EDU

CSDN学院 IT实战派

数据结构和算法

常见经典算法

讲师：Samuel

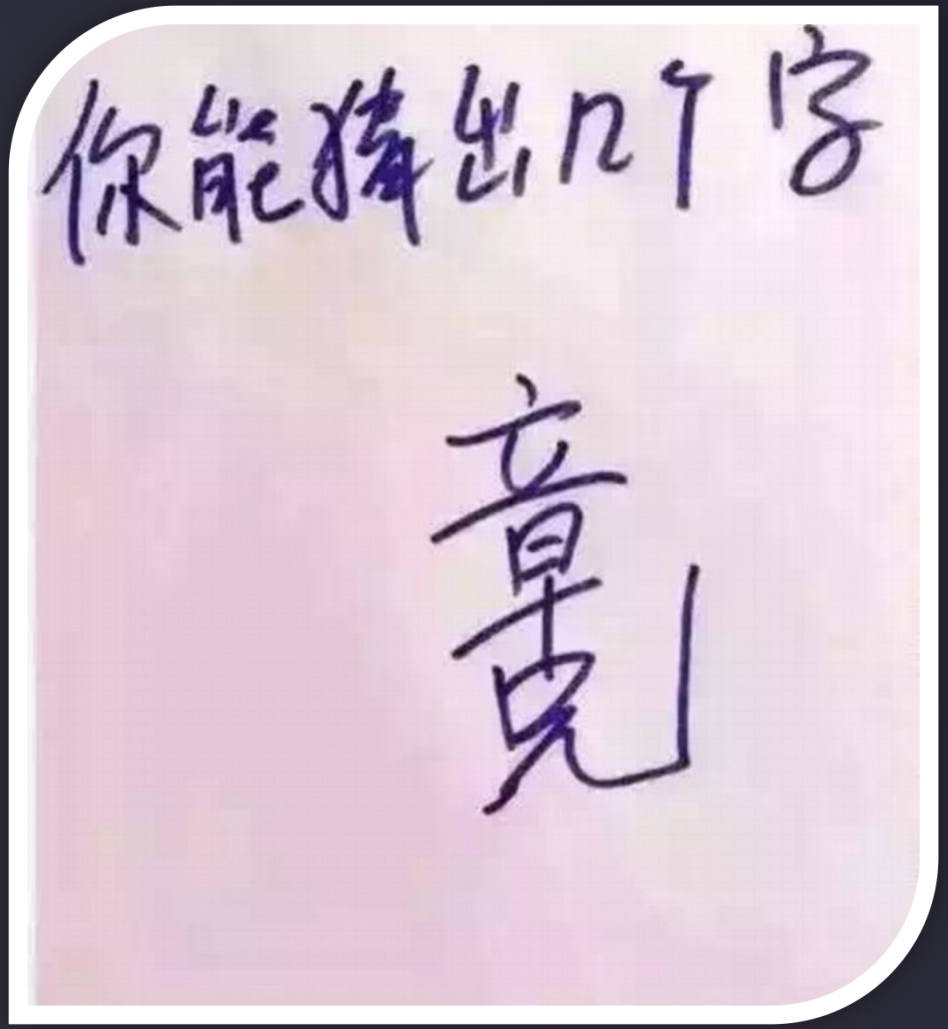
本章概述

1 什么是算法

- 算法（Algorithm）是指解题方案的准确而完整的描述，是一系列解决问题的清晰指令
- 算法代表着用系统的方法描述解决问题的策略机制
- 也就是说，能够对一定规范的输入，在有限时间内获得所要求的输出
- 如果一个算法有缺陷，或不适合于某个问题，执行这个算法将不会解决这个问题
- 不同的算法可能用不同的时间、空间或效率来完成同样的任务
- 一个算法的优劣可以用空间复杂度与时间复杂度来衡量

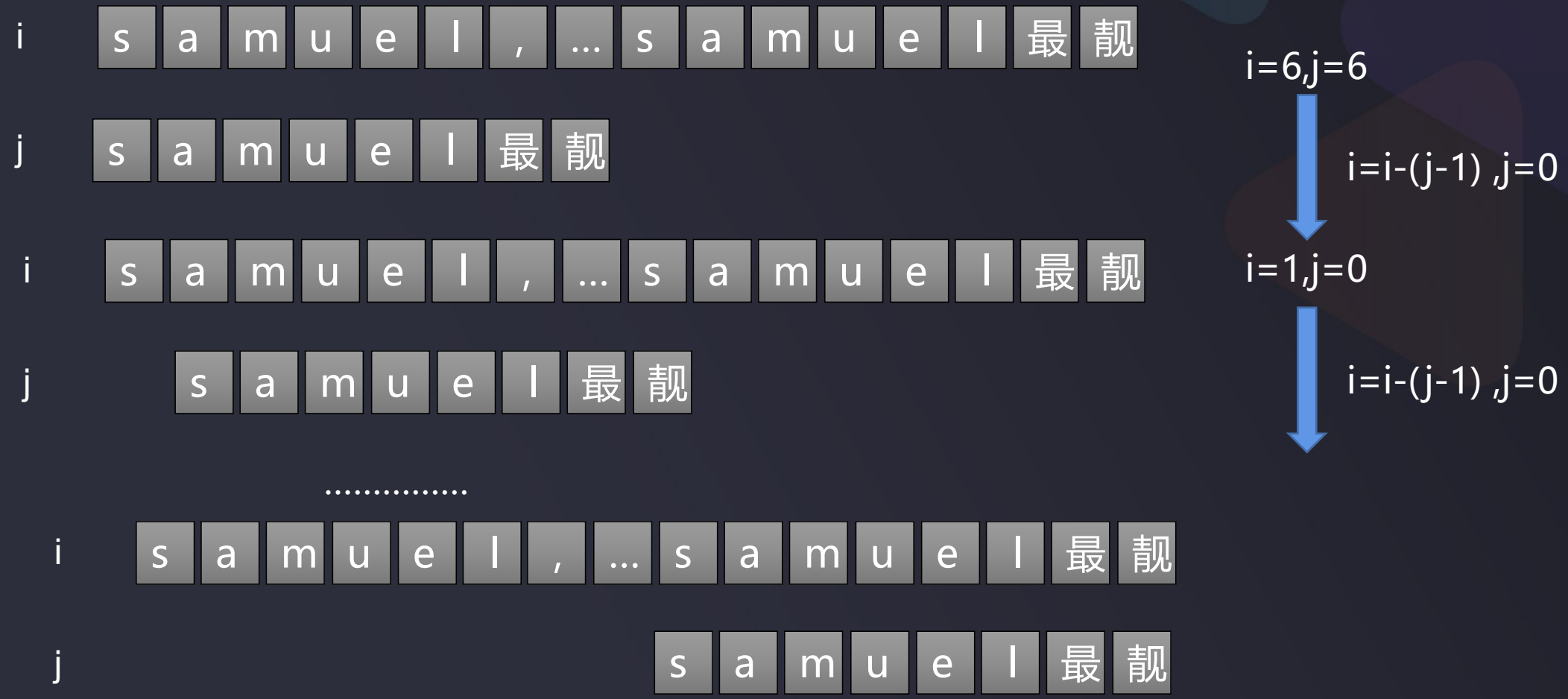
本章概述

2 算法无处不在



本章概述

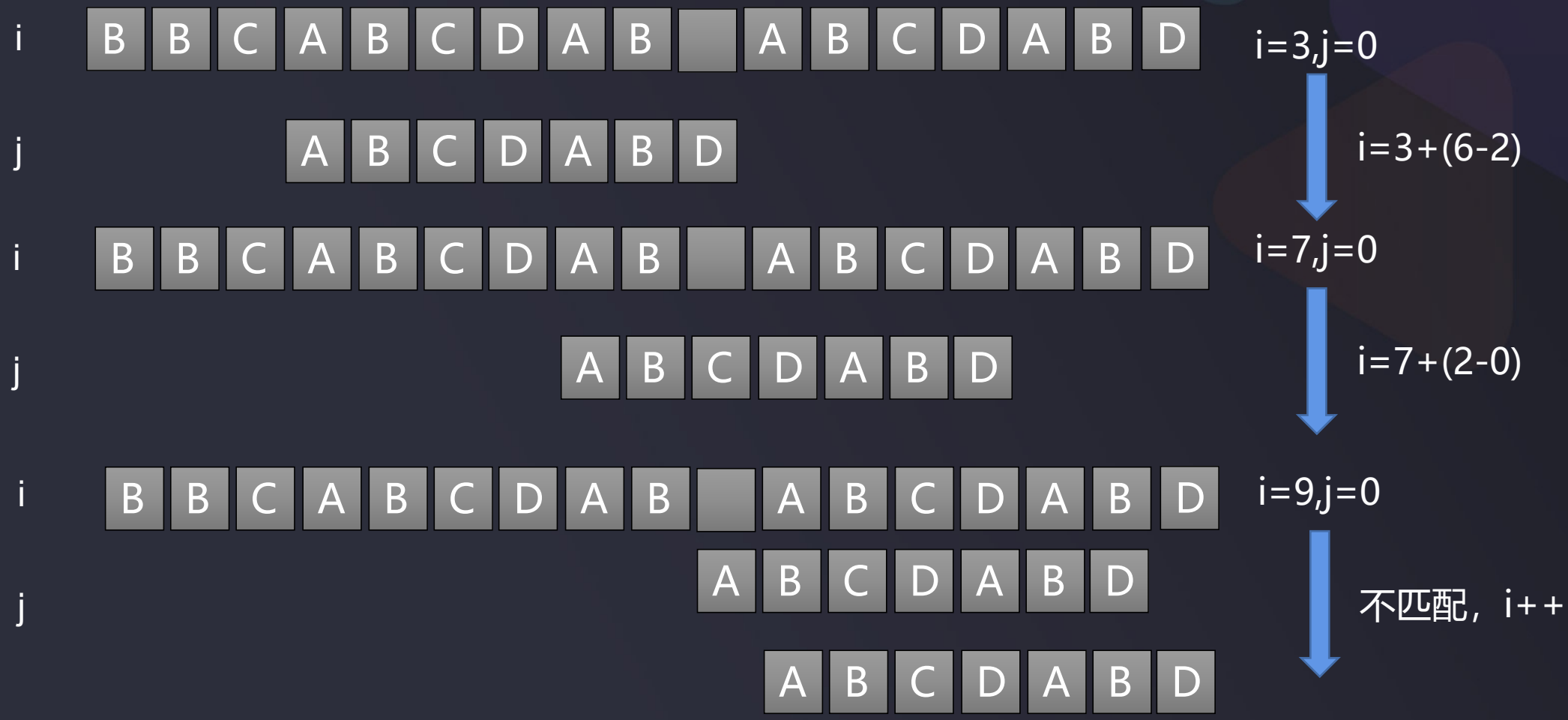
3 字符串暴力匹配



本章概述

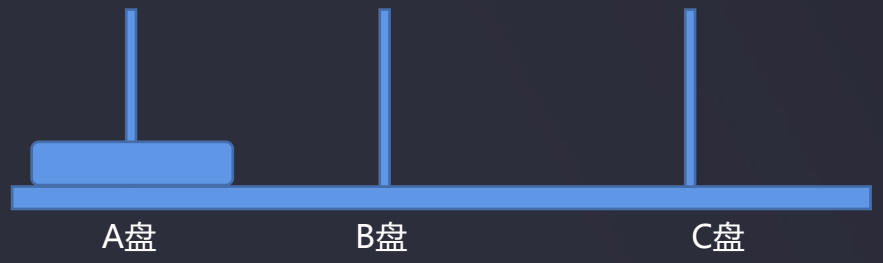
4 KMP算法

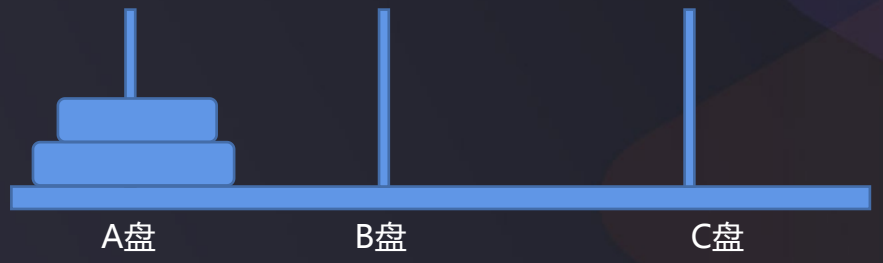
搜索词	A	B	C	D	A	B	D
部分匹配值	0	0	0	0	1	2	0

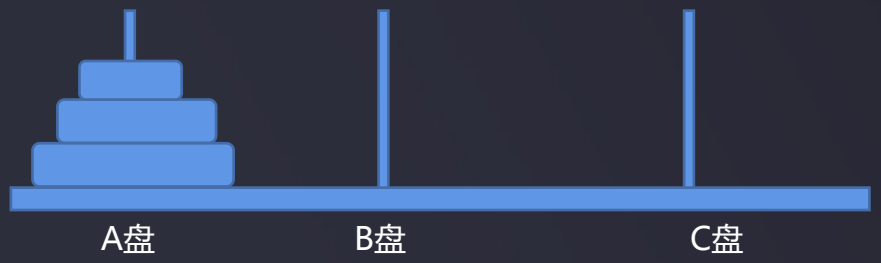


本章概述

5 汉诺塔







Content

分治算法

1 什么是分治算法（divide-and-conquer）

- 分治法就是把一个复杂的问题分成两个或更多的相同或相似的子问题
- 再把子问题分成更小的子问题……直到最后子问题可以简单的直接求解，原问题的解即子问题的解的合并
- 这个技巧是很多高效算法的基础，如排序算法，傅立叶变换等

2 分治算法实现步骤

- 分解：将原问题分解为若干个规模较小，相互独立，与原问题形式相同的子问题
- 解决：若子问题规模较小而容易被解决则直接解，否则递归地解各个子问题
- 合并：将各个子问题的解合并为原问题的解

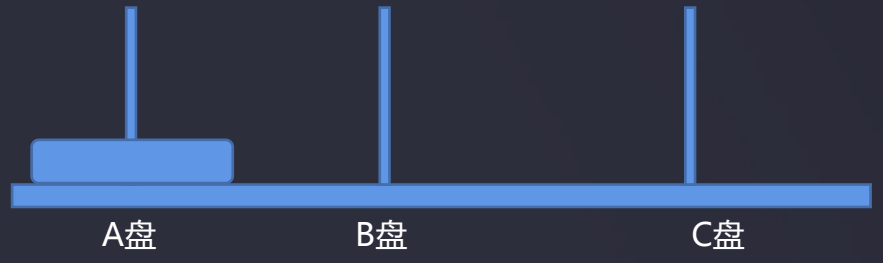
分治算法

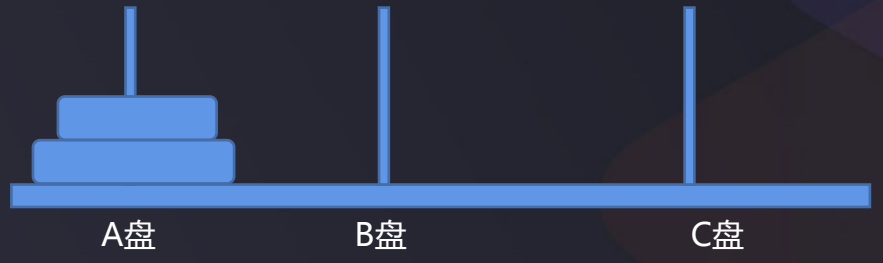
3 经典应用

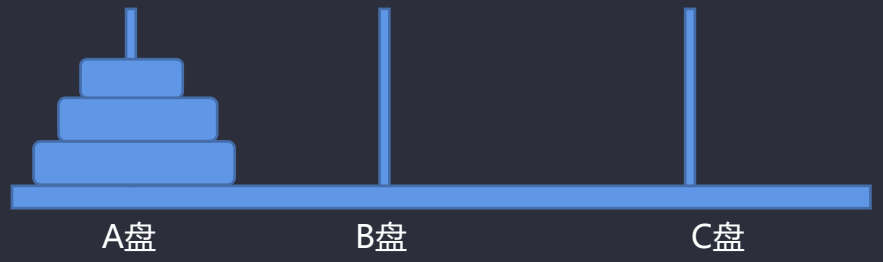
- 二分搜索
- 大整数乘法
- 棋盘覆盖
- 合并排序
- 快速排序
- 线性时间选择
- 最接近点对问题
- 循环赛日程表
- 汉诺塔

分治算法

4 汉诺塔问题







- 如果是有一个盘， A->C
- 如果我们有 n >= 2 情况，我们总是可以看做是两个盘

  1号盘是最下边的盘 2号盘是其他的上面所有的盘
  - 先把 最上面的盘 A->B
  - 把最下边的盘 A->C
  - 把B塔的所有盘 从 B->C
- http://www.hannuota.cn/

动态规划算法

1 动态规划算法(Dynamic Programming)

- 核心思想：动态规划算法与分治算法类似，将大问题划分为小问题进行解决，先求解子问题，然后从这些子问题的解得到原问题的解
- 与分治法不同的是，适合于用动态规划求解的问题，经分解得到子问题往往不是互相独立的
- 动态规划对于解决多阶段决策问题的效果是明显的，但是动态规划也有一定的局限性
  - 它没有统一的处理方法，必须根据问题的各种性质并结合一定的技巧来处理
  - 当变量的维数增大时，总的计算量及存贮量急剧增大
  - 受存贮量及计算速度的限制，用动态规划方法来解决较大规模的问题，这就是维数障碍

动态规划算法

2 背包问题

- 背包问题是动态规划问题的一个最佳实践
- 有一个背包，容量是C(capacity)，现有n种不同的物品，编号为0…n-1,其中每一件物品重量是w(i),价值为v(i)
- 问可以向这个背包中盛放哪些物品，使得在不超过背包容量的基础上，物品的总价值最大

- F(n,C)考虑将n个物品放到容量为C的背包，使得价值最大
- F(i,c) = F(i-1,c)

  = v(i)+F(i-1,c-w(i))
- F(i,c) = max(F(i-1,c),v(i)+F(i-1,c-w(i))

动态规划算法

3 背包问题

物品	重量	价格
吉他(1号)	1	6
音响(2号)	2	10
电脑(3号)	3	12

5

	0	1	2	3	4	5
0	0	0	0	0	0	0
1(只可选1号)	0	6	6	6	6	6
2(可选1号2号)	0	6	10	16	16	16
3(123都可选)	0	6	10	16	18	22

动态规划算法

4 背包问题

- F(n,C)函数 将n个物品放到容量为C的背包，使得价值最大
- F(i,c) = F(i-1,c)

  = v(i)+F(i-1,c-w(i))
- F(i,c) = max(F(i-1,c),v(i)+F(i-1,c-w(i))

- 对应到二维数组中，对于给定的n个物品，v[i]、w[i]分别为第i个物品的价值和重量

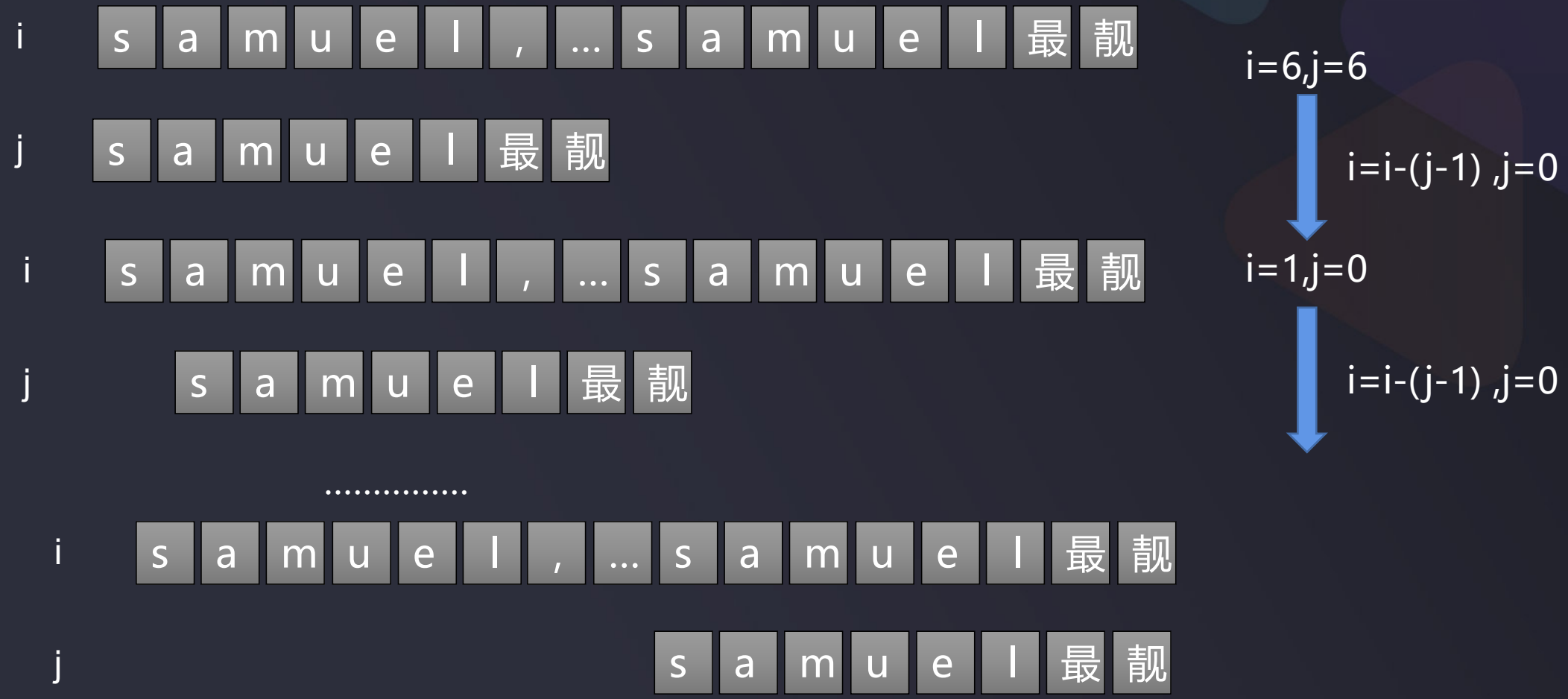
  v[i][j]表示在前i个物品中能够装入容量为j的背包中的最大价值
  - v[i][0]=v[0][j]=0
  - 当w[i]> j 时：v[i][j]=v[i-1][j]
  - 当j>=w[i]时： v[i][j]=max{v[i-1][j], v[i]+v[i-1][j-w[i]]}

字符串匹配

1. 字符串匹配问题
   - 有一个字符串一str1=“samuel，这条街最靓的崽，samuel最靓！”
   - 还有一个子串二str2=“samuel最靓”
   - 判断str1是否包含str2，如果存在，就返回第一次出现的位置，如果不存在就返回-1
   - 解决该问题可以有两种比较常见的解决方案
     - 暴力匹配
     - KMP算法

字符串匹配

2 字符串暴力匹配



KMP算法

1 KMP算法

- Knuth-Morris-Pratt 字符串查找算法，简称为 “KMP算法”
- 对比暴力匹配算法来说，不同地是每次向后移动的位数不再是只后移1位
- KMP算法中i的移动位数 = 已匹配的字符数 - 对应的部分匹配值
- 计算部分匹配值，需要先了解字符串的前缀和后缀的概念
  - 前缀：字符串bread的前缀有 b、br、bre、brea
  - 后缀：字符串bread的后缀有read、ead、ad、d

KMP算法

2 部分匹配表

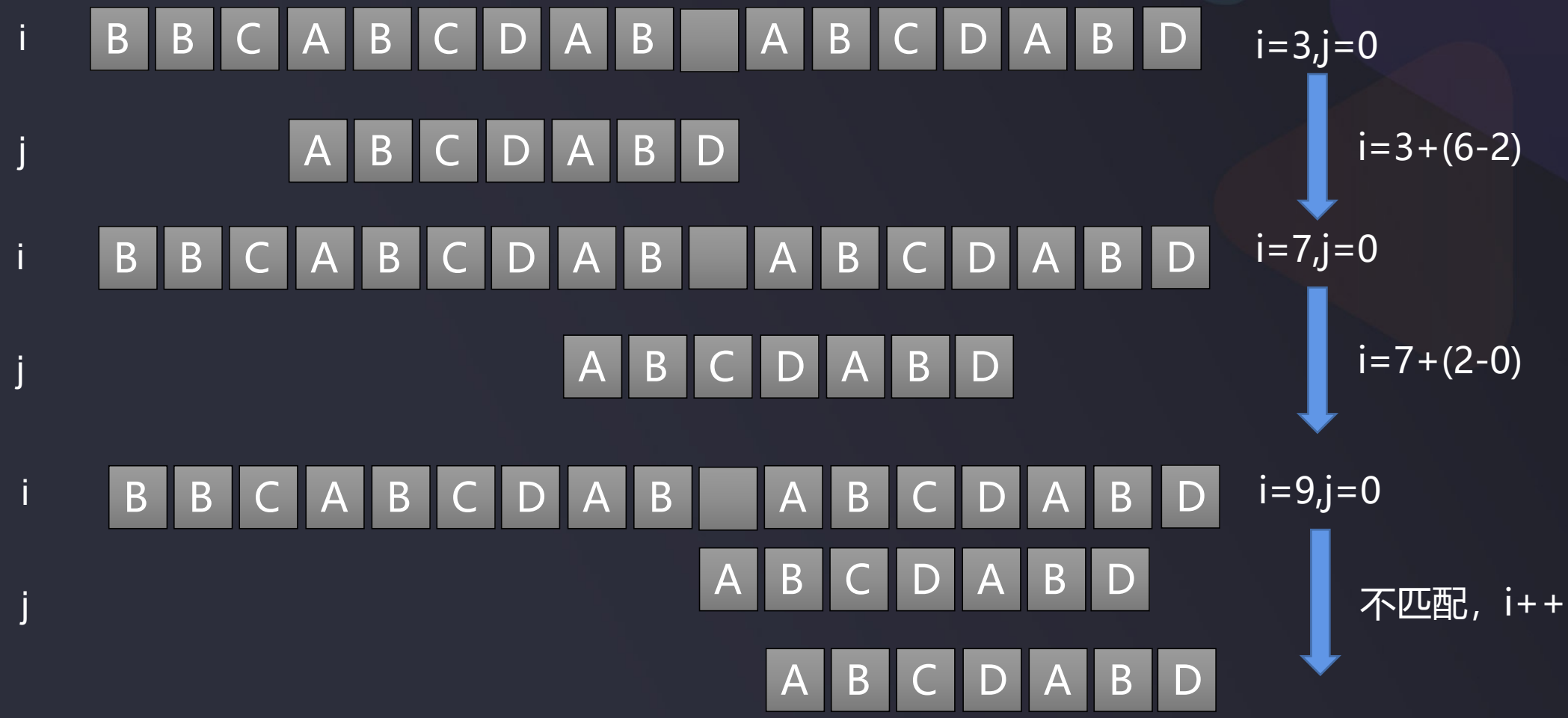
- 部分匹配值就是”前缀”和”后缀”的最长的共有元素的长度
- 字符串中的每个字符的部分匹配值最终得到一个部分匹配表，比如“ABCDABD”

	前缀	后缀	共同	值
A	无	无	无	0
AB	A	B	无	0
ABC	A、AB	BC、C	无	0
ABCD	A, AB, ABC	BCD, CD, D	无	0
ABCDA	A, AB, ABC, ABCD	BCDA, CDA, DA, A	A	1
ABCDAB	A, AB, ABC, ABCD, ABCDA	BCDAB, CDAB, DAB, AB, B	AB	2
ABCDABD	A, AB, ABC, ABCD, ABCDA, ABCDAB	BCDABD, CDABD, DABD, ABD, BD, D	无	0

KMP算法

3 KMP实现

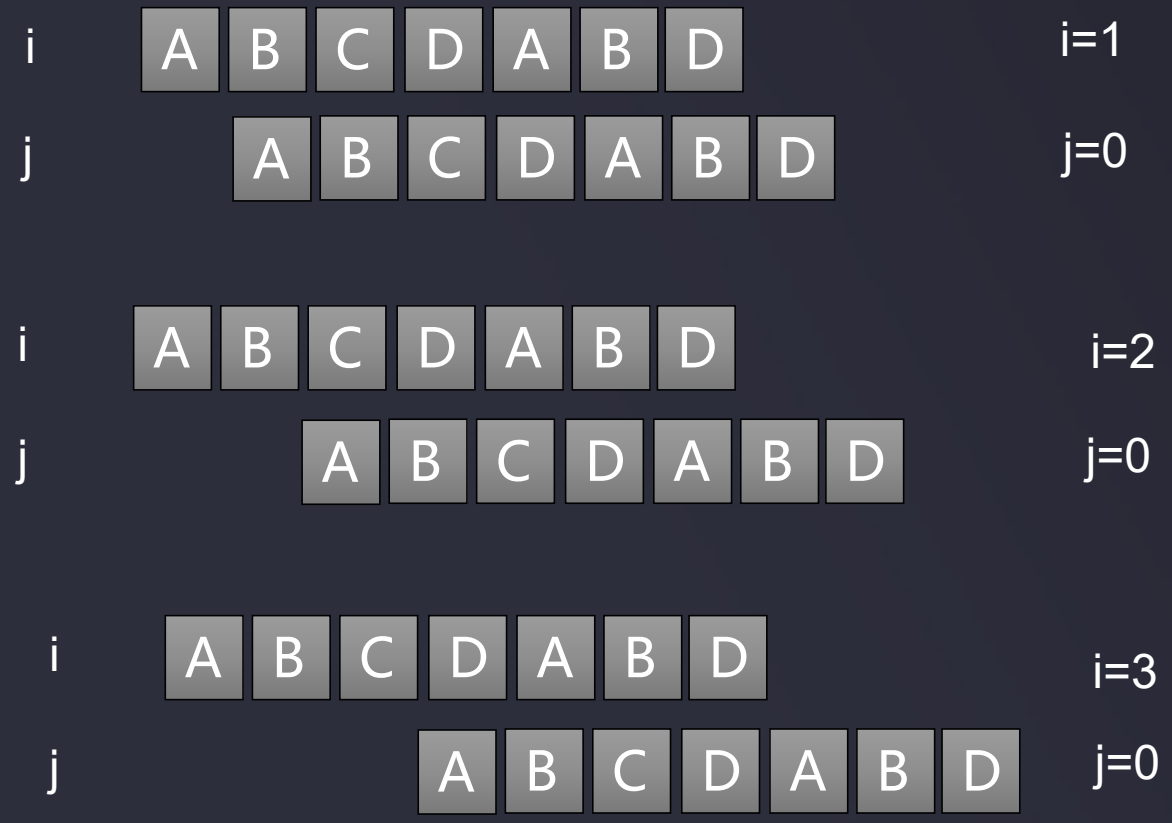
搜索词	A	B	C	D	A	B	D
部分匹配值	0	0	0	0	1	2	0



KMP算法

4 部分匹配值的实现

搜索词	A	B	C	D	A	B	D
部分匹配值	0	0	0	0			



str[i]!=str[j]
next[i]=j
i++

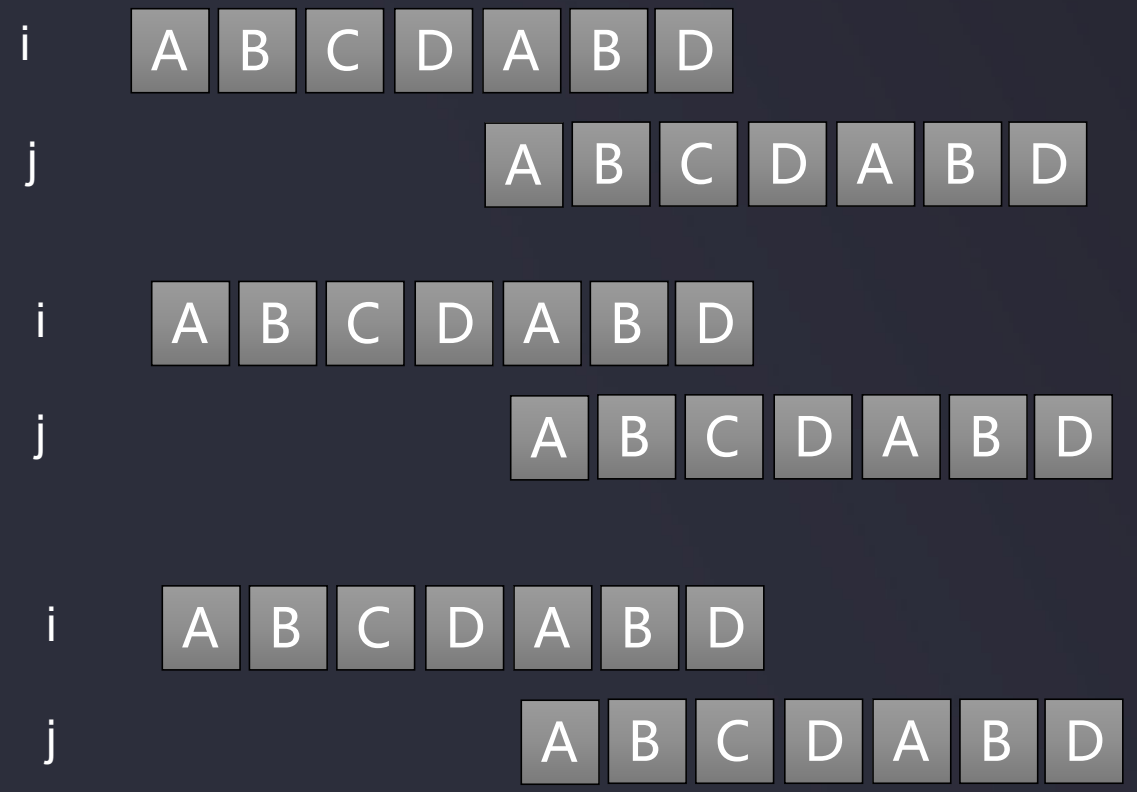
str[i]!=str[j]
next[i]=j
i++

str[i]!=str[j]
next[i]=j
i++

KMP算法

4 部分匹配值的实现

搜索词	A	B	C	D	A	B	D
部分匹配值	0	0	0	0	1	2	0



str[i]==str[j],j++
next[i]=j
i++

str[i]==str[j],j++
next[i]=j
i++

str[i]!=str[j], j=next[j-1]
next[i]=j
i++

贪心算法

1 贪心算法

- 贪心算法是指在对问题进行求解时，在每一步选择中都采取最好或者最优的选择，从而希望能够导致结果是最好或者最优的算法
- 贪心算法所得到的结果不一定是最优的结果，但是都是相对近似最优解的结果
- 贪心算法的代码实现比较简单，难点在如何确定一个问题是否可以使用贪心算法解决

贪心算法

2 电台覆盖问题

- 贪心算法其中一个最佳实践为集合覆盖问题
- 假设存在下面需要付费的广播台，以及广播台信号可以覆盖的地区
- 如何选择最少的广播台，让所有的地区都可以接收到信号

广播台	覆盖地区
K1	"北京", "上海", "天津"
K2	"广州", "北京", "深圳"
K3	"成都", "上海", "杭州"
K4	"上海", "天津"
K5	"杭州", "大连"

贪心算法

3 电台覆盖问题

maxkey

广播台	覆盖地区		
K1	北京	上海	天津
K2	广州	北京	深圳
K3	成都	上海	杭州
K4	上海	天津	
K5	杭州	大连	

all

北京 上海 天津 广州
深圳 成都 杭州 大连

select

K1 北京 上海 天津
K2 广州 北京 深圳
K3 成都 上海 杭州
K5 杭州 大连

EDU

CSDN学院 IT实战派

下节课再见，记得关注公众号