Solutions for Oxbridge Varsity Competition

Hugo Eberhard, Yichen Huang, Xingjian Bai

Oxford & Cambridge

April 17, 2021

Table of Contents

Treehouse

Problem Statement

- Given n points on the plane, choose three of them that form a triangle without any points inside.

Solution1: Sorting

Problem Statement

- Given n points on the plane, choose three of them that form a triangle without any points inside.

- Choose three points with the smallest x-values.
- Complexity: O(n logn)

Solution 2: Iteration

Problem Statement

- Given n points on the plane, choose three of them that form a triangle without any points inside.

- Start with any three points, iterate through the remaining points.

Solution 2: Iteration

Problem Statement

- Given n points on the plane, choose three of them that form a triangle without any points inside.

- Start with any three points, iterate through the remaining points.
- Use **cross product** to check if the new point is inside current triangle. If so, update the triangle.
- Complexity: O(n)

Performance

Problem Statement

- Given n points on the plane, choose three of them that form a triangle without any points inside.

- 19 teams got accepted!
- team **Treeniceratops** got the first blood in 5 min.

Table of Contents

Bank Robbery

Problem Statement

- Given a graph with n nodes out of which k are "police stations", answer q queries for the closest police station from a given node.

First attempt: BFS q times

Problem Statement

- Given a graph with n nodes out of which k are "police stations", answer q queries for the closest police station from a given node.

- For each query, do a BFS from the node we're querying about until we reach a police station, and output the distance to that station (taking care to choose the station of lowest index if there are multiple ones of same distance).
- Complexity: O((n+m)q), too slow

Second attempt: BFS k times

Problem Statement

- Given a graph with n nodes out of which k are "police stations", answer q queries for the closest police station from a given node.

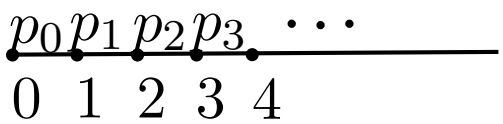
- Do a BFS from each of the k police stations, keeping track for each node of the closest police station that reached it. Can answer queries in O(1).
- Complexity: O((n+m)k), still too slow

Second attempt: BFS k times

Problem Statement

- Given a graph with n nodes out of which k are "police stations", answer q queries for the closest police station from a given node.

- Do a BFS from each of the k police stations, keeping track for each node of the closest police station that reached it. Can answer queries in O(1).
- Complexity: O((n+m)k), still too slow
- Can improve slightly by stopping the BFS if we reach a node that has already been reached faster. Not enough.



Solution: BFS from all k police stations simulteaeously

Problem Statement

- Given a graph with n nodes out of which k are "police stations", answer q queries for the closest police station from a given node.

- Do BFS from all k police stations simultaneously. Same code except we push all police stations in the beginning. Answer queries in O(1).
- Complexity: O(n+m+r), this is enough

Table of Contents

Happyyear

Problem Statement

- n integer are uniformly randomly distributed in $[1, n]$, for a subsegment of length k, the contribution is $w[k]$, output the expected product of all $n - k + 1$ subsegments

Solution

- $E * n^n$ is just the sum over all possible results. Use $dp[x][y]$ to denote the sum of all possible results with x element, the largest of which is smaller than or equal to y.
- if segment do not include y, it is $dp[x][y-1]$
- otherwise, iterate t, the first appearance of y. Then we know that there are $len = min(x-k+1, t) - max(t-k+1, 1)$ segments of length k passes t. These segment provides a contribution of $w[y]^{len}$, then the left and right part divided by t are individual.
- $dp[x][y] = dp[x][y-1] + \sum dp[t-1][y-1] * dp[x-t][y] * y^{len}$
- Complexity $= O(n^3)$

Table of Contents

Names

Problem Statement

- Given n strings, find the smallest integer l, such that we can pick a subsequence with length no larger than l for each string, all of which are different.

Brute Force: Bipartite Graph Matching

Problem Statement

- Given n strings, find the smallest integer l, such that we can pick a subsequence with length no larger than l for each string, all of which are different.

- **Bisect answer interval**: turn an optimization task into a verification task.
- Given length l, we want to know if length l satisfies the given conditions.

Brute Force: Bipartite Graph Matching

Problem Statement

- Given n strings, find the smallest integer l, such that we can pick a subsequence with length no larger than l for each string, all of which are different.

- **Bisect answer interval**: turn an optimization task into a verification task.
- Given length l, we want to know if length l satisfies the given conditions.
- (Brute Force) List all names on the left, list all subsequences (with length no larger than l) on the right.
- Check whether maximum matching is n pairs.

A 'Greedy' Observation

Problem Statement

- Given n strings, find the smallest integer l, such that we can pick a subsequence with length no larger than l for each string, all of which are different.

- If a name has more than n choices, it will always be matched.

A 'Greedy' Observation

Problem Statement

- Given n strings, find the smallest integer l, such that we can pick a subsequence with length no larger than l for each string, all of which are different.

- If a name has more than n choices, it will always be matched.
- So we can ignore this name in matching and add "1" contribution in the end (equivalently, we can just consider the first n choices of each name in matching).

A 'Greedy' Observation

Problem Statement

- Given n strings, find the smallest integer l, such that we can pick a subsequence with length no larger than l for each string, all of which are different.

- If a name has more than n choices, it will always be matched.
- So we can ignore this name in matching and add "1" contribution in the end (equivalently, we can just consider the first n choices of each name in matching).

Pruning + Matching

Problem Statement

- Given n strings, find the smallest integer l, such that we can pick a subsequence with length no larger than l for each string, all of which are different.

- In the bipartite graph, there are n left nodes, $O(n^2)$ right nodes, and $O(n^2)$ edges .
- Classic bipartite matching algorithm:
  - Hungarian algorithm: $O(n^3)$

Pruning + Matching

Problem Statement

- Given n strings, find the smallest integer l, such that we can pick a subsequence with length no larger than l for each string, all of which are different.

- In the bipartite graph, there are n left nodes, $O(n^2)$ right nodes, and $O(n^2)$ edges .
- Classic bipartite matching algorithm:
  - Hungarian algorithm: $O(n^3)$
  - Dinic with maximum flow: Theoretically $O(n^6)$ (V^2E), but dinic always runs much faster, and in this specific task there might be a smaller upper bound.

Pruning + Matching

Problem Statement

- Given n strings, find the smallest integer l, such that we can pick a subsequence with length no larger than l for each string, all of which are different.

- In the bipartite graph, there are n left nodes, $O(n^2)$ right nodes, and $O(n^2)$ edges .
- Classic bipartite matching algorithm:
  - Hungarian algorithm: $O(n^3)$
  - Dinic with maximum flow: Theoretically $O(n^6)$ (V^2E), but dinic always runs much faster, and in this specific task there might be a smaller upper bound.
- Overall Complexity: $O(n^3\ logn)$

An Important Detail: Subsequence

Problem Statement

- Given n strings, find the smallest integer l, such that we can pick a subsequence with length no larger than l for each string, all of which are different.

- Calculate the shortest n subsequences of each name

An Important Detail: Subsequence

Problem Statement

- Given n strings, find the smallest integer l, such that we can pick a subsequence with length no larger than l for each string, all of which are different.

- Calculate the shortest n subsequences of each name
- Easy to get TLE
- One can use Trie (prefix tree)
- One can use DP to extend one character at a time (**Los Patrons**'s solution)

Performance

Problem Statement

- Given n strings, find the smallest integer l, such that we can pick a subsequence with length no larger than l for each string, all of which are different.

- 4 teams got accepted.
- **Los Patrons** took the first blood at 81 min.

Table of Contents

Orz

Problem Statement

- Given a sequence of competitions in which the top a_i participants get the job, answer q queries on the form "what's the probability that you qualify from at least one of the competitions in the range $[l, r]$, if your rank in each competition is uniformly random in the interval $[1, x]$".

Orz

- You don't qualify from the i^{th} competition with probability $1 - \frac{a_i}{x}$, so you don't qualify from any of the competitions with probability $\prod_{i=l}^{r}(1 - \frac{a_i}{x})$. Hence we seek $1 - \prod_{i=l}^{r}(1 - \frac{a_i}{x})$, so it's enough to be able to calculate this product fast, to a precision of 10^{-6}.

Actual Problem Statement

- Given a sequence of integers a_i, answer q queries on the form "what's $\prod_{i=l}^{r}(1 - \frac{a_i}{x})$", where l, r, x are given in each query.

Insight 1: If a_i are large it's easy

- If $a_i > \frac{x}{2}$ for all a_i, the value of $\prod_{i=l}^{r}(1 - \frac{a_i}{x}) < (\frac{1}{2})^{r-l+1}$.
- Hence if the difference between r and l is greater than 20, we can just output 1, as $(\frac{1}{2})^{20} < 10^{-6}$ and we just need a precision of 10^{-6}.
- If the difference between l and r is smaller than 20 we can just calculate the value by iterating through the interval.

Insight 2: If a_i are small we can Taylor expand

- Take logarithm: $ln(\prod_{i=l}^{r}(1-\frac{a_i}{x})) = \sum_{i=l}^{r} ln(1-\frac{a_i}{x})$.
- Since $\frac{a_i}{x} < 1$, we have $-ln(1-\frac{a_i}{x}) = \sum_{k=1}^{\infty} \frac{1}{k}(\frac{a_i}{x})^k$.
- Let $A = max(a_i)$, then we can pre-compute the prefix sums $\sum_{i=1}^{t} \frac{1}{k}(\frac{a_i}{A})^k$ for each value of t and for $k = 1, 2, ..., p$ for some suitable value of p (which we will find later). This lets us compute the sum $\sum_{i=l}^{r} \frac{1}{k}(\frac{a_i}{A})^k$ in O(1) for any k, so we can get approximations for the sum of the logarithms to p^{th} order in O(p), by summing up these values, each multiplied by $(\frac{A}{x})^k$.
- For each logarithm, the error is $-ln(1-\frac{a_i}{x}) - \sum_{k=1}^{p} \frac{1}{k}(\frac{a_i}{x})^k = \sum_{k=p+1}^{\infty} \frac{1}{k}(\frac{a_i}{x})^k$, which is decreasing in $\frac{a_i}{x}$. So for example if $a_i < x/2$ we can write a short program to find out the worst case for any given p.

Insight 2: If a_i are small we can Taylor expand

- If the worst case error in any logarithm is E, the total error is at most $(r - l + 1)E$, and our answer will be at most a factor $e^{(r-l+1)E}$ off. Doing a bit of maths and dividing into cases, we find that $p \approx 15$ will give a precise enough answer. (Note that the more factors we have, the smaller the product and hence the smaller the effect of the relative error on the absolute error, which is what matters. This is what can be used to get $p \approx 15$, but even ignoring this effect $p \approx 35$ can easily be shown to be precise enough, since the factor by which we're off will be within 10^{-6} of 1 and the answer is always less than 1. If implemented efficiently, $p = 35$ will pass the test data.)
- (Also, to be on the safe side, we can pick $p = 17$ or 18, which will definitely be precise enough.)

Combining the insights to get a solution

We can combine the two insights in the following way:

- Do all the pre-computation for the "small a_i"-solution up to $p = 17$.
- Set up some data structure to be able to do fast RMQ on the a_i's, eg a sparse table (for O(1)) or a segment tree (for O(log(n)).
- For each query, we recursively split the interval at the largest element, which we can find with our RMQ. Then we multiply together $(1 - \frac{a_i}{x})$ for the values we split at, and save the sum of all the Taylor expansions of the logarithms at those points.
- If the current product ever becomes less than 10^{-6} we can output 1 and go to the next query.

Combining the insights to get a solution

- If the largest element left in an interval is ever smaller than $\frac{x}{2}$, we can stop the recursion in that interval.
- When we exit the recursion it's either because we know the answer is $< 10^{-6}$, or because all elements left in $[l, r]$ are smaller than $\frac{x}{2}$. In the second case, we can do the "small a_i"-solution on the remaining elements (taking care to remove the contribution of the large elements, which we have already taken care of).
- Complexity: $\mathrm{O}(nlog(n) + 20q)$

A different approach

- It might also be possible to solve this problem using polynomial interpolation (although we haven't tried to implement it).
- If we multiply the product by x^n it becomes a polynomial, we could then do FFT to get the polynomial in $O(nlog^2(n))$.
- With some more FFT technology we can calculate the result of $f(k)$ for m different k in something like $O(klog^2(n))$.
- Finally we use a segment tree to do queries in logarithmic time.
- This approach is very implementation heavy, and we have not tried it.

Table of Contents

MetroBuilding

Problem Statement

- Given n points, each has two strings $A[i]$ and $B[i]$. calculate the maximum spanning tree, under that the value of edge (u, v) is $|LCP(A[u], B[v])| + |LCP(B[u], A[v])|$

Introduction: Boruvka algorithm for MST

- the process of Boruvka algorithm is divided into different stages. In each stage, for each vertex u, find the edge (u, v) with largest value and that u and v are not currently connected.
- sort all the edges found in that stage, and iterate each edge (u, v). if u and v are not connected, link them.
- keep playing stages until everything is connected.
- in the worst case, we can reduce the number of connected components by a half, so $O(logn)$ stages in total.

Solution 1: Trie + persistent SGT merge

- in each stage, build a trie for $A[u]$ over all u. Maintain a segment tree for each node in the trie. For a trie node x, its segment tree should contain the following: for all vertex u in the subtree of u (i.e. the string from the trie root to x is a prefix of $A[u]$), the segment tree should contain the hash of all the prefix of $B[u]$.
- To do this, just insert all the prefix of $B[u]$ in the leaf (the node when finishing the traversal of $A[u]$), and do a persistent segment tree merge in the end.
- there are at most $\sum$ prefix of $B[u]$, so Complexity of SGT merge is $O(\sum \log(\sum))$

Solution 1: Trie + persistent SGT merge

- Now, how to find the largest edge? For a vertex u. Travel its $B[u]$ over the trie. Whenever we processed a character, suppose we are at node x in trie, fix $LCP(B[u], A[v]) = x$ (or longer), now we want the largest $LCP(A[u], B[v])$ in all v in the subtree of x.
- we maintain the hash for all prefix of $A[u]$. Starting from the longest prefix, and keep discarding the last character until we find the corresponding hash value in the SGT of x (which contains all prefix of $B[v]$ of v in subtree of x). Since we will have stricter restrictions when going down, (i.e. if we can't find hash(C) in SGT of x, then we can't find hash(C) in SGT of son[x]), we don't have to start from the whole string every time.

Solution 1: Trie + persistent SGT merge

- In order to check if the edge is linked to another component, for each hash value, we also need to maintain, for each value in SGT, the largest and smallest component id it can come from. Check whether $id(get(u)) > small || id(get(u)) < large$
- In each stage, travelling on trie cost $\sum B$ and searching in SGT cost $O(\log n) * (\sum A + \sum B)$. In total $O(\sum \log \sum \log n)$

Solution 2: SQRT decomposition

- if $|max(A[i], B[i])| < k$, there is a brute force algorithm using Boruvka.
- In each stage, process a unordered map $((X, Y) -> cnt)$, meaning how many vertex u has X as a prefix of $A[u]$ and Y as prefix of $B[u]$.
- for each connected component, delete all the contribution of u in that unordered map (so that we won't find edges linking to same component). Iterate all prefix of $A[u]$ from shortest to longest. Suppose we have S now. Consider all prefix of $B[u]$, from longest to shortest. Suppose we have T now. while we find $cnt[T, S] = 0$, remove the last element of T.
- Complexity $= O(nk \log n)$

Solution 2: SQRT decomposition

- consider setting a bound σ. Get all vertices u with $|A[u]| + |B[u]| > \sigma$, run a brute force Kruskal for them. If we calculate LCP with SA, the Complexity $= \frac{n^2}{\sigma^2} + n \log n$
- after that, only keeps the first σ character for $A[i]$ and $B[i]$. run what we described on that last slide. Complexity $= O(n \log n \sigma$
- take $\sigma = \sqrt[3]{\frac{n}{\log n}}$ we yield $n(\sqrt[3]{n \log n} + \log n)$

Thanks!

Contact:
[Hugo Eberhard] hugoeberhard@gmail.com
[Yichen Huang] yichen.huang@stcatz.ox.ac.uk
[Xingjian Bai] xingjian.bai@sjc.ox.ac.uk