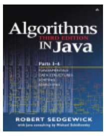
Week 1: references and exercises

- Building blocks for Java programs, collections
- Notable functions: factorial, Fibonacci
- Abstract data type, interface, client
- TADs bag, tail (file FIFO), stack
- Paintings
- Euclid's algorithm

References

-  [Sedgewick & Wayne 2011] §1.1 , §1.2, §1.3
-  [Sedgewick 2003] §3.1, §3.2, §4.1, §4.2, §4.7
- Java tutorial on collections

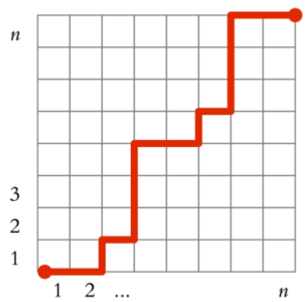
1. Exercises: maths

1.1 Double Factor

Use the Stirling formula to characterize the asymptotic growth of the double factorial $(2k+1)!! = 1 \cdot 3 \cdot 5 \cdot 7 \cdots (2k+1)$and $(2k)!! = 2 \cdot 4 \cdot 6 \cdot 8 \cdots (2k)$.
Hint : write $(2k)!! = k! \cdot \prod_{i=1}^{k} 2$and $(2k+1)!! = \frac{(2k+1)!}{(2k)!!}$, and use the Stirling formula $n! \sim \sqrt{2\pi n}(n/e)^n$.

1.2 Paths on a grid

A *monotonous way* on the grid $n \times n$is a sequence $(x_0, y_0), (x_1, y_1), \ldots, (x_{2n}, y_{2n})$with $(x_0, y_0) = (0, 0)$,$(x_{2n}, y_{2n}) = (n, n)$and $(x_{i+1}, y_{i+1}) \in \{(x_i + 1, y_i), (x_i, y_i + 1)\}$for all $i = 1, 2, \ldots, 2n - 1$. In other words, one can move either to the right or upward by a square.



- Show that the number of monotonic paths is $C(n) = \binom{2n}{n} = \frac{(2n)!}{n! \cdot n!}$.
- Using the Stirling formula to characterize the asymptotic growth of $C(n)$and $\ln C(n)$.

1.3 Number of Pell

Pell numbers are defined $P(n)$ for $n = 0, 1, 2 \ldots$
by $P(0) = 0, P(1) = 1$ and $P(n) = 2P(n-1) + P(n-2)$ for $n > 1$. Give an asymptotic characterization.

> **Hint** : For an elegant proof, introduce the generating function $p(x) = \sum_{n=0}^{\infty} P(n)x^n$, substitute the recurrences in the infinite sum, and find its solution as a sum of geometric sequences.

1.4 Euclid in binary

The largest common divisor is often computed by a binary version of Euclid's algorithm, discovered by Joseph Stein in 1967. The advantage of the binary version is that no whole division is needed, bit shift ($\times 2$ or $/2$), and subtraction. The algorithm uses the following recurrences:

1. $\gcd(x, y) = \gcd(y, x)$
2. if x and y are even, then $\gcd(x, y) = 2 \cdot \gcd(x/2, x/2)$
3. if x is even and y odd, then $\gcd(x, y) = \gcd(x/2, y)$
4. if $x \geq y$, then $\gcd(x, y) = \gcd(x - y, y)$

Give an efficient algorithm based on these rules which computes $\gcd(x, y)$ in a logarithmic time (bounded by $c \lg(ab)$).
Hint : Note that you must determine when to use which rule. In particular, it must be ensured that the fourth rule is not used too often.

2. Exercises: tail and stack

2.1 Tail with size management

Show a FIFO queue implementation based on an array with dynamic expansion / reduction.
Hint : Be careful when copying elements when a new table is allocated.

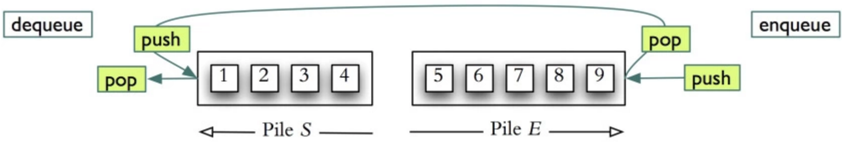
In 2.2 and 2.3, one wants to implant a tail (file FIFO) using one or two cells exclusively through the interface of the TA which supports the operations $\mathsf{push}(x)$, $\mathsf{pop}()$ and $\mathsf{isEmpty}()$, in a damped time bounded by a constant.

2.2 Cut the grass under the foot

Implant a tail (with operations $\mathsf{enqueue}(x)$, $\mathsf{dequeue}()$) using **a stack** . Analyze the calculation time, and the memory usage of the operations according to the size of the queue.
Hint : It is clear that we can implant $\mathsf{enqueue}$ by push. In order to implant $\mathsf{dequeue}$, one needs to access the bottom of the stack. So develop a recursive algorithm to remove an element at the bottom of the stack, using only push and pop.

2.3 Two is better

- Implant a queue (file FIFO) with the operations $\mathsf{enqueue}(x)$ and $\mathsf{dequeue}()$, using two stacks.
  Hint : Use both batteries to store items at both ends. Stack / stack ($E \rightarrow S$) only when it becomes necessary.



- Add the operation delete(k) that scrolls $k > 0$ or stops earlier when the queue is empty. The implantation should never lead to an overflow of the underlying piles. The operation does not return any value.
- What is the calculation time of the operations enqueue and delete(k) in the worst case? Give an implementation that ensures that the computation time is at most linear in m any sequence of m operations - that is, the amortized cost of operations is constant. Demonstrate a specific terminal.
  Hint : Set up a credit-debit proof.

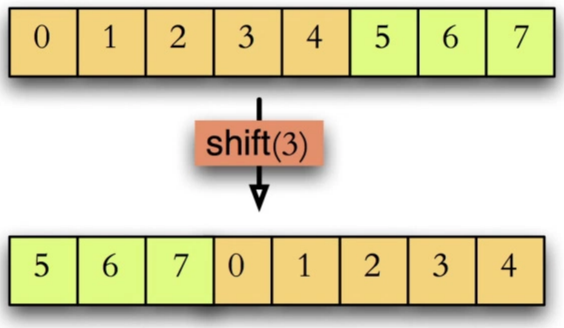
3. Exercises: tables

3.1 Local maximum

In a table $T[0..n-1]$, there is a local maximum at $0 < k < n-1$ si $T[k-1], T[k+1] < T[k]$. Give an algorithm that finds all local maxima.

3.2 Circular offset

Suppose we want to make circular shifts in a table $A[0..n-1]$. An offset by δ corresponds to the *parallel* execution of the assignments $A[(i+\delta) \bmod n] \leftarrow A[i]$.



When $\delta = 1$, it is clear that one can perform the offset in place by a simple loop:

```
SHIFT1 (A [0..n-1])
{
    i ⟵ 0; shuttle ⟵ A [0];
    do
    {
        i ⟵ (i + 1) mod n;
        exchange shuttle ⟷ A [i]
    } while (i≠0)
}
```

Give an algorithm $\mathsf{Shift}(A, \delta)$that executes a circular offset with δpositions **in place** . The algorithm must use a constant workspace (excluding input $A[0..n-1]$), and execute in a linear time (in n), for any choice of δ(even for pe, $\delta = n/3$).

Hint : What happens if you write $i \leftarrow (i+\delta) \bmod n$in the loop instead of $i \leftarrow (i+1) \bmod n$?