香港電腦奧林匹克競賽
Hong Kong Olympiad in Informatics

2026 Mini Competition (Teams) Editorial

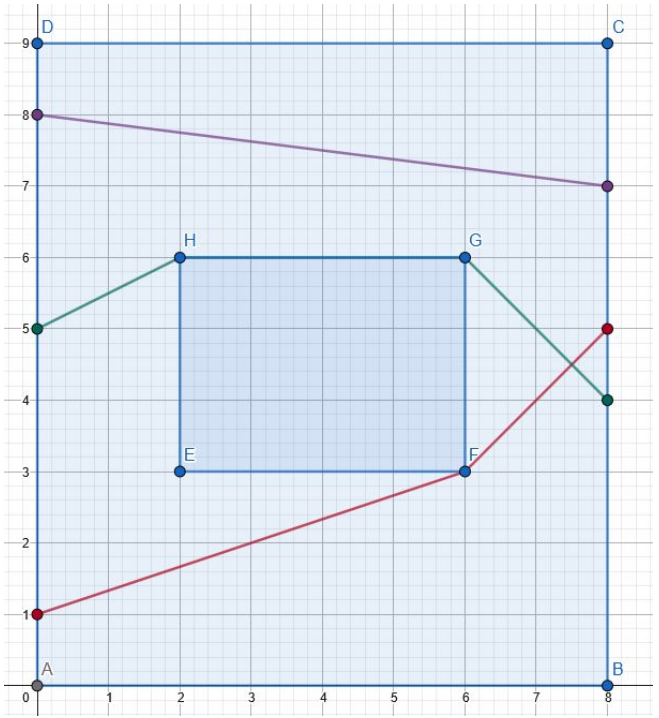
2026-04-11

M26CA
A Walk in the Park

The Problem

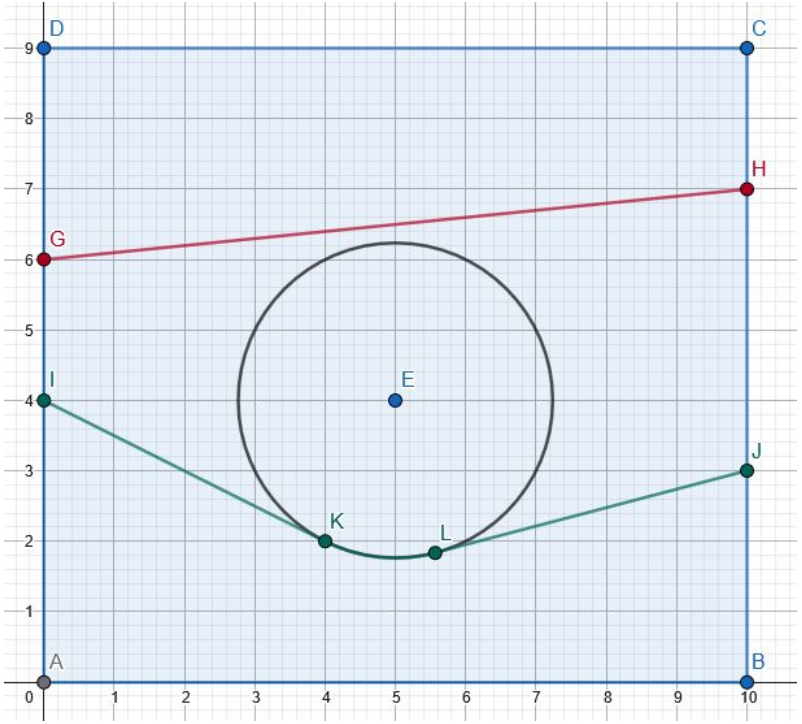
You are given a convex polygon strictly inside a rectangle and Q queries from the left hand side of the rectangle to the right hand side of the rectangle.

Find the minimum distance without going in the polygon for each query.



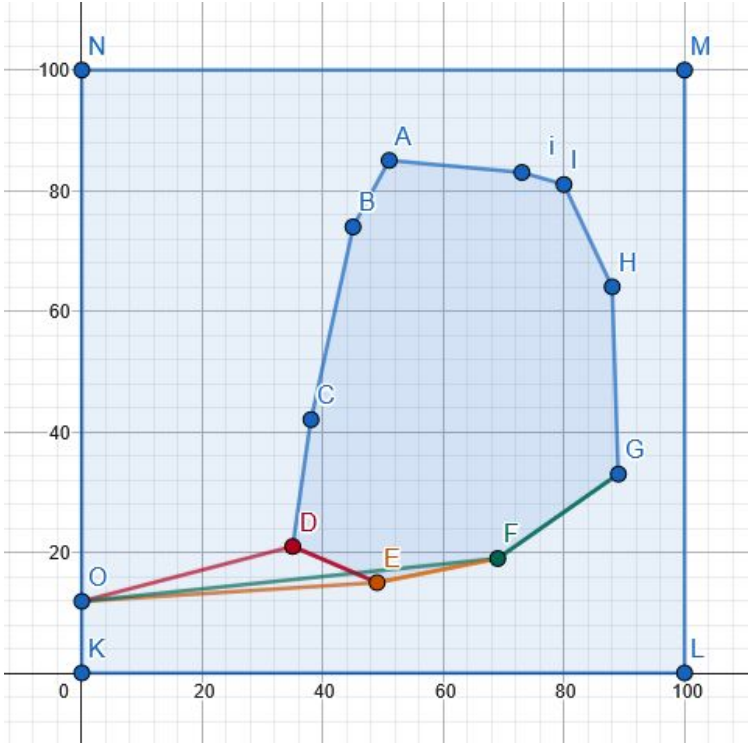
Solution

- Consider the case where we have a circle instead of a convex polygon
- The answer is either a straight line or two tangents and an arc
- Don’t worry about how to find it using code (you can still think about it), only focus on the idea



Solution

- For the convex polygon case, the idea is similar
- Binary search on the “tangent point” such that the final polyline is convex
- The “tangent point” should be the first point i on the hull such that (0, A) to (X_i, Y_i) to (X_{i+1}, Y_{i+1}) forms a left turn
- Perform the binary search four times for (0, A) and (N, B), and also on lower hull and upper hull
- Answer is the straight line if “tangent point” of (0, A) is greater than “tangent point” of (N, B), or if their “tangent point” are equal and (0, A) to “tangent point” to (N, B) forms a right turn



Example of finding the tangent of (0, A) to the lower hull (DEFG), the “tangent point” is shown in orange.

Implementation notes

- You are strongly recommended to use a custom struct to store a point
- Use cross product / dot product to check direction of turns, avoid using atan2 / slope (very annoying edge cases when crossing quadrants, double is not accurate)
- Reuse code for multiple binary searches

```
template<class T>
struct Point {
    typedef Point P;
    T x, y;
    explicit Point(T x=0, T y=0) : x(x), y(y) {}
    bool operator<(P p) const { return tie(x,y) < tie(p.x,p.y); }
    bool operator==(P p) const { return tie(x,y)==tie(p.x,p.y); }
    P operator+(P p) const { return P(x+p.x, y+p.y); }
    P operator-(P p) const { return P(x-p.x, y-p.y); }
    P operator*(T d) const { return P(x*d, y*d); }
    P operator/(T d) const { return P(x/d, y/d); }
    T dot(P p) const { return x*p.x + y*p.y; }
    T cross(P p) const { return x*p.y - y*p.x; }
    T cross(P a, P b) const { return (a-*this).cross(b-*this); }
```

Template from kactl

M26CB
Brainrot

The Problem

You are given a (large) integer N.

Rearrange its digits so that it does not contain “67” as a substring and does not contain leading zeroes.

Find the smallest such rearrangement.

Solution

- Denote f[i] to be the frequency of the digit i.
- Case 1: f[6] = 0 or f[7] = 0.
  - Simply sort the digits.
- Case 2: f[6] > 0, f[7] > 0, f[8] + f[9] > 0.
  - Sort the digits 0 to 6 first.
  - Then put one 8 or 9 (take 8 if both digits exist).
  - Then sort the remaining digits.
- Case 3: f[6] > 0, f[7] > 0, f[8] = f[9] = 0.
  - Sort the digits 0 to 5 first.
  - Then put all the digit ‘7’s, then finally all the digit ‘6’s.
- Remember to handle leading zeroes.

M26CC
Closest Pair

The Problem

Given a tree of N nodes with weighted edges.
The tree structure is known, but the edge distance is unknown.

Query(S): returns any closest pair (u, v) among elements of set S.
Find a diameter within (N - 3) queries.

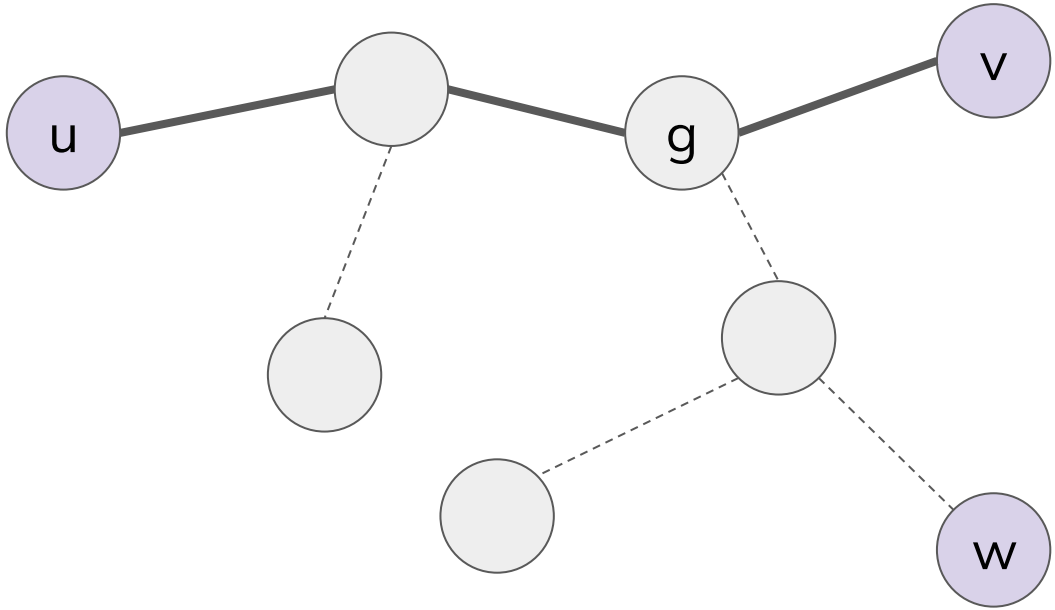
Solution

- **Fact 1:** If (u, v) is a diameter of T, then if we add a node w to T to form T', a diameter of T' is between two nodes in {u, v, w}.

- **Fact 2:** For every three nodes u, v, w, there exists a "center" x where the paths u <-> x, v <-> x and w <-> x are edge-disjoint.
- **Corollary:** Query({u, v, w, x}) (indirectly) gives us the farthest pair among (u, v), (u, w) and (v, w).

- Simply add nodes one-by-one and maintain the diameter.
- Since we only have to consider leaves, # of queries = (# leaves) - 2 ≤ N - 3.

Solution



M26CD
Dual Duel

The Problem

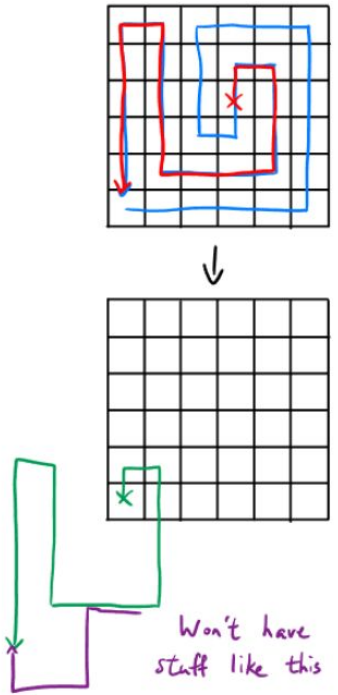
There is an N x N grid ($N \leq 50$). You start at the bottom-left cell (0, 0).

There are two “teleporters” inside the grid that teleports you to your starting location (bottom left), but you don’t know where they are.

Construct a sequence of moves up to 5000 ($\sim 2N^2$) such that you visit both teleporters at least once.

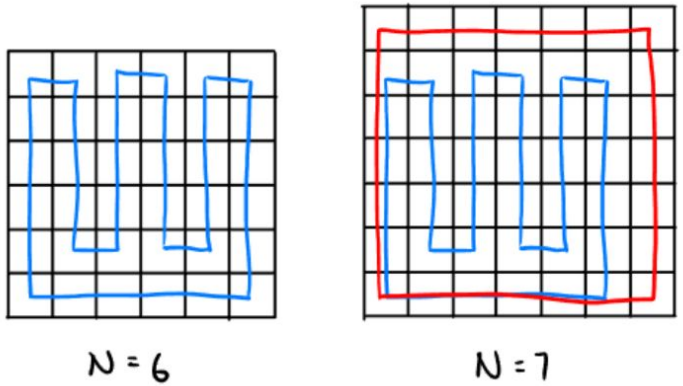
Solution

- Main idea: find a cycle that passes through every cell once, then visit the cells again in reverse order (it should also form a cycle).
  - For the two teleporters X and Y, if you first land on X in the first cycle, you will first land on Y in the second cycle. Then you don't have to be afraid of one teleporter "blocking the other".
  - Also, after each cycle, **we will always return back to (0, 0)**. You can prove so by considering the "net movement" and the "maximum movement" of the remaining moves.



Solution

- The task now is to construct a cycle that passes through each cell once, i.e. a Hamiltonian cycle. This is impossible for odd N, but we can solve that issue by constructing multiple Hamiltonian cycles.
- There are many ways to construct such cycles. My solution is as follows:



M26CE
Erase The Maximums

The Problem

Given a N × N board, all $A_{i,j}$ form a permutations from 1 to N^2.

There are 2N-1 turns:

- In odd turns, find the maximum value of remaining numbers of each remaining row, erase the row with the largest row-maximum.
- In even turns, find the maximum value of remaining numbers of each remaining column, erase the column with the largest column-maximum.

Report the actions of every turn.

Solution

- **Fact 1:** The largest row-maximum is the largest number. (same for column-maximum)

- So we save the positions of all the numbers.
- Loop through all the numbers from largest to smallest.
- Use two arrays to track whether a row/column was erased.
- Time Complexity: $O(N^2)$.

M26CF
Fingertip Mahjong

The Problem

Alice and Bob play the game Fingertip Mahjong. There are 18 columns in total:

- Alice's columns A[1], A[2], ..., A[9] where column i wants tile "i-萬" (denoted W_i);
- Bob's columns B[1], B[2], ..., B[9], where column i wants tile "i-筒" (denoted T_i).
- Each column contains 4 tiles, ordered from top to bottom.
- Read the statement for the detailed rules.

Simulate the game, and output (1) the number of rounds played, (2) sequence of tile removals during the whole process.

Solution

- Direct simulation. (Trust me, it's not that bad.)
- The only tricky part is dealing with Chinese input/output.
- The best strategy is to map each Chinese token in the input into an integer ID [0...17], and map back the ID to the corresponding Chinese token to output.
- Python users: `print("九筒"[0])` would output 九, so this time you have an advantage over C++ (though, it does not help much).

Note on Code Style

```
5  vector<string> tiles = {
6    "一萬", "二萬", "三萬",
7    "四萬", "五萬", "六萬",
8    "七萬", "八萬", "九萬",
9    "一筒", "二筒", "三筒",
10   "四筒", "五筒", "六筒",
11   "七筒", "八筒", "九筒"
12 };
```

Using vector / array gives you the tile → ID mapping for free!

Avoid writing code like this. It's not that neat...

```
25 string conv(int x) {
26     if (x == 1) return "一萬";
27     if (x == 2) return "二萬";
28     if (x == 3) return "三萬";
29     if (x == 4) return "四萬";
30     if (x == 5) return "五萬";
31     if (x == 6) return "六萬";
32     if (x == 7) return "七萬";
33     if (x == 8) return "八萬";
34     if (x == 9) return "九萬";
35     if (x == 10) return "一筒";
36     if (x == 11) return "二筒";
37     if (x == 12) return "三筒";
38     if (x == 13) return "四筒";
39     if (x == 14) return "五筒";
40     if (x == 15) return "六筒";
41     if (x == 16) return "七筒";
42     if (x == 17) return "八筒";
43     return "九筒";
44 }
```

M26CG
Game of XOR

The Problem

There is an unknown array A of length N.

Given Q range XOR result of the form (L[i], R[i], X[i]) where (^ is XOR)

$$A[L[i]] \wedge A[L[i] + 1] \wedge \ldots \wedge A[R[i]] = X[i]$$

Recover any possible A or report impossible.

Solution

- Consider the “partial XOR” array where B[R] = A[1] ^ A[2] ^ … A[R].
- Then the given information becomes

$$B[L[i] - 1] \wedge B[R[i]] = X[i]$$

- Which means we can model this as an edge between node (L[i] - 1) and node R[i] with edge weight X[i].

- Simply run DFS on this graph to compute a possible B array (no solution if have contradiction), then recover A from B.

M26CH
Harmonious String

The Problem

Call a binary string *harmonious* if it satisfies the following condition:

- It can be made empty by matching and cancelling out adjacent pairs.
  e.g. 100100 -> 1100 -> 00 ->

You have a binary string S. You can perform the following operation:

- Pick some (L, R) such that $1 \le L \le R \le N$, and flip S[L..R].
  e.g. Flip (1, 4) from 100100 -> 011000

Find the lexicographically smallest pair of (L, R) that makes the string *harmonious*, or (0, 0) if no operations are needed.

Solution

Step 1. Reduce the condition for *harmonious*:

- The problem is equivalent to removing substrings “00” or “11”.
- Let’s consider flipping all **even** positions in the original substring.
  e.g. 100100 -> 110001
- After the transformation, the problem becomes removing “01” or “10”.
- Then, the string is *harmonious* iff # of 0s = # of 1s. Proof:
  - Note that the quantity (# of 0s - # of 1s) is **invariant** (doesn’t change) after any amount of operations. To reach the empty string, this quantity must be 0. The condition is necessary.
  - It can be shown that if there exist 0s and 1s in the string, there must exist a substring “01” or “10”. Just consider when the string “changes”. The condition is sufficient.

Solution

Step 1. Reduce the condition for *harmonious*:

- Therefore, a string is *harmonious* if and only if after flipping all **even** positions in the string, # of 1s = # of 0s.

A few notes:

- You can also choose to flip all odd positions, the condition still holds.
- An equivalent expression would be the following:
  - A string is *harmonious* iff # of 1s in odd positions = # of 1s in even positions.
- This “flipping all even positions” trick is quite common. The motivation behind this is to find invariants in operations (# of 0s - # of 1s).

Solution

Step 2. Now, make observations for flipping:

Observation: If an operation is needed, it is **always possible** to have L = 1.

- Consider the string after flipping even positions. We consider the quantity # of 0s - # of 1s. If it is 0, the string is already harmonious.
- Let D_i be the value of this quantity after flipping the range [1, i].
  - For convenience, we let D_0 as that value without flipping.
- Then, we have the following:
  - Each time you go from D_i to D_{i+1}, you flip 0 -> 1 or 1 -> 0. Therefore, the quantity either increases or decreases by 2, i.e. $|D_i - D_{i+1}| = 2$.
  - Consider D_0 and D_N. All 0s become 1s and vice versa. Therefore they sum up to 0, i.e. $D_0 + D_N = 0$.

Solution

Step 2. Now, make observations for flipping:

Observation: If an operation is needed, it is **always possible** to have L = 1.

- Let D_i be the value of (# of 0s - # of 1s) after flipping the range [1, i].
- Then, we have $|D_i - D_{i+1}| = 2$ and $D_0 + D_N = 0$.
- From this, by "intermediate value theorem", there exists some $D_i = 0$.
  - A visual representation is shown next slide to hopefully give more intuition.

So the solution is to simply sweep through the string once while maintain the value D_i. The implementation is very simple!

There are also other solutions that don't require this observation (sweeping through all possible values of L), but requires maintaining more data.

Solution

D_i = # of 0s − # of 1s

S = 110001 → 100100

1. $|D_i - D_{i+1}| = 2$

e.g. $i = 2$

010100 ($D_2 = 2$)

↓

011100 ($D_3 = 0$)

# of 0s −= 1

# of 1s += 1

↓

$|D_2 - D_3| = 2$

2. $D_0 + D_6 = 0$

D_0: 100100 (2)

D_6: 011011 (−2)

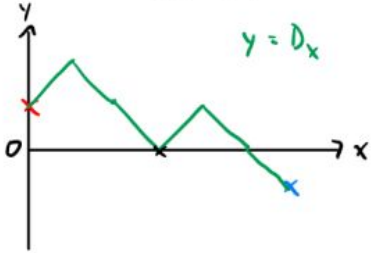
# of 0s in $D_0 + D_6 = 6$

# of 1s in $D_0 + D_6 = 6$

↓

$D_0 + D_6 = 0$

3. Exist $D_i = 0$



$D_0 = 2$, $D_6 = -2$

Because the line is continuous,
and we know it goes from positive to negative,
it will definitely pass through 0.

M26CI
I Like Spamming Copypasta

The Problem

Given a tree with N nodes.

Find the number of permutations P[1..N] such that for every index i except the first, the node P[i] is directly connected to some node that occurs before it in the permutation.

Solution

- Suppose we fix the value of P[1]. Root the tree at it.
- Let dp[i] be the number of permutations containing the nodes in the subtree of node i, such that
  - For all u and v, if u is an ancestor of v, then u appears before v in the permutation.
- Obviously, dp[i] = product {u is children of i} of dp[u], then multiplied by the multinomial coefficient of subtree sizes of u (where u is children of i).
- So we can easily compute the answer for a fixed P[1] is just dp[P[1]].
- Simply compute the answer for each possible value of P[1]. We can do this trivially with rerooting tree DP.
- Time Complexity: O(N).

M26CJ
Jury's Ambitious Proposal

The Problem

Given N problems and M contests, assign each problem to at most 1 contest.

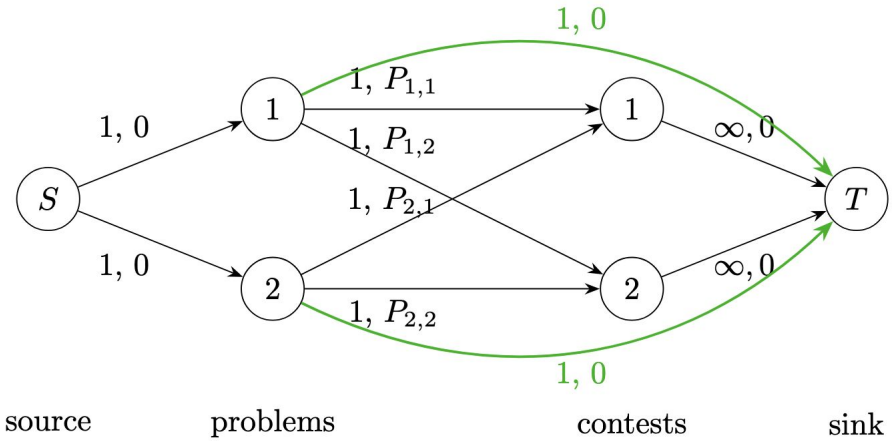
For each contest j, the number of problems A[j] must satisfy
F[j] – R[j] ≤ A[j] ≤ F[j] + R[j], where there is **C * |A[j] – F[j]|** dissatisfaction.

Dissatisfaction of assigning problem X to contest Y is P[X, Y].

Give an assignment satisfying requirements + minimize total dissatisfaction.

Solution

- Without the restrictions F[j], R[j], it is a simple greedy problem (pick the contest with minimum dissatisfaction, or don't pick at all).
- However, you can also solve it using min-cost max-flow.
- The answer is the minimum cost to achieve N units of flow.

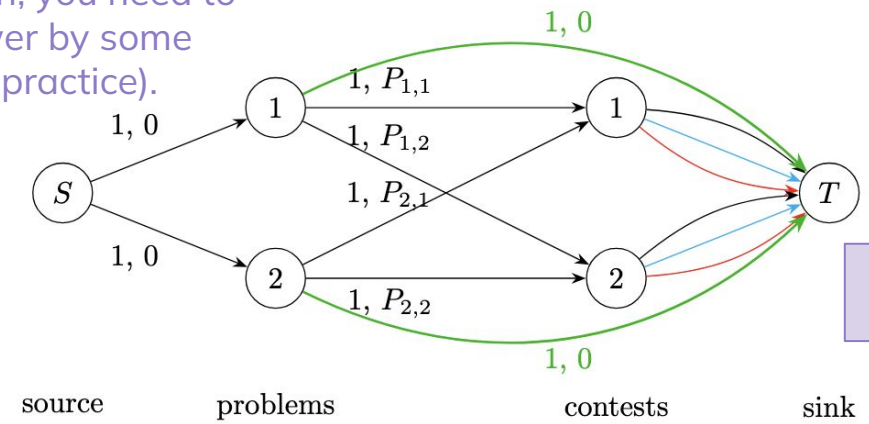


Solution

- With the restrictions, we need to change our modelling slightly.
- C' is a very large constant (say 10^{12}, see table below).
- Check that the minimum cost is *slightly larger* than –C' * sum(F[j] – R[j]).

Note: to obtain the actual min dissatisfaction, you need to add the answer by some offset (left as practice).

Color	flow	unit cost
●	$F_j - R_j$	$-C'$
●	R_j	$-C$
●	R_j	C



Time complexity: $O(N^2M \log (N+M) + NM^2)$

(comes from min-cost max-flow alone)

M26CK
Kounterstrategy

The Problem

Alice and Bob play a game on the string, S = “” initially.

Alice moves first. On each turn, perform **one** of the following operations:

S = S + “a”	S = S + “b”	S = reverse(S)	S = S + S

After K turns, let f_ab(S) be the number of “ab” substrings in S, and f_ba(S) be the number of “ba” substrings in S.

- f_ab(S) > f_ba(S): Alice wins;
- f_ab(S) < f_ba(S): Bob wins;
- f_ab(S) = f_ba(S): Draw.

Solution

- Consider the quantity D = f_ab(S) – f_ba(S).
- **Observation 1:** D is one of –1, 0 or 1, and its value depends only on the start and end characters of S.
  - If S = a…a or S = b…b then D = 0 (Draw).
  - If S = a…b then D = 1 (Alice wins).
  - If S = b…a then D = –1 (Bob wins).
- **Observation 2:** Alice can make S = a…a after odd-numbered rounds.
  - Proof: by mathematical induction on number of Alice operations.
  - If Alice made S = a…a on round 2N – 1, after round 2N (Bob) either S = a…a or S = a…b.
  - Alice has a strategy to make S = b…a never happen regardless of Bob's moves.
  - Consequently, Bob can **never win**.

Solution

- Consider the quantity D = f_ab(S) – f_ba(S). We know –1 ≤ D ≤ 1.
- We know Bob can **never win**. When does Alice win then?

- K even: The game ends in a draw.
  - Bob makes the last move.
  - If the value of D before his move is 1, he will reverse the string to negate D (it is now –1).
  - So Alice won't do that. She will make S = a…a on round K – 1, so Bob can append "a" to force a draw.
- K odd and K > 1: Alice wins.
  - Strategy: make S = a…a until round K – 2. After round K – 1, S must start with "a".
  - So just append "b" to the end and make S = a…b. This way, D = 1 and Alice wins.
- K = 1: The game ends in a draw.