Solution Outlines

SWERC Judges

SWERC 2010

Statistics

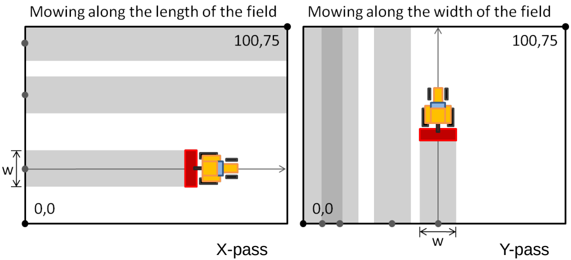
Problem	1^{st} team solving	Time
A - Lawnmower	Dirt Collector	15
B - Periodic points		
C - Comparing answers	Stack of Shorts	71
D - Fake scoreboard	Dirt Collector	256
E - Palindromic DNA	UPC-2	256
F - Jumping monkey	Dirt Collector	112
G - Sensor network		
H - Assembly line	Techies	131
I - Locks and keys	UMU Null	172
J - 3-sided dice	Techies	76

Statistics

Problem	AC	Total	Success Rate
A - Lawnmower	35	69	51%
B - Periodic points	0	6	0%
C - Comparing answers	6	122	5%
D - Fake scoreboard	1	26	4%
E - Palindromic DNA	1	5	20%
F - Jumping monkey	6	64	10%
G - Sensor network	0	10	0%
H - Assembly line	3	35	9%
I - Locks and keys	2	19	11%
J - 3-sided dice	3	106	3%

A: Lawnmower

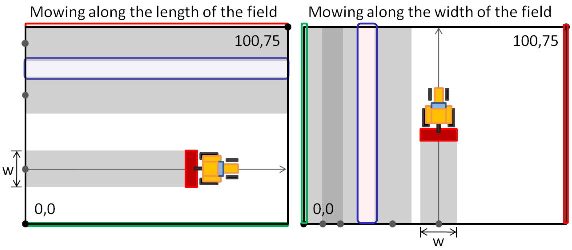
Idea original: Manuel Abellanas
Enunciado: Manuel Freire



Testcase OK **iff** X-passes OK **&&** Y-passes OK

Lawnmower

Solution



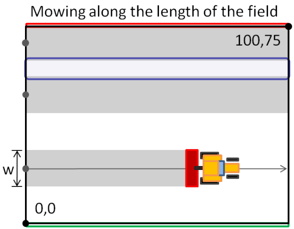
X-Pass OK **iff**

- Touches start-bound
  smallest pass starts at <= w/2
- Touches end-bound
  biggest pass starts at >= Y_END - w/2
- No 'gaps' between passes
  distance <= w between *adjacent* passes

(Same for Y-Pass)

Lawnmower

Solution



X-Pass OK **iff**

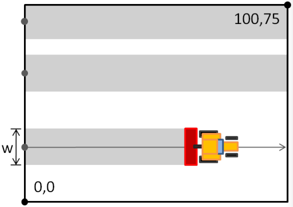
- Touches start-bound
  smallest pass starts at <= w/2
- Touches end-bound
  biggest pass starts at >= Y_END - w/2
- No 'gaps' between passes
  distance <= w between *adjacent* passes

so, SORT THOSE PASSES

Lawnmower

Solution

Mowing along the length of the field




```
#include <algorithm>
#include <cstdio>
#include <vector>
using namespace std;
bool test(float w, int max, int n) {
    vector<float> v(n);
    for (int i=0; i<n; i++) fscanf(stdin, "%f", &v[i]);
    sort(v.begin(), v.end());
    bool ok = (v[0] <= w/2) && ((max - v[n-1]) <= w/2);
    for (int i=0; i<n-1 && ok; i++) {
        if (v[i+1] - v[i] > w) ok = false;
    }
    return ok;
}
int main() {
    int n1, n2;
    float w;
    while(fscanf(stdin, "%d %d %f", &n1, &n2, &w), w > 0) {
        fprintf(stderr, "\t%d %d %f:\n", n1, n2, w);
        bool x = test(w, 75, n1), y = test(w, 100, n2);
        fprintf(stdout, "%s\n", x && y ? "YES" : "NO");
    }
    return 0;
}
```

X-Pass OK **iff**

- Touches start-bound
  smallest pass starts at <= w/2
- Touches end-bound
  biggest pass starts at >= Y_END - w/2
- No 'gaps' between passes
  distance <= w between *adjacent* passes

so, SORT THOSE PASSES

Periodic Points

Solution

Categories: Math + DP

Problem

Number of solutions to $f^n(x) = x$ in $[0, m] \Leftrightarrow$ number of intersections between the graph of f^n and the diagonal of the square $[0, m] \times [0, m]$.

Difficulty: It is impossible to store a description of f^n since data grows exponentially.

First Remark

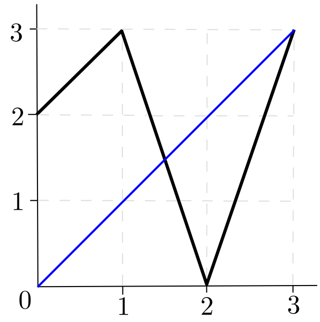
f^n is also piecewise linear, but the number of pieces may grow exponentially with n.

Periodic Points

Solution

Example from the statement

Graph of f, case $n = 1$.
Number of intersections $= 2$.

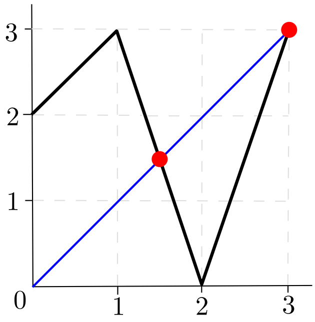


Periodic Points

Solution

Example from the statement

Graph of f, case $n = 1$.
Number of intersections $= 2$.

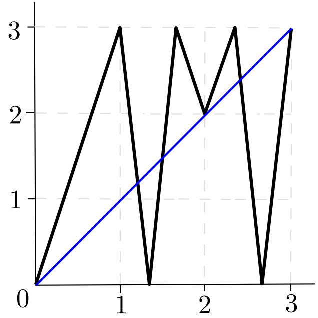


Periodic Points

Solution

Example from the statement

Graph of f^2, case $n = 2$.
Number of intersections $= 6$.

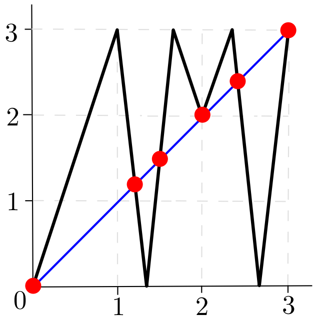


Periodic Points

Solution

Example from the statement

Graph of f^2, case $n = 2$.
Number of intersections $= 6$.

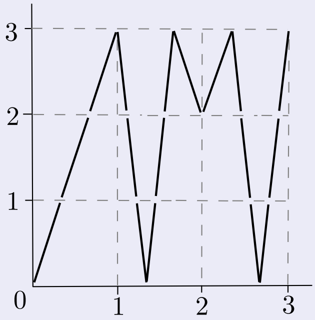


Periodic Points

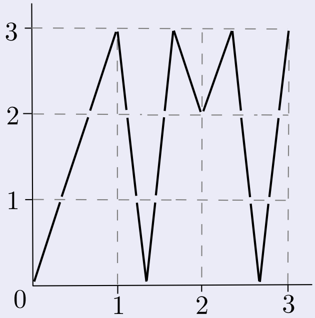
Solution

Idea

Graph of f^n consists of a union of linked sub-intervals going from $y = a$ to $y = a + 1$,a or $a - 1$.



Idea

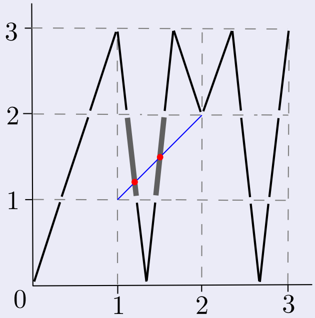


Focus on any unit interval:
Intersections between f^n and the diagonal in the interval $(k, k+1) =$ sub-intervals going from $y = k$ to $y = k + 1$ or vice-versa.

Periodic Points

Solution

Idea



Focus on any unit interval:

Intersections between f^n and the diagonal in the interval $(k, k+1) =$ sub-intervals going from $y = k$ to $y = k+1$ or vice-versa.

Periodic Points

Solution

Shape the idea

- Construct $m \times m$ matrix A

$$A_{i,j} = \begin{cases} 0 & \text{if } f([i, i+1]) \subset [j, j+1] \\ 1 & \text{otherwise} \end{cases} \tag{1}$$

Periodic Points

Solution

Shape the idea

- Construct $m \times m$ matrix A
- $A_{i,j}$ = number of subintervals between $y = j$ and $y = j + 1$ contained in the graph of f in $(i, i + 1)$

Periodic Points

Solution

Shape the idea

- Construct $m \times m$ matrix A
- $(A^n)_{i,j}$ = number of subintervals between $y = j$ and $y = j + 1$ contained in the graph of f^n in $(i, i + 1)$

Periodic Points

Solution

Shape the idea

- Construct $m \times m$ matrix A
- $(A^n)_{i,j}$ = number of subintervals between $y = j$ and $y = j + 1$ contained in the graph of f^n in $(i, i + 1)$
- Result: $A^n_{0,0} + A^n_{1,1} + \ldots A^n_{m-1,m-1} = trace(A^n)$

Shape the idea

- Construct $m \times m$ matrix A
- $(A^n)_{i,j}$ = number of subintervals between $y = j$ and $y = j + 1$ contained in the graph of f^n in $(i, i + 1)$
- Result: $A^n_{0,0} + A^n_{1,1} + \ldots A^n_{m-1,m-1} = trace(A^n)$
- Wait a moment! Attention to endpoints of $[k, k + 1]$

Periodic Points

Solution

Integer coordinate points

Compute if $f^n(k) = k$ for any $k \in \{0, 1, \ldots, m\}$.
Complexity: $O(m \cdot n)$ ad-hoc iteration, $O(m \cdot log(n))$ binary exp.

Algorithm

- Compute $trace(A^n)$ using binary exp ($O(m^3 \cdot log(n))$)
- Modify the answer because of integer coordinate points. Tricky!

Comparing answers

Solution

Categories: Linear Algebra, Randomized algorithms

Transform into matrix multiplication testing

- Matrix A. Entry $a_{i,j}$ is the number of roads from location i to location j.

Comparing answers
Solution

Categories: Linear Algebra, Randomized algorithms

Transform into matrix multiplication testing

- Matrix A. Entry $a_{i,j}$ is the number of roads from location i to location j.
- Matrix $B = A^2$. Entry $b_{i,j}$ is the number of paths of length 2 from location i to location j.

Comparing answers

Solution

Categories: Linear Algebra, Randomized algorithms

Transform into matrix multiplication testing

- Matrix A. Entry $a_{i,j}$ is the number of roads from location i to location j.
- Matrix $B = A^2$. Entry $b_{i,j}$ is the number of paths of length 2 from location i to location j.
- We are given the entries of a matrix C. We want to check $C = B^2$.

Comparing answers

Solution

Important observation

We do not need to compute B^2 to check if it is C!!

Comparing answers

Solution

Important observation

We do not need to compute B^2 to check if it is C!!

Witness for non-equality

Let x be a vector. If $B^2 x \neq Cx$, we call x a witness for non-equality of B^2 and C. Such a witness always exists whenever $B^2 \neq C$ (let x_i be 1 for the index of a column in which B^2 and C differ, and 0 everywhere else).

Comparing answers

Solution

Finding a witness

If $B^2 \neq C$, a randomly chosen vector will of elements in $\{0, \ldots, x\}$ be a witness with probability at least $1 - 1/x$.
Proof:

Testing whether a vector is a witness

It involves three multiplications of a matrix by a vector, which takes time $\Theta(n^2)$.

Comparing answers

Solution

Finding a witness

If $B^2 \neq C$, a randomly chosen vector will of elements in $\{0, \ldots, x\}$ be a witness with probability at least $1 - 1/x$.
Proof:
$B^2x \neq Cx \iff (B^2 - C)x = 0$. As $B^2 - C \neq 0$, let h be a row of $B^2 - C$ which has a nonzero element h_i. Then, $h^T x = 0$ implies $x_i = \dfrac{\sum_{j \neq i} h_j x_j}{c_i}$. But x_i is a random element in $\{0, \ldots, x\}$, so that is true with probability at most $\dfrac{1}{x+1}$.

Testing whether a vector is a witness

It involves three multiplications of a matrix by a vector, which takes time $\Theta(n^2)$.

Comparing answers

Solution

Final algorithm

- Pick a small constant number of random vectors in $\{0, \ldots, x\}^n$, for $x \geq 2$.
- Check whether for any of them $B^2 x \neq C$.
- If that is the case, return $B^2 \neq C$("NO"), otherwise return $B^2 = C$("YES").

Comparing answers

Solution

References

- Freivalds' Algorithm: Freivalds, R. Information Processing 77, Proceedings of IFIP Congress 77,

Fake scoreboard

Solution

Categories: greedy / max flow

Problem

Reconstruct a $0 - 1$ matrix from its row and column sums r_i, c_j. Output lexicographically smallest solution.

Fake scoreboard
Solution

Categories: greedy / max flow

Problem

Reconstruct a $0-1$ matrix from its row and column sums r_i, c_j. Output lexicographically smallest solution.

Reduction to decision version

- Is there a solution?
- Given a partialy-filled matrix, can it be extended to a solution?

If each of these can be answered in time T, we have an $O(n^2 T)$ algorithm for our task.

Fake scoreboard

First approach (greedy)

Decision algorithm

1. Focus on the topmost row; must have r_1 ones.
2. Sort column sums $c_1, \ldots, c_n$ in decreasing order.
3. Put ones in the r_1 columns with largest sums.
4. Decrease column sums and proceed with the next row.

Running time: $O(n^2 \log n)$. Does it work?

Fake scoreboard

First approach (greedy)

Decision algorithm

1. Focus on the topmost row; must have r_1 ones.
2. Sort column sums $c_1, \ldots, c_n$ in decreasing order.
3. Put ones in the r_1 columns with largest sums.
4. Decrease column sums and proceed with the next row.

Running time: $O(n^2 \log n)$. Does it work?
Take any solution; suppose $c_i \geq c_j$ and the first row looks like 01 on these columns.
Then some other row below must look like 10; exchanging the two values in both rows leads to another solution.

Fake scoreboard

First approach (greedy)

Decision algorithm

1. Focus on the topmost row; must have r_1 ones.
2. Sort column sums $c_1, \ldots, c_n$ in decreasing order.
3. Put ones in the r_1 columns with largest sums.
4. Decrease column sums and proceed with the next row.

Running time: $O(n^2 \log n)$. Does it work?
Take any solution; suppose $c_i \geq c_j$ and the first row looks like 01 on these columns.
Then some other row below must look like 10; exchanging the two values in both rows leads to another solution.
Repeat n^2 times, trying to place zeroes. Total time: $O(n^4 \log n)$ AC.
More careful implementation: $O(n^3 \log n)$.

Fake scoreboard

Second approach (max flow)

Decision algorithm

Build a graph with:

- a source, a sink, n row vertices and n column vertices;
- r_i units of flow from source to row i;
- edge of capacity one from row i to column j;
- c_j units of flow from column j to sink;

$\exists$ solution $\Leftrightarrow$ max. flow is $\sum r_i = \sum c_j$.

Fake scoreboard

Second approach (max flow)

Decision algorithm

Build a graph with:

- a source, a sink, n row vertices and n column vertices;
- r_i units of flow from source to row i;
- edge of capacity one from row i to column j;
- c_j units of flow from column j to sink;

$\exists$ solution $\Leftrightarrow$ max. flow is $\sum r_i = \sum c_j$.

- $2n + 2$ vertices, $n^2 + 2n$ edges.
- Unit network $\Rightarrow$ Dinic's algorithm takes $O(E\sqrt{V}) = O(n^{2.5})$ time.
- Repeat n^2 times. Overall: $\Omega(n^{4.5})$ TLE.

Fake scoreboard

Second approach: how to make it faster

- **Idea**: no need to compute max flow from scratch every time.
- If current solution does not use edge (i, j), there is a 0 already in that position.
- Otherwise, remove the edge and try to push the missing unit of flow through another path.
- We can write a 0 iff the last step succeeded.

Fake scoreboard

Second approach: how to make it faster

- **Idea**: no need to compute max flow from scratch every time.
- If current solution does not use edge (i, j), there is a 0 already in that position.
- Otherwise, remove the edge and try to push the missing unit of flow through another path.
- We can write a 0 iff the last step succeeded.

$O(n^{2.5})$ initial max-flow computation; $O(E) = O(n^2)$ additional for each decision.
Overall: $O(n^4)$ AC.
Fastest solution by far! (for input size $n \sim 80$)

Palindromic DNA

Categories: 2SAT

Problem

Transform a given string $s \in \{A, G, C, T\}^n$ subject to the following:

1. several given pairs of characters should be equal;
2. for each character $s[i]$, we can increase it, decrease it, or leave it unmodified ($A \implies G \implies C \implies T \implies A$)
3. cannot modify two consecutive characters

Palindromic DNA

Categories: 2SAT

Problem

Transform a given string $s \in \{A, G, C, T\}^n$ subject to the following:

1. several given pairs of characters should be equal;
2. for each character $s[i]$, we can increase it, decrease it, or leave it unmodified ($A \implies G \implies C \implies T \implies A$)
3. cannot modify two consecutive characters

Observations

For each pair of positions that should be equal:

- if $s[i] = s[j]$, need to apply same operation to both;
- if $dist(s[i], s[j]) = 1$, exactly one of them has to change (in the right direction);
- if $dist(s[i], s[j]) = 2$, both need to change in reverse directions.

Palindromic DNA

First solution

We can write all constraints in terms of two sets of variables x_i, y_i:

- $x_i = \textit{true}$ iff s_i is changed;
- $y_i = \textit{true}$ iff s_i is increased and *false* if it is decreased;

Palindromic DNA

First solution

We can write all constraints in terms of two sets of variables x_i, y_i:

- $x_i = \textit{true}$ iff s_i is changed;
- $y_i = \textit{true}$ iff s_i is increased and *false* if it is decreased;

- Each constraint involves just two variables $\implies$ 2SAT problem.
- We can write all constraints as sets of implications, e.g. $x_i \implies x_j$, $x_i \implies \overline{x_{i+1}}$ or $y_i \implies \overline{y_j}$.

Palindromic DNA

First solution

We can write all constraints in terms of two sets of variables x_i, y_i:

- $x_i = \mathit{true}$ iff s_i is changed;
- $y_i = \mathit{true}$ iff s_i is increased and *false* if it is decreased;

- Each constraint involves just two variables $\implies$ 2SAT problem.
- We can write all constraints as sets of implications, e.g. $x_i \implies x_j$, $x_i \implies \overline{x_{i+1}}$ or $y_i \implies \overline{y_j}$.
- There is no solution iff a contradiction arises, i.e. both $x_i \implies \bar{x}_i$ AND $\bar{x}_i \implies x_i$.
- Complexity: $O(\mathit{variables} + \mathit{constraints}) = O(n \cdot \mathit{subsets})$ using linear-time SCC algo AC.

Palindromic DNA

First solution

We can write all constraints in terms of two sets of variables x_i, y_i:

- $x_i = \textit{true}$ iff s_i is changed;
- $y_i = \textit{true}$ iff s_i is increased and *false* if it is decreased;

- Each constraint involves just two variables $\implies$ 2SAT problem.
- We can write all constraints as sets of implications, e.g. $x_i \implies x_j$, $x_i \implies \overline{x_{i+1}}$ or $y_i \implies \overline{y_j}$.
- There is no solution iff a contradiction arises, i.e. both $x_i \implies \bar{x}_i$ AND $\bar{x}_i \implies x_i$.
- Complexity: $O(\textit{variables} + \textit{constraints}) = O(n \cdot \textit{subsets})$ using linear-time SCC algo AC.

Naive SCC algo with n DFS's will time out, but...

Palindromic DNA

Second solution

- We can group together all positions that should have equal characters.
- There remain only $O(n)$ constraints between these clusters (each saying that two consecutive characters cannot be modified at the same time).

Palindromic DNA

Second solution

- We can group together all positions that should have equal characters.
- There remain only $O(n)$ constraints between these clusters (each saying that two consecutive characters cannot be modified at the same time).
- Try each of four possible assigments to elements in a cluster and recursively deal with implications to other clusters in a DFS fashion.
- If no contradiction is found, set this assignment and go on to next cluster (no backtracking).

Palindromic DNA

Second solution

- We can group together all positions that should have equal characters.
- There remain only $O(n)$ constraints between these clusters (each saying that two consecutive characters cannot be modified at the same time).
- Try each of four possible assigments to elements in a cluster and recursively deal with implications to other clusters in a DFS fashion.
- If no contradiction is found, set this assignment and go on to next cluster (no backtracking).
- Equivalent to running naive SCC algo on the "reduced" graph with $E = O(V)$; runs in $O(V\ E) = O(n^2)$ AC.

Jumping Monkey

Solution

Categories: Graphs, DP / BFS

Idea

For each possible shooting place, keep track of the possible places where the monkey can be.

Jumping Monkey

Solution

Categories: Graphs, DP / BFS

Idea

For each possible shooting place, keep track of the possible places where the monkey can be.

DP / BFS

State = subset of places where the monkey can be

Jumping Monkey

Solution

Categories: Graphs, DP / BFS

Idea

For each possible shooting place, keep track of the possible places where the monkey can be.

DP / BFS

State = subset of places where the monkey can be
For each possible place where the monkey can be, shoot at it, and recompute the list of possible places where the monkey can move. For each state, keep track of the place where the shot was done. Cost: $O(n^3 \cdot 2^n)$ (2^n possible states, n possible shots, n^2 possible neighbours).

Jumping Monkey

Solution

Categories: Graphs, DP / BFS

Idea

For each possible shooting place, keep track of the possible places where the monkey can be.

DP / BFS

State = subset of places where the monkey can be
For each possible place where the monkey can be, shoot at it, and recompute the list of possible places where the monkey can move. For each state, keep track of the place where the shot was done. Cost: $O(n^3 \cdot 2^n)$ (2^n possible states, n possible shots, n^2 possible neighbours).
Algorithm terminates if either no new states are found or state 0 is reached. In the latter case, recompute the path using the list of shots.

Possible optimizations

- Use bitmasks for the set of neighbours. Computation of the next state is done in $O(n^2)$. Total complexity is $O(n^2 \cdot 2^n)$.

Possible optimizations

- Use bitmasks for the set of neighbours. Computation of the next state is done in $O(n^2)$. Total complexity is $O(n^2 \cdot 2^n)$.
- Let the state be $\{V_{i_1}, V_{i_2}, ..., V_{i_n}\}$. Precompute the OR of the neighbours of $\{V_{i_1}, ..., V_{i_k}\}$ and $\{V_{i_k}, ..., V_{i_n}\}$. If you shoot to the vertex V_{i_k}, the next state is the OR of the neighbours $\{V_{i_1}, ..., V_{i_{k-1}}\}$ and $\{V_{i_{k+1}}, ..., V_{i_n}\}$ and can be done in constant time. Total cost is $O(n \cdot 2^n)$.

Sensor Network

Solution

Problem

Find minimum $\lambda(A)$ between all spanning trees of the graph (or spanning subgraphs).

Inefficient approach

Fix an edge x and take edges with increasing weight until it spans the whole graph. Total running time worst case $\Omega(m^2) \Rightarrow$ TLE .

Sensor Network

Solution

Good Algorithm

1. Sort the edges by increasing weigh.
2. F = set of edges kept (forming a forest). Set $F = \emptyset$.
3. For each edge x, add x to F. If $\exists$ cycle in F, remove lightest edge in the cycle. If $|F| = n - 1$, F is a spanning tree; check $sol < \lambda(F)$.

Total running time: $O(n \cdot m)$ AC .
There is also a (very hard) $O(m \log n)$ solution using link-cut trees of Sleator and Tarjan (union-find with deletions).

References

- Camerini, Maffioli, Martello, Toth: “Most and least uniform spanning trees”, Discrete Appl. Math. (1986).
- Sleator, Tarjan: “A data structure for dynamic trees”. In Proceedings of STOC (1981).

Sensor Network

Solution

Categories: Graphs

Data into a graph

Each sensor controls exactly two doors...

Sensor Network

Solution

Categories: Graphs

Data into a graph

Each sensor controls exactly two doors...

- Doors $\Rightarrow$ Vertices ($n \leq 300$).
- Sensors $\Rightarrow$ Edges ($m \leq n(n-1)/2$).
- Voltage $\Rightarrow$ Weight (ω).

Sensor Network

Solution

Categories: Graphs

Data into a graph

Each sensor controls exactly two doors...

- Doors $\Rightarrow$ Vertices ($n \leq 300$).
- Sensors $\Rightarrow$ Edges ($m \leq n(n-1)/2$).
- Voltage $\Rightarrow$ Weight (ω).

- “Neighboring” sensors $\Rightarrow$ Adjacent edges.
- Active sensors (admissible subset) $\Rightarrow$ $A \subset E$ connected, spans the graph.
- *margin* $\Rightarrow$ $\lambda(A) = \max_{x,y \in A}\{|\omega_x - \omega_y|\}$.
- Minimum *margin* when A is a tree.

Sensor Network

Solution

Problem

Find minimum $\lambda(A)$ between all spanning trees of the graph (or spanning subgraphs).

Inefficient approach

Fix an edge x and take edges with increasing weight until it spans the whole graph. Total running time worst case $\Omega(m^2) \Rightarrow$ TLE .

Sensor Network

Solution

Good Algorithm

1. Sort the edges by increasing weigh.
2. F = set of edges kept (forming a forest). Set $F = \emptyset$.
3. For each edge x, add x to F. If $\exists$ cycle in F, remove lightest edge in the cycle. If $|F| = n - 1$, F is a spanning tree; check $sol < \lambda(F)$.

Total running time: $O(n \cdot m)$ AC .
There is also a (very hard) $O(m \log n)$ solution using link-cut trees of Sleator and Tarjan (union-find with deletions).

References

- Camerini, Maffioli, Martello, Toth: “Most and least uniform spanning trees”, Discrete Appl. Math. (1986).
- Sleator, Tarjan: “A data structure for dynamic trees”. In Proceedings of STOC (1981).

H - Assembly line

Categories: Dynamic Programming

Idea

cost(i,j,k) is minimum cost to assemble pieces from i to j obtaining a component of type k.

$$
\begin{aligned}
cost(i,i,k) &= 0 \text{ if the component at position } i \text{ is of type } k. \\
cost(i,i,k) &= \infty \text{ if the component at position } i \text{ is not of type } k. \\
cost(i,j,k) &= MIN\{cost(i,m,a) + cost(m+1,j,b) + C_{a,b} \\
&\qquad \mid m \in [i,j), a,b \in \text{types of pieces}, R_{a,b} = k\}
\end{aligned}
$$

where $C_{a,b}$ is the cost of assembling two pieces of types a and b, and $R_{a,b}$ is the type of the resulting component.

Solution

Fill a matrix of $length \times length \times symbols$ by diagonals.

Cost $O(length^3.symbols^2)$

Locks and keys

Solution

Categories: graphs

Problem

Output a path between two nodes on a tree with restrictions on the edges that can be traversed.

Locks and keys

Solution

Categories: graphs

Problem

Output a path between two nodes on a tree with restrictions on the edges that can be traversed.

First step

Compute whether a path is even possible by keeping a set of nodes that can be visited and a set of available keys. Once a lock is encountered, check for the key and augment the path if the key is found. Runtime: $O(V + C)$.

Locks and keys

Solution

Categories: graphs

Problem

Output a path between two nodes on a tree with restrictions on the edges that can be traversed.

First step

Compute whether a path is even possible by keeping a set of nodes that can be visited and a set of available keys. Once a lock is encountered, check for the key and augment the path if the key is found. Runtime: $O(V + C)$.
The length of the maximum available path allows to keep track of a path which goes to and from the root and opens the locks as they are encountered.

Speedups

Topological Sort of the colors of the keys(which keys do I need to take before others?) to avoid taking unnecessary keys.

Speedups

Topological Sort of the colors of the keys(which keys do I need to take before others?) to avoid taking unnecessary keys.
LCA to compute shortest path on the tree.

3-sided dice

Solution

Categories: Geometry

View the problem geometrically

- Key idea: See a die as a point in three dimensions (for each outcome, a coordinate with the probability of that outcome).

3-sided dice

Solution

Categories: Geometry

View the problem geometrically

- Key idea: See a die as a point in three dimensions (for each outcome, a coordinate with the probability of that outcome).
- The set of dice for which we answer "YES", is the set of convex combinations of the given dice with non-zero coefficients for all points.
- The set of valid points is the interior of the triangle determined by the three points.

3-sided dice

Solution

Transform to two dimensions

- We want to see whether a point is in a triangle. We know how to do that in 2D.

3-sided dice

Solution

Transform to two dimensions

- We want to see whether a point is in a triangle. We know how to do that in 2D.
- Key idea: All the dice in the input, seen as a point in 3 dimensions, are in the plane $x + y + z = 10,000$.
- We can just project all the input into the $z = 0$ plane (i.e. drop third coordinate). Then check if the fourth dice is in the interior of the triangle in the plane determined by the other ones.

Handle special cases

- The triangle could be degenerate (i.e. the three points are in the same segment)
- In that case, we need to check if the fourth dice is in the interior of the segment.

3-sided dice

Solution

Final algorithm

- Check if the three points are in the same segment.
- If they are, check if the point is in the interior of the segment.
- Otherwise, check it is in the interior of the triangle.

3-sided dice

Solution

Alternative solution

- See the problem as a linear system of three equations with three variables.
- Handle the case in which the determinant is 0, so the matrix has rank 1 or 2.