Dynamic programming

3.1 The Binomial Coefficient

✓ Divide-and-conquer approach

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

$$C_k^n = C_{k-1}^{n-1} + C_k^{n-1} \quad 0 < k < n$$
$$= 1 \quad k = 0 \text{ or } k = n$$

✓ Algorithm 3.1

This algorithm computes $2C_k^n - 1$ terms to determine C_k^n

✓ Proof by induction

$$T_k^{n+1} = 1 + T_k^n + T_{k-1}^n = 1 + 2C_k^n - 1 + 2C_{k-1}^n - 1$$

$$= 2C_k^n + 2C_{k-1}^n - 1$$

$$= 2\frac{n!}{k!(n-k)!} + 2\frac{n!}{(k-1)!(n-k+1)!} - 1$$

$$= 2\frac{n!(n-k+1)}{k!(n-k+1)!} + 2\frac{n!k}{k!(n-k+1)!} - 1$$

$$= 2\frac{(n+1)n!}{k!(n-k+1)!} - 1 = 2\frac{(n+1)!}{k!(n-k+1)!} - 1$$

$$= 2C_k^{n+1} - 1$$

✓ Dynamic Programming approach

Figure 3.1

Algorithm 3.2

Time complexity of algorithm 3.2 is

$1+2+3+\ldots+k+(k+1)+(k+1)\ldots(k+1)$

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

All-Pairs shortest path

✓ A sample Graph—Figure 3.2

The representation matrix W—Fig 3.3

$D^{(k)}[i,j]$ be the 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 nodes

$$D^{(k)}[i,j]=\min\{\, D^{(k-1)}[i,j],\ D^{(k-1)}[i,k]+D^{(k-1)}[k,j]\}$$

$D^0 \Rightarrow D^1 \Rightarrow D^2 \Rightarrow \ldots\ldots \Rightarrow D^{n-1} \Rightarrow D^n$

✓ Algorithm 3.3

D0	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

D1	1	2	3	4	5
1	0	1	∞	1	5
2	9	0	3	2	14
3	∞	∞	0	4	∞
4	∞	∞	2	0	3
5	3	4	∞	4	0

D2	1	2	3	4	5
1	0	1	4	1	5
2	9	0	3	2	14
3	∞	∞	0	4	∞
4	∞	∞	2	0	3
5	3	4	7	4	0

D3	1	2	3	4	5
1	0	1	4	1	5
2	9	0	3	2	14
3	∞	∞	0	4	∞
4	∞	∞	2	0	3
5	3	4	7	4	0

D4	1	2	3	4	5
1	0	1	3	1	4
2	9	0	3	2	5
3	∞	∞	0	4	7
4	∞	∞	2	0	3
5	3	4	6	4	0

D5	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

✓ Finding path routes

P[i,j]: highest index of an intermediate node on the shortest path from v_i to v_j

Algorithm 3.4 reserves path information

Algorithm 3.5 prints path routes

P0	1	2	3	4	5
1	0	0	0	0	0
2	0	0	0	0	0
3	0	0	0	0	0
4	0	0	0	0	0
5	0	0	0	0	0

P1	1	2	3	4	5
1	0	0	0	0	0
2	0	0	0	0	0
3	0	0	0	0	0
4	0	0	0	0	0
5	0	1	0	1	0

P2	1	2	3	4	5
1	0	0	2	0	0
2	0	0	0	0	0
3	0	0	0	0	0
4	0	0	0	0	0
5	0	1	2	1	0

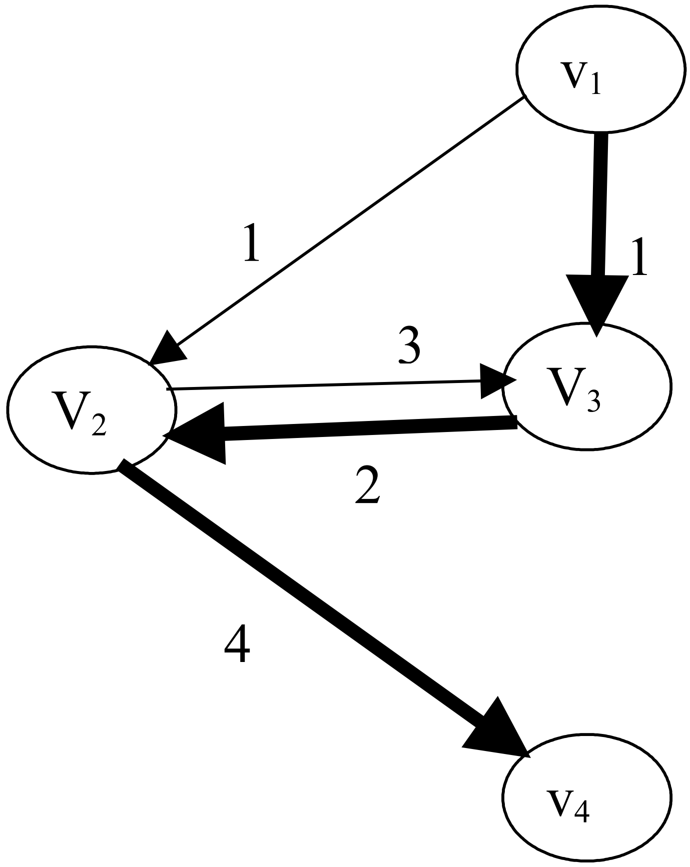
P3	1	2	3	4	5
1	0	0	2	0	0
2	0	0	0	0	0
3	0	0	0	0	0
4	0	0	0	0	0
5	0	1	2	1	0

P4	1	2	3	4	5
1	0	0	4	0	4
2	0	0	0	0	4
3	0	0	0	0	4
4	0	0	0	0	0
5	0	1	4	1	0

P5	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

✓ Principle of optimality: a problem possess this property if an optimum solution to an instance always contains optimum solutions to all subinstances

The shortest path problem does but the longest path problem does not



3.4 Chained matrix multiplication

- ✓ $A_{ij} \times B_{jk}$ needs ijk multiplications

  $A_{20\times 2} \times (B_{2\times 30} \times (C_{30\times 12} \times D_{12\times 8}))$

  $(A_{20\times 2} \times B_{2\times 30}) \times (C_{30\times 12} \times D_{12\times 8})$

  $A_{20\times 2} \times ((B_{2\times 30} \times C_{30\times 12}) \times D_{12\times 8})$

  $((A_{20\times 2} \times B_{2\times 30}) \times C_{30\times 12}) \times D_{12\times 8}$

  $(A_{20\times 2} \times (B_{2\times 30} \times C_{30\times 12}) \times D_{12\times 8})$

- ✓ Brute-force method

  Let t_n be the number of different orders where we can multiply n matrices.

  $A_1A_2 \ldots A_{n-1}A_n$ can be multiply by either

  $A_1(A_2 \ldots A_{n-1}A_n)$ or $(A_1A_2 \ldots A_{n-1})A_n$

  $t_n \geq t_{n-1} + t_{n-1} = 2t_{n-1}$ and $t_2 = 1$

  $\Rightarrow t_n \geq 2^{n-2}$

- ✓ Let A_i be a matrix of $d_{i-1} \times d_i$

  A_2A_3 needs $d_1 \times d_2 \times d_3$ 乘法

  $A_1(A_2A_3)$ needs $d_0 \times d_1 \times d_3$ 乘法

  $(A_1A_2A_3)A_4$ needs $d_0 \times d_3 \times d_4$ 乘法

  $A_2(A_3A_4A_5)$ needs $d_1 \times d_2 \times d_5$ 乘法

- ✓ Dynamic Programming approach

  Let M[i,j] be the minimum number of × needed to multiply A_i through A_j

  $$M[i,j] = \min_{i \le k \le j-1} \{M[i,k] + M[k+1,j] + d_{i-1} d_k d_j\}$$
  $$\mathrm{M[i,i]} = 0$$

  Examples 3.5, 3.6 and Algorithm 3.6

- ✓ Every case time complexity

  $$\sum_{diagonal=1}^{n-1} (n - diagonal) \times diagonal$$
  $$= n \sum_{diagonal=1}^{n-1} diagonal - \sum_{diagonal=1}^{n-1} diagonal^2$$
  $$= \frac{n^2(n-1)}{2} - \frac{(n-1)(n)(2n-1)}{6} = \frac{n^3 - n}{6} = \frac{n(n-1)(n+1)}{6}$$
  $$\in \Theta(n^3)$$

- ✓ Algorithm 3.7 prints the sequence of matrix multiplication

3.5 Optimal binary search tree

✓ Every key has a distinct access frequency.
These frequencies are known.
How can we build the most efficient binary search tree?
Example 3.7

✓ Let c_m be the level of key_m in the tree.
Assume that key_k is the root of the tree.
Let c'_m be the level of key_m in the subtree after removing the root key_k.
Let A[i,j] be the average number of comparisons needed for the optimal binary search tree constructed for key_i through key_j

$$A[1,n] = \sum_{i=1}^{n} p_i c_i = p_k c_k + \sum_{i=1}^{k-1} p_i c_i + \sum_{i=k+1}^{n} p_i c_i$$

$$= p_k + \sum_{i=1}^{k-1} p_i (c'_i + 1) + \sum_{i=k+1}^{n} p_i (c'_i + 1)$$

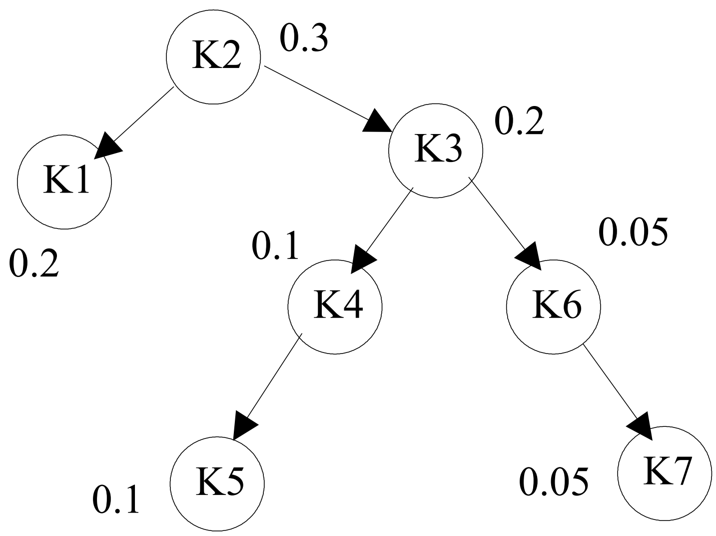
$$= p_k + \sum_{i=1}^{k-1} p_i c'_i + \sum_{i=k+1}^{n} p_i c'_i + \sum_{i=1}^{k-1} p_i + \sum_{i=k+1}^{n} p_i$$

$$= \sum_{i=1}^{n} p_i + \sum_{i=1}^{k-1} p_i c'_i + \sum_{i=k+1}^{n} p_i c'_i$$

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

Similarly, we can get the following

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



A[1,1]=0.2, A[5,5]=0.1, A[7,7]=0.05

A[4,5]=0.2+A[5,5]=0.3

=(1×0.1)+(2×0.05)=0.3

A[3,7]=0.5+A[4,5]+A[6,7]=0.5+0.3+0.15

=0.95=(1×0.2)+(2×0.1)+(2×0.05)+(3×0.1)+(3×0.0)

A[1,7]=1.0+A[1,1]+A[3,7]=1.0+0.2+0.95=2.15

=(1*0.3)+(2*0.2)+(2*0.2)+(3*0.15)+(4*0.15)

- ✓ The recursive relation

$$A[1,n] = \min_{1 \le k \le j} \sum_{i=1}^{n} p_i + A[1,k-1] + A[k+1,n]$$

$$A[i,j] = \min_{i \le k \le j} \sum_{m=i}^{j} p_m + A[i,k-1] + A[k+1,j]$$

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

$$A[i,i-1] = 0 \text{ and } A[j+1,j] = 0$$

- ✓ Algorithm 3.9 finds the matrix A

  Algorithm 3.10 builds the tree

  Example 3.9

- ✓ Please compute the following data

Key1	0.2
Key2	0.1
Key3	0.05
Key4	0.15
Key5	0.15
Key6	0.2
Key7	0.05
Key8	0.1

Diagonal=1 Diagonal=2 Diagonal=3

j / i	0	1	2	3	4	5	6	7	8
1	0	0.2	0.4	0.55	0.95				
2		0	0.1	0.2	0.5				
3			0	0.05	0.25	0.55			
4				0	0.15	0.45	0.85		
5					0	0.15	0.5	0.6	
6						0	0.2	0.30	0.55
7							0	0.05	0.2
8								0	0.1
9									0

Diagonal=1 Diagonal=2 Diagonal=3

j / i	0	1	2	3	4	5	6	7	8
1	0	1	1	1	2				
2		0	2	2	4				
3			0	3	4	4			
4				0	4	4,5	5		
5					0	5	6	6	
6						0	6	6	6
7							0	7	8
8								0	8

9									0

3.6 The traveling salesman problem

- ✓ A tour in a directed graph is a path from a node to itself that passes through each of the other nodes exactly once.

  An optimum tour is such a path of minimum length.

  Figure 3.16 is the directed graph.

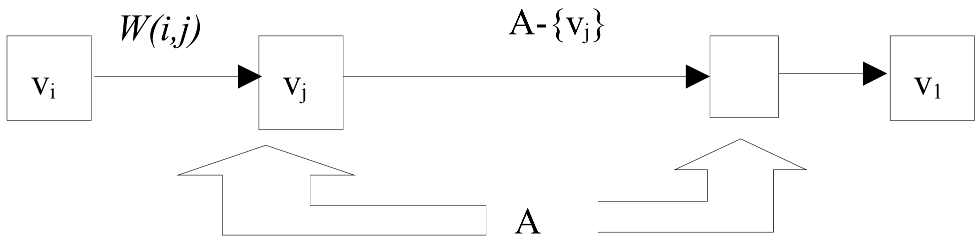
  Figure 3.17 is the adjacency matrix.

- ✓ Let D[v_i,A] is the length of a shortest path from v_i to v_1 passing each node in A exactly once.

$$\text{optimal length} = \min_{2\le j\le n}(W[1,j]+D[v_j, V-\{v_1,v_j\}])$$

$$D[v_i,A]=\min_{j\in A}(W[i,j]+D[v_j,A-\{v_j\}]) \text{ if } A\neq\phi$$

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



- ✓ Examples 3.10, 3.11

  Algorithm 3.11
- ✓ Theorem 3.1: $\sum_{k=1}^{n} kC_k^n = n2^{n-1}$

$$kC_k^n = k\frac{n!}{(n\text{-}k)!k!} = \frac{n!}{(n\text{-}k)!(k-1)!} = \frac{(n-1)!\,n}{(n\text{-}k)!(k-1)!} = nC_{k-1}^{n-1}$$

$$\sum_{k=1}^{n} kC_k^n = n\sum_{k=1}^{n} C_{k-1}^{n-1} = n\sum_{k=0}^{n-1} C_k^{n-1} = n2^{n-1}$$

- ✓ Time complexity analysis

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

$$n \quad (n-1-k)C_k^{n-1} = (n-1)C_k^{n-2}$$

$$\therefore T(n) = (n-1)\sum_{k=1}^{n-2} kC_k^{n-2} = (n-1)(n-2)2^{n-3} \in \Theta(n^2 2^n)$$

- ✓ The memory size D[v_i,A] and P[v_i,A]

$$\sum_{k=1}^{n-2}(n-1-k)C_k^{n-1} = \sum_{k=1}^{n-2}(n-1)C_k^{n-2} = (n-1)\sum_{k=1}^{n-2} C_k^{n-2}$$
$$= (n-1)\times 2^{n-2} \in \Theta(n2^n)$$

- ✓ Example 3.12