Lecture 10

30 October 2018

Unit 25: **Tower of Hanoi**
Unit 26: **Permutation**
Unit 27: **N Queens**

PE2

Saturday, 10 November, 2018
1pm - 4pm

PE2

Covers until Lecture 9
& Assignment 7

PE 1

still grading 😭

PE 1

Prioritising grading of assignments over PE1

Lecture 11

Video Lecture Only

Lecture 11

Recording
next Tuesday
4pm - 6pm
(same venue)

My Office Hour

Not Available Tomorrow

Better Code Design

1.

Avoid Long Complex Functions

2.

A function should do one thing and one thing only.

```
void max(long list[], long length)
{
  long max_so_far = list[0];
  for (long i = 1; i != length; i += 1) {
    if (list[i] > max_so_far) {
      max_so_far = list[i];
    }
  }
  cs1010_println_long(max_so_far);
}
```

can I reuse this in selection sort?

3.

Avoid duplicate code

```
double ratio1 = a / (a + b + c + d);
double ratio2 = b / (a + b + c + d);
double ratio3 = c / (a + b + c + d);
double ratio4 = d / (a + b + c + d);
double sum = (a + b + c + d);
double ratio2 = b / sum;
double ratio3 = c / sum;
double ratio4 = d / sum;
double metered_fare(long distance)
{
  double fare = 3.40;

  distance -= 1000;
  if (distance <= 0) {
    return fare;
  }

  if (distance <= 9200) {
    fare += 0.22 * (distance / 400);
    if (distance % 400 > 0) {
      fare += 0.22;
    }
  } else {
    fare += 0.22 * (9200 / 400);
  }

  distance -= 9200;
  if (distance <= 0) {
    return fare;
  }

  fare += 0.22 * (distance / 350);
  if (distance % 350 > 0) {
    fare += 0.22;
  }

  return fare;
}
```

```
double metered_fare(long distance)
{
  double fare = 3.40;

  distance -= 1000;
  if (distance <= 0) {
    return fare;
  }

  if (distance <= 9200) {
    fare += 0.22 * (distance / 400);
    if (distance % 400 > 0) {
      fare += 0.22;
    }
  } else {
    fare += 0.22 * (9200 / 400);
  }

  distance -= 9200;
  if (distance <= 0) {
    return fare;
  }

  fare += 0.22 * (distance / 350);
  if (distance % 350 > 0) {
    fare += 0.22;
  }

  return fare;
}
```

```
double remaining_fare(long distance, long unit, double fare_per_unit)
{
   double fare = 0;
   fare = fare_per_unit * (distance / unit);
   if (distance % unit > 0) {
      fare += fare_per_unit;
   }
   return fare;
}

   :

  distance -= 1000;
  if (distance <= 0) {
    return fare;
  }

  if (distance <= 9200) {
    fare += remaining_fare(distance, 400, 0.22);
   } else {
    fare += 0.22 * (9200 / 400);
  }

  distance -= 9200;
  if (distance <= 0) {
    return fare;
  }

  fare += remaining_fare(distance, 350, 0.22);

   :
```

70%

of your as05 contain memory bugs

😱

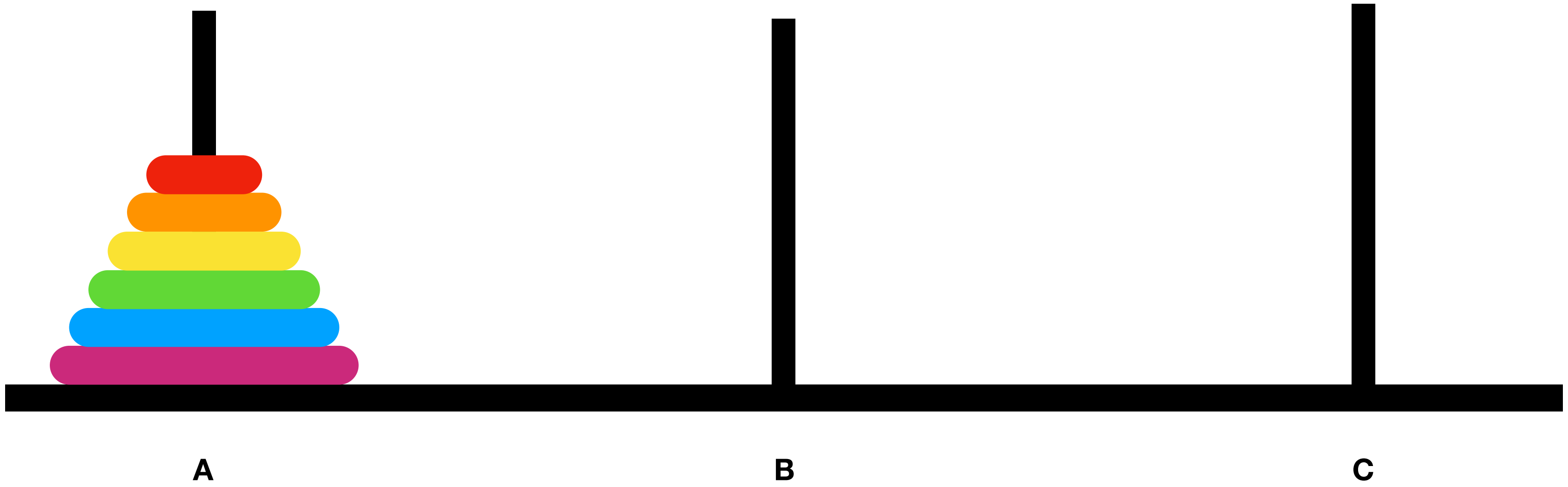
```
world = calloc(rows, sizeof(char*));
for (long i = 0; i < rows; i += 1) {
  world[i] = calloc(columns + 1, sizeof(char));
  world[i] = cs1010_read_word();
}
```

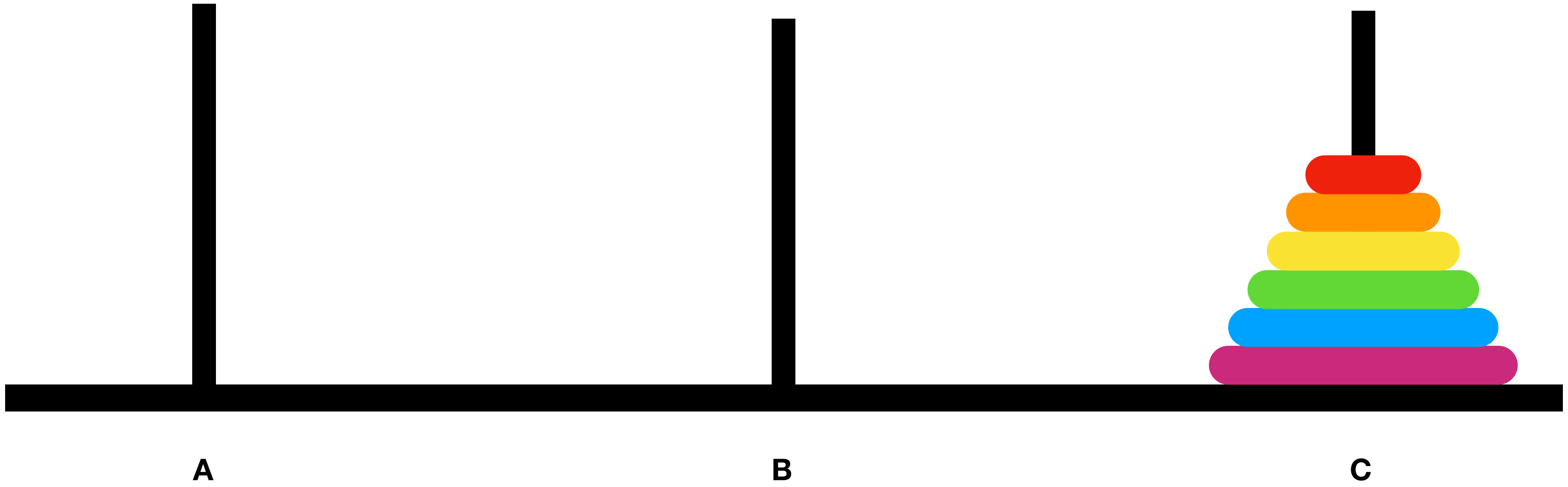
```
network = calloc(rows, sizeof(char*));
for (long i = 0; i < rows; i += 1) {
  network[i] = cs1010_read_word();
}
```

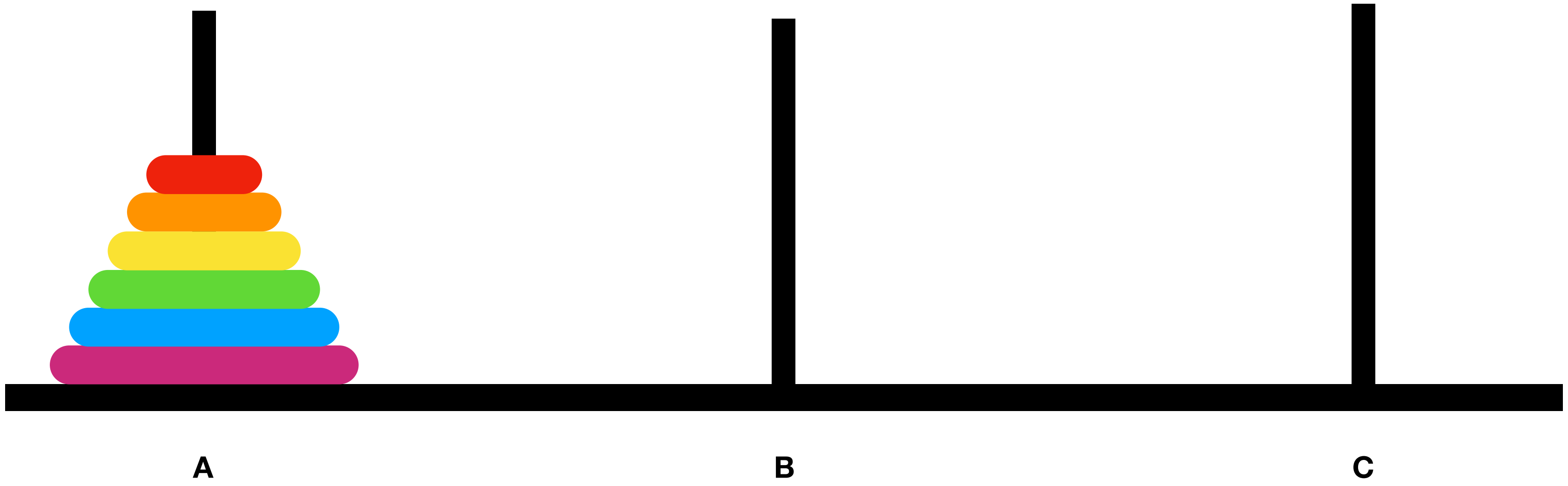
```
network[i][j] == ‘1’ ?
```

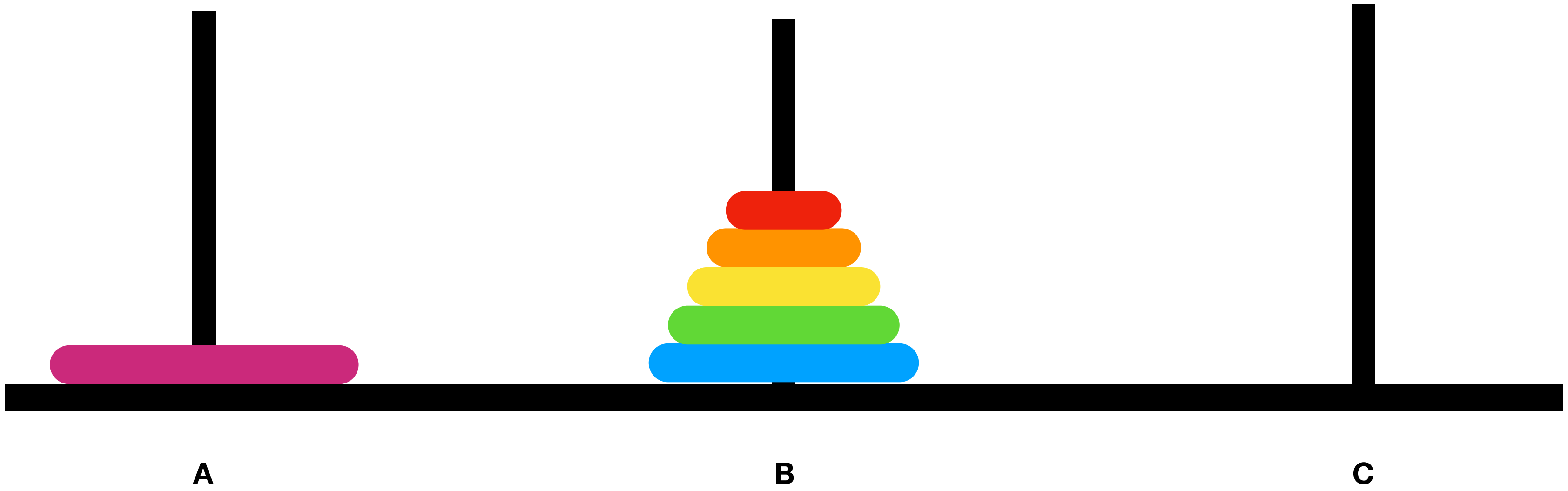
Problem Solving with Recursions

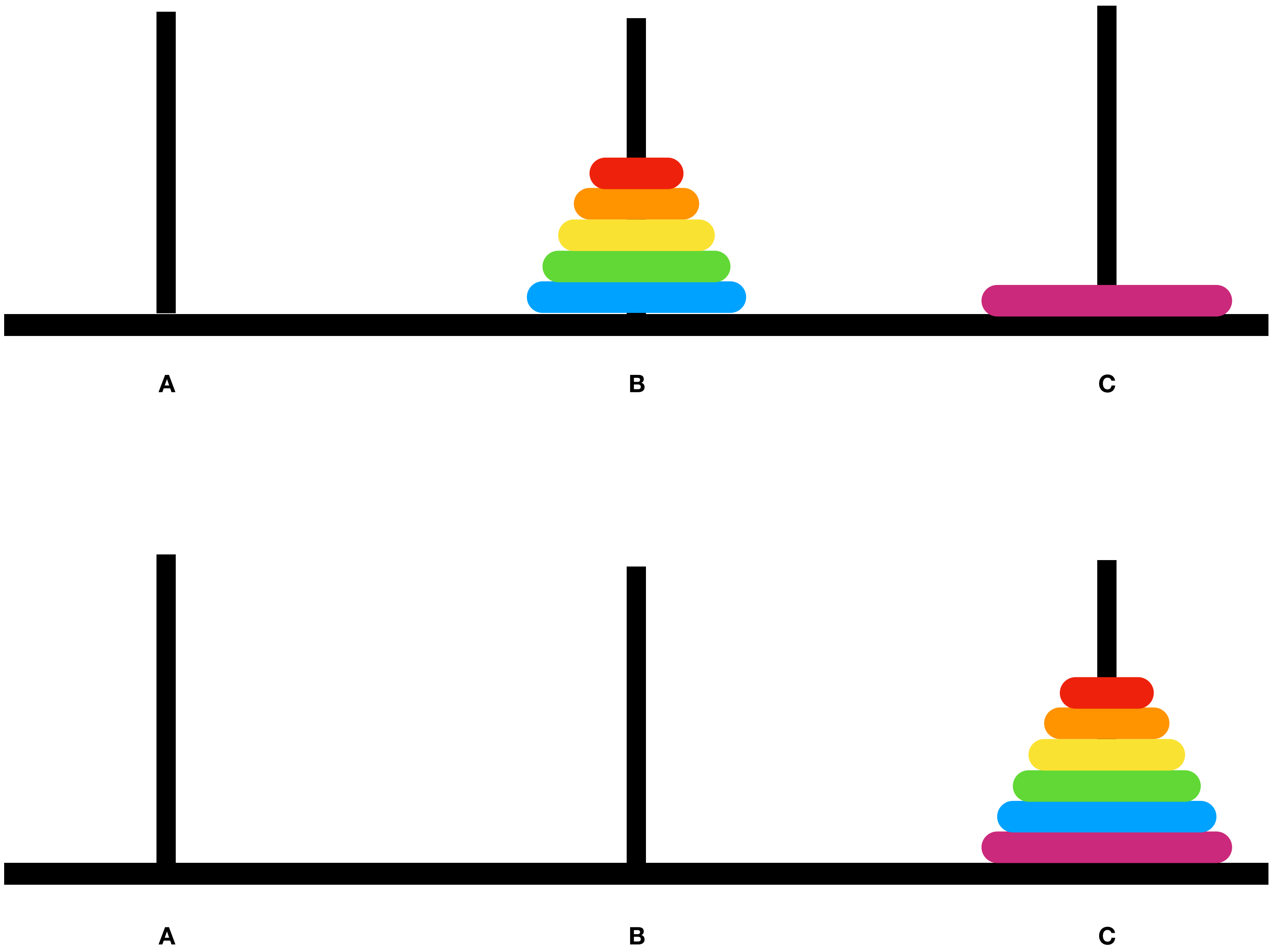
Tower of Hanoi


A
B
C

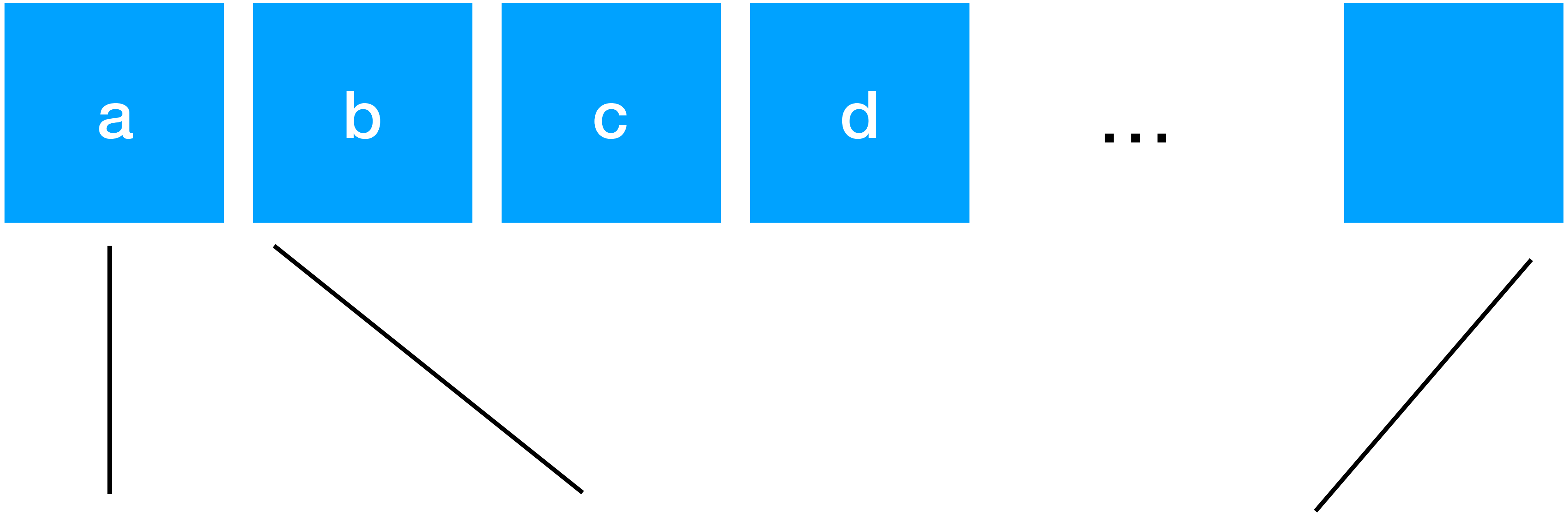

A
B
C


A
B
C


A
B
C

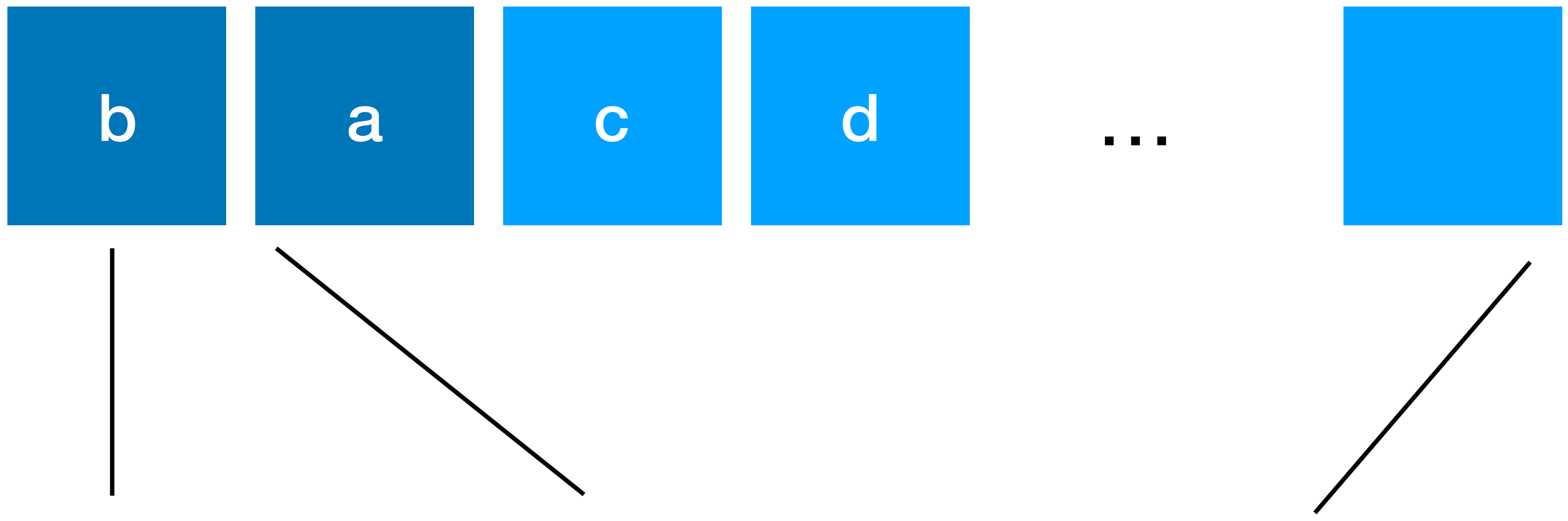

A
B
C
A
B
C

Permutation



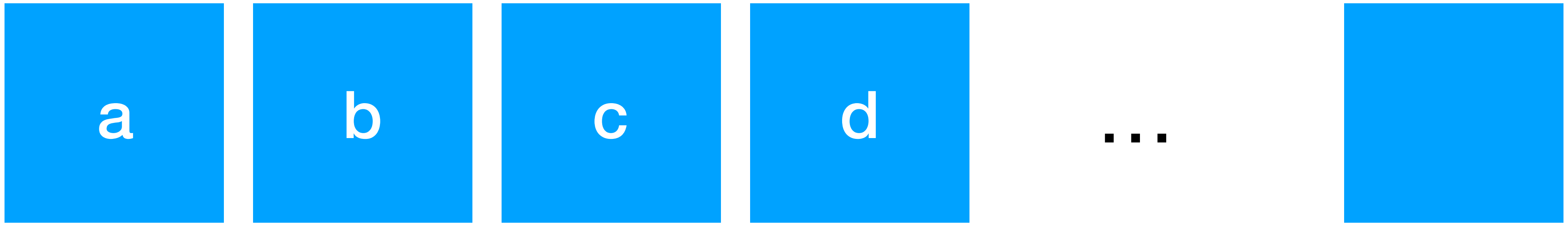
Fixed

Generate all permutations of length n - 1

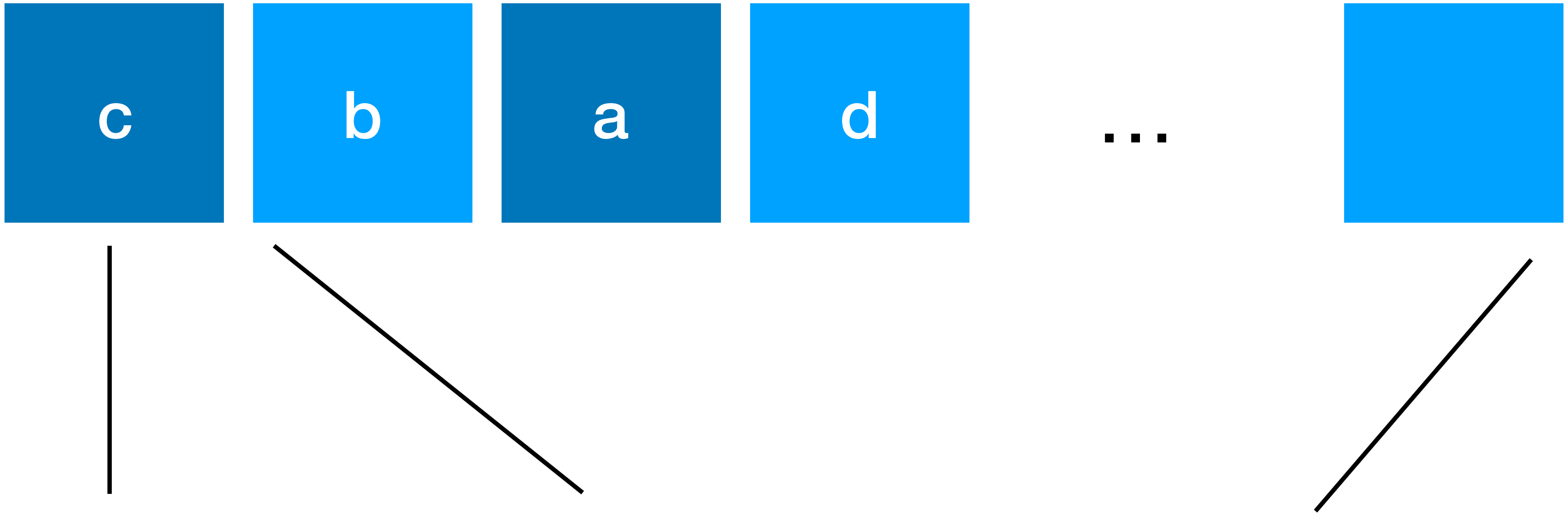


Fixed

Generate all permutations
of length n - 1



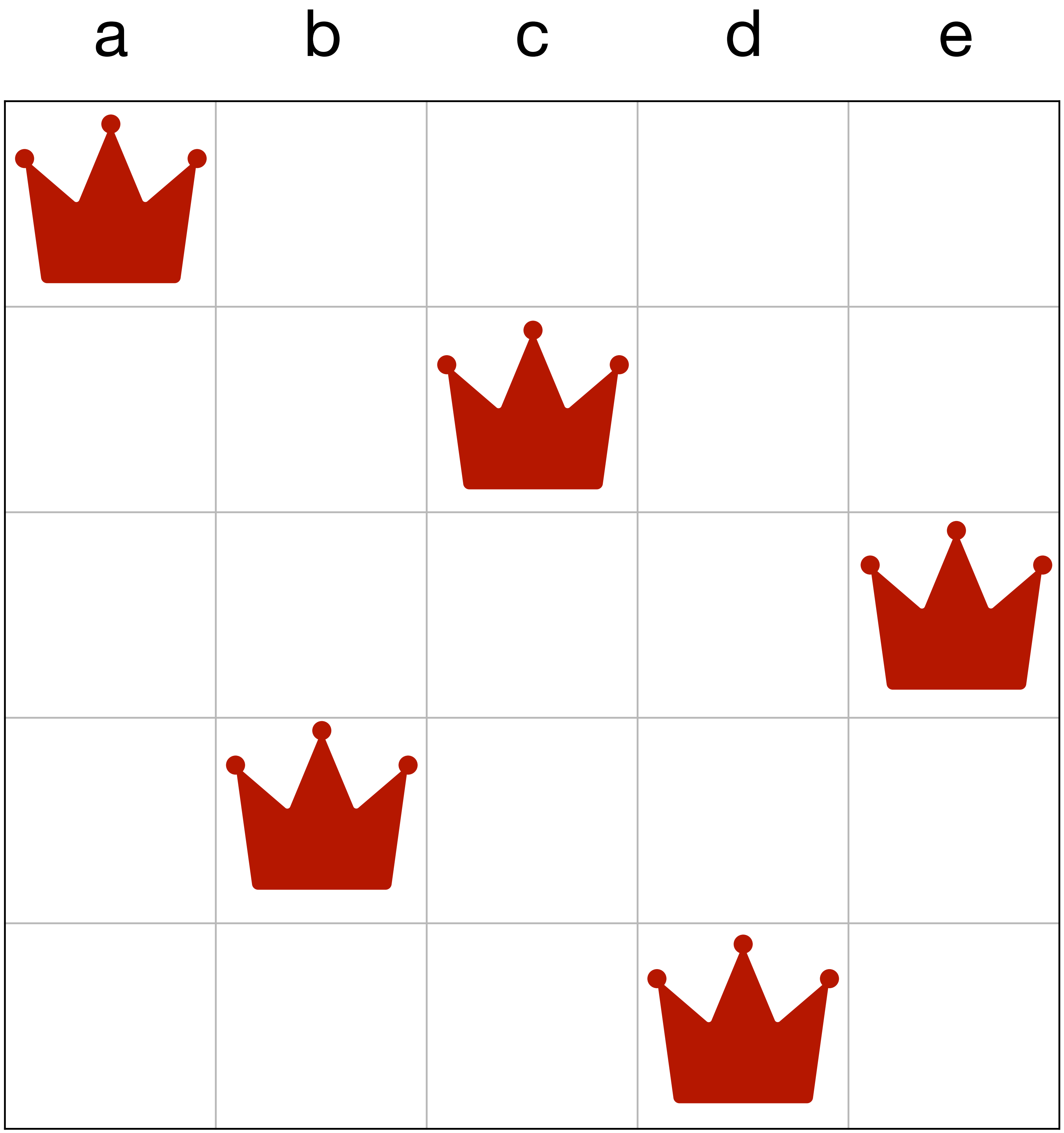
a b c d ...

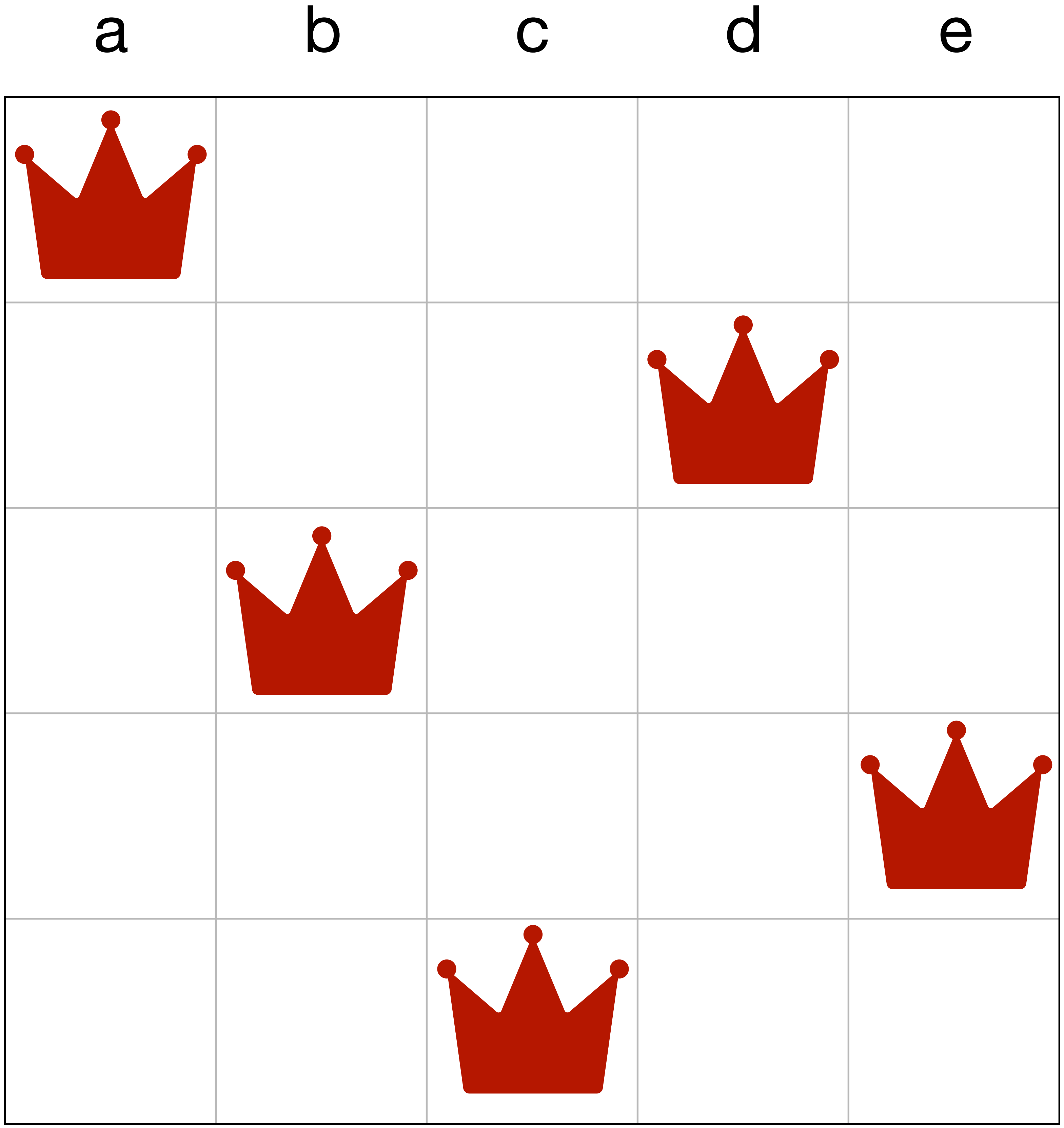


Fixed

Generate all permutations of length n - 1

N Queens


a
b
c
d
e


a
b
c
d
e

Approach 1: generate all permutation and test

Approach 2:

avoid redundant work