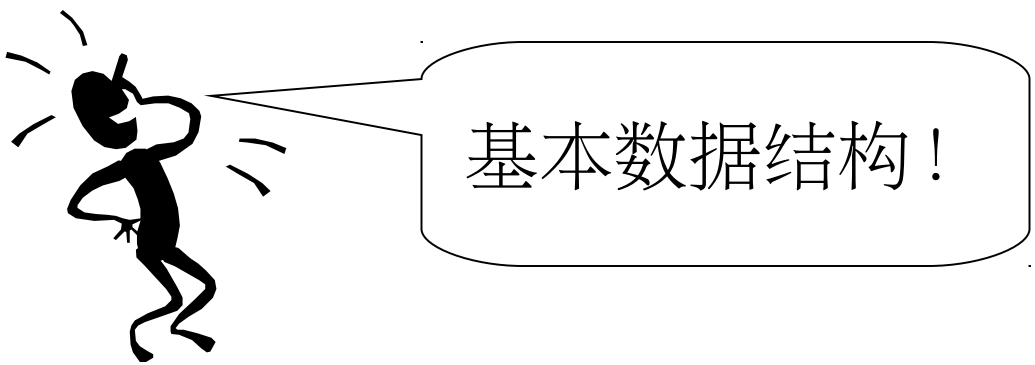
基本数据结构在信息学竞赛中的应用

安徽省芜湖市第一中学
朱晨光

引言

- 本篇论文将介绍几种基本数据结构在信息学竞赛中的应用，并通过几道例题集中体现这些数据结构的重要作用。
  - 题目难度→
  - 编程复杂度
  - 容易查错



第一部分——基本数据结构的介绍

- 一、线性表（线性表的顺序存储结构）

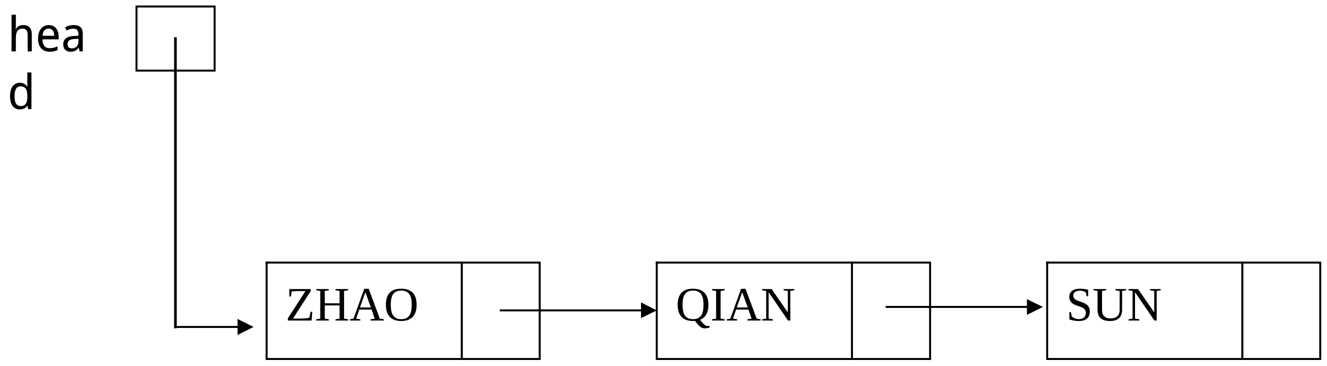
存储地址	内存状态	元素在线性表中的序号
b	a_1	1
b+L	a_2	2
……	……	……
b+(i-1)L	a_i	i
……	……	……
b+(n-1)L	a_n	n
b+nL		
……		
b+(maxlen-1)L		

空闲

线性表

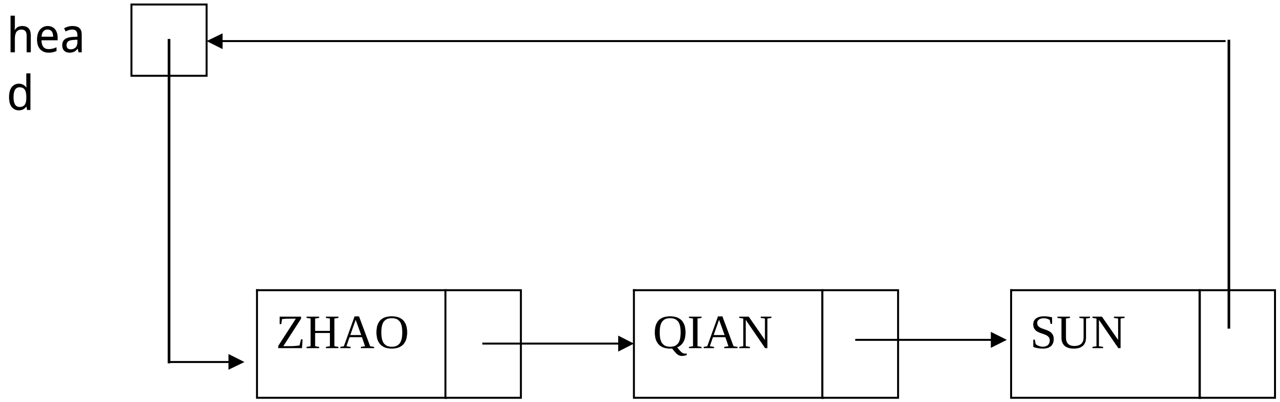
- 二、线性表的链式存储结构

线性链表：



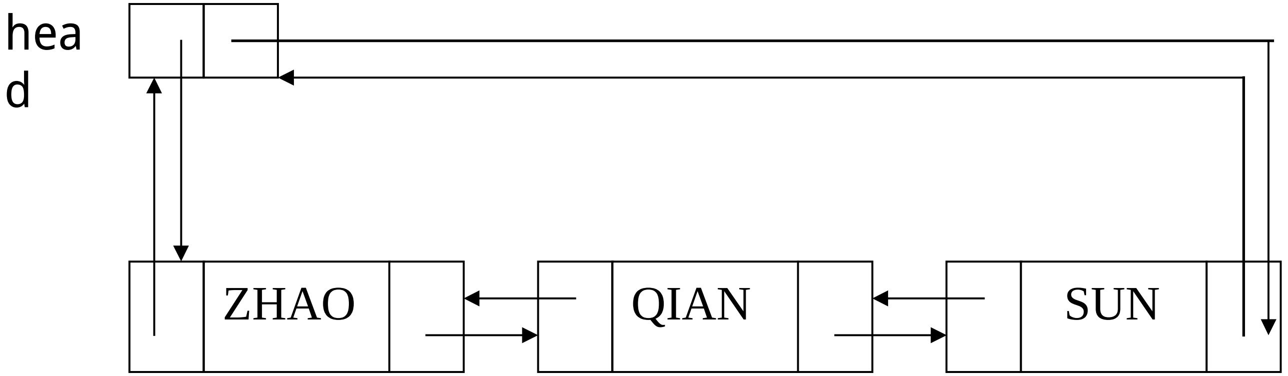
线性表的链式存储结构

循环链表:

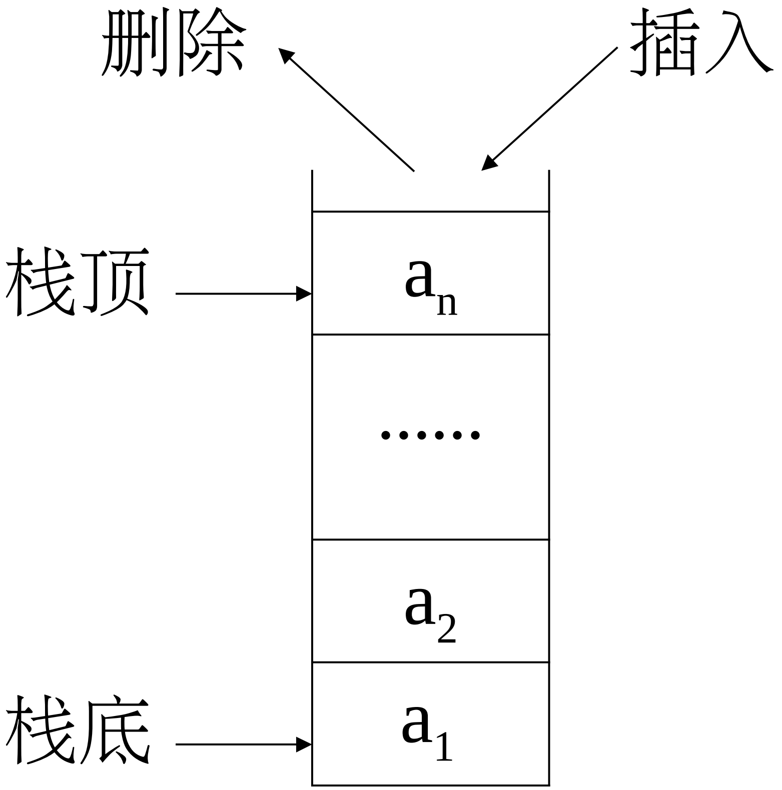


线性表的链式存储结构

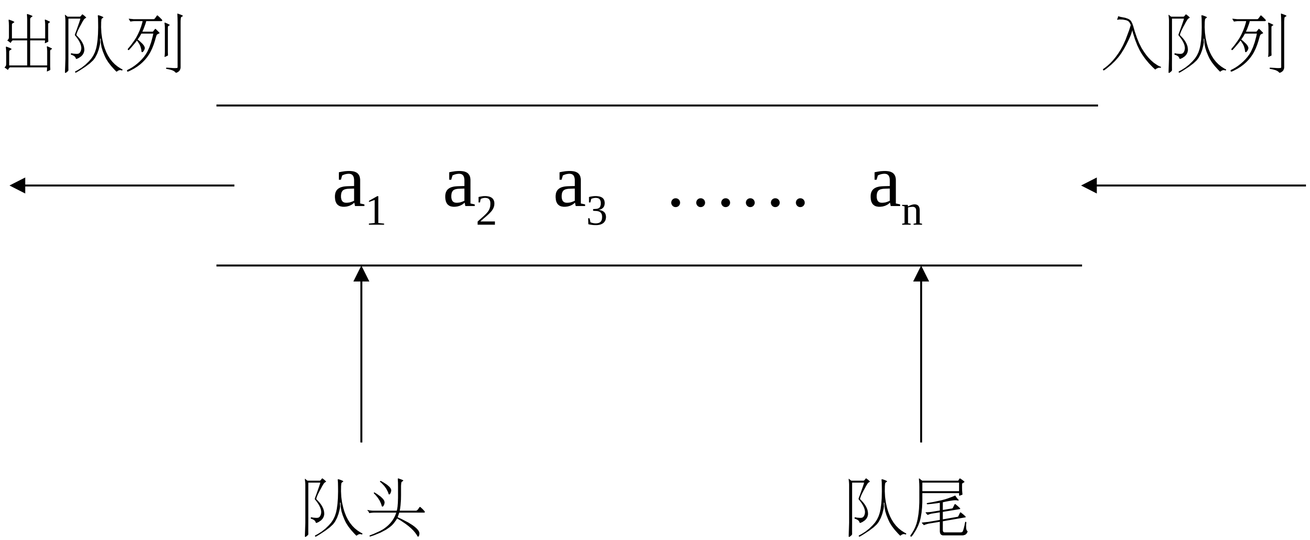
双向链表:



栈



队列



第二部分——基本数据结构的应用

栈的应用 [例 1] 求 01 矩阵中最大的全零矩形

线性表的应用 [例 2] 营业额统计

队列的应用 [例 3] 瑰丽华尔兹

线性表的应用——营业额统计

- 给定 N （$1 \le N \le 32767$） 天的营业额 $a_1,a_2,\ldots\ldots,a_n$.
- 定义一天的最小波动值等于

  min{| 该天以前某一天的营业额 - 该天营业额 |}
- 特别地，第一天的最小波动值即为 a_1

  试求 N 天的最小波动值之和

- 例如： $N=3,a_1=9,a_2=3,a_3=8$ ，则各天最小波动值依次为 9， 6， 1， 和为 16

分析

这道题目的规模很大，如果简单地用两重循环解决，时间复杂度高达 $O(N^2)$，难以满足要求。

算法低效的原因在于没有高效地将数据组织起来，而是松散地存储在数组中，导致对于每一个营业额都需要检查前面所有的营业额。

分析

实际上，有一种高级数据结构——平衡树可以解决这个问题。我们可以在将一天的营业额插入平衡树的过程中得到该天的最小波动值。方法是求出所有在插入路径上的数字与改天营业额差的绝对值，从中取出最小值。这种算法的时间复杂度为 $O(N\log_2 N)$.

进一步改进

平衡树→左旋，右旋，双旋……

高效

高出错率 

那么，基本数据结构能否在这里得到应用呢?

Yes!

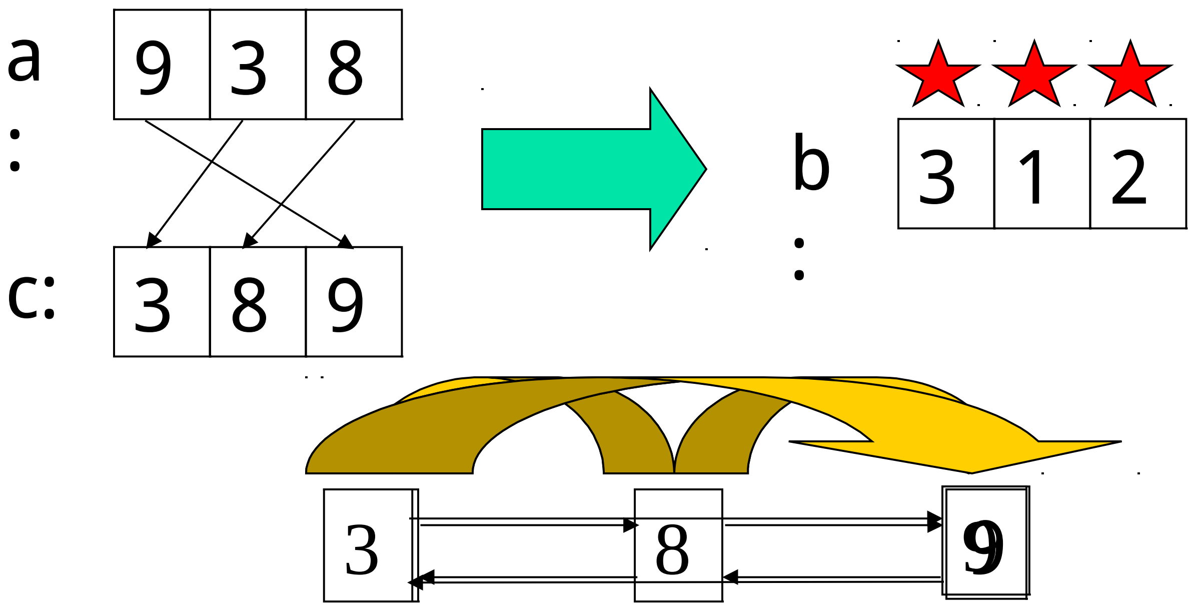
应用基本数据结构——双向链表

- 将这 N 个元素进行排序，得到序列 c，同时记录原来第 i 个元素在排序后的位置 b_i
- 将排序后的序列在静态数组中建成一个双向链表
- 按照从 a_n 到 a_1 的顺序依次处理每个元素

双向链表

- 对于 a_n， 查看 b_n 的前驱 pre[b_n] 与后继 next[b_n] 所指的数
- 最小波动值必然是 a_n 与这两个数中的某一个的差的绝对值
- 处理完 a_n 后，我们把它从双向链表中删除，接着处理 a_{n-1}

动画演示



最小波动值为 min{|3-9|,|8-9|}=min{6,1}=1

最小波动值为 |3-9|=6

最小波动值为 9

最小波动值总和为 1+6+9=16

时间复杂度分析

排序 $O(N\log_2 N)$

建表 $O(N)$

处理 $O(N)$

$$O(N\log_2 N)$$

小结

平衡树　　　　编程复杂度高

基本数据结构——双向链表

没有增加时间复杂度

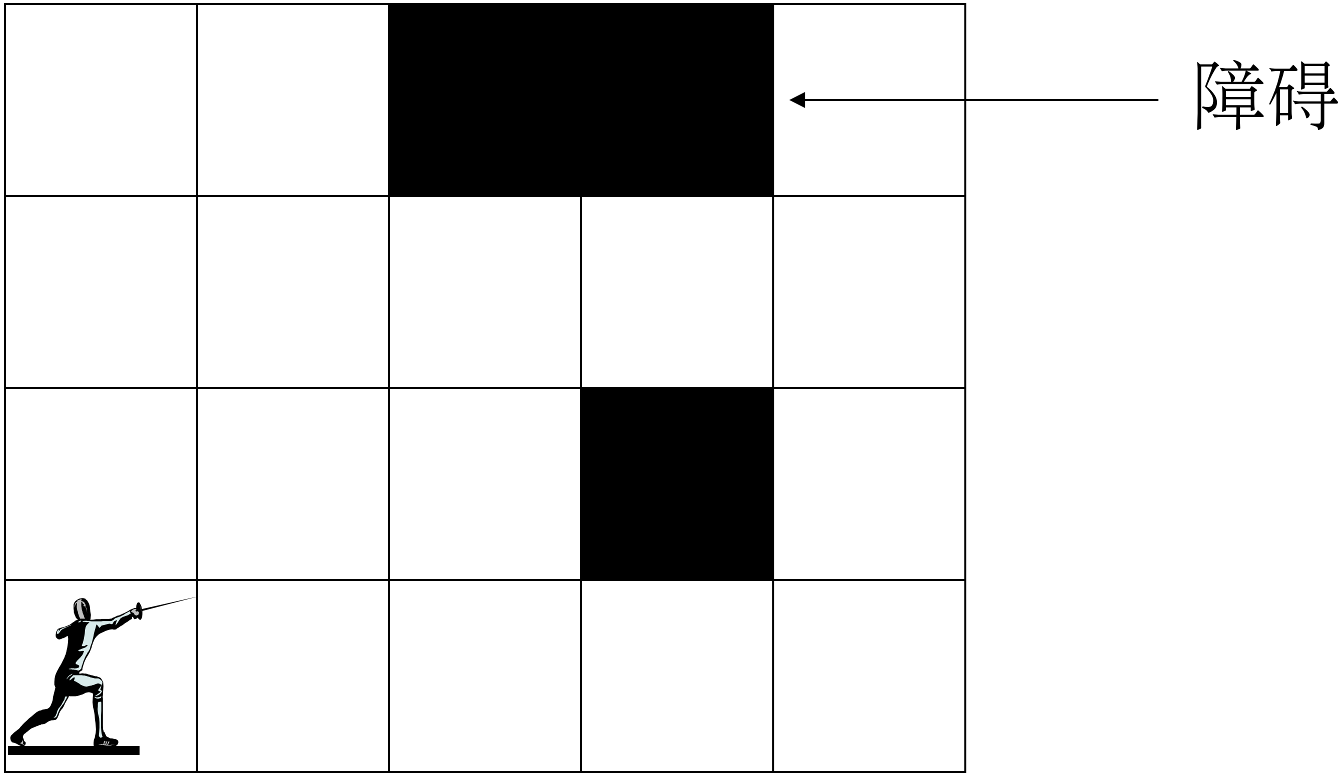
大大降低编程复杂度

思路清晰　　　　　　见解独到

队列的应用——瑰丽华尔兹

- 给定一个 N 行 M 列的矩阵，矩阵中的某些方格上有障碍物。有一个人从矩阵中的某个方格开始滑行。每次滑行都是向一个方向最多连续前进 c_i 格（也可以原地不动）。但是这个人在滑行中不能碰到障碍物。现按顺序给出 K 次滑行的方向（东、南、西、北中的一个）以及 c_i ，试求这个人能够滑行的最长距离（即格子数）。
- 数据范围：$1 \le N,M \le 200$， $K \le 200$ $\sum_{i=1}^{K} c_i \le 40000$

样例



分析

- 本题是一个求最值的问题。根据题目中 K 次滑行的有序性以及数据范围，可以很容易设计出这样一种动态规划算法:

令 f(k,x,y)= 此人 k 次滑行后到达 (x,y) 方格时已经滑行的最长距离。

动态规划的状态转移方程如下：（这里只给出向右滑行的转移方程）

f(0,startx,starty)=0

f(k,x,y)=max{f(k-1,x,y),f(k-1,x,y-1)+1,f(k-1,x,y-2)+2,……,f(k-1,x,y')+y-y'}

（其中 y' 为满足 y'=1 或 (x,y'-1) 上有障碍或 $y'=y-c_k$ 的最大值）

时间复杂度分析

现在来分析这个动态规划算法的时间复杂度:

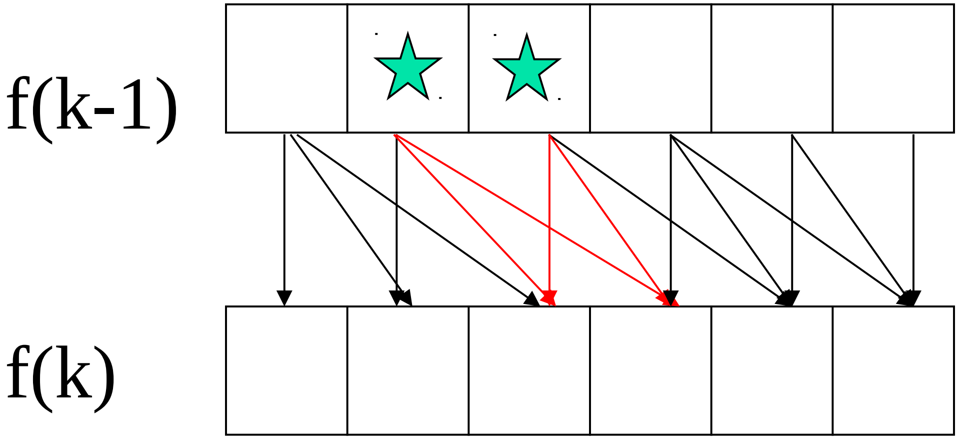
显然，状态总数为 O(KMN)，而每次状态转移在最坏情况下的时间复杂度为 O(max{M,N})，因此总的时间复杂度为 O(KMN*max{M,N})=O(1.6*10^9)，难以承受。

因此，我们需要对这个动态规划算法进行优化！

分析

令 $c_k=2$， $x=1$

y: 1 2 3 4 5 6



分析

对于一个具体的例子 $k=2$， $x=1$， $c_2=2$ 可以列出如下等式:

$f(2,1,1)=\max\{f(1,1,1)\}$

$f(2,1,2)=\max\{f(1,1,1)+1,f(1,1,2)\}$

$f(2,1,3)=\max\{f(1,1,1)+2,f(1,1,2)+1,f(1,1,3)\}$

$f(2,1,4)=\max\{f(1,1,2)+2,f(1,1,3)+1,f(1,1,4)\}$

……

分析

如果我们定义一个序列 a，使得 $a_i=f(1,1,i)-i+1$，则以上等式可以写成:

$f(2,1,1)=\max\{a_1\}=\max\{a_1\}$

$f(2,1,2)=\max\{a_1+1,a_2+1\}=\max\{a_1,a_2\}+1$

$f(2,1,3)=\max\{a_1+2,a_2+2,a_3+2\}=\max\{a_1,a_2,a_3\}+2$

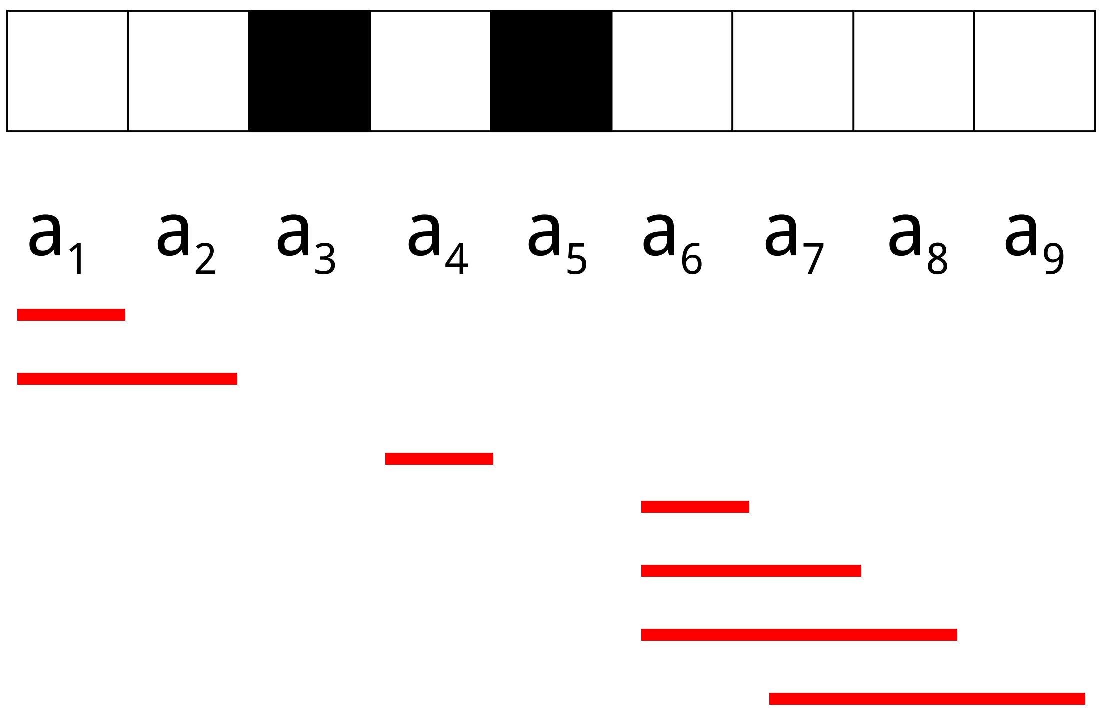
$f(2,1,4)=\max\{a_2+3,a_3+3,a_4+3\}$

$\quad\quad\quad\quad=\max\{a_2,a_3,a_4\}+3$

……

分析

现在，让我们加入对障碍物的考虑



分析

线段树——专门计算区间内数据的最值

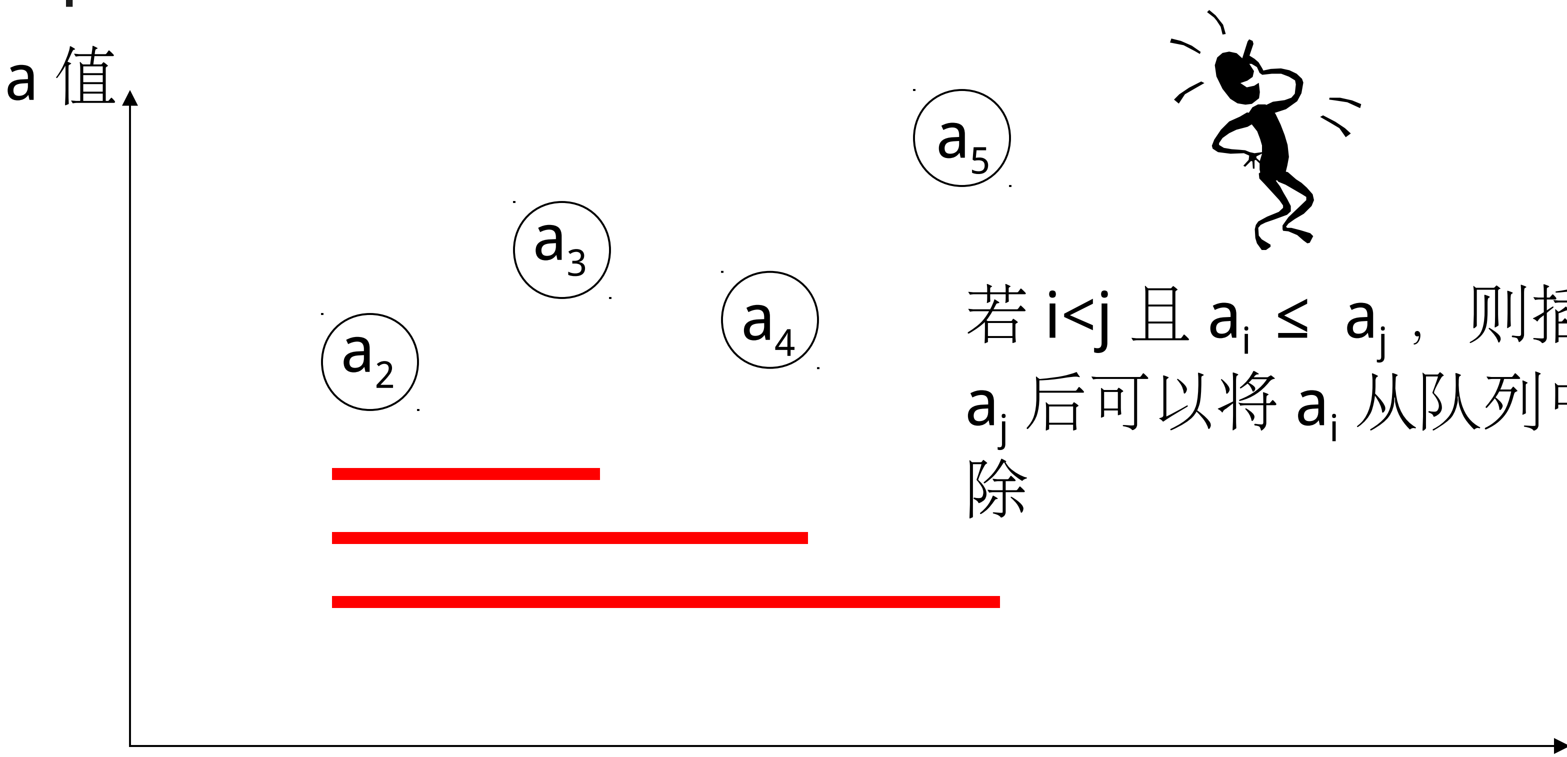
建立 O(M)

查找 $O(\log_2 M)$

算法时间复杂度 $O(KMN\log_2\{M,N\})$

编程复杂度高 时间复杂度仍不能令人满意

应用基本数据结构——队列



若 i<j 且 $a_i \le a_j$，则插入 a_j 后可以将 a_i 从队列中删除

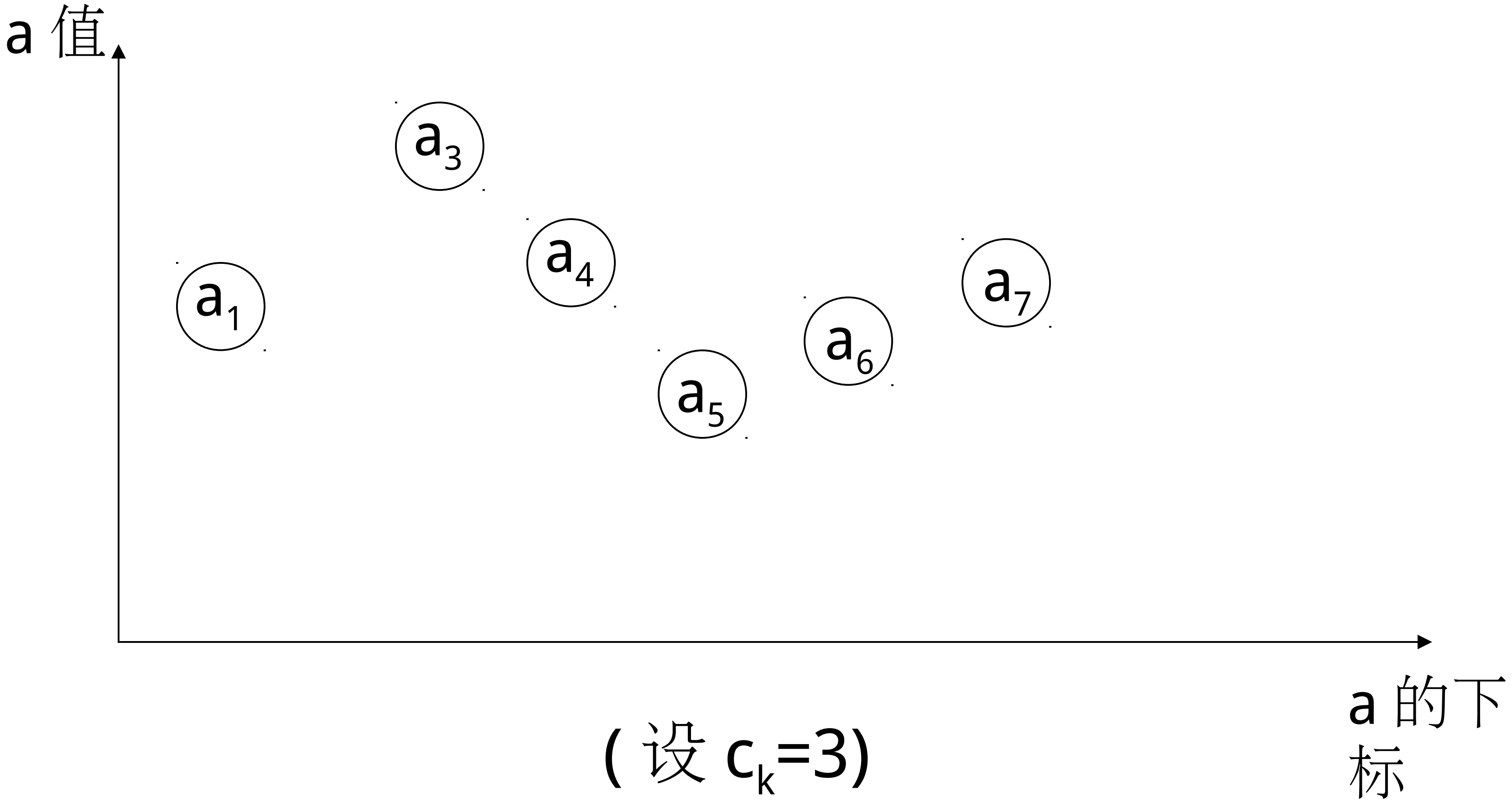
队列中所有元素呈严格递减顺序

a 的下标

队列的操作

删除	队头指针加一
插入	从队尾开始，依次删除所有不大于 a_i 的 a 值，再将 a_i 插入队尾
查找	队头元素即为最大值

动画演示



(设 c_k=3)

时间复杂度分析

一个元素最多被插入一次，删除一次

在一行中，插入与删除的总时间复杂度为 O(max{M,N})

每次询问最大值的时间复杂度仅为 O(1)

此算法的时间复杂度为 O(KMN)

小结

动态规划　　状态转移的时间复杂度太高

编程复杂度高

线段树

时间复杂度仍不能令人满意

降低了时间复杂度

队列

减少了编程错误的可能性

总结

基本数据结构

易于实现　　　　普遍适用

数据结构中的精华

昨天

理论经过数十年时间的检验
巧妙地解决了诸多问题
应用于算法中的诸多方面

明天

现在还不为人所知的基本数据结构的应用将被发现和研究

总结

基本数据结构　　　　　　局限性

辨证关系　　　　　　　　螺旋式发展

回归

高级数据结构

摒弃

基本数据结构

灵活掌握基本数据结构及其操作

对某些高级数据结构有所了解