CSE216
Foundations of Computer Science

Instructor: Zhoulai Fu

State University of New York, Korea

Plan

- Slowing down a bit before moving forward
- A lab exercise
- Hanoi explained
- Exercises for better understanding function types

Warm-up Lab Exercise

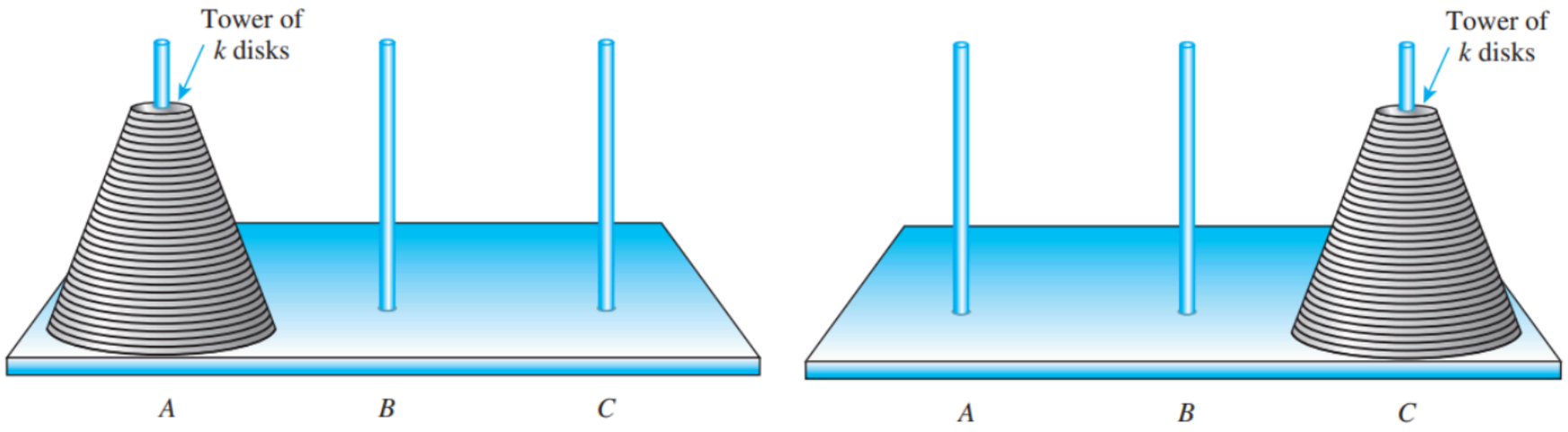
- Write a power function of type **power: int -> int -> int** using only recursion, without using “**” or other build-in functions in Ocaml

Hanoi Explained

Example: Towers of Hanoi

Problem

- There are k disks on peg 1. Your aim is to move all k disks from peg 1 to peg 3 with the minimum number of moves. You can use peg 2 as an auxiliary peg. The constraint of the puzzle is that at any time, you cannot place a larger disk on a smaller disk.
  What is the minimum number of moves required to transfer all k disks from peg 1 to peg 3?



Demo: https://www.mathsisfun.com/games/towerofhanoi.html

Example: Towers of Hanoi

Solution

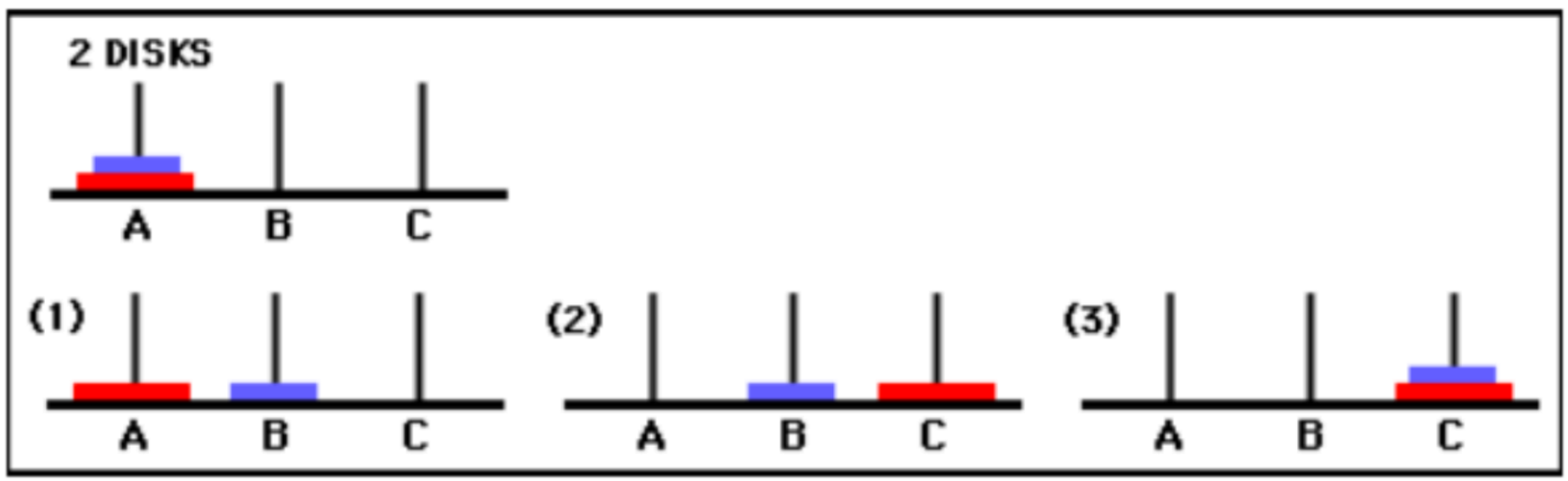
Suppose $k = 1$. Then, the 1-step solution is:
1. Move disk 1 from peg A to peg C.

Example: Towers of Hanoi

Solution

Suppose $k = 2$. Then, the 3-step solution is:

1. Move disk 1 from peg A to peg B.
2. Move disk 2 from peg A to peg C.
3. Move disk 1 from peg B to peg C.

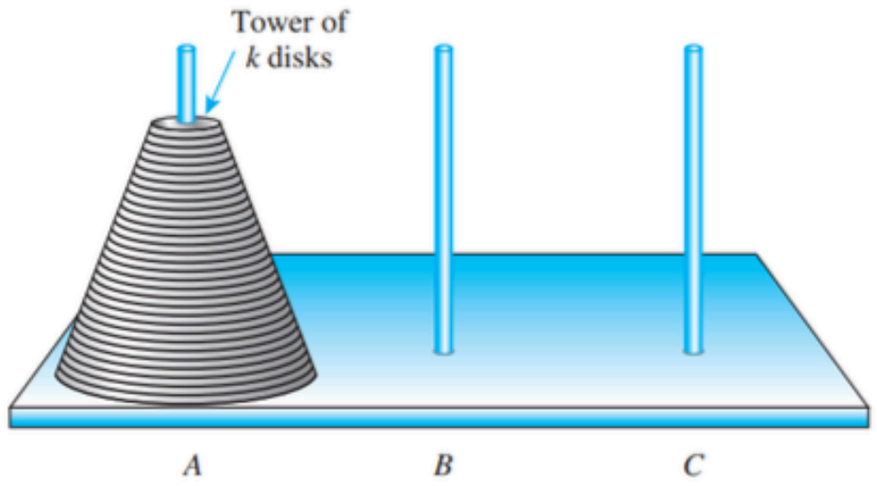


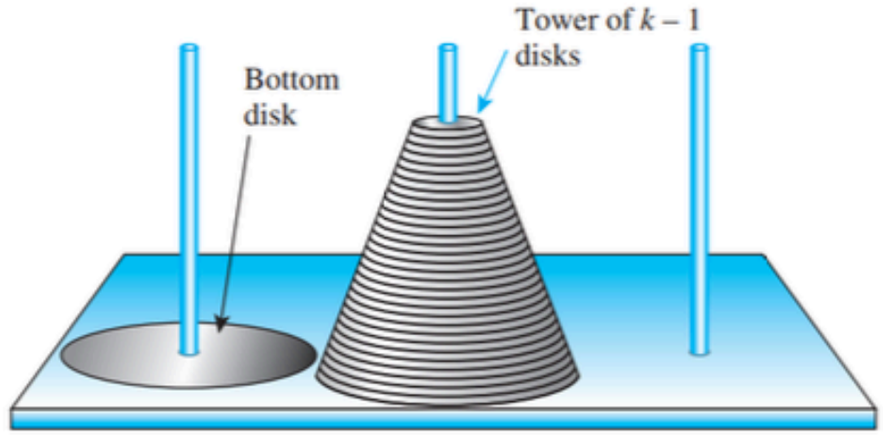
Example: Towers of Hanoi

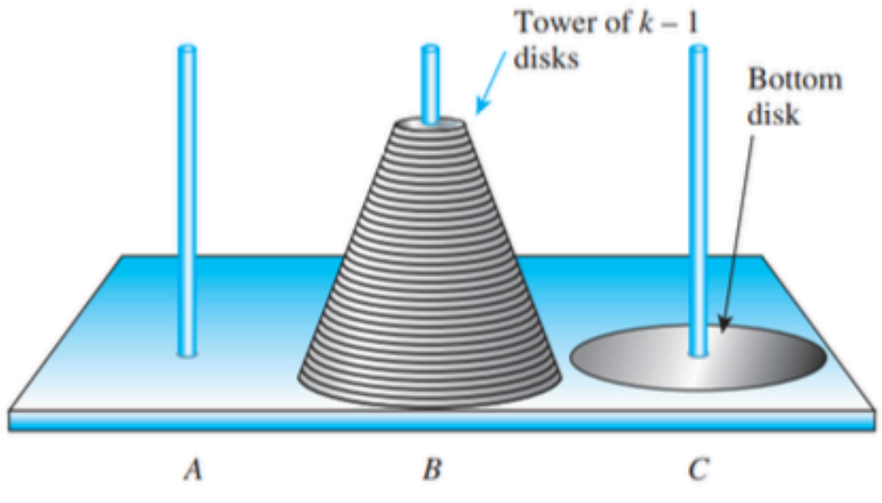
Solution

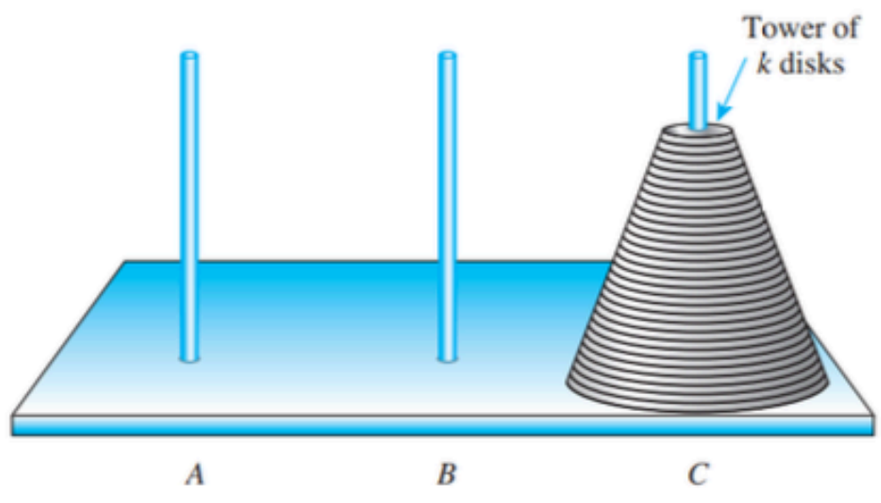
For any $k \geq 2$, the recursive solution is:

1. Transfer the top $k-1$ disks from peg A to peg B.
2. Move the bottom disk from peg A to peg C.
3. Transfer the top $k-1$ disks from peg B to peg C.





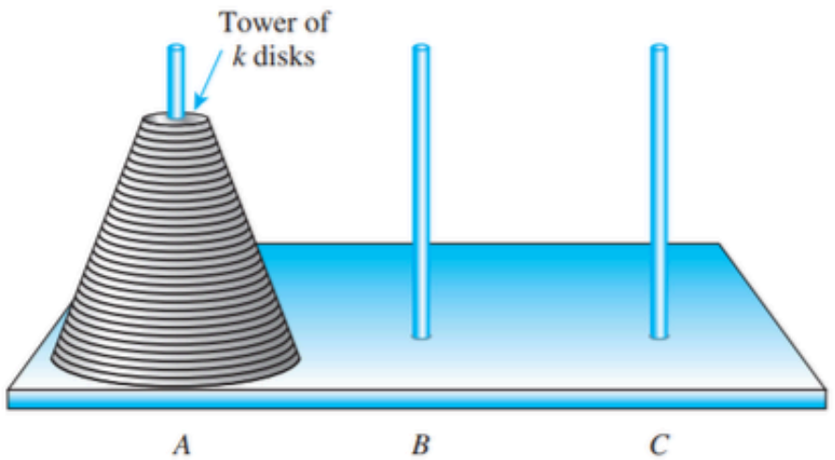


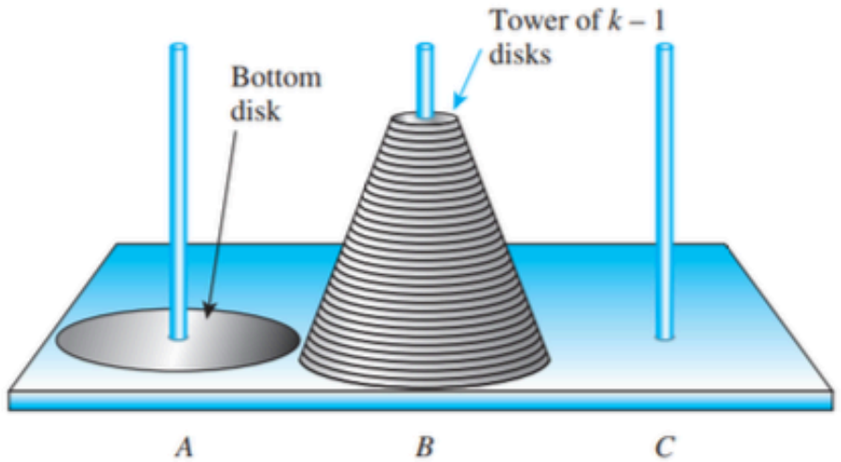


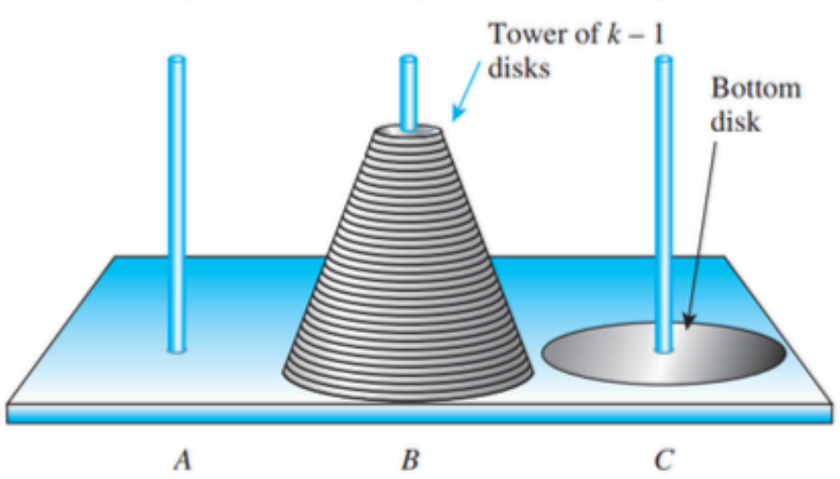
Example: Towers of Hanoi

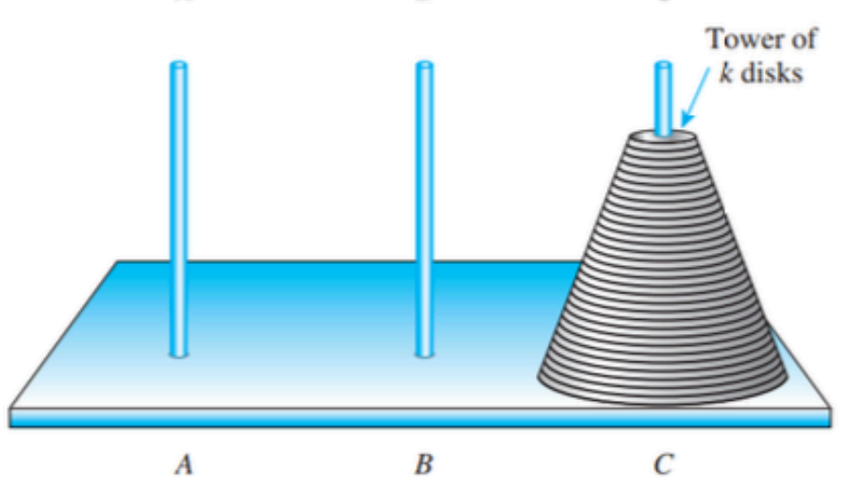
TOWERS-OF-HANOI(k, A, C, B)

1. if $k = 1$ then
2. Move disk k from A to C.
3. elseif $k \geq 2$ then
4. TOWERS-OF-HANOI$(k-1, A, B, C)$
5. Move disk k from A to C.
6. TOWERS-OF-HANOI$(k-1, B, C, A)$









Exercises on function types

Exercise 1 (done last time)

- Write down the types of the defined functions in OCaml:
  - a) let double x = 2*x;;
  - b) let square x = x*x;;
  - c) let twice f x = f (f x);;
  - d) let quad = twice double;;
  - e) let fourth = twice square;;

Exercises 2

- Write down the types of the defined functions in OCaml:
  - a) let tripleFloat x = 3.0*.x;;
  - b) let thrice f x = f(f(f(x)));;
  - c) let composition f g x = f(g(x));;
  - d) let div x y = x/y;;
  - e) let triple3 = thrice tripleFloat;;

Exercise 3

- Define twice such that takes f and x as an input, applies f to x a total of 2 times.

Exercise 4

- Generalize twice to a function repeat, such that repeat f n x applies f to x a total of n times. That is,
  - repeat f 0 x yields x
  - repeat f 1 x yields f x
  - repeat f 2 x yields f (f x) (which is the same as twice f x)
  - repeat f 3 x yields f (f (f x))

Exercise 5: Any Difference Here?

```
let abs = fun x -> if x<0 then -x else x

let abs (x:int) = fun x -> if x<0 then -x else x

let abs x = fun x -> if x<0 then -x else x

let abs (x:int) : int = if x<0 then -x else x

let abs:int -> int = fun x -> if x<0 then -x
else x
```

Exercise 6:
Determine function types

```
let rec p x y  =
  if y=0 then 1 else x * (p x (y-1)) ;;
```

```
let rec p (x, y) =
    if y=0 then 1 else x * p (x, y-1) ;;
```

Exercise 7: Where is the bug?

```
(*buggy*)
# let rec p x y  =
  if y=0 then 1 else x * (p x y-1) ;;
val p : int -> int -> int = <fun>
```

```
# p 3 2 ;;
Stack overflow during evaluation (looping
recursion?)
```