AMPPZ problems analysis 2016 (Moscow ACM ICPC Workshop edition)

Competition Jury

Institute of Computer Science, University of Wrocław

17 listopada 2016

J – Sharing Chocolate

Problem

We have n rectangular chocolates with sizes $h_i \times w_i$. We perform k *bisplitting* operations: choose the largest chocolate and split it into four roughly equal rectangular parts. Find the largest chocolate left after performing k opeartions for several values of k.

J – Sharing Chocolate

Suppose we have a single chocolate. Note that after performing several operations the number of different chocolate sizes will be small since each dimension of a new chocolate won't be far from the half the dimension of the old one.

J – Sharing Chocolate

Suppose we have a single chocolate. Note that after performing several operations the number of different chocolate sizes will be small since each dimension of a new chocolate won't be far from the half the dimension of the old one.
We will store a `priority_queue` that stores all different chocolate sizes along with the number for each size. Now we can perform the split simultaneously for all chocolates with the same size.

J – Sharing Chocolate

Suppose we have a single chocolate. Note that after performing several operations the number of different chocolate sizes will be small since each dimension of a new chocolate won't be far from the half the dimension of the old one.
We will store a `priority_queue` that stores all different chocolate sizes along with the number for each size. Now we can perform the split simultaneously for all chocolates with the same size.
Since there are not too many different sizes, this works fast.

D – Minimal Support of Transportation

For a given weighted graph find the minimal subset of edges that intersects with any MST.

Consider Kruskal algorithm for building MST. On its first step Kruskal's algorithm considers E' — the set of all edges of minimal weight. It then takes the maximal number of edges of E' so that not to form cycles. Then it "merges" the ends of each edge of E' and proceeds to the next weight.

Consider Kruskal algorithm for building MST. On its first step Kruskal's algorithm considers E' — the set of all edges of minimal weight. It then takes the maximal number of edges of E' so that not to form cycles. Then it "merges" the ends of each edge of E' and proceeds to the next weight.
If E' is not a spanning subgraph, MST will have to contain at least one edge of greater weight. We can skip the edges of E' altogether and proceed to greater weights.

D – Minimal Support of Transportation

A *global cut* of a graph is a way to partition its vertices into two non-empty halves S and T. The size of the cut is the number of edges between S and T.

D – Minimal Support of Transportation

A *global cut* of a graph is a way to partition its vertices into two non-empty halves S and T. The size of the cut is the number of edges between S and T.

Fact

The size of a minimal subset intersecting with any spanning tree of a graph is equal to the size of its minimal global cut.

D – Minimal Support of Transportation

A *global cut* of a graph is a way to partition its vertices into two non-empty halves S and T. The size of the cut is the number of edges between S and T.

Fact

The size of a minimal subset intersecting with any spanning tree of a graph is equal to the size of its minimal global cut.

we can find minimal global cut with a number of different algorithms, such as Stoer-Wagner algorithm, Karger's randomized method or simply finding a max-flow between a certain vertex s and all other vertices.

K – Johhny–Bohr model (Maciej Duleba)

Zadanie

Given a small (multi)set of natural numbers B ($|B| \leq 10$), we define a set A as follows:

- $n \in A$ ($n \leq 10^{15}$),
- $\left(\forall_{x \in \mathbb{N} \cup \{0\}}\right) (x \in A) \Rightarrow (\forall_{b \in B}) \left(\frac{x}{b} \in A\right)$.

What is the smallest possible cardinality of A?

K – Johhny–Bohr model (Maciej Duleba)

Zadanie

Given a small (multi)set of natural numbers B ($|B| \leq 10$), we define a set A as follows:

- $n \in A$ ($n \leq 10^{15}$),
- $\left(\forall_{x \in \mathbb{N} \cup \{0\}}\right)(x \in A) \Rightarrow (\forall_{b \in B})\left(\frac{x}{b} \in A\right)$.

What is the smallest possible cardinality of A?

Observation

The answer is always rather small.

K – Johhny–Bohr model (Maciej Duleba)

- The worst case is when B is the smallest ten prime numbers. Then, $|A| = 458123$.

K – Johhny–Bohr model (Maciej Duleba)

- The worst case is when B is the smallest ten prime numbers. Then, $|A| = 458123$.
- We can afford to explicitly generate all the elements of A.

K – Johhny–Bohr model (Maciej Duleba)

- The worst case is when B is the smallest ten prime numbers. Then, $|A| = 458123$.
- We can afford to explicitly generate all the elements of A.
- We need to store the already generated **distinct** numbers (for example with a `std::unordered_set`), beware of repetitions in B and the special case of $1 \in B$.

G – Parking Lot (Karol Pokorski)

Problem

Implement a data structure to simulate parallel parking on a line:

- parking in a shortest gap between two cars (if there is a tie: the leftmost),
- leaving the parking.

G – Parking Lot (Karol Pokorski)

Problem

Implement a data structure to simulate parallel parking on a line:

- parking in a shortest gap between two cars (if there is a tie: the leftmost),
- leaving the parking.

- Registration plates are converted to ids (for example, with `std::unordered_map`).

G – Parking Lot (Karol Pokorski)

Problem

Implement a data structure to simulate parallel parking on a line:

- parking in a shortest gap between two cars (if there is a tie: the leftmost),
- leaving the parking.

- Registration plates are converted to ids (for example, with `std::unordered_map`).
- Gaps are stored in two structures (for example, in a `std::set` with an appropriately modified comparator):
  - sorted according to the positions of their beginning,
  - sorted according to the length (if there is a tie: position of their beginning).

G – Parking Lot (Karol Pokorski)

Problem

Implement a data structure to simulate parallel parking on a line:

- parking in a shortest gap between two cars (if there is a tie: the leftmost),
- leaving the parking.

- Registration plates are converted to ids (for example, with `std::unordered_map`).
- Gaps are stored in two structures (for example, in a `std::set` with an appropriately modified comparator):
  - sorted according to the positions of their beginning,
  - sorted according to the length (if there is a tie: position of their beginning).
- We also store ids of the currently parked cars.

G – Parking Lot (Karol Pokorski)

- When a car arrives:
  - find the shortest gap (or output `NIE`),
  - remove the gap from both structures,
  - and possibly insert the new shorter gap there.

G – Parking Lot (Karol Pokorski)

- When a car arrives:
    - find the shortest gap (or output `NIE`),
    - remove the gap from both structures,
    - and possibly insert the new shorter gap there.
- When a car departs:
    - check if the car is present (and possibly output `BRAK`),
    - remove the adjacent gap from both structures,
    - insert the new gap (or, possible, merge the two adjacent gaps).

C – Generating Polygons (Karol Pokorski)

Problem

Generate a polygon with all sides parallel to the coordinate axes with a specified area ($1 \leq a \leq 10^{12}$) and perimeter ($4 \leq p \leq 10^6$).

C – Generating Polygons (Karol Pokorski)

Problem

Generate a polygon with all sides parallel to the coordinate axes with a specified area ($1 \leq a \leq 10^{12}$) and perimeter ($4 \leq p \leq 10^6$).

- Multiple possible solutions. We will present just one of them (possibly not the simplest one).

C – Generating Polygons (Karol Pokorski)

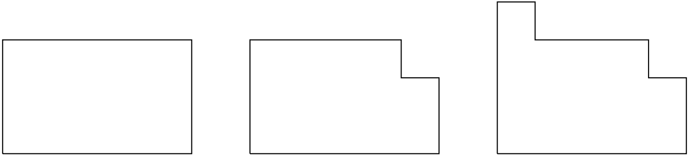
Problem

Generate a polygon with all sides parallel to the coordinate axes with a specified area ($1 \leq a \leq 10^{12}$) and perimeter ($4 \leq p \leq 10^6$).

- Multiple possible solutions. We will present just one of them (possibly not the simplest one).
- Find a rectangle with area and perimeter similar to the desired values (with a binary search or simple formula).

We try to make small local changes to the rectangle. The modifications are of two kinds:

- change the area, but leave the perimeter intact,
- change the perimeter by 2 and the area by 1.



H – Anagrams (Karol Pokorski)

Problem

Construct the shortest word with exactly n anagrams ($n \leq 10^{12}$).

H – Anagrams (Karol Pokorski)

Problem

Construct the shortest word with exactly n anagrams ($n \leq 10^{12}$).

Formula for the number of anagrams

$$m = \binom{a_1}{a_1} \cdot \binom{a_1 + a_2}{a_2} \cdot \binom{a_1 + a_2 + a_3}{a_3} \cdot \ldots$$

where a_i is the number of occurrences of the letter i.

H – Anagrams (Karol Pokorski)

Problem

Construct the shortest word with exactly n anagrams ($n \leq 10^{12}$).

Formula for the number of anagrams

$$m = \binom{a_1}{a_1} \cdot \binom{a_1 + a_2}{a_2} \cdot \binom{a_1 + a_2 + a_3}{a_3} \cdot \ldots$$

where a_i is the number of occurrences of the letter i.

Spoiler

We will try to backtractk.

Question

How many distinct letter might be required?
In particular, is restricting the alphabet to a–z a possible problem?

H – Anagrams (Karol Pokorski)

Question

How many distinct letter might be required?
In particular, is restricting the alphabet to a–z a possible problem?

Answer

No! Word with k distinct letter has at least $k!$ anagrams.

$$15! > 10^{12}$$

H – Anagrams (Karol Pokorski)

- We insert the letters one-by-one assuming that $a_1 \geq a_2 \geq a_3 \geq \ldots$.
- After inserting the next letter, the current number of anagrams must be a divisor of n. Otherwise, we can immediately terminate.

H – Anagrams (Karol Pokorski)

- We insert the letters one-by-one assuming that $a_1 \geq a_2 \geq a_3 \geq \dots$.
- After inserting the next letter, the current number of anagrams must be a divisor of n. Otherwise, we can immediately terminate.
- If $a_1 = 1$ then the situation is simple because $a_i = 1$,

H – Anagrams (Karol Pokorski)

- We insert the letters one-by-one assuming that $a_1 \geq a_2 \geq a_3 \geq \ldots$.
- After inserting the next letter, the current number of anagrams must be a divisor of n. Otherwise, we can immediately terminate.
- If $a_1 = 1$ then the situation is simple because $a_i = 1$,
- If $a_1 > 1$ (but not too large) then the situation is also simple because $a_1 \geq a_2 \geq a_3 \geq \ldots$ (there are few possibilities for the values $a_2, a_3, \ldots$),

H – Anagrams (Karol Pokorski)

- We insert the letters one-by-one assuming that $a_1 \geq a_2 \geq a_3 \geq \dots$.
- After inserting the next letter, the current number of anagrams must be a divisor of n. Otherwise, we can immediately terminate.
- If $a_1 = 1$ then the situation is simple because $a_i = 1$,
- If $a_1 > 1$ (but not too large) then the situation is also simple because $a_1 \geq a_2 \geq a_3 \geq \dots$ (there are few possibilities for the values a_2, a_3, $\dots$),
- If $a_1 >> 1$ (really large) then the situation is also not too bad because after adding a_2 occurrences of `b` the current number of anagrams $\binom{a_1+a_2}{a_2}$ will be very large. So there are few possibilities for the remaining choices, because the final product of binomial coefficients must be equal to n.

A – Weak pseudorandom generator (Karol Pokorski)

Problem

Given a linear congruential generator

$$t_n = (a \cdot t_{n-1} + b) \bmod p$$

find x such that $t_x = N$.

Problem

Given a linear congruential generator

$$t_n = (a \cdot t_{n-1} + b) \bmod p$$

find x such that $t_x = N$.

- Forget about the modulo for the time being.
- $t_1 = a \cdot t_0 + b$.
- $t_2 = a \cdot t_1 + b = a \cdot (a \cdot t_0 + b) + b = a^2 \cdot t_0 + ab + b$.
- $t_n = a^n \cdot t_0 + b \cdot (a^{n-1} + a^{n-2} + \cdots + a^0)$.

Explicit formula for t_m

$$t_n = a^n \cdot t_0 + b \cdot \frac{a^n - 1}{a - 1}$$

Explicit formula for t_m

$$t_n = a^n \cdot t_0 + b \cdot \frac{a^n - 1}{a - 1}$$

- We know t_n, t_0, a, b, so we only have to do some simple manipulations to obtain a^n.

A – Weak pseudorandom generator (Karol Pokorski)

Explicit formula for t_m

$$t_n = a^n \cdot t_0 + b \cdot \frac{a^n - 1}{a - 1}$$

- We know t_n, t_0, a, b, so we only have to do some simple manipulations to obtain a^n.

Solving for a^n

$$a^n = \frac{t_n \cdot (a-1) + b}{t_0 \cdot (a-1) + b}$$

- Now we bring back the modulo and replace division with modular multiplicative inverse.
- n can be extracted by taking a discrete logarithm. This can be solved with the baby–step giant–step algorithm in time $O(\sqrt{p} \cdot \log p)$.
- Beware of division by 0, (multiple) corner cases, and use `long long`.

E – Taxi (Karol Pokorski)

Zadanie

We have n taxi firms ($\leq 10^5$). Each taxi of i-th firm can fit c_i people (≤ 15), and a ride that is x kilometers long is worth $s_i + p_i \cdot (x - 1)$.
We are given q queries ($\leq 10^5$) for driving m_i people ($\leq 10^6$) at d_i kilometers ($\leq 10^6$). For each query find the optimal cost to transporting all people with (possibly several) taxi firms.

E – Taxi (Karol Pokorski)

- Separate the firms in groups by capacity. Let's consider each group separately.

E – Taxi (Karol Pokorski)

- Separate the firms in groups by capacity. Let's consider each group separately.
- The cost for hiring each firm is linear in the distance.
- Find the convex hull of these linear functions. We now know the distance intervals such that a particular taxi firm has the optimal price.

E – Taxi (Karol Pokorski)

- Separate the firms in groups by capacity. Let's consider each group separately.
- The cost for hiring each firm is linear in the distance.
- Find the convex hull of these linear functions. We now know the distance intervals such that a particular taxi firm has the optimal price.
- Answer the queries offline, initially sort them by distance.
- We can store pointers to optimal firms for each capacity, and maintain them while increasing the distance. We now only have to consider only ≤ 15 firms with different capacities.

Observation

Each query is now a knapsack problem: we have to collect total mass $\leq m_i$ using items (taxi firms) with weights c_i and costs d_i.

E – Taxi (Karol Pokorski)

Observation

Each query is now a knapsack problem: we have to collect total mass $\leq m_i$ using items (taxi firms) with weights c_i and costs d_i.

Problem

Solving this problem naively works in $O(\max(c_i) \cdot m_i)$.

E – Taxi (Karol Pokorski)

- Consider a taxi firm that offers the best per-passenger cost. Suppose that this firm has capacity C. Note that we shouldn't use $\geq C$ taxis from any other firm since we can interchange them with the "optimal" firm and optimize the cost.

E – Taxi (Karol Pokorski)

- Consider a taxi firm that offers the best per-passenger cost. Suppose that this firm has capacity C. Note that we shouldn't use $\geq C$ taxis from any other firm since we can interchange them with the "optimal" firm and optimize the cost.

Solution

We can now reduce the backpack capacity to C^2. The rest will be filled with taxis of capacity C.
We have to try all possibilities for the total weights of "small" taxis.

F – Ansisocial network (Paweł Gawrychowski)

Zadanie

We have an undirected graph on $n \leq 250\,000$ vertices and $m \leq 2\,000\,000$ edges. We have many operations like: for given $v_1, v_2, \ldots, v_s$ flip edges (v_i, v_j) for all $1 \leq i < j \leq s$. Which vertices are isolated in the resulting graph?

F – Ansisocial network (Paweł Gawrychowski)

- Choose a number $x_u \in \{0, 1\}$ randomly for each vertex u.

F – Ansisocial network (Paweł Gawrychowski)

- Choose a number $x_u \in \{0, 1\}$ randomly for each vertex u.
- Let E' be the set of edges of the resulting graph. If the vertex v if isolated then $\sum_{(u,v)\in E'} x_u = 0 \bmod 2$.

F – Ansisocial network (Paweł Gawrychowski)

- Choose a number $x_u \in \{0,1\}$ randomly for each vertex u.
- Let E' be the set of edges of the resulting graph. If the vertex v if isolated then $\sum_{(u,v)\in E'} x_u = 0 \bmod 2$.
- ... and if it isn't then $\sum_{(u,v)\in E'} x_u \neq 0 \bmod 2$ with probability $\frac{1}{2}$!

- Choose a number $x_u \in \{0, 1\}$ randomly for each vertex u.
- Let E' be the set of edges of the resulting graph. If the vertex v if isolated then $\sum_{(u,v)\in E'} x_u = 0 \bmod 2$.
- ...and if it isn't then $\sum_{(u,v)\in E'} x_u \neq 0 \bmod 2$ with probability $\frac{1}{2}$!
- Let's call this value the sum of $s(v)$. First we can count the sums in the original graph.

- Choose a number $x_u \in \{0,1\}$ randomly for each vertex u.
- Let E' be the set of edges of the resulting graph. If the vertex v if isolated then $\sum_{(u,v)\in E'} x_u = 0 \bmod 2$.
- ... and if it isn't then $\sum_{(u,v)\in E'} x_u \neq 0 \bmod 2$ with probability $\frac{1}{2}$!
- Let's call this value the sum of $s(v)$. First we can count the sums in the original graph.
- To update all sums after one operation in $O(s)$ time, we have to count $t = \sum_{i=1}^{s} x_{v_i} \bmod 2$, and then add $t + x_{v_i} \bmod 2$ to each $s(v_i)$.

F – Ansisocial network (Paweł Gawrychowski)

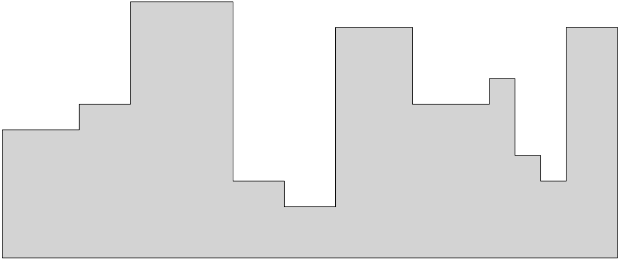
- Choose a number $x_u \in \{0, 1\}$ randomly for each vertex u.
- Let E' be the set of edges of the resulting graph. If the vertex v if isolated then $\sum_{(u,v)\in E'} x_u = 0 \bmod 2$.
- ...and if it isn't then $\sum_{(u,v)\in E'} x_u \neq 0 \bmod 2$ with probability $\frac{1}{2}$!
- Let's call this value the sum of $s(v)$. First we can count the sums in the original graph.
- To update all sums after one operation in $O(s)$ time, we have to count $t = \sum_{i=1}^{s} x_{v_i} \bmod 2$, and then add $t + x_{v_i} \bmod 2$ to each $s(v_i)$.
- To be safe, we can repeat the whole procedure 50 times. In fact, you can perform all these procedures by storing a random `long long` for each vertex and using `xor` instead of modulo 2 sum.

Zadanie

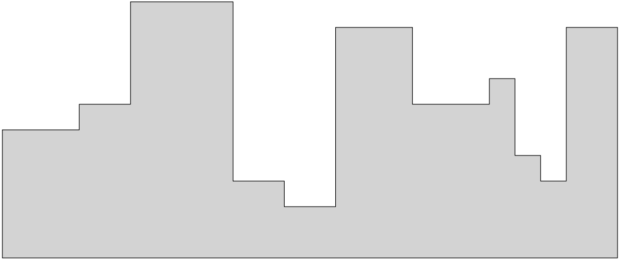
We have a rooted tree on n vertices, $n \leq 5\,000$. Vertex i has bandwidth c_i and initially contains s_i tokens that we want to move to the root. Each second each vertex moves at most c_i tokens it contains to its parent. How many seconds will pass until tokens end up in the root?

For each vertex let's store the number of transmitted tokens for each second. The can be described as a graph of a function.

For each vertex let's store the number of transmitted tokens for each second. The can be described as a graph of a function.



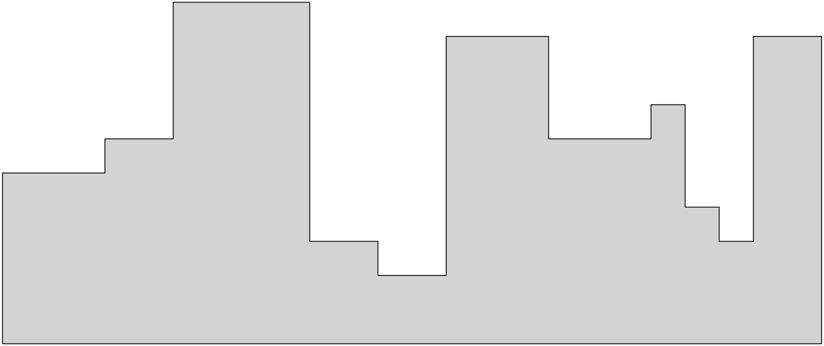
For each vertex let's store the number of transmitted tokens for each second. The can be described as a graph of a function.



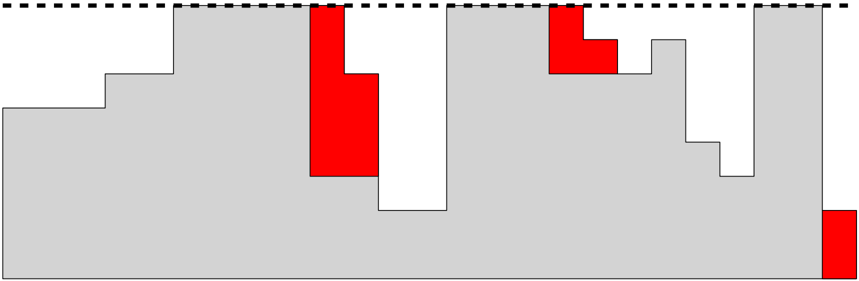
The key observation

This graph contains $O(n)$ steps.

The only operation we need is to build the graph for a vertex u given the graphs for each of its children. First we sum up all the children's functions. Then we take the limit c_u into account.



The only operation we need is to build the graph for a vertex u given the graphs for each of its children. First we sum up all the children's functions. Then we take the limit c_u into account.



- We can perform such "truncation" in $O(n)$ if we scan from left to right.

- We can perform such "truncation" in $O(n)$ if we scan from left to right.
- Summing two structures is effectively merging two sorted lists and can be done $O(n)$ as well.

- We can perform such "truncation" in $O(n)$ if we scan from left to right.
- Summing two structures is effectively merging two sorted lists and can be done $O(n)$ as well.
- We have to make $O(n)$ operations of these kinds, thus the total complexity is $O(n^2)$ (?).

I – Mountain Hike (Jakub Tarnawski)

Problem

We are given an acyclic graph on n vertices and m weighted edges, $n \leq 1\,000$, $m \leq 10\,000$, with selected vertices s_1, t_1, s_2, t_2. Find two vertex-distjoint paths $s_1 \to t_1$ and $s_2 \to t_2$ with the lowest possible total cost.

I – Mountain Hike (Jakub Tarnawski)

Problem

We are given an acyclic graph on n vertices and m weighted edges, $n \leq 1\,000$, $m \leq 10\,000$, with selected vertices s_1, t_1, s_2, t_2. Find two vertex-distjoint paths $s_1 \to t_1$ and $s_2 \to t_2$ with the lowest possible total cost.

- Sort the vertices by increasing altitude so that the new numbers are $1, 2, \ldots, n$.

Problem

We are given an acyclic graph on n vertices and m weighted edges, $n \leq 1\,000$, $m \leq 10\,000$, with selected vertices s_1, t_1, s_2, t_2. Find two vertex-distjoint paths $s_1 \to t_1$ and $s_2 \to t_2$ with the lowest possible total cost.

- Sort the vertices by increasing altitude so that the new numbers are $1, 2, \ldots, n$.
- We have to separately consider the cases when $|\{s_1, t_1, s_2, t_2\}| < 4$.

I – Mountain Hike (Jakub Tarnawski)

Problem

We are given an acyclic graph on n vertices and m weighted edges, $n \leq 1\,000$, $m \leq 10\,000$, with selected vertices s_1, t_1, s_2, t_2. Find two vertex-distjoint paths $s_1 \to t_1$ and $s_2 \to t_2$ with the lowest possible total cost.

- Sort the vertices by increasing altitude so that the new numbers are $1, 2, \dots, n$.
- We have to separately consider the cases when $|\{s_1, t_1, s_2, t_2\}| < 4$.
- We now construct a new graph with vertices being ordere pairs (u, v), $u \neq v$. We want to add edges in such a way that each path from (s_1, s_2) to (u, v) corresponds to to vertex-disjoint paths $s_1 \to u$ and $s_2 \to v$.

Problem

We are given an acyclic graph on n vertices and m weighted edges, $n \leq 1\,000$, $m \leq 10\,000$, with selected vertices s_1, t_1, s_2, t_2. Find two vertex-distjoint paths $s_1 \to t_1$ and $s_2 \to t_2$ with the lowest possible total cost.

- Sort the vertices by increasing altitude so that the new numbers are $1, 2, \ldots, n$.
- We have to separately consider the cases when $|\{s_1, t_1, s_2, t_2\}| < 4$.
- We now construct a new graph with vertices being ordere pairs (u, v), $u \neq v$. We want to add edges in such a way that each path from (s_1, s_2) to (u, v) corresponds to to vertex-disjoint paths $s_1 \to u$ and $s_2 \to v$.
- In a sense, the new graph simulates moving the vertices in parallel.

- There are tow cases. If $u < v$ or $v = t_2$ we try to guess the next vertex on the path that currently ends in u. To do that, we want to try all edges (u, u') such that $u' \neq v$. For each such of the original graph edge we add the edge in the new graph from (u, v) to (u', v) with the same cost.

I – Mountain Hike (Jakub Tarnawski)

- There are tow cases. If $u < v$ or $v = t_2$ we try to guess the next vertex on the path that currently ends in u. To do that, we want to try all edges (u, u') such that $u' \neq v$. For each such of the original graph edge we add the edge in the new graph from (u, v) to (u', v) with the same cost.
- ... If $v < u$ or $u = t_1$, try to move v forward using all edges (v, v') such that $v' \neq u$.

- There are tow cases. If $u < v$ or $v = t_2$ we try to guess the next vertex on the path that currently ends in u. To do that, we want to try all edges (u, u') such that $u' \neq v$. For each such of the original graph edge we add the edge in the new graph from (u, v) to (u', v) with the same cost.
- ... If $v < u$ or $u = t_1$, try to move v forward using all edges (v, v') such that $v' \neq u$.
- Now we simple find the shortest path from (s_1, s_2) to (t_1, t_2) in the new graph!

I – Mountain Hike (Jakub Tarnawski)

- There are tow cases. If $u < v$ or $v = t_2$ we try to guess the next vertex on the path that currently ends in u. To do that, we want to try all edges (u, u') such that $u' \neq v$. For each such of the original graph edge we add the edge in the new graph from (u, v) to (u', v) with the same cost.
- ...If $v < u$ or $u = t_1$, try to move v forward using all edges (v, v') such that $v' \neq u$.
- Now we simple find the shortest path from (s_1, s_2) to (t_1, t_2) in the new graph!
- The new graph is acyclic, thus instead of Dijkstra we can implement DP that processed the vertices in the topological order in total linear time.

I – Mountain Hike (Jakub Tarnawski)

- There are tow cases. If $u < v$ or $v = t_2$ we try to guess the next vertex on the path that currently ends in u. To do that, we want to try all edges (u, u') such that $u' \neq v$. For each such of the original graph edge we add the edge in the new graph from (u, v) to (u', v) with the same cost.
- ...If $v < u$ or $u = t_1$, try to move v forward using all edges (v, v') such that $v' \neq u$.
- Now we simple find the shortest path from (s_1, s_2) to (t_1, t_2) in the new graph!
- The new graph is acyclic, thus instead of Dijkstra we can implement DP that processed the vertices in the topological order in total linear time.
- New graph has $O(nm)$ edges, thus the solution has the same complexity.

B – Home Alone: Johnny Lost in New York (Paweł Gawrychowski)

Problem

Build a Hamiltonian path between cells (s_x, s_y) and (t_x, t_y) on an $n \times m$ grid, $4 \leq n, m \leq 1\,000$.

B – Home Alone: Johnny Lost in New York (Paweł Gawrychowski)

Problem

Build a Hamiltonian path between cells (s_x, s_y) and (t_x, t_y) on an $n \times m$ grid, $4 \leq n, m \leq 1\,000$.

Observation

Adjacent cells have different colors. Thus (s_x, s_y) and (t_x, t_y) must have different colors exactly when $n \cdot m$ is even.

B – Home Alone: Johnny Lost in New York (Paweł Gawrychowski)

Problem

Build a Hamiltonian path between cells (s_x, s_y) and (t_x, t_y) on an $n \times m$ grid, $4 \leq n, m \leq 1\,000$.

Observation

Adjacent cells have different colors. Thus (s_x, s_y) and (t_x, t_y) must have different colors exactly when $n \cdot m$ is even.

$n, m \geq 4$

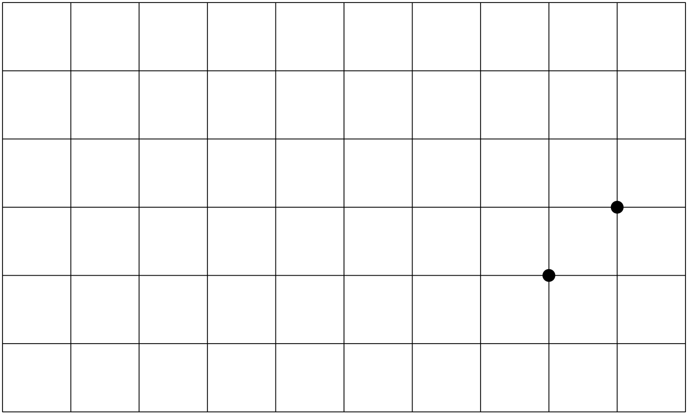
If that condition holds, the answer always exists.

- . . . We can perform brute-force if $n, m \in \{4, 5, 6\}$. But is there a proof? And how to construct the path?

B – Home Alone: Johnny Lost in New York (Paweł Gawrychowski)

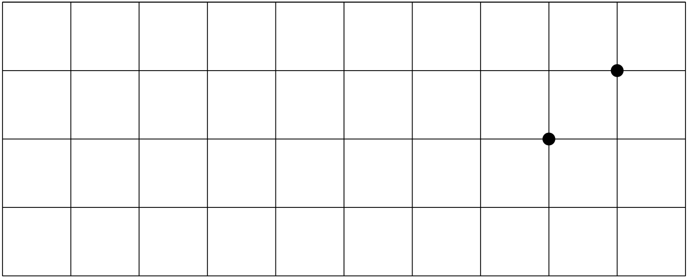
- ... We can perform brute-force if $n, m \in \{4, 5, 6\}$. But is there a proof? And how to construct the path?
- Let's do an induction argument by $n \cdot m$.

Let $n \geq 7$ and the top two rows don't contain start or finish points.



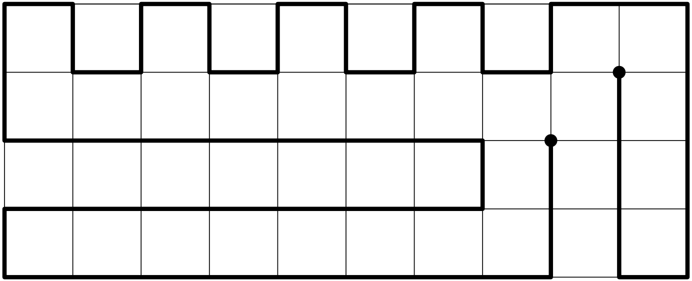
B – Home Alone: Johnny Lost in New York (Paweł Gawrychowski)

Let $n \geq 7$ and the top two rows don't contain start or finish points.



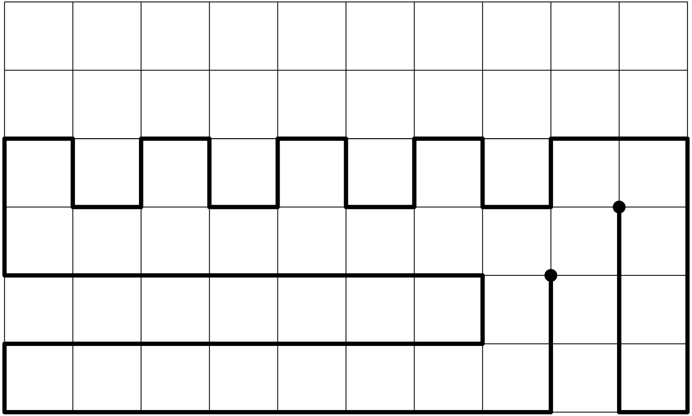
B – Home Alone: Johnny Lost in New York (Paweł Gawrychowski)

Let $n \geq 7$ and the top two rows don't contain start or finish points.



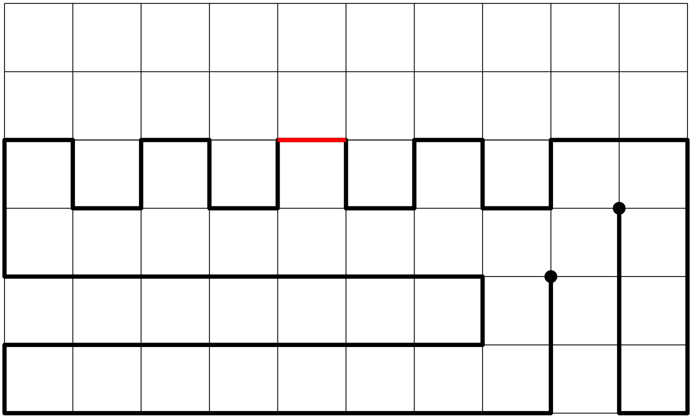
B – Home Alone: Johnny Lost in New York (Paweł Gawrychowski)

Let $n \geq 7$ and the top two rows don't contain start or finish points.



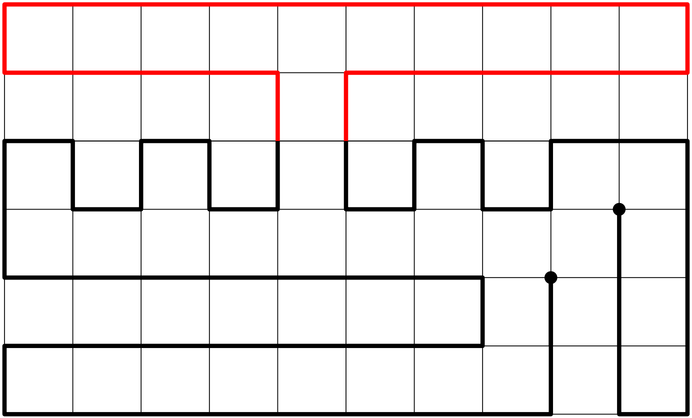
B – Home Alone: Johnny Lost in New York (Paweł Gawrychowski)

Let $n \geq 7$ and the top two rows don't contain start or finish points.



B – Home Alone: Johnny Lost in New York (Paweł Gawrychowski)

Let $n \geq 7$ and the top two rows don't contain start or finish points.



- The same trick also works if the bottom two rows don't contain start or finish.

B – Home Alone: Johnny Lost in New York (Paweł Gawrychowski)

- The same trick also works if the bottom two rows don't contain start or finish.
- Thus we only have to deal with the case when the start is in the top two rows, and the finish is not in top three rows (remember that n is not too small).

B – Home Alone: Johnny Lost in New York (Paweł Gawrychowski)

- The same trick also works if the bottom two rows don't contain start or finish.
- Thus we only have to deal with the case when the start is in the top two rows, and the finish is not in top three rows (remember that n is not too small).
- We can reduce to the previous case by erasing the top two rows and find the answer recursively on the $(n-2) \times m$ grid, with the start point chosen among the third row in a suitable way.

The same construction applies when $m \geq 7$. If both n and m are small, we can simply perform brute-force. It is true that the answer exists whenever $n, m \geq 4$ and the parity condition is satisfied.

B – Home Alone: Johnny Lost in New York (Paweł Gawrychowski)

Complexity

A straightforward implementation that rotates, flips and pastes the path pieces works in $O((n+m)^3)$.

B – Home Alone: Johnny Lost in New York (Paweł Gawrychowski)

Complexity

A straightforward implementation that rotates, flips and pastes the path pieces works in $O((n+m)^3)$.

We need to do faster! The hardest part is to flip and rotate the path pieces.

B – Home Alone: Johnny Lost in New York (Paweł Gawrychowski)

Complexity

A straightforward implementation that rotates, flips and pastes the path pieces works in $O((n+m)^3)$.

We need to do faster! The hardest part is to flip and rotate the path pieces.

Better structure

We can store the path as a list of commands like "forward" or "turn left". Also introduce a command "reverse all further turn directions". Now flipping and pasting can be done in $O(1)$.

B – Home Alone: Johnny Lost in New York (Paweł Gawrychowski)

Complexity

A straightforward implementation that rotates, flips and pastes the path pieces works in $O((n+m)^3)$.

We need to do faster! The hardest part is to flip and rotate the path pieces.

Better structure

We can store the path as a list of commands like "forward" or "turn left". Also introduce a command "reverse all further turn directions". Now flipping and pasting can be done in $O(1)$.

The resulting solution works in $O(n^2)$.