n-1; i+=2)
        {
                if (list[i] < list[i + 1])
                        swap(list[i], list[i + 1]);
                if (list[i]> max)
                        max = list[i];
                if (list[i+1]< min)
                        min = list[i+1];
        }
}
```

مرتبه اجرایی الگوریتم:

$$(\frac{n-2}{2} \times 3) + 1 = \frac{3n}{2} - 2$$

پیدا کردن کوچکترین و بزرگترین عنصر در یک لیست

- روش دوم(فرض تعداد خانه ها فرد)

```
void min_max_3(int list[n], int &max, int &min)
{
        max = min = list[0];
        for (int i = 1; i < n - 1; i += 2)
        {
                if (list[i] < list[i + 1])
                        swap(list[i], list[i + 1]);
                if (list[i]> max)
                        max = list[i];
                if (list[i + 1]< min)
                        min = list[i + 1];
        }
}
```

پیدا کردن کوچکترین و بزرگترین عنصر در یک لیست

- روش دوم(فرض تعداد خانه ها فرد)

```
void min_max_3(int list[n], int &max, int &min)
{
        max = min = list[0];
        for (int i = 1; i < n - 1; i += 2)
        {
                if (list[i] < list[i + 1])
                        swap(list[i], list[i + 1]);
                if (list[i]> max)
                        max = list[i];
                if (list[i + 1]< min)
                        min = list[i + 1];
        }
}
```

مرتبه اجرایی الگوریتم:

$$(\frac{n-1}{2} \times 3) = \frac{3n}{2} - \frac{3}{2}$$

پیدا کردن کوچکترین و بزرگترین عنصر در یک لیست

- روش سوم(استفاده از روش تقسیم و غلبه)
  - در این روش اگر تعداد عناصر ۱ عضو بود max و min همان عنصر می شود و اگر ۲ عنصر بود به کمک ۱ شرط max و min پیدا می شود و در صورتی که تعداد عناصر بیش از ۲ عضو باشد به صورت زیر عملیات بازگشتی جستجو max و min را انجام می دهیم:
    - ۱- آرایه را به دو قسمت تقسیم می کنیم.
    - ۲- بزرگترین و کوچکترین عنصر را در قسمت سمت چپ آرایه پیدا می کنیم.
    - ۳- بزرگترین و کوچکترین عنصر را در قسمت راست آرایه پیدا می کنیم.
    - ۴- بین دو عنصر بزرگتر به دست آمده از مرحله قبل max را پیدا می کنیم و همچنین بین ۲ عنصر کوچکتر در مرحله قبل min را مشخص می کنیم.

پیدا کردن کوچکترین و بزرگترین عنصر در یک لیست

- روش سوم(استفاده از روش تقسیم و غلبه)

```
void min_max_4(int list[],int low,int high, int &max, int &min)
{
        if (low == high)
                max = min = list[low];
        else if (low == high - 1)
        {
                if (list[low] > list[high])
                {
                        max = list[low];
                        min = list[high];
                }
                else
                {
                        max = list[high];
                        min = list[low];
                }
        }
```

پیدا کردن کوچکترین و بزرگترین عنصر در یک لیست

- روش سوم(استفاده از روش تقسیم و غلبه)

```
else
        {
                int m = (low + high) / 2;
                int max1, max2, min1, min2;
                min_max_4(list, low, m, max1, min1);
                min_max_4(list, m+1, high, max2, min2);
                if (max1 > max2)
                        max = max1;
                else
                        max = max2;
                if (min1 < min2)
                        min = min1;
                else
                        min = min2;
        }
}
```

پیدا کردن کوچکترین و بزرگترین عنصر در یک لیست

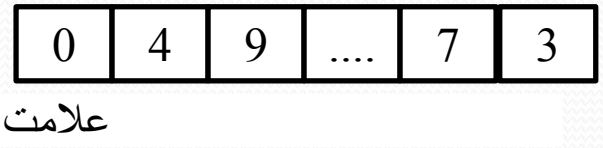
- روش سوم(استفاده از روش تقسیم و غلبه)
  - مرتبه اجرایی

$$T(n)\cong \begin{cases} 1 & n==1 \\ 2 & n==2 \\ 2T(\frac{n}{2}) + 2 & else \end{cases}$$

$$\theta\left(n^{log_2^2}\right)=\theta(n) \qquad a\neq 1$$

ضرب اعداد صحیح بزرگ

- ذخیره سازی به کمک آرایه،سمت چپ ترین خانه ی آرایه به عنوان علامت در نظر می گیریم



- جمع و تفریق از مرتبه O(n)می باشد

2 4 3 ….. 4 6
+
1 6 2 ….. 2 3

…….. 6 9

هر عدد nرقم
نیاز به n عمل جمع

ضرب اعداد صحیح بزرگ

- برای ضرب باید هر رقم از عدد b در تمام ارقام a ضرب شود$O(n^2)$

a 2 4 3 ….. 4 6

×

b 1 6 2 ….. 2 3

ضرب اعداد صحیح بزرگ

- روش تقسیم و غلبه برای ضرب اعداد صحیح بزرگ
- فرض کنید که می خواهیم ضرب $U \times V$ را انجام دهیم که U و Vدو عدد صحیح بزرگ هستند

$U= X * 10^m + Y$

$V= W * 10^m + Z$

n= تعداد ارقام

$m= [\frac{n}{2}]$

U=732154932

$U= 73215 * 10^4 + 4932$

X , Y , Z , W هر کدام دارای تقریبا $\frac{n}{2}$ رقم هستند

$$U \times V = (X * 10^m + Y) * (W * 10^m + Z) = X*W*10^{2m} + (X*Z+W*Y)*10^m + Y*Z$$

برای هر کدام از ضرب ها، باید ضرب $\frac{n}{2} \times \frac{n}{2}$ رقم انجام شود

تقسیم عدد بزرگ به دو عدد با نصف ارقام عدد را آنقدر ادامه می دهیم تا تعداد ارقام به یک تعداد خاص(آستانه) مثلا ۵ رقم برسد

ضرب اعداد صحیح بزرگ

- روش تقسیم و غلبه برای ضرب اعداد صحیح بزرگ

```
large_int  Product ( large_int  U , large_int  V)
{
        large_int   X,W,Y,Z;
        int  n,m;
        if (U == 0 || V==0)
                return 0;
        n=max(number of digits U , number of digits V);
        if (n<=threshold)
                return U*V;
        else
        {
                m= n/2;
                X= U Divide 10^m;
                Y= U Rem 10^m;
                W= V divide 10^m;
                Z= V Rem 10^m;
                return Product(X,W)*10^2m + ( Product(X,Z) + Product(W,Y))*10^m + Product(Y,Z);
        }
}
```

ضرب اعداد صحیح بزرگ

- مرتبه اجرایی

$$T(n) \cong \begin{cases} C_1 & n == threshold \\ 4T\left(\frac{n}{2}\right) + C_2 n & else \end{cases}$$

a=4
b=2
k=1

$$\theta\left(n^{log_2^4}\right) = \theta(n^2)$$

ضرب اعداد صحیح بزرگ

- با تغییر زیر می توان مرتبه اجرایی را کاهش داد

$U \times V = (X * 10^m + Y) * (W * 10^m + Z) = X*W*10^{2m} + (X*Z+W*Y)*10^m + Y*Z$

$(X*Z+W*Y) = (X + Y) * (W + Z) - X*W - Y*Z$

$U \times V = X*W*10^{2m} + ((X + Y) * (W + Z) - X*W - Y*Z)*10^m + Y*Z$

$$T(n) \cong \begin{cases} C_1 & n == \text{threshold} \\ 3T\left(\frac{n}{2}\right) + C_2 n & else \end{cases}$$

a=3
b=2
k=1

$$\theta\left(n^{log_2^3}\right)$$

مسئله انتخاب selection

- پیدا کردن kامین کوچکترین و بزرگترین عنصر یک آرایه
- در حالت کلی می توان آرایه را مرتب کرد که مرتبه آن O(nlogn) میشود سپس kامین min و max را پیدا میکنیم.
- اگر k=1 باشد همان مسئله پیدا کردن کوچکترین و بزرگترین عنصر در یک لیست می باشد که مرتبه جرایی آن O(n) می شود.

- اگر k=2 باشد
  - روش ساده
  - اول max را پیدا میکنیم(n-1حرکت) سپس آن را حذف می‌کنیم و در n-1 عنصر باقی مانده دوباره max را پیدا میکنیم (n-2حرکت) بنابراین برای پیدا کردن دومین بزرگ ترین عنصر نیازبه 2n-3 حرکت داریم. مرتبه جرایی آن O(n) می شود

مسئله انتخاب selection

- اگر k=2 باشد میتوان از روش تورنمنت(حذفی) استفاده کرد
- در این روش ابتدا مثل قبل مسابقه دو به دو بین اعضا انجام می شود و این عملیات تا مشخص شدن max ادامه پیدا می‌کند که تعداد مسابقات در این حالت n-1 می‌شود حال برای پیدا کردن دومین بزرگترین عنصر باید max را بین بازنده هایی که از بزرگ ترین عنصر باخته اند پیدا کنیم. تعداد این عناصر Log(n) می شود(ارتفاع درخت) حال مسئله پیدا کردن max نیاز به log(n)-1 عمل دارد. پس برای پیدا کردن دومین max به روش تورنمنت نیاز به n+log(n)-2 عملیات است که از روش قبل کمتر است

7 , 24 , 17 , 18 , 30 , 5 , 40 , 13

24 18 30 40

24 40

40

پیدا کردن ماکزیمم بین سه عدد زیر

24 , 30 , 13

مقدار دومین ماکزیمم برابر با عدد 30 می شود

مسئله انتخاب selection

- اگر k>=3 باشد آیا میتوان از روش تورنمنت(حذفی) استفاده کرد؟

7 , 24 , 17 , 18 , 30 , 5 , 40 , 13
24 18 30 40
24 40
40

ماکزیمم سوم عدد 24 می باشد که در بین اعداد نشان داده شده می باشد

7 , 24 , 17 , 18 , 14 , 5 , 40 , 13
24 18 14 40
24 40
40

ماکزیمم سوم عدد 18 می باشد که در بین اعداد نشان داده شده نمی باشد

مسئله انتخاب selection

- روش کلی پیدا کردن nامین ماکزیمم ویا nامین مینیمم
  - استفاد از روش partition
- در این روش ابتدا یک عنصر را انتخاب می کنیم که به آن عنصر لولا(pivot) می گوییم سپس مکان مناسب عنصر لولا در آرایه را پیدا کرده و عنصر لولا را درآن مکان قرار می دهیم مکان مناسب عنصر لولا باید دارای خصوصیات زیر باشد
- ۱-اگر هدف پیدا کردن kامین ماکزیمم باشد، عنصر لولا را در مکانی قرار می دهیم که تمام عناصر سمت راست آن از عنصر لولا بزرگتر باشند و تمام عناصر سمت چپ عنصر لولا از آن کوچکتر باشند.
- ۲-اگر هدف پیدا کردن kامین مینیمم باشد، عنصر لولا را در مکانی قرار می دهیم که تمام عناصر سمت راست آن از عنصر لولا کوچکتر باشند و تمام عناصر سمت چپ لولا از عنصر لولا بزرگتر باشند.
- پس از مشخص شدن مکان مناسب عنصر لولا آن گاه چک می کنیم ببینیم موقعیت مکان عنصر لولا نسبت به k چگونه است
- اگر با هم برابر بودند که عنصر لولا kامین ماکزیمم یا مینیمم میباشد
- اگر مکان عنصر لولا بشتر از k باشد عملیات partition را در نیمه اول آرایه انجام می دهیم
- اگر مکان عنصر لولا کمتر از k باشد عملیات partition را در نیمه دوم آرایه انجام می دهیم

مسئله انتخاب selection

- روش کلی پیدا کردن nامین ماکزیمم ویا nامین مینیمم

```
void partition(int a[], int left, int right,int &pivot_point)
{
        if (left<right)
        {
                int i = left, j = right + 1, pivot = a[left];
                do
                {
                        do
                        {i++;} while (a[i]>pivot);
                        do
                        {j--;} while (a[j]<pivot);
                        if (i<j)
                                swap(a[i], a[j]);
                } while (i<j);
                swap(a[left], a[j]);
                pivot_point = j;
        }
}
```

مسئله انتخاب selection

- روش کلی پیدا کردن nامین ماکزیمم ویا nامین مینیمم

```
int selection(int a[], int left, int right, int k)
{
        if (left == right)
                return a[left];
        else
        {
                int pivot_point;
                partition(a, left, right, pivot_point);
                if (pivot_point == k)
                        return a[pivot_point];
                if (pivot_point < k)
                        return selection(a, pivot_point + 1, right, k);
                else
                        return selection(a,left, pivot_point - 1, k);
        }
}
```

مسئله انتخاب selection

- مرتبه اجرایی selection
- در بهترین حالت و حال متوسط
- انتظار داریم که هر فراخوانی دوباره تابع selection به اندازه تقریبا نصف آرایه اصلی باشد و از طرفی مرتبه اجرایی تابع partition برابر با O(n) می باشد بنابراین

$$T(n)=n+\frac{n}{2}+\frac{n}{4}+\frac{n}{8}+\cdots=n\left(1+\frac{1}{2}+\frac{1}{4}+\frac{1}{8}+\cdots\right)=O(n)$$

مسئله انتخاب selection

- مرتبه اجرایی selection
- بدترین حالت
  - زمانی اتفاق می افتد که آرایه از قبل مرتب باشد. در این حالت یک سمت عنصر لولا عنصری وجود ندارد و در سمت دیگر آن تعداد n-1عنصر وجود دارد و مجبور شویم در سمت n-1 عنصر جستجو را انجام دهیم

$$T(n)=\begin{cases}1 & n=1\\ T(n-1)+n & n>1\end{cases}$$

$$T(n)=T(n-1)+n=T(n-2)+(n-1)+n=\dots=1+2+3+\dots+(n-1)+n=\frac{n(n+1)}{2}\rightarrow O(n^2)$$

روش تقسیم و غلبه

- در چه زمان هایی از روش تقسیم و غلبه استفاده نمی کنیم؟
- هنگامی که یک مسئله به طول n به بیش از یک مسئله به طول تقریبا n تبدیل شود آنگاه اگر از روش تقسیم و غلبه برای حل مسئله استفاده کنیم، مرتبه اجرایی نمایی خواهد شد.

مثال

$T(n)=2T(n-1)+ n \qquad n>1$

- اگر یک مسئله به اندازه n به تقریبا n مسئله به اندازه $\frac{n}{C}$ تبدیل شود آنگاه روش تقسیم و غلبه برای حل مسئله روش مناسبی نیست.

مثال

$T(n)=n\ T\left(\frac{n}{5}\right)$

برنامه نویسی پویا
(Dynamic Programming)

- این روش یک روش پایین به بالا (down-up) برای حل مسئله است
- در این روش ابتدا مسئله کوچکتر حل شده و نتایج مربوط به آن ذخیره شده و سپس از این نتایج برای حل مسئله بزرگتر استفاده می شود.

- مثال: مسئله nامین جمله از سری فیبوناچی را به دو روش برنامه نویسی تقسیم و غلبه و پویا پیاده سازی کنید و با هم مقایسه کنید.

برنامه نویسی پویا
(Dynamic Programming)

- مثال: مسئله nامین جمله از سری فیبوناچی را به دو روش برنامه نویسی تقسیم و غلبه و پویا پیاده سازی کنید و با هم مقایسه کنید.

تقسیم و غلبه

```
unsigned fib1(unsigned n)
{
  if (n == 1 || n == 2)
     return 1;
  else
  {
     return fib(n - 1) + fib(n - 2);
  }
}
```

مرتبه اجرایی O(n)

پویا

```
const int m = 100;
int fib[m];
unsigned fib2(unsigned n)
{
  if (n == 1 || n == 2)
     return 1;
else
  {  fib[0] = 1;
     fib[1] = 1;
     for (int i = 2; i < n; i++)
          fib[i] = fib[i - 1] + fib[i - 2];
     return fib[n - 1];
  }
}
```

مرتبه اجرایی نمایی