هدف : ماتریس ۱ تا m را به چه روشی با یکدیگر ضرب کنیم تا کمترین هزینه ضرب و جمع را بپردازیم ؟

$$\mathbf{M_1 \times M_2 \times M_3 \times \dots \times M_n}$$

مثال :

فرض می کنیم چهار ماتریس داریم به طوری که سطر و ستون آن به ترتیب زیر باشد :

ماتریس	M_1	M_2	M_3	M_4
ستون×سطر	2×5	5×3	3×8	8×1

ماتریس	M_1	M_2	M_3	M_4
ستون×سطر	2×5	5×3	3×8	8×1

ضرب زنجیره ای ماتریس ها

ترتیب‌های مختلف ضرب ماتریس‌ها ← هزینه‌های (تعداد عمل ضرب) مختلف

به عنوان مثال

$$((M_1 \times M_2) \times M_3) \times M_4 = \boxed{(2 \times 5 \times 3)} + (2 \times 3 \times 8) + (2 \times 8 \times 1)$$

تعداد ضرب عددی لازم برای ضرب ماتریسهای M1 , M2

$$= 30 + 48 + 16 = 94$$

$$(M_1 \times M_2) \times (M_3 \times M_4) = (2 \times 5 \times 3) + (3 \times 8 \times 1) + (2 \times 3 \times 1)$$

$$= 30 + 24 + 6 = 60$$

بنابر این تعداد کل حالات ضرب n ماتریس از رابطه‌ی بازگشتی ذیل محاسبه می‌شود :

$$T(n)=T(1)T\boxed{(n-1)+}T(2)T(n-2)+T(3)T(n-3)+\ldots+T(n-1)T(1) = \boxed{\sum_{i=1}^{n-1} T(i)T(n-i)}$$

تعداد حالت ممکن برای ضرب ماتریسهای دوم تا nاُم

ضرب زنجیره ای ماتریس ها

تعداد حالات های ضرب n+1 ماتریس برابر با عدد کاتالان است

اگر n برابر ۴ باشد آنگاه عدد کاتالان برابر ۵ است . به عبارت دیگر تعداد کل حالات ممکن ضرب زنجیره ای ۴ ماتریس برابر ۵ است .

$$((M_1 \times M_2) \times M_3) \times M_4$$
$$(M_1 \times (M_2 \times M_3)) \times M_4$$
$$(M_1 \times M_2) \times (M_3 \times M_4)$$
$$M_1 \times ((M_2 \times M_3) \times M_4)$$
$$M_1 \times (M_2 \times (M_3 \times M_4))$$

- در این مسئله ، هدف یافتن بهینه ترین حالت ضرب است ، از نظر کمترین تعداد اعمال محاسباتی

ضرب زنجیره ای ماتریس ها

گام اول :

در گام اول درخت مربوط به انواع شکستن را رسم می کنیم .

ضرب زنجیره ای ماتریس ها

فرض کنید

ماتریس M_i دارای ابعاد $r_{i-1} * r_i$ باشد

$M_{i,j}$ حداقل تعداد عملیات ضرب برای محاسبه $M_i \times M_{i+1} \times \ldots M_j$

برای محاسبه $M_{i,j}$ ضرب $M_i \times \ldots \times M_j$ را به صورت زیر می نویسیم

$(M_i \times \ldots \times M_k)(M_{k+1} \times \ldots \times M_j)$

بنابر این $M_{i,j}$ از رابطه زیر بدست می آید

$$M_{i,j} = \min(M_{i,k} + M_{k+1,j} + r_{i-1} * r_k * r_j)$$

$$i \le k < j$$

مرحله ۰ : هیچ عمل ضرب ماتریسی. معادل با $M_{i,i} = 0$ $i=1,2,\ldots,n$ می باشد

مرحله ۱ : یک عمل ضرب ماتریسی. در این مرحله کلیه $M_{i,i+1}$, $i=1,2,\ldots,n-1$ را محاسبه و ذخیره می کنیم

مرحله 2 : دو عمل ضرب ماتریسی. در این مرحله کلیه $M_{i,i+2}$, $i=1,2,\ldots,n-2$ را محاسبه و ذخیره می کنیم

مرحله n-1 : n-1 عمل ضرب ماتریسی. در این مرحله $M_{1,n}$ محاسبه می شود

ضرب زنجیره ای ماتریس ها

مثال :

حداقل تعداد ضرب لازم برای ضرب ماتریس‌های Mi تا Mj:

$$M_{i,j} = \min_{i<=k<j}(M_{i,k} + M_{k+1,j} + r_i * r_k * r_j)$$

M_1	M_2	M_3	M_4
2×5	5×3	3×8	8×1

M11 = M22 = M33 = M44 = 0

M12 = min (M11 + M22 + 2 × 5 × 3) = 30

M23 = min (M22 + M33 + 5 × 3 × 8) = 120

M34 = min (M33 + M44 +3 × 8 × 1) = 24

M13 = min (M11 + M23 + 2 × 5 × 8 , M12 + M33 + 2 × 3 × 8)

= min (120+80 , 30+48) = min(200 , 78) = 78

M24 = min (M22 + M34 + 5 × 3 × 1 , M23 + M44 + 5 × 8 × 1)

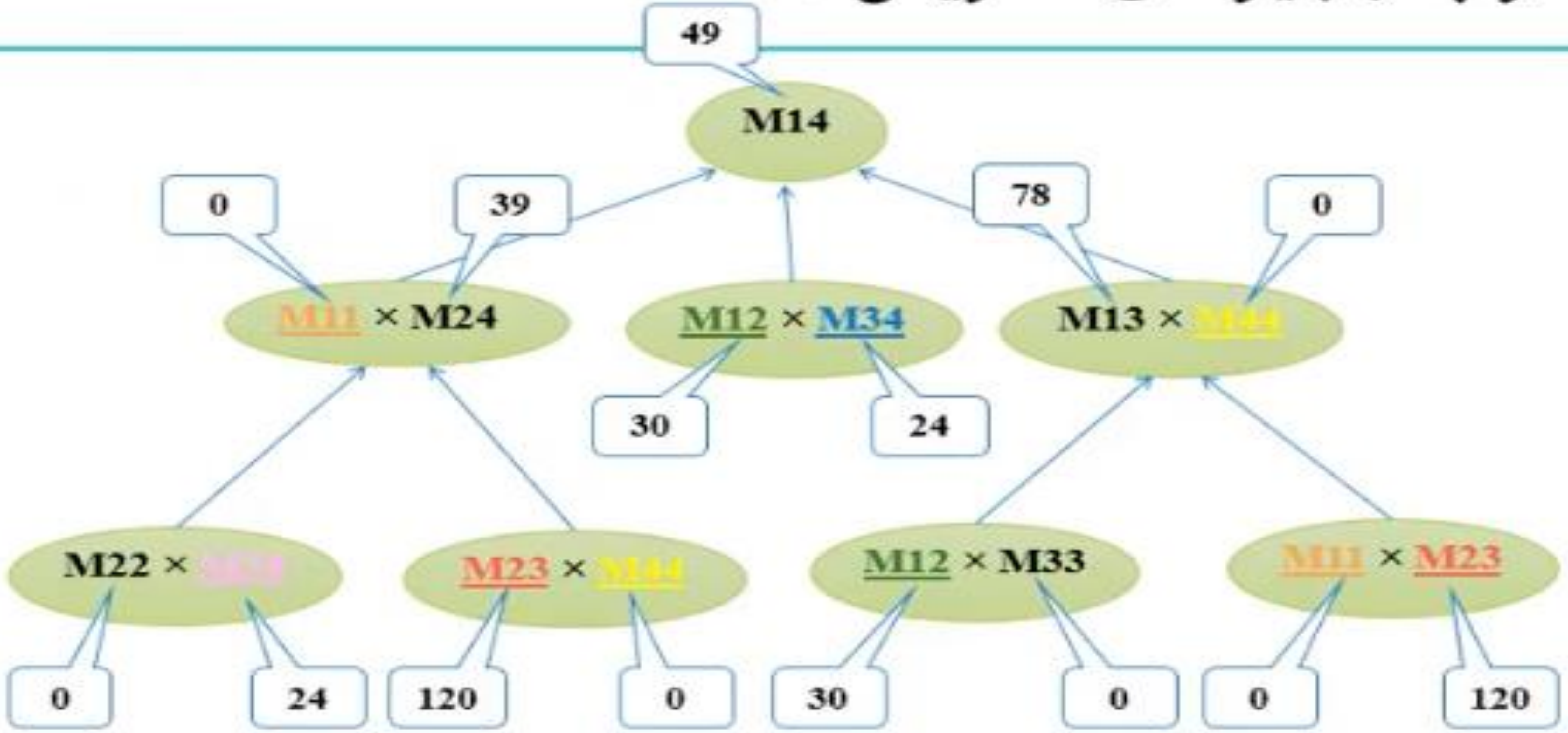
= min (24+15 , 120+40) = min(39 , 160) = 39

M14 = min (M11 + M24 + 2×5×1 , M12+M34 +2×3×1 , M13+M44+ 2×8 ×1)

= min (39+10 , 30+24+6 , 78+16) = min (49,60,94) = 49

M14=49			
M13=78	M24=39		
M12=30	M23=120	M34=24	
M11=0	M22=0	M33=0	M44=0

ضرب زنجیره ای ماتریس ها



ضرب زنجیره ای ماتریس ها

```python
def MatrixChainOrder(p, n):
    m = [[0 for x in range(n)] for x in range(n)]
    # cost is zero when multiplying one matrix.
    for i in range(1, n):
        m[i][i] = 0
    # L is chain length.
    for L in range(2, n):
        for i in range(1, n-L + 1):
            j = i + L-1
            m[i][j] = float('inf')
            for k in range(i, j):
                # q = cost / scalar multiplications
                q = m[i][k] + m[k + 1][j] + p[i-1]*p[k]*p[j]
                if q < m[i][j]:
                    m[i][j] = q
    return m[1][n-1]
```

```python
arr = [2, 5, 3, 8,1]
size = len(arr)
print("Minimum number of multiplications
    str(MatrixChainOrder(arr, size)))
```

- Space Complexity: $\theta\left(n^2\right)$
- Time Complexity: $\theta\left(n^3\right)$

ضریب چند جمله ای

$$\binom{n}{k} = \frac{n!}{(n-k)!k!}$$

محاسبه !k و !n برای اعداد بزرگ کمی مشکل است، می توان برای بدست آوردن ضریب چند جمله ای از رابطه بازگشتی زیر استفاده کرد

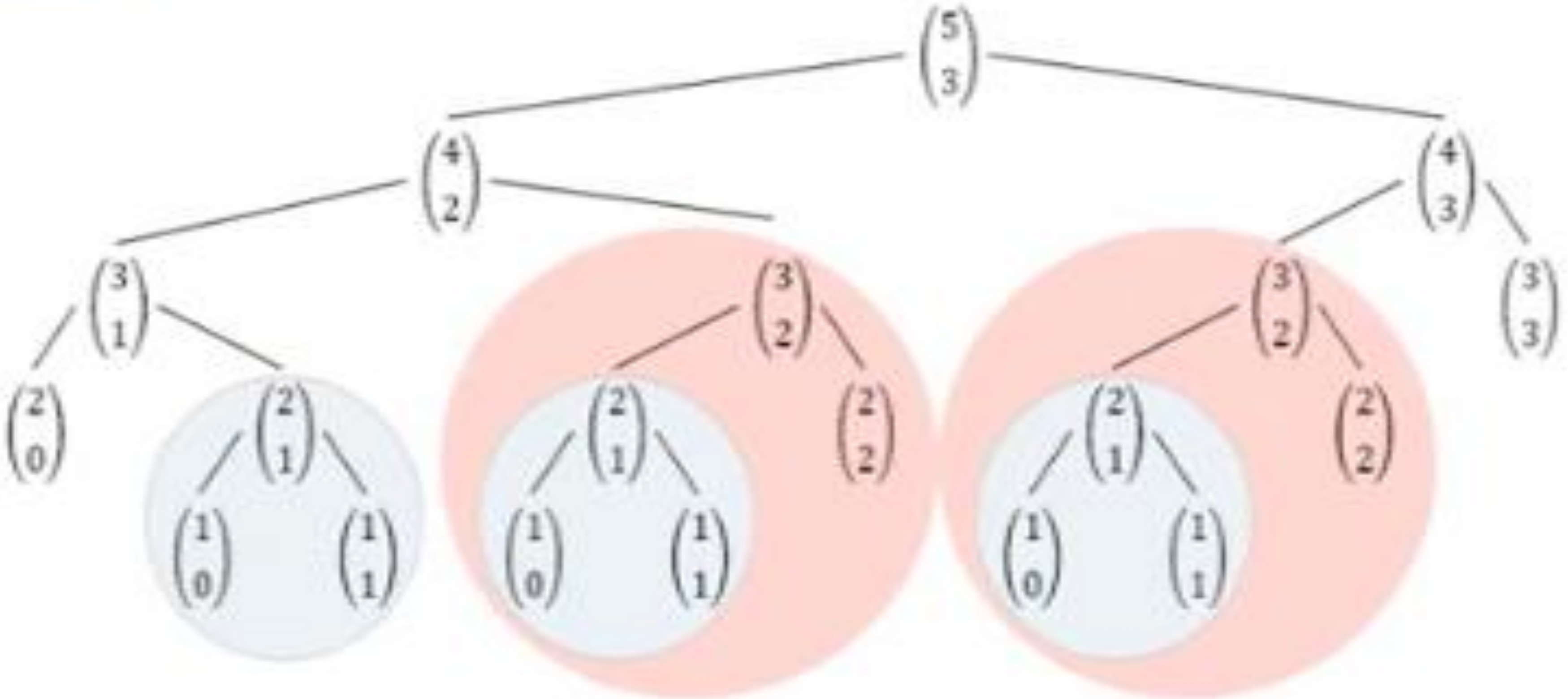
$$\binom{n}{k} = \begin{cases} 1 & k=0 \| k=n \\ \binom{n-1}{k-1} + \binom{n-1}{k} & 0<k<n \end{cases}$$

ضریب چند جمله ای (روش تقسیم و حل)

```python
def bin_c(n,k) :
   if (k == 0 or n==k):
     return 1
   else:
     return bin_c(n-1,k-1)+bin_c(n-1,k)

n= 5
k=3
print(bin_c(n,k))
```

$$\binom{n}{k} = \begin{cases} 1 & k=0 \,||\, k=n \\ \binom{n-1}{k-1} + \binom{n-1}{k} & 0<k<n \end{cases}$$



ضریب چند جمله ای

برنامه نویسی پویا

در روش پویا از یک ماتریس (B) برای ذخیره اعداد میانی استفاده می شود.

مراحل

ارائه ی یک خاصیت بازگشتی به منظور محاسبه راه حل یک نمونه و با توجه به راه حل نمونه های کوچک تر

$$B[i][j]=\begin{cases}1 & j=0, j=i\\ B[i-1][j-1]+B[i-1][j] & 0\le j\le i\end{cases}$$

حل نمونه ها از پایین به بالا با استفاده از رابطه ی بازگشتی و ذخیره راه حل ها در ماتریس B

ضریب چند جمله ای

مثال

K →

N ↓

N \ K	0	1	2	3
0	1			
1	1	1		
2	1	2	1	
3	1	3	3	1
4	1	4	6	4
5	1	5	10	10

$$\binom{5}{3}$$

پیاده سازی ضریب چند جمله ای (روش پویا)

```python
def binomialCoef(n, k):
    C = [[0 for x in range(k+1)] for x in range(n+1)]
    # Calculate value of Binomial Coefficient in bottom up manner
    for i in range(n+1):
        for j in range(min(i, k)+1):
            # Base Cases
            if j == 0 or j == i:
                C[i][j] = 1
            # Calculate value using previously stored values
            else:
                C[i][j] = C[i-1][j-1] + C[i-1][j]
    return C[n][k]
n=5
k=3
print("Result using Dynamic Programming Method id:   " , binomialCoef(n,k))
```

درخت جستجوی دودوئی بهینه (Optimal BST)

مسئله

فرض کنید n کلید متمایز k1<k2<k3 وجود دارد. احتمال آنکه کلید Ki جستجو کنیم برابر با pi است. می خواهیم با این کلید ها BST بسازیم به طوریکه میانگین زمان جستجو به صورت رابطه زیر مینیمم باشد

$$\sum_{i=1}^{n} C_i P_i$$

Pi : احتمال اینکه k i مورد جستجو قرار بگیرد

Ci : تعداد مقایسه لازم برای یافتن ki

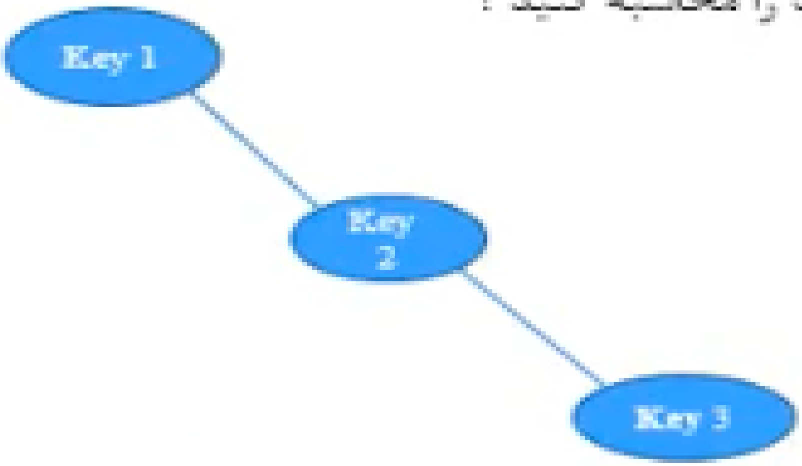
- تعداد درختان bst مجزاکه میتوان با n کلید رسم کرد (عدد کاتالان).

$$\frac{1}{n+1}\binom{2n}{n}$$

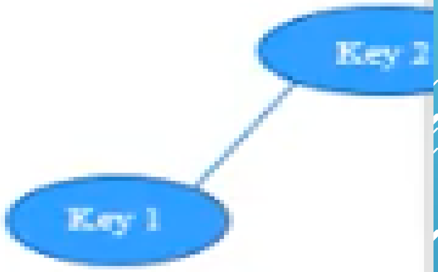
درخت جستجوی دودوئی بهینه
(Optimal BST)

مثال : با سه کلید **key1<key2<key3** ۵ درخت جستجوی دودوئی ساختیم. با فرض p1= 0.7 ، p2=0.2 و p3=0.1 باشد مانگین زمان جستجو برای هر درخت را محاسبه کنید ؟

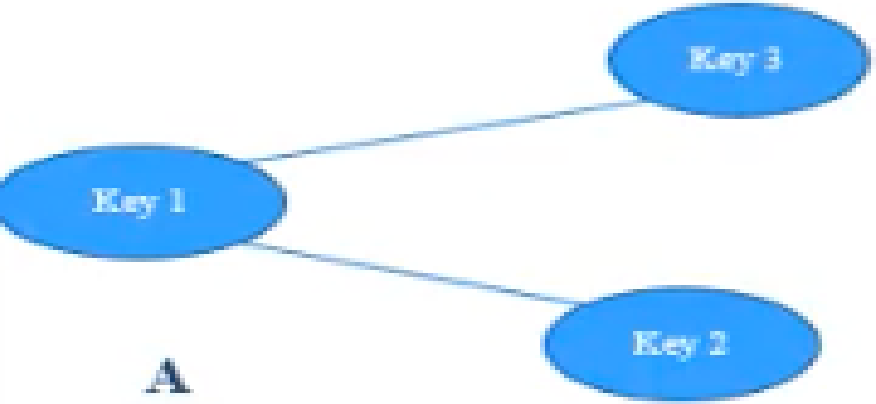
a) 2(0.7)+3(0.2)+1(0.1)=2.1
b) 1(0.7)+3(0.2)+2(0.1)=1.5
c) 3(0.7)+2(0.2)+1(0.1)=2.6
d) 2(0.7)+1(0.2)+2(0.1)=1.8
e) 1(0.7)+2(0.2)+3(0.1)=1.4



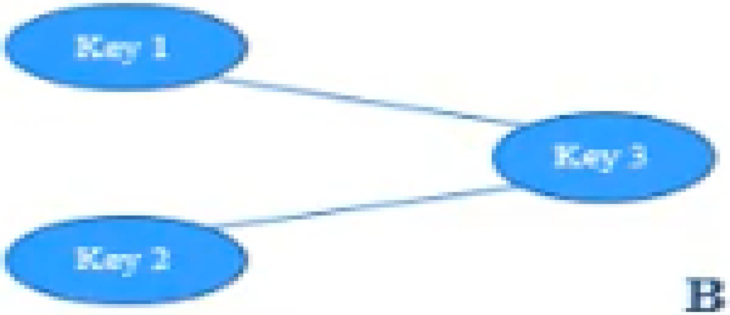
E



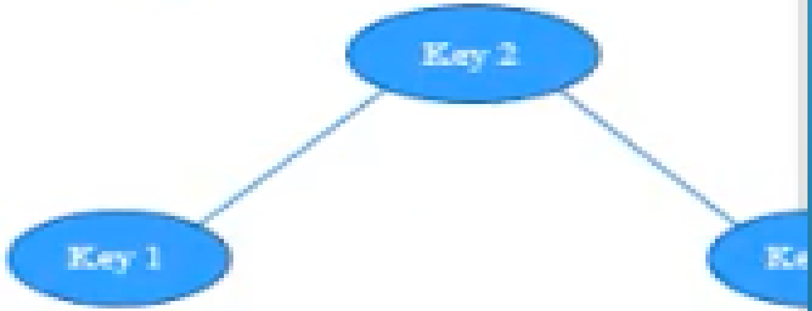
C



A

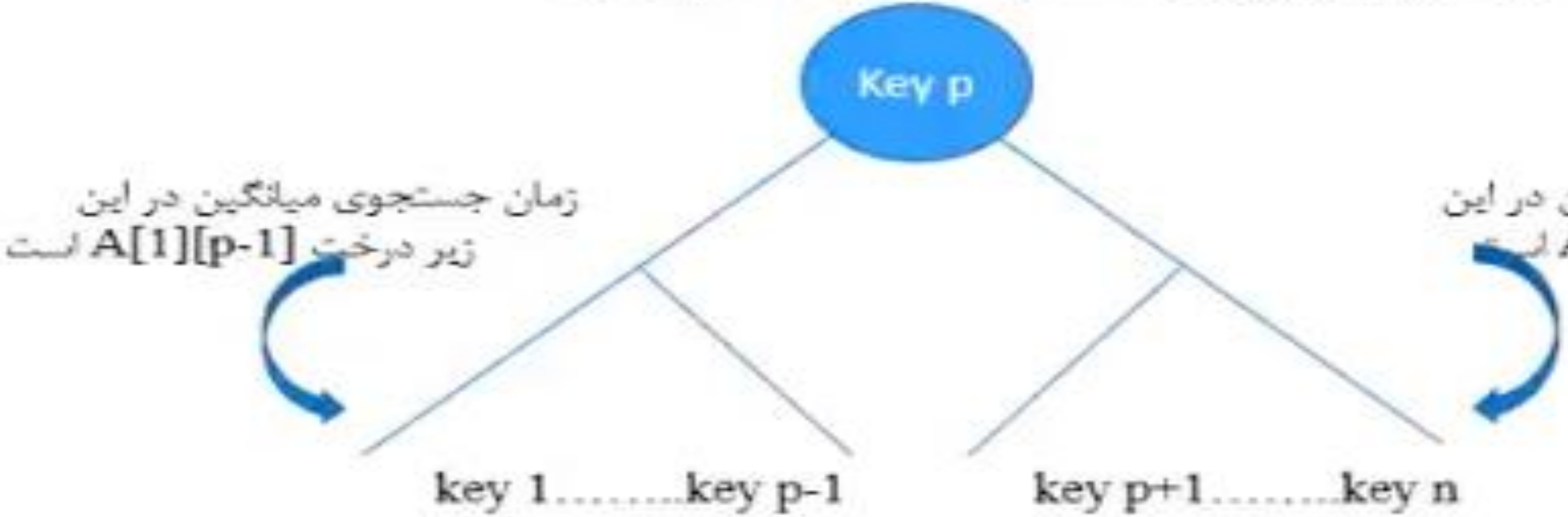


B



درخت جستجوی دودوئی بهینه (Optimal BST)

- زمان جستجوی میانگین در زیر درخت جستجوی دودویی بهینه شامل کلیدهای i تا j ام: A[i][j]
- هدف (جواب) مساله : A[1][n]
- برای حل مساله بصورت بازگشتی کافی است در هر مرحله یکی از کلیدها را در ریشه قرار داده (حالت بهینه) و سایر کلیدها را در سمت راست و چپ آن تقسیم کنیم، سپس برای هر زیردرخت سمت راست و چپ مساله تکرار میشود.



$$A[1,k-1]+A[k+1,n]+\sum_{m=1}^{n} p_m$$