CERC 2014: Presentation of solutions

Jagiellonian University

Some numbers

Total submits: 1002
Accepted submits: 312

Some numbers

Total submits: 1002
Accepted submits: 312

First accept: 00:07:56, problem C
University of Zagreb
(Stjepan Glavina, Ivan Katanic, Gustav Matula)

Last accept: 4:59:44, problem F
Eötvös Loránd University
(Attila János Dankovics, András Mészáros, Ágoston Weisz)

Some numbers

Most determined team:

University of Debrecen
(Martin Kelemen, Róbert Tóth, Attila Zabolai)
11 attempts at problem D

Problem H
Good morning!

Submits: 145
Accepted: 73

First solved by:
Masaryk University
(Jaromir Kala, David Klaska, Tomáš Lamser)
00:13:43

Author: Adam Polak

SOLVED!

Problem C
Sums

Submits: 241
Accepted: 68

First solved by:
University of Zagreb
(Stjepan Glavina, Ivan Katanic, Gustav Matula)
00:07:56

Author: Damian Straszak

Use the formula:

$$k + (k+1) + \ldots + (k+d-1) = \frac{(2k+d-1)d}{2}$$

To end up with equation:

$$(2k+d-1)d = 2N$$

Want to solve it for $k, d \in \mathbb{N}$, $d \geq 2$.

$$(2k + d - 1)d = 2N$$

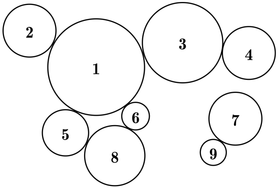
- Check all divisors $d \geq 2$ of $2N$.
- Easy to do in $O(\sqrt{N})$ time.
- "IMPOSSIBLE" if and only if $N = 2^r$

Problem D
Wheels

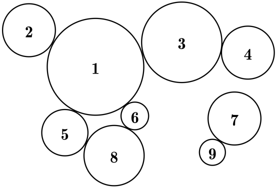
Submits: 123
Accepted: 68

First solved by:
University of Warsaw
(Patryk Czajka, Michał Makarewicz, Jan Kanty Milczek)
0:13:26

Author: Lech Duraj

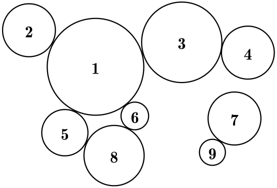


We have n wheels on the plane. The wheel 1 is turning at 1 turn per minute. What other wheels are turning, and how fast?



We have n wheels on the plane. The wheel 1 is turning at 1 turn per minute. What other wheels are turning, and how fast?

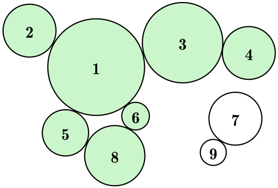
We construct a graph of n nodes – nodes x and y are connected by an edge iff the wheels x and y touch each other.



We have n wheels on the plane. The wheel 1 is turning at 1 turn per minute. What other wheels are turning, and how fast?

We construct a graph of n nodes – nodes x and y are connected by an edge iff the wheels x and y touch each other.

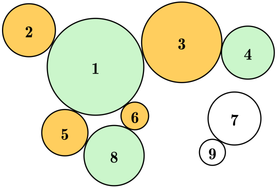
Machine not jammed $\Leftrightarrow$ The graph is bipartite.



We have n wheels on the plane. The wheel 1 is turning at 1 turn per minute. What other wheels are turning, and how fast?

We construct a graph of n nodes – nodes x and y are connected by an edge iff the wheels x and y touch each other.

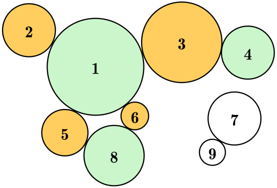
If a wheel is turning, the whole connected component is turning.



We have n wheels on the plane. The wheel 1 is turning at 1 turn per minute. What other wheels are turning, and how fast?

We construct a graph of n nodes – nodes x and y are connected by an edge iff the wheels x and y touch each other.

We compute the connected component of node 1. Wheels of even distance are turning clockwise, other – counterclockwise



We have n wheels on the plane. The wheel 1 is turning at 1 turn per minute. What other wheels are turning, and how fast?

We construct a graph of n nodes – nodes x and y are connected by an edge iff the wheels x and y touch each other.

Every point on the boundary is travelling the same distance per minute – exactly $2\pi R_1$. Therefore the wheel i makes R_1/R_i turns per minute.

Problem I
Bricks

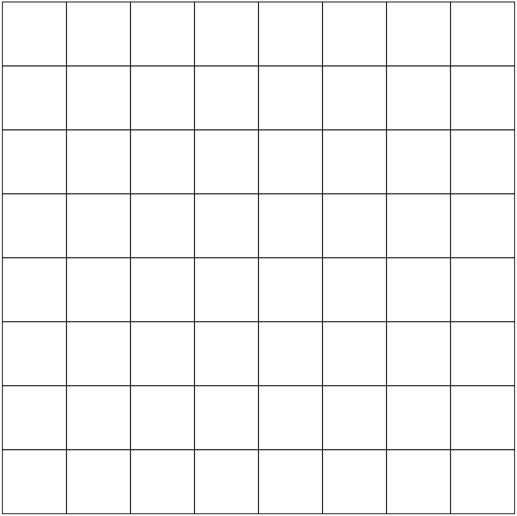
Submits: 195
Accepted: 50

First solved by:
University of Wrocław
(Bartłomiej Dudek, Maciej Dulęba, Mateusz Gołębiewski)
00:39:27

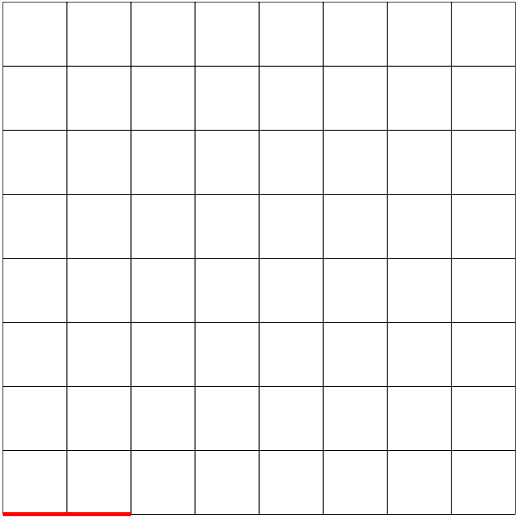
Author: Damian Straszak

- View it as a geometrical problem on a plane.
- Start at $(0, 0)$,
- white brick means "go one step right",
- black brick means "go one step up".

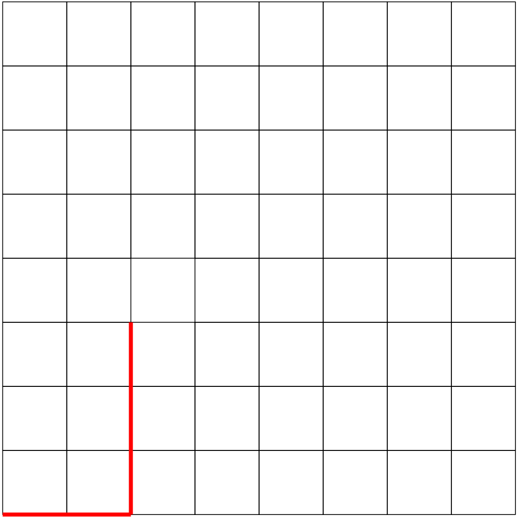
Example: 2W, 3B, 1W, 3B, 4W, 2B, 1W



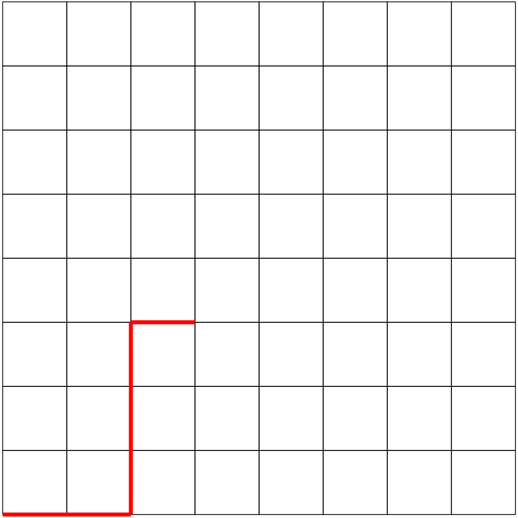
Example: 2W, 3B, 1W, 3B, 4W, 2B, 1W



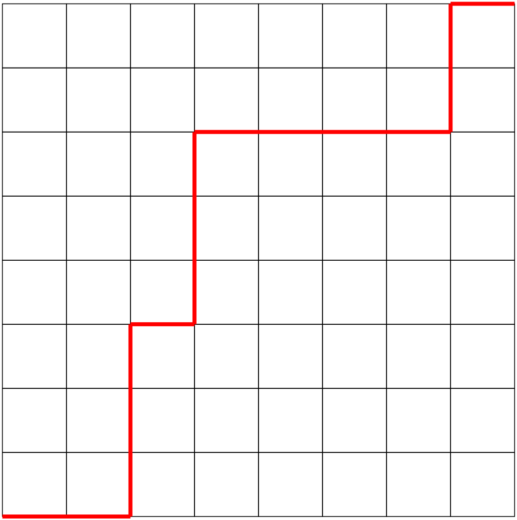
Example: 2W, 3B, 1W, 3B, 4W, 2B, 1W



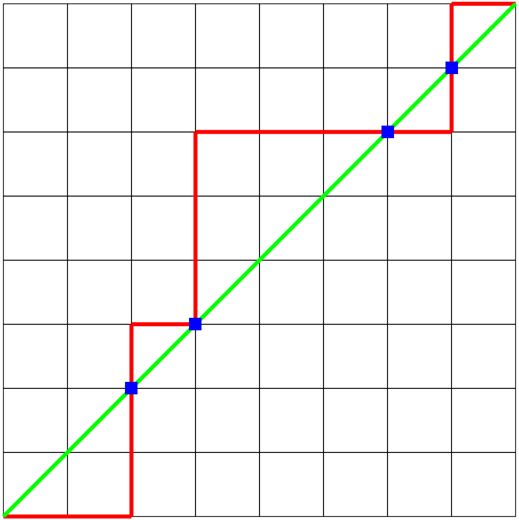
Example: 2W, 3B, 1W, 3B, 4W, 2B, 1W



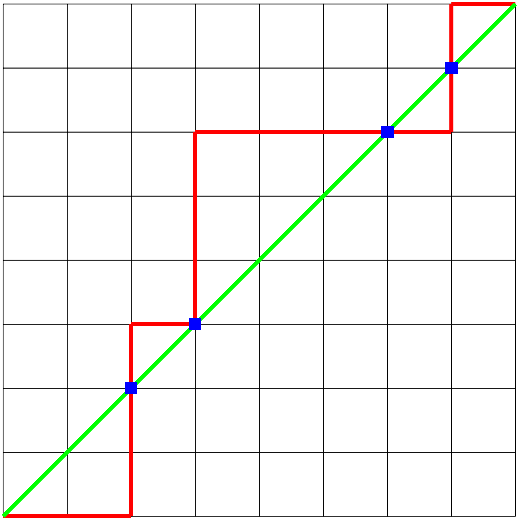
Example: 2W, 3B, 1W, 3B, 4W, 2B, 1W



Example: 2W, 3B, 1W, 3B, 4W, 2B, 1W



Example: 2W, 3B, 1W, 3B, 4W, 2B, 1W



Goal: count the number of intersection points with integral coordinates!

How to count them?

- There are at most n of them.
- For each horizontal/vertical segment calculate the potential intersection.
- Avoid floating point arithmetic.
- Complexity $O(n)$.

Problem F
Vocabulary

Submits: 91
Accepted: 22

First solved by:
University of Zagreb
(Stjepan Glavina, Ivan Katanic, Gustav Matula)
00:55:23

Author: Adam Polak

Problem:
You are given three strings, some of the characters are turned to question marks. In how many ways you can substitute question marks with letters so that the three strings are in strict lexicographical order?

First: to make things easier, if the strings are not of the equal length, pad them with `'a'-1`.

First: to make things easier, if the strings are not of the equal length, pad them with `'a'-1`.

For each string, for each its suffix, precompute the number of different ways it can be substituted (basically $26^{\text{<number of ? on the suffix>}}$).

First: to make things easier, if the strings are not of the equal length, pad them with `'a'-1`.

For each string, for each its suffix, precompute the number of different ways it can be substituted (basically $26^{<\text{number of ? on the suffix}>}$).

For each suffix, precompute the number of ways the first and the second string can be substituted (ignoring the third string) so that these two suffixes are in strict lexicographical order.
Do the analogous precomputing for the second and the third string.

Now, iterate over i from 0 to $n-1$ and consider the case when first i characters of all the three strings are equal.

Now, iterate over i from 0 to $n-1$ and consider the case when first i characters of all the three strings are equal.

There must be some difference on i-th position – there are three cases:

- $A[i] < B[i] < C[i]$,
- $A[i] < B[i] = C[i]$,
- $A[i] = B[i] < C[i]$.

Now, iterate over i from 0 to $n-1$ and consider the case when first i characters of all the three strings are equal.

There must be some difference on i-th position – there are three cases:

- $A[i] < B[i] < C[i]$,
- $A[i] < B[i] = C[i]$,
- $A[i] = B[i] < C[i]$.

The number of ways the prefix can be substituted is simply $26^{<\text{number of positions with three ?}>}$ before the first position where non-? characters are different, and 0 after this position.

Now, iterate over i from 0 to $n-1$ and consider the case when first i characters of all the three strings are equal.

There must be some difference on i-th position – there are three cases:

- $A[i] < B[i] < C[i]$,
- $A[i] < B[i] = C[i]$,
- $A[i] = B[i] < C[i]$.

The number of ways the prefix can be substituted is simply $26^{<\text{number of positions with three ?}>}$ before the first position where non-? characters are different, and 0 after this position.

The number of ways the suffixes can be substituted can be easily computed from precomputed values.

Now, iterate over i from 0 to $n-1$ and consider the case when first i characters of all the three strings are equal.

There must be some difference on i-th position – there are three cases:

- $A[i] < B[i] < C[i]$,
- $A[i] < B[i] = C[i]$,
- $A[i] = B[i] < C[i]$.

The number of ways the prefix can be substituted is simply $26^{<\text{number of positions with three ?}>}$ before the first position where non-? characters are different, and 0 after this position.

The number of ways the suffixes can be substituted can be easily computed from precomputed values.

The whole solution runs in $O(n)$ time.

Problem E
Can't stop playing

Submits: 98
Accepted: 15

First solved by:
University of Warsaw
(Paweł Kura, Bartosz Tarnawski, Kamil Żyła)
01:32:38

Author: Arkadiusz Pawlik

If we reach a configuration of the form:

$$\ldots, 16, 8, 32, \ldots$$

then we are stuck.

The only valid configurations are:

$$(l_1, l_2, \ldots, l_q, r_s, r_{s-1}, \ldots, r_1)$$

with

$$l_1 < l_2 < \ldots < l_q \quad \text{and} \quad r_s > r_{s-1} > \ldots > r_1$$

$$l_i, r_j \in \{1, 2, 4, 8, \ldots\} \text{ for all } i, j$$

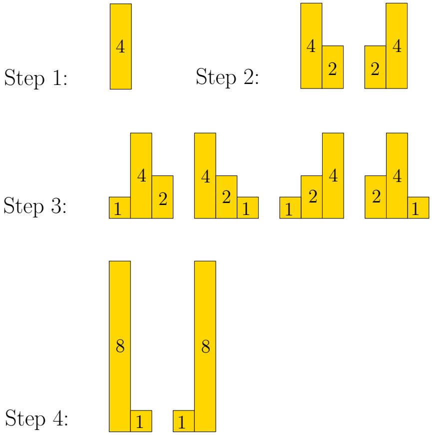
The algorithm:

- Keep the set of reachable configurations after each step,
- compute the new set by trying two options for every old reachable configuration.

How many valid configurations are there?

- Not many!
- For fixed sum $= l_1 + l_2 + \ldots + l_q + r_1 + r_2 + \ldots + r_s \leq 2^{13}$
- there are at most 2^{13} valid configurations!

Example: 4, 2, 1, 2



$$(l_1, l_2, \dots, l_q, r_s, r_{s-1}, \dots, r_1)$$

How to represent a configuration?

- at each step the sum $= l_1 + l_2 + \dots + l_q + r_1 + r_2 + \dots + r_s$ is fixed,
- it is enough to store $(l_1, l_2, \dots, l_q)$,
- or even $l = l_1 + l_2 + \dots + l_q$,
- only one 32-bit variable!

- Worst case $\approx 2^{13} \cdot 1000$ unit operations per test case.
- Recovering the answer: reverse the algorithm...

Problem K
The Imp

Submits: 54
Accepted: 6

First solved by:
University of Warsaw
(Kamil Dębowski, Błażej Magnowski, Marek Sommer)
02:16:23

Author: Lech Duraj

There are n items, each item i with $value(i)$ and cost $cost(i)$. We pick and item, pay the cost, and the adversary (The Imp) choose to either give us the item or destroy it, forcing us to pay for next item.

If the adversary can destroy at most k items, what is our total gain?

There are n items, each item i with $value(i)$ and cost $cost(i)$. We pick and item, pay the cost, and the adversary (The Imp) choose to either give us the item or destroy it, forcing us to pay for next item.

If the adversary can destroy at most k items, what is our total gain?

Let us change the game slightly. Instead of one item at a time, we give The Imp our whole strategy:

There are n items, each item i with $value(i)$ and cost $cost(i)$. We pick and item, pay the cost, and the adversary (The Imp) choose to either give us the item or destroy it, forcing us to pay for next item.

If the adversary can destroy at most k items, what is our total gain?

Let us change the game slightly. Instead of one item at a time, we give The Imp our whole strategy:

A strategy is the sequence $(s_1, \ldots, s_{k+1})$ of items. The Imp chooses which item we can keep, and we pay for this one and all the previous ones.

There are n items, each item i with $value(i)$ and cost $cost(i)$. We pick and item, pay the cost, and the adversary (The Imp) choose to either give us the item or destroy it, forcing us to pay for next item.

If the adversary can destroy at most k items, what is our total gain?

Let us change the game slightly. Instead of one item at a time, we give The Imp our whole strategy:

A strategy is the sequence $(s_1, \ldots, s_{k+1})$ of items. The Imp chooses which item we can keep, and we pay for this one and all the previous ones.

This does not help us (of course), but also doesn't help The Imp, who already knew the strategy.

Key observation:

We can assume that for the optimal strategy $s_1, \ldots, s_n$ we have $value(s_1) \leq value(s_2) \leq \ldots \leq value(s_{k+1})$!

Key observation:

We can assume that for the optimal strategy $s_1, \ldots, s_n$ we have $value(s_1) \leq value(s_2) \leq \ldots \leq value(s_{k+1})$!

This can be proven by "exchange" argument:

If $s_i > s_{i+1}$ and we swap s_i with s_{i+1}, then every Imp's possible move will yield worse result for him (and thus better for us).

So, the optimal strategy is a sequence increasing wrt. *value*.

So, the optimal strategy is a sequence increasing wrt. *value*.

With that observation, quite a few strategies work. We will show the arguably easiest to prove – dynamic strategy:

We sort the items so that $value(1) \leq value(2) \leq \ldots value(n)$.
Let $dp[i][q]$ be the optimal gain for the set of items $\{i, i+1, \ldots, n\}$ with The Imp able to cast his spell q times.

We sort the items so that $value(1) \leq value(2) \leq \dots value(n)$.
Let $dp[i][q]$ be the optimal gain for the set of items $\{i, i+1, \dots, n\}$ with The Imp able to cast his spell q times.

If we want to start the game with picking item i, then either:

- Imp lets us have it: $value(i) - cost(i)$.
- Imp destroys it: $-cost(i) + dp[i+1][q-1]$.

Of course, Imp will choose worse option.

We sort the items so that $value(1) \leq value(2) \leq \ldots value(n)$.
Let $dp[i][q]$ be the optimal gain for the set of items $\{i, i+1, \ldots, n\}$ with The Imp able to cast his spell q times.

If we want to start the game with picking item i, then either:

- Imp lets us have it: $value(i) - cost(i)$.
- Imp destroys it: $-cost(i) + dp[i+1][q-1]$.

Of course, Imp will choose worse option.

If we ignore the item i, then we should never go back to it, so the solution is $dp[i+1][q]$.

We sort the items so that $value(1) \leq value(2) \leq \dots value(n)$.
Let $dp[i][q]$ be the optimal gain for the set of items $\{i, i+1, \dots, n\}$ with The Imp able to cast his spell q times.

If we want to start the game with picking item i, then either:

- Imp lets us have it: $value(i) - cost(i)$.
- Imp destroys it: $-cost(i) + dp[i+1][q-1]$.

Of course, Imp will choose worse option.

If we ignore the item i, then we should never go back to it, so the solution is $dp[i+1][q]$.

Summing that up: $dp[i][q] =$
$max(dp[i+1][q], min(value(i) - cost(i), -cost(i) + dp[i+1][q-1]))$.

Complexity: $O(nk)$.

Why $k \leq 9$?

There is also an improved backtrack that works in $O(k! \log n + n \log n)$. It would get an OK as well.

Problem A
Parades

Submits: 26
Accepted: 5

First solved by:
University of Wrocław
(Bartłomiej Dudek, Maciej Dulęba, Mateusz Gołębiewski)
02:56:01

Author: Paweł Komosa

Problem:
You are given a tree on $n \leq 1000$ nodes with maximum degree $d \leq 10$, and $m \leq \binom{n}{2}$ paths (given as their endpoints). You need to find the maximum size edge-disjoint subset of these paths.

Problem:
You are given a tree on $n \leq 1000$ nodes with maximum degree $d \leq 10$, and $m \leq \binom{n}{2}$ paths (given as their endpoints). You need to find the maximum size edge-disjoint subset of these paths.

Solution:
$O(n^2 + nd2^d + m)$ bottom-up dynamic programming with maximum matching computation in each node.

Solution:

- Root the tree in any node.
- For each subtree compute (in bottom-up order):
  - the maximum number of disjoint paths within this subtree;
  - which nodes in this subtree are still accessible from its root without using any edge from the above paths.

Solution:

- Root the tree in any node.
- For each subtree compute (in bottom-up order):
  - the maximum number of disjoint paths within this subtree;
  - which nodes in this subtree are still accessible from its root without using any edge from the above paths.

Key observation:
It is never beneficial to leave two nodes accessible instead of connecting them with a path.

How to compute DP values for a node u knowing values for its children:

How to compute DP values for a node u knowing values for its children:

- For every child v: if any of the accessible nodes from v's subtree can be connected by a path with u, connect it (at most once per child).

How to compute DP values for a node u knowing values for its children:

- For every child v: if any of the accessible nodes from v's subtree can be connected by a path with u, connect it (at most once per child).
- Build an auxiliary graph with children as nodes. Two children are connected with an edge if there is a path on the input that connects two accessible nodes from subtrees of these children.

How to compute DP values for a node u knowing values for its children:

- For every child v: if any of the accessible nodes from v's subtree can be connected by a path with u, connect it (at most once per child).
- Build an auxiliary graph with children as nodes. Two children are connected with an edge if there is a path on the input that connects two accessible nodes from subtrees of these children.
- Calculate the maximum matching on the auxiliary graph. There are at most 10 nodes, so a simple exponential time algorithm (backtracking or DP on all subsets) is sufficient.

How to compute DP values for a node u knowing values for its children:

- For every child v: if any of the accessible nodes from v's subtree can be connected by a path with u, connect it (at most once per child).
- Build an auxiliary graph with children as nodes. Two children are connected with an edge if there is a path on the input that connects two accessible nodes from subtrees of these children.
- Calculate the maximum matching on the auxiliary graph. There are at most 10 nodes, so a simple exponential time algorithm (backtracking or DP on all subsets) is sufficient.
- If two children get matched in the auxiliary graph, select a path that connects two accessible nodes from subtrees of these children.

How to compute DP values for a node u knowing values for its children:

- For every child v: if any of the accessible nodes from v's subtree can be connected by a path with u, connect it (at most once per child).
- Build an auxiliary graph with children as nodes. Two children are connected with an edge if there is a path on the input that connects two accessible nodes from subtrees of these children.
- Calculate the maximum matching on the auxiliary graph. There are at most 10 nodes, so a simple exponential time algorithm (backtracking or DP on all subsets) is sufficient.
- If two children get matched in the auxiliary graph, select a path that connects two accessible nodes from subtrees of these children.
- If a children remains unmatched, make all its accessible nodes also accessible from u. Also u itself remains accessible from u.

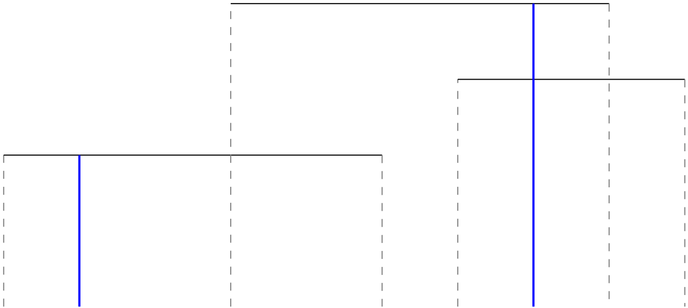
Problem L
Outer space invaders

Submits: 8
Accepted: 3

First solved by:
Charles University in Prague
(Pavol Rohár, Jakub Šafin, Tomáš Šváb)
02:38:09

Author: Bartosz Walczak

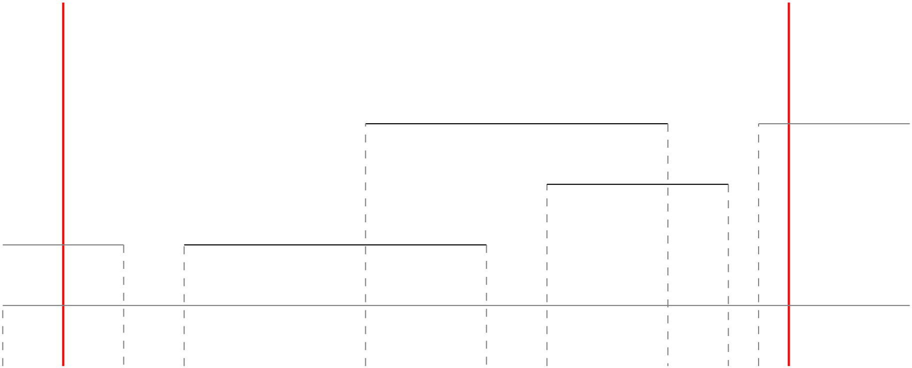
Problem (slightly rephrased):
You are given $n \leq 300$ horizontal segments $(a_i, d_i) - (b_i, d_i)$. You have to draw some vertical segments, starting at the OX axis, in such a way that they touch (or intersect) all horizontal segments and have the minimal total length.

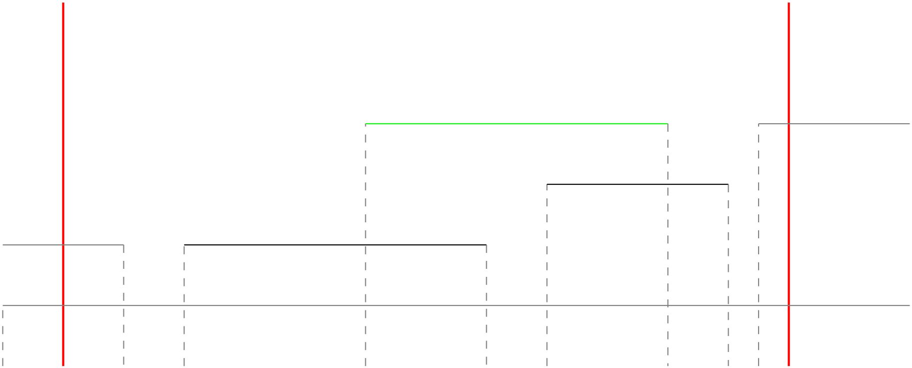


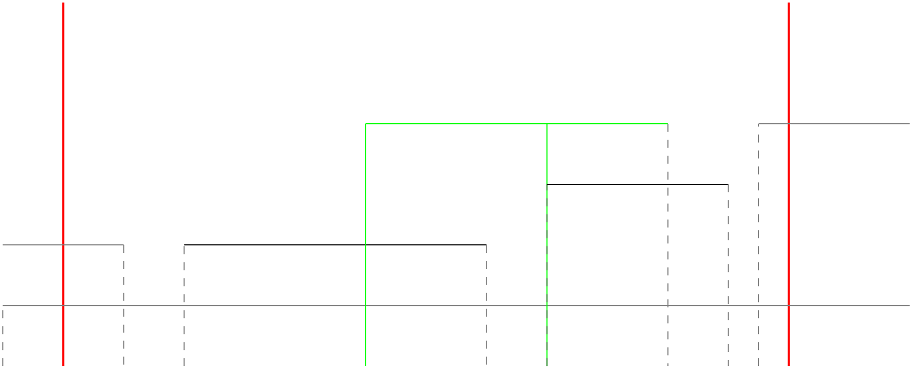
Observation: it is always OK to draw horizontal segments only at some a_i.

Solution: dynamic programming with $O(n^2)$ states and $O(n)$ time per state.

$DP[i][j] =$ minimal length of vertical segments intersecting horizontal segments fully contained in (a_i, a_j) interval.





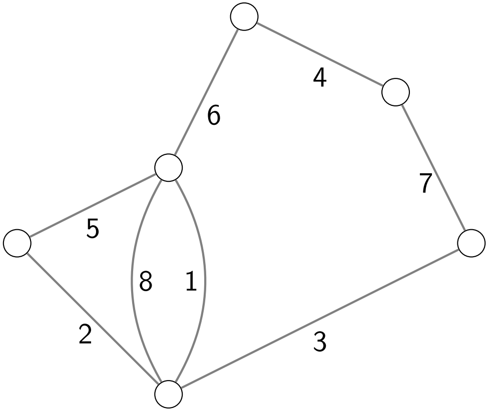


Problem J
Pork barrel

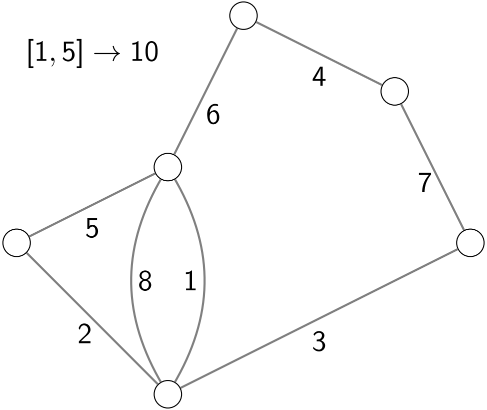
Submits: 2
Accepted: 1

First solved by:
University of Zagreb
(Stjepan Glavina, Ivan Katanic, Gustav Matula)
02:46:03

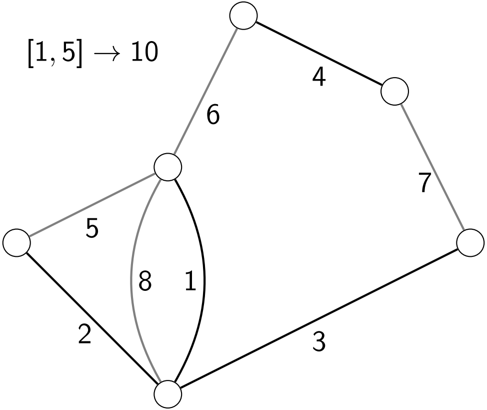
Author: Michał Sapalski, Grzegorz Herman

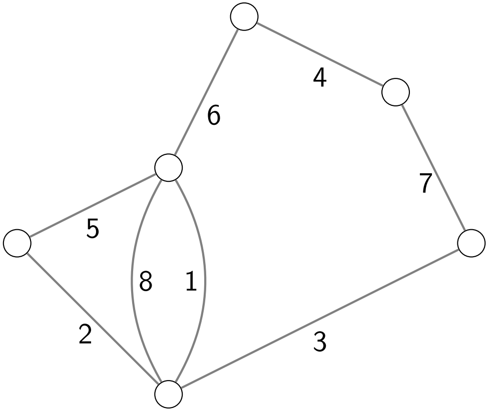

4
6
7
5
8
1
2
3

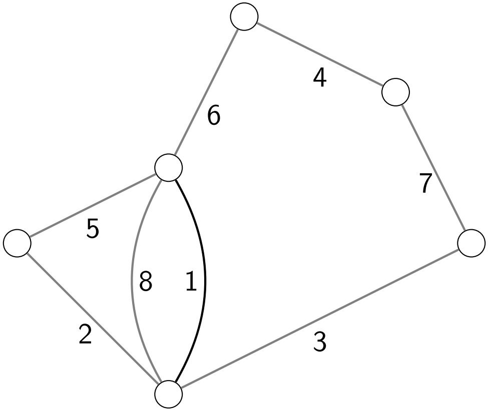
$[1, 5] \rightarrow 10$

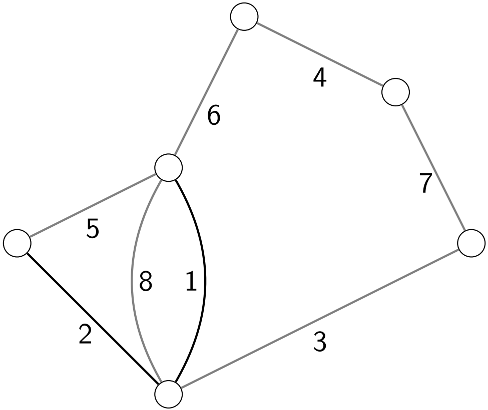


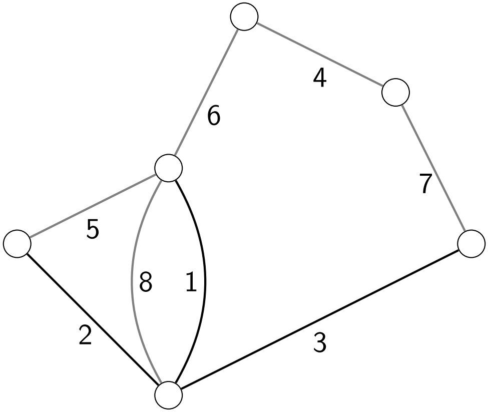
$[1,5] \to 10$

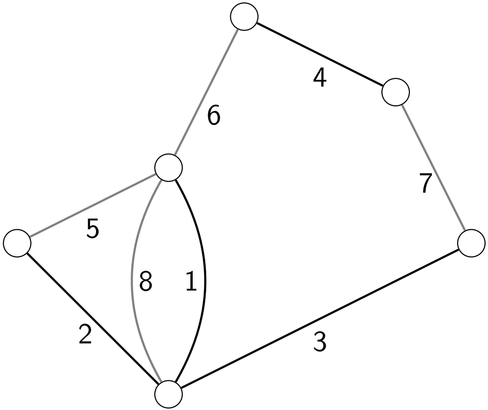


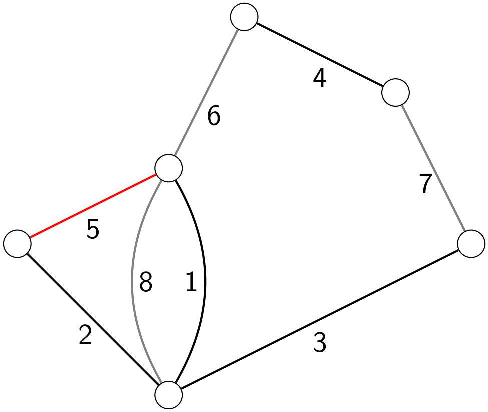

6
4
7
5
8
1
2
3

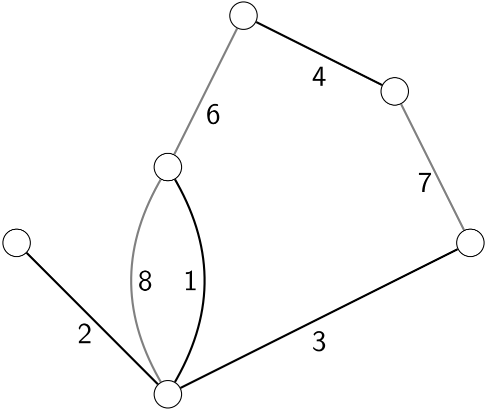

4
6
7
5
8
1
2
3

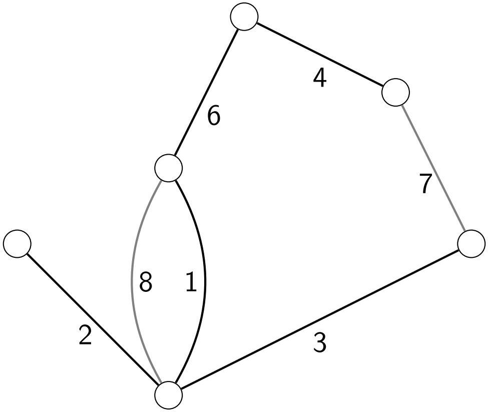

4
6
7
5
8
1
2
3

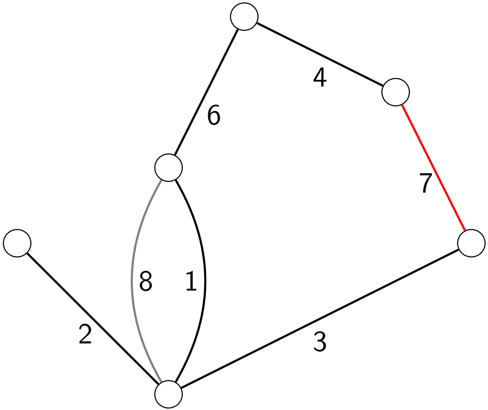

4
6
7
5
8
1
2
3

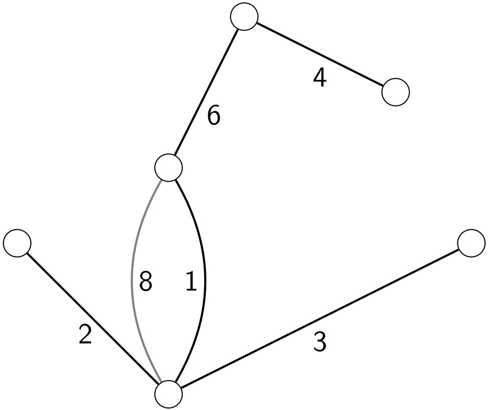

4
6
7
5
8
1
2
3

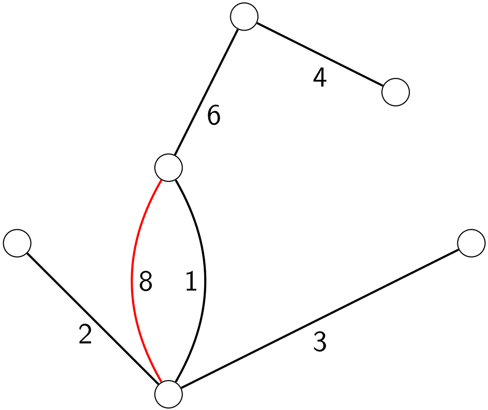

4
6
7
5
8
1
2
3

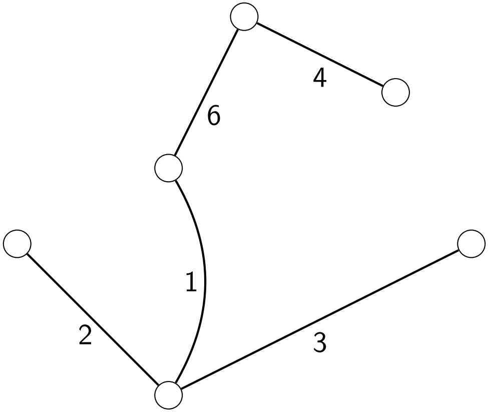

4
6
7
8
1
2
3

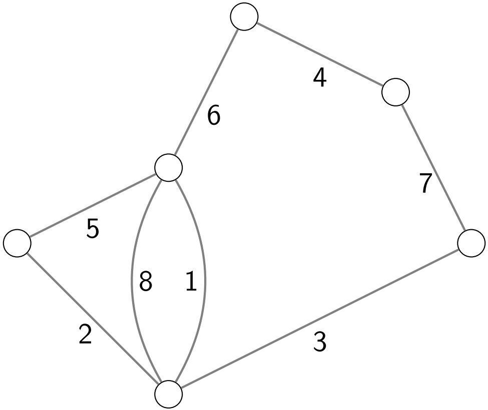

4
6
7
8
1
2
3

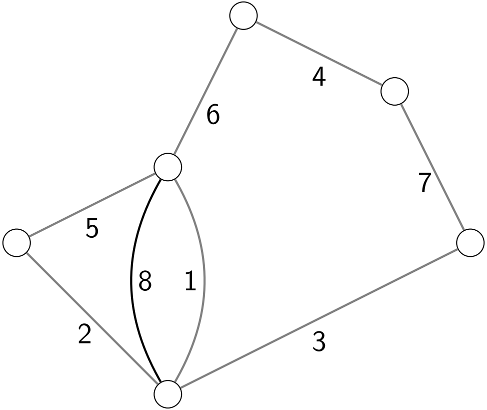

4
6
7
8
1
2
3

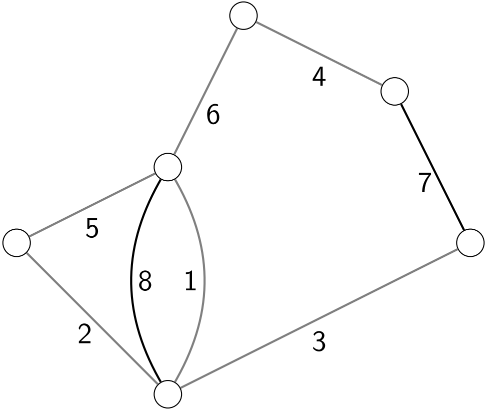

4
6
8
1
2
3

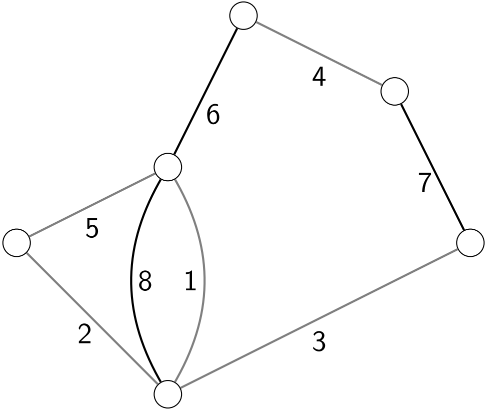

4
6
8
1
2
3

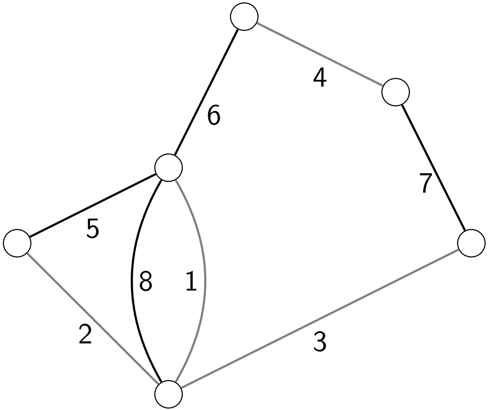

4
6
1
2
3

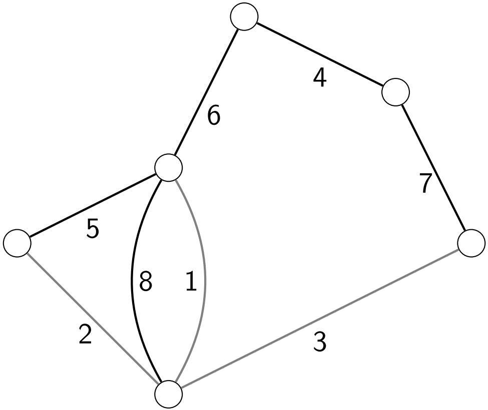

4
6
7
5
8
1
2
3

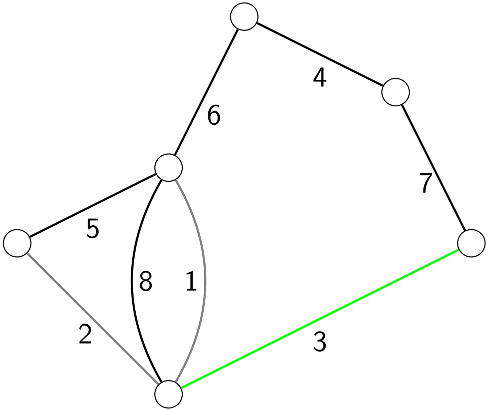

4
6
7
5
8
1
2
3

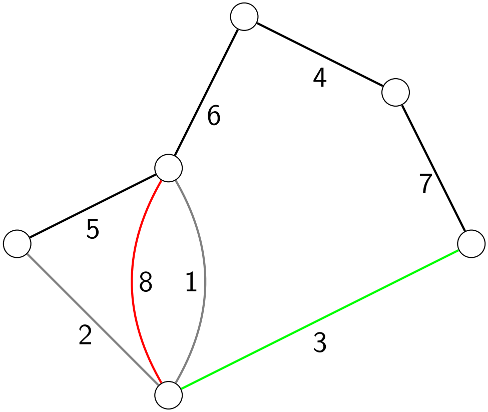

4
6
7
5
8
1
2
3

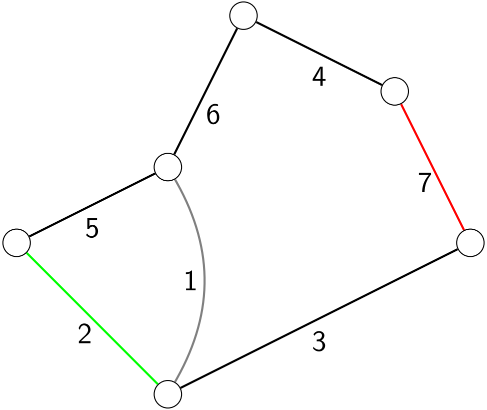

4
6
7
5
8
1
2
3

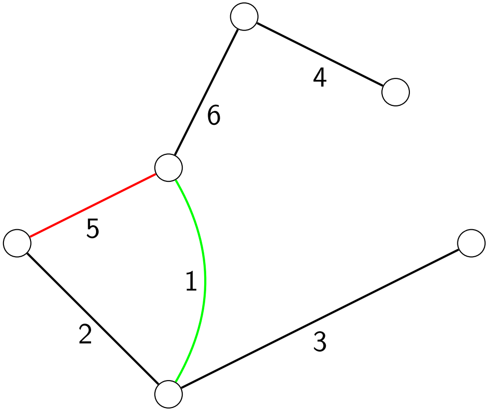

4
6
7
5
8
1
2
3

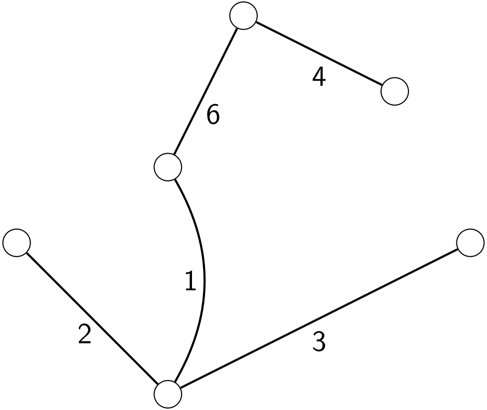

4
6
7
5
8
1
2
3

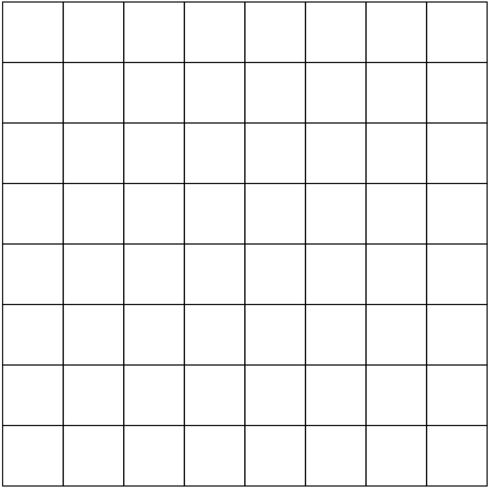

6
4
7
5
8
1
2
3


6
4
7
5
8
1
2
3


6
4
7
5
1
2
3


4
6
5
1
2
3


4
6
1
2
3



8							

8							
8	7						

8							
8	7						
8	7	6					

8							
8	7						
8	7	6					
8	7	6	5				

8							
8	7						
8	7	6					
8	7	6	5				
8	7	6	5	4			

8							
8	7						
8	7	6					
8	7	6	5				
8	7	6	5	4			
	7	6	5	4	3		

8							
8	7						
8	7	6					
8	7	6	5				
8	7	6	5	4			
	7	6	5	4	3		
		6	5	4	3	2	

8							
8	7						
8	7	6					
8	7	6	5				
8	7	6	5	4			
	7	6	5	4	3		
		6	5	4	3	2	
		6		4	3	2	1

8							
8	7						
8	7	6					
8	7	6	5				
8	7	6	5	4			
	7	6	5	4	3		
		6	5	4	3	2	
		6		4	3	2	1

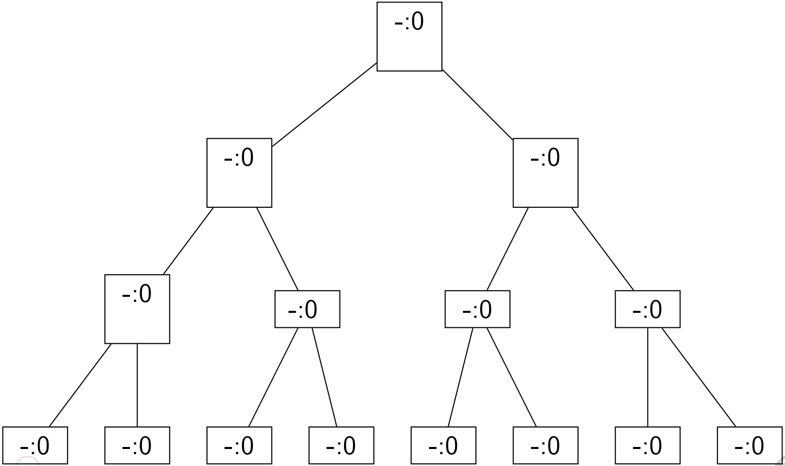
$[5, 7] \to 18$

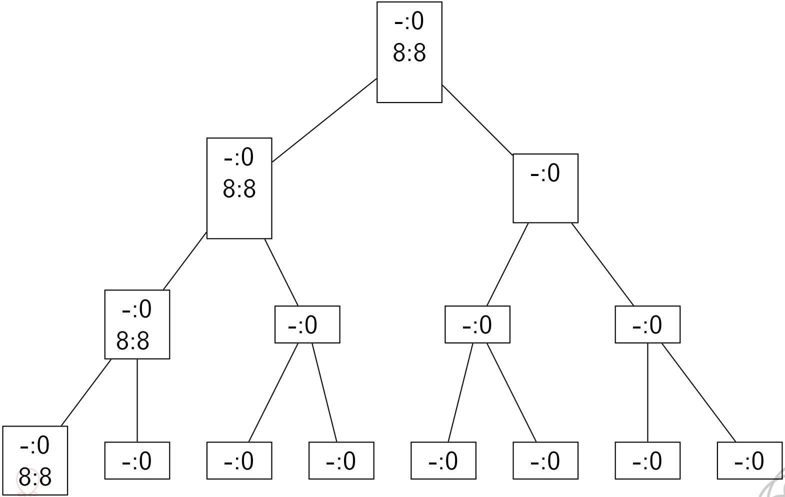
8							
8	7						
8	7	6					
8	7	6	5				
8	7	6	5	4			
	7	6	5	4	3		
		6	5	4	3	2	
		6		4	3	2	1

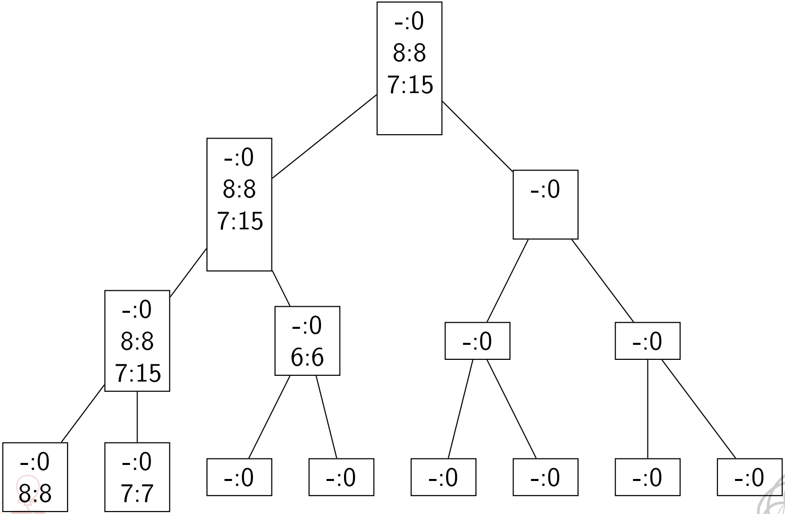
$[4, 8] \to 30$

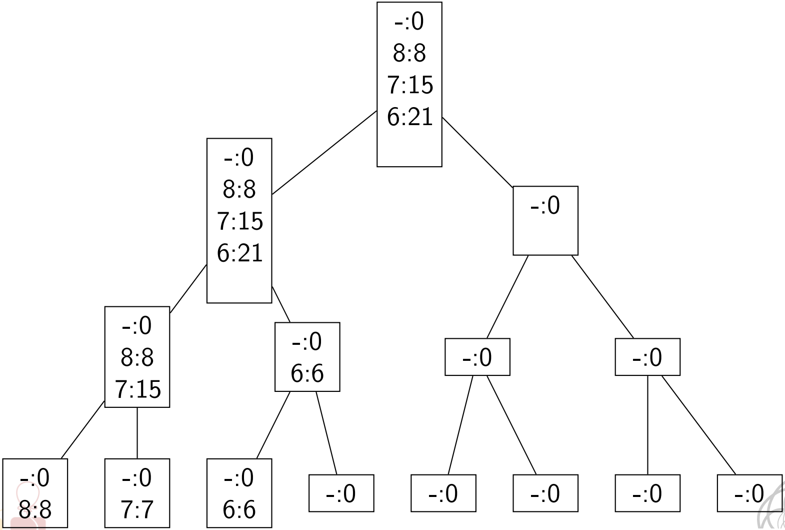
8							
8	7						
8	7	6					
8	7	6	5				
8	7	6	5	4			
	7	6	5	4	3		
		6	5	4	3	2	
		6		4	3	2	1

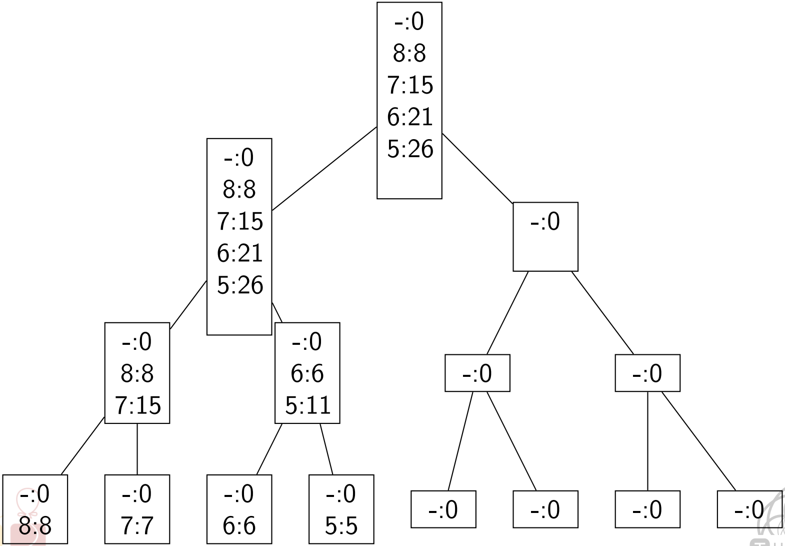
$[1, 5] \to 10$

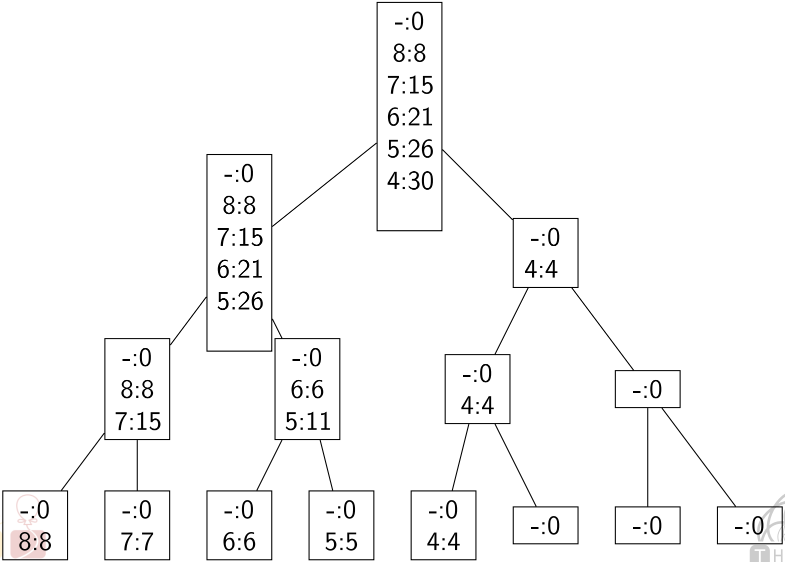

-:0
-:0
-:0
-:0
-:0
-:0
-:0
-:0
-:0
-:0
-:0
-:0
-:0
-:0
-:0

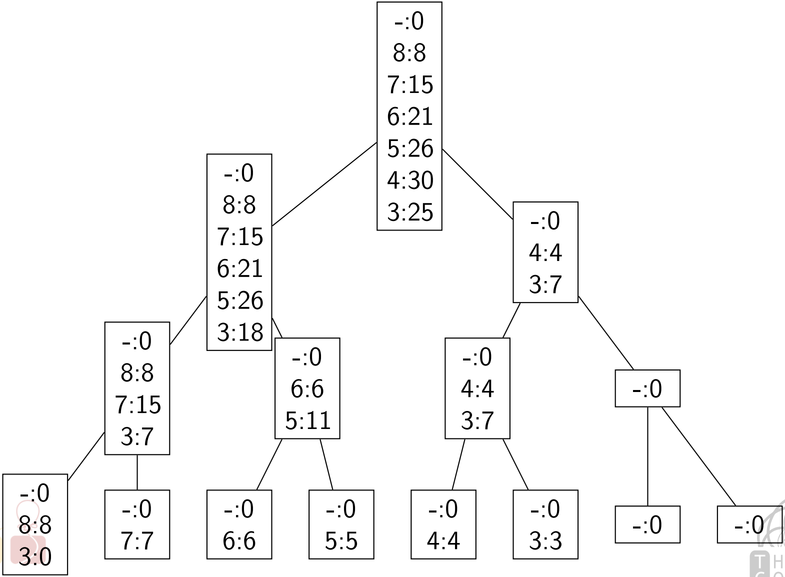

-:0
8:8
-:0
8:8
-:0
-:0
8:8
-:0
-:0
-:0
-:0
8:8
-:0
-:0
-:0
-:0
-:0
-:0
-:0

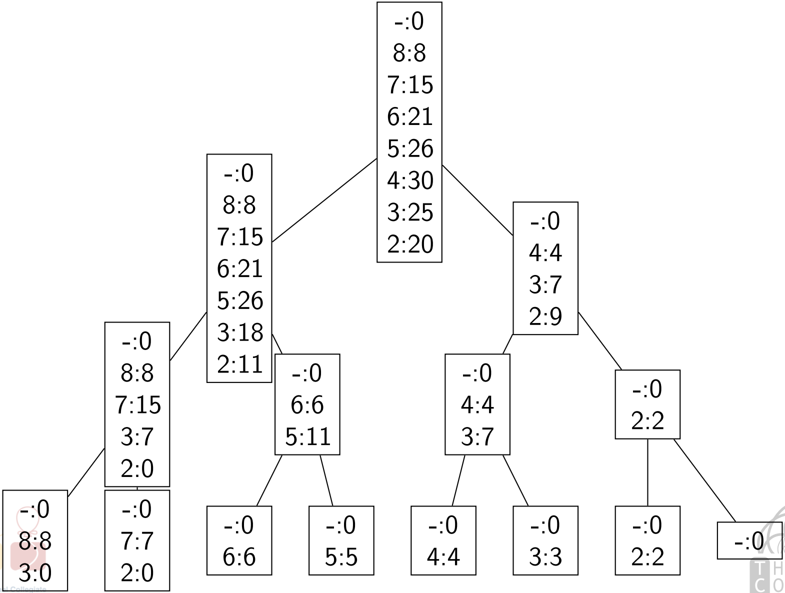

-:0
8:8
7:15
-:0
8:8
7:15
-:0
-:0
8:8
7:15
-:0
6:6
-:0
-:0
-:0
8:8
-:0
7:7
-:0
-:0
-:0
-:0
-:0
-:0

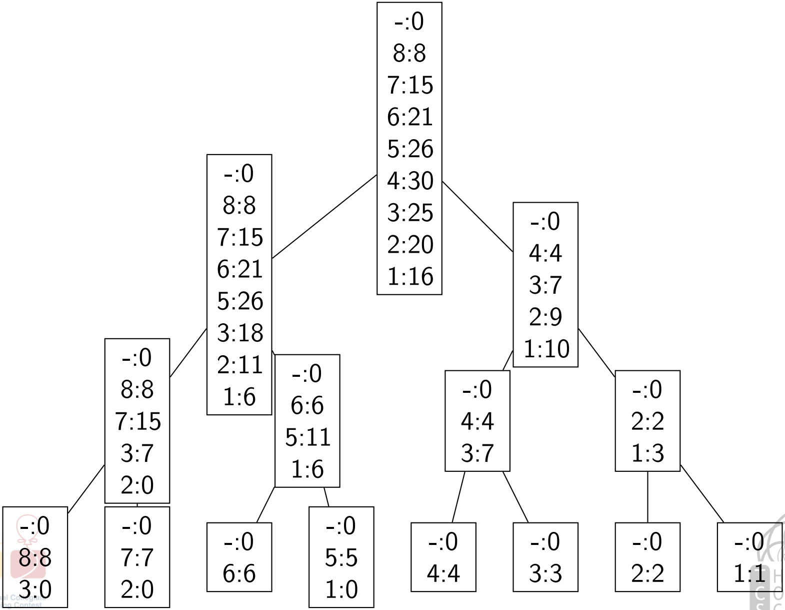

-:0
8:8
7:15
6:21
-:0
8:8
7:15
6:21
-:0
-:0
8:8
7:15
-:0
6:6
-:0
-:0
-:0
8:8
-:0
7:7
-:0
6:6
-:0
-:0
-:0
-:0
-:0

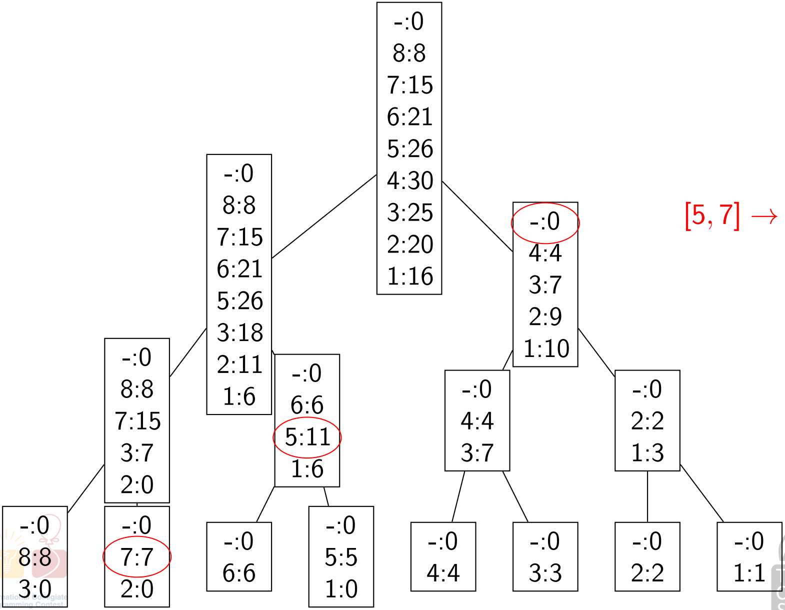

-:0
8:8
7:15
6:21
5:26
-:0
8:8
7:15
6:21
5:26
-:0
-:0
8:8
7:15
-:0
6:6
5:11
-:0
-:0
-:0
8:8
-:0
7:7
-:0
6:6
-:0
5:5
-:0
-:0
-:0
-:0


-:0 8:8 7:15 6:21 5:26 4:30
-:0 8:8 7:15 6:21 5:26
-:0 4:4
-:0 8:8 7:15
-:0 6:6 5:11
-:0 4:4
-:0
-:0 8:8
-:0 7:7
-:0 6:6
-:0 5:5
-:0 4:4
-:0
-:0
-:0


-:0 8:8 7:15 6:21 5:26 4:30 3:25
-:0 8:8 7:15 6:21 5:26 3:18
-:0 4:4 3:7
-:0 8:8 7:15 3:7
-:0 6:6 5:11
-:0 4:4 3:7
-:0
-:0 8:8 3:0
-:0 7:7
-:0 6:6
-:0 5:5
-:0 4:4
-:0 3:3
-:0
-:0


-:0 8:8 7:15 6:21 5:26 4:30 3:25 2:20
-:0 8:8 7:15 6:21 5:26 3:18 2:11
-:0 4:4 3:7 2:9
-:0 8:8 7:15 3:7 2:0
-:0 6:6 5:11
-:0 4:4 3:7
-:0 2:2
-:0 8:8 3:0
-:0 7:7 2:0
-:0 6:6
-:0 5:5
-:0 4:4
-:0 3:3
-:0 2:2
-:0


-:0 8:8 7:15 6:21 5:26 4:30 3:25 2:20 1:16
-:0 8:8 7:15 6:21 5:26 3:18 2:11 1:6
-:0 4:4 3:7 2:9 1:10
-:0 8:8 7:15 3:7 2:0
-:0 6:6 5:11 1:6
-:0 4:4 3:7
-:0 2:2 1:3
-:0 8:8 3:0
-:0 7:7 2:0
-:0 6:6
-:0 5:5 1:0
-:0 4:4
-:0 3:3
-:0 2:2
-:0 1:1



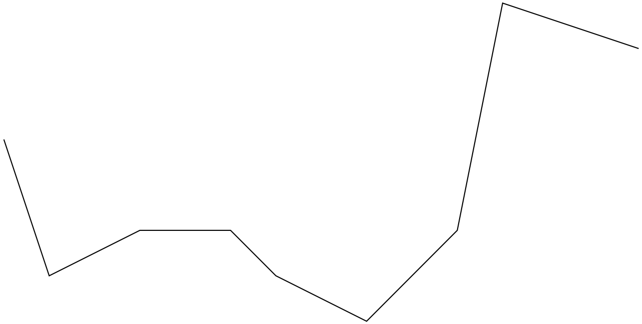
$[5, 7] \rightarrow 18$

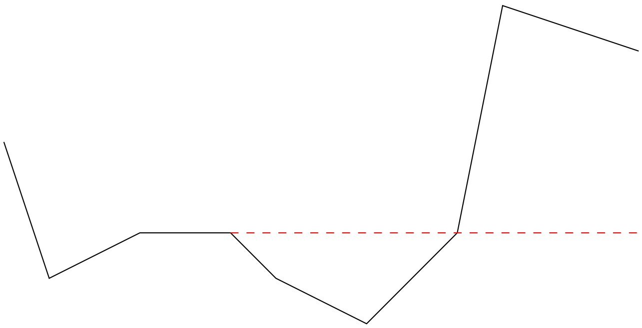
Problem B
Mountainous landscape

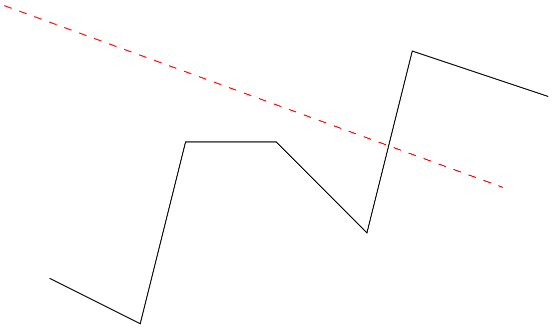
Submits: 11
Accepted: 1

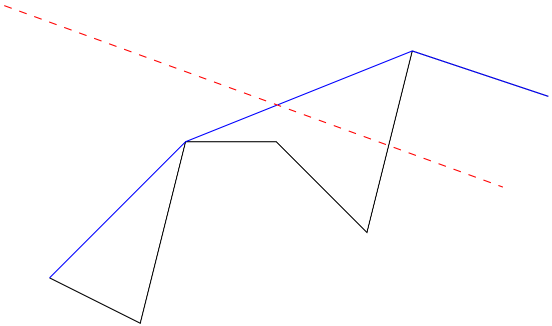
First solved by:
AGH University of Science and Technology
(Miłosz Łakomy, Adam Obuchowicz, Martyna Wałaszewska)
03:50:05

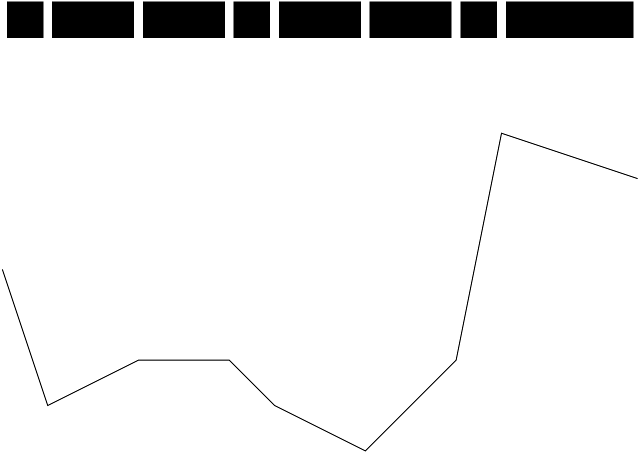
Author: Grzegorz Herman

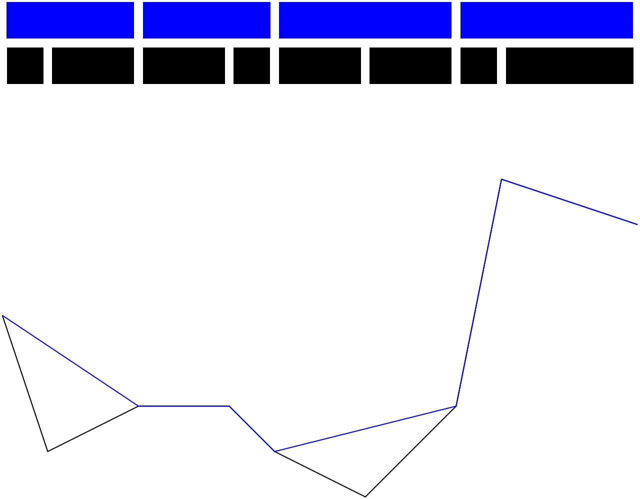


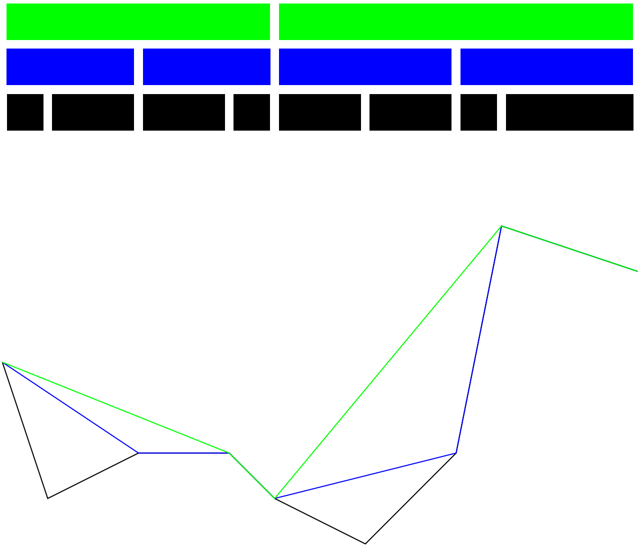


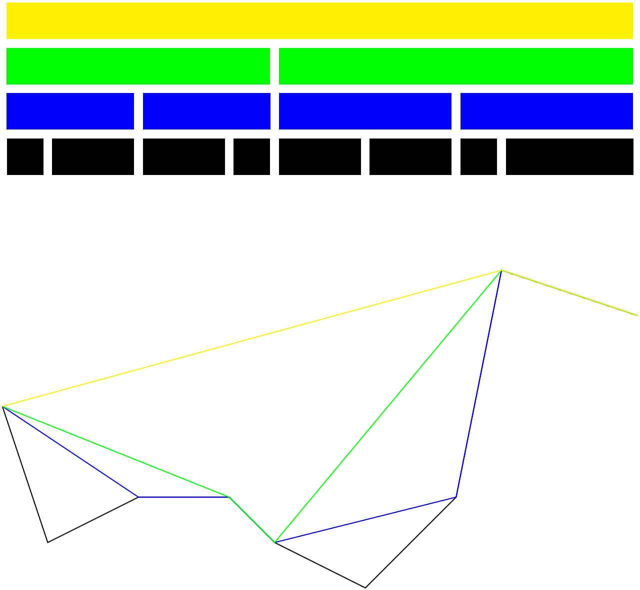


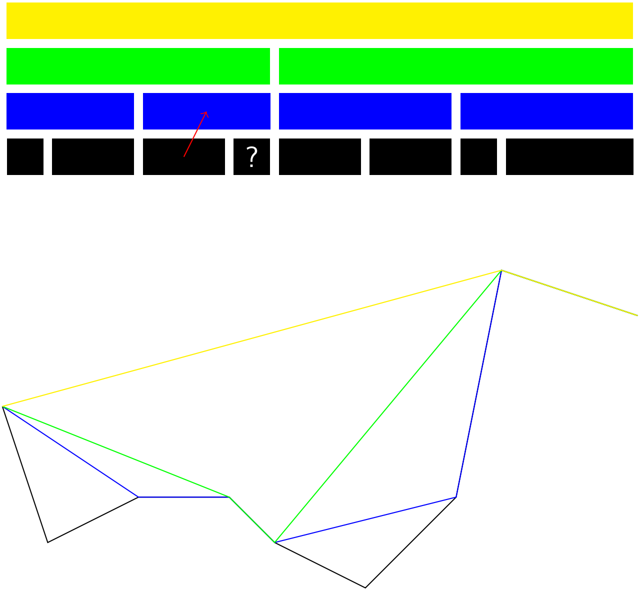


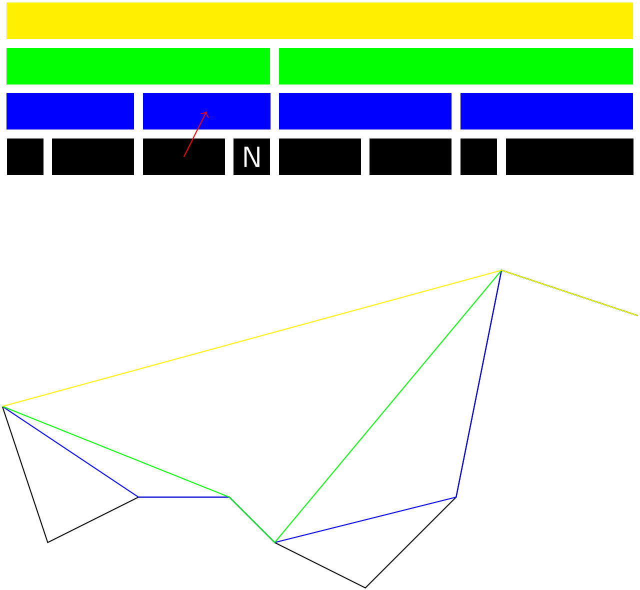




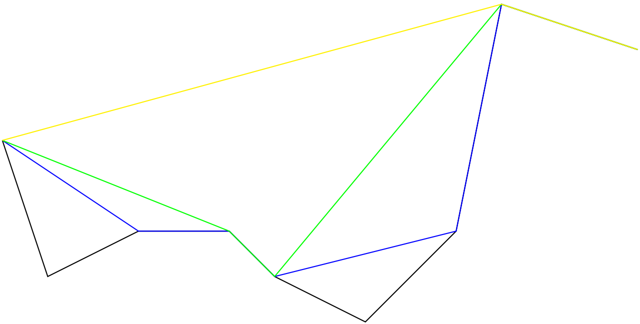


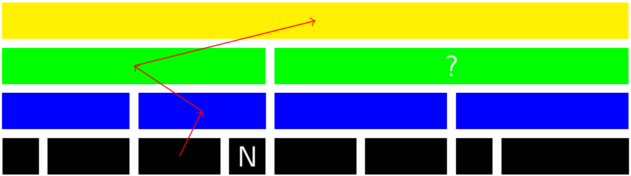


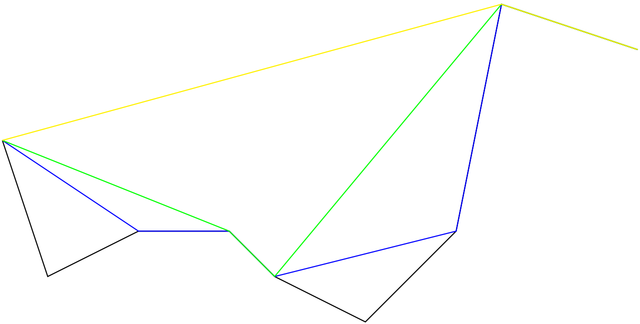


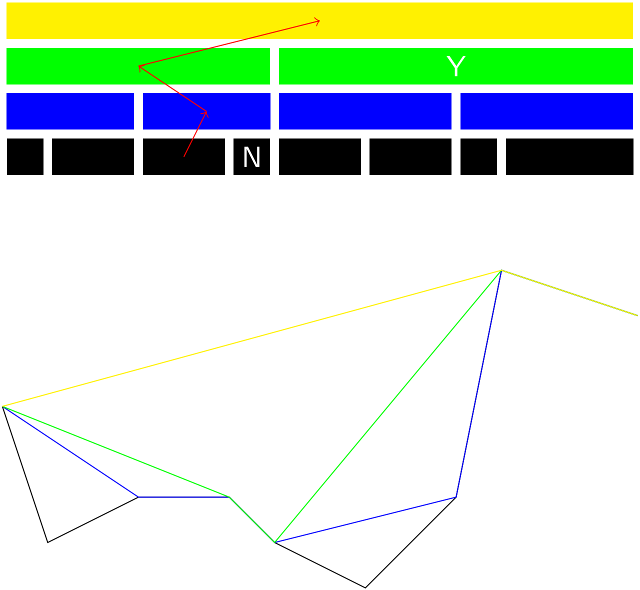

N

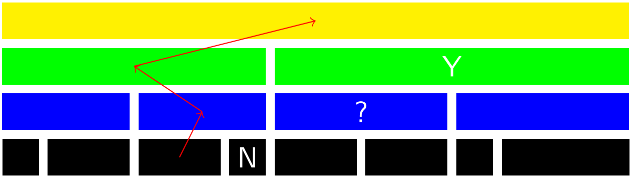

N

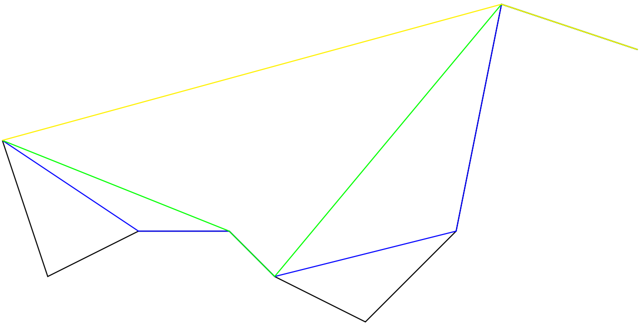


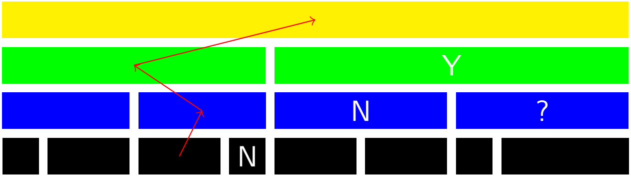

?
N

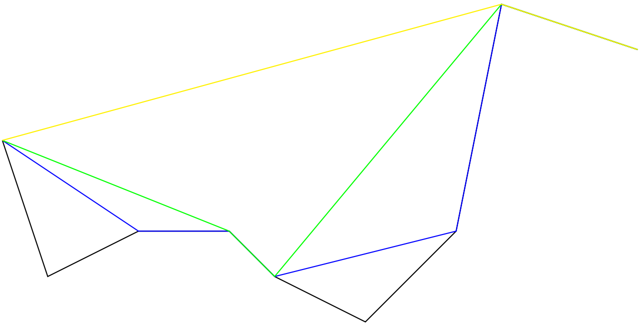


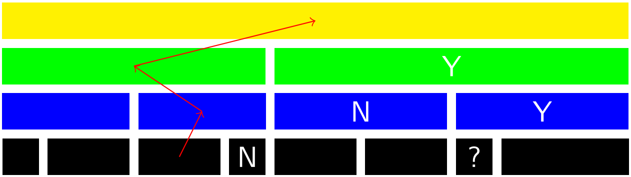

Y
N

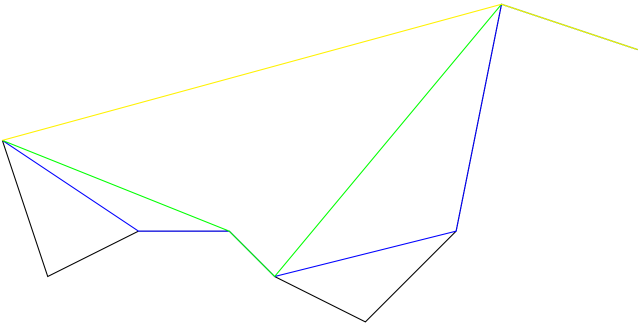

Y
?
N

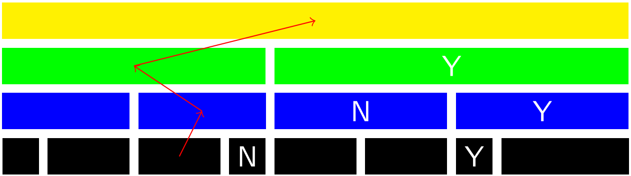


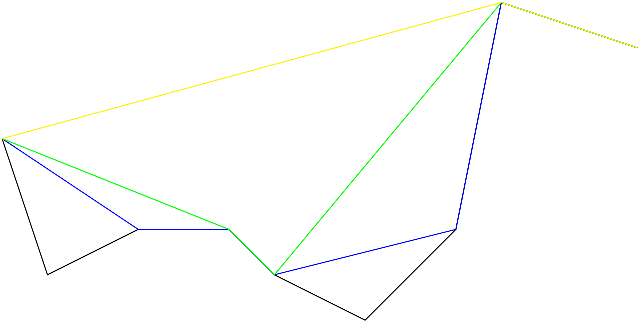

Y
N
?
N

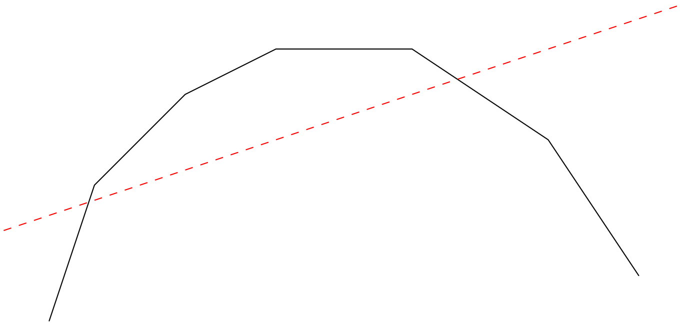


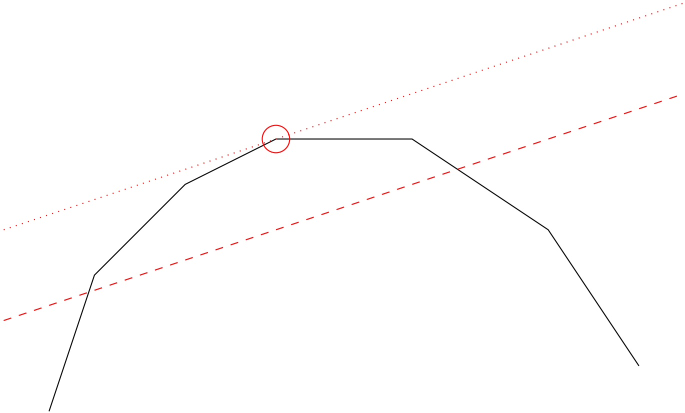

Y
N
Y
N
?

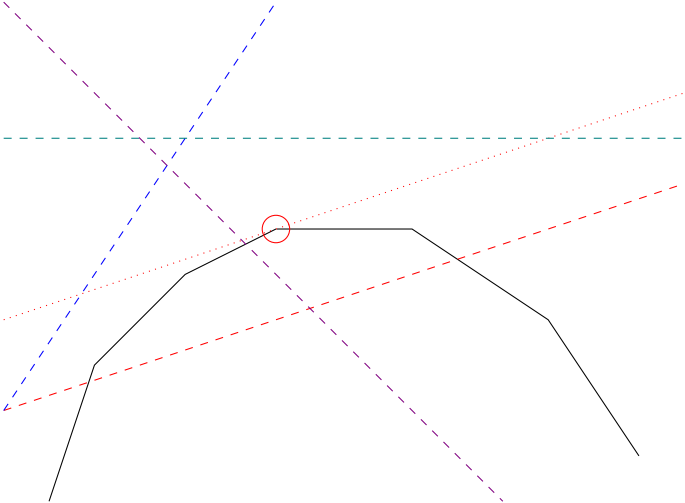


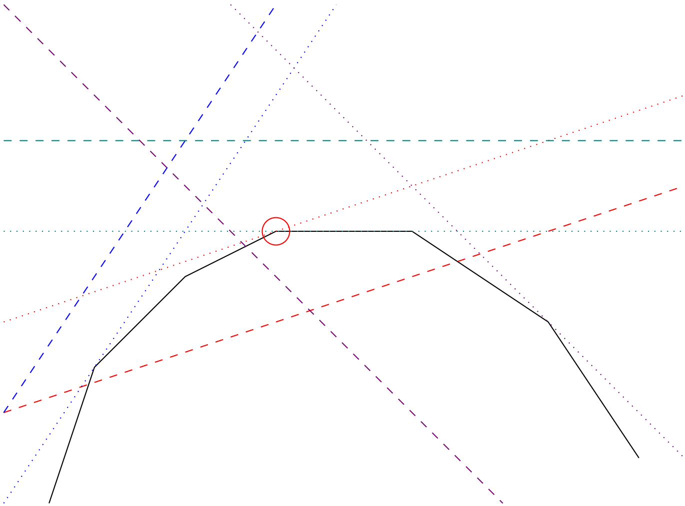

Y
N
Y
N
Y

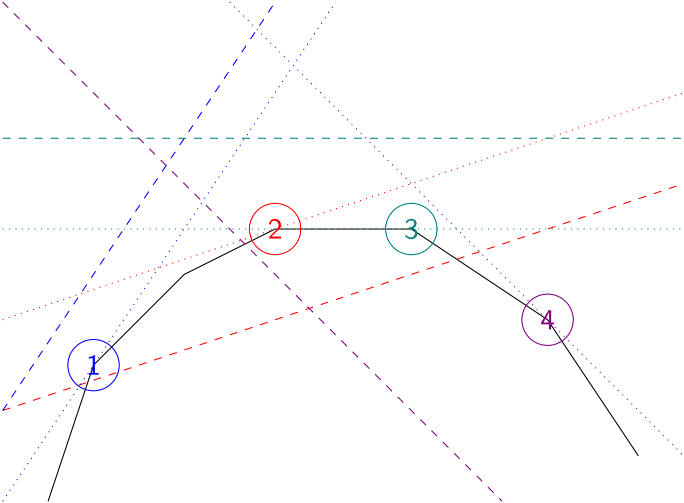


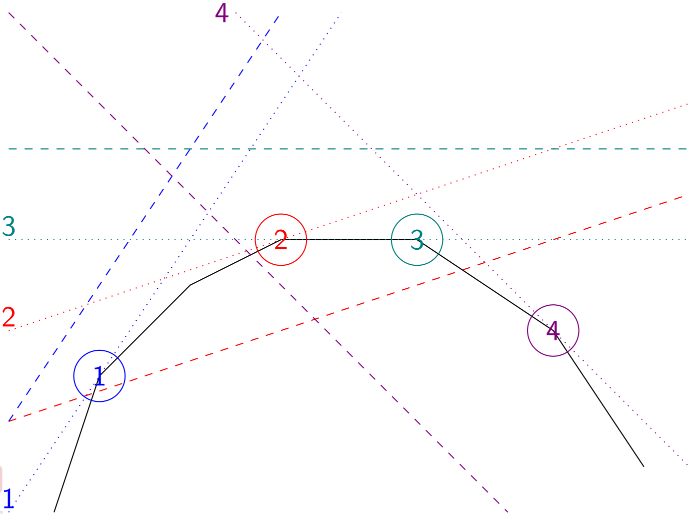




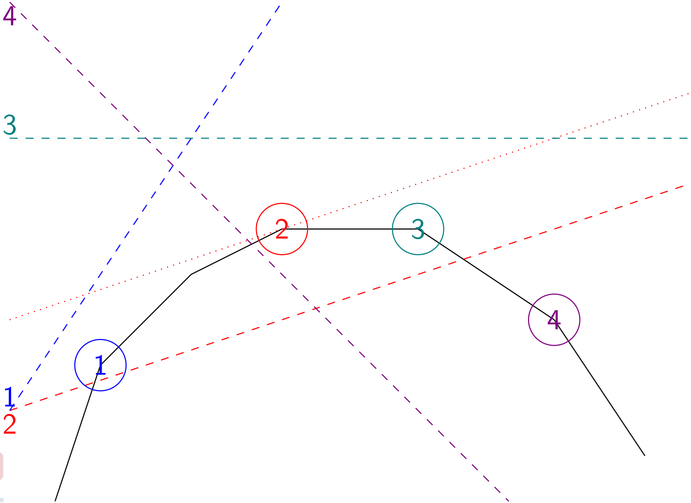





1
2
3
4



4
3
2
1
2
3
1
4


4
3
1
2
1
2
3
4

Problem G
Virus synthesis

Submits: 8
Accepted: 0

First solved by:

—

—

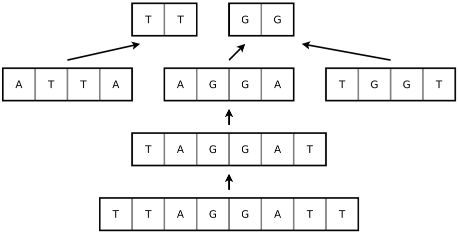
Author: Arkadiusz Pawlik

There is an algorithm for finding maximal palindromes in a given word, called *Manacher's algorithm*. During the execution, it enumerates *all* the palindromes.

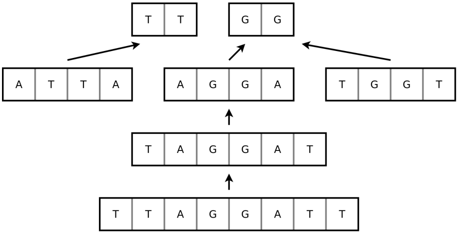
T	G	G	T	T	A	G	G	A	T	T	A

There is an algorithm for finding maximal palindromes in a given word, called *Manacher's algorithm*. During the execution, it enumerates *all* the palindromes.

T	G	G	T	T	A	G	G	A	T	T	A
			T	T		G	G				
T	G	G	T		A	G	G	A			
				T	A	G	G	A	T		
			T	T	A	G	G	A	T	T	
								A	T	T	A



Suppose that we have all the palindromes enumerated: $p_1, p_2, \ldots, p_s$. For every p_i we memorize $trim(p_i)$ – the palindrome p_i without first and last letter.



Suppose that we have all the palindromes enumerated: $p_1, p_2, \ldots, p_s$. For every p_i we memorize $trim(p_i)$ – the palindrome p_i without first and last letter.

Now, let us compute the minimal cost of synthesizing the words p_1, p_2, ..., p_s, in order of increasing lengths.

For every palindrome p_i we compute:

- $full[i]$ – cost of creating p_i.
- $half[i]$ – cost of creating half of p_i.

For every palindrome p_i we compute:

- $full[i]$ – cost of creating p_i.
- $half[i]$ – cost of creating half of p_i.

How can we obtain p_i?

A	A	C	C	A	A	C	C	A	A

For every palindrome p_i we compute:

- $\mathit{full}[i]$ – cost of creating p_i.
- $\mathit{half}[i]$ – cost of creating half of p_i.

How can we obtain p_i?

A	A	C	C	A	A	C	C	A	A

- by replicating $\mathit{half}[i]$: $\mathit{full}[i] = \mathit{half}[i] + 1$,

For every palindrome p_i we compute:

- $full[i]$ – cost of creating p_i.
- $half[i]$ – cost of creating half of p_i.

How can we obtain p_i?

A	A	C	C	A	A	C	C	A	A

- by replicating $half[i]$: $full[i] = half[i] + 1$,
- by synthesizing $trim[i]$ and adding first and last letter: $full[i] = full[trim(i)] + 2$,

For every palindrome p_i we compute:

- $\mathit{full}[i]$ – cost of creating p_i.
- $\mathit{half}[i]$ – cost of creating half of p_i.

How can we obtain p_i?

A	A	C	C	A	A	C	C	A	A

- by replicating $\mathit{half}[i]$: $\mathit{full}[i] = \mathit{half}[i] + 1$,
- by synthesizing $\mathit{trim}[i]$ and adding first and last letter: $\mathit{full}[i] = \mathit{full}[\mathit{trim}(i)] + 2$,
- by synthesizing palindrome p_k that is a preffix of p_i and adding some letters at the end: $\mathit{full}[i] = \mathit{full}[k] + \mathit{length}(i) - \mathit{length}(k)$.

For every palindrome p_i we compute:

- $\mathit{full}[i]$ – cost of creating p_i.
- $\mathit{half}[i]$ – cost of creating half of p_i.

How can we obtain first half of p_i?

A	A	C	C	A	A	C	C	A	A

For every palindrome p_i we compute:

- $\mathit{full}[i]$ – cost of creating p_i.
- $\mathit{half}[i]$ – cost of creating half of p_i.

How can we obtain first half of p_i?

A	A	C	C	A	A	C	C	A	A

- by adding the first letter to half of $\mathit{trim}(i)$: $\mathit{half}[i] = \mathit{half}[\mathit{trim}(i)] + 1$,

For every palindrome p_i we compute:

- $full[i]$ – cost of creating p_i.
- $half[i]$ – cost of creating half of p_i.

How can we obtain first half of p_i?

A	A	C	C	A	A	C	C	A	A

- by adding the first letter to half of $trim(i)$: $half[i] = half[trim(i)] + 1$,
- by synthesizing palindrome p_l that is a prefix of $half(p_i)$ and adding some letters at the end: $half[i] = full[l] + length(i)/2 - length(l)$.

If we know the cost of synthesizing all palindromes in the word, it is easy to compute the cost of the whole word.

We can only obtain the word from its palindrome subword – we simply check all of them.

This solution has complexity $O(n \log n)$. The $\log n$ factor comes from searching for prefix palindromes (not trivial, but quite easy).

Credits

Thanks to our betareaders and betatesters:

Szymon Gut
Grzegorz Gutowski
Witold Jarnicki
Robert Obryk

Credits

. . . and to all of you for solving the problems.

Thank you!