1-5 数据与数据结构 (I)

魏恒峰

hfwei@nju.edu.cn

2017 年 11 月 13 日

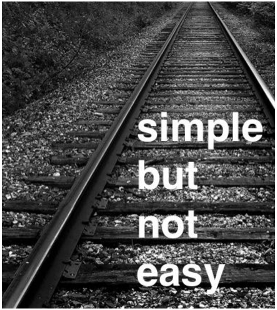
Permutations

Permutations

Generating All Permutations

Stackable/Queueable Permutations

Generating All Permutations



DH 2.9: # of Permutations

Prove that the number of permutations of n (distinct) elements is $n!$.

DH 2.9: # of Permutations

Prove that the number of permutations of n (distinct) elements is $n!$.

.

For a_1: We have n choices.

For a_2: We have $n-1$ choices.

For $\cdots$: $\cdots$

Then, # of perms is

$$n \times (n-1) \times \cdots \times 1 = n!$$

$\square$

DH 2.9: # of Permutations

Prove that the number of permutations of n (distinct) elements is $n!$.

“坊间” 证明.

For a_1: We have n choices.

For a_2: We have $n - 1$ choices.

For $\cdots$: $\cdots$

Then, # of perms is

$$n \times (n-1) \times \cdots \times 1 = n!$$

□

DH 2.9: # of Permutations

Prove that the number of permutations of n (distinct) elements is $n!$.

“坊间” 证明.

For a_1: We have n choices.

For a_2: We have $n-1$ choices.

For $\cdots$: $\cdots$

Then, # of perms is

$$n \times (n-1) \times \cdots \times 1 = n!$$

□

Prove by mathematical induction on n.

DH 2.9: # of Permutations

Prove that the number of permutations of n (distinct) elements is $n!$.

Prove by mathematical induction on n.

$$P(n) : \# \text{ of perms of } n \text{ (distinct) element is } n!$$

DH 2.9: # of Permutations

Prove that the number of permutations of n (distinct) elements is $n!$.

Prove by mathematical induction on n.

$$P(n) : \#\text{ of perms of } n \text{ (distinct) element is } n!$$

B.S. $P(1)$

I.H. $P(n)$

I.S. $P(n) \to P(n+1)$

DH 2.9: # of Permutations

Prove that the number of permutations of n (distinct) elements is $n!$.

Prove by mathematical induction on n.

$$P(n) : \#\text{ of perms of } n \text{ (distinct) element is } n!$$

B.S. $P(1)$

I.H. $P(n)$

I.S. $P(n) \to P(n+1)$

$$\underbrace{(n+1)}_{\text{1st choice}} \times \underbrace{n!}_{I.H.} = (n+1)!$$

$\square$

DH 2.11: Generate All Permutations

Design an algorithm which, given a positive integer n, generates/prints all the permutations of $[0 \cdots n)$.

DH 2.11: Generate All Permutations

Design an algorithm which, given a positive integer n, generates/prints all the permutations of $[0 \cdots n)$.

```
void perms (A[], n) {
  if (n == 1)
    print ''A[0]''
  else
    for (int i = 0; i < n; ++i)
      print ''A[i]''
      perms(A ← A \ A[i], n - 1)
      print ''\n''
}
```

DH 2.11: Generate All Permutations

Design an algorithm which, given a positive integer n, generates/prints all the permutations of $[0 \cdots n)$.

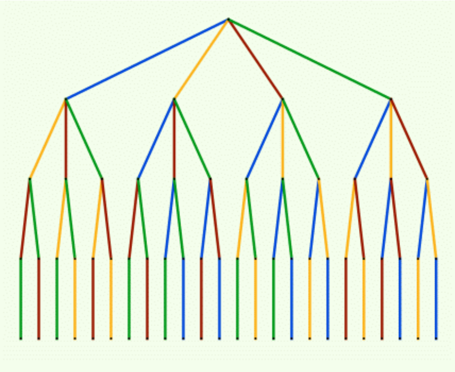
```
void perms (A[], n) {
  if (n == 1)
    print ''A[0]''
  else
    for (int i = 0; i < n; ++i)
      print ''A[i]''
      perms(A ← A \ A[i], n - 1)
      print ''\n''
}
```

generate-perms.c


ERROR


ERROR

$$A = [0, 1, 2, 3] \qquad n = 4$$



```
void perms (prifix, A[], n) {
  if (n == 1)
    print ''prifix ++ A[0]''
  else
    for (int i = 0; i < n; ++i)
      perms(prefix ← prefix ++ A[i],
            A ← A \ A[i], n - 1)
      print ''\n''
}
```

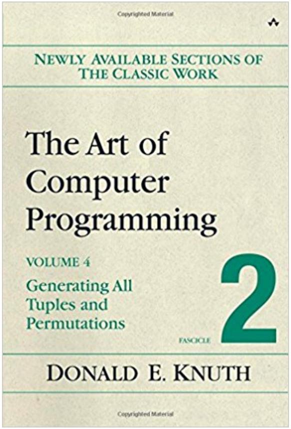
```
void perms (prefix, A[], n) {
  if (n == 1)
    print ''prefix ++ A[0]''
  else
    for (int i = 0; i < n; ++i)
      perms(prefix ← prefix ++ A[i],
            A ← A \ A[i], n - 1)
      print ''\n''
}
```

```
perms('''', A, n);
```

```
void perms (prefix, A[], n) {
  if (n == 1)
    print ''prefix ++ A[0]''
  else
    for (int i = 0; i < n; ++i)
      perms(prefix ← prefix ++ A[i],
            A ← A \ A[i], n - 1) // Space???
      print ''\n''
}
```

```
perms('''', A, n);
```

For more about "Generating All Permutations":



DH 2.10: Permutation Checking

- An integer n
- An array of integers P of length n

To check whether P is a permutation of $1 \cdots n$?

DH 2.10: Permutation Checking

- An integer n
- An array of integers P of length n

To check whether P is a permutation of $1 \cdots n$?

1. Boolean array $[1 \cdots n]$

DH 2.10: Permutation Checking

- An integer n
- An array of integers P of length n

To check whether P is a permutation of $1 \cdots n$?

1. Boolean array $[1 \cdots n]$

$$\underbrace{O(n)}_{\texttt{time}} \quad \underbrace{O(n)}_{\text{space}}$$

DH 2.10: Permutation Checking

- An integer n
- An array of integers P of length n

To check whether P is a permutation of $1 \cdots n$?

1. Boolean array $[1 \cdots n]$

   $\underbrace{O(n)}_{\texttt{time}}$ $\underbrace{O(n)}_{\text{space}}$

2. Sort and then scan

DH 2.10: Permutation Checking

- An integer n
- An array of integers P of length n

To check whether P is a permutation of $1 \cdots n$?

1. Boolean array $[1 \cdots n]$

$$\underbrace{O(n)}_{\texttt{time}} \quad \underbrace{O(n)}_{\texttt{space}}$$

2. Sort and then scan

$$\underbrace{O(n \log n)}_{\texttt{time}} \quad \underbrace{O(1)}_{\texttt{space}}$$

DH 2.10: Permutation Checking

- An integer n
- An array of integers P of length n

To check whether P is a permutation of $1 \cdots n$?

1. Boolean array $[1 \cdots n]$

$$\underbrace{O(n)}_{\texttt{time}} \quad \underbrace{O(n)}_{\text{space}}$$

2. Sort and then scan

$$\underbrace{O(n \log n)}_{\texttt{time}} \quad \underbrace{O(1)}_{\text{space}}$$

$$\underbrace{O(n)}_{\texttt{time}} \quad \underbrace{O(1)}_{\text{space}}$$

DH 2.10: Permutation Checking

- An integer n
- An array of integers P of length n

To check whether P is a permutation of $1 \cdots n$?

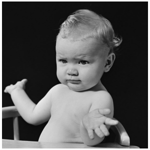
1. Boolean array $[1 \cdots n]$

$\underbrace{O(n)}_{\texttt{time}}$ $\underbrace{O(n)}_{\texttt{space}}$

2. Sort and then scan

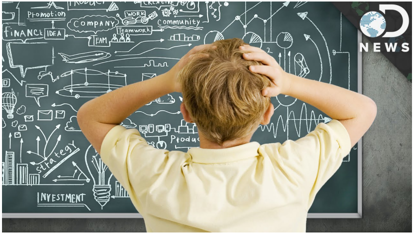
$\underbrace{O(n \log n)}_{\texttt{time}}$ $\underbrace{O(1)}_{\texttt{space}}$

$\underbrace{O(n)}_{\texttt{time}}$ $\underbrace{O(1)}_{\texttt{space}}$

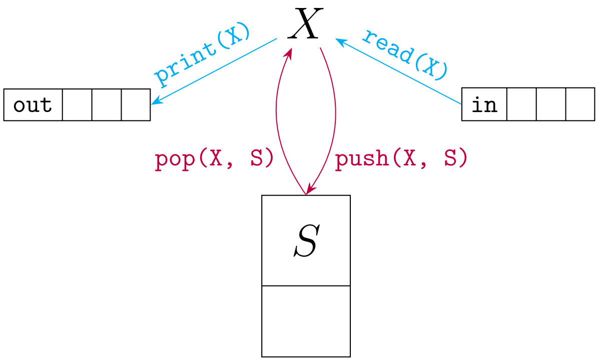


Stackable Permutations

Stackable Permutations

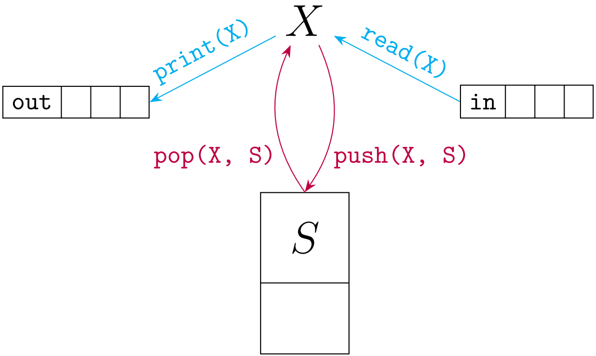


Definition (Stackable Permutations)

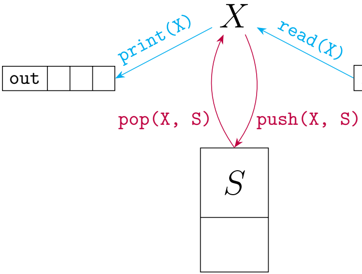


Definition (Stackable Permutations)

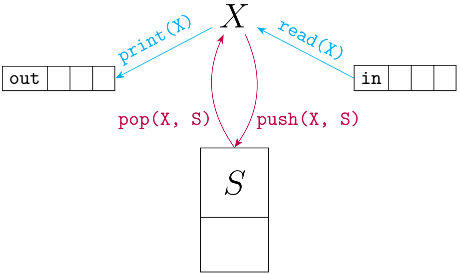
$$\boxed{\texttt{out} = (a_1, \cdots, a_n) \xleftarrow[X=0]{S=\emptyset} \texttt{in} = (1, \cdots, n)}$$



Definition (Stackable Permutations)

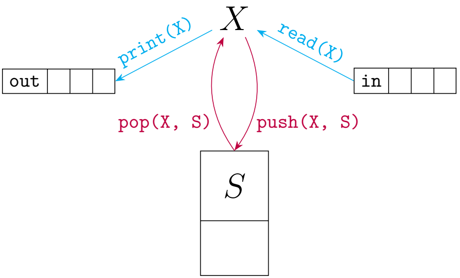


Definition (Stackable Permutations)



Q_1 : Meaning of "`read, print, push, pop`"?

Definition (Stackable Permutations)

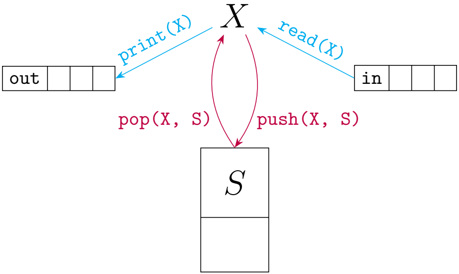


Q_1 : Meaning of "`read`, `print`, `push`, `pop`"?

Q_2 : Using only "`read`, `print`, `push`, `pop`"?

$$a == X$$

Definition (Stackable Permutations)

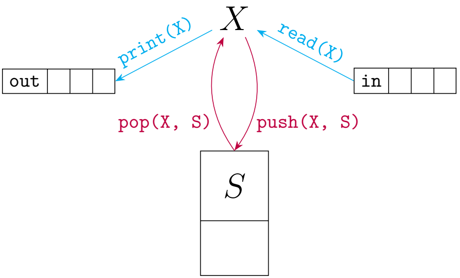


Q_1 : Meaning of "`read, print, push, pop`"?

Q_2 : Using only "`read, print, push, pop`"?

$$a == X \qquad a > X \ (a < X)$$

Definition (Stackable Permutations)



Q_1 : Meaning of "`read, print, push, pop`"?

Q_2 : Using only "`read, print, push, pop`"?

$$a == X \qquad a > X \ (a < X) \qquad \texttt{top}(S)$$

DH 2.12: Stackable Permutations

(a) **Show** that the following permutations *are* stackable:

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

(ii) $(3, 4, 2, 1)$

(iii) $(3, 5, 7, 6, 8, 4, 9, 2, 10, 1)$

DH 2.12: Stackable Permutations

(a) **Show** that the following permutations *are* stackable:

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

(ii) $(3, 4, 2, 1)$

(iii) $(3, 5, 7, 6, 8, 4, 9, 2, 10, 1)$

DH 2.12: Stackable Permutations

(a) **Show** that the following permutations *are* stackable:

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

(ii) $(3, 4, 2, 1)$

(iii) $(3, 5, 7, 6, 8, 4, 9, 2, 10, 1)$

DH 2.13: Stackable Permutations Checking Algorithm

To check whether a given permutation can be obtained by a stack.

`read` `print` `push` `pop` `is-empty`

```
X = 0    S = ∅    in != EOF
```

DH 2.13: Stackable Permutations Checking Algorithm

To check whether a given permutation can be obtained by a stack.

`read` `print` `push` `pop` `is-empty`

```
X = 0     S = ∅     in != EOF
```

```
foreach 'a' in out:
  if (! is-empty(S)
      && 'a' == top(S))
    pop(S, X)
    print(X)
  else ··· // T.B.C
```

DH 2.13: Stackable Permutations Checking Algorithm

To check whether a given permutation can be obtained by a stack.

read print push pop is-empty

X = 0   S = $\emptyset$ in != EOF

```
foreach 'a' in out:
  if (! is-empty(S)
      && 'a' == top(S))
    pop(S, X)
    print(X)
  else ··· // T.B.C
```

```
else // T.B.C
  while (in != EOF)
    read(X)
    if (X == 'a')
      print(X)
      break
    else
      push(X, S)
  ERR
```

DH 2.13: Stackable Permutations Checking Algorithm

To check whether a given permutation can be obtained by a stack.

`read` `print` `push` `pop` `is-empty`

```
X = 0     S = ∅     in != EOF
```

```
foreach 'a' in out:
  if (! is-empty(S)
      && 'a' == top(S))
    pop(S, X)
    print(X)
  else ··· // T.B.C
```

```
else // T.B.C
  while (in != EOF)
    read(X)
    if (X == 'a')
      print(X)
      break
    else
      push(X, S)
  ERR   // How???
```

DH 2.12: Stackable Permutations

(b) **Prove** that the following permutations are *not* stackable:

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

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

DH 2.12: Stackable Permutations

(b) **Prove** that the following permutations are *not* stackable:

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

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

DH 2.12: Stackable Permutations

(b) **Prove** that the following permutations are *not* stackable:

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

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

$$(3, 1, 2)$$

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

DH 2.12: Stackable Permutations

(b) **Prove** that the following permutations are *not* stackable:

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

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

$$(3, 1, 2)$$

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

$$\boxed{\texttt{out} = \cdots a_i \cdots a_j \cdots a_k : i < j < k \land a_j < a_k < a_i}$$

DH 2.12: Stackable Permutations

(b) **Prove** that the following permutations are *not* stackable:

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

$$(3, 1, 2)$$

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

$$\boxed{\texttt{out} = \cdots a_i \cdots a_j \cdots a_k : i < j < k \land a_j < a_k < a_i}$$

312-Pattern

Theorem (Stackable Permutations)

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

$$312\text{-}\textit{Pattern}: \boxed{\textit{out} = \cdots a_i \cdots a_j \cdots a_k : i < j < k \land a_j < a_k < a_i}$$

Theorem (Stackable Permutations)

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

$$312\text{-}\mathit{Pattern}: \boxed{\mathit{out} = \cdots a_i \cdots a_j \cdots a_k : i < j < k \land a_j < a_k < a_i}$$

Proof.

NO PROOF WARRANTY

$\square$

DH 2.12: Stackable Permutations

(c) How many permutations of A_4 *cannot* be obtained by a stack?

$$(1,4,2,3),(2,4,1,3),(3,1,2,4),(3,1,4,2),(3,4,1,2)$$
$$(4,1,2,3),(4,1,3,2),(4,2,1,3),(4,2,3,1),(4,3,1,2)$$

DH 2.12: Stackable Permutations

(c) How many permutations of A_4 *cannot* be obtained by a stack?

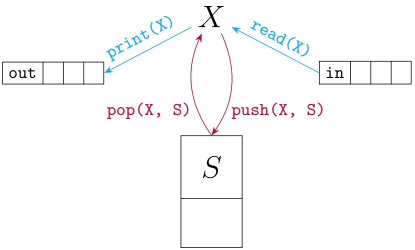
$$(1, 4, 2, 3), (2, 4, 1, 3), (3, 1, 2, 4), (3, 1, 4, 2), (3, 4, 1, 2)$$
$$(4, 1, 2, 3), (4, 1, 3, 2), (4, 2, 1, 3), (4, 2, 3, 1), (4, 3, 1, 2)$$

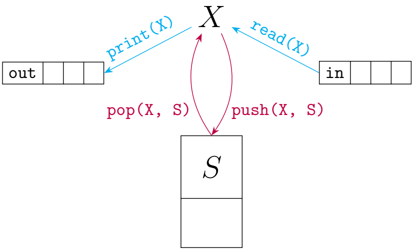
DH 2.12: Stackable Permutations

(c) How many permutations of A_4 *cannot* be obtained by a stack?

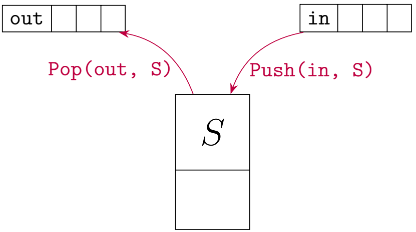
$$(1, 4, 2, 3), (2, 4, 1, 3), (3, 1, 2, 4), (3, 1, 4, 2), (3, 4, 1, 2)$$
$$(4, 1, 2, 3), (4, 1, 3, 2), (4, 2, 1, 3), (4, 2, 3, 1), (4, 3, 1, 2)$$

Q : What about A_n?

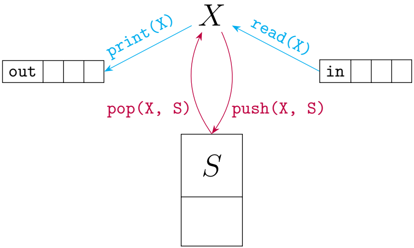

X
print(X)
read(X)
out
in
pop(X, S)
push(X, S)
S

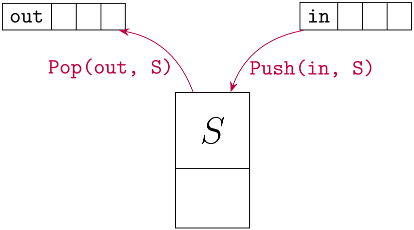


X
print(X)
read(X)
out
in
pop(X, S)
push(X, S)
S

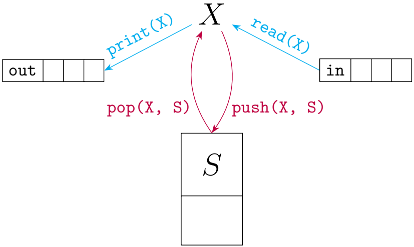


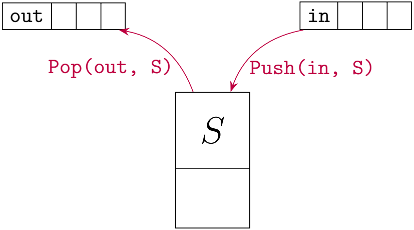
out
in
Pop(out, S)
Push(in, S)
S



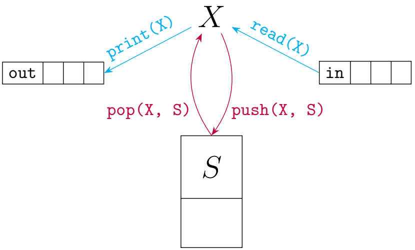


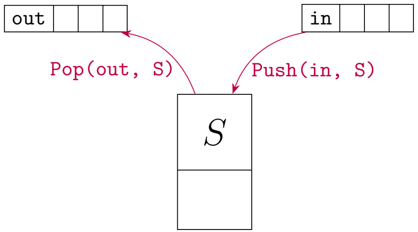
Q : Are $S + X$ and S are **equivalent**?





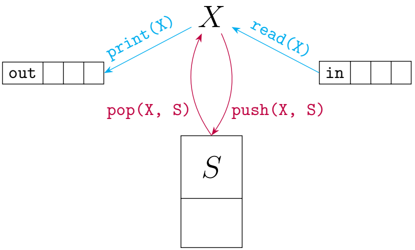
Q : Are $S + X$ and S are **equivalent**?

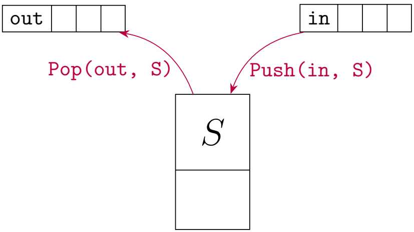




Q : Are $S + X$ and S are **equivalent**?

Producing the same set of permutations.

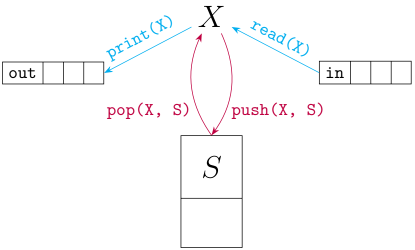


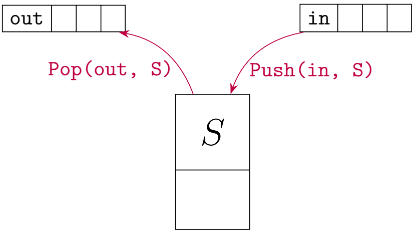


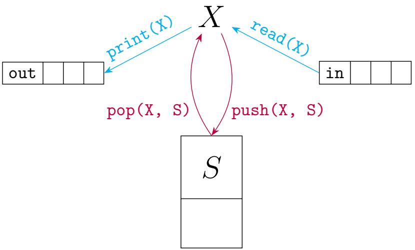
Q : Are $S + X$ and S are **equivalent**?

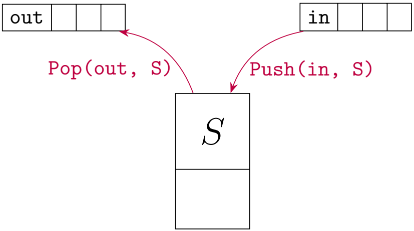
Producing the same set of permutations.

Accepting the same set of *admissible* operation sequences.

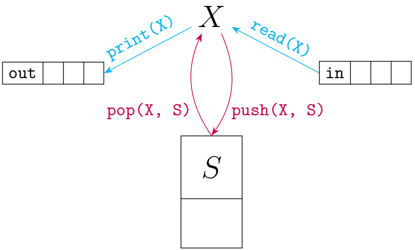

X
print(X)
read(X)
out
in
pop(X, S)
push(X, S)
S

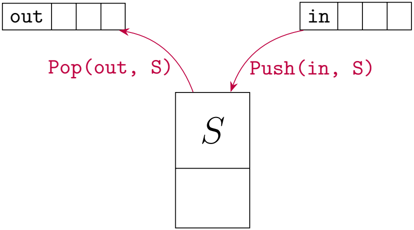

out
in
Pop(out, S)
Push(in, S)
S





By simulations.

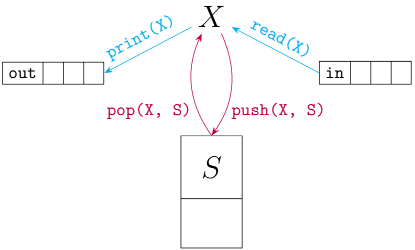


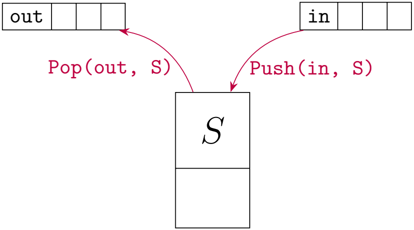


By simulations.

Simulate S by $S + X$:

- Push
- Pop



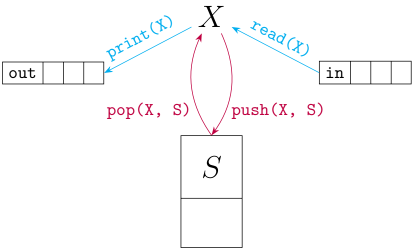


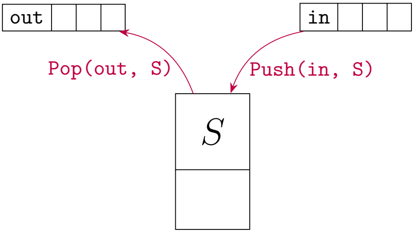
By simulations.

Simulate S by $S + X$:

- Push
- Pop

Simulate $S + X$ by S:





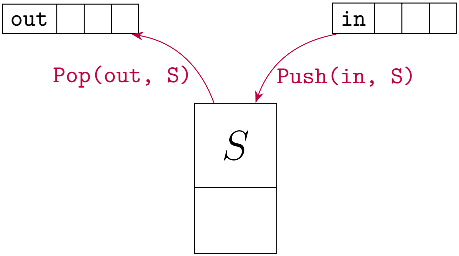
By simulations.

Simulate S by $S + X$:

- Push
- Pop

Simulate $S + X$ by S:

By iterative transformations.

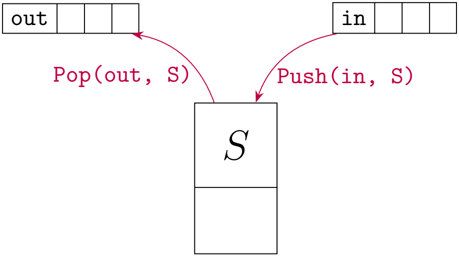


$(1, 2, 3)$: Push Pop Push Pop Push Pop

$(3, 2, 1)$: Push Push Push Pop Pop Pop

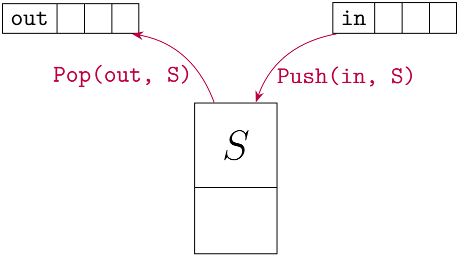
DH 2.12: Stackable Permutations

How many permutations of $\{1 \cdots n\}$ are stackable on the model S?



DH 2.12: Stackable Permutations

How many permutations of $\{1 \cdots n\}$ are stackable on the model S?



Q : How many *admissible* operation sequences of "Push" and "Pop"?

Definition (Admissible Operation Sequences)

An operation sequence of "`Push`" and "`Pop`" is *admissible* if and only if

Definition (Admissible Operation Sequences)

An operation sequence of "Push" and "Pop" is *admissible* if and only if

(i) # of "Push" $= n$ # of "Pop" $= n$

Definition (Admissible Operation Sequences)

An operation sequence of "`Push`" and "`Pop`" is *admissible* if and only if

(i) $\#$ of "$\texttt{Push}$" $= n$ $\qquad$ $\#$ of "$\texttt{Pop}$" $= n$

(ii) $\forall$ prefix : $(\#$ of "$\texttt{Pop}$"$) \leq (\#$ of "$\texttt{Push}$"$)$

Definition (Admissible Operation Sequences)

An operation sequence of "Push" and "Pop" is *admissible* if and only if

(i) # of "Push" $= n$ # of "Pop" $= n$

(ii) $\forall$ prefix : (# of "Pop") $\leq$ (# of "Push")

$$\boxed{\text{\# of stackable perms} = \text{\# of admissible operation sequences}}$$

Definition (Admissible Operation Sequences)

An operation sequence of "Push" and "Pop" is *admissible* if and only if

(i) # of "Push" $= n$ # of "Pop" $= n$

(ii) $\forall$ prefix : (# of "Pop") $\leq$ (# of "Push")

$$\boxed{\text{\# of stackable perms} = \text{\# of admissible operation sequences}}$$

Theorem

Different admissible operation sequences correspond to different permutations.

Theorem

Different admissible operation sequences correspond to different permutations.

Proof.

$$\texttt{Push Push Push Pop Pop } \textcolor{red}{\texttt{Push}} \cdots$$

$$\texttt{Push Push Push Pop Pop } \textcolor{red}{\texttt{Pop}} \cdots$$

□

Theorem

The number of admissible operation sequences of "Push" and "Pop" is $\binom{2n}{n} - \binom{2n}{n-1}$.

Theorem

The number of admissible operation sequences of "Push" and "Pop" is $\binom{2n}{n} - \binom{2n}{n-1}$.

Proof: The Reflection Method.

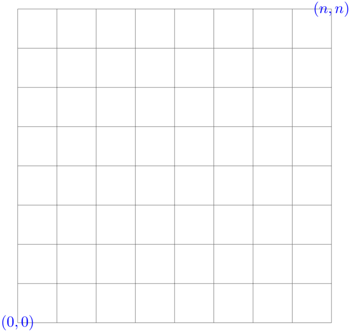
$$\texttt{Push} : \rightarrow \qquad \texttt{Pop} : \uparrow$$

Theorem

The number of admissible operation sequences of "Push" and "Pop" is $\binom{2n}{n} - \binom{2n}{n-1}$.

Proof: The Reflection Method.

$$\texttt{Push} : \rightarrow \qquad \texttt{Pop} : \uparrow$$



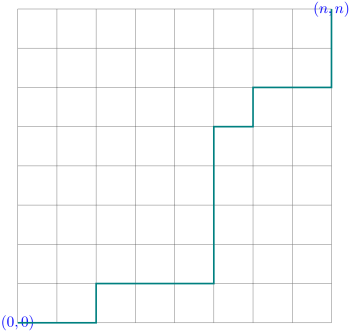
□

Theorem

The number of admissible operation sequences of "Push" and "Pop" is $\binom{2n}{n} - \binom{2n}{n-1}$.

Proof: The Reflection Method.

$$\texttt{Push} : \rightarrow \qquad \texttt{Pop} : \uparrow$$



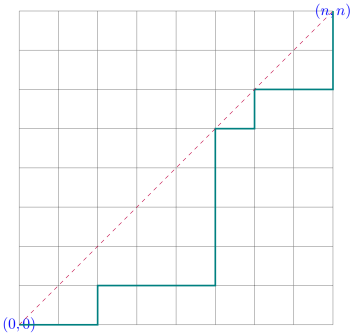
$\square$

Theorem

The number of admissible operation sequences of "Push" and "Pop" is $\binom{2n}{n} - \binom{2n}{n-1}$.

Proof: The Reflection Method.

$$\texttt{Push} : \rightarrow \qquad \texttt{Pop} : \uparrow$$

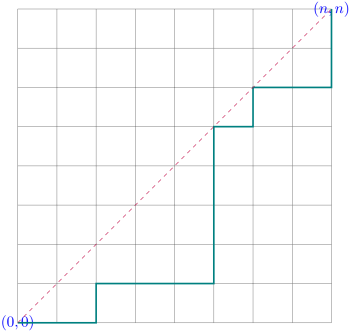


Theorem

The number of admissible operation sequences of "Push" and "Pop" is $\binom{2n}{n} - \binom{2n}{n-1}$.

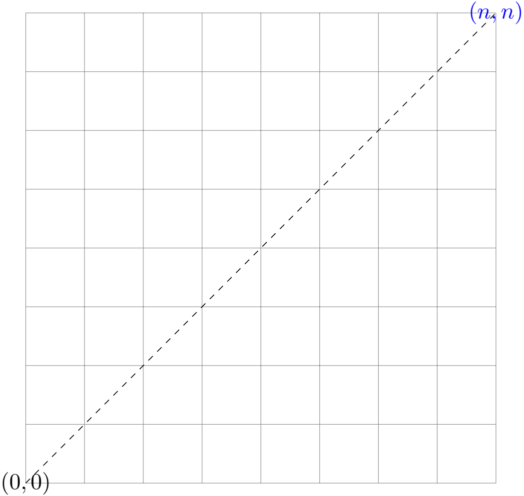
Proof: The Reflection Method.

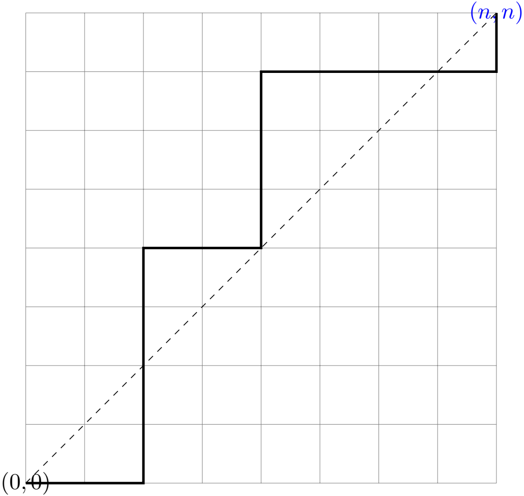
$$\texttt{Push} : \rightarrow \qquad \texttt{Pop} : \uparrow$$



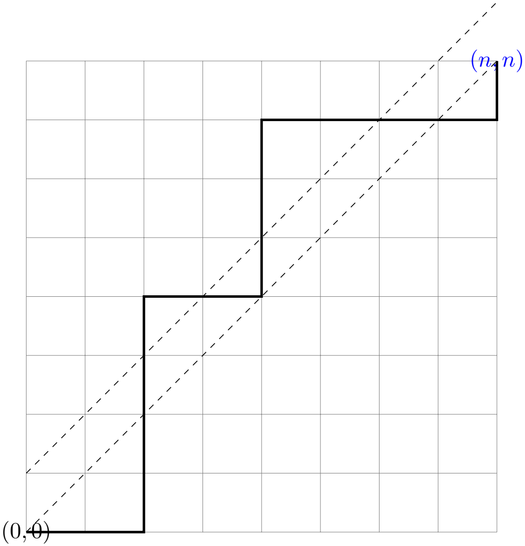
$$\underbrace{\binom{2n}{n}}_{\text{all}} - \underbrace{\binom{2n}{n-1}}_{\text{inadmissible}}$$

□

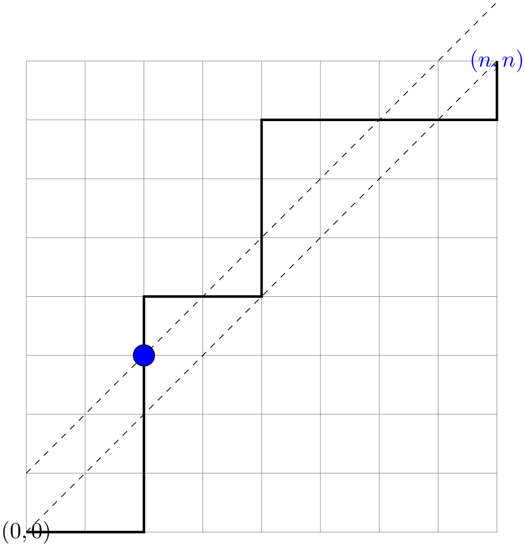

(n, n)
$(0, 0)$

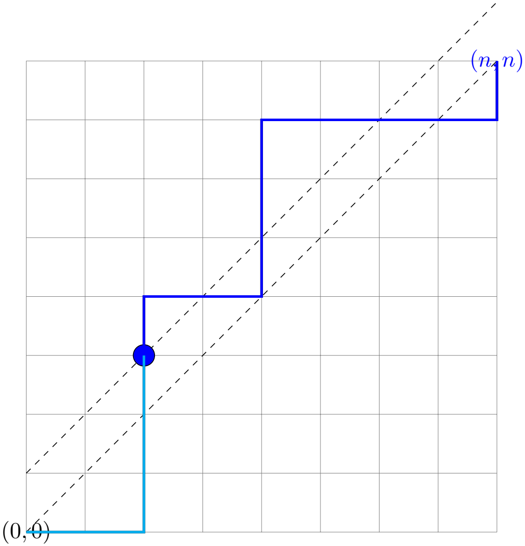


(n, n)
$(0, 0)$

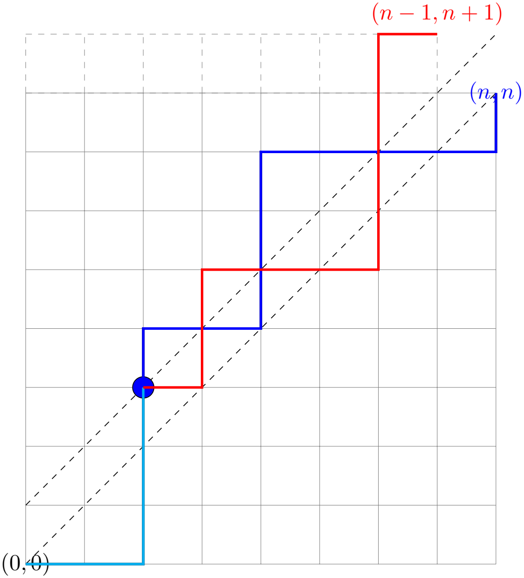


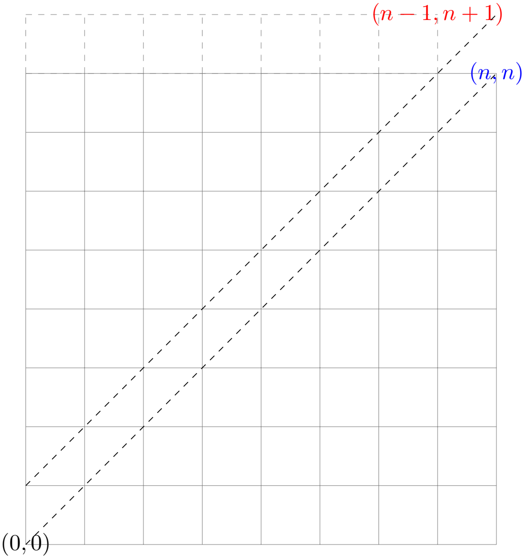
(n, n)
$(0, 0)$

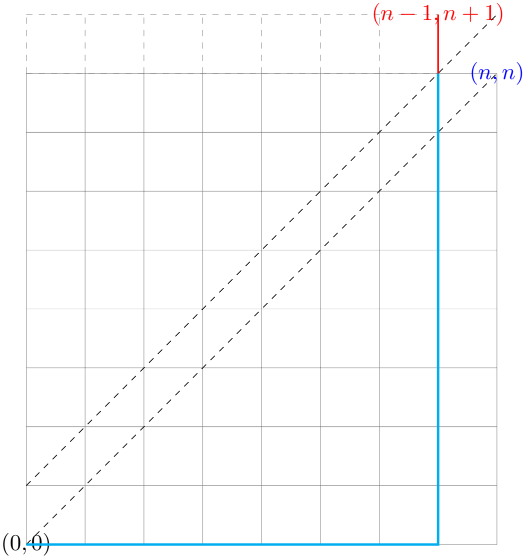

(n, n)
$(0, 0)$

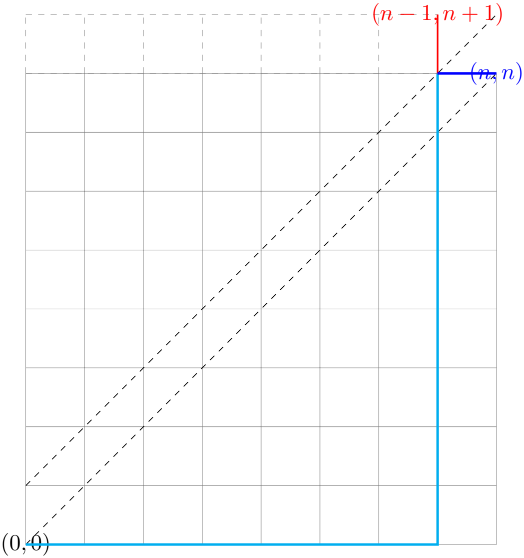


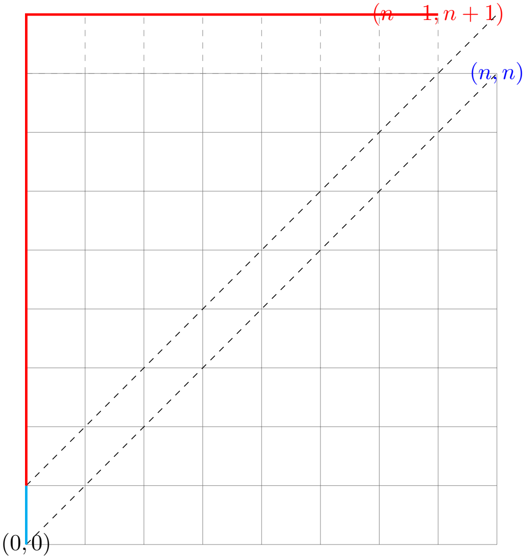
(n, n)
$(0, 0)$

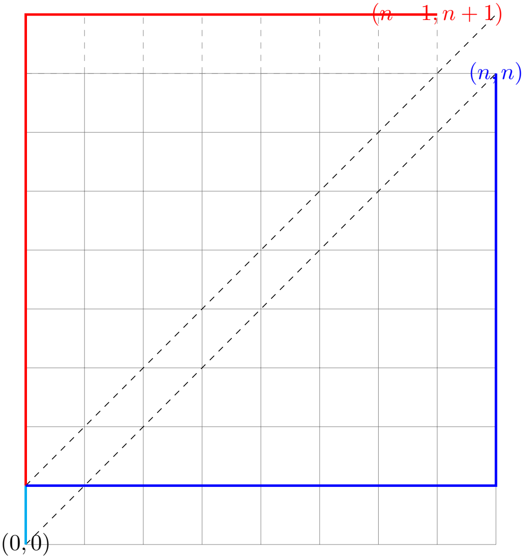

$(n-1, n+1)$
(n, n)
$(0, 0)$


$(n-1, n+1)$
(n, n)
$(0, 0)$


$(n-1, n+1)$
(n, n)
$(0, 0)$


$(n-1, n+1)$
(n, n)
$(0, 0)$

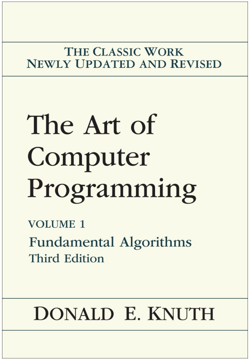

$(n-1, n+1)$
(n, n)
$(0, 0)$


$(n-1, n+1)$
(n, n)
$(0, 0)$

Catalan Number

$$(3, 2, 1) : ((())) \qquad (1, 2, 3) : ()()()$$

For more about "Stackable Permutations" (Section $2.2.1$):




Thank You!