NCPC 2014
Presentation of solutions

Heads of Jury: Michał Pilipczuk and Lukáš Poláček

2014-10-04
## Problem authors

- Pål Grønås Drange (UiB)
- Markus Dregi (UiB)
- Jaap Eldering (Imperial)
- Tommy Färnqvist (LiU)
- Robin Lee (Google)
- Michał Pilipczuk (UiB and UoW)
- Lukáš Poláček (KTH)
- Fredrik Svensson (Autoliv Electronics)
- Marc Vinyals (KTH)
Problem

For a graph with edge labels in \(\{0, 1, 2\}\), minimize a set of vertices \(S\) s.t. for every edge \(e\), exactly \(\text{lab}(e)\) of its endpoints are in \(S\).

Insight

If \(\text{lab}(e) \in \{0, 2\}\) the state of both endpoints are decided, propagate information through the entire connected component.

Solution

Suppose \(\text{lab}(e) = 1\) for every edge in \(G\), we two-color.

- Find an uncolored vertex \(v\), color it with some color \(\text{red}\).
- Do DFS through the graph, color the neighborhood of the current vertex with the opposite color (\(\text{blue}\)).
- For each connected component, take the smallest color class.

**Time complexity:** \(O(n + m)\).
Problem

Given a simple graph with maximum degree $\Delta(G) \leq 4$ and an integer $k \leq 15$, compute whether the independent set number $\alpha(G) \geq k$.

That is, does there exist a set of vertices $S$ of size at least $k$ such that for every two vertices $u$ and $v$ of $S$, $uv$ is not an edge of the graph.

Insight

- If a vertex $v$ is not in a maximal independent set, at least one of its neighbors are.
- If $n \geq 5k$, then the instance is a yes-instance.
- If there is no solution containing $v$ then every solution contains at least two of $v$’s neighbors.
Solution

- Branching on putting either $v$ or one of its neighbors in the solution yields a $5^k \cdot n$ time algorithm. **Too slow.** (Unless clever heuristics)

- After picking the first vertex, there is always a vertex of degree at most 3. Branching on the lowest degree vertex yields $4^k n$ time provided that connected components are solved separately. **Can get accepted**, depending on implementation.

- Branching on picking either $v$ or pairs of neighbors yields a $3^k \cdot n$ time algorithm, due to the recurrence $T(k) = T(k - 1) + 6T(k - 2)$. **Accepted.**

- Combining the two algorithms, solving components independently, branching on lowest degree vertex and trying pairs of neighbors, gives the recurrence $T(k) = T(k - 1) + 3T(k - 2)$ yielding a $2.31^k n$ time algorithm. **Even more accepted.**
Problem

Calculate \( S_n = \sum_{k=0}^{n} C_k C_{n-k} \), where \( C_n \) is the \( n \)th Catalan number.

Suggested solutions

1. Look at the given formula for \( C_n \) and figure out that \( C_n \) satisfies the recurrence \( C_0 = 1 \) and \( C_{n+1} = \sum_{k=0}^{n} C_k \cdot C_{n-k} \), which means that the numbers \( S_n \) are just the Catalan numbers shifted one place to the right.

2. Calculate \( S_n \) for some values of \( n \) offline and notice the pattern...

Speed up solution

Calculate \( \binom{2n}{n} \) by a series of alternating multiplications and divisions to speed up the computation. (Not needed for AC.)
Two players have two dice each. The player who throws bigger sum wins. Who has higher chances of winning?

For each player, calculate the probability distribution of his/her throws (calculate the probability of each possible outcome).

Using this information, determine the probability of winning for both players.

Insight: both distributions are symmetric around the mean.

Therefore it’s enough to compare the expected values of both probability distributions – compare the sum of both lines of input.
**Problem**

Given a histogram and two moves (remove_row, remove_column): find the minimum number of moves to clear the whole histogram.

**Insight**

If we remove:

- a column, we should remove the tallest one.
- a row, we should remove the lowest one.

If we remove the tallest $X$ columns, the number of rows left to remove will be the height of the biggest column remaining: the $(X + 1)^{th}$ tallest.

**Solution**

Try each possible $X$ and choose the one that gives a minimum answer, for $O(n \log n)$ time complexity.
**Problem**

Given a graph $G$, answer a number of queries of the following form:

We place tokens at vertices $A$ and $B$. The goal is to swap the tokens, and we look for a swapping procedure that maximizes the minimum distance between the tokens during the swapping.

**Insight**

Construct a *graph of states* $H$:

- $V(H)$ — pairs of vertices of $G$, represent tokens’ positions.
- $E(H)$ — transitions of tokens.

**Goal**: a max-min safeness path between $(A, B)$ and $(B, A)$ in $G$

- **First approach**: search in $H$ for every query.
- **Time complexity**: $O(n^3 m \log n)$, too slow.
Solution

- Answer all the $n(n - 1)$ possible queries, memoize the answers.
- Sort vertices of $H$ by safeness, and construct $H$ by adding the vertices from the highest safeness to the lowest.
- Maintain a list of connected components, and merge them accordingly when introducing edges.
- Answer to query $(A, B) =$ first moment when $(A, B)$ and $(B, A)$ fall into the same connected component.
- Always merge the smaller component into the larger $\Rightarrow$ Amortized time for a merge is $O(\log n)$.
- Queries answered while iterating through the smaller component.
- **Time complexity:** $O(nm + n^2 \log n)$. 

Problem Author: Michał Pilipczuk

NCPC 2014 solutions
Problem

Given a set of dependencies, find the largest subset of labels that satisfies all dependencies.

Insight

Out-degree of every vertex is 1, hence each connected component forms a cycle plus some tributaries. If we take anything we must take the cycle, then some fraction of the remainder.

- This is 0-1 knapsack with variable item sizes.
Solution

For each component:

- Find the cycle size $C_{min}$ with forward depth-first search.
- Find the full size $C_{max}$ with backward depth-first search.

Now run a modification on the standard algorithm of knapsack. When processing a component, instead of updating just $dp[a + 1][b + C_{min}]$, include all sizes up to $C_{max}$ too.

Total of sizes to try is $n$, so time complexity remains $O(nm)$.
**Problem**
Check if two sets of angles are related by a global rotation.

**Insight**
Represent the angles as relative numbers: sort the lists and calculate the differences (modulo $360^\circ$). Now the problem boils down to checking if these sequences are equal up to a circular shift.

**Solution**
If sequence $X$ is a rotation of $Y$, then $X$ will be a substring of $YY$. Use the KMP substring search algorithm to check this in $O(n)$ time.

Alternative solutions:
- Use a rolling hash to compare $X$ to all rotations of $Y$.
- Obtain ‘minimal’ representations of $X$, $Y$ and compare these.
I – How many squares?

Problem
Given a set of lines in the plane, count the number of squares formed by these lines.

Solution
- Sort and group the lines w.r.t. their direction.
- Take a group of lines $A$ and a group of perpendicular lines $B$.
- List and sort all the distances between pairs of lines from $A$, and all the distances between pairs of lines from $B$.
- Iterate through these lists with two pointers. If distance $d$ was listed $a$ times on the list for $A$, and $b$ times on the list for $B$, then increase the result by $a \cdot b$.
- **Time complexity:** $O(n^2 \log n)$. 

Problem Author: Michał Pilipczuk

NCPC 2014 solutions
Problem

Cars are arriving to single lane road segment. Let as few as possible wait more than they can bear before getting irritated.

Solution

Dynamic programming or memoization

- \( \min_{\text{time or impossible}} = f(\text{cars west}, \text{cars east}, \text{last direction}, \text{num irritated}) \)
- \( \text{cars west} \) and \( \text{cars east} \) denote the number of cars that passed from west and east respectively.
- Find the lowest \( \text{num irritated} \) that gives a possible solution.

Time complexity: \( O(n^3) \)
  - Too slow if the time is one of the function parameters.

Space complexity: \( O(n^3) \), but \( O(n^2) \) also possible.
**Problem**

Passengers enter, leave and wait for a train. Check if the input is consistent.

**Solution**

Simulate the train journey. Keep the number of passengers \( p \).

At each station:

- Check that not more than \( p \) passengers leave.
- Update \( p \).
- Check that \( p \) is not more than the capacity.
- Check that passengers wait only if the train is full.

Check that the train is empty at the last station.

Problem Author: Michał Pilipczuk and Marc Vinyals

NCPC 2014 solutions