动态规划

陈长建

计算机科学系

关于第一次实验

- **院楼103**，10月13日（**明天**）上午8:30-12:00（现场验收）
- 第一次实验离线题（离线准备）
  - 1. 分治法查找最大最小值
  - 2. 分治法实现合并排序
  - 3. 实现题1-3 最多约数问题
- 在线题
  - OJ系统还在准备中

关于第一次实验

- 注意事项
  - 1. OJ系统使用：请参考系统使用说明书提前注册，注册用户名要求为：JK0406_学号（请严格按照此格式命名，否则无法获取大家完成的情况）。
  - 2. 本系统仅支持C、C++、Java，请大家提前熟悉。
  - 3. 在线题会随机为每位同学分配一道题目，会在实验开始的时候进行公布。
  - 4. 离线题请大家按照课件中的要求准备运行环境和实验报告，助教现场进行验收。

回顾-快速排序

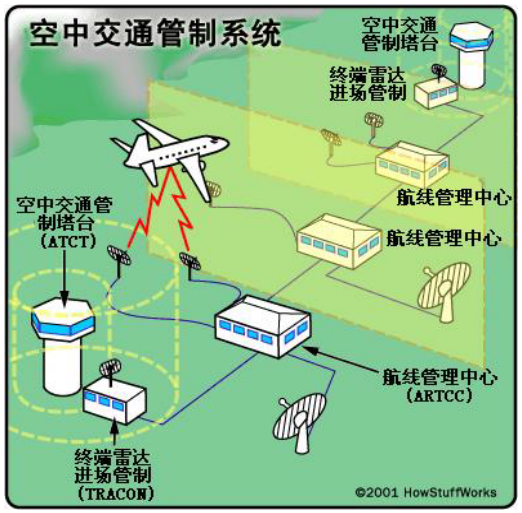
- 快速排序是一种基于划分的排序方法
- 快速排序的平均时间复杂度?
- 快速排序的最坏时间复杂度?

回顾-线性时间选择

- 问题描述
  - 给出含有n个元素表A[0:n-1]，要求确定其中的第k小元素。
- 平均时间复杂度?
- 最坏时间复杂度?
  - 能否更好?

分治-例7 最接近点对问题

- 给定平面上n个点的集合S，找其中的一对点，使得在n个点组成的所有点对中，该点对间距离最小。
  - 应用：空中交通管制



直接求解方法

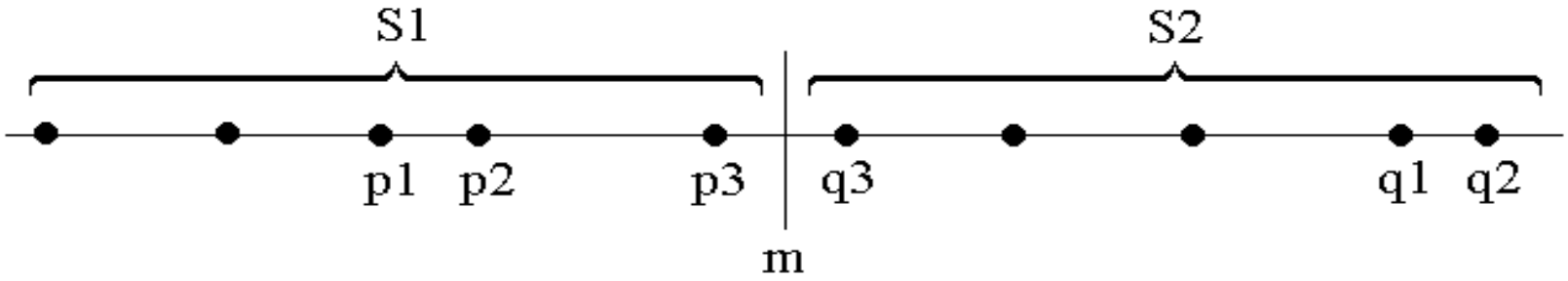
- 分别计算每一对点之间的距离，然后找出距离最小的那一对。
- 时间复杂度

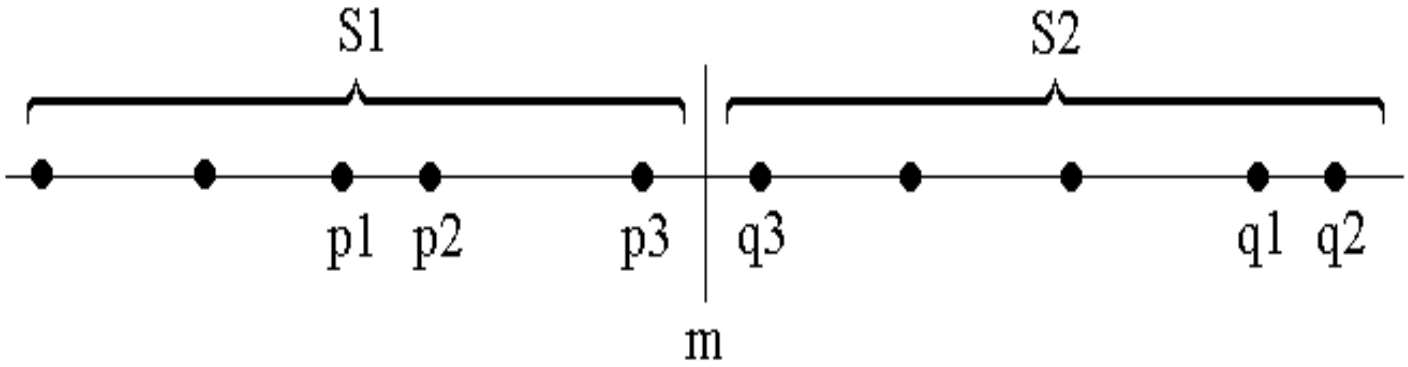
$$C(n) = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} 2 = 2\sum_{i=1}^{n-1} n - i$$

$$= 2[(n-1) + (n-2) + \ldots + 1]$$

$$= (n-1)n \in \Theta(n^2)$$

- 为了使问题易于理解和分析，先来考虑**一维**的情形。此时，S中的n个点退化为x轴上的n个实数 x1,x2,…,xn。最接近点对即为这n个实数中相差最小的2个实数。
  - 假设我们用x轴上某个点m将S划分为2个子集S1和S2，基于**平衡子问题**的思想，用S中各点坐标的中位数来作分割点。
  - 递归地在S1和S2上找出其最接近点对{p1,p2}和{q1,q2}，并设**d=min{|p1-p2|,|q1-q2|}**，S中的最接近点对或者是{p1,p2}，或者是{q1,q2}，或者是某个{p3,q3}，其中p3∈S1且q3∈S2。
  - {p3,q3}可能的数量为$n^2/4$
  - **能否在线性时间内找到p3,q3?**





能否在线性时间内找到p3,q3?

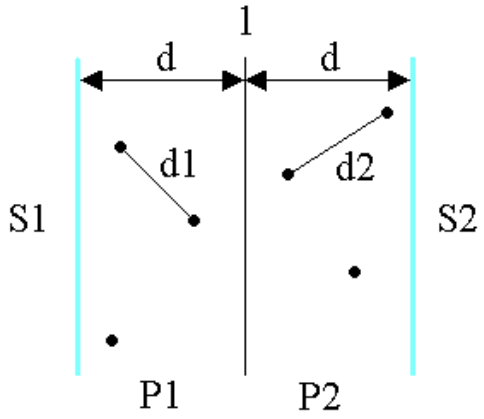
- 如果S的最接近点对是{p3,q3}，即|p3-q3|<d，则p3和q3两者与m的距离不超过d，即p3∈(m-d,m]，q3∈(m,m+d]。
- 由于在S1中，每个长度为d的半闭区间至多包含一个点（否则必有两点距离小于d），并且m是S1和S2的分割点，因此(m-d,m]中至多包含S中的一个点。由图可以看出，如果(m-d,m]中有S中的点，则此点就是S1中最大点。
- 因此，我们用线性时间就能找到区间(m-d,m]和(m,m+d]中所有点，即p3和q3。从而我们用线性时间就可以将S1的解和S2的解合并成为S的解。

```
public static double cpair1(S)
{     n=|S|;
      if (n < 2) return  ∞;
      m=S中各点坐标的中位数；；
     d1=cpair1(S1); // S1={x∈S|x<=m}, 构造S1和S2
     d2=cpair1(S2); // S2={x∈S|x>m}, 构造S2
     p=max(S1);
     q=min(S2);
     d=min(d1,d2,q-p);
     return d;  }
```

$$T(n) = \begin{cases} O(1) & n < 4 \\ 2T(n/2) + O(n) & n \geq 4 \end{cases}$$

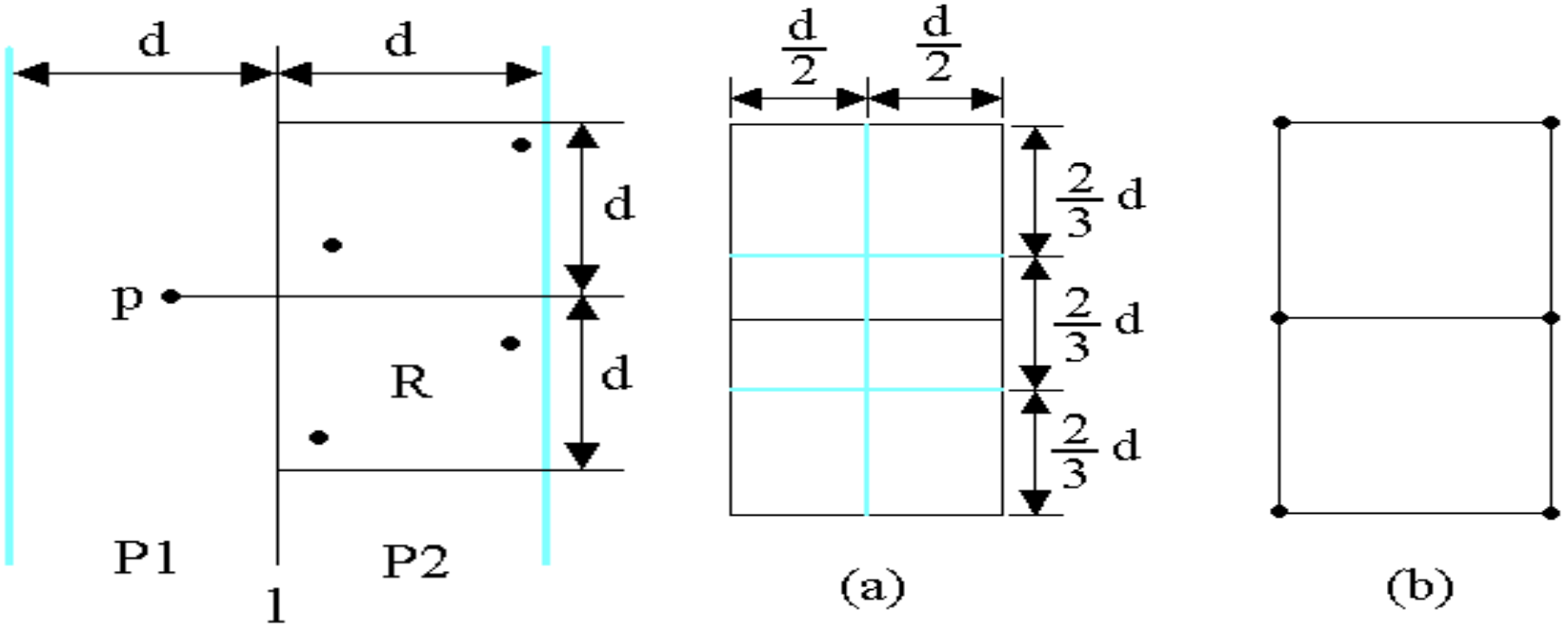
二维情形

- 下面来考虑二维的情形。
- 选取一垂直线 l: x = m 来作为分割直线。其中m为S中各点x坐标的中位数。由此将S分割为S1和S2。
- 递归地在S1和S2上找出其最小距离d1和d2，并设d = min{d1,d2}，S中的最接近点对或者是d，或者是某个{p,q}，其中p∈P1 且 q∈P2。
- 能否在线性时间内找到p,q?



二维情形

- 考虑P1中任意一点p，它若与P2中的点q构成最接近点对的候选者，则必有distance(p，q)＜d。满足这个条件的P2中的点一定落在一个d×2d的矩形R中
- 由d的意义可知，P2 中任何 2 个 S 中的点的距离都不小于d。由此可以推出矩形R中最多只有6个S中的点。
- 因此，在分治法的合并步骤中最多只需要计算 n/2×6 个点对。



二维情形

证明:将矩形R的长为2d的边3等分，将它的长为d的边2等分，由此导出6个(d/2)×(2d/3)的矩形。若矩形R中有多于6个S中的点，则由鸽舍原理易知至少有一个(d/2)×(2d/3)的小矩形中有2个以上S中的点。设u，v是位于同一小矩形中的2个点，则:

$$(x(u)-x(v))^2+(y(u)-y(v))^2 \le (d/2)^2+(2d/3)^2$$

$$\le \frac{25}{36}d^2$$

distance(u,v)<d。这与d的意义相矛盾。

二维情形

- 要检查哪6个点?
  - 将p和P2中所有S2的点投影到垂直线 l上。由于能与p点一起构成最接近点对候选者的S2中点一定在矩形R中，所以它们在直线l上的投影点距p在l上投影点的距离小于d，这种投影点最多只有6个。
- 因此，若将P1和P2中所有S中点按其y坐标排好序，则对P1中所有点，对排好序的点列作一次扫描，就可以找出所有最接近点对的候选者。对P1中每一点最多只要检查P2中排好序的相继6个点。

```
double cpair2(S)
{
    n=|S|;
    if (n < 2) return ;
1、  m=S中各点x间坐标的
    中位数;构造S1和S2;
    S1={p∈S|x(p)<=m},
    S2={p∈S|x(p)>m}
2、  d1=cpair2(S1);
    d2=cpair2(S2);
3、  dm=min(d1,d2);
```

4、设P1 是S1中距垂直分割线 l 的距离在dm 之内的所有点组成的集合；

P2 是S2 中距分割线 l 的距离在dm 之内所有点组成的集合；

将P1 和P2 中点依其 y 坐标值排序；

并设X 和Y 是相应的已排好序的点列；

5、通过扫描 X 以及对于 X 中每个点检查Y中与其距离在dm 之内的所有点(最多6个)可以完成合并；

当X中的扫描指针逐次向上移动时，Y中的扫描指针可在宽为2dm的区间内移动；

设dl 是按这种扫描方式找到的点对间的最小距离；

```
6、  d=min(dm,dl);
    return d;
}
```

最接近点对问题

$$T(n)=\begin{cases} O(1) & n<4 \\ 2T(n/2)+O(n) & n\geq 4 \end{cases}$$

复杂度分析

n≥4,

$T(n)=2T(n/2)+O(n)=2[2T(n/4)+O(n/2)]+O(n)$

$=2^2T(n/2^2)+2O(n)=\cdots\cdots$

$n=2^k$ $=2^k+kO(n)$

$k\geq 2$ **$T(n)=O(n\log n)$**

分治-例8 循环赛日程表

- 有n=2k个运动员。设计一个满足以下要求的比赛日程表：
  - (1)每个选手必须与其他n-1个选手各赛一次；
  - (2)每个选手一天只能赛一次；
  - (3)循环赛一共进行n-1天。

思想

- 按分治策略，将所有的选手分为两半
  - n个选手的比赛日程表就可以通过为n/2个选手设计的比赛日程表来决定。
- 递归地对选手进行分割，直到只剩下2个选手时，

只要让这2个选手进行比赛就可以了。

示例

1	2	3	4	5	6	7	8
2	1	4	3	6	5	8	7
3	4	1	2	7	8	5	6
4	3	2	1	8	7	6	5
5	6	7	8	1	2	3	4
6	5	8	7	2	1	4	3
7	8	5	6	3	4	1	2
8	7	6	5	4	3	2	1

第二次作业

- 1、算法实现题 2-2 马的Hamilton周游路线问题
- 2、 算法实现题 2-3 半数集问题
- 3、算法实现题 2-6 排列的字典序问题
- 4、算法实现题 2-7 集合划分问题

- 提交时间：10月21日

本章要点:

- 理解动态规划算法的概念
- 掌握动态规划算法的基本要素
- 掌握设计动态规划算法的步骤
- 通过范例学习动态规划算法设计策略
  - 最大子段和、最长公共子序列、矩阵链连乘、凸多边形最优三角剖分、0-1背包问题

动态规划的历史（1/5）

- 有许多问题，用穷举法才能得到最佳解。苦输入量n稍大一些，计算量太大，特别对渐近时间复杂性为输入量的指数函数的问题，计算机无法完成。
- 采用动态规划（Dynamic programming）能得到比穷举法更有效的算法。动态规划的指导思想是，在每种情况下，列出各种可能的局部解，从局部解中挑出那些有可能产生最佳的结果而扬弃其余，从而大大缩减了计算量。

动态规划的历史（2/5）

- 动态规划遵循“最佳原理”
  - “一个最优策略的子策略总是最优的”。动态规划是运筹学的一个分支，它是解决多阶段决策过程最优化的一种方法，大约产生于50年代，由美国数学家贝尔曼（R·Bellman）等人，根据一类多阶段决策问题的特点，把多阶段决策问题变换为一系列互相联系单阶段问题，然后逐个加以解决。

动态规划的历史（3/5）

- 动态规划开始只是应用于多阶段决策性问题，后来渐渐被发展为解决离散最优化问题的有效手段，进一步应用于一些连续性问题上。然而，动态规划更像是一种思想而非算法，它没有固定的数学模型，没有固定的实现方法，其正确性也缺乏严格的理论证明。因此，一直以来动态规划的数学理论模型是一个研究的热点。

动态规划的历史（4/5）


贝尔曼,R.

- 贝尔曼，R Richard Bellman (1920～1984)

  美国数学家，美国科学院院士，动态规划的创始人。1920年8月26日生于美国纽约。1984年3月19日逝世。1941年在布鲁克林学院毕业，获理学士学位，1943年在威斯康星大学获理学硕士学位，1946年在普林斯顿大学获博士学位。1946～1948年在普林斯顿大学任助理教授,1948～1952年在斯坦福大学任副教授,1953～1956年在美国兰德公司任研究员，1956年后在南加利福尼亚大学任数学教授、电气工程教授和医学教授。

  贝尔曼因提出动态规划而获美国数学会和美国工程数学与应用数学会联合颁发的第一届维纳奖金（1970），卡内基－梅隆大学颁发的第一届迪克森奖金(1970)，美国管理科学研究会和美国运筹学会联合颁发的诺伊曼理论奖金(1976)。1977年贝尔曼当选为美国艺术与科学研究院院士和美国工程科学院院士。

动态规划的历史（5/5）

- 贝尔曼因在研究多段决策过程中提出动态规划而闻名于世。1957年他的专著《动态规划》出版后，被迅速译成俄文、日文、德文和法文，对控制理论界和数学界有深远影响。贝尔曼还把不变嵌入原理应用于理论物理和数学分析方面，把两点边值问题化为初值问题，简化了问题的分析和求解过程。1955年后贝尔曼开始研究算法、计算机仿真和人工智能，把建模与仿真等数学方法应用到工程、经济、社会和医学等方面，取得许多成就。贝尔曼对稳定性的矩阵理论、时滞系统、自适应控制过程、分岔理论、微分和积分不等式等方面都有过贡献。

  贝尔曼曾是《数学分析与应用杂志》及《数学生物科学杂志》的主编，《科学与工程中的数学》丛书的主编。已出版30本著作和7本专著，发表了600多篇研究论文。

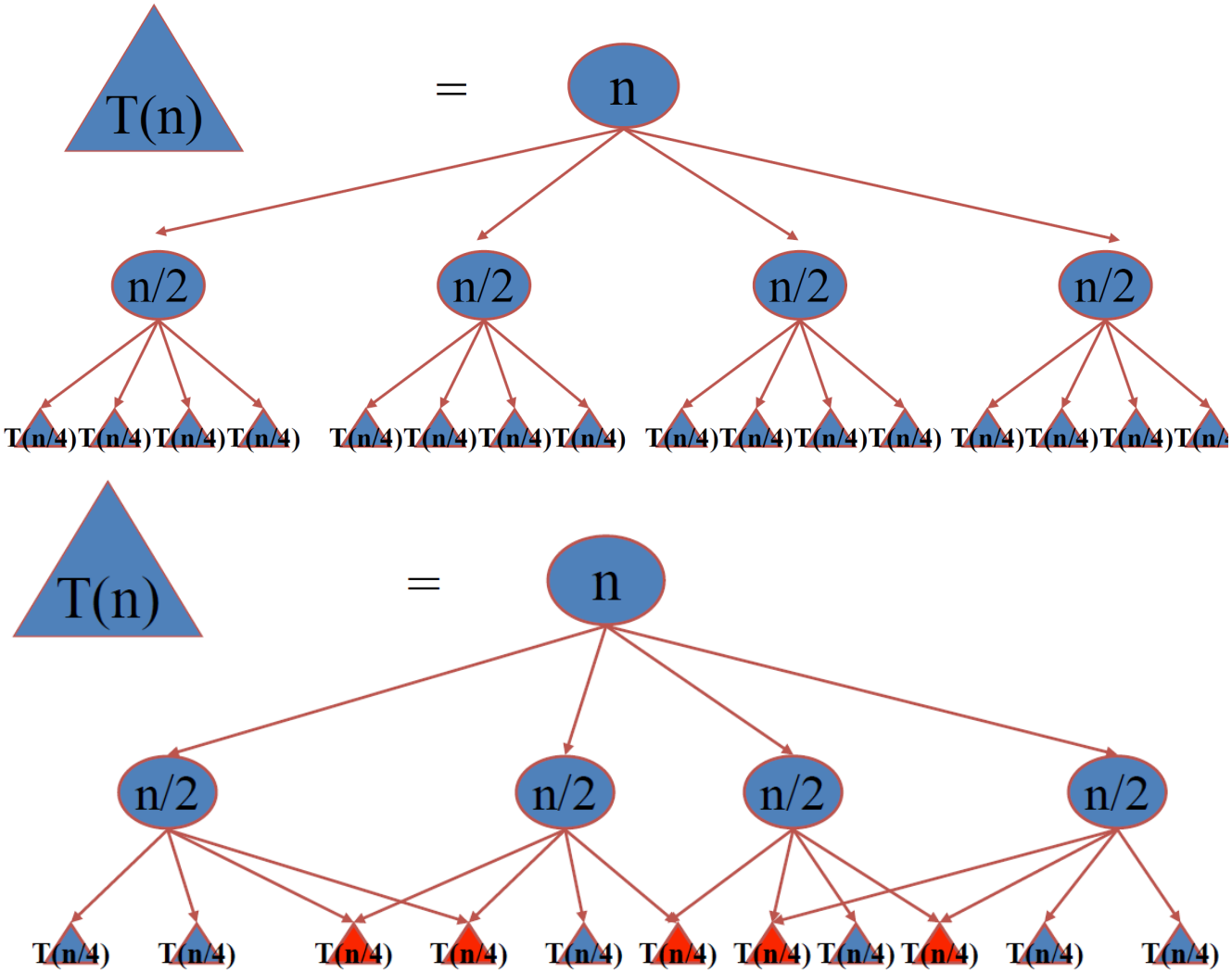
动态规划的基本思想

- 动态规划算法与分治法类似,其基本思想也是将待求解问题分解成若干个子问题,先求解子问题,然后从这些子问题的解得到原问题的解。

- 动态规划算法与分治法不同的是，经分解得到的子问题往往不是相互独立的，有大量子问题会重复出现。

- 为了避免重复计算，动态规划法是用一个表来存放一计算过的子问题。

动态规划和分治法的对比



动态规划算法适用于求解最优化问题

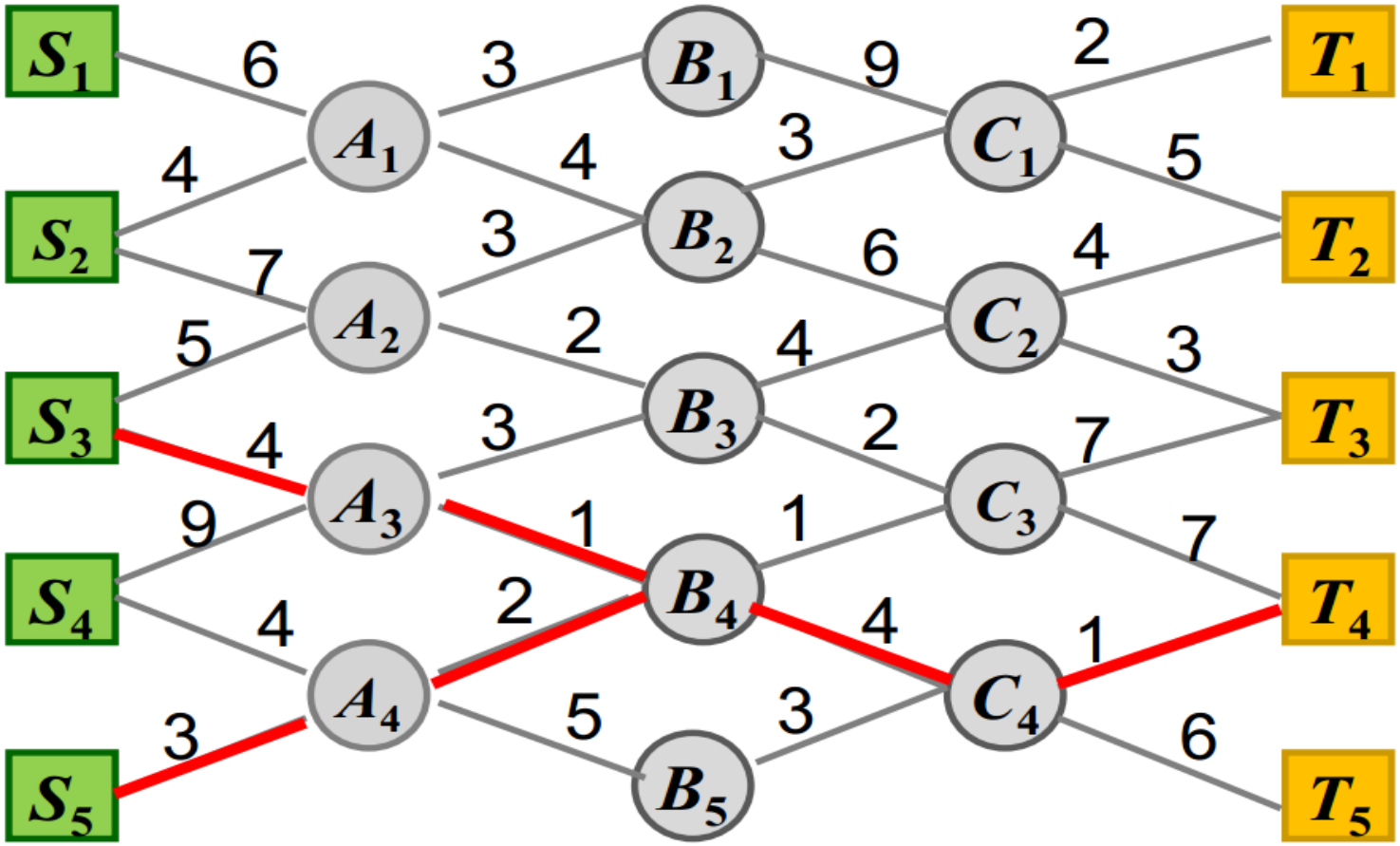
通常按如下四步骤设计动态规划算法：

- 找出最优解的性质，并刻画其结构特征；
- 递归定义求最优值的公式；
- 以自底向上方式计算最优值；
- 根据计算最优值时得到的信息，构造最优解。

例1-最短路径问题（1/10）

- 最短路径问题描述：
  - 输入：
    - 起点集合{ S1, S2, ..., Sn } ，
    - 终点集合{T1, T2, ... , Tm}，
    - 中间结点集，{A1, ... ,B1, ... , C1....}
    - 边集E，对于任意边e有长度
  - 最短路径问题描述：
    - 输出：一条从起点到终点的最短路

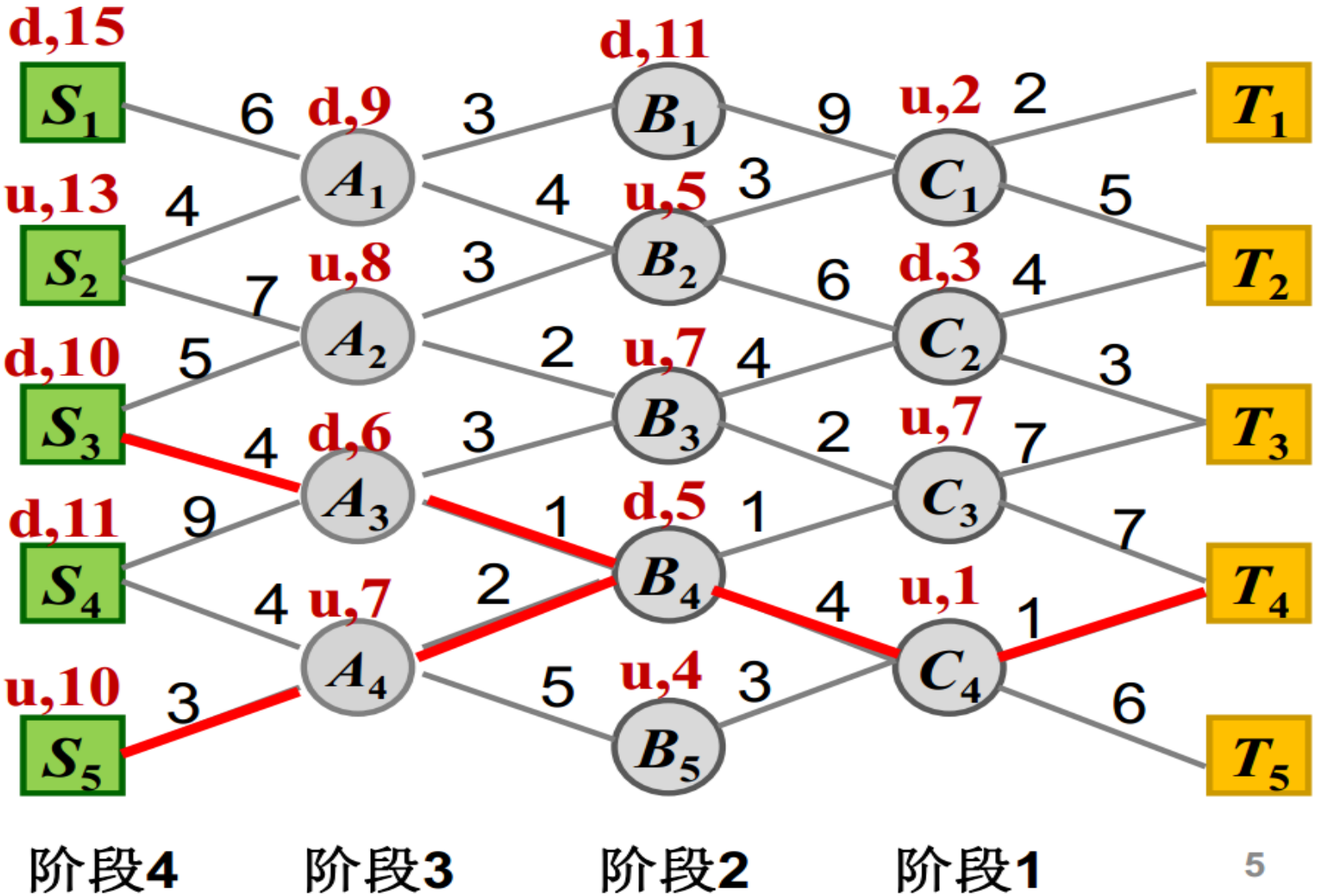
例1-最短路径问题（2/10）



例1-最短路径问题（3/10）

- 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。

例1-最短路径问题（4/10）

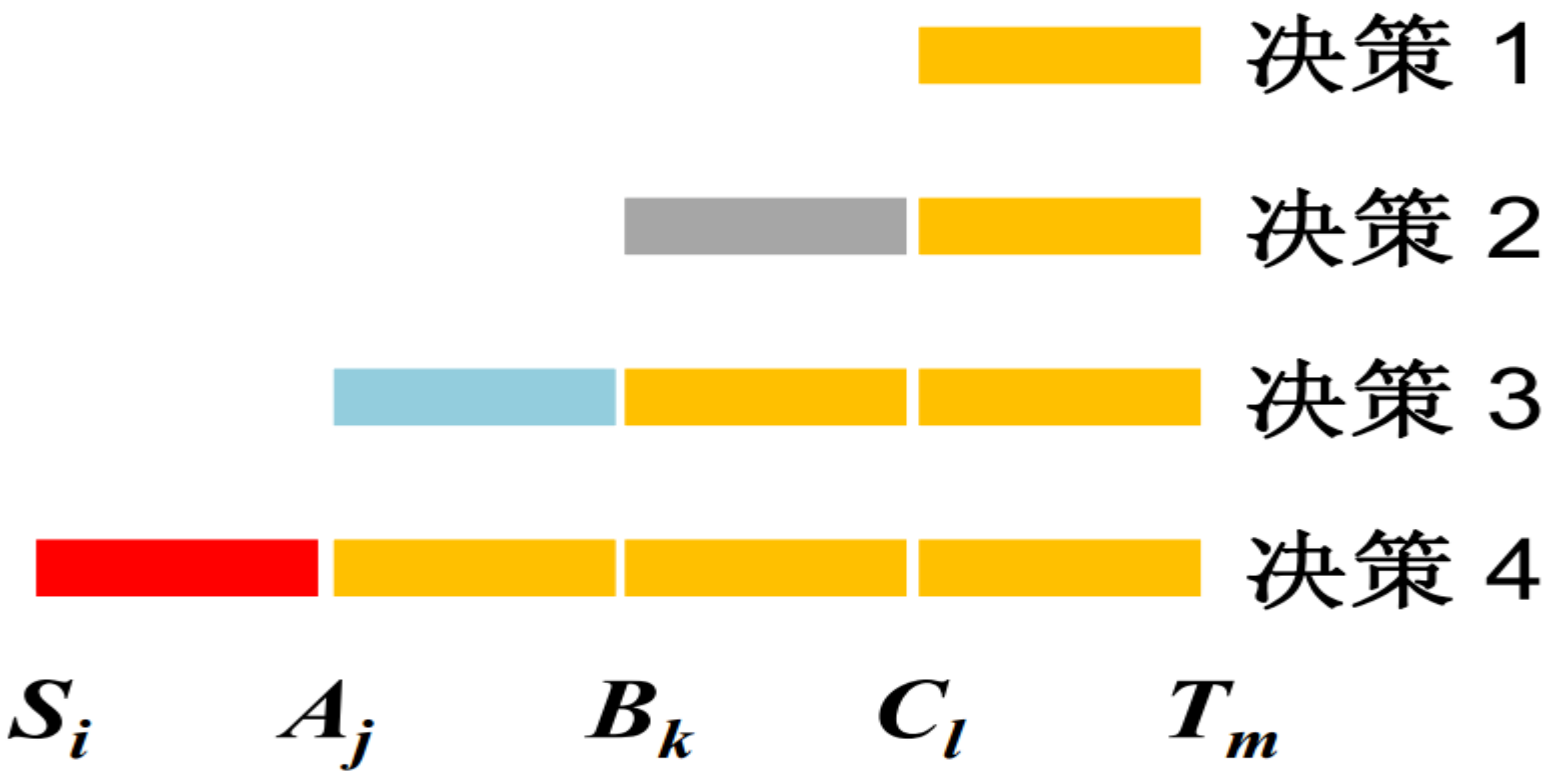


例1-最短路径问题（5/10）

- 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。

例1-最短路径问题（6/10）

前边界不变，后边界前移



例1-最短路径问题（7/10）

$$F(C_l) = \min_m \{C_l T_m\} \quad \text{决策 1}$$

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

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

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

优化函数值之间存在依赖关系

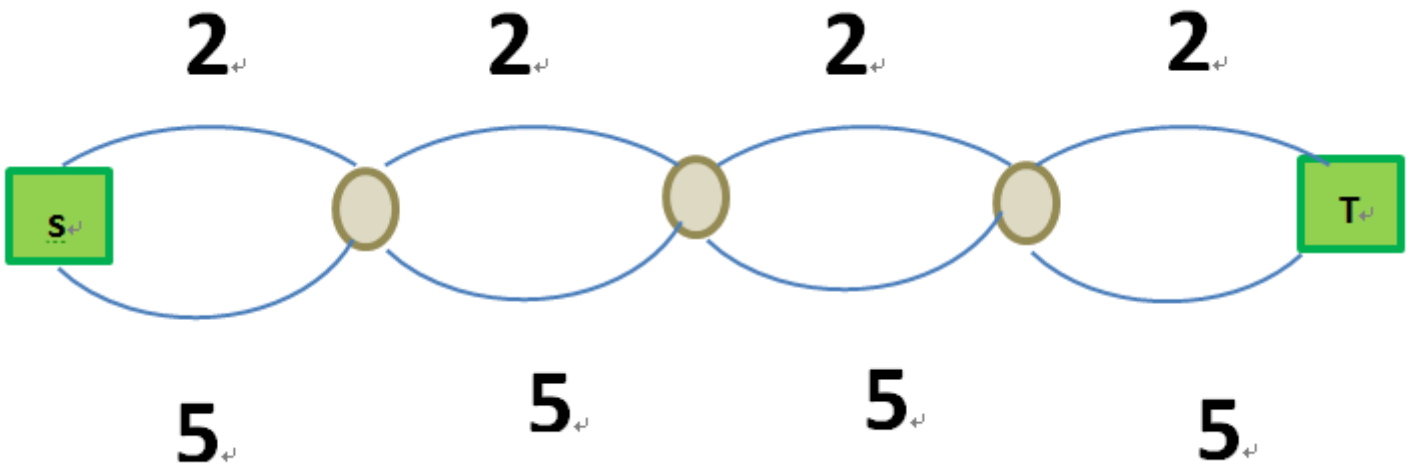
例1-最短路径问题（8/10）

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

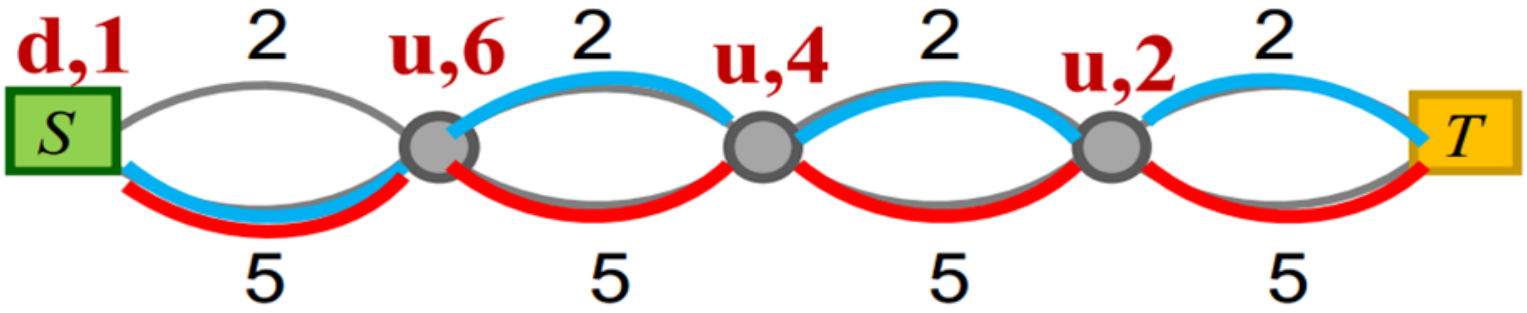
优化原则：一个最优决策序列的任何子序列本身一定是相对于子序列的初始和结束状态的最优决策序列

例1-最短路径问题（9/10）

求总长模10的最小路径



例1-最短路径问题（10/10）



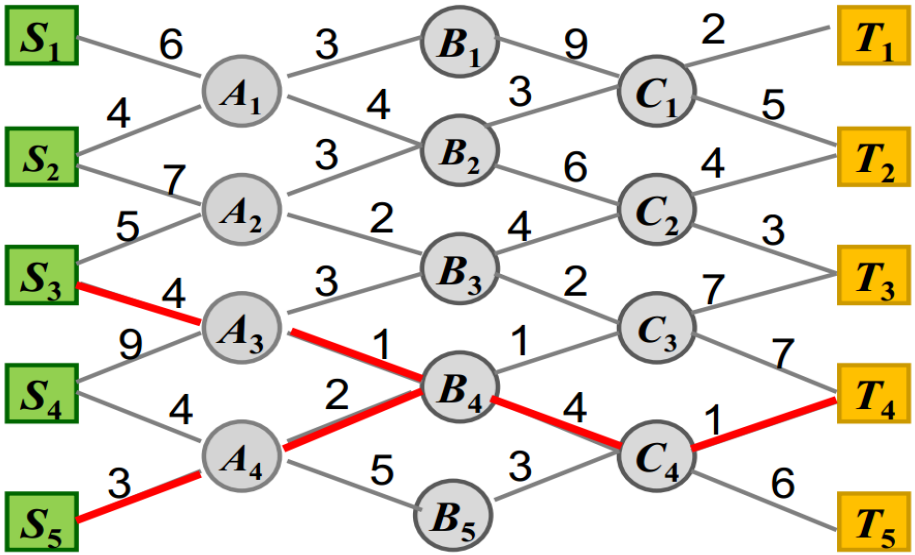
动态规划算法的解：下，上，上，上

最优解：下，下，下，下

不满足优化原则，不能用动态规划

4个步骤分解

- 找出最优解的性质，并刻画其结构特征；
  - 对于第l列中的某一个节点，其到终点的最短路径是其到第l+1列中任意一个节点加上该节点到终点距离中最短一个 **正确性?**



4个步骤分解

- 递归定义求最优值的公式；

$$F(C_l) = \min_m \{C_l T_m\}$$

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

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

$$F(S_i) = \min_i \{S_i A_j + F(A_j)\}$$

4个步骤分解

- 以自底向上方式计算最优值（递归实现）

$$F(C_l) = \min_m \{C_l T_m\}$$

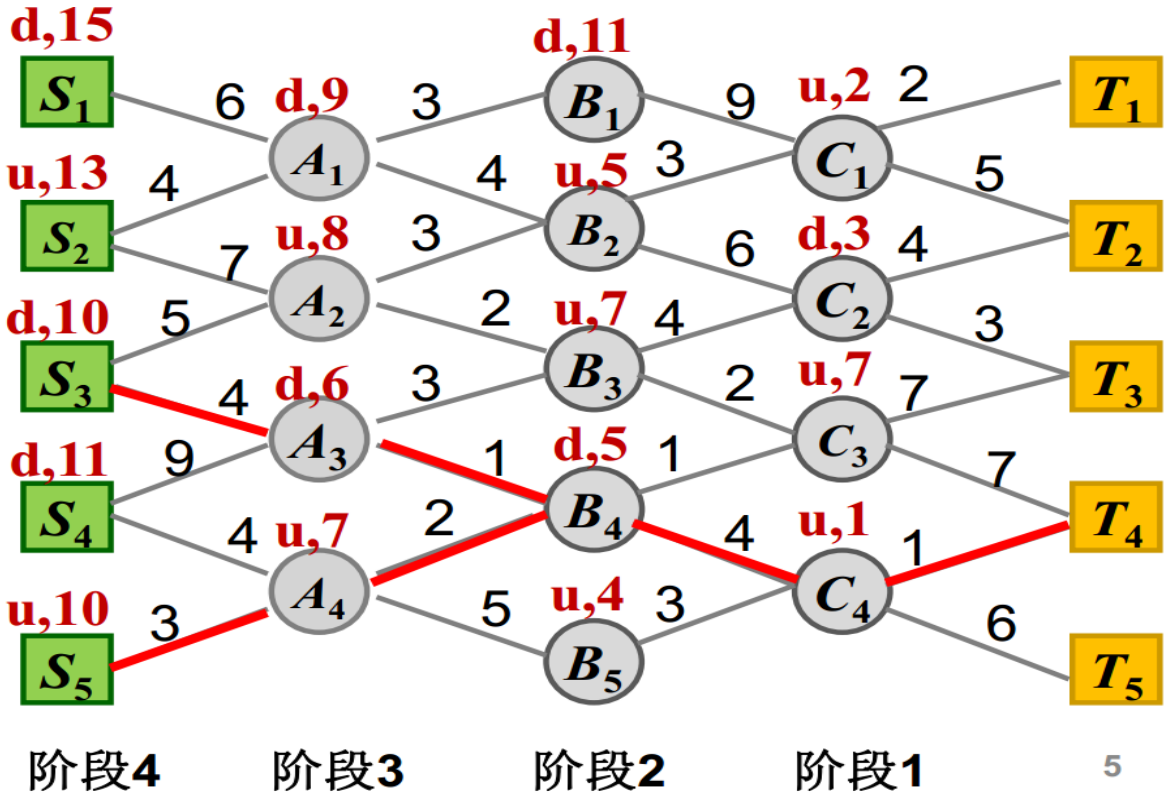
$$F(B_k) = \min_l \{B_k C_l + F(C_l)\}$$

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

$$F(S_i) = \min_i \{S_i A_j + F(A_j)\}$$

4个步骤分解

- 以自底向上方式计算最优值；



4个步骤分解

- 根据计算最优值时得到的信息，构造最优解。

$$F(S_i) = \min_i \{S_i A_j + F(A_j)\}$$

时间复杂度

- 每次处理一层
- 每层只需要对比该层的边数（出边）
- 总体复杂度为 O(|E|)

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。

例2 - 矩阵连乘问题

给定n个矩阵{A_1， A_2，…, A_n},其中A_i与A_{i+1}是可乘的。考察这N个矩阵的连乘积 $A_1A_2...A_n$

分析：

由于矩阵乘法满足结合律，故计算矩连乘积 $A_1A_2...A_n$ 可以有多个计算次序。

例如：
- $(A_1(A_2(A_3A_4)))$ 等等
- $(A_1((A_2A_3)A_4))$
- $((A_1A_2)(A_3A_4))$

先考虑两个矩阵乘积的计算量

设A为ra×ca矩阵 B为rb×cb矩阵，

```
void  matrixMultiply(int **a, int **b,  int **c,  int ra,  int ca,  int
rb, int cb )
{   if(ca!=rb) error(“Can’ t multiply”);
    for(i=0; i<ra; ++i)
      for(j=0; j<cb; ++j)
        {    c[i][j] =0;
             for(k=0;k<ca; ++k)  c[i][j]+=a[i][k]*b[k][j];
 }
}
```

复杂度：O(ra × rb × cb)

不同计算次序会导致不同的计算量

例如 $\{A_1, A_2, A_3\}$

A_1 : [10*100]

A_2 : [100*5]

A_3 : [5*50]

- 第1种加括号

  $((A_1A_2)A_3)$

计算量：10*100*5+10*5*50 =7500

- 第2种加括号

  $(A_1(A_2A_3))$

计算量：100*5*50+10*100*50 =75000

所谓矩阵连乘问题：

对于给定n个矩阵{A_1， A_2，…, A_n}，其中A_i的维数是$p_{i-1} \times p_i$

如何确定计算矩阵连乘积$A_1A_2...A_n$的计算次序，使得计算矩阵连乘积的数乘次数最少。

矩阵连乘的递归求解方法

记 A[i:j]为 $A_iA_{i+1}...A_j$ ，考察计算A[1:n]的最优计算次序。

不妨设

计算A[1:n]最优次序的最后一次乘积位置在 K 处

$$(\ (A_1A_2...A_k)\ (A_{k+1}...A_n)\ )\quad k=1,2,...n-1$$

其计算量为:

(1)计算 A[1:k]

(2)计算 A[k+1:n]

(3)最后一次矩阵乘积$p_0 \times p_k \times p_n$

1. 分析最优解的结构

计算A[1:n]的最优次序所包含的子链计算

A[1:k] 和 A[k+1:n]也一定是最优次序的。

事实上

若有一个计算A[1:k]的次序所需的计算量更少，则取代之。类似，计算A[k+1:n]的次序也一定是最优的。

因此，矩阵连乘计算次序问题的最优解，包含了其子问题的最优解。

2. 递归定义求最优值的公式

- 设计算A[i : j]所需的最少数乘次数为m[i][j], 则原问题的最优值为m[1][n]。
- 不妨设

计算A[i:j]最优次序的最后一次乘积位置在 K 处

$(\ (A_iA_2\ldots A_k)\ (A_{k+1}\ldots A_j))$ k=i,2,...j-1

则

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

2. 递归定义求最优值的公式

计算最优值：

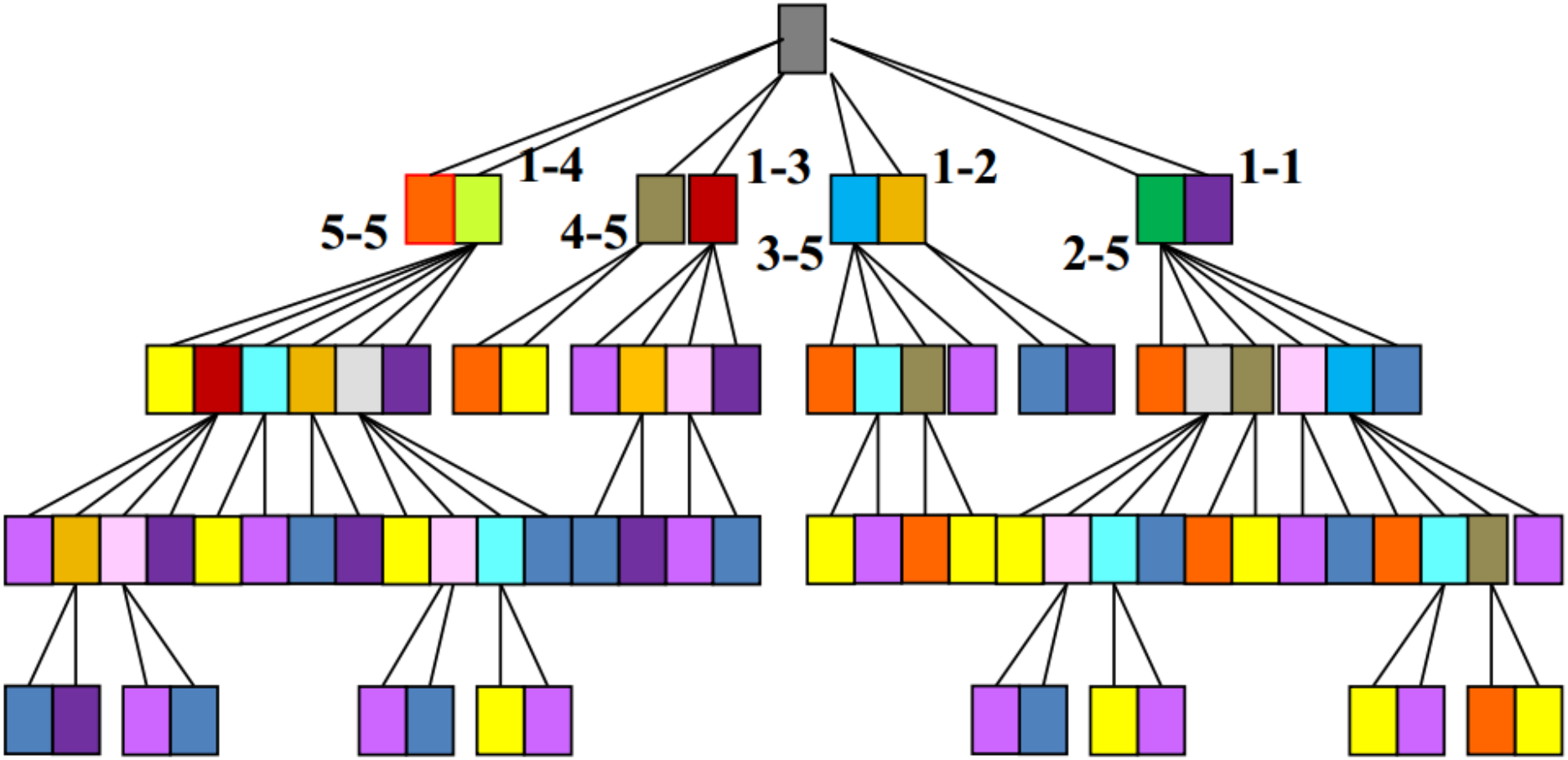
对于1≤i≤j≤n不同的有序对(i,j)对应于不同的子问题。因此，不同子问题的个数最多只有

$$\binom{n}{2}+n=\Theta(n^2)$$

在递归计算时，许多子问题被重复计算多次。这也是该问题可用动态规划算法求解的又一显著特征。

用动态规划算法解此问题，可依据其递归式以自底向上的方式进行计算。在计算过程中，保存已解决的子问题答案。每个子问题只计算一次，而在后面需要时只要简单查一下，从而避免大量的重复计算，最终得到多项式时间的算法。

2. 递归定义求最优值的公式



2. 递归定义求最优值的公式

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

3. 以自底向上方式计算最优值

动态规划实现的关键：

- 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

3. 以自底向上方式计算最优值

矩阵连乘的不同子问题：

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

3. 以自底向上方式计算最优值

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

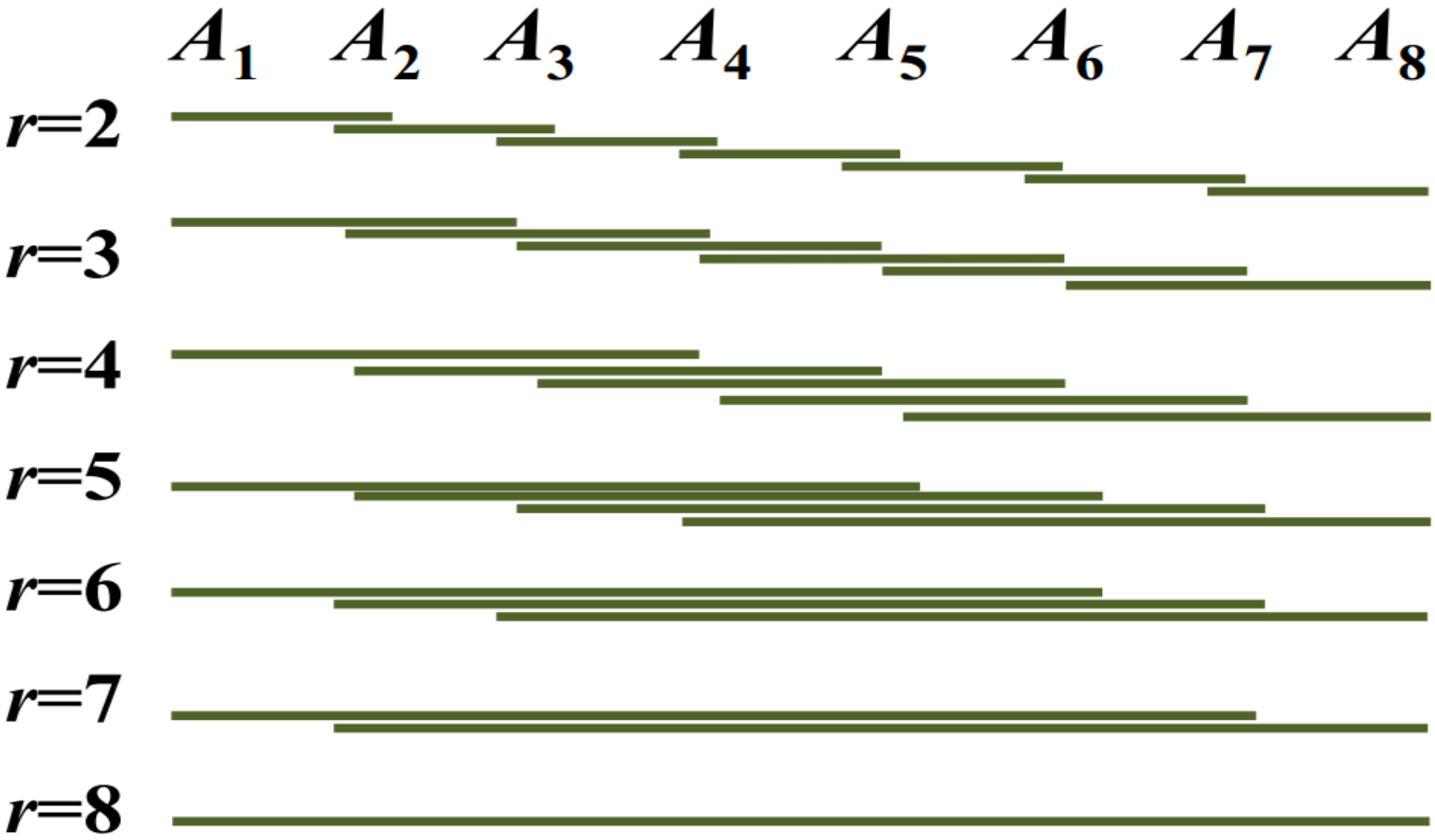
长度为3： 1..3, 2..4, 3..5, … , n-2..n

…

长度为n-1： 1..n-1, 2..n

长度为n： 1..n

3. 以自底向上方式计算最优值



3. 以自底向上方式计算最优值

```
void   MatricChain(int *p,  int n,  int **m,  int **s)
{    for(i=1;i<=n; ++i)  m[i][i]=0; //单个矩阵无计算
     for(r=2; r<=n; ++r)    //连乘矩阵的个数
        for(i=1; i<n-r; ++i)
        {    j=i+r-1;
             m[i][j]=m[i][i]+m[i+1][j]+p[i-1]*p[i]*p[j];
             s[i][j]=i;
             for(k=i+1; k<j; ++k)
            {   t=m[i][k]+m[k+1][j]+p[i-1]*p[k]*p[j];
                if(t<m[i][j])  { m[i][j]=t;   s[i][j]=k;
}     }   }
```

算法复杂度分析：
算法matrixChain的主要计算量取决于算法中对r，i和k的3重循环。循环体内的计算量为O(1)，而3重循环的总次数为$O(n^3)$。因此算法的计算时间上界为$O(n^3)$。算法所占用的空间显然为$O(n^2)$。

3. 以自底向上方式计算最优值

实例分析：

输入： $P = < 30, 35, 15, 5, 10, 20 >, n = 5$

矩阵链： $A1\ A2\ A3\ A4\ A5$，其中

$A1: 30\times35$， $A2: 35\times15$， $A3: 15\times5$，

$A4: 5\times10$， $A5: 10\times20$

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。

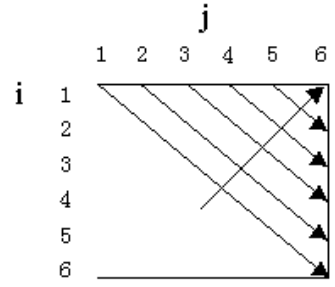
3. 以自底向上方式计算最优值

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+35×15×20 2625+1000+35×5×20

4375+0+35×10×20 = 7125



(a) 计算次序

j

i	1	2	3	4	5	6
1	0	15750	7875	9375	11875	15125
2		0	2625	4375	7125	10500
3			0	750	2500	5375
4				0	1000	3500
5					0	5000
6						0

(b) m[i][j]

j

i	1	2	3	4	5	6
1	0	1	1	3	3	3
2		0	2	3	3	3
3			0	3	3	3
4				0	4	5
5					0	5
6						0

(c) s[i][j]

3. 以自底向上方式计算最优值

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：　$s[1,5]=3 \Rightarrow (A1A2A3)(A4A5)$

$s[1,3]=1 \Rightarrow A1(A2A3)$

输出

计算顺序：　$(A1(A2A3))(A4A5)$

最少的乘法次数：　$m[1,5]=11875$

4. 构造最优解

```
void  Traceback( int i,  int j,  int **s)
{
      if (i==j)  return;
      k=s[i][j];
      Traceback(i, k, s);
      Traceback(k+1, j, s);
      printf("A[%d:%d] *A[%d:%d ] \n",  i, k, k+1, j);
}
```