Dynamic Programming

Solve sub-problems just once and save answers in a table

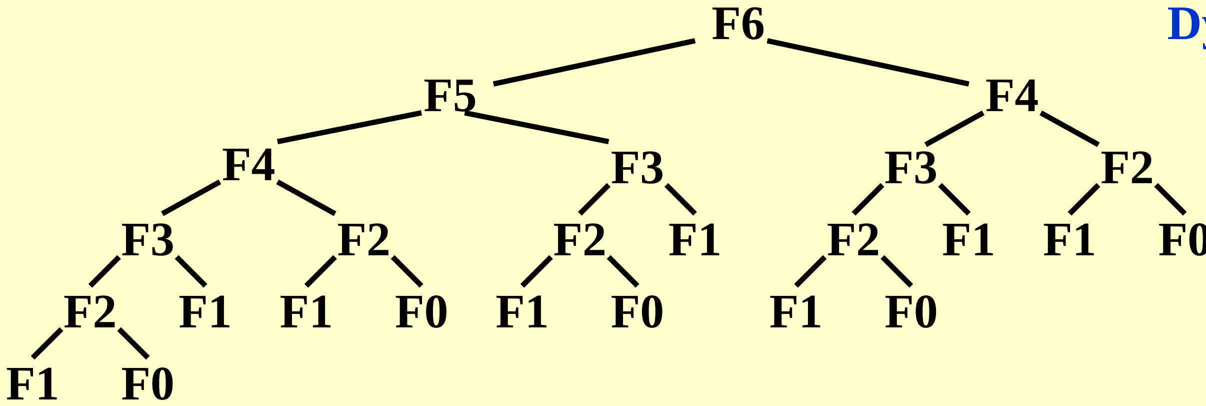
Use a table instead of recursion

1. **Fibonacci Numbers:** $F(N) = F(N-1) + F(N-2)$

```
int  Fib( int N )
{
   if ( N <= 1 )
      return  1;
   else
      return  Fib( N - 1 ) + Fib( N - 2 );
}
```

$$T(N) \geq T(N-1) + T(N-2)$$

$$\Rightarrow \quad T(N) \geq F(N)$$



Trouble-maker: The growth of redundant calculations is explosive.

Solution: Record the two most recently computed values to avoid recursive calls.

```
int  Fibonacci ( int N )
{   int  i, Last, NextToLast, Answer;
    if ( N <= 1 )  return  1;
    Last = NextToLast = 1;    /* F(0) = F(1) = 1 */
    for ( i = 2; i <= N; i++ ) {
        Answer = Last + NextToLast;   /* F(i) = F(i-1) + F(i-2) */
        NextToLast = Last; Last = Answer;  /* update F(i-1) and F(i-2) */
    }  /* end-for */
    return  Answer;
}
```

$$T(N) = O(N)$$

2. Ordering Matrix Multiplications

〖Example〗 Suppose we are to multiply 4 matrices

$$M_{1\,[\,10\times20\,]} * M_{2\,[\,20\times50\,]} * M_{3\,[\,50\times1\,]} * M_{4\,[\,1\times100\,]}\,.$$

If we multiply in the order

$$M_{1\,[\,10\times20\,]} * (M_{2\,[\,20\times50\,]} * (M_{3\,[\,50\times1\,]} * M_{4\,[\,1\times100\,]}))$$

Then the computing time is

$$50\times1\times100 + 20\times50\times100 + 10\times20\times100 = 125{,}000$$

If we multiply in the order

$$(M_{1\,[\,10\times20\,]} * (M_{2\,[\,20\times50\,]} * M_{3\,[\,50\times1\,]})) * M_{4\,[\,1\times100\,]}$$

Then the computing time is

$$20\times50\times1 + 10\times20\times1 + 10\times1\times100 = 2{,}200$$

Problem: In which order can we compute the product of n matrices with minimal computing time?

Let b_n = number of different ways to compute $M_1 \cdot M_2 \cdot \cdots \cdot M_n$. Then we have $b_2 = 1, b_3 = 2, b_4 = 5, \cdots$

Let $M_{ij} = M_i \cdot \cdots \cdot M_j$. Then $M_{1n} = M_1 \cdot \cdots \cdot M_n = M_{1i} \cdot M_{i+1\,n}$

$$\Rightarrow \quad b_n = \sum_{i=1}^{n-1} b_i b_{n-i}$$ wh

b

$$b_n = O\left(\frac{4^n}{n\sqrt{n}}\right)$$

If $j - i = k$, then the only values M_{xy} required to compute M_{ij} satisfy $y - x < k$.

Suppose we are to multiply n ma s $M_1 * \cdots \cdots * M_n$ where M_i is an $r_{i-1} \times r_i$ matrix. Let be the cost of the optimal way to compute $M_i * \cdots \cdots * M_j$. Th we have the recurrence equations:

$$m_{ij} = \begin{cases} 0 & \text{if } i = j \\ \min\limits_{i \le l < j} \{ m_{il} + m_{l+1\,j} + r_{i-1} r_l r_j \} & \text{if } j > i \end{cases}$$

```
/* r contains number of columns for each of the N matrices */
/* r[ 0 ] is the number of rows in matrix 1 */
/* Minimum number of multiplications is left in M[ 1 ][ N ] */
void OptMatrix( const long r[ ], int N, TwoDimArray M )
{   int  i, j, k, L;
    long  ThisM;
    for( i = 1; i <= N; i++ )   M[ i ][ i ] = 0;
    for( k = 1; k < N; k++ ) /* k = j - i */
       for( i = 1; i <= N - k; i++ ) { /* For each position */
            j = i + k;    M[ i ][ j ] = Infinity;
            for( L = i; L < j; L++ ) {
               ThisM = M[ i ][ L ] + M[ L + 1 ][ j ]
                          + r[ i - 1 ] * r[ L ] * r[ j ];
               if ( ThisM < M[ i ][ j ] )  /* Update min */
                     M[ i ][ j ] = ThisM;
            }  /* end for-L */
       }  /* end for-Left */
}
```

$$
\begin{array}{ccccc}
m_{1,N} & & & & \\
m_{1,N-1} & m_{2,N} & & & \\
\vdots & \vdots & \ddots & & \\
m_{1,2} & m_{2,3} & \cdots & m_{N-1,N} & \\
m_{1,1} & m_{2,2} & \cdots & m_{N-1,N-1} & m_{N,N}
\end{array}
$$

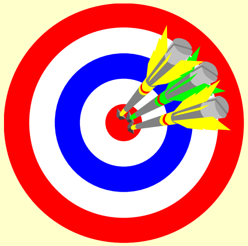
$$T(N) = O(N^3)$$

To record the ordering please refer to Figure 10.46 on p.388

$$m_{ij} = \begin{cases} 0 & \text{if } i = j \\ \min\limits_{i \le l < j} \{ m_{il} + m_{l+1\,j} + r_{i-1} r_l r_j \} & \text{if } j > i \end{cases}$$

3. Optimal Binary Search Tree

—— The best for static searching (without insertion and deletion)



Given N words $w_1 < w_2 < \ldots\ldots < w_N$, and the probability of searching for each w_i is p_i. Arrange these words in a binary search tree in a way that minimize the expected total access time. $T(N) = \sum_{i=1}^{N} p_i \cdot (1 + d_i)$

〖Example〗 Given the following table of probabilities:

word	break	case	char	do	return	switch	void
probability	0.22	0.18	0.20	0.05	0.25	0.02	0.08

Discussion 10:
Please draw the trees obtained by greedy methods and by AVL rotations. What are their expected total access times?

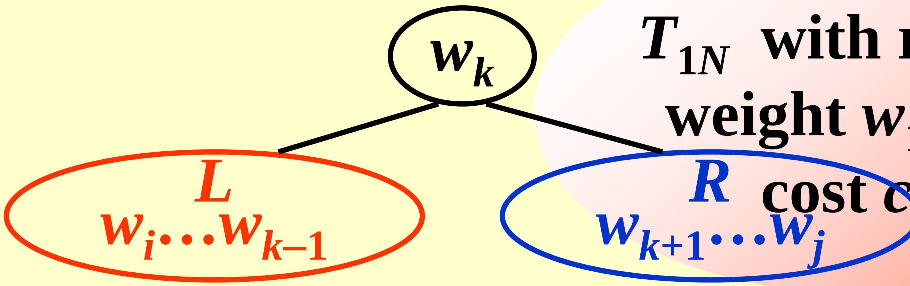
T_{ij} ::= OBST for w_i , ……, w_j ($i < j$)

c_{ij} ::= cost of T_{ij} ($c_{ii} = 0$)

r_{ij} ::= root of T_{ij}

w_{ij} ::= weight of $T_{ij} = \sum_{k=i}^{j} p_k$ ($w_{ii} = p_i$)

T_{ij}



T_{1N} with root r_{1N}, weight w_{1N}, and cost c_{1N} .

c_{ij} = p_k + cost(L) + cost(R) + weight(L) + weight(R)

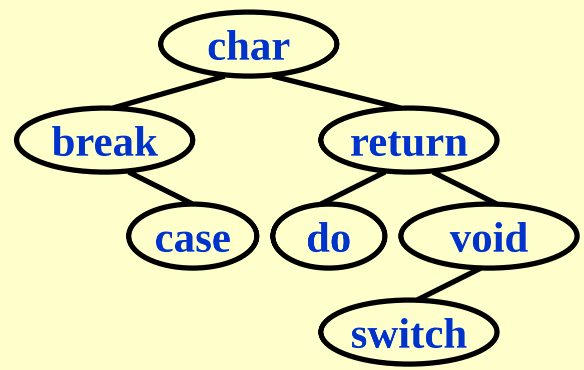
$= p_k + c_{i,\,k-1} + c_{k+1,\,j} + w_{i,\,k-1} + w_{k+1,\,j} = w_{ij} + c_{i,\,k-1} + c_{k+1,\,j}$

T_{ij} is optimal $\Rightarrow$ $r_{ij} = k$ is such that $c_{ij} = \min_{i < l \le j} \{ w_{ij} + \underline{c_{i,l-1} + c_{l+1,j}} \}$

$$c_{ij} = \min_{i<l\le j}\{w_{ij} + c_{i,l-1} + c_{l+1,j}\}$$

word	break	case	char	do	return	switch	void
probability	0.22	0.18	0.20	0.05	0.25	0.02	0.08

break.. break		case..case		char.. char		do..do		return..return		switch..switch		void.. void	
0.22	break	0.18	case	0.20	char	0.05	do	0.25	return	0.02	switch	0.08	void
break.. case		case.. char		char..do		do.. return		return..switch		switch.. void			
0.58	break	0.56	char	0.30	char	0.35	return	0.29	return	0.12	void		
break.. char		case..do		char.. return		do.. switch		return.. void					
1.02	case	0.66	char	0.80	return	0.39	return	0.47	return				
break..do		case.. return		char.. switch		do.. void							
1.17	case	1.21	char	0.84	return	0.57	return						
break.. return		case.. switch		char.. void									
1.83	char	1.27	char	1.02	return								
break.. switch		case.. void											
1.89	char	1.53	char										
break.. void													
2.15	char												



$$T(N) = O(N^3)$$

Please read 10.33 on p.419 for an O(N^2) algorithm.

4. All-Pairs Shortest Path

For all pairs of v_i and v_j ($i \neq j$), find the shortest path between.

Method 1 Use **single-source algorithm** for |V| times.

$T = O(\,|V|^3\,)$ – works fast on sparse graph.

Method 2 Define

$$D^k[\,i\,][\,j\,] = \min\{\text{ length of path } i \to \{\, l \le k \,\} \to j \,\}$$

and $D^{-1}[\,i\,][\,j\,] = \text{Cost}\,[\,i\,][\,j\,]$. Then the length of the shortest path from i to j is $D^{N-1}[\,i\,][\,j\,]$.

Algorithm

Start from D^{-1} and successively generate $D^0, D^1, \ldots, D^{N-1}$. If D^{k-1} is done, then either

① $k \notin$ the shortest path $i \to \{\, l \le k \,\} \to j \;\Rightarrow\; D^k = D^{k-1}$; or

② $k \in$ the shortest path $i \to \{\, l \le k \,\} \to j$

$= \{$ the S.P. from i to k $\} \cup \{$the S.P. from k to j $\}$

$\Rightarrow\; D^k[\,i\,][\,j\,] = D^{k-1}[\,i\,][\,k\,] + D^{k-1}[\,k\,][\,j\,]$

$$\therefore\quad D^k[i][j] = \min\{D^{k-1}[i][j],\, D^{k-1}[i][k] + D^{k-1}[k][j]\},\; k \ge 0$$

```
/* A[ ] contains the adjacency matrix with A[ i ][ i ] = 0 */
/* D[ ] contains the values of the shortest path */
/* N is the number of vertices */
/* A negative cycle exists iff D[ i ][ i ] < 0 */
void AllPairs( Tw DimArray A, TwoDimArray D, int N )
{   int  i, j, k;
    for ( i = 0; i < N; i+         itialize D */
         for( j = 0; j < N; j+
              D[
    for( k
         for( i
              for( j = 0; j < N; j
                 if( D[ i ][ k ] + D[ k ][ j ] < D[ i ][ j ] )
                      /* Update shortest path */
                      D[ i ][ j ] = D[ i ][ k ] + D[ k ][ j ];
}
```

Works if there are negative edge costs, but no negative-cost cycles.

$T(N) = O(N^3)$, **but faster in a *dense* graph.**

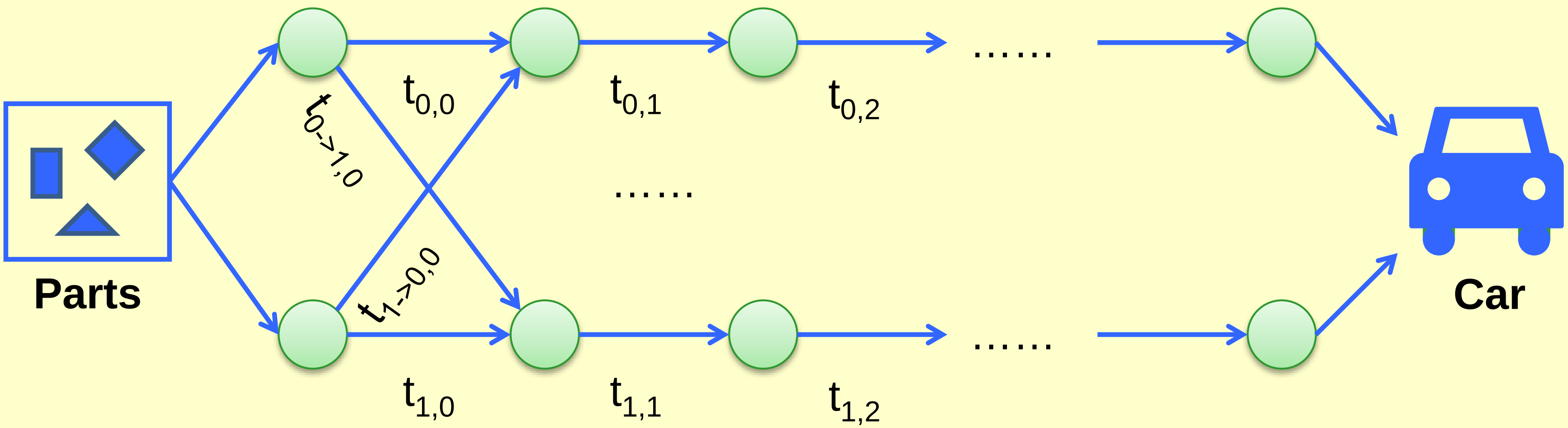
To record the paths please refer to Figure 10.53 on p.393

How to design a DP method?

- ☞ Characterize an optimal solution
- ☞ Recursively define the optimal values
- ☞ Compute the values in some order
- ☞ Reconstruct the solving strategy

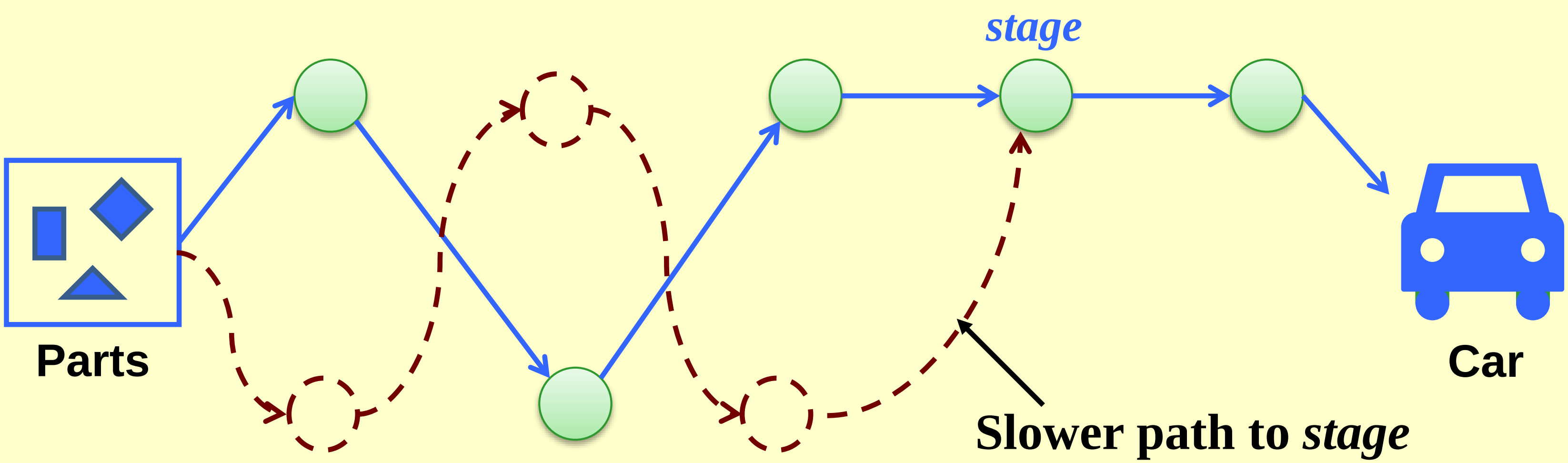
5. Product Assembly

- **Two assembly lines for the same car**
- **Different technology (time) for each stage**
- **One can change lines between stages**
- **Minimize the total assembly time**



Exhaustive search gives O(2^N) time + O(N) space

☞ Characterize an optimal solution

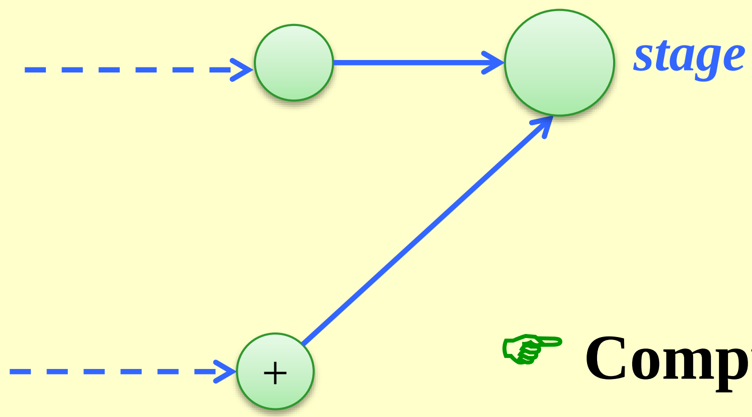


Optimal solution

☝ **An optimal solution contains an optimal solution of a sub-problem!**

☞ Recursively define the optimal values

An optimal path to *stage* is based on an optimal path to *stage*–1



☞ Compute the values in some order

O(*N*) time + O(*N*) space

```
f[0][0]=0;  f[1][0]=0;
for (stage=1; stage<=n; stage++){
  for (line=0; line<=1; line++){
     f[line][stage]
          f[  line][stage-1] + t_process[  line][stage-1]

  }
}
Solution = min(f[0][n],f[1][n]);
```

☞ Reconstruct the solving strategy

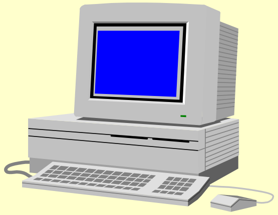
```
f[0][0]=0; L[0][0]=0;
f[1][0]=0; L[1][0]=0;
for(stage=1; stage<=n; stage++){
  for(line=0; line<=1; line++){
    f_stay = f[  line][stage-1] + t_process[  line][stage-1];
    f_move = f[1-line][stage-1] + t_transit[1-line][stage-1];
    if (f_stay<f_move){
      f[line][stage] = f_stay;

    }
    else {
      f[line][stage] = f_move;

    }
  }
}
```

```
line = f[0][n]<f[1][n]?0:1;
for(stage=n; stage>0; stage--){
  plan[stage] = line;
  line = L[line][stage];
}
```

Elements of DP:

- ☞ **Optimal substructure**
- ☞ **Overlapping sub-problems**

Discussion 11:
When *can't* we apply dynamic programming?



Research Project 4

The Best Peak Shape (26)

In many research areas, one important target of analyzing data is to find the best "peak shape" out of a huge amount of raw data full of noises. Now given *N* input numbers ordered by their indices, you may remove some of them to keep the rest of the numbers in a peak shape. You are supposed to find the best peak shape.

Detailed requirements can be downloaded from
https://pintia.cn/

Reference:

Introduction to Algorithms, 3rd Edition: Ch.15, p. 359-413； ***Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest and Clifford Stein. The MIT Press. 2009***