algorithm

5.动态规划a

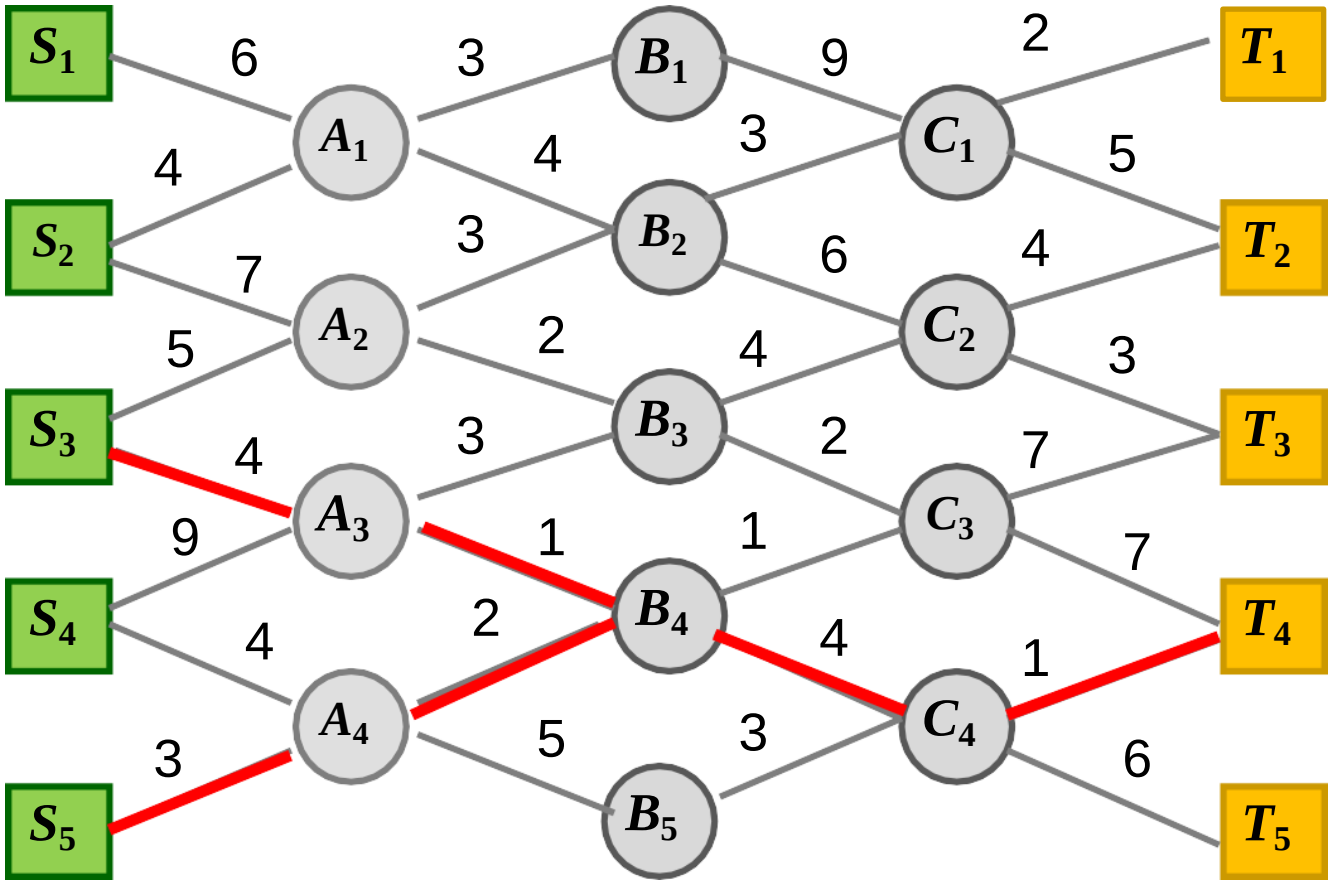
本节内容

- 4.1 最短路径问题
- 4.2 动态规划算法设计
- 4.3 动态规划算法的递归实现
- 4.4 动态规划算法的迭代实现

最短路径问题

- 问题：求最短路径
- 输入：
  - 起点集合 $\{ S_1, S_2, ..., S_n \}$ ，
  - 终点集合$\{T_1, T_2, ... , T_m\}$，
  - 中间结点集，
  - 边集 E，对于任意边 e 有长度
- 输出：
  - 一条从起点到终点的最短路

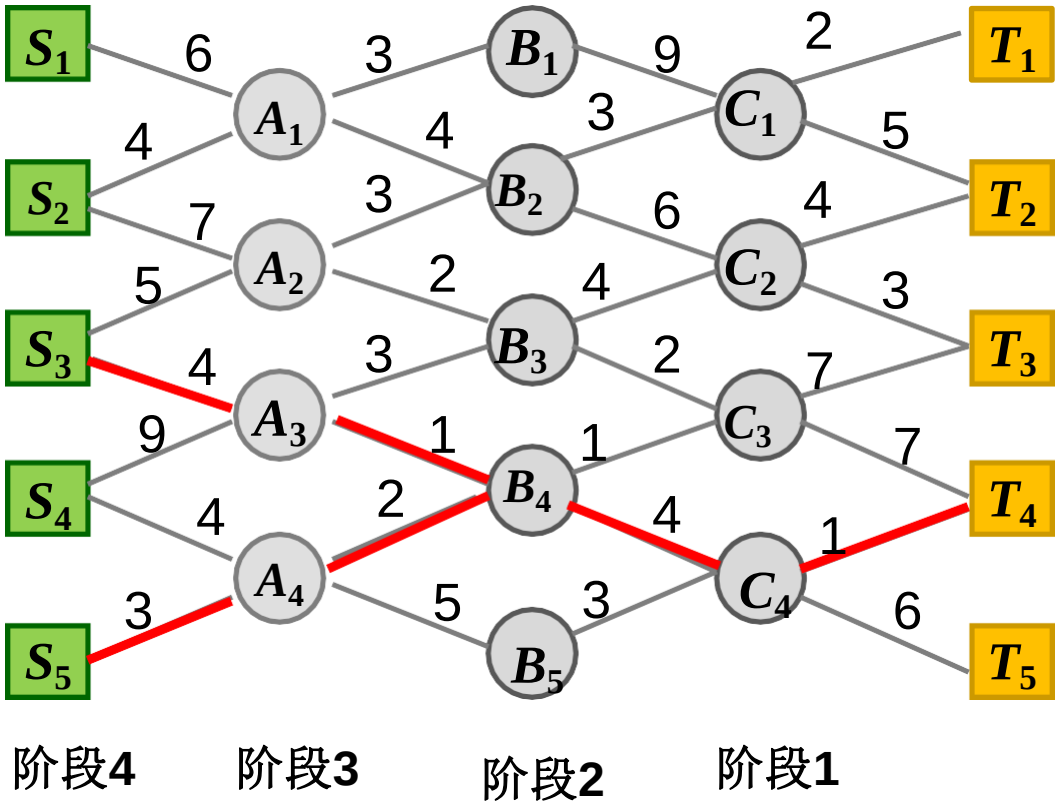
找出从S到T的最短路径



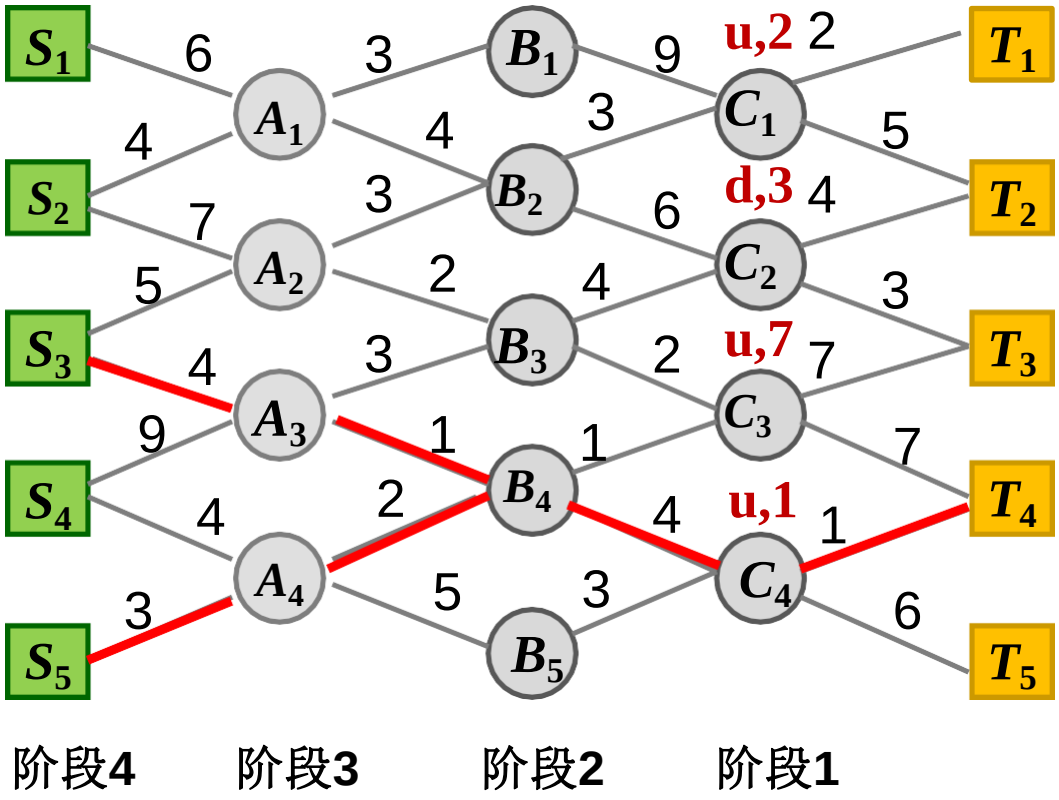
算法设计

- 蛮力算法：
  - 考察每一条从某个起点到某个终点的路径，计算长度，从其中找出最短路径
  - 在上述实例中，如果网络的层数为k，那么路径条数将接近于2^k
- 动态规划算法：
  - 多阶段决策过程
  - 每步求解的问题是后面阶段求解问题的子问题
  - 每步决策将依赖于以前步骤的决策结果

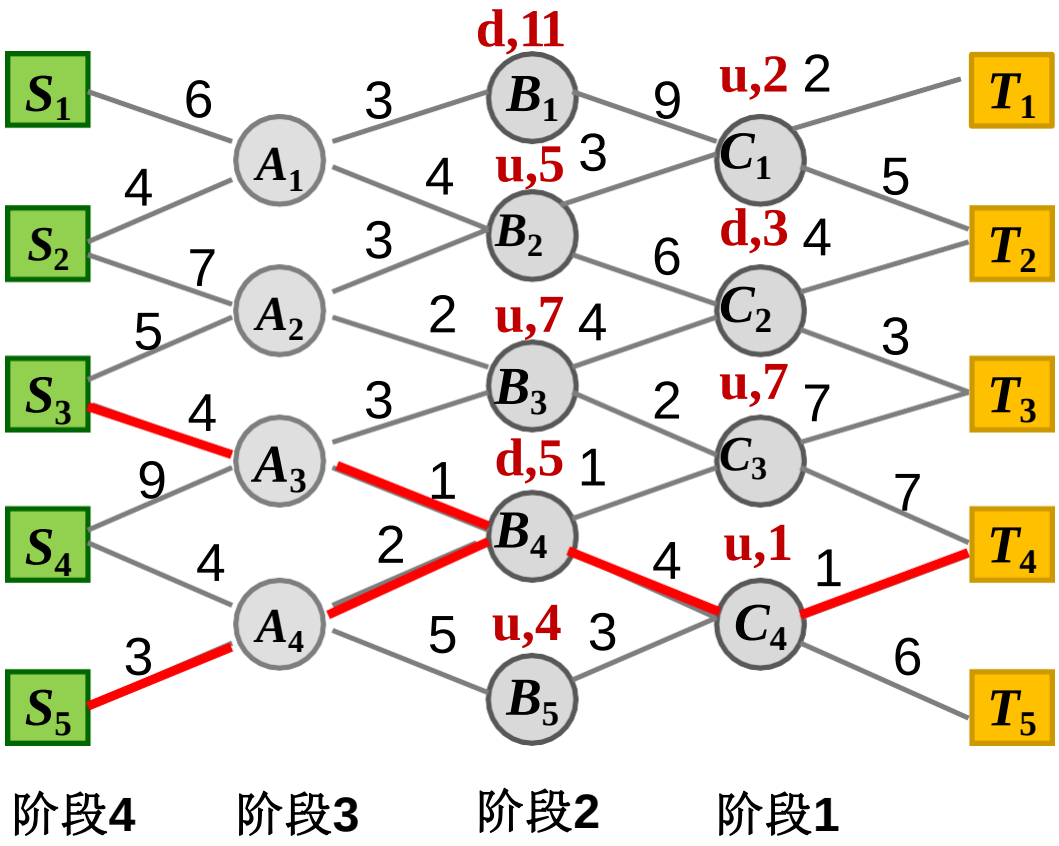
动态规划求解


S_1
S_2
S_3
S_4
S_5
A_1
A_2
A_3
A_4
B_1
B_2
B_3
B_4
B_5
C_1
C_2
C_3
C_4
T_1
T_2
T_3
T_4
T_5
6
3
9
2
4
4
3
5
7
3
6
4
5
2
4
3
4
3
2
7
9
1
1
7
4
2
4
1
3
5
3
6
阶段4
阶段3
阶段2
阶段1

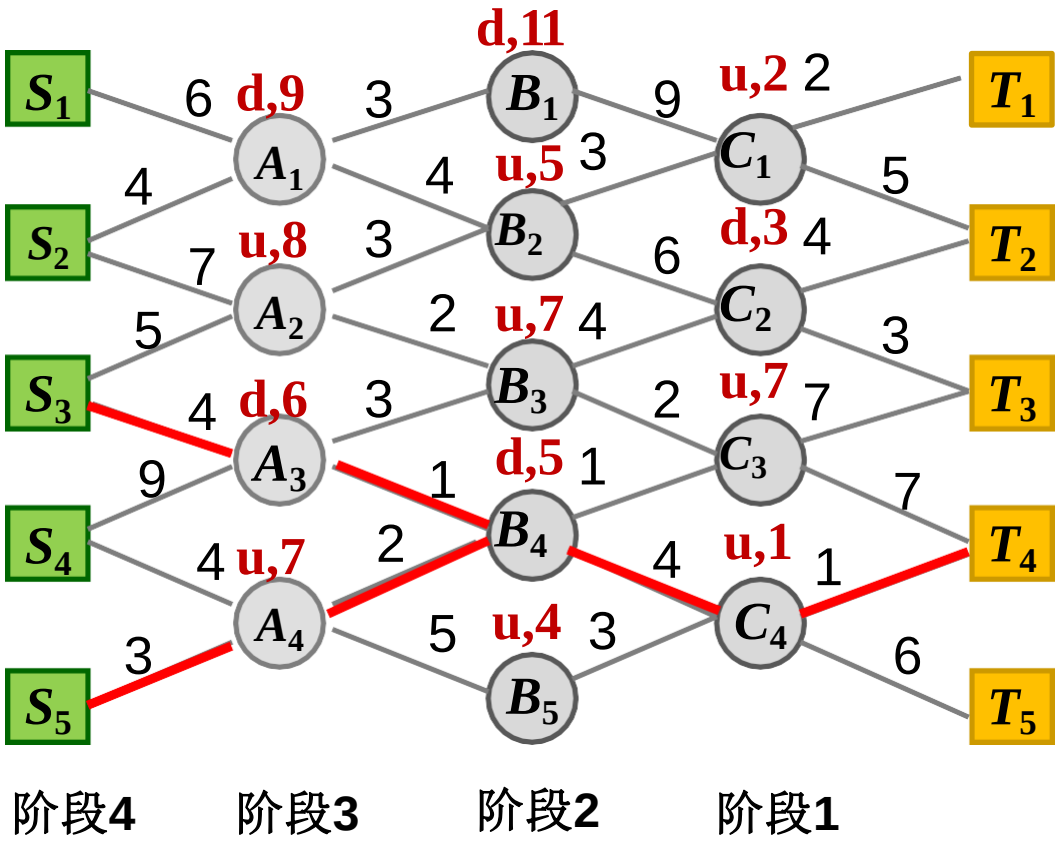
动态规划求解


S_1
S_2
S_3
S_4
S_5
A_1
A_2
A_3
A_4
B_1
B_2
B_3
B_4
B_5
C_1
C_2
C_3
C_4
T_1
T_2
T_3
T_4
T_5
u,2
d,3
u,7
u,1
阶段4
阶段3
阶段2
阶段1

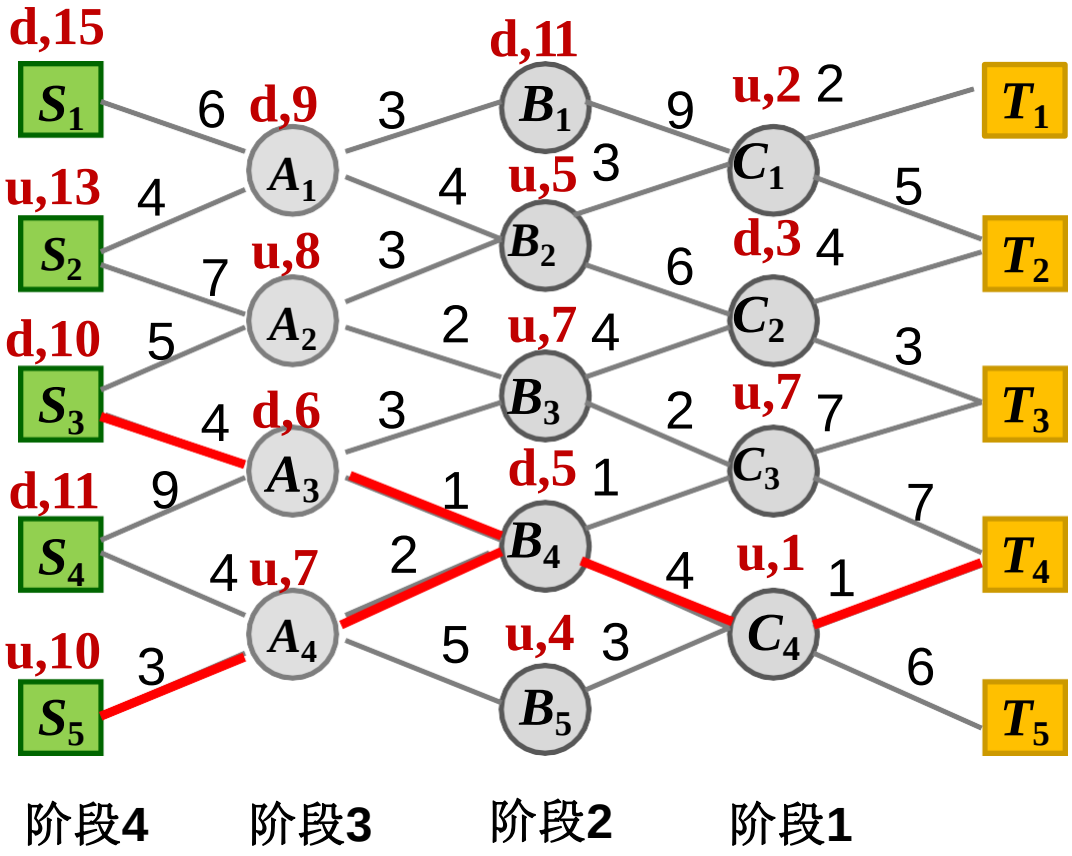
动态规划求解


d,11
u,2
d,3
u,7
u,5
d,5
u,1
u,4
S_1
S_2
S_3
S_4
S_5
A_1
A_2
A_3
A_4
B_1
B_2
B_3
B_4
B_5
C_1
C_2
C_3
C_4
T_1
T_2
T_3
T_4
T_5
阶段4
阶段3
阶段2
阶段1

动态规划求解

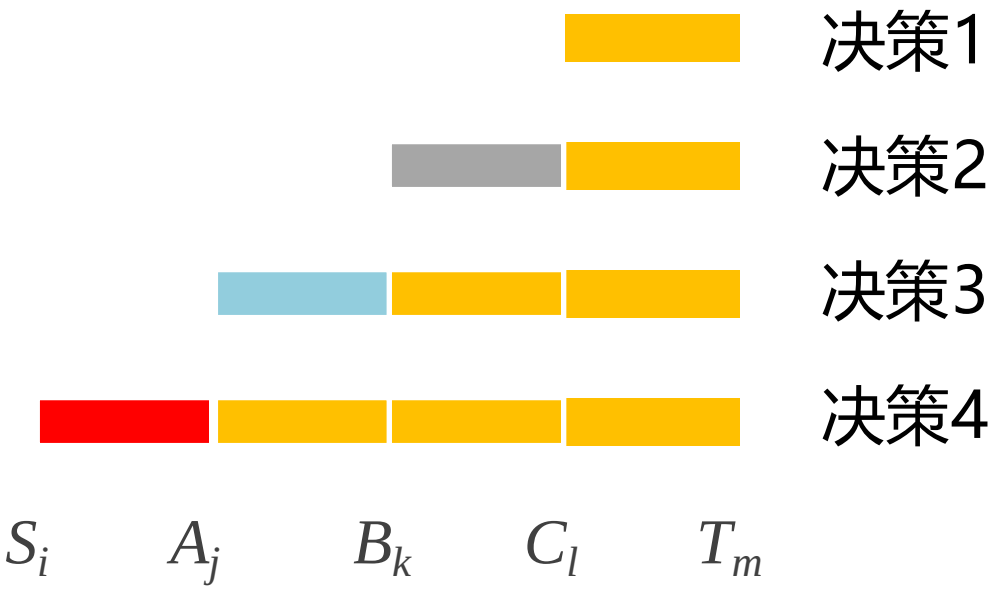

d,11
S_1
S_2
S_3
S_4
S_5
A_1
A_2
A_3
A_4
B_1
B_2
B_3
B_4
B_5
C_1
C_2
C_3
C_4
T_1
T_2
T_3
T_4
T_5
6
d,9
3
9
u,2
2
4
4
u,5
3
5
7
u,8
3
6
d,3
4
5
2
u,7
4
3
4
d,6
3
2
u,7
7
9
1
d,5
1
7
4
u,7
2
4
u,1
1
3
5
u,4
3
6
阶段4
阶段3
阶段2
阶段1

动态规划求解


d,15
S1
6
d,9
A1
3
d,11
B1
9
u,2
C1
2
T1
u,13
S2
4
4
u,5
B2
3
5
T2
7
u,8
A2
3
6
d,3
C2
4
d,10
S3
5
2
u,7
B3
4
3
T3
4
d,6
A3
3
2
u,7
C3
7
d,11
S4
9
1
d,5
B4
1
7
T4
4
u,7
A4
2
4
u,1
C4
1
u,10
S5
3
5
u,4
B5
3
6
T5
阶段4
阶段3
阶段2
阶段1

子问题界定

后边界不变, 前边界前移



最短路长的依赖关系

$$F(C_l) = \min_m\{C_l T_m\}$$ 决策1

$$F(B_k) = \min_l\{B_k C_l + F(C_l)\}$$ 决策2

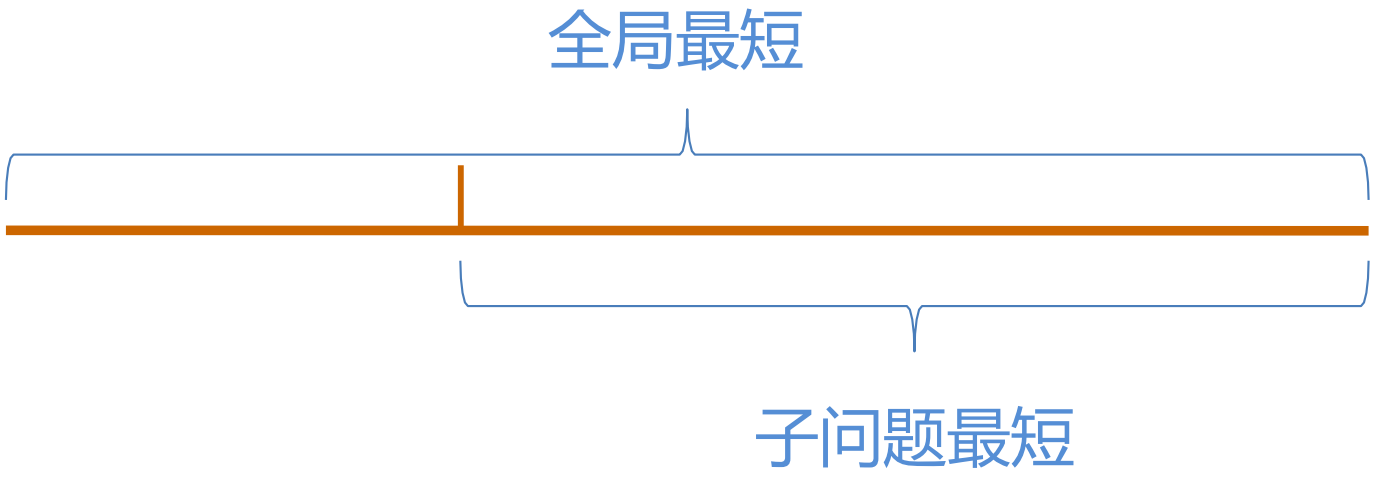
$$F(A_j) = \min_k\{A_j B_k + F(B_k)\}$$ 决策3

$$F(S_i) = \min_j\{S_i A_j + F(A_j)\}$$ 决策4

优化函数值之间存在依赖关系

优化原则：最优子结构性质

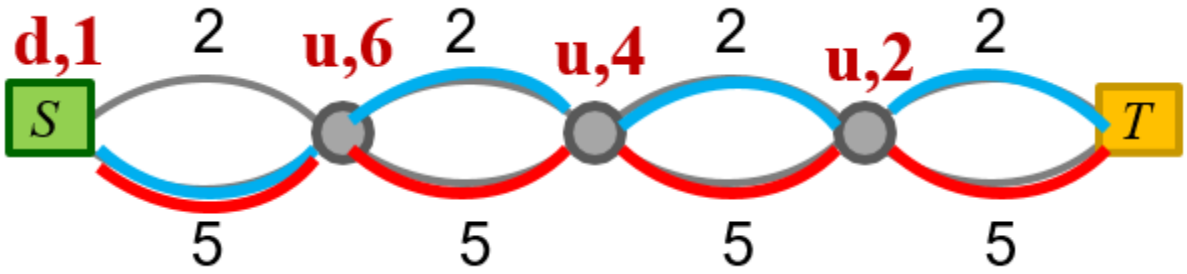
- 优化函数的特点：
  - 任何最短路的子路径相对于子问题始、终点最短



- 优化原则：
  - 一个最优决策序列的任何子序列，本身一定是相对于子序列的初始和结束状态的最优决策序列

一个反例

- 求总长模10的最小路径



- 动态规划算法的解：下, 上, 上, 上
- 最优解：下, 下, 下, 下
- 不满足优化原则，不能用动态规划

动态规划

- 求解过程是多阶段决策过程，每步处理一个子问题，可用于求解组合优化问题
- 适用条件：
- 问题要满足优化原则或最优子结构性质，即：一个最优决策序列的任何子序列本身一定是相对于子序列的初始和结束状态的最优决策序列

动态规划算法设计

动态规划设计要素：

1. 问题建模，优化的目标函数是什么？约束条件是什么？
2. 如何划分子问题（边界）？
3. 问题的优化函数值与子问题的优化函数值存在着什么依赖关系？ (递推方程)
4. 是否满足优化原则？
5. 最小子问题怎样界定？其优化函数值，即初值等于什么？

矩阵链相乘

- 问题：设 $A_1, A_2, \ldots, A_n$ 为矩阵序列，
- A_i 为 $P_{i-1} \times P_i$ 阶矩阵，$i = 1, 2, \ldots, n$. 试确定矩阵的乘法顺序，使得元素相乘的总次数最少

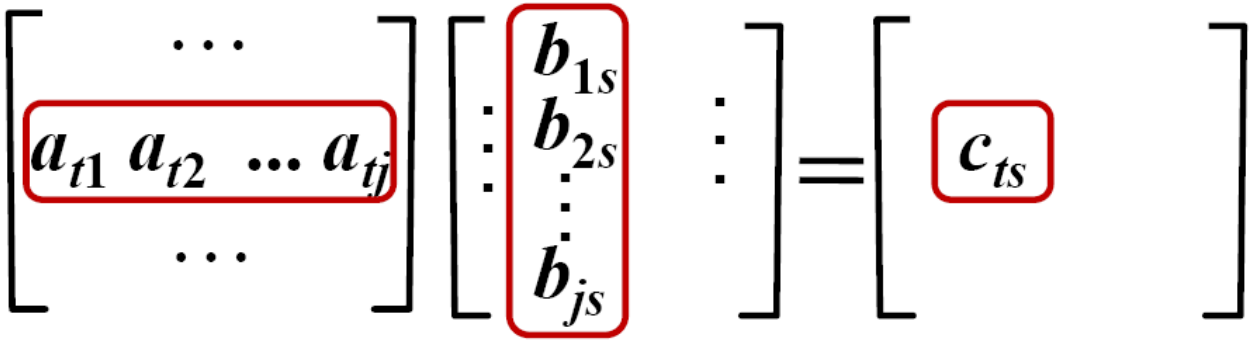
- 输入：向量

$$P = < P_0, P_1, \ldots, P_n >,$$

- 其中 $P_0, P_1, \ldots, P_n$ 为 n 个矩阵的行数与列数
- 输出：矩阵链乘法加括号的位置

矩阵相乘基本运算次数

- 矩阵A：i 行 j 列，矩阵B：j 行 k 列
- 以元素相乘作基本运算，计算 AB的工作量



$$c_{ts} = a_{t1}\,b_{1s} + a_{t2}\,b_{2s} + \ldots + a_{tj}\,b_{js}$$

- AB: i 行 k列，计算每个元素需要做 j 次乘法, 总计乘法次数为 i j k

实例

- 实例： $P = <10, 100, 5, 50>$

  A_1: 10×100， A_2: 100×5， A_3: 5×50，

- 乘法次序

  $(A_1A_2)A_3$: 10×100×5+10×5×50 = 7500

  $A_1(A_2A_3)$: 10×100×50+100×5×50=75000

- 第一种次序计算次数最少

蛮力算法

- 加n个括号的方法有 $\frac{1}{n+1}\binom{2n}{n}$ 种
- 是一个Catalan数，是指数级别

Stirling 公式

$$n! = \sqrt{2\pi n}\left(\frac{n}{e}\right)^n\left(1+\Theta\left(\frac{1}{n}\right)\right)$$

$$W(n) = \Omega\left(\frac{1}{n+1}\frac{(2n)!}{n!n!}\right)$$

$$= \Omega\left(\frac{1}{n+1}\frac{\sqrt{2\pi 2n}\left(\frac{2n}{e}\right)^{2n}}{\sqrt{2\pi n}\left(\frac{n}{e}\right)^n\sqrt{2\pi n}\left(\frac{n}{e}\right)^n}\right) = \Omega\left(2^{2n}/n^{\frac{3}{2}}\right)$$

动态规划算法

- 子问题划分
  - $A_{i..j}$：矩阵链 $A_i A_{i+1} \ldots A_j$，边界i, j
  - 输入向量：$< P_{i-1}, P_i, \ldots, P_j >$
  - 其最好划分的运算次数：$m[i, j]$

- 子问题的依赖关系
  - 最优划分最后一次相乘发生在矩阵 k 的位置，即

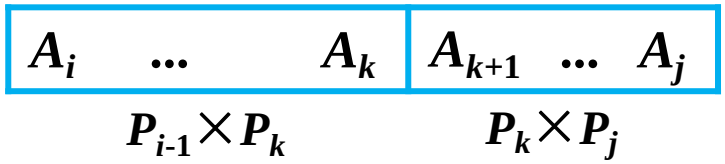
$$\boldsymbol{A_{i..j} = A_{i..k} A_{k+1..j}}$$

  - $A_{i..j}$ 最优运算次数依赖于 $A_{i..k}$ 与 $A_{k+1..j}$ 的最优运算次数

优化函数的递推方程

- 递推方程：
- $m[i,j]$：得到 $A_{i..j}$ 的最少的相乘次数

$$m[i,j]=\begin{cases}0 \\ \min\limits_{i\le k<j}\{m[i,k]+m[k+1,j]+P_{i-1}P_kP_j\} & i<j\end{cases}$$



- 该问题满足优化原则

动态规划算法设计要素

- 多阶段决策过程，每步处理一个子问题，界定子问题的边界
- 列出优化函数的递推方程及初值
- 问题要满足优化原则或最优子结构性质，即：一个最优决策序列的任何子序列本身一定是相对于子序列的初始和结束状态的最优决策序列

动态规划算法的递归实现

- 算法1 RecurMatrixChain (P, i, j)　　//子问题i-j

1. $m[i,j] \leftarrow \infty$
2. $s[i,j] \leftarrow i$
3. for $k \leftarrow i$ to $j-1$ do　　//划分位置k
4. $q \leftarrow$ RecurMatrixChain(P,i,k) + RecurMatrixChain($P,k+1,j$) + $p_{i-1} p_k p_j$
5. if $q < m[i,j]$　　//找到更好的解
6. then $m[i,j] \leftarrow q$
7. $s[i,j] \leftarrow k$
8. return $m[i,j]$

- 这里没有写出算法的全部描述（递归边界）

算法分析

- 时间复杂度的递推方程

$$T(n) \geq \begin{cases} O(1) & n = 1 \\ \sum_{k=1}^{n-1} (T(k) + T(n-k) + O(1)) & n > 1 \end{cases}$$

$$T(n) \geq O(n) + \sum_{k=1}^{n-1} T(k) + \sum_{k=1}^{n-1} T(n-k)$$

$$T(n) \geq O(n) + 2\sum_{k=1}^{n-1} T(k)$$

时间复杂度

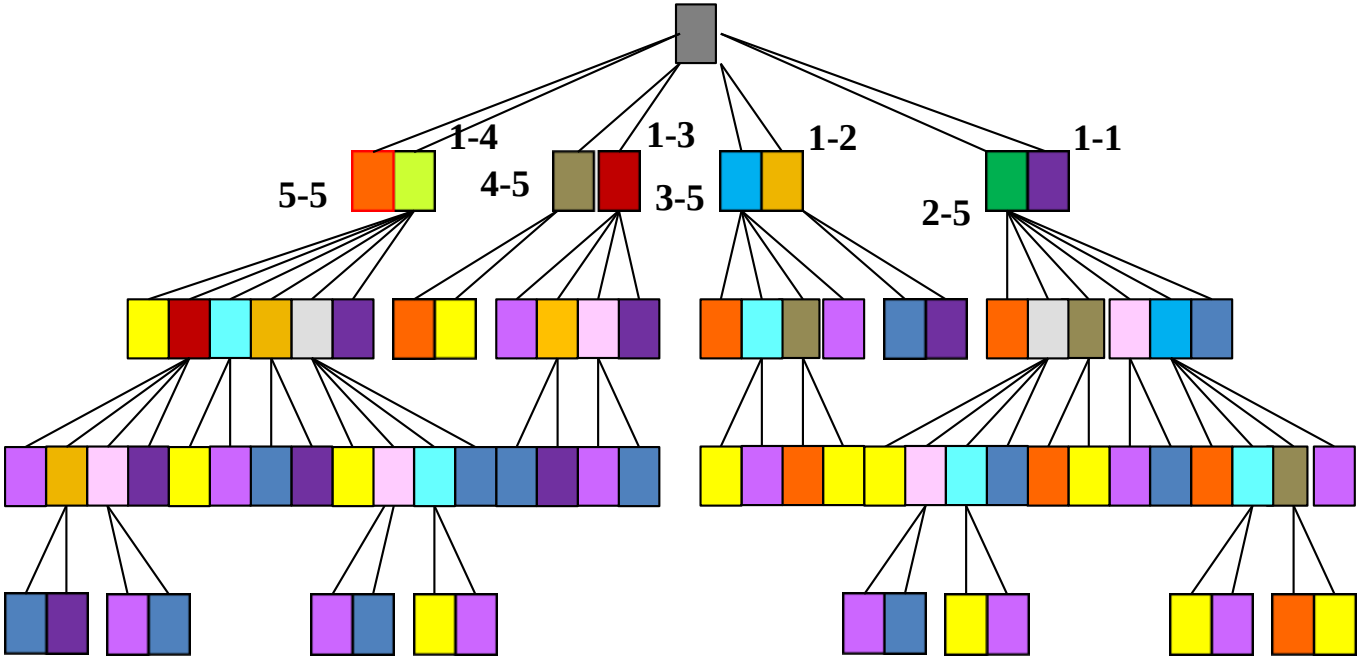
- 数学归纳法证明：$T(n) \geq 2^{n-1}$
- n=2, 显然为真
- 假设对于任何小于n 的 k, 命题为真

$$T(n) \geq O(n) + 2\sum_{k=1}^{n-1} T(k)$$

$$\geq O(n) + 2\sum_{k=1}^{n-1} 2^{k-1}$$

$$= O(n) + 2(2^{n-1} - 1) \geq 2^{n-1}$$

子问题的产生, n=5



- 划分： $A_{1..4}A_{5..5}$， $A_{1..3}A_{4..5}$, $A_{1..2}A_{3..5}$,$A_{1..1}A_{2..5}$
- 产生 8 个子问题，即第一层的 8 个结点.

子问题的计数

边界	次数	边界	次数	边界	次数
1-1	**8**	1-2	**4**	2-4	**2**
2-2	**12**	2-3	**5**	3-5	**2**
3-3	**14**	3-4	**5**	1-4	**1**
4-4	**12**	4-5	**4**	2-5	**1**
5-5	**8**	1-3	**2**	1-5	**1**

- 边界不同的子问题：15 个
- 递归计算的子问题：81 个

结论

- 与蛮力算法相比较，动态规划算法利用了子问题优化函数间的依赖关系，时间复杂度有所降低
- 动态规划算法的递归实现效率不高，原因在于同一子问题多次重复出现，每次出现都需要重新计算一遍
- 采用空间换时间策略，记录每个子问题首次计算结果，后面再用时就直接取值，每个子问题只算一次

动态规划算法的迭代实现

- 迭代计算的关键：
- 每个子问题只计算一次
- 迭代过程
  - 从最小的子问题算起
  - 考虑计算顺序，以保证后面用到的值前面已经计算好
  - 存储结构保存计算结果——备忘录
- 解的追踪
  - 设计标记函数标记每步的决策
  - 考虑根据标记函数追踪解的算法

矩阵链乘法不同子问题

- 长度1：只含1个矩阵，有n个子问题 (不需要计算)
- 长度2：含2个矩阵，n-1个子问题
- 长度3：含3个矩阵，n-2个子问题
- ...
- 长度n-1：含n-1个矩阵，2个子问题
- 长度 n：原始问题，只有1个

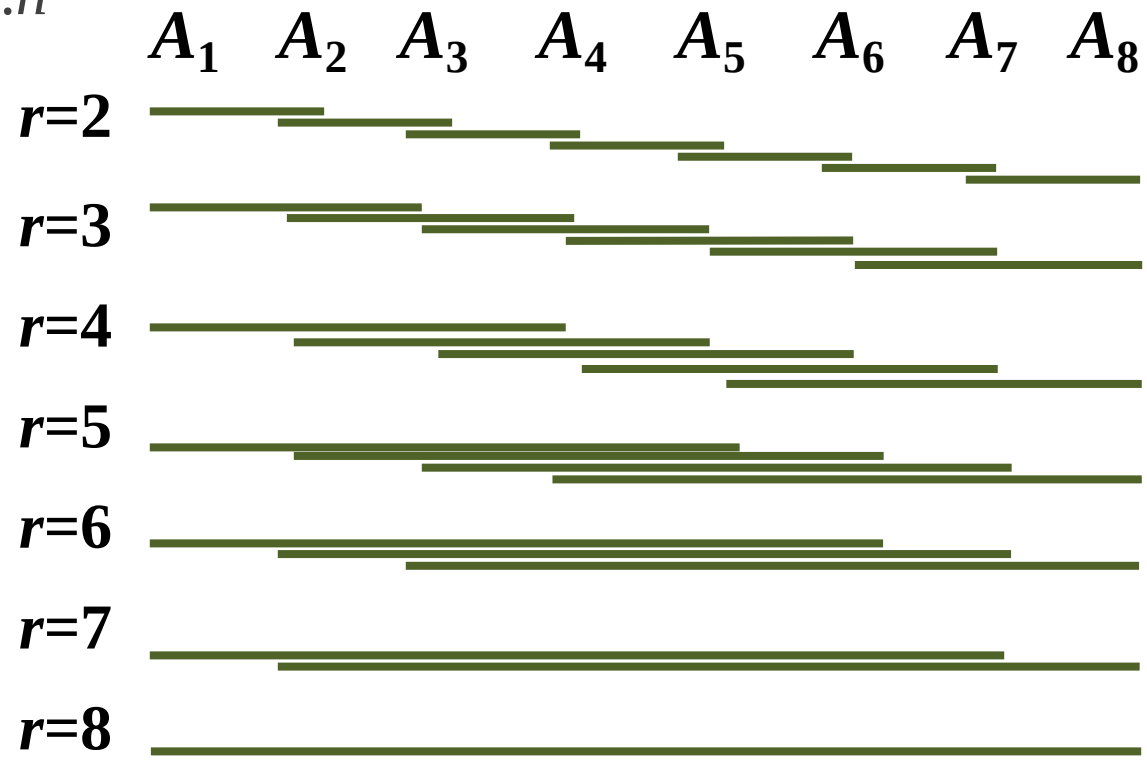
矩阵链乘法迭代顺序

- 长度为 1：初值，$m[i, i] = 0$
- 长度为 2：1..2, 2..3, 3..4, ... , n-1..n
- 长度为 3：1..2, 2..3, 3..4, ... , n-1..n

. . .

- 长度为 n-1：1..n-1,2..n
- 长度为 n：1..n

- n=8时，计算顺序如右图所示



伪代码

算法MatrixChain (P, n)

1. 令所有的 $m[i,i]$初值为0
2. for $r \leftarrow 2$ to n do // r为链长
3. for $i \leftarrow 1$ to $n-r+1$ do // 左边界i
4. $j \leftarrow i+r-1$ // 右边界j
5. $m[i,j] \leftarrow m[i+1,j]+p_{i-1}p_i p_j$ //$k=i$
6. $s[i,j] \leftarrow i$ //记录k
7. for $k \leftarrow i+1$ to $j-1$ do // 遍历k
8. $t \leftarrow m[i,k]+m[k+1,j]+p_{i-1}p_k p_j$
9. if $t<m[i,j]$
10. then $m[i,j] \leftarrow t$ //更新解
11. $s[i,j] \leftarrow k$

二维数组m与s为备忘录

时间复杂度

- 根据伪码：行 2, 3, 7 都是$O(n)$，循环执行$O(n^3)$次，内部为$O(1)$

$$W(n) = O(n^3)$$

- 根据备忘录：估计每项工作量, 求和. 子问题有$O(n^2)$个，确定每个子问题的
- 最少乘法次数需要对不同划分位置比较，需要$O(n)$时间

$$W(n) = O(n^3)$$

- 追踪解工作量 $O(n)$，总工作量$O(n^3)$

实例

- 输入： $P = < 30, 35, 15, 5, 10, 20 >$,
  $n = 5$
- 矩阵链：$A_1 A_2 A_3 A_4 A_5$，其中
  - A_1: 30×35，A_2: 35×15，A_3: 15×5，
    - A_4: 5×10，A_5: 10×20
- 备忘录：存储所有子问题的最小乘法次数及得到这个值的划分位置

备忘录 $m[i,j]$

- $P = < 30, 35, 15, 5, 10, 20 >$

$r=1$	$m[1,1]=0$	$m[2,2]=0$	$m[3,3]=0$	$m[4,4]=0$	$m[5,5]=0$
$r=2$	$m[1,2]=15750$	$m[2,3]=2625$	$m[3,4]=750$	$m[4,5]=1000$	
$r=3$	$m[1,3]=7875$	$m[2,4]=4375$	$m[3,5]=2500$		
$r=4$	$m[1,4]=9375$	$m[2,5]=7125$			
$r=5$	$m[1,5]=11875$				

- $m[2,5] = \min\{ 0+2500+35\times15\times20, 2625+1000+35\times5\times20, 4375+0+35\times10\times20\} = 7125$

标记函数 $s[i,j]$

r=2	s[1,2]=1	s[2,3]=2	s[3,4]=3	s[4,5]=4	
r=3	s[1,3]=1	s[2,4]=3	s[3,5]=3		
r=4	s[1,4]=3	s[2,5]=3			
r=5	s[1,5]=3				

- 解的追踪：$s[1,5]=3 \Rightarrow (A_1A_2A_3)(A_4A_5)$
  $s[1,3]=1 \Rightarrow A_1(A_2A_3)$
- 输出
- 计算顺序：$(A_1(A_2A_3))(A_4A_5)$
- 最少的乘法次数：$m[1,5]=11875$

两种实现的比较

- 递归实现：时间复杂性高，空间较小
- 迭代实现：时间复杂性低，空间消耗多

- 原因：递归实现子问题多次重复计算，子问题计算次数呈指数增长，迭代实现每个子问题只计算一次

- 动态规划时间复杂度：
- 备忘录各项计算量之和 + 追踪解工作量
- 通常追踪工作量不超过计算工作量，是问题规模的多项式函数

动态规划算法的要素

- 划分子问题，确定子问题边界，将问题求解转变成多步判断的过程
- 定义优化函数,以该函数极大(或极小) 值作为依据,确定是否满足优化原则
- 列优化函数的递推方程和边界条件
- 自底向上计算，设计备忘录 (表格)
- 考虑是否需要设立标记函数
- 用递推方程或备忘录估计时间复杂度