Chapter 3

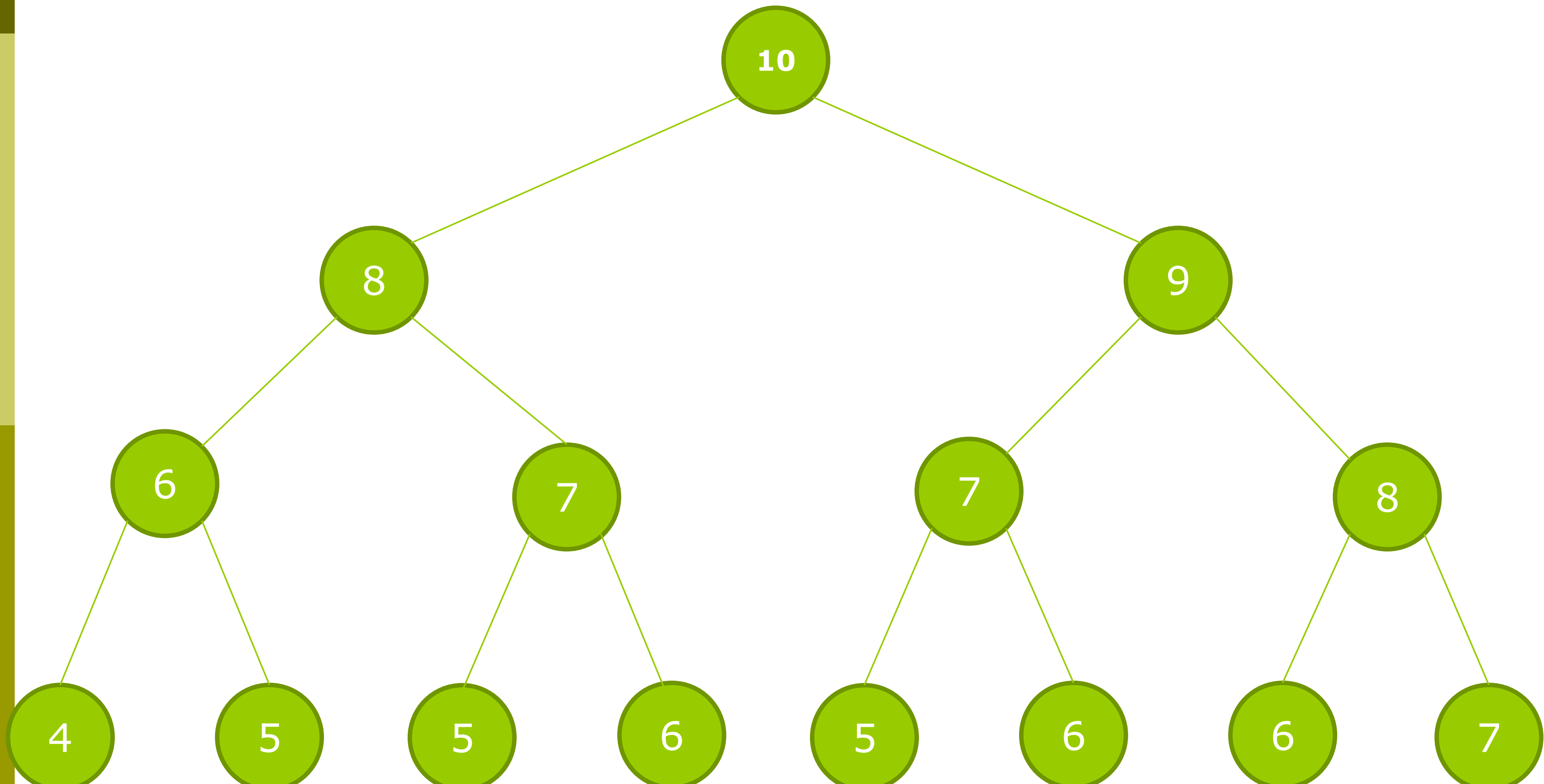
Dynamic Programming
&
Catalan Numbers

برنامه‌نویسی پویا و اعداد کاتالان

چه زمانی روش تقسیم و حل ناکارامد است؟

- پس از تقسیم ...
  - مسایل کوچکتر باهم ارتباط ندارند؛ مانند جستجوی ادغامی
    - در این حالت تقسیم و حل مناسب است.
  - مسایل کوچکتر باهم ارتباط دارند؛ مانند رشته فیبوناچی
    - در این حالت تقسیم و حل مناسب نیست. زیرا یک مساله به صورت تکراری حل می‌شود.
- برنامه نویسی پویا
  - روشی پایین به بالا
  - butom-up approach
  - نیاز به ساختمان داده مناسب برای ذخیره سازی راه‌حل‌های میانی

فیبوناچی
چندین حل تکراری برای محاسبه یک مقدار



الگوریتم فیبوناچی - یادآوری

- Algorithm 1.7: *n*th Fibonacci Term (Iterative)

```
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 ];
      }
```

گام‌های حل مساله

- ایجاد یک رابطه بازگشتی برای حل مساله
- حل این رابطه بازگشتی به صورت پایین به بالا

The binomial coefficient
ضرایب دوجمله‌ای

- Definition

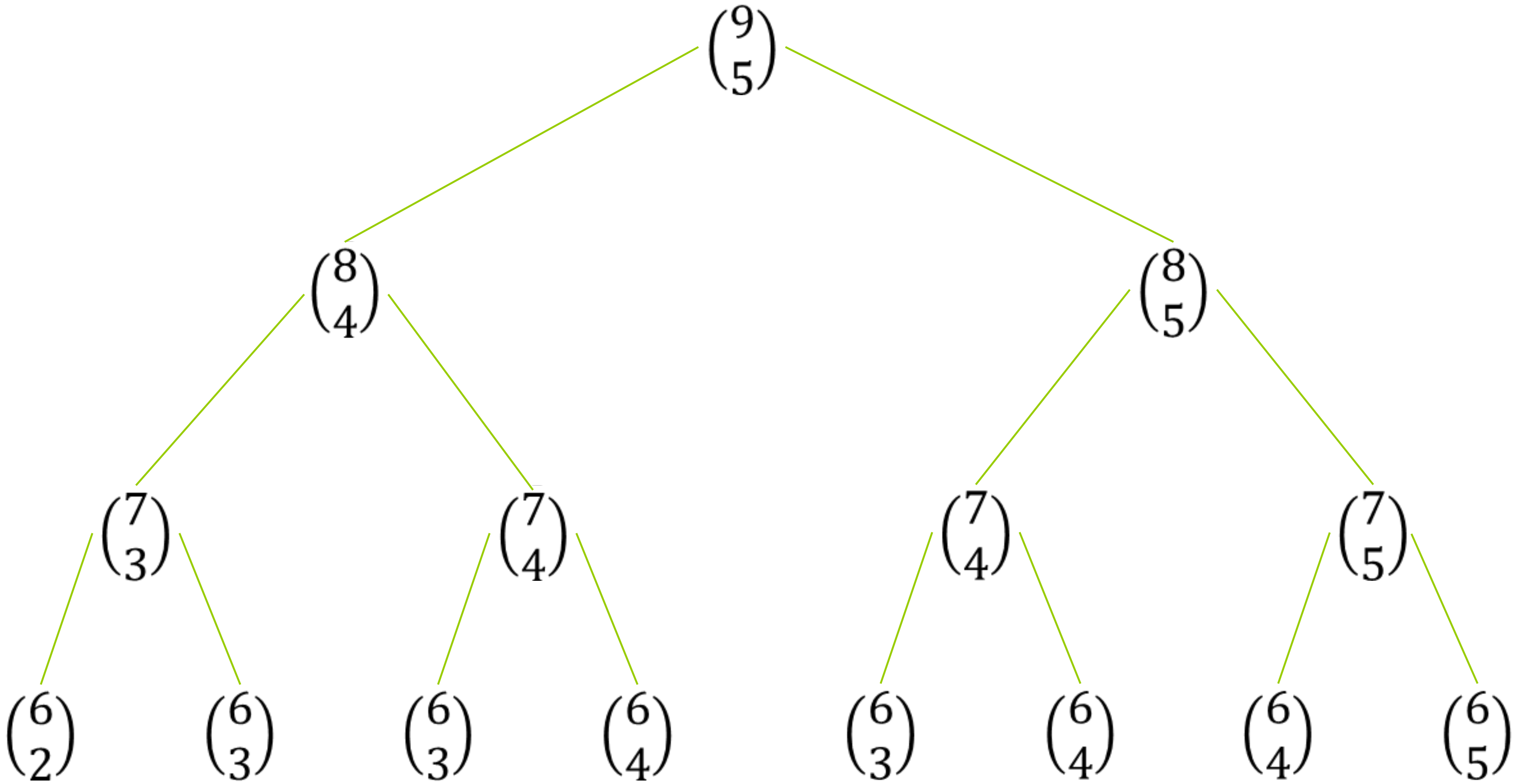
$$\binom{n}{k} = \frac{n!}{k!(n-k)!} \qquad for \quad 0 \le k \le n$$

- Recursive definition

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

The binomial coefficient

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



Binomial Coefficient Using Divide-and-Conquer

```
int bin (int n, int k) {
    if ( k = = 0 || n = = k)
            return 1;
    else
            return bin (n-1, k - 1)+bin (n - 1, k);
    }
```

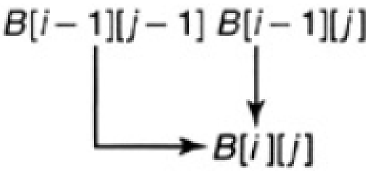
Using dynamic programming

- استفاده از آرایه B برای ذخیره سازی ضرایب
- گام‌ها
  - **ایجاد رابطه بازگشتی**

$$B[i][j] = \begin{cases} B[i-1][j-1] + B[i-1][j] & 0 < j < i \\ 1 \quad j = 0 \quad or \quad j = i \end{cases}$$

  - **حل به صورت پایین به بالا**

	0	1	2	3	4	j	k
0	1						
1	1	1					
2	1	2	1				
3	1	3	3	1			
4	1	4	6	4	1		
i							
n							



Binomial Coefficient Using Dynamic Programming

```
int bin2 (int n, int k) {
      index i, j;
      int B[0..n] [0..k];
      for (i = 0; i <= n; i++)
            for (j = 0; j <= minimum (i, k); j++)
                  if (j = = 0 || j = = i)
                        B[i][ j] = 1;
                  else
                  B[i][j] = B[i - 1][j - 1] + B[i - 1] [j];
      return B[n][k];
}
```

Time complexity

- The number of passes through the for-*j* loop for each value of *i*

i	**0**	**1**	**2**	**3**	**…**	k	$k+1$	**…**	n
Number of passes	1	2	3	4	…	$k+1$	$k+1$	…	$k+1$

$$1+2+3+4+\cdots+k+\underbrace{(k+1)+(k+1)+\cdots+(k+1)}_{n-k+1 \quad times}=$$

$$\frac{k(k+1)}{2}+(n-k+1)(k+1)=\frac{(2n-k+2)(k+1)}{2}\in\Theta(nk)$$

Floyd's algorithm for shortest paths
کوتاهترین راه از هر شهر به شهر دیگر

مساله‌ای که مسافران هوایی با آن مواجه هستند، تعیین کوتاه‌ترین راه برای پرواز از شهری به شهر دیگر است.

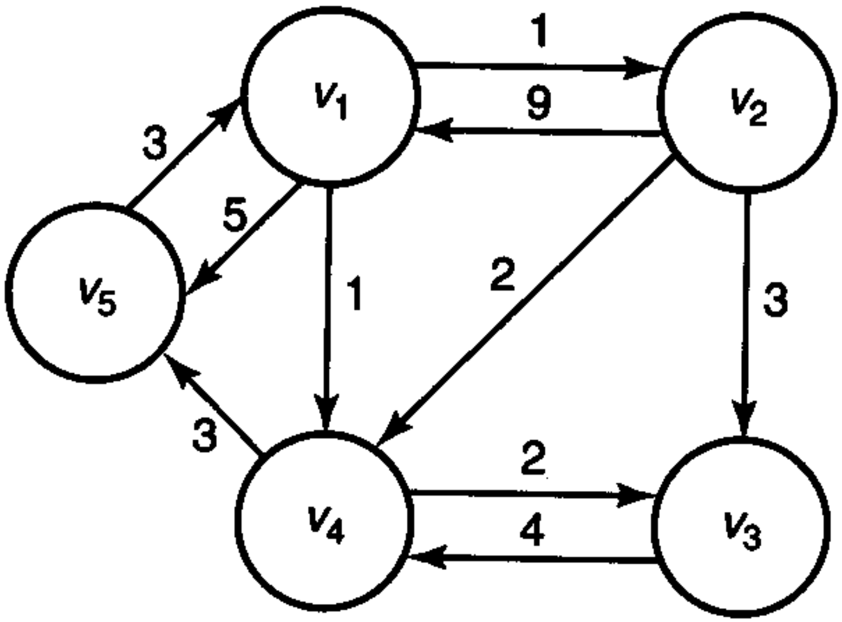


Figure 3.2 ● A weighted, directed graph.

مساله بهینه سازی
Optimization Problem

- تعریف:
  - بیش از یک راه حل وجود داشته باشد.
  - هر راه حل هزینه‌ای دارد.
  - هدف، یافتن راه حل با کمترین هزینه است.
- مساله فلوید یک مساله بهینه‌سازی است.

نمایش گراف با استفاده از یک ماتریس مجاورت (Adjacency matrix)

$$W[i][j] = \begin{cases} weight \quad on \quad edge \quad if \quad there \quad is \quad an \quad edge \quad from \quad v_i \quad to \quad v_j \\ \infty \quad if \quad there \quad is \quad no \quad edge \quad from \quad v_i \quad to \quad v_j \\ 0 \quad if \quad i = j \end{cases}$$

	1	2	3	4	5
1	0	1	∞	1	5
2	9	0	3	2	∞
3	∞	∞	0	4	∞
4	∞	∞	2	0	3
5	3	∞	∞	∞	0

W

	1	2	3	4	5
1	0	1	3	1	4
2	8	0	3	2	5
3	10	11	0	4	7
4	6	7	2	0	3
5	3	4	6	4	0

D

Figure 3.3 • *W* represents the graph in Figure 3.2 and *D* contains the lengths of the shortest paths. Our algorithm for the Shortest Paths problem computes the values in *D* from those in *W*.

الگوریتم فلوید

- Create a sequence of $n+1$ arrays $D^{(k)}$, where $0 \le k \le n$ and where

  $D^{(k)}[i][j]$ = length of a shortest path from v_i to v_j using only vertices in the set $\{v_1, v_2, \ldots, v_k\}$ as intermediate vertices

  $D^{(0)} = W$ and $D^{(n)} = D$

- Try to determine $D[2][5]$

To compute $D^{(k)}$ from $D^{(k-1)}$

- Case 1. At least one shortest path from v_i to v_j, using only vertices in $\{v_1, v_2, \ldots, v_k\}$ as intermediate vertices, does not use v_k. Then $D^{(k)}[i][j] = D^{(k-1)}[i][j]$
- Case 2. All shortest paths from v_i to v_j, using only vertices in $\{v_1, v_2, \ldots, v_k\}$ as intermediate vertices, do use v_k. Then $D^{(k)}[i][j] = D^{(k-1)}[i][k] + D^{(k-1)}[k][j]$

Put together

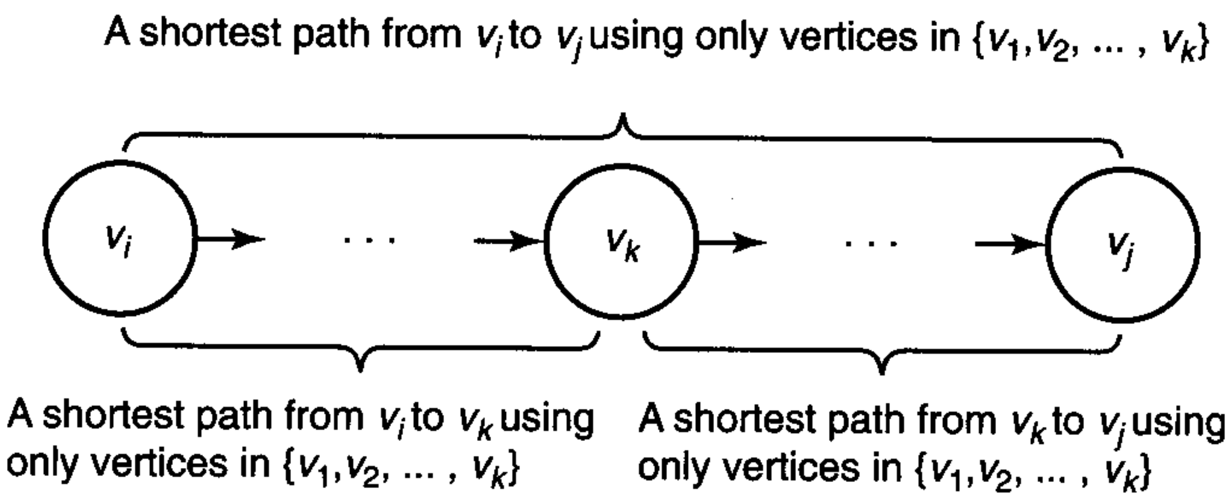
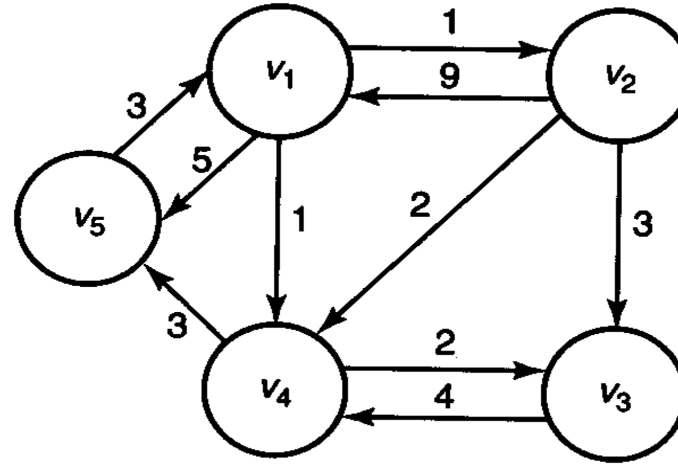


Figure 3.4 ● The shortest path uses v_k.

- $D^{(k)}[i][j] = minimum(D^{(k-1)}[i][j], D^{(k-1)}[i][k] + D^{(k-1)}[k][j])$
- Try to compute $D^{(2)}[5][4]$

$$D^{0}=\begin{bmatrix} 0 & 1 & \infty & 1 & 5 \\ 9 & 0 & 3 & 2 & \infty \\ \infty & \infty & 0 & 4 & \infty \\ \infty & \infty & 2 & 0 & 3 \\ 3 & \infty & \infty & \infty & 0 \end{bmatrix}$$



$$D^{1}[1][2]=Min\{D^{0}[1][2],D^{0}[1][1]+D^{0}[1][2]\}=1$$

$$D^{1}[1][3]=Min\{D^{0}[1][3],D^{0}[1][1]+D^{0}[1][3]\}=\infty$$

$$D^{1}[1][4]=Min\{D^{0}[1][4],D^{0}[1][1]+D^{0}[1][4]\}=1$$

$$D^{1}[1][5]=Min\{D^{0}[1][5],D^{0}[1][1]+D^{0}[1][5]\}=5$$

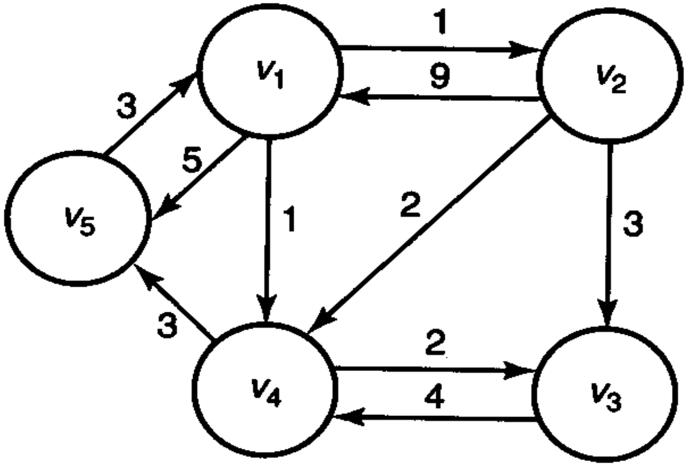
$$D^{1}[2][1]=Min\{D^{0}[2][1],D^{0}[2][1]+D^{0}[1][1]\}=Min\{9,9+0\}=9$$

$$D^{1}[2][3]=Min\{D^{0}[2][3],D^{0}[2][1]+D^{0}[1][3]\}=Min\{3,9+\infty\}=3$$

$$D^{1}[2][4]=Min\{D^{0}[2][4],D^{0}[2][1]+D^{0}[1][4]\}=Min\{2,9+1\}=2$$

$$D^{1}[2][5]=Min\{D^{0}[2][5],D^{0}[2][1]+D^{0}[1][5]\}=Min\{\infty,9+5\}=14$$

$$D^0 = \begin{bmatrix} 0 & 1 & \infty & 1 & 5 \\ 9 & 0 & 3 & 2 & \infty \\ \infty & \infty & 0 & 4 & \infty \\ \infty & \infty & 2 & 0 & 3 \\ 3 & \infty & \infty & \infty & 0 \end{bmatrix} \quad D^1 = \begin{bmatrix} 0 & 1 & \infty & 1 & 5 \\ 9 & 0 & 3 & 2 & \underline{14} \\ \infty & \infty & 0 & 4 & \infty \\ \infty & \infty & 2 & 0 & 3 \\ 3 & \underline{4} & \infty & \underline{4} & 0 \end{bmatrix}$$



$$D^2 = \begin{bmatrix} 0 & 1 & \underline{4} & 1 & 5 \\ 9 & 0 & 3 & 2 & 14 \\ \infty & \infty & 0 & 4 & \infty \\ \infty & \infty & 2 & 0 & 3 \\ 3 & 4 & \underline{7} & 4 & 0 \end{bmatrix} \quad D^3 = \begin{bmatrix} 0 & 1 & 4 & 1 & 5 \\ 9 & 0 & 3 & 2 & 14 \\ \infty & \infty & 0 & 4 & \infty \\ \infty & \infty & 2 & 0 & 3 \\ 3 & 4 & 7 & 4 & 0 \end{bmatrix}$$

$$D^4 = \begin{bmatrix} 0 & 1 & 3 & 1 & \underline{4} \\ 9 & 0 & 3 & 2 & \underline{5} \\ \infty & \infty & 0 & 4 & \underline{7} \\ \infty & \infty & 2 & 0 & 3 \\ 3 & 4 & \underline{6} & 4 & 0 \end{bmatrix} \quad D^5 = \begin{bmatrix} 0 & 1 & 3 & 1 & 4 \\ \underline{8} & 0 & 3 & 2 & 5 \\ \underline{10} & \underline{11} & 0 & 4 & 7 \\ \underline{6} & \underline{7} & 2 & 0 & 3 \\ 3 & 4 & 6 & 4 & 0 \end{bmatrix}$$

The algorithm

```
void floyd (int n, const number W[][], number D[][])
{
index i, j, k;
D = W;
for (k = 1; k<=n; k++)
      for (i = 1; i <= n; i++)
            for (j = 1; j <= n; j++)
                  D[i][j] = minimum(D[i][j], D[i][k] + D[k][j]);
}
```

Every-case time complexity

- Basic operation: The instruction (if) in the for-*j* loop
- Input size: n, the number of vertices in the graph
- Time complexity:

  $T(n) = n \times n \times n = n^3 \in \Theta(n^3)$

Floyd's algorithm for shortest paths

```
void floyd2 (int n, const number W[][], number D[] [],
   index P[] [])
{
   index, i, j, k;
   for(i = 1; i <= n; i++)
         for (j = 1; j <= n; j++)
               P[i][j] = 0;
   D = W
   for (k = 1; k <= n; k++)
         for(i = 1; i <= n; i++)
               for(j = 1; j <= n; j++)
                     if (D[i][k] + D[k][j] < D[i][j]) {
                           P[i][j] = k;
                           D[i][j] = D[i][k] + D[k][j]; }
}
```

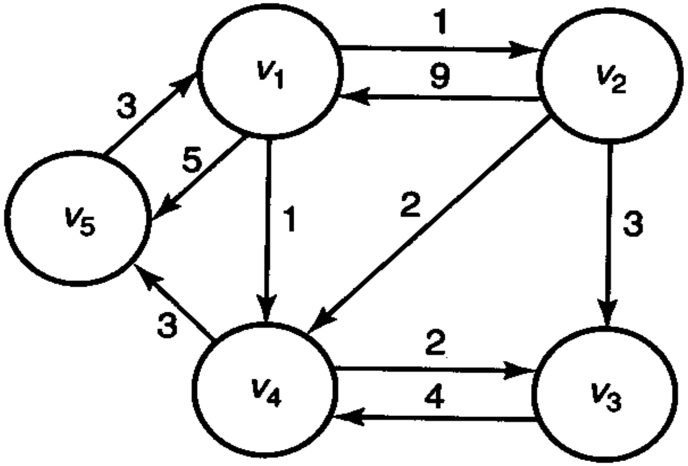
Sample output

- ماتریس P برای یافتن مسیر بهینه

	1	2	3	4	5
1	0	0	4	0	4
2	5	0	0	0	4
3	5	5	0	0	4
4	5	5	0	0	0
5	0	1	4	1	0

Figure 3.5 ● The array *P* produced when Algorithm 3.4 is applied to the graph in Figure 3.2.

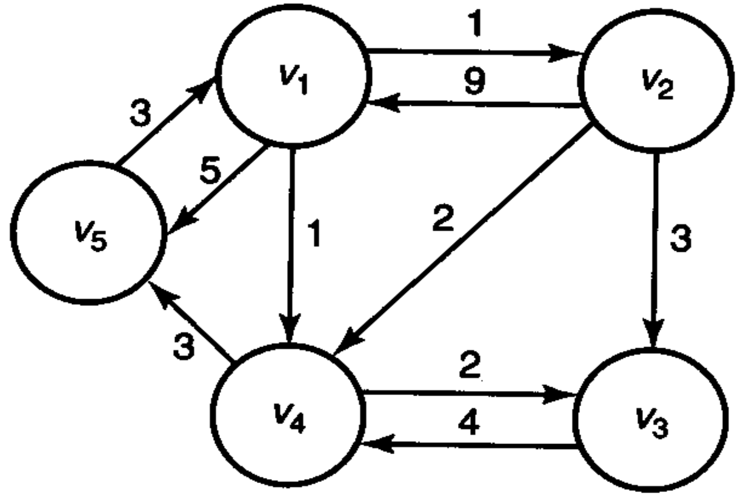
نمایش کوتاهترین مسیر




```
void path (index q, r)
{
  if (P[ q ] [ r ] ! = 0){
        path (q, P[q] [r]);
        cout < < "v" < < P[ q ] [ r ];
        path (P[ q ] [ r ], r);
  }
}
```

Try to determine path(5,3)

```
void path (index q, r)
{
  if (P[q][r] ! = 0)
  {
    path (q, P[q][r]);
    cout < < "v" < < P[q][r];
    path (P[q][r], r);
  }
}
```



$$\underbrace{path(3,1)}$$

$$\underbrace{path(3,5)} \qquad v5 \qquad \underbrace{path(5,1)}$$

$$\underbrace{path(3,4)}_{-} \qquad v4 \qquad \underbrace{path(4,5)}_{-}$$

	1	2	3	4	5
1	0	0	4	0	4
2	5	0	0	0	4
3	5	5	0	0	4
4	5	5	0	0	0
5	0	1	4	1	0

Dynamic programming and optimization problems

- **گام‌ها برای حل یک مساله به روش پویا**
  - یافتن یک رابطه بازگشتی که راه حل بهینه را به دست دهد.
  - محاسبه هزینه راه‌حل بهینه به روش پایین به بالا
  - یافتن راه‌حل بهینه (مسیر بهینه) به روش پایین به بالا

Principle of optimality
اصل بهینگی – (اختیاری)

The optimal solution to the instance contains optimal solutions to all subsinstances

Principle of optimality (Example)

if v_k is a vertex on an optimal path form v_i to v_j, then the subpaths from v_i to v_k and from v_k to v_j must also be optimal.

Principle of optimality

It is necessary to show that the principle applies before using dynamic programming to obtain the solution

Longest simple paths problem

- The longest path from v_1 to v_4 is [v_1, v_3, v_2, v_4]. However, the subpath [v_1, v_3] is not optimal

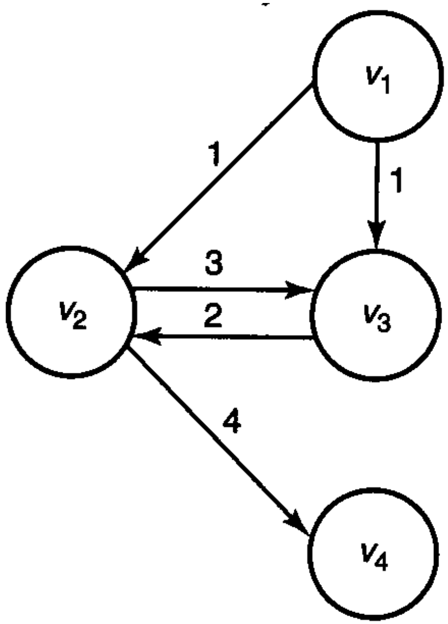


Figure 3.6 ● A weighted, directed graph with a cycle.

ضرب زنجیره‌ای ماتریس‌ها

- To multiply an $i \times j$ matrix times a $j \times k$ matrix, the number of elementary multiplications is: $i \times j \times k$ (why?)

- ترتیب‌های (اولویت‌بندی‌های) مختلفی برای ضرب چند ماتریس درهم وجود دارد که در نتیجه تعداد ضرب لازم متفاوت خواهد بود.

Example

$$A \quad \times \quad B \quad \times \quad C \quad \times \quad D$$
$$20\times 2 \quad 2\times 30 \quad 30\times 12 \quad 12\times 8$$

A(B(CD)) =>The number of multiplications =	3680
A((BC)D)	1232
(AB)(CD)	8880
(A(BC))D	3120
((AB)C)D	10320

Using dynamic programming

- Some properties
  - the number of columns in A_{k-1} equals the number of rows in A_k
  - We can let d_0 be the number of rows in A_1 and d_k be the number of columns in A_k for $1 \leq k \leq n$, the dimension of A_k is $d_{k-1} \times d_k$

Introducing a sequence of arrays

- $M[i][j]$ = minimum number of multiplications needed to multiply A_i through A_j, if $i < j$.
- $M[i][i] = 0$
- Example

$$
\begin{array}{cccccc}
A_1 & \times\ A_2 & \times\ A_3 & \times\ A_4 & \times\ A_5 & \times\ A_6 \\
5 \times 2 & 2 \times 3 & 3 \times 4 & 4 \times 6 & 6 \times 7 & 7 \times 8 \\
d_0\ \ d_1 & d_1\ \ d_2 & d_2\ \ d_3 & d_3\ \ d_4 & d_4\ \ d_5 & d_5\ \ d_6
\end{array}
$$

What is $M[4][6]$?

To multiply 6 matrices

- Possible factorizations
  - $A_1 \ (A_2 \ A_3 \ A_4 \ A_5 \ A_6 \ )$
  - $(A_1 \ A_2) \ (A_3 \ A_4 \ A_5 \ A_6 \ )$
  - $(A_1 \ A_2 \ A_3 \ ) \ (A_4 \ A_5 \ A_6 \ )$
  - $(A_1 \ A_2 \ A_3 \ A_4 \ ) \ (A_5 \ A_6 \ )$
  - $(A_1 \ A_2 \ A_3 \ A_4 \ A_5 \ ) \ A_6$

ضرب آخر

- The number of multiplications for the *k*th factorization:

  $M[1][k] + M[k+1][6] + d_0 \, d_k \, d_6$

- Therefore (Recursive Equation)

$$M[1][6] = \min_{1 \le k \le 5} imum(M[1][k] + M[k+1][6] + d_0 \, d_k \, d_6)$$

$$\begin{cases} M[i][j] = \underset{i \le k \le j-1}{\min imum}\left(M[i][k] + M[k+1][j] + d_{i-1} d_k d_j\right), \quad if \quad i < j \\ M[i][i] = 0 \end{cases}$$

Diagonal 1 Diagonal 2 Diagonal 3 Diagonal 4 Diagonal 5

	1	2	3	4	5	6	
1	0	30	64	132	226	348	Final answer
2		0	24	72	156	268	
3			0	72	198	366	
4				0	168	392	
5					0	336	
6						0	

Example

$$M[1][3] = Min\{M[1][1] + M[2][3] + d_0 d_1 d_3, M[1][2] + M[3][3] + d_0 d_2 d_3\}$$

$$= Min\{0 + 24 + 40, 30 + 0 + 60\} = 64$$

$$M[2][4] = Min\{M[2][2] + M[3][4] + d_1 d_2 d_4, M[2][3] + M[4][4] + d_1 d_3 d_4\}$$

$$= Min\{0 + 72 + 36, 24 + 0 + 48\} = 72$$

$$M[1][4] = Min\{M[1][1] + M[2][4] + d_0 d_1 d_4, M[1][2] + M[3][4] + d_0 d_2 d_4$$

$$M[1][3] + M[4][4] + d_0 d_3 d_4\}$$

$$= Min\{0 + 72 + 60, 30 + 72 + 90, 64 + 0 + 120\} = 132$$

$$\begin{array}{cccccc} A_1 & \times A_2 & \times A_3 & \times A_4 & \times A_5 & \times A_6 \\ 5 \times 2 & 2 \times 3 & 3 \times 4 & 4 \times 6 & 6 \times 7 & 7 \times 8 \\ d_0 \; d_1 & d_1 \; d_2 & d_2 \; d_3 & d_3 \; d_4 & d_4 \; d_5 & d_5 \; d_6 \end{array}$$

The algorithm

```
int minmult (int n, const int d [], index P [][] )
{
  index i, j, k, diagonal;
  int M [1 .. n][1 .. n];
  for (i = 1; i < = n; i++) M[i][i] = 0;
  for (diagonal = 1; diagonal <= n - 1; diagonal++) // Diagonal-1 is
    for (i = 1; i <= n - diagonal; i++) { // just above the
          j = i + diagonal; // main diagonal
          M[i][j] = minimum (M[i][k] + M[k + 1][j] + d[i - 1]*d[k]*d[j]);
                      i ≤ k ≤ j–1
          P[i][j] = a value of k that gave the minimum;
          }
return M[1][n];
}
```

Every-case time complexity

- Basic operation: instructions executed for each value of *k*
- Input size: *n*, the number of matrices to be multiplied
- Time complexity

$$\sum_{diagonal=1}^{n-1}[(n-diagonal)\times diagonal] = \frac{n(n-1)(n+1)}{6} \in \Theta(n^3)$$

To obtain an optimal order from array P

	1	2	3	4	5	6
1		1	1	1	1	1
2			2	3	4	5
3				3	4	5
4					4	5
5						5

Figure 3.9 ● The array *P* produced when Algorithm 3.6 is applied to the dimensions in Example 3.5.

- What is the optimal order?

Print optimal order

```
void order (index i, index j)
  {
      if (i == j)
            cout << "A" << i;
      else {
            k = P[i] [j];
            cout << "(";
            order (i, k);
            order (k + 1, j);
            cout << ")";
            }
  }
```

Optimal Binary Search Trees

درخت جستجوی دودویی بهینه

- هر گره درخت دودویی حداکثر دو فرزند دارد.
- هر گره یک مقداردارد
- فرزندان راست از مقدار گره بزرگتر و فرزندان چپ از مقدار این گره کوچکتر هستند.

Examples

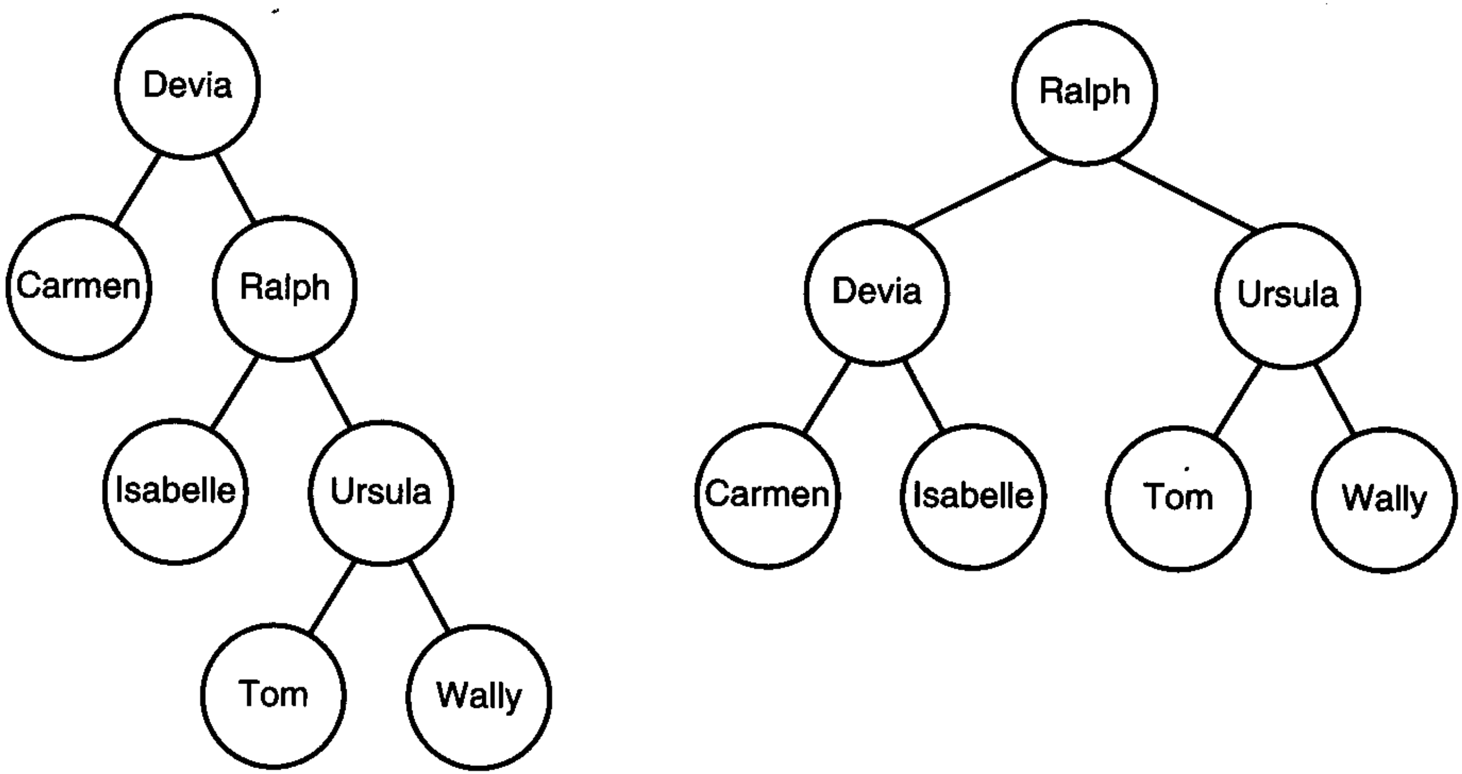


Figure 3.10 ● Two binary search trees.

Balanced tree
درخت متوازن

- عمق (سطح) یک گره: تعداد یال‌ها در مسیر از ریشه تا گره.
- عمق درخت: حداکثر عمق تمام گره‌های درخت.
- درخت دودویی متعادل: عمق دو زیردرخت هر گره هرگز بیش از ۱ نیست.
- درخت جستجوی دودویی بهینه: میانگین زمان لازم برای یافتن یک کلید به حداقل می‌رسد.

The search algorithm

- Algorithm 3.8: Search Binary Tree

  Problem: Determine the node containing a key in a binary search tree. It is assumed that the key is in the tree.

```
void search (node_pointer tree, keytype keyin,
   node_pointer& p)
   {
   bool found;
   p = tree; found = false;
   while (! found)
         if (p->key == keyin) found = true;
         else if (keyin < p-> key);
                  p = p-> left; // Advance to left child.
         else p = p-> right; // Advance to right child.
   }
```

The average search time
زمان جستجوی میانگین

- Search time: the number of comparisons done to locate a key
- Search time for *key* is: *depth*(*key*) + 1
- The average search time:

$$\sum_{i=1}^{n} c_i \, p_i$$

Where n is the number of keys, p_i the probability that Key_i is the search key, c_i the number of comparisons needed to find Key_i

Find out the average search time

- $P_1 = 0.7$, $P_2 = 0.2$, $P_3 = 0.1$

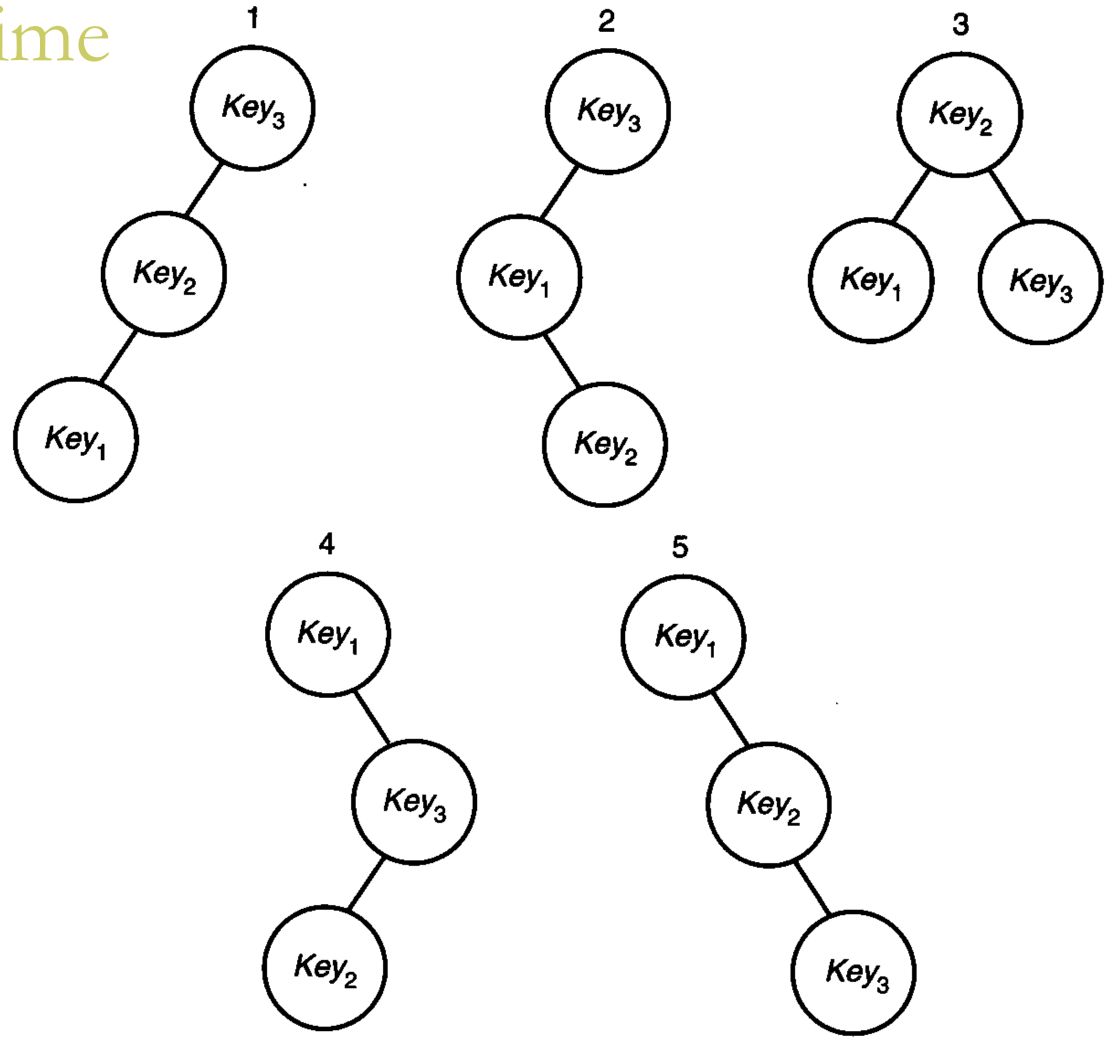


Figure 3.11 ● The possible binary search trees when there are three keys.

Brute force search complexity to find optimal solution

تعداد گره	تعداد درخت
۱	۱
۲	۲
۳	۵
۴	۱۴
۵	۴۲
۶	۱۳۲
. . .	
n	$\frac{1}{n+1}\binom{2n}{n}$

To develop an efficient algorithm

- Principle of optimality applies
- Let $A[i][j]$ = minimum value of $\sum_{m=i}^{j} c_m p_m$
- $A[i][i] = p_i$

To search a key in Tree *k*

- Let Tree *k* be the optimal tree that Key_k is at the root

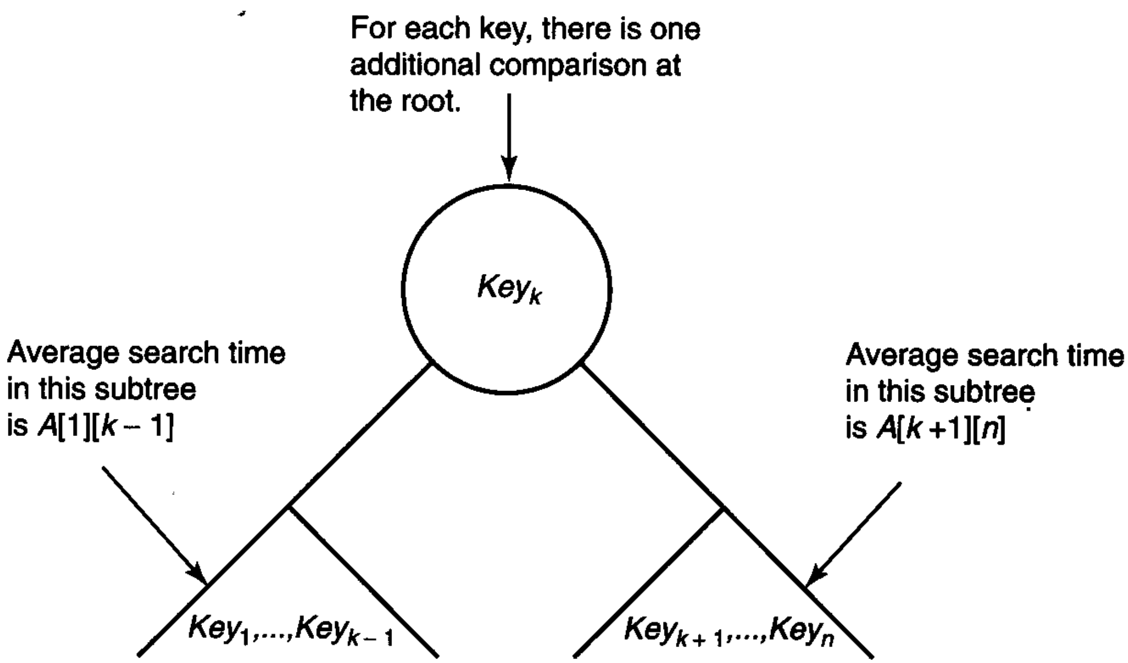


Figure 3.13 ● Optimal binary search tree given that Key_k is at the root.

The average search time

$$\underbrace{A[1][k-1]}_{\substack{\text{Average time in}\\ \text{left subtree}}} + \underbrace{p_1 + \cdots + p_{k-1}}_{\substack{\text{Additional time}\\ \text{comparing at root}}} + \underbrace{p_k}_{\substack{\text{Average time}\\ \text{searching for}\\ \text{root}}} + \underbrace{A[k+1][n]}_{\substack{\text{Average time in}\\ \text{right subtree}}} + \underbrace{p_{k+1} + \cdots + p_n}_{\substack{\text{Additional time}\\ \text{comparing at root}}},$$

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

Recursive

$$A[1][n] = \underset{1 \le k \le n}{Minimum}(A[1][k-1] + A[k+1][n]) + \sum_{m=1}^{n} p_m$$

$$A[i][j] = \underset{i \le k \le j}{minimum}(A[i][k-1] + A[k+1][j]) + \textstyle\sum_{m=i}^{j} p_m \qquad i < j$$

$$A[i][i] = p_i$$

$A[i][i-1]$ and $A[j+1][j]$ are defined to be 0.

The algorithm(Matrix initialization)

$$
\begin{aligned}
&for\ (i = 1;\ i <= n;\ i++)\\
&\{\\
&\qquad A[i][i-1] = 0;\\
&\qquad A[i][i] = p[i];\\
&\qquad R[i][i] = i;\\
&\qquad R[i][i-1] = 0;\\
&\}\\
&A[n+1][n] = 0;\\
&R[n+1][n] = 0;
\end{aligned}
$$

The algorithm (cont'd)

$$
\begin{aligned}
&\boldsymbol{for}\ (diagonal = 1;\ diagonal <= n - 1;\ diagonal++)\\
&\quad \boldsymbol{for}\ (i = 1;\ i <= n - diagonal;\ i++)\\
&\quad\{\\
&\qquad j = i + diagonal;\\
&\qquad A[i][j] = \underset{i \le k \le j}{minimum}\ (A[i][k-1] + A[k+1][j]) + \sum_{m=i}^{j} P_m\\
&\qquad R[i][j] = a\ value\ of\ k\ that\ gave\ the\ minimum;\\
&\quad\}\\
&\boldsymbol{minavg}\ A[1][n];
\end{aligned}
$$

Every-case time complexity

- Basic operation: The instructions executed for each value of *k*
- Input size: *n*, the number of keys
- Time complexity:

$$T(n) = \frac{n(n-1)(n+4)}{6} \in \Theta(n^3)$$

Build optimal binary search tree

```
node_pointer tree (index i, j)
{
    index k;
    node_pointer p;
    k = R[i][j];
    if (k == 0)
        return NULL;
    else{
        p = new nodetype;
        p->key = Key[k];
        p->left = tree(i, k - 1);
        p->right = tree (k + 1, j);
        return p;
    }
}
```

$$A[1][n] = \underset{1 \le k \le n}{Minimum}(A[1][k-1] + A[k+1][n]) + \sum_{m=1}^{n} p_m$$

Example (1)

- Keys: Don Isabelle Ralph Wally

  Key[1] Key[2] Key[3] Key[4]

  $p_1=3/8$ $p_2=3/8$ $p_3=1/8$ $p_4=1/8$
- Arrays produced

	0	1	2	3	4
1	0	$\frac{3}{8}$	$\frac{9}{8}$	$\frac{11}{8}$	$\frac{7}{4}$
2		0	$\frac{3}{8}$	$\frac{5}{8}$	1
3			0	$\frac{1}{8}$	$\frac{3}{8}$
4				0	$\frac{1}{8}$
5					0

A

	0	1	2	3	4
1	0	1	1	2	2
2		0	2	2	2
3			0	3	3
4				0	4
5					0

R

Figure 3.14 • The arrays *A* and *R*, produced when Algorithm 3.9 is applied to the instance in Example 3.9.

The resultant tree

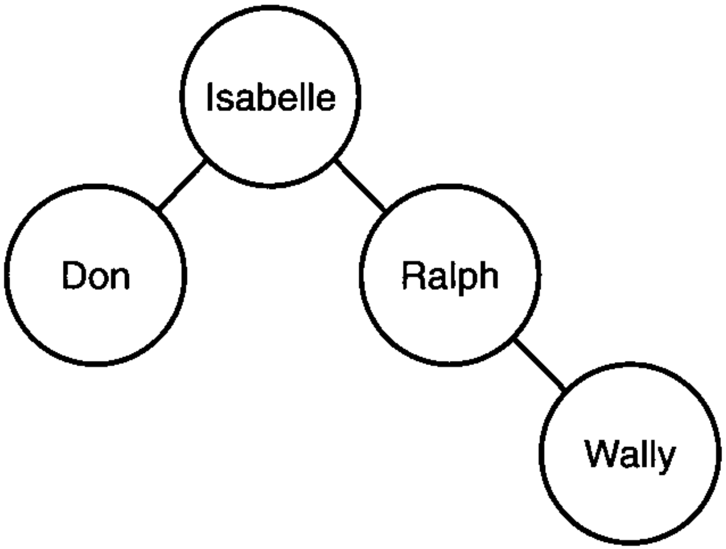


Figure 3.15 ● The tree produced when Algorithms 3.9 and 3.10 are applied to the instance in Example 3.9.

$$A[1][n] = \underset{1 \le k \le n}{Minimum}\left(A[1][k-1] + A[k+1][n]\right) + \sum_{m=1}^{n} p_m$$

Example (2)

- Keys: A B C D

  $p_1=1/8$ $p_2=4/8$ $p_3=1/8$ $p_4=2/8$

Initialization

$$A = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccccc} 0 & 1 & 2 & 3 & 4 \end{array} \\ \left|\begin{array}{ccccc} 0 & 1/8 & & & \\ & 0 & 4/8 & & \\ & & 0 & 1/8 & \\ & & & 0 & 2/8 \\ & & & & 0 \end{array}\right| \end{array}$$

$$A[1][n] = \underset{1 \le k \le n}{Minimum}(A[1][k-1] + A[k+1][n]) + \sum_{m=1}^{n} p_m$$

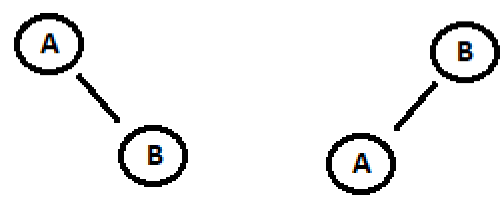
Example (2) …

- Keys: A B C D

p_1=1/8 p_2=4/8 p_3=1/8 p_4=2/8

$$A = \begin{array}{c|ccccc|} & 0 & 1 & 2 & 3 & 4 \\ 1 & 0 & 1/8 & ? & & \\ 2 & & 0 & 4/8 & & \\ 3 & & & 0 & 1/8 & \\ 4 & & & & 0 & 2/8 \\ 5 & & & & & 0 \end{array}$$

A[1][2] = min{A[1][0]+A[2][2],
A[1][1]+A[3][2]}
+1/8+4/8 = 6/8



A[1][0] A[2][2] A[1][1] A[3][2]

$$A[1][n] = \underset{1 \le k \le n}{Minimum}(A[1][k-1] + A[k+1][n]) + \sum_{m=1}^{n} p_m$$

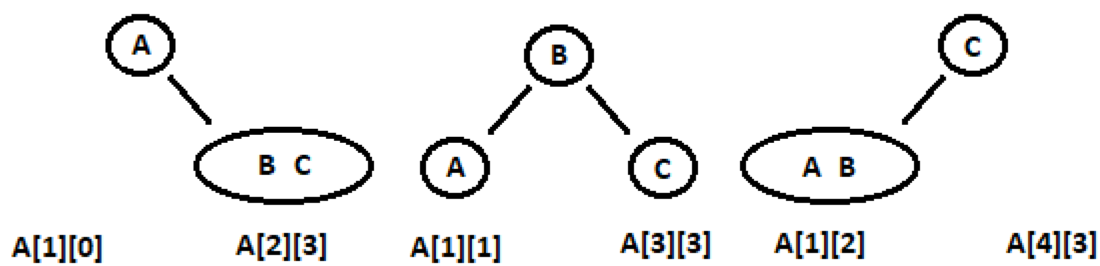
Example (2) …

- Keys: A B C D

  $p_1=1/8$ $p_2=4/8$ $p_3=1/8$ $p_4=2/8$

$$A = \begin{array}{c|ccccc|} & 0 & 1 & 2 & 3 & 4 \\ 1 & 0 & 1/8 & 6/8 & ? & \\ 2 & & 0 & 4/8 & 6/8 & \\ 3 & & & 0 & 1/8 & 4/8 \\ 4 & & & & 0 & 2/8 \\ 5 & & & & & 0 \end{array}$$

A[1][3] = min{A[1][0]+A[2][3],
A[1][1]+A[3][3],
A[1][2]+A[4][3]}
+1/8+4/8+1/8



The traveling sales person problem
مساله فروشنده دوره‌گرد

- Tour (Hamilton circuit): a path from a vertex to itself that passes through each of the other vertices exactly once
- Optimal tour: such a path of minimum length
- Brute force algorithm is
  $(n\text{-}1)(n\text{-}2)\cdots 1 = (n\text{-}1)!$
- Principle of optimality applies

Representation of the graph

	1	2	3	4
1	0	2	9	∞
2	1	0	6	4
3	∞	7	0	8
4	6	3	∞	0

Figure 3.17 ● The adjacency matrix representation W of the graph in Figure 3.16.

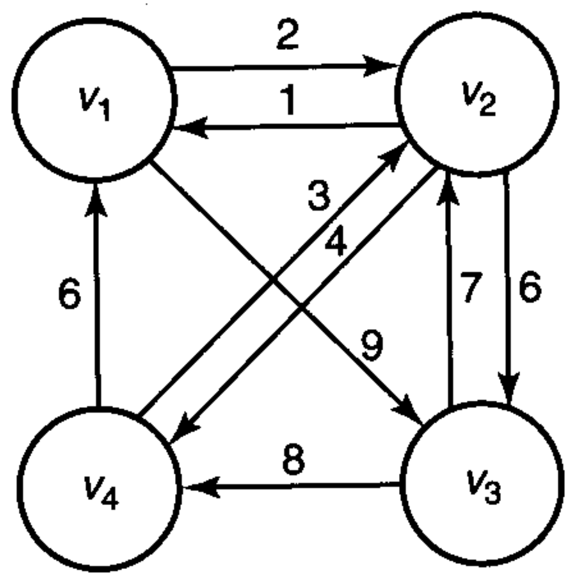
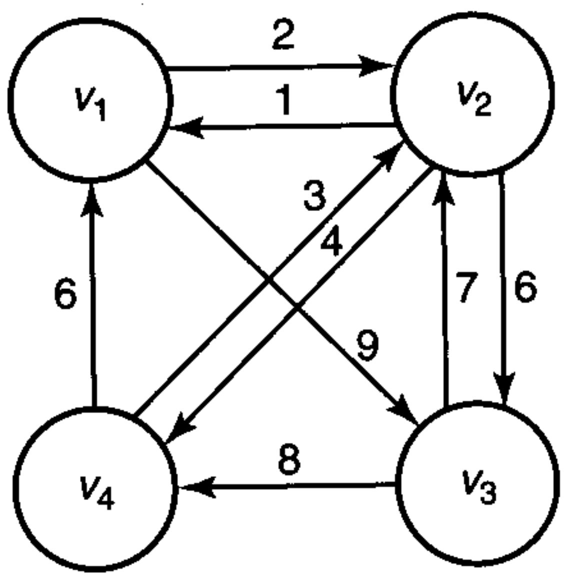


Figure 3.16 ● The optimal tour is $[v_1, v_3, v_4, v_2, v_1]$.

Preparation

- Let $D[v_i][A]$ = length of a shortest path from v_i to v_1 passing through each vertex in A exactly once
- Compute $D[v_2][A]$ when $A = \{v_3\}$ and $A = \{v_3, v_4\}$



The algorithm

- Length of an optimal tour =

$$\underset{2\le j\le n}{Minimum}(W[1][j]+D[v_j][V-\{v_1,v_j\}])$$

- In general for $i \neq 1$ and v_i not in A

$$D[v_i][A]=\underset{j:v_j\in A}{Minimum}(W[i][j]+D[v_j][A-\{v_j\}]) \qquad if \quad A\neq\phi$$

$$D[v_i][\phi]=W[i][1]$$

Compute the optimal tour

	1	2	3	4
1	0	2	9	∞
2	1	0	6	4
3	∞	7	0	8
4	6	3	∞	0

Figure 3.17 ● The adjacency matrix representation W of the graph in Figure 3.16.

The Algorithm

```
void travel (int n, const number W[][], index P[][], number& minlength)
{
  index i, j, k;
  number D[1..n][subset of V - {v1}];
  for (i = 2; i <= n; i++) D[i][∅] = W[i][1];


  for (k = 1; k <= n - 2; k++)
    for (all subsets A ∈ V - {v1} containing k vertices)
          for (i such that i ≠ 1 and vi is not in A){
                    D[i][A] = minimum (W[i][j] + D[j][A - {vj}]);
                              j: vj ∈ A
                    P[i][A] = value of j that gave the minimum;
          }
   D[1][V - {v1}] = minimum (W[1][j] + D[j][V - {v1, vj}]);
                    2 ≤ j ≤ n
   P[1][V - {v1}] = value of j that gave the minimum;
    minlength = D[1][V - {v1}];
}
```

Time complexity

$$\sum_{k=1}^{n} k \binom{n}{k} = n\,2^{n-1}$$

$$T(n) = \sum_{k=1}^{n-2} (n-1-k)\, k \binom{n-1}{k}.$$

$$(n-1-k)\binom{n-1}{k} = (n-1)\binom{n-2}{k}.$$

$$T(n) = (n-1) \sum_{k=1}^{n-2} k \binom{n-2}{k}.$$

Every-case time complexity

- Basic operation: the instructions executed for each value of v_j
- n, the number of vertices in the graph
- Time complexity

  $T(n) = (n\text{-}1)(\text{n-}2)2^{n\text{-}3} \in \Theta(n^2 2^n)$
- Memory complexity:

  $M(n) = 2 \times n2^{n\text{-}1} = n2^n \in \Theta(n2^n)$

Catalan Number

عدد کاتالان

مقدمه

- **کاتالان** در ریاضیات ترکیبی، نام یک سری از اعداد طبیعی است که در مسائل شمارشی کاربرد دارد.
- این سری به افتخار ریاضیدان بلژیکی **شارل کاتالان** (قرن نوزدهم) نام نهاده شد.

N	Catalan	Fibonacci
0	1	0
1	1	1
2	2	1
3	5	2
4	14	3
5	42	5
6	132	8
7	429	13
8	1430	21
9	4862	34
…	…	…
20	6564120420	6765

رابطه کاتالان

$$C_n = \frac{1}{n+1}\binom{2n}{n} = \frac{(2n)!}{(n+1)!\,n!} \qquad \text{for } n \geq 0.$$

$$C_0 = 1 \quad \text{and} \quad C_{n+1} = \sum_{i=0}^{n} C_i\,C_{n-i} \quad \text{for } n \geq 0.$$

$$C_0 = 1 \quad \text{and} \quad C_{n+1} = \frac{2(2n+1)}{n+2} C_n\,,$$

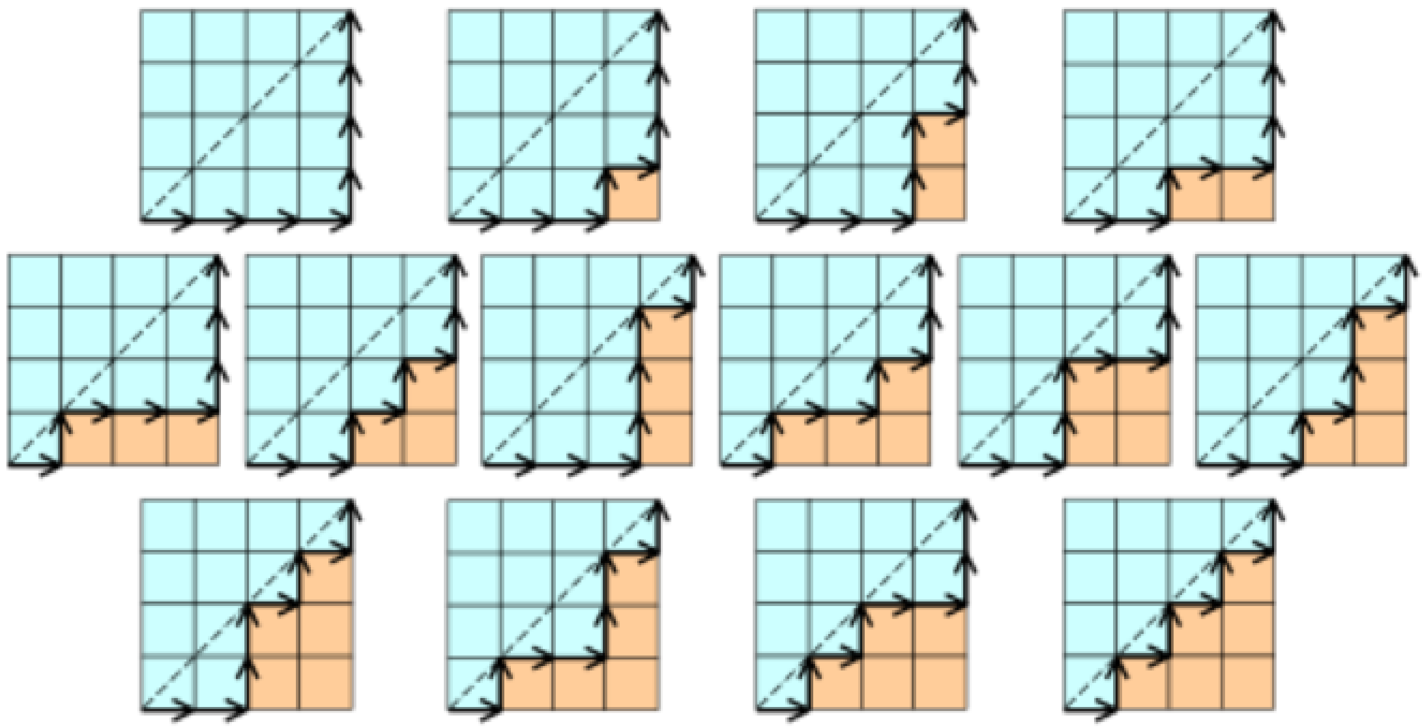
مسایل کاتالانی – پرانتزگذاری

- تعداد راه‌های مختلفی است که چند عامل می‌توانند پرانتزگذاری شوند.
  - مساله ضرب ماتریس‌ها
- $n =$ تعداد ضرب

$$((ab)c)d \quad (a(bc))d \quad (ab)(cd) \quad a((bc)d) \quad a(b(cd))$$

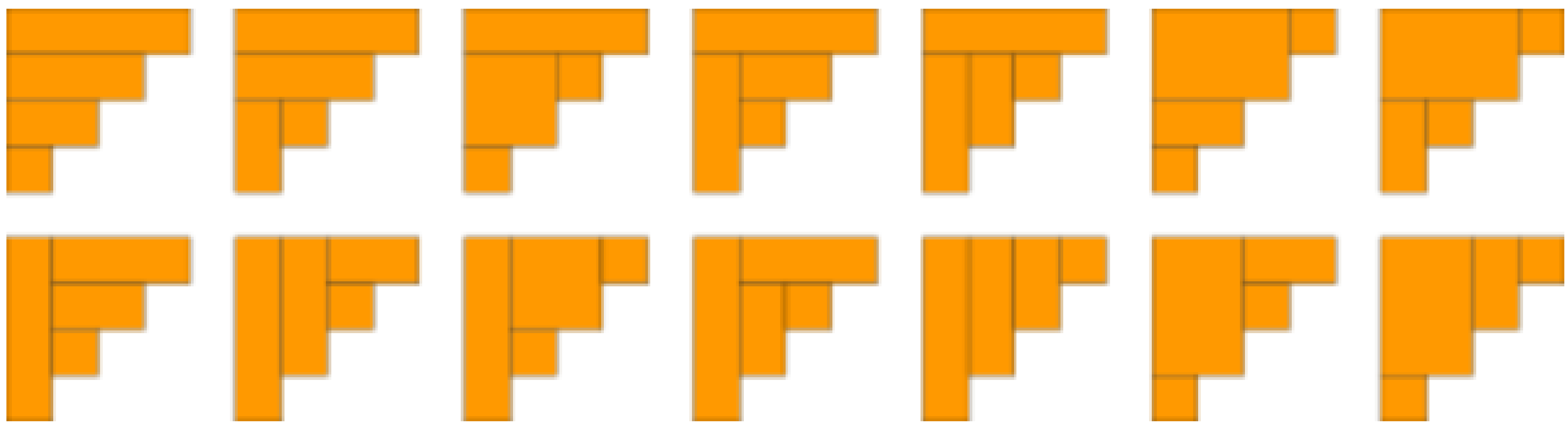
مسایل کاتالانی
مسیرهای پایین قطر اصلی غیر نزولی

- تمام مسیرهای غیر نزولی پایین قطر فرعی برای رسیدن از مبدا به مقصد



مسایل کاتالانی
تمام حالت‌های ممکن کاشی‌کاری راه پله $n \times n$ با n کاشی

- $N = 4$



مسایل کاتالانی-مثلث‌بندی چندضلعی محدب