Exercise 16.2-5

练习 16.2-5

- Describe an efficient algorithm that, given a set $\{x_1, x_2, ..., x_n\}$ of points on the real line, determines the smallest set of unit-length closed intervals that contains all of the given points. Argue that your algorithm is correct.

- **Note:** it should not be difficult to come up with the algorithm. Try to write out the detailed correctness proof!

- 设计一个高效算法，对实数线上给定的一个点集 $\{x_1, x_2, ..., x_n\}$，求一个单位长度闭区间的集合，包含所有给定的点，并要求此机和最小。证明你的算法是正确的。

- **注意：**想出这个算法应该不难。试着写出详细的正确性证明！

Solution to Exercise 16.2-5

- **Algorithm:**
  Sort the points by location and consider the points in order.
  At each iteration, let $x_{(i)}$ be the smallest uncovered point,
  choose the interval $[x_{(i)}, x_{(i)} + 1]$, then remove the points covered by this interval.

- **Proof:**
  Consider the following subproblem: For each number k, what is the largest m such that k unit intervals suffice to cover all points $\{x_{(1)}, \ldots, x_{(m)}\}$ (in sorted order)?

  Denote the answer for case k by $m(k)$.
  That is, k unit intervals cover $\{x_{(1)}, \ldots, x_{(m(k))}\}$ according to the greedy algorithm.
  We show by induction that the greedy algorithm yields the correct answer for all k.

Solution to Exercise 16.2-5

- Induction basis ($k = 0$): It is trivially clear that $m(0) = 0$.

- Induction step: Assume given a certain value k.
  Suppose the answer $m(k)$ is correct, i.e. the greedy algorithm correctly answers how many points can be covered by k intervals.

  We need to prove that $m(k+1)$ is correct.
  Suppose (for contradiction) that there is a covering by $k+1$ intervals of the points $\{x_{(1)}, \ldots, x_{(m(k+1)+1)}\}$. By the greedy algorithm, any interval covering $x_{(m(k+1)+1)}$ cannot cover any of the points $x_{(1)}, \ldots, x_{(m(k)+1)}$, so the remaining k intervals must cover all of $\{x_{(1)}, \ldots, x_{(m(k)+1)}\}$, contradicting the assumption that the answer $m(k)$ is correct.

Exercise 17.1-1+ 练习 17.1-1+

- If the set of stack operations included a MULTIPUSH(S, *k*, *i*) operation, which pushes *k* copies of *i* onto stack *S*, would the $O(1)$ bound on the amortized cost of stack operations continue to hold?
- Give an analysis that shows that even in the presence of MULTIPUSH with a suitable amortized cost, PUSH/POP/MULTIPOP have $O(1)$ amortized cost.

- 如果栈操作包括 MULTIPUSH(S, *k*, *i*) 操作，它将 *k* 个数据项 *i* 压入栈 *S* 中，那么栈操作摊还代价的界还是 $O(1)$ 吗?
- 分析表明，即使存在具有适当摊还代价的MULTIPUSH， 那么PUSH/POP/MULTIPOP也具有 $O(1)$摊还代价。

Solution to Exercise 17.1-1

The following program:

$$\textsc{Multipush}(S, (n-1)^2, 1)$$

$$\left.\begin{array}{l}\textsc{Multipop}(S, n)\\ \qquad\vdots \\ \textsc{Multipop}(S, n)\end{array}\right\}\ n-1 \text{ times}$$

has running time in $\Omega(n^2)$ but has n operations.

Therefore, when using aggregate analysis,
the amortized running time of a single operation cannot be lower than $O(n^2) / n = O(n)$.

Solution to Exercise 17.1-1+

- Prices for operations:
- 运营价格：

Operation	PUSH	POP	MULTIPOP	MULTIPUSH	操作
Amortized cost	2	0	0	$2k$	摊还代价

- MULTIPUSH: when k items are pushed onto the stack, it actually costs k units. The k additional units of cost are used to pay the credit of the items pushed onto the stack.
- MULTIPUSH：当堆栈上压入k元素时，实际上需要花费k个单元。k个额外的代价单元用于支付压入堆栈上的项目的信用。

Exercise 17.3-7

Design a data structure to support the following two operations for a dynamic multiset *S* of integers, which allows duplicate values:

- INSERT(*S*, *x*) inserts *x* into *S*.
- DELETE-LARGER-HALF(*S*) deletes the largest $\lceil |S|/2 \rceil$ elements from *S*.

Explain how to implement this data structure so that any sequence of *m* INSERT and DELETE-LARGER-HALF operations runs in $O(m)$ time. Your implementation should also include a way to output the elements of *S* in $O(|S|)$ time.

练习 17.3-7

微动态整数多重集 S （允许包含重复制）设计一种数据结构，支持如下两个操作：

- INSERT(*S*, *x*) 将 *x* 插入 *S* 中。
- DELETE-LARGER-HALF(*S*) 将最大的 $\lceil |S|/2 \rceil$ 个元素从 *S* 中删除。

解释如何实现这种数据结构，使得任意 *m* 个 INSERT 和 DELETE-LARGER-HALF 操作的序列能在 $O(m)$ 时间内完成。还要实现一个能在 $O(|S|)$ 时间内输出所有元素的操作。

Solution to Exercise 17.3-7

Use a dynamic table T to store the elements of S.

- INSERT(S, x) calls TABLE-INSERT(T, x).
- DELETE-LARGER-HALF(S) does the following:
  1. Call the $O(|S|)$ SELECT algorithm to find the lower median m of the data in T. This algorithm moves the median to position $T.table[\lfloor (|S|-1) / 2 \rfloor]$, and the elements $T.table[\lfloor (|S|-1) / 2 \rfloor + 1]$, ..., $T.table[|S|]$ contain values $\geq m$.
  2. Set $T.table.num = \lfloor (|S|-1) / 2 \rfloor$ and shrink the dynamic table. This also requires time in $O(|S|)$.
- Output the elements of S: No specific order was prescribed. Just print $T.table[0]$, $T.table[1]$, ..., $T.table[T.num-1]$.

Solution to Exercise 17.3-7

- Potential analysis using the potential function $\Phi(T) = 3 \times T.num - T.size$.
  Note that $\Phi(T) \geq T.num$.

- INSERT(S, x) changes the potential from p to $p+3$,
  or (if the old $T.num == T.size$) from the old $2 \times T.num$ to the new $T.num$.
  Therefore, the amortized cost of INSERT is in $O(1)$.

- DELETE-LARGER-HALF(S) changes the potential from p to $p/2$.
  The difference is in $\Omega(T.num)$ and can pay for the $O(|S|)$ running time that is needed.
  Therefore, the amortized cost of DELETE-LARGER-HALF is 0.

Exercise

- Write pseudocode for TABLE-DELETE (e.g. adapt from TABLE-INSERT)

练习

- 请写TABLE-DELETE的伪代码（可以适应TABLE-INSERT的）。

Table Shrinking　表收缩

$T.size/4 \le T.num \le T.size$

TABLE-DELETE(T,x)
if $T.num < 1$ **or** $T.table[T.num-1] \neq x$ **then error**
if $T.num == 1$
    free $T.table$
    $T.size = 0$
else if $T.num \le T.size / 4$
    allocate *new-table* with $T.size / 2$ slots
    move items from $T.table$ to *new-table*
    free $T.table$
    $T.table$ = *new-table*
    $T.size = T.size / 2$
$T.num = T.num - 1$

table becomes empty: free all space

表变空：空余内存

table is ¼ full and needs to be shrunk

表满四分之一需要收缩

$T.size/4 \le T.num \le T.size$

$T.size/4 \lneq T.num \le T.size$
(or $T.num = 1$ and $T.size = 0$)