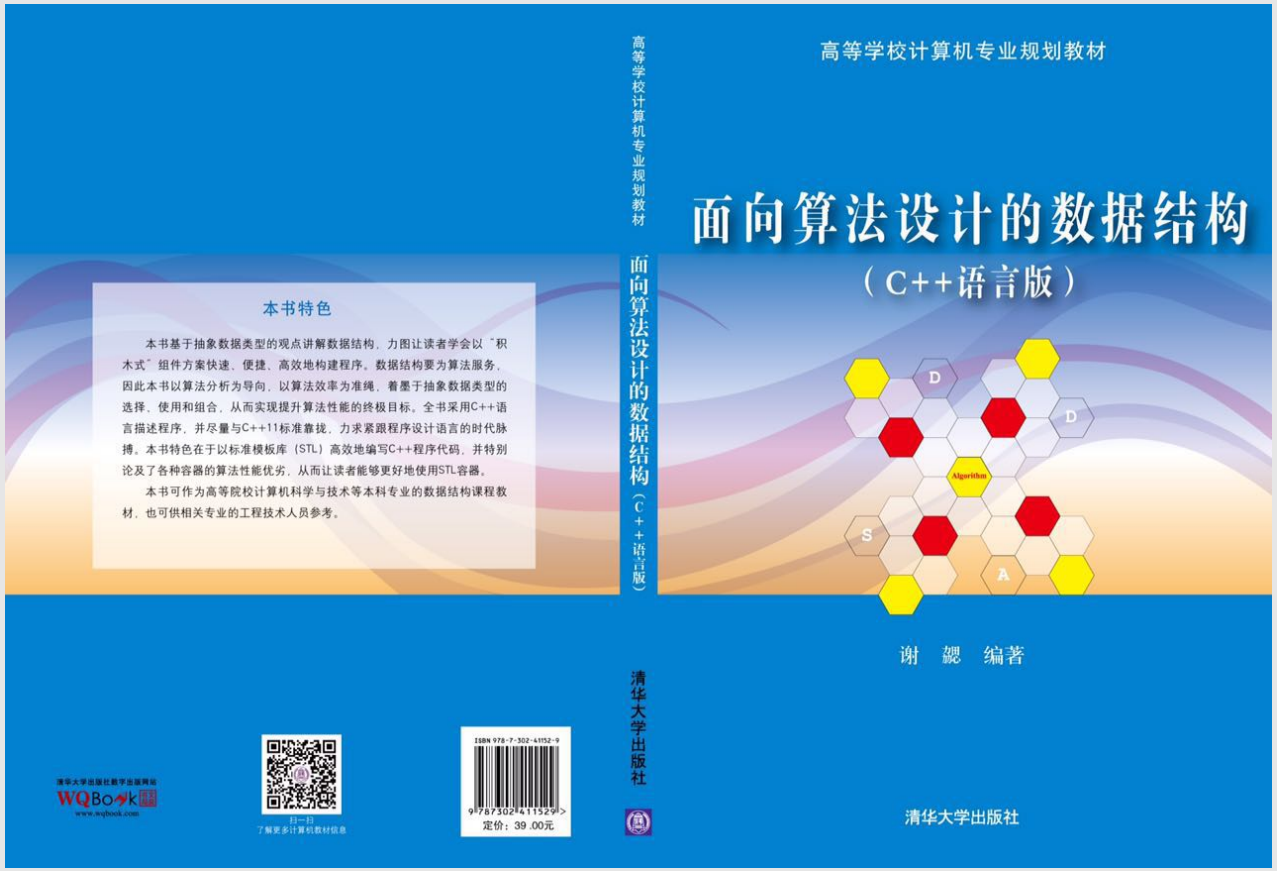
第2章 抽象数据类型

谢勰 (微博: @算海无涯-X)

August 19, 2016

教材

- 谢鑑, 《面向算法设计的数据结构（C++语言版）》. 清华大学出版社, 2015.



数据组织

- 抽象数据类型(Abstract Data Type, ADT).
  - 定义域(domain): 由值组成的集合, 它描述数据的取值范围.
  - 数据类型(data type)/类型(type)包含定义域D和施于D上的一组操作.
  - 抽象数据类型: 仅提供接口(interface), 即对自身的说明和施加于其上的操作规程, 通常还会给出操作的依赖条件以及操作执行后的结果. 但是, 其中关于数据类型的运作方式(包括数据如何表示与存储还有操作的具体实现方案)是不公开的, 这部分通常称为实现(implementation).
- 对于算法设计者来说, 通常面对的是大量甚至海量的数据, 如何组织它们? 采用何种形式?

数据组织 (续)

- 基于算法设计视角的数据组织: 将数据整体以某种抽象数据类型的形式表达, 对所处理数据的操作都交由抽象数据类型完成, 而并不去考虑抽象数据类型的内部实现.
- 对于此类抽象数据类型, 其实现部分正是数据结构和施加于其上的操作, 粗略地说, 也就是数据的静态组织形态和动态操作算法. 事实上, 数据结构这个概念还没有一个能为大家所接受的通用定义.
- 对于常见的数据结构, 我们会对其起名来更好地表达和交流, 例如AVL树. 事实上, 一旦选择了某个数据结构作为实现, 抽象数据类型的性能也就完全确定, 从这个意义上说抽象数据类型的挑选本质上是确定其数据结构.

使用iPod Shuffle——抽象数据类型

图源自: https://www.ifixit.com/Teardown/iPod+Shuffle+4th+Generation+Teardown/3559.

拆解iPod Shuffle——数据结构

图源自：https://www.ifixit.com/Teardown/iPod+Shuffle+4th+Generation+Teardown/3559.

在数据集中查找给定值

- 在杂乱无章的数据中查找是实际中经常遇到的问题:
  - 我们需要在手写的电话号码簿中找某位朋友的号码.
  - 在未经整理的支票中找出一张符合指定要求的支票.
- 这些数据都是未经排序的, 现实原因是:
  - 无法做到总有时间整理数据.
  - 可能由于实物载体的限制导致数据无法排序.
- 在计算机中又该如何解决呢?
  - 数据存储问题.
  - 查找算法问题.

存储模型

- 从最简单的物理存储方式看，数据按照到达的顺序存放在数组中比较符合上述模型的要求. 不妨假定这些数据为int型，且各不相同.
- 设数据存放在长为1000的int型数组data中，该数组前N个位置(即data[0]到data[N - 1])存放着当前保存的数据.
- 任务——指定某个值k，确认数据中是否存在k这个值:
  - 若存在则输出相应的位置;
  - 否则需要输出一个实际中并不存在的位置(例如N).

线性查找

- 容易想到，可从起始处依次向后查询，若找到则终止，否则一直查询到终止位置. 这种在无序数组中依次查找的算法一般称为顺序查找(sequential search).
- 由于顺序查找在线性排列的数据结构(例如后文将会提到的链表结构)中同样适用，所以也叫线性查找(linear search).
- 程序: https://github.com/xiexiexx/DSAD/blob/master/Ch02/linearsearch.cpp.
- 算法基本已无太大改进空间(后文会提到哨兵方式)，我们看看数据存储模型. 可以看出数组方式虽然简单直接，但缺点亦可显见.

缺点一: 长度受限制

- 初始设定好之后的数组大小有限制, 在程序运行过程中数组是静态化的, 因此在后续访问中不可越界(主要指的是上界).
- 长度实时可变并且操作方便的"数组"则是一个较好的方案, 这种抽象数据类型一般被称之为向量(vector).
- 可以使用C++中的向量(即vector容器)作为解决方案, 注意其核心仍然是一个动态分配的数组, 但它提供了很好的动态化策略.
- 我们可以很容易地将线性查找改写为向量版本. 程序: https://github.com/xiexiexx/DSAD/blob/master/Ch02/linearsearch_vector.cpp.

缺点二：有序则难变

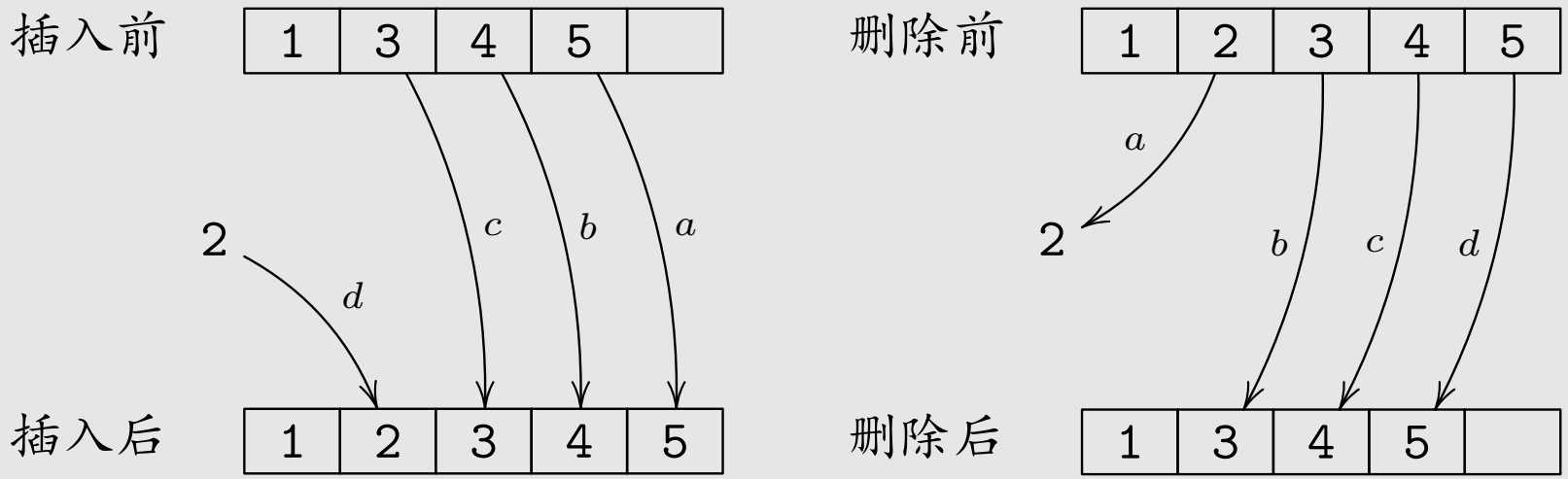


图 1: 在有序数组/向量中进行插入和删除

以链表作为解决方案

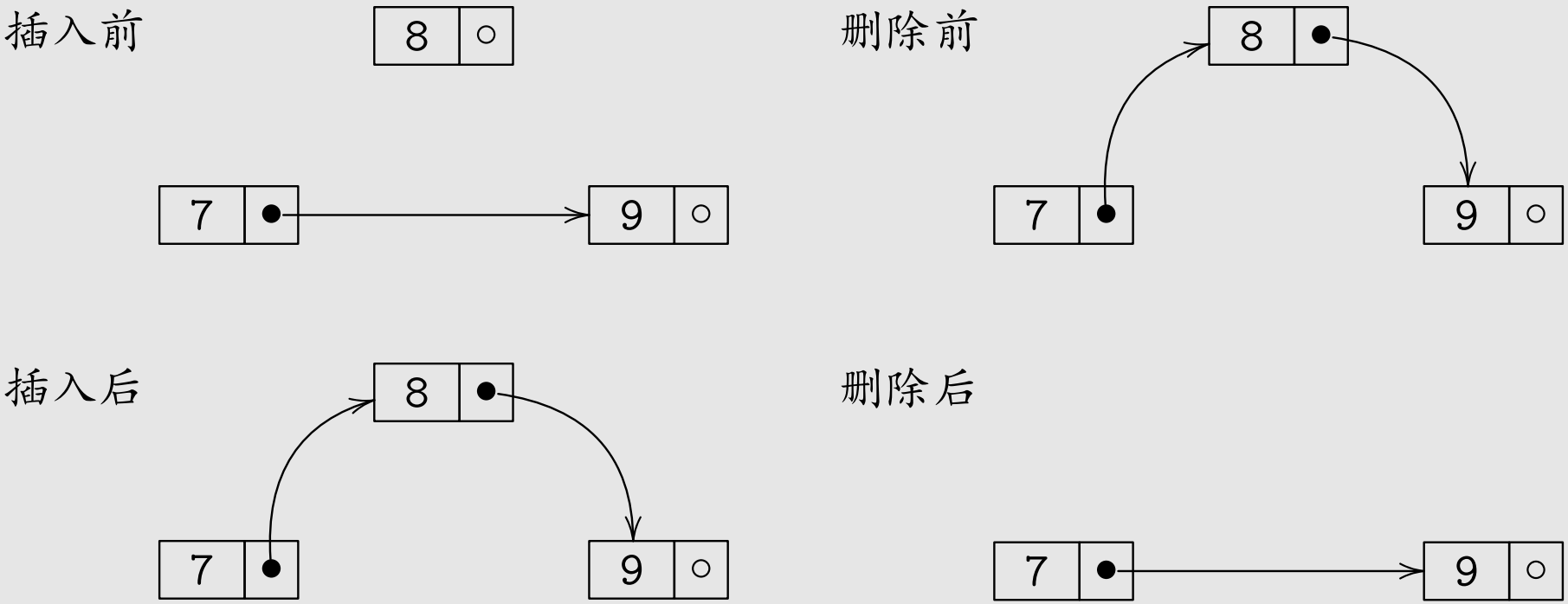


图 2: 在有序链表中进行插入和删除

缺点三：查变难两全

从数据的逻辑组织来看，数组，向量和链表都是线性结构．由于线性结构的无分支性，维护序关系通常都会牵一发而动全身，数据集如果要达到快速变动的要求则意味着无法对它及时整理，在这种场景下进行查找肯定困难重重．

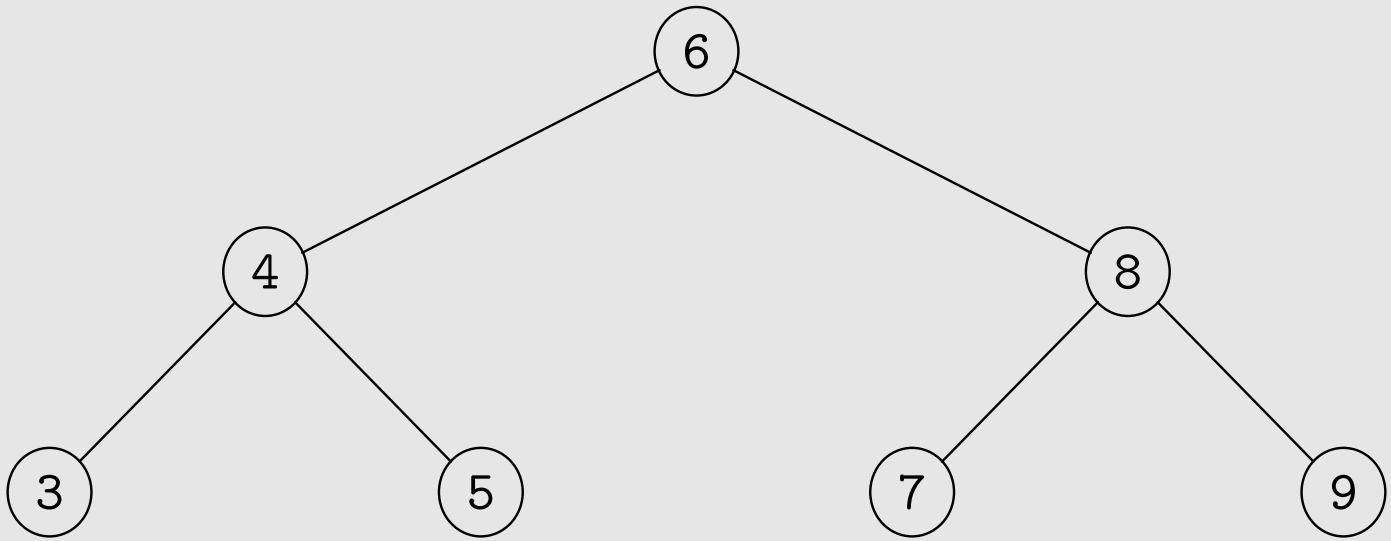


图 3：解决方案：二叉查找树

抽象数据类型视角

- 提出所涉及操作应具备的功能，筛选出可用的抽象数据类型.
- 仔细分析可用的各种抽象数据类型的性能和适用情况.
- 基于算法分析和其他因素最终选择合适的抽象数据类型.
- 注意，内部实现是数据结构的内容，可以不必过于深入了解.

Example

假设有若干种抽象数据类型，它们均可存放数据，且提供三种操作：查找特定元素；插入新元素；删除已有元素. 这是查找问题实例的一般化.

数组(包括普通数组和动态分配的数组)、链表和树是三种最基本的数据结构存储形态，利用它们可以构造出复杂的结构.

集合

- 设集合X存放T型元素，key是T型对象.
- 类型D<T>与T相关，X中每个元素都唯一对应一个D<T>*型指针p，而如果对p使用p->data可得到所对应元素(T型)，
- 称X是满足SET要求的一个实例，若在X上能执行SET所要求的操作.

表 1: 抽象数据类型SET

操作	功能
X.insert(key);	将key放入X中. 至于数据如何组织则是X自己的事.
X.erase(p);	将指针p所对应的元素删除，前提是p与X中元素有所对应.
p = X.search(key);	若key在X中则返回相应的指针赋给p，否则将p赋值为NULL.
p = X.maximum_at();	返回X中最大元素的指针赋给p，若X为空集则将p赋值NULL.
p = X.minimum_at();	返回X中最小元素的指针赋给p，若X为空集则将p赋值NULL.
key = p->data;	以p->data对key赋值，前提是p与X中元素有所对应.
n = X.size();	将X的大小赋给n，若X为空集则将n赋值0.

功能与实现

我们给出有序向量(S1)和无序向量(S2)的接口，它们拥有类似于集合的功能.

表 2: 模板类S1/S2的主要功能

操作	功能
`X.insert(key);`	将key放入X中.
`X.erase(pos);`	将pos位置的元素删除，前提是pos $\in [0, $ `X.size()`$)$.
`pos = X.search(key);`	若key在X中则返回所对应的位置赋给pos，否则给pos赋值为`X.size()`.
`pos = X.maximum_at();`	返回X中最大元素的位置赋给pos，若X为空集则给pos赋值`X.size()`(此时为0).
`pos = X.minimum_at();`	返回X中最小元素的位置赋给pos，若X为空集则给pos赋值`X.size()`(此时为0).
`key = X(pos);`	以pos位置的元素值(也即`X(pos)`)对key赋值，前提是pos $\in [0, $ `X.size()`$)$.
`n = X.size();`	将X的大小赋给n，若X为空集则给n赋值0.

向量的伸缩

0 1 2 3 4 5 6 7

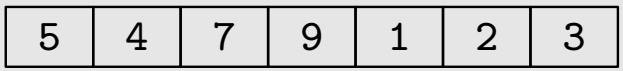
(a) 向量初始长度为8

执行V.resize(2 * V.size());之后:

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

(b) 向量长度增长到16

执行V.resize(7);之后:



0 1 2 3 4 5 6

(c) 向量长度缩短到7

实现与对比

- 有序向量: https://github.com/xiexiexx/DSAD/blob/master/Ch02/orderedvector.h.
- 无序向量: https://github.com/xiexiexx/DSAD/blob/master/Ch02/unorderedvector.h.

表 3: 有序向量与无序向量性能对比 ($n < 1024$)

操作	有序向量(S1)的效率	无序向量(S2)的效率
`insert`	$O(n)$	$O(1)$
`erase`	$O(n)$	$O(1)$
`search`	$O(\log n)$	$O(n)$
`maximum_at`	$O(1)$	$O(n)$
`minimum_at`	$O(1)$	$O(n)$

例

若数据存于S2<T>型对象X中，将其按从小到大的次序在屏幕上输出，并分析算法的时间复杂度.

解

```
// 方案1：依次取出最小元，直到无元素可取为止.
for ( int i = X.minimum_at(); i != X.size(); i = X.minimum_at() )
{
    cout << X(i) << ' ';
    X.erase(i);
}
// 方案2：比方案1更清晰易读，也更多地站在抽象数据类型的观点去考虑问题.
while (X.size() > 0)    // 思考为何不用X.size() != 0
{
    int i = X.minimum_at();
    cout << X(i) << ' ';
    X.erase(i);
}
```

例

令C是一个由m个元素组成的集合，设计算法并选择合适的模板类，将输入数据中不在C的部分保存于集合D中，即实现输入数据与C的差集. 假定我们可以调用无参数的input函数获得输入数据，当返回值为e时输入终止，并设输入数据无重复.

解

```
for ( T data = input(); data != e; data = input() )
    if (C.search(data) == C.size()) // 若data不在C中则插入D中.
        D.insert(data);
```

若输入元素个数为n，在最坏情况下采用不同类完成输入整理工作的时间复杂度为:

- C为S1<T>型对象，D为S1<T>型对象. 需$O(n(\log m + n))$时间.
- C为S1<T>型对象，D为S2<T>型对象. 需$O(n \log m)$时间.
- C为S2<T>型对象，D为S1<T>型对象. 需$O(n(m + n))$时间.
- C为S2<T>型对象，D为S2<T>型对象. 需$O(nm)$时间.

最优选择: C为S1<T>型对象，D为S2<T>型对象.

STL容器一览

- 向量容器——vector.
- 列表容器——list.
- 栈容器——stack.
- 向量容器——vector.
- 列表容器——list.
- 栈容器——stack.
- 队列容器——queue.
- 优先级队列容器——priority_queue.

STL容器一览 (续)

- 集合容器——set.
- 多重集合容器——multiset.
- 映射容器——map.
- 多重映射容器——multimap.
- 无序集合容器——unordered_set.
- 无序多重集合容器——unordered_multiset.
- 无序映射容器——unordered_map.
- 无序多重映射容器——unordered_multimap.

应用视角下的数据结构思维

- 算法场景：接受一个大约10MB的英文序列但具体长度未知，需要将该序列逆置后存为文本文件.
- 抽象数据类型选择：显然应选择“后进先出”的栈来完成任务.
- 数据结构选择：由于长度未知，可选vector，deque，list来实现栈，但vector性能最佳!
- 精细调优：利用vector的reserve功能提前预留出10MB存储空间，可以极大地提高效率.

课堂练习

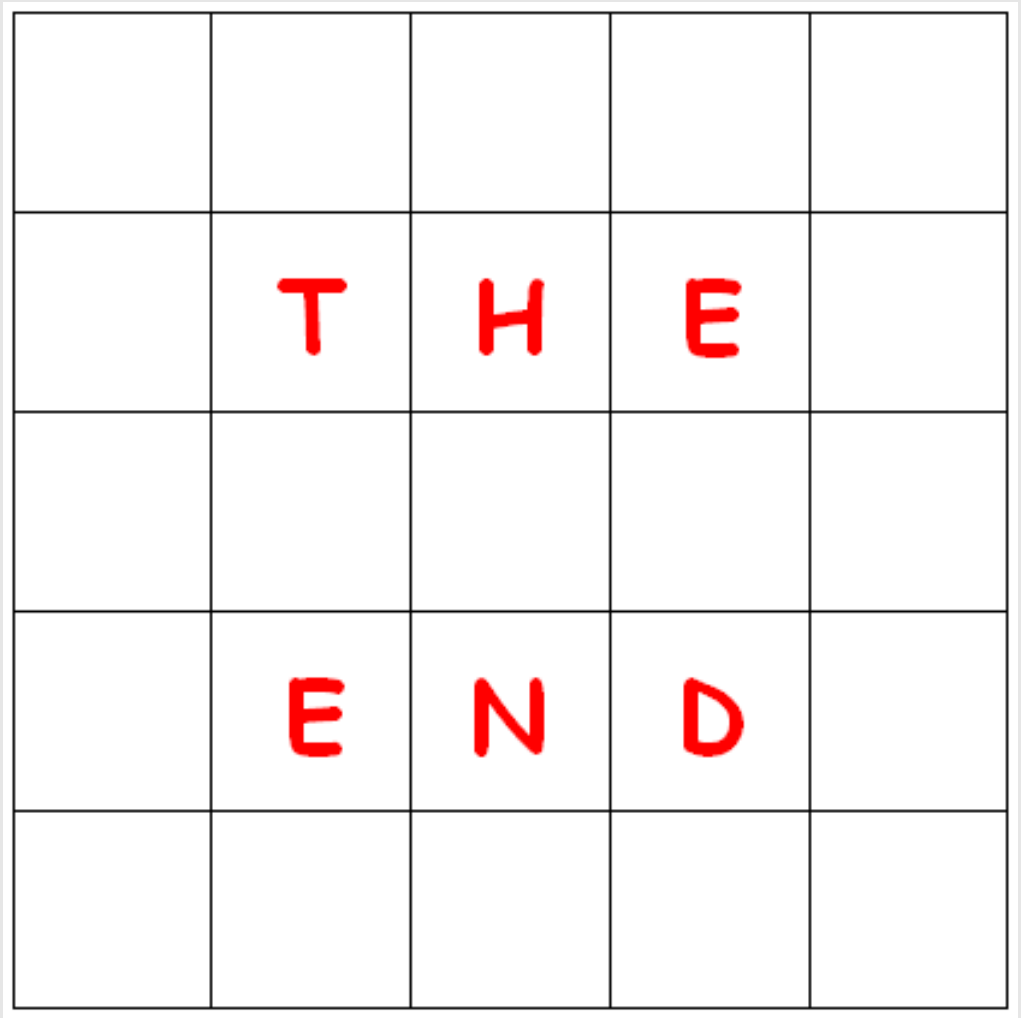
习题

设有set<T>型容器X，设计算法寻找X中第k大的元素并分析算法时间复杂度.

解

如何解决?

本章完结



For more Information, please visit:

PROJECT: DSAD.

WEBSITE: https://github.com/xiexiexx/dsad.

EMAIL: DSAD2015@163.com.

WEIBO: @算海无涯-X.