1-5 数据与数据结构 (II)

魏恒峰

hfwei@nju.edu.cn

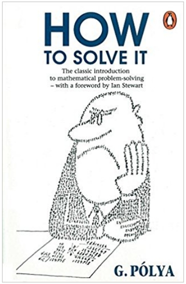
2017 年 11 月 27 日

温故而知新

— 孔子

温故而知新

— 孔子



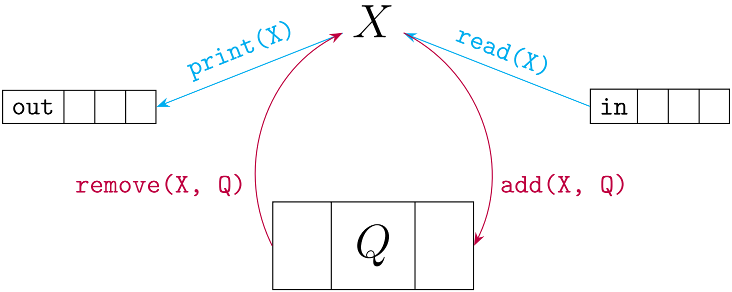
Looking Back!

Stackable/Queueable Permutations

Treesort Algorithm

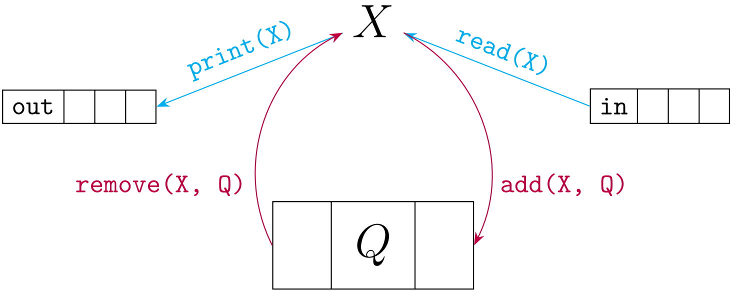
Queueable Permutations

DH 2.14: Queueable Permutations



DH 2.14: Queueable Permutations

$$\boxed{\texttt{out} = (a_1, \cdots, a_n) \xleftarrow[X=0]{Q=\emptyset} \texttt{in} = (1, \cdots, n)}$$



DH 2.14: Queueable Permutations

(a) Show that the permutations given in Excecise 2.12(b) are queueable.

  (i) $(3, 1, 2)$
  (ii) $(4, 5, 3, 7, 2, 1, 6)$

DH 2.14: Queueable Permutations

(a) Show that the permutations given in Excecise 2.12(b) are queueable.

  (i) $(3, 1, 2)$

  (ii) $(4, 5, 3, 7, 2, 1, 6)$

DH 2.14: Queueable Permutations

(a) Show that the permutations given in Excecise 2.12(b) are queueable.

(i) $(3,1,2)$

(ii) $(4,5,3,7,2,1,6)$

DH 2.14: Queueable Permutations

(b) Prove that every permutation are queueable.

DH 2.14: Queueable Permutations

(b) Prove that every permutation are queueable.

```
X = 0      Q = ∅      in != EOF
```

```
foreach 'a' ∈ out:
  if ('a' == in)
    read(X)
    print(X)
  else if ('a' > in)
    add-Q-till('a')
  else // ('a' < in)
    cycle-Q-till('a')
```

DH 2.14: Queueable Permutations

(b) Prove that every permutation are queueable.

```
X = 0        Q = ∅        in != EOF
```

```
foreach 'a' ∈ out:
  if ('a' == in)
    read(X)
    print(X)
  else if ('a' > in)
    add-Q-till('a')
  else // ('a' < in)
    cycle-Q-till('a')
```

```
add-Q-till('a'):
  while (('x' ∈ in) != 'a')
    add(X, Q)
  read(X)
  print(X)
```

DH 2.14: Queueable Permutations

(b) Prove that every permutation are queueable.

```
X = 0      Q = ∅      in != EOF
```

```
foreach 'a' ∈ out:
  if ('a' == in)
    read(X)
    print(X)
  else if ('a' > in)
    add-Q-till('a')
  else // ('a' < in)
    cycle-Q-till('a')
```

```
add-Q-till('a'):
  while (('x' ∈ in) != 'a')
    add(X, Q)
  read(X)
  print(X)
```

```
cycle-Q-till('a'):
  while (('x' ∈ Q) != 'a')
    remove(X, Q)
    add(X, Q)
  remove(X, Q)
  print(X)
```

DH 2.14: Queueable Permutations

(b) Prove that every permutation are queueable.

```
X = 0     Q = ∅     in != EOF
```

```
foreach 'a' ∈ out:
  if ('a' == in)
    read(X)
    print(X)
  else if ('a' > in)
    add-Q-till('a')
  else // ('a' < in)
    cycle-Q-till('a')
```

```
add-Q-till('a'):
  while (('x' ∈ in) != 'a')
    add(X, Q)
  read(X)
  print(X)
```

```
cycle-Q-till('a'):
  while (('x' ∈ Q) != 'a')
    remove(X, Q)
    add(X, Q)
  remove(X, Q)
  print(X)
```

DH 2.14: Queueable Permutations

(b) Prove that every permutation are queueable.

Proof.

```
foreach 'a' ∈ out:
  if ('a' >= in)
    add-Q-till('a')
  else // ('a' < in)
    cycle-Q-till('a')
```

DH 2.14: Queueable Permutations

(b) Prove that every permutation are queueable.

Proof.

```
foreach 'a' ∈ out:
  if ('a' >= in)
    add-Q-till('a')
  else // ('a' < in)
    cycle-Q-till('a')
```

```
foreach 'a' ∈ out:
  if ('a' ∈ in)
    add-Q-till('a')
  else // ('a' ∈ Q)
    cycle-Q-till('a')
```

□

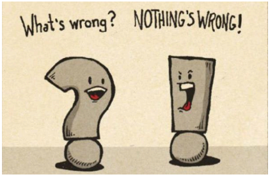
DH 2.14: Queueable Permutations

(b) Prove that every permutation are queueable.

Proof.

```
foreach 'a' ∈ out:
  if ('a' >= in)
    add-Q-till('a')
  else // ('a' < in)
    cycle-Q-till('a')
```

```
foreach 'a' ∈ out:
  if ('a' ∈ in)
    add-Q-till('a')
  else // ('a' ∈ Q)
    cycle-Q-till('a')
```



□

Pseudocode

Pseudocode


Make everything as
SIMPLE
as possible,
but not simpler.
- ALBERT EINSTEIN (1879-1955)
QUOTESEVERLASTING.COM

Pseudocode

"Executable" at an abstract level.

DH 2.14: Queueable Permutations

(b) Prove that every permutation are queueable.

An "AHA!" Proof.

```
foreach 'a' ∈ in:
  add(X, Q)

foreach 'a' ∈ out:
  cycle-Q-till('a')
```

DH 2.14: Queueable Permutations

(b) Prove that every permutation are queueable.

An "AHA!" Proof.

```
foreach 'a' ∈ in:
  add(X, Q)

foreach 'a' ∈ out:
  cycle-Q-till('a')
```

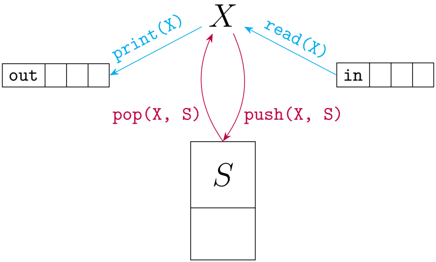
□

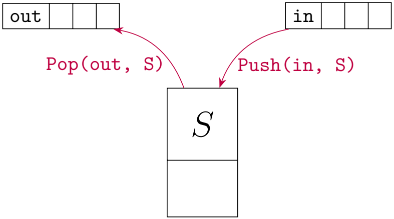
DH 2.14: Queueable Permutations

(c) Prove that every permutation can be obtained by two stacks.

DH 2.14: Queueable Permutations

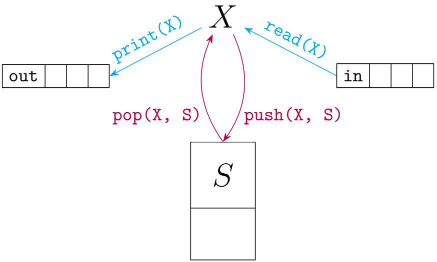
(c) Prove that every permutation can be obtained by two stacks.

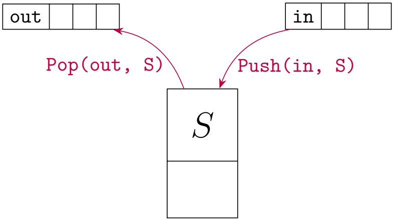




DH 2.14: Queueable Permutations

(c) Prove that every permutation can be obtained by two stacks.





We can similarly speak of a permutation obtained by two stacks, if we permit the push and pop operations on two stacks S and S'.

DH 2.14: Queueable Permutations

(c) Prove that every permutation can be obtained by two stacks.

2 stacks model

DH 2.14: Queueable Permutations

(c) Prove that every permutation can be obtained by two stacks.

2 stacks model

```
foreach 'a' ∈ in:
  read(X)
  push(X, S′)

foreach 'a' ∈ out:
  if ('a' <= top(S′)) // ∈ S′
    transfer-till(S′, S, 'a')
  else // ∈ S
    transfer-till(S, S′, 'a')
```

DH 2.14: Queueable Permutations

(c) Prove that every permutation can be obtained by two stacks.

2 stacks model

```
foreach 'a' ∈ in:
  read(X)
  push(X, S′)

foreach 'a' ∈ out:
  if ('a' <= top(S′)) // ∈ S′
    transfer-till(S′, S, 'a')
  else // ∈ S
    transfer-till(S, S′, 'a')
```

Can you PROVE it?

DH 2.15: Algorithm for Queueable Permutations

Extend the algorithm you were asked to design in Exercise 2.13, so that **if** the given permutation cannot be obtained by a stack, the algorithm will print the series of operations on two stacks that will generate it.

DH 2.15: Algorithm for Queueable Permutations

Extend the algorithm you were asked to design in Exercise 2.13, so that **if** the given permutation cannot be obtained by a stack, the algorithm will print the series of operations on two stacks that will generate it.

DH 2.15: Algorithm for Queueable Permutations

Extend the algorithm you were asked to design in Exercise 2.13, so that **if** the given permutation cannot be obtained by a stack, the algorithm will print the series of operations on two stacks that will generate it.

```
two-stackable-perm(in, X, S, S′)
```

DH 2.15: Algorithm for Queueable Permutations

Extend the algorithm you were asked to design in Exercise 2.13, so that **if** the given permutation cannot be obtained by a stack, the algorithm will print the series of operations on two stacks that will generate it.

```
two-stackable-perm(in, X, S, S′)
```

```
if (! stackable-perm(in, X, S))
  two-stackable-perm(in, X, S, S′)
```

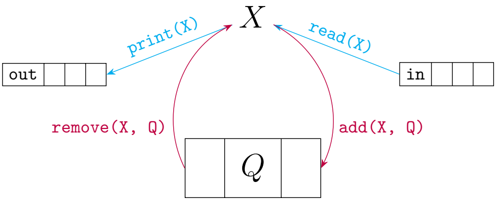
DH 2.15: Algorithm for Queueable Permutations

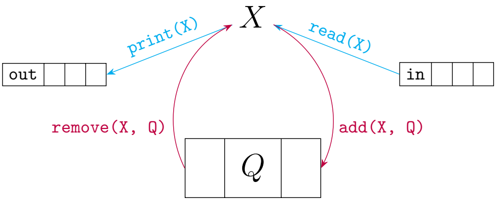
Extend the algorithm you were asked to design in Exercise 2.13, so that **if** the given permutation cannot be obtained by a stack, the algorithm will print the series of operations on two stacks that will generate it.

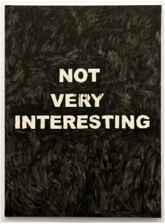
```
two-stackable-perm(in, X, S, S′)
```

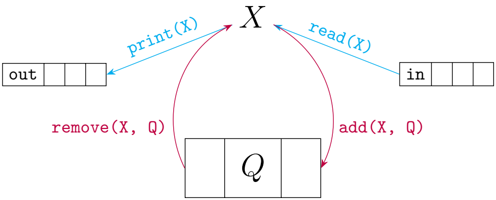
```
if (! stackable-perm(in, X, S))
  two-stackable-perm(in, X, S, S′)
```

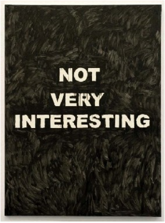
Embedding "transfer" into "stackable-perm".

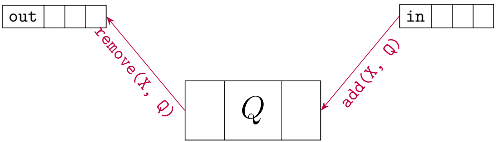

X
print(X)
read(X)
out
in
remove(X, Q)
add(X, Q)
Q

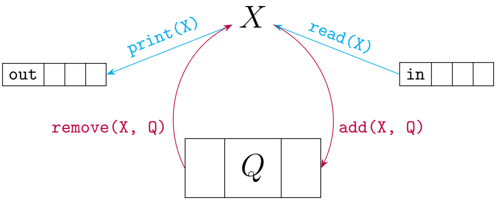

X
print(X)
read(X)
out
in
remove(X, Q)
add(X, Q)
Q

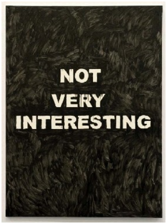

NOT
VERY
INTERESTING

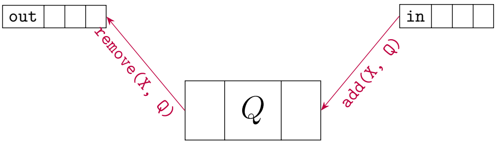

X
print(X)
read(X)
out
in
remove(X, Q)
add(X, Q)
Q

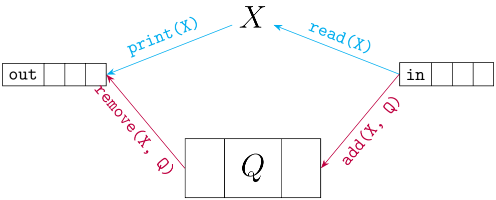

NOT
VERY
INTERESTING

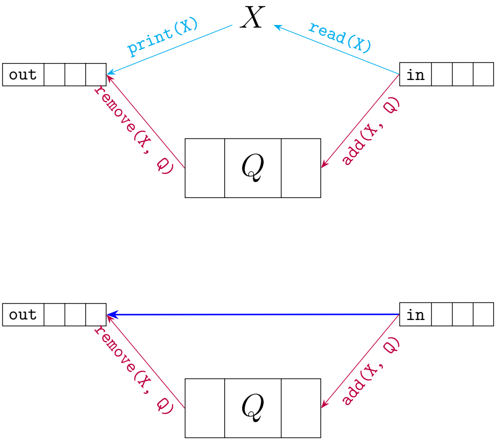

out
in
remove(X, Q)
add(X, Q)
Q

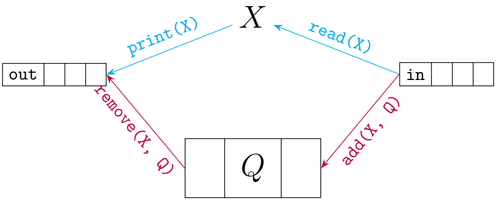

X
print(X)
read(X)
out
in
remove(X, Q)
add(X, Q)
Q

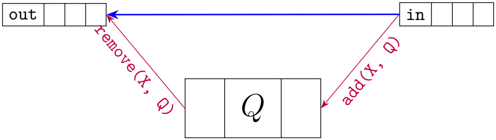

NOT
VERY
INTERESTING


out
in
remove(X, Q)
add(X, Q)
Q

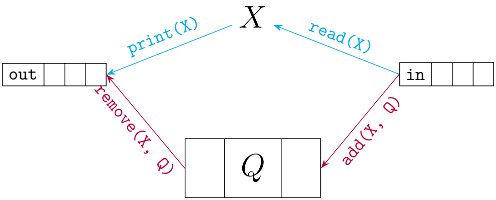

X
print(X)
read(X)
out
in
remove(X, Q)
add(X, Q)
Q

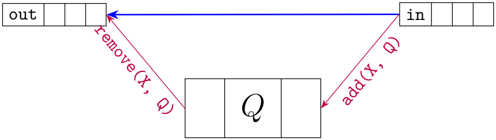

X
print(X)
read(X)
out
in
remove(X, Q)
add(X, Q)
Q
out
in
remove(X, Q)
add(X, Q)
Q


X
print(X)
read(X)
out
in
remove(X, Q)
add(X, Q)
Q


out
in
remove(X, Q)
add(X, Q)
Q

3 2 1


X
print(X)
read(X)
out
in
remove(X, Q)
add(X, Q)
Q


out
in
remove(X, Q)
add(X, Q)
Q


哎哟，
有点意思

3 2 1

Theorem (Queueable Permutations)

A permutation $(a_1, \cdots, a_n)$ *is queueable* $\iff$ *it is not the case that*

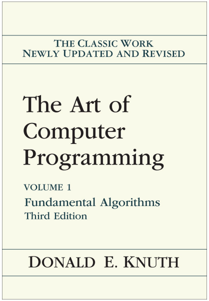
$$321\text{-}\textit{Pattern} : \boxed{\textit{out} = \cdots a_i \cdots a_j \cdots a_k : i < j < k \land a_i > a_j > a_k}$$

Theorem (Queueable Permutations)

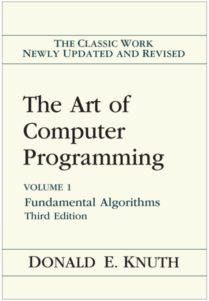
A permutation $(a_1, \cdots, a_n)$ *is queueable* $\iff$ *it is not the case that*

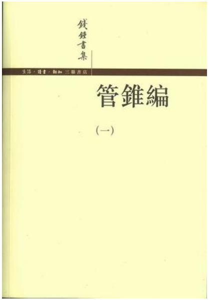
$$321\text{-}\textit{Pattern} : \boxed{\textit{out} = \cdots a_i \cdots a_j \cdots a_k : i < j < k \land a_i > a_j > a_k}$$

Chapter 2 — Information Structures 232

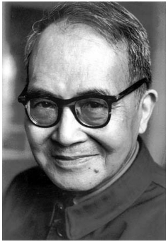

THE CLASSIC WORK
NEWLY UPDATED AND REVISED
The Art of
Computer
Programming
VOLUME 1
Fundamental Algorithms
Third Edition
DONALD E. KNUTH


Don
Knuth


THE CLASSIC WORK
NEWLY UPDATED AND REVISED
The Art of Computer Programming
VOLUME 1
Fundamental Algorithms
Third Edition
DONALD E. KNUTH


錢鍾書集
管錐編
(一)


Don Knuth




Thank
You!