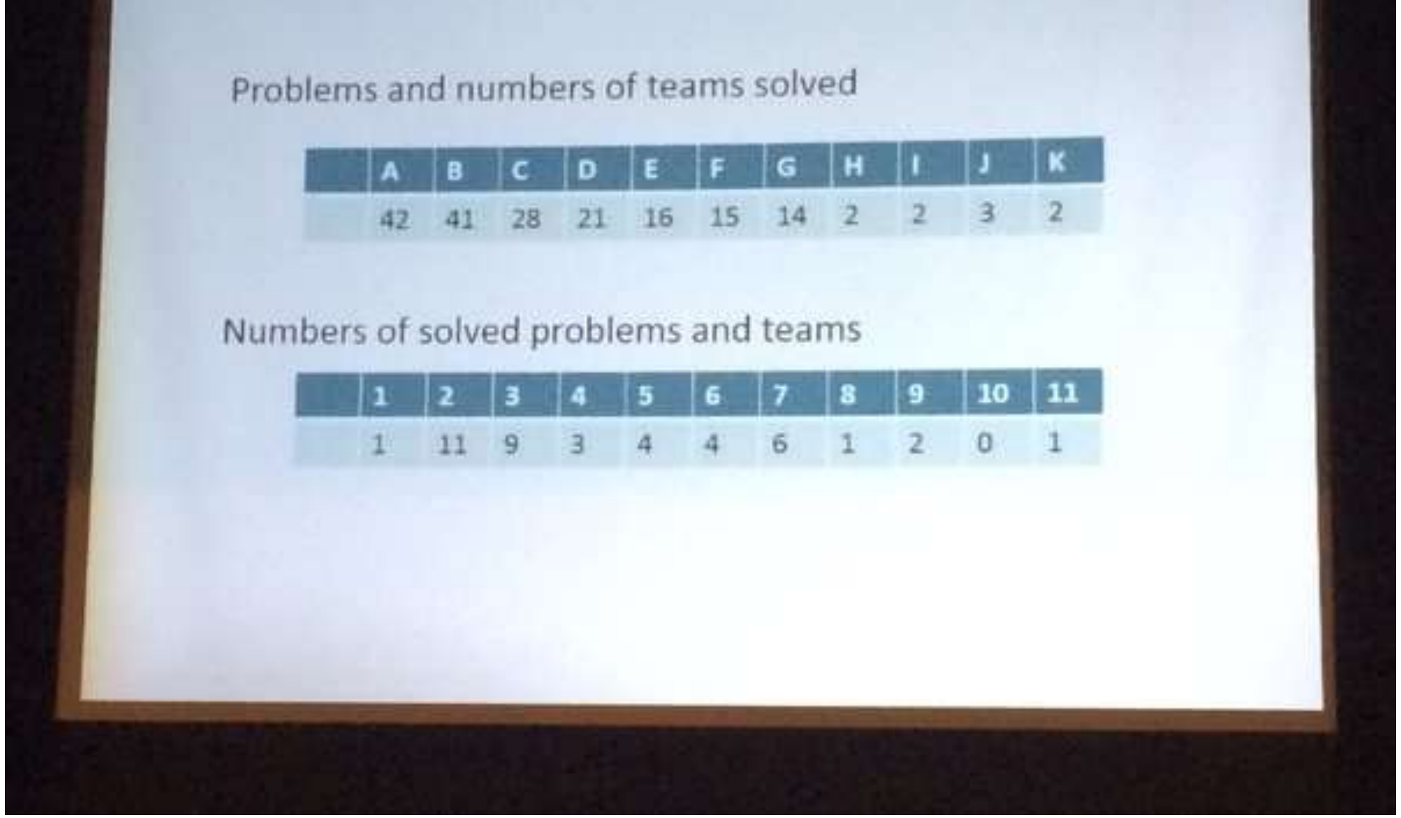
ICPC 2015, Tsukuba *Unofficial* Commentary

Mitsuru Kusumoto (ir5)

Summary



Problems and numbers of teams solved

	A	B	C	D	E	F	G	H	I	J	K
	42	41	28	21	16	15	14	2	2	3	2

Numbers of solved problems and teams

	1	2	3	4	5	6	7	8	9	10	11
	1	11	9	3	4	4	6	1	2	0	1

https://twitter.com/icpc2015tsukuba/status/670846511375212544

- Every team solved at least one problem. *happy*
- SJTU is so cool.
- Although the solution of problem I is simple, it turned out that the problem was difficult.

A: Decimal Sequences

- Straightforward. Just search a substring from 0,1,2,....
- Solution has at most ceil($\log_{10}$(n+1)) digits, so time complexity is O(n^2logn).

- Homework(not so hard): Solve this problem in time O(nlogn).

B: Squeeze the Cylinders

- Move the cylinders from the leftmost one.
- The position of the i-th cylinder is determined using Pythagorean theorem.
- Time complexity is $O(N^2)$.

C: Sibling Rivalry

- Using matrix multiplication, we can compute a set of vertices reachable from each vertex after a (resp., b, and c) steps.
  - Let's denote the set of reachable vertices from a vertex v by R(v, a).
- For a vertex v, let f(v) := "the number of minimum required turns to reach the goal." If it is impossible to go to the goal from v, f(v) = ∞.
- Obviously, f(goal) = 0.
- Also, as you want to minimize # of turns while the bro wants to maximize it, f(v) = max_{t=a,b,c} min_{w∈R(v, a)} f(w) holds.
- Initialize f(v)=∞ and f(goal)=0, and update the value of f(·) by iteration until converges. Time complexity is $O(n^4)$. This can be reduced to $O(n^3)$ though.

D: Wall Clocks

- At first, compute the range of visible wall for each person. This is a cyclic interval.
- There are n cyclic intervals, so there are at most 2n candidate positions to put clocks.
- Determine one candidate position to put a clock, and remove intervals that contains the clock. After this, the cyclic intervals can be regarded as standard intervals on a line.
- **Greedy works:** sort the intervals by the leftmost position, and put a clock at the rightmost position of the leftmost interval among the remaining intervals.
- Time complexity is $O(n^2)$.

E: Bringing Order to Disorder

- Enumerate all the ascending sequences with n digits (e.g., 0011239). There are at most combin(14+10-1, 10-1)≒8×10^5 such sequences.
- Compute sum(·) and prod(·) for each ascending sequence, and compare their sum/prod with the given sequence.
  - If sum/prod of some ascending sequence s' is strictly less than that of the given sequence, all the permutation of s' is a solution. The number of them is computed by $n!/(m_0! \cdot m_1! \cdot \ldots \cdot m_9!)$, where m_i is the number of digits i in s'.
- If sum/prod of s' is the same with the given sequence, some of permutation of s' are solution and some of them are not.
  - The number of such permutation s' is at most 38. (This is hard to estimate, but probably you can believe that such number is quite small.)
  - If the given is 8274612, the solution should be like 827y***, where $0 \leq y < 4$ and * is arbitrary digit. We can count up such sequences in time $n \cdot 10^2$.

E: Bringing Order to Disorder

NOTE:

- There are other solutions like digit DP or meet-in-the-middle (so called “半分全列挙” in Japan.)
- But watch out for the time limit. Meet-in-the-middle solution takes $10^7 \cdot \log_2 10^7$ time, which is probably dangerous.

F: Deadlock Detection

- Problem setting may seem a little complicated?

- Binary-search the deadlock-unavoidable time.
- To check if the current state is deadlock-unavoidable or not, try greedy strategy:
  - If one process can acquire all the required resources, give away the required resources to the process. Iterate this until either all the processes terminate or fall into dead-lock.
- Time complexity is some kind of $O(\log n \cdot \mathrm{poly}(p,r,t))$.

G: Do Geese See God?

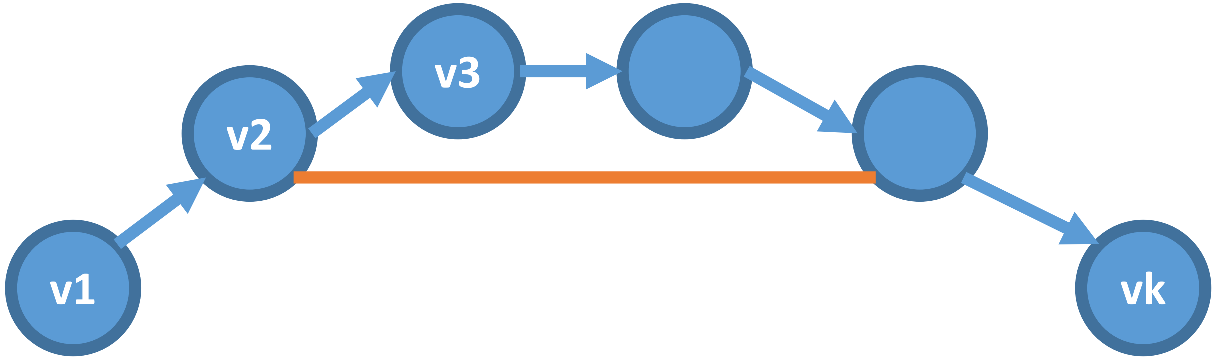
- k-th? The shortest? They are complicated, solve from simpler problem.
- First, consider how to compute the shortest length of the palindrome.
  - Let f[i][j] := the shortest length of palindrome that is a supersequence of S[i..j]. Then f[i][j] can be computed by DP like edit-distance in O(n2) time.
- If k=1, the solution is computed by backtracking f[·][·].
  - If S[i]=S[j], backtrack (i,j)->(i+1,j-1) works.
  - Otherwise, lexicographically smaller one between (i,j)->(i,j-1) and (i,j)->(i+1,j) works.
- If k>1, count up the number of different palindromes for each substrings S[i..j]. Then the similar strategy works.
- Time complexity is $O(n^2)$.

H: Rotating Cutter Bits

- When we fix the workpiece, we would see that the cutter bit moves along a circle with radius L and center (0, 0) without any rotation.
- Thus, the region that the cutter bit passes is a minkowski-sum of (the boundary of the cutter bit) and (The boundary of a circle with radius L and center (0,0)).
- The number of lattice points is up to 4×10^8, which is too large to check one by one.
  - But their x,y-coordinates are small, we can count up the number of remaining points on each slices.
- Time complexity is $O(\text{CoordMax} \times (n+m)^2)$.

I: Routing a Marathon Race

- There are only 40 vertices.
- Just performing **a dfs search** suffices with the following pruning:



If we have solution v1→v2→…→vk and there is an edge (v_i, v_j),
short-cut of vi→vj would yield a better solution.
This means that, in the best solution, there's no such short-cut edges.

I: Routing a Marathon Race

This pruning may look inefficient, but this is actually efficient.

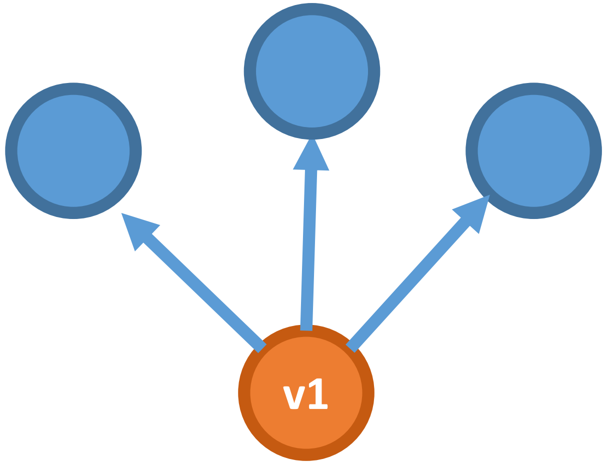
Let $f(n)$:= max number of paths with “no-short-cuts” in n-vertex graph.

If the degree of start vertex is d, there’s d choices for the first step. After that, n-d vertices are available. So,

$$f(n) \leq \max_d d \times f(n-d).$$

From this, we can prove that $f(n) \leq e^{n/e}$ holds.
(Hint: Use Jensen’s inequality.)

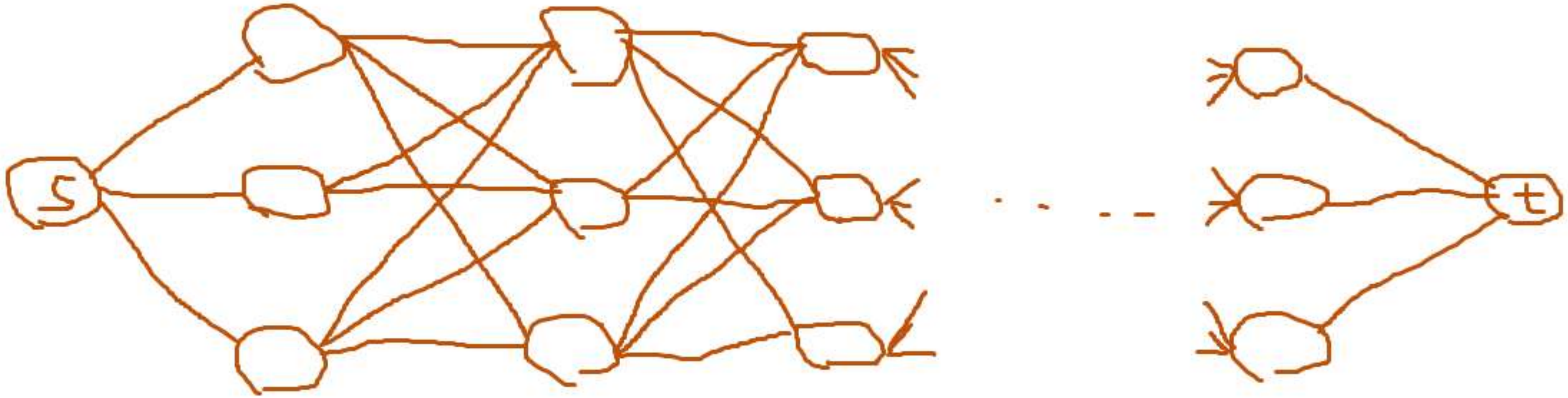
In this problem, $f(40) \leq 2500000$ holds.



I: Routing a Marathon Race

- Still, watch out for time limit. Naive $2500000 \cdot 40^2$ is dangerous.
- Use bitwise-operator to drop n factor. This works fast.

- The worst case is as follows.



J: Post Office Investigation

- A vertex v is called a *dominator* of a vertex w if every path from the starting vertex (1) to v passes w.
  - See wikipedia. https://en.wikipedia.org/wiki/Dominator_(graph_theory)
- Dominators can be represented as a *dominator tree*.

- If we have the dominator tree, we can answer the queries using LCA.

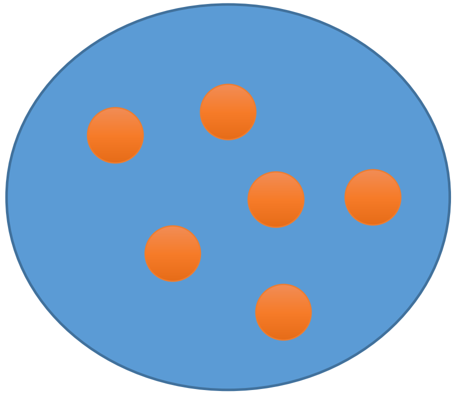
J: Post Office Investigation

- There is a linear time algorithm to compute the dominator tree.
  - Lengauer, Thomas, and Robert Endre Tarjan. "**A fast algorithm for finding dominators in a flowgraph.**" *ACM Transactions on Programming Languages and Systems (TOPLAS)* 1.1 (1979): 121-141.
  - ...But since there is a special constraint that the size of every SCC is ≤ 10, we can solve this problem without such heavy knowledge.

- First, consider the case where given graph is acyclic.
  - This case is easy: Compute the dominators in topological order (from root node).
  - Note that dom(v) = {v} ∪ ∩_{w: (w,v)∈E} dom(w) holds.

J: Post Office Investigation

- What if |SCC|≤10?
- Consider each SCC in the topological order.
- Let C={v1,v2,...,vk} be an SCC, and let D = V - C.
- From the property of SCC, (dom(v1)∩D),...,(dom(vk)∩D) are same.
- dom(vi)∩C may be different.
- For each i, perform the following: block vertex vi, and perform a BFS from vertices reachable from start vertex (1). If vertex vj becomes unreachable, we can see that vi is a dominator of vj.
- From the relation of the dominators, we can construct the dominator tree.
- Time complexity is $O(n|MaxSCC|^2+q\log n)$.



SCC

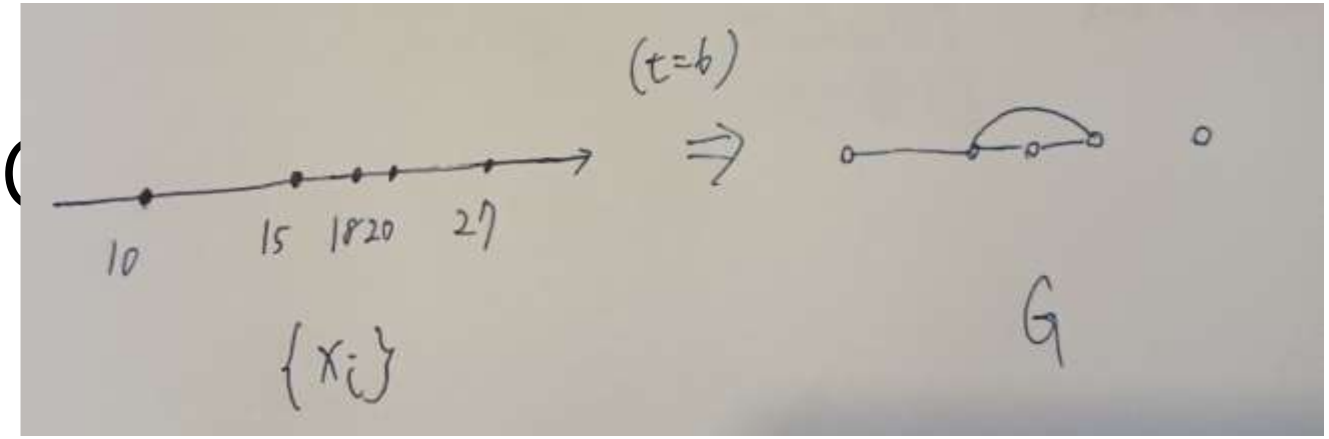
K: Min-Max Distance Game

For fixed integer t, let’s transform the game as follows:

- If the distance between result stones is ≥ t, Alice (maximizer) wins.
- Otherwise, Bob (minimizer) wins.

If we can determine who wins in this transformed game,
we can also solve the original game by binary search of t.

K: Min-Max Distance (



Let's consider a graph G like this:

- Each vertex corresponds to a stone.
- If $|x_i - x_j| < t$, there is an edge between stone i and j.

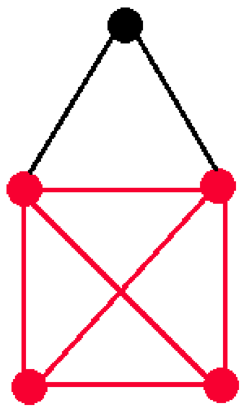
Now, we can consider the game is as follows:

- If there is no edge at the end of the game, Alice (maximizer) wins.
- Otherwise, Bob (minimizer) wins.

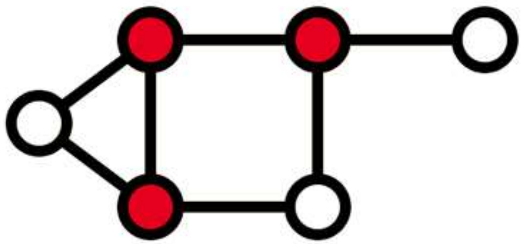
K: Min-Max Distance Game

- Alice wants to remove edges in the graph.
  - If **<u>vertex cover</u>** is small enough, Alice wins by removing vertices in the vertex cover.
- Bob wants to leave edges in the graph.
  - If **<u>clique size</u>** is large enough, Bob wins by removing vertices outside of the clique.

And surprisingly, **<u>converse also holds</u>**.



↑Clique



↑Vertex Cover

K: Min-Max Distance Game

Proof??

K: Min-Max Distance Game

Proof??

The game consists of n-2 turns.

(i) When Alice takes last turn:

Bob takes floor(n/2)-1 turns. If there is a clique with n-(floor(n/2)-1)=1+ceil(n/2) vertices, Bob always wins by taking the other vertices. Otherwise, we can prove that Alice wins by induction. The case when n=3 is trivial. Assume that this holds when there are less than n stones.

1. When n is even (so it's Bob's turn), even if Bob removes any stone, the maximum clique becomes ceil(n/2)=(ceil(n-1)/2). By induction, Alice wins.

2. Suppose when n is odd (so it's Alice's turn.) If clique is < ceil(n/2), Alice can remove any stone. If max clique = ceil(n/2), removing the center stone (the ceil(n/2)-th stone) would reduce the size of max clique. Thus Alice wins.

(ii) When Bob takes last turn:

Alice takes floor(n/2)-1 turns. In the similar manner, if the vertex cover of the graph is at most floor(n/2)-1, Alice wins by removing all the vertex covers. Otherwise, we can prove that Bob wins, again, by induction. The case n=3 is trivial. Assume that this holds when there are less than n stones.

1. When n is even (so it's Alice's turn), even if Alice removes any stone, the minimum vertex cover becomes at least floor(n/2)-1=floor((n-1)/2). By induction, Bob wins.

2. Suppose when n is odd (so it's Bob's turn.) If min vertex cover > floor(n/2), Bob can remove any stone. Consider when min vertex cover = floor(n/2). For a connected component C, we refer to the ratio (min vertex cover)/|C| as *density*. Every connected component contains a path graph. So is |C| is even, the density of C is ≥ 0.5. Since floor(n/2) < 0.5, there should be a component with density < 0.5. In such a component C, the size of the min vertex cover does not change even if we remove one vertex of the end of the path. Bob should choose such vertex.

K: Min-Max Distance Game

- In general, max clique and min vertex cover are hard to compute.
- But because graph structure is special, we can compute them in O(n) time.
- Time complexity is O(nlog(XCoordinateMax)).