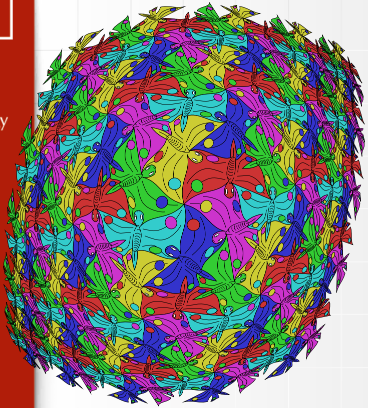
Wrocław
University
of Science
and Technology



Algorithmic Methods (Algorithms Part IV)

ver. 12 z drobnymi modyfikacjami!

Wojciech Myszka

2022-12-16 08:05:02 +0100

Wrocław University of Science and Technology

How to develop an algorithm?

My answer is very short:

How to develop an algorithm?

My answer is very short:

I do not know!

Searches and Traversal

1. Many algorithmic problems give rise to the need to **traverse** certain structures.
2. At times, the structure we have to traverse is present explicitly as one of the data structures defined in the algorithm (array, vector, tree. . .)
3. At times it is some implicit abstract structure that perhaps cannot be actually “seen,” but that exists under the surface.
4. When the task has a number of variants — is sufficient to traverse them all

In each case, the algorithm is reduced to review all elements of the structure or to find the most natural way of traversing the data structure at hand (whether explicit or implicit) and thus devise the algorithm.

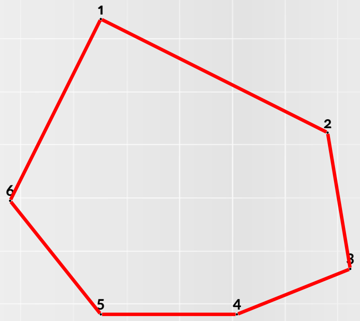
Loop

```
1. **Start**
2. $i \leftarrow 0$
3. *Do something…*
4. $i \leftarrow i + 1$
5. **If** $i \leq N$ **go to** step 3.
6. **Else** — **Stop**
```

Traversing

Example

- Suppose we are given a simple convex polygon,
- we are interested in finding two points of maximal distance on its borderline.

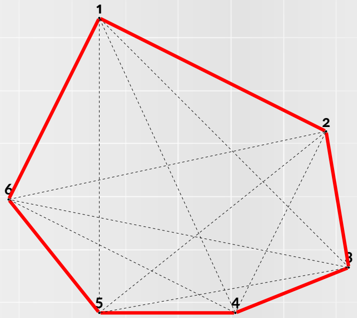


How to solve this problem?

Traversing

Example

- Suppose we are given a simple convex polygon,
- we are interested in finding two points of maximal distance on its borderline.



How to solve this problem?

Example cont

	1	2	3	4	5	6
1	0	11.2	15.6			
2	11.2	0				
3	15.6		0			
4				0	6	
5				6	0	6.4
6					6.4	0

Vertices:

1 (4,13)
2 (14,8)
3 (15,2)
4 (10,0)
5 (4,0)
6 (0,5)

Example cont

Simplest case

```
1.   max = 0
2.   **for** i = 1 to 6 **do**
       2.1  **for** j = 1 to 6 **do**
       2.2  **if** d[i, j] > max **then**
       2.3  max = d[i,j]
```

Example cont

Simplest case

```
1.  max = 0
2.  **for** i = 1 to N **do**
      2.1  **for** j = 1 to N **do**
      2.2  **if** d[i, j] > max **then**
      2.3  max = d[i,j]
```

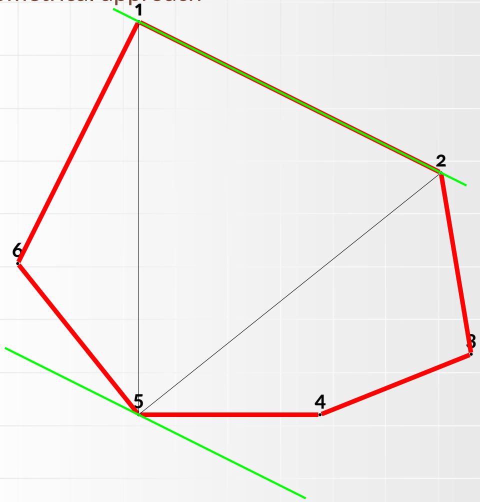
Example cont

Simplest case

```
1.  max = 0
2.  for i = 2 to N do
      2.1 for j = 1 to i — 1 do
      2.2 if d[i, j] > max then
      2.3 max = d[i,j]
```

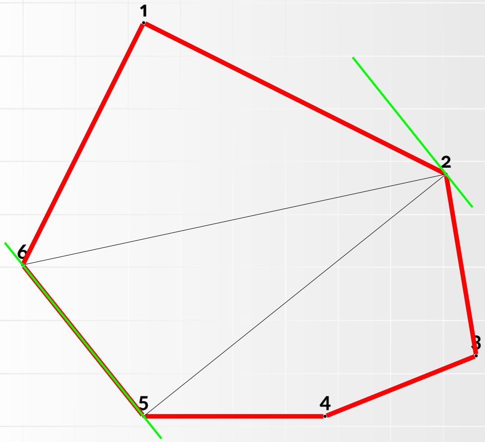
Example cont.

“Geometrical approach”



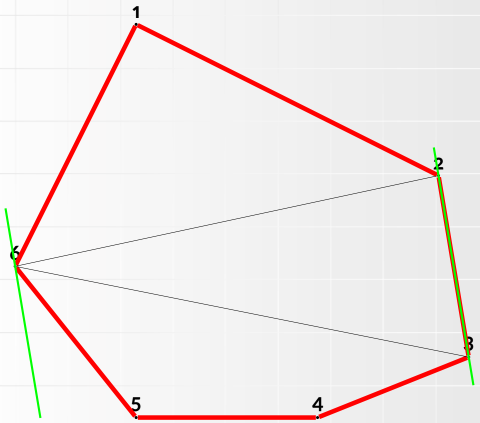
Example cont.

"Geometrical approach"



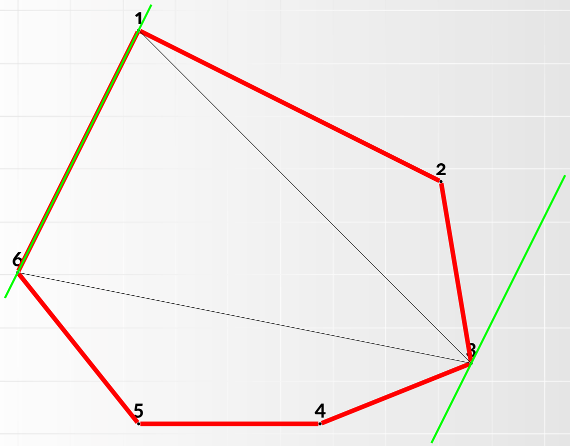
Example cont.

“Geometrical approach”



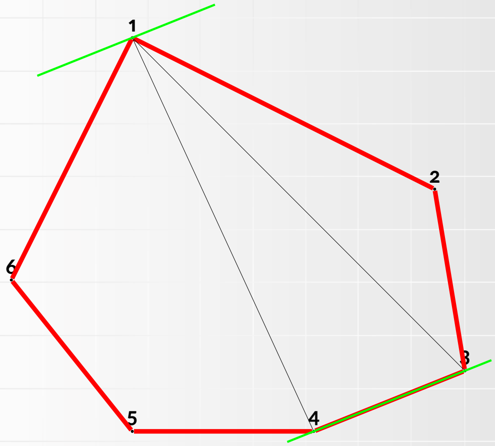
Example cont.

“Geometrical approach”



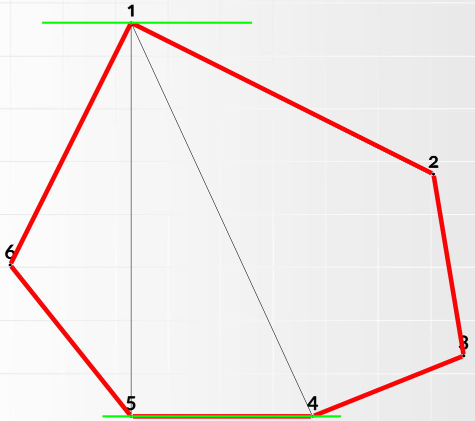
Example cont.

“Geometrical approach”



Example cont.

"Geometrical approach"



Divide and Conquer

Idea is very simple:

- When we can not solve the big task — sometimes it can be divided into several smaller tasks.
- If this are to big — we will divide them further…

Division…

Example

1. Let’s assume that we have a long, unordered list L (of numbers).
2. We are interested in finding the largest and smallest elements appearing in the list.
3. We know one method of finding maximum (and minimum). It is easy…

Maximum & Minimum

1. if L consists of one element, then that element is taken as both the minimum and the maximum; if it consists of two elements, then the smaller is taken as its minimum and the larger as its maximum;
2. otherwise do the following:
   2.1 split L into two halves, L_{left} and L_{right};
   2.2 find their extremal elements MIN_{left}, MAX_{left}, MIN_{right}, and MAX_{right};
   2.3 select the smaller of MIN_{left} and MIN_{right}; it is the minimal element of L;
   2.4 select the larger of MAX_{left} and MAX_{right}; it is the maximal element of L.

The above procedure is used for finding the minimum and maximum in each sublist.

Definition

- Recursion is the process of repeating items in a self-similar way.

Definition

- Recursion is the process of repeating items in a self-similar way.
- Recursion is the process a procedure goes through when one of the steps of the procedure involves invoking the procedure itself. A procedure that goes through recursion is said to be ‘recursive’.

Definition

- Recursion is the process of repeating items in a self-similar way.
- Recursion is the process a procedure goes through when one of the steps of the procedure involves invoking the procedure itself. A procedure that goes through recursion is said to be ‘recursive’.

Recursive humor (from the dictionary)

Recursion, see Recursion.

Wrocław University of Science and Technology

Factorial

Classical definition

$0! = 1$

$n! = \prod_{i=1}^{n} i$

or

$n! = 1 \times 2 \times 3 \times \cdots \times n$

Factorial

Classical definition

$0! = 1$

$n! = \prod_{i=1}^{n} i$

or

$n! = 1 \times 2 \times 3 \times \cdots \times n$

Recursive definition

$0! = 1$

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

Factorial

Values

```
In[1]:= 10!
Out[1]= 3628800
In[3]:= 20!
Out[3]= 2432902008176640000
In[4]:= 30!
Out[4]= 265252859812191058636308480000000
In[5]:= 40!
Out[5]= 815915283247897734345611269596115894272000000000
```

Factorial

Algorithm ("Flow diagram")

Classical

```
1. If $n = 0$, the factorial is equal to 1; Stop.
2. $factorial \leftarrow 1$
3. Repeat for $i$ changing from 1 to $n$
   $factorial \leftarrow factorial \times i$
4. End.
```

Factorial

Algorithm ("Flow diagram")

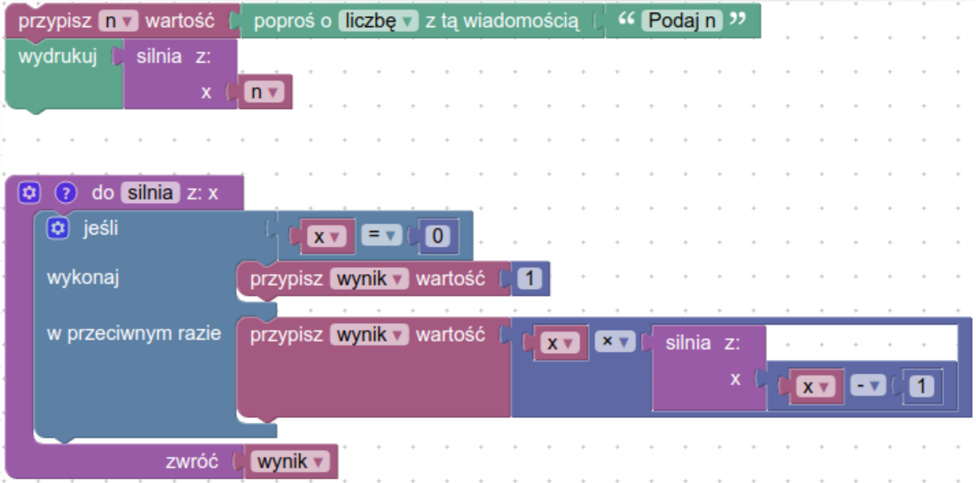
Classical

```
1. If $n = 0$, the factorial is equal to 1; Stop.
2. $factorial \leftarrow 1$
3. Repeat for $i$ changing from 1 to $n$
   $factorial \leftarrow factorial \times i$
4. End.
```

Recursive procedure

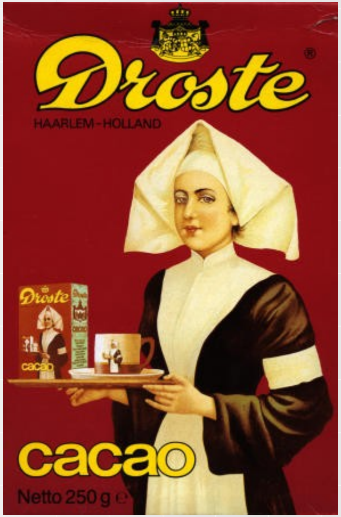
```
1. Function Factorial (parameter is $n$)
2. result $\leftarrow 1$
3. If $n = 0$; End
4. $result \leftarrow result \times$ Factorial($n - 1$)
5. End
```

Factorial in Blockly



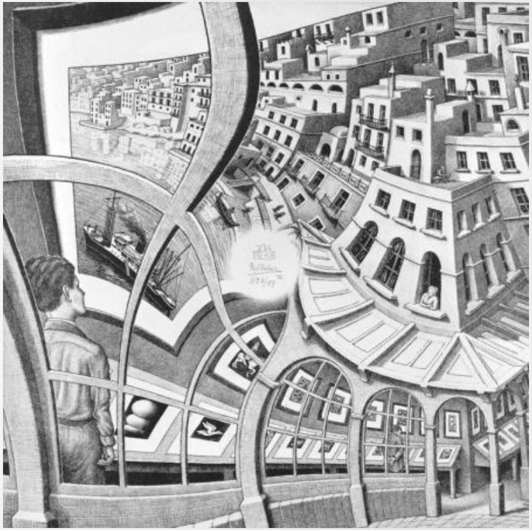
```
przypisz n ▾ wartość poproś o liczbę ▾ z tą wiadomością “ Podaj n ”
wydrukuj silnia z:
           x  n ▾

do silnia z: x
    jeśli  x ▾ = ▾ 0
    wykonaj  przypisz wynik ▾ wartość 1
    w przeciwnym razie  przypisz wynik ▾ wartość x ▾ × ▾ silnia z:
                                                            x  x ▾ - ▾ 1
    zwróć wynik ▾
```

Visual form of recursion: Droste Effect



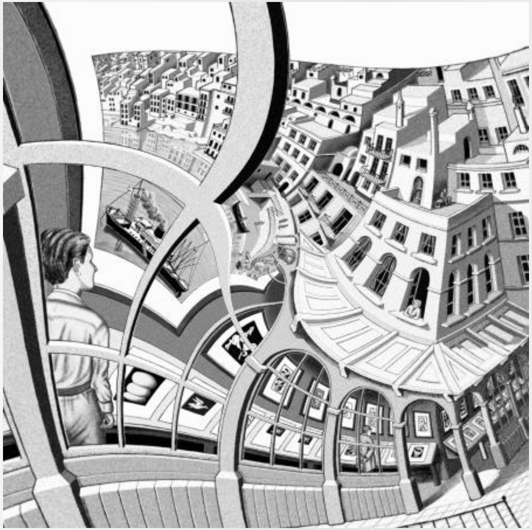
Recursion — Escher

“Prentententoonstelling” (M.C. Escher 1956)



Recursion — Escher

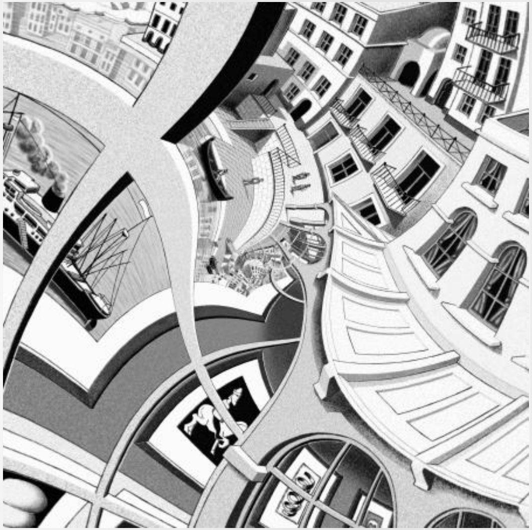
“Prententententoonstelling” (M.C. Escher 1956)



Wrocław University of Science and Technology

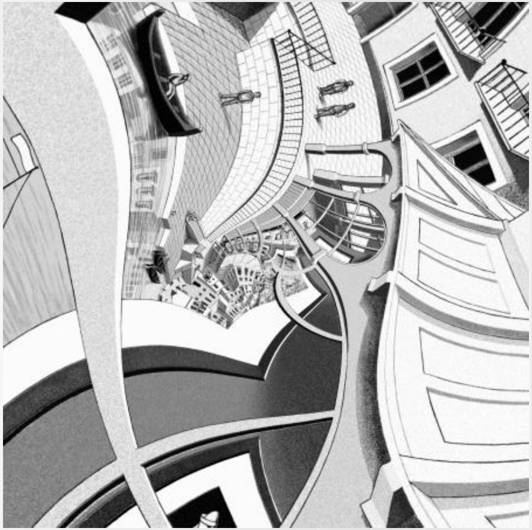
Recursion — Escher

“Prententententoonstelling” (M.C. Escher 1956)



Recursion — Escher

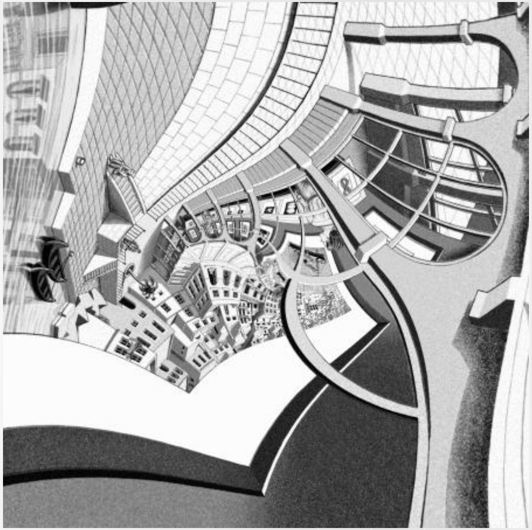
"Prententententoonstelling" (M.C. Escher 1956)



Wrocław University of Science and Technology

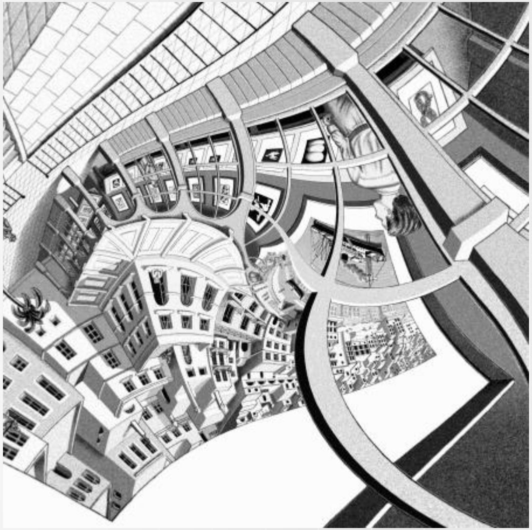
Recursion — Escher

“Prententententoonstelling” (M.C. Escher 1956)



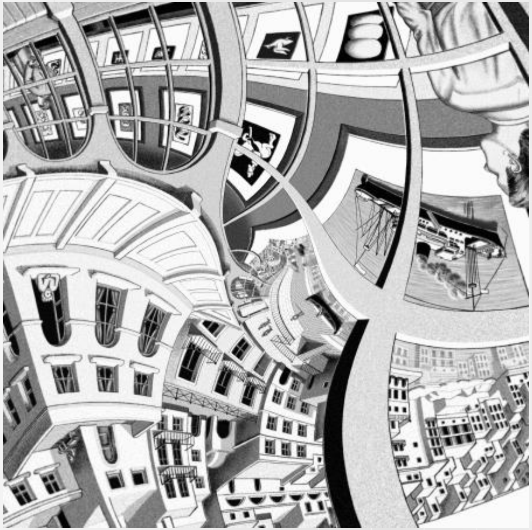
Recursion — Escher

"Prententententoonstelling" (M.C. Escher 1956)



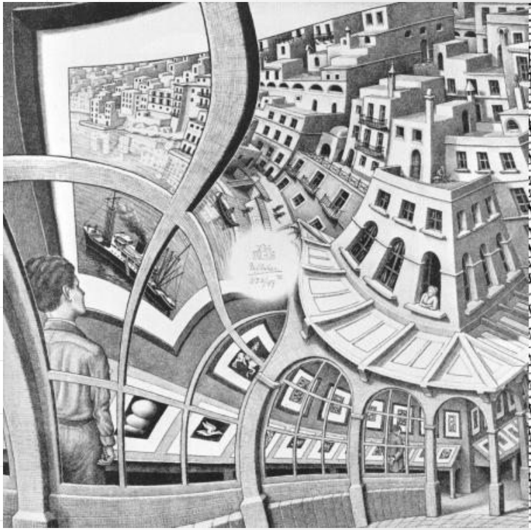
Recursion — Escher

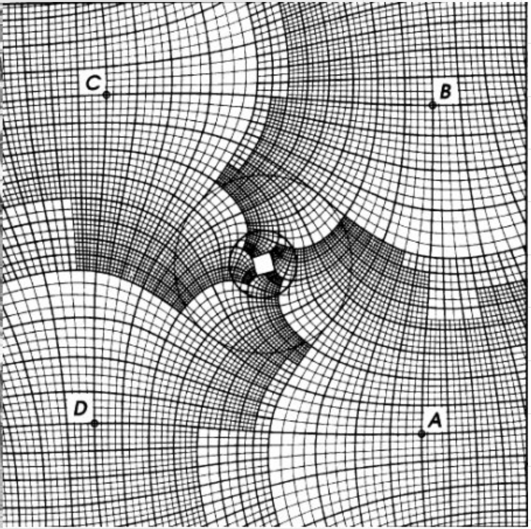
"Prententententoonstelling" (M.C. Escher 1956)



Recursion — Escher

“Prententententoonstelling” (M.C. Escher 1956)





Recursion — Escher

“Prententententoonstelling” (M.C. Escher 1956)

Prententententoonstelling means Print Exhibition.
Based on research project Escher and the Droste effect on Math Department of University of Leiden

Fibonacci series

Definition

fib(0) = 0

fib(1) = 1

fib(n) = fib(n - 1) + fib(n - 2), for $n \geq 2$

Fibonacci series

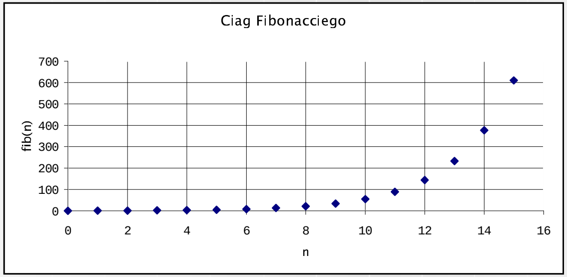
Definition

fib(0) = 0

fib(1) = 1

fib(n) = fib(n - 1) + fib(n - 2), for $n \geq 2$

0	0	
1	1	
2	1	=B2+B1
3	2	=B3+B2
4	3	
5	5	
6	8	
7	13	
8	21	
9	34	
10	55	
11	89	
12	144	
13	233	
14	377	
15	610	



Wrocław University of Science and Technology

Fibonacci series in nature

Greatest common divisor

gcd(0,n)=n

$$\gcd(k, n) = \begin{cases} n & \text{for } k = 0; \\ \gcd(n \bmod k, k) & \text{for } k > 0. \end{cases}$$

Greatest common divisor

gcd(0,n)=n

$$\gcd(k,n) = \begin{cases} n & \text{for } k = 0; \\ \gcd(n \bmod k, k) & \text{for } k > 0. \end{cases}$$

Homework

- Compare this with the previously mentioned flow diagram of Euclidean Algorithm.
- Program this in Blockly

Binomial coefficient

"Classical"

$$\binom{n}{0} = 1$$

$$\binom{n}{k} = \frac{n(n-1)\cdots(n-k+1)}{k!}$$

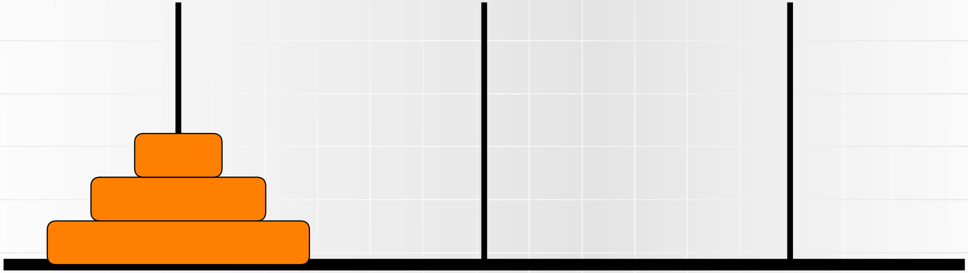
Recurrent version

$$\binom{n}{k} = \binom{n-1}{k} + \binom{n-1}{k-1}$$

$$\binom{n}{0} = 1$$

Towers of Hanoi

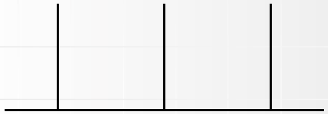
The example is based on a rather ancient puzzle known as the Towers of Hanoi, originating with Hindu priests in the great temple of Benares. Suppose we are given three pegs, A, B, and C. On the first peg, A, there are three rings piled in descending order of magnitude, while the others are empty. We are interested in moving the rings from A to B, perhaps using C in the process. **By the rules of the game, rings are to be moved one at a time, and at no instant may a larger ring be placed atop a smaller one.**



Towers of Hanoi

Example

Three rings

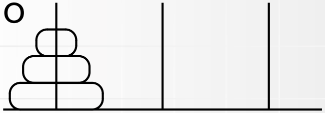


Towers of Hanoi

Example

Three rings

0

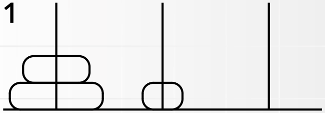


Towers of Hanoi

Example

Three rings

1

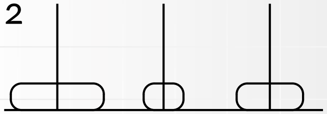


Towers of Hanoi

Example

Three rings

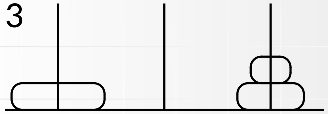
2



Towers of Hanoi

Example

Three rings

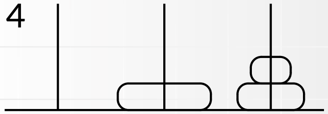


Towers of Hanoi

Example

Three rings

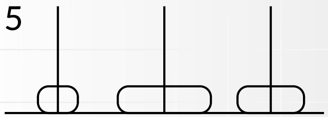
4



Towers of Hanoi

Example

Three rings

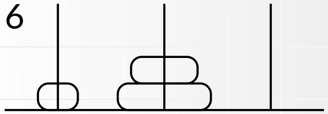


Towers of Hanoi

Example

Three rings

6

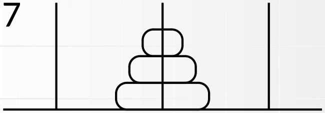


Towers of Hanoi

Example

Three rings

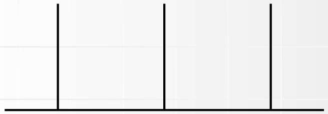
7



Towers of Hanoi

Example

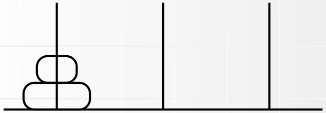
Two rings



Towers of Hanoi

Example

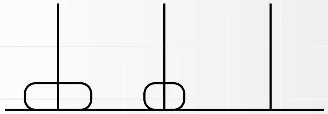
Two rings



Towers of Hanoi

Example

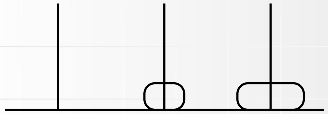
Two rings



Towers of Hanoi

Example

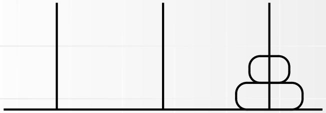
Two rings



Towers of Hanoi

Example

Two rings



Towers of Hanoi I

Why this name?

In a temple in the city of Benares, India, are 64 discs of precious diamants piled up to a tower. Each of these round discs is just a little smaller than the disc below.

Some holy men, monks, have the god given the task to move this tower to a new place inside the temple. By doing this, they have to obey some holy rules. The discs are just allowed to be placed at three marked places inside the temple.

The first place is the one where the tower was before they started to move it, the second place is the place of the destination, the third place is right between the start and destination.

The discs are so heavy and precious, that is the holy rules allow just a movement of one disc at a time.

Towers of Hanoi II

Why this name?

The last rule says that it is at no time allowed to place a disc on top of a smaller disc, while it is always allowed to place a disc on any disc with a greater diameter.

At the time when the monks have finished their work and the whole tower is moved from its starting place to its destination, at that very time, the tower will collapse and turn to dust and with this tower the whole earth will cease to exist.

There are many variations on this legend. For instance, in some tellings, the temple is a monastery and the priests are monks. The temple or monastery may be said to be in different parts of the world — including Hanoi, Vietnam, and may be associated with any religion.

When the whole earth will cease to exist?

Towers of Hanoi

The end of the world.

- If we have three rings, we need 7 moves (when there are only two — 3).

Towers of Hanoi

The end of the world.

- If we have three rings, we need 7 moves (when there are only two — 3).
- Seven moves means $2^3 - 1$ and $2^2 - 1 = 3$.

Towers of Hanoi

The end of the world.

- If we have three rings, we need 7 moves (when there are only two — 3).
- Seven moves means $2^3 - 1$ and $2^2 - 1 = 3$.
- When there are 64 rings, we will need $2^{64} - 1$ moves or 18 446 744 073 709 551 615 moves.

To name that value:
eighteen quintillion, four hundred forty-six quadrillion, seven hundred forty-four trillion, seventy-three billion, seven hundred nine million, five hundred fifty-one thousand, six hundred fifteen

Wrocław University of Science and Technology

Towers of Hanoi

The end of the world.

- If we have three rings, we need 7 moves (when there are only two — 3).
- Seven moves means $2^3 - 1$ and $2^2 - 1 = 3$.
- When there are 64 rings, we will need $2^{64} - 1$ moves or $18\ 446\ 744\ 073\ 709\ 551\ 615$ moves.
- Let us assume that one move takes 1 second.

Towers of Hanoi

The end of the world.

- If we have three rings, we need 7 moves (when there are only two — 3).
- Seven moves means $2^3 - 1$ and $2^2 - 1 = 3$.
- When there are 64 rings, we will need $2^{64} - 1$ moves or $18\ 446\ 744\ 073\ 709\ 551\ 615$ moves.
- Let us assume that one move takes 1 second.
- The year has $365 \times 24 \times 3600$ seconds ($31\ 536\ 000$)

Towers of Hanoi

The end of the world.

- If we have three rings, we need 7 moves (when there are only two — 3).
- Seven moves means $2^3 - 1$ and $2^2 - 1 = 3$.
- When there are 64 rings, we will need $2^{64} - 1$ moves or 18 446 744 073 709 551 615 moves.
- Let us assume that one move takes 1 second.
- The year has $365 \times 24 \times 3600$ seconds (31 536 000)
- Thus, this work will take $2^{64}/31536000$ years (roughly 585 billion years)

Towers of Hanoi

How to solve this problem

- When there is only one ring — no problem!.

Towers of Hanoi

How to solve this problem

- When there is only one ring — no problem!.
- For two rings, the solution is trivial.

Towers of Hanoi

How to solve this problem

- When there is only one ring — no problem!.
- For two rings, the solution is trivial.
- If there are three rings, we can divide problem into three simple tasks:

Towers of Hanoi

How to solve this problem

- When there is only one ring — no problem!.
- For two rings, the solution is trivial.
- If there are three rings, we can divide problem into three simple tasks:
  1. Move two “upper” rings to the third peg.

Towers of Hanoi

How to solve this problem

- When there is only one ring — no problem!.
- For two rings, the solution is trivial.
- If there are three rings, we can divide problem into three simple tasks:
  1. Move two “upper” rings to the third peg.
  2. Move the greatest ring to the second peg.

Towers of Hanoi

How to solve this problem

- When there is only one ring — no problem!.
- For two rings, the solution is trivial.
- If there are three rings, we can divide problem into three simple tasks:
  1. Move two “upper” rings to the third peg.
  2. Move the greatest ring to the second peg.
  3. Once again, move the two rings to the peg with the greatest ring.

Towers of Hanoi

General case

When there is *N* rings:

1. Let's move $N - 1$ rings from peg *A* to peg *C*.
2. The remaining disc (greatest one) is moved from peg *A* to *B*.
3. For the remaining (N-1)'s apply the above algorithm disc (peg B can be used as working, because at the very bottom is the largest ring).
4. Repeat this procedure until you finish the job.

(The first peg is now called *A*, the second *B*, and the third *C*.)

Towers of Hanoi

Recurrent procedure (algorithm)

Subroutine **move** N (discs) **from** X **to** Y **using** Z:

```
1. If $N = 1$ then output “$X \to Y$”;
2. otherwise (i.e., if N is greater than 1) do the following:
    2.1 call move $N - 1$ from $X$ to $Z$ using $Y$;
    2.2 output “$X \to Y$”;
    2.3 call move $N - 1$ from $Z$ to $Y$ using $X$;
3. return;
```

Symbolic notation $A \to B$ means “move the disc from the peg A to the peg B”.

This procedure **instructs** only the operator (monk).

Towers of Hanoi

Realisation (for $N = 3$)

Start

move 3 **from** A **to** B **using** C

1. **move** 2 **from** A **to** C **using** B
   - 1.1 **move** 1 **from** A **to** B **using** C
     - 1.1.1 $A \rightarrow B$
   - 1.2 $A \rightarrow C$
   - 1.3 **move** 1 **from** B **to** C **using** A
     - 1.3.1 $B \rightarrow C$
2. $A \rightarrow B$
3. **move** 2 **from** C **to** B **using** A
   - 3.1 **move** 1 **from** C **to** A **using** B
     - 3.1.1 $C \rightarrow A$
   - 3.2 $C \rightarrow B$
   - 3.3 **move** 1 **from** A **to** B **using** C
     - 3.3.1 $A \rightarrow B$

Greedy Algorithms I

and Railroad Contractors

- Many algorithmic problems call for producing a type of best result from an appropriate set of possibilities.
- Consider a network of cities and a lazy rail-road contractor. The contractor was paid to lay out rails so that it would be possible to reach any city from any other.
- The contract, however, did not specify any criteria, such as the need for certain non stop rail connections, or a maximum number of allowed cities on the path connecting any two others.
- Hence, our contractor, being lazy, is interested in laying down the cheapest (that is, the shortest) combination of rail segments.

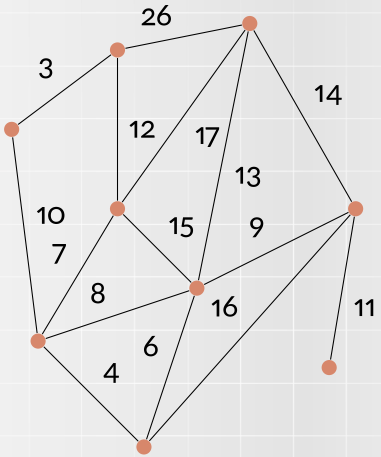
Greedy Algorithms II

and Railroad Contractors

- Assume that not all cities can be connected by direct segments of rail to all others due to objective reasons such as physical obstacles, and that the distances are given only between those pairs of cities that can be connected.
- We further assume that the cost of directly connecting city A with B is proportional to the distance between them. Moreover, we do not allow rail-road junctions outside cities.

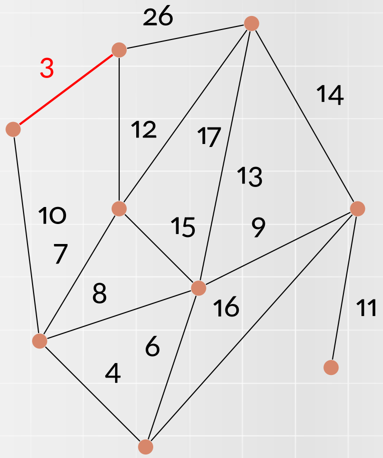
Greedy Algorithms

and Railroad Contractors



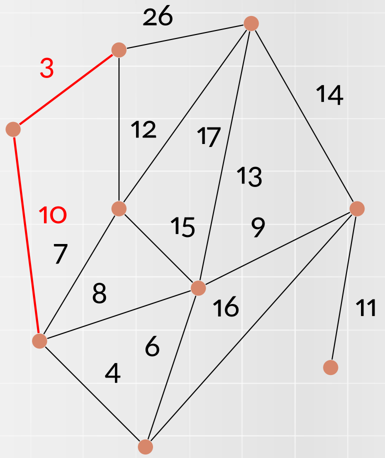
Greedy Algorithms

and Railroad Contractors



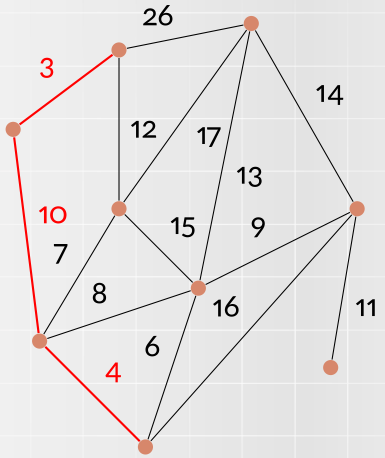
Greedy Algorithms

and Railroad Contractors



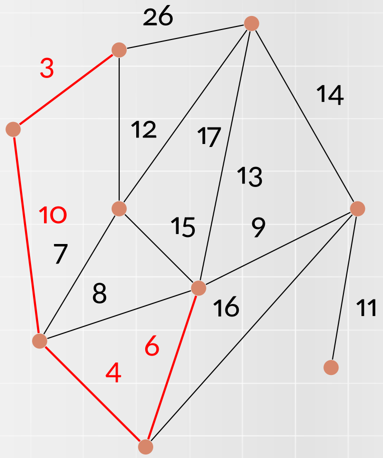
Greedy Algorithms

and Railroad Contractors



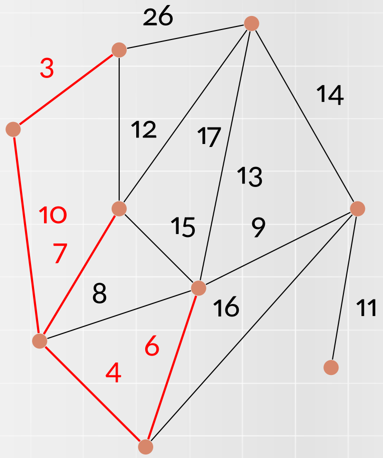
Greedy Algorithms

and Railroad Contractors



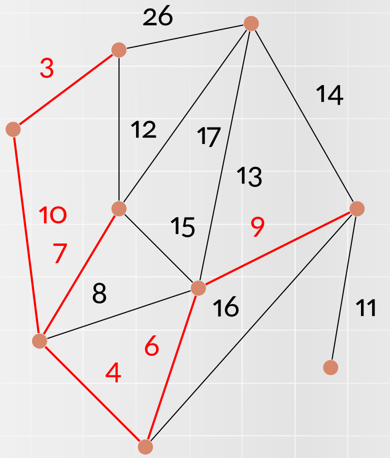
Greedy Algorithms

and Railroad Contractors



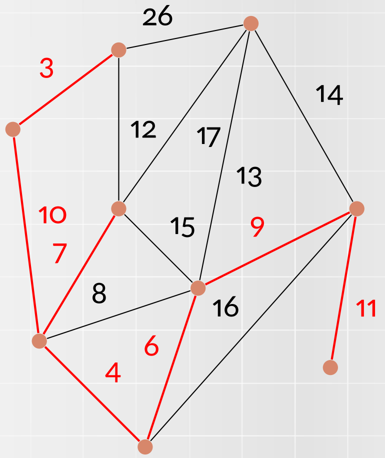
Greedy Algorithms

and Railroad Contractors



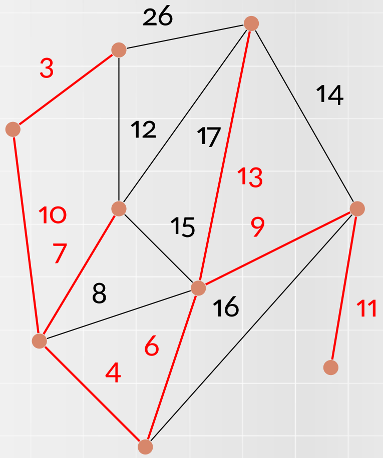
Greedy Algorithms

and Railroad Contractors



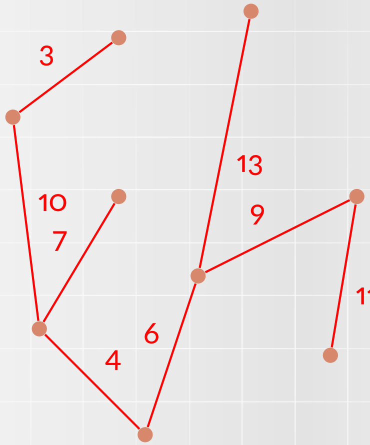
Greedy Algorithms

and Railroad Contractors



Greedy Algorithms

and Railroad Contractors



This graph is called a *minimal spanning tree*

Greedy Algorithms

Greedy Algorithm. . . Why that name?

Wrocław University of Science and Technology

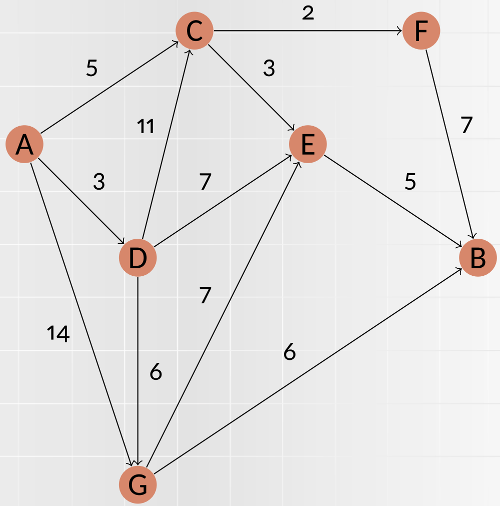
Dynamic Programming

and Weary Travellers

- We have (like in the previous example) a network of cities.
- Instead of *lazy contractor*, we have now **weary traveller**.
- Our task in previous example is to find the cheapest set of connections allowing travel between any two cities from the net.
- Now, we have to find the cheapest (shortest) path from city A to city B (using the existing network).

Dynamic Programming

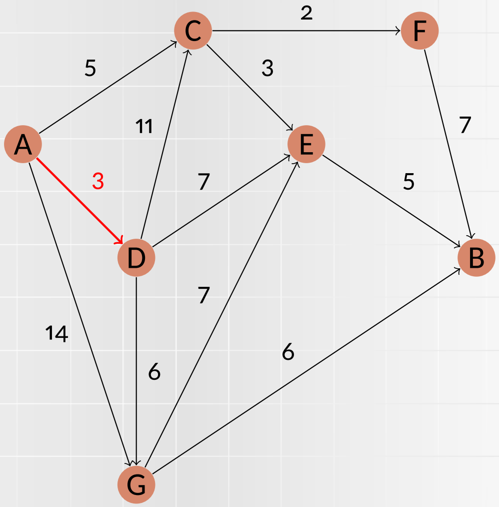
Example



Dynamic Programming

Example

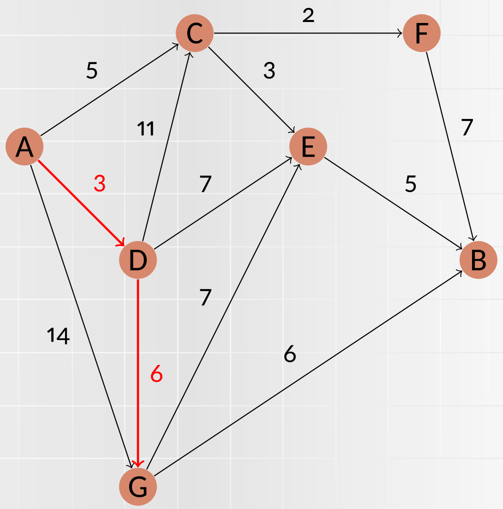
Greedy solution



Dynamic Programming

Example

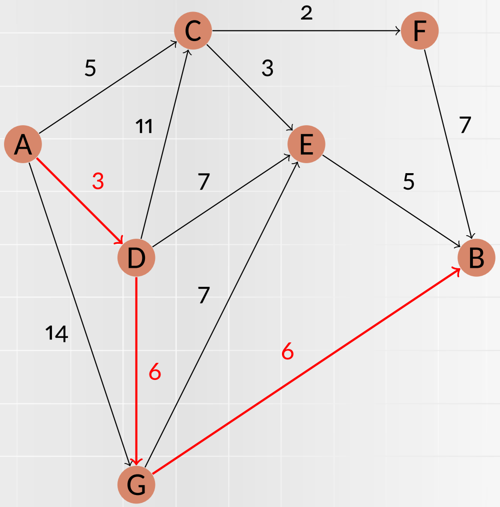
Greedy solution



Dynamic Programming

Example

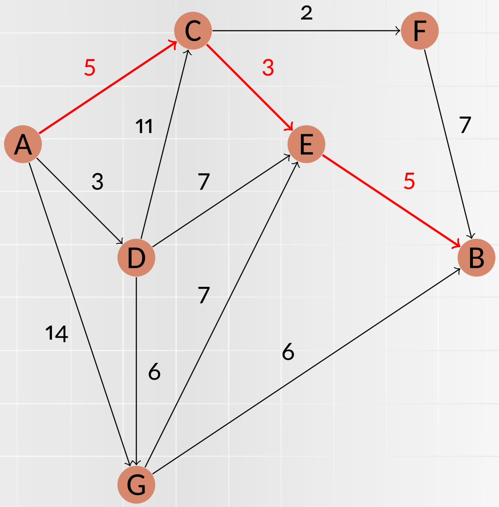
Greedy solution
Total cost 15



Dynamic Programming

Example

Optimal Solution
Total cost 13



Dynamic Programming

- The greedy algorithm finds a path of length 15, which is not as good.
- Thus, we will need a slightly different method. The one which better "predicts the next steps".
- In the first step, we have three options to get from A to B:
    1. go to C
    2. go to D
    3. go to G
- cost of the first one is 5 + the (cheapest) cost of travel from C to B;
- cost of the second one is 3 + the (cheapest) cost of travel from D to B;
- and the cost of the third is 14 + the (cheapest) cost of the travel from G to B.

Dynamic Programming

How should the optimal algorithm look like?

- Let $L(x)$ denotes minimal cost of the path from a node x to the node B
- On the each stage, when we can go from the node V to nodes C_1, $C_2, \ldots C_N$ we are choosing such a path, that

  $$\text{cost-from-}V\text{-to-}C_K + L(C_K)$$

  is minimal

This can be noted as:

$$L(V) = min_K(\text{cost-from-}V\text{-to-}C_K + L(C_K))$$

Dynamic Programming

This notation suggests using recursion. However, be warned: It is very dangerous (or inefficient).

Dynamic Programming

This notation suggests using recursion. However, be warned: It is very dangerous (or inefficient).
There are special methods for solving such problems called *dynamic programming*.

Wrocław
University
of Science
and Technology

MERRY CHRISTMAS

Wishing you a peaceful,
warm, hearty Christmas
and good luck
with your plans
in the New Year