AACPP

We will be implementing and discussing solutions to interesting problems from collegiate-level competitive programming contests like ACM or ICPC.

Grading

There will be 12 meetings in the semester, and we plan to cover 6 tasks in a two-week cycle.

First, a problem statement is presented and uploaded to the judging system. You will have one week to individually think about it, design and implement a solution. Correct and fast code receives 10 points, with partial credit for suboptimal or partial solutions. For example, $O(n^2)$ solutions where $O(n)$ is optimal can be expected to receive around 3-4 points.

If you correctly solve the task, you can write a short report/presentation on the solution. It should contain an explanation of the statement (translating the story to a model), some sub-optimal or partial solution, how to improve it to get the optimal one, and time and memory complexity analysis. We will grade them out of 5 points.

After a week we meet and discuss the solution. You are encouraged to volunteer to present the analysis you have submitted for grading.

In the intervening week it is possible to upload the solution for additional 10 points. This way, even if you don't come up with the correct approach on your own, you can still get the points by implementing the solution presented after a week.

Note that this system still encourages at least attempting to solve the task initially, as the 10 -point packages are "independent"! For example, someone who uploads a partial solution first week for 3 points and then completes the task in the second week for 10 gets a total of 13. Solving the task correctly in the first week therefore grants 20 points in total, while "just" submitting in the second week gives at most 10.

In other words, there is 150 points in the course to get − for each of the 6 tasks: in the first week 10 for the solution + 5 for the report, in the second 10 for retries.

We will revisit the points to grades mapping at the end, in case it turns out our task choice was too difficult. In any case, 61 points should be guaranteed to pass.

Example - A Journey to Greece

Structure of a task: story, input specification, output specification, limits, example test.

Solution strategy

First look at limits to figure out what the hard part is. In this case you can recognise the TSP, so there must be some detail that makes it possible to pull off in reasonable time.

Make observations to simplify the task:

- The limit on P is merely 15.
- Most vertices are irrelevant, we can contract the edges and leave only shortest paths between vertices in the target set.
- The "stay" costs are static - we have to pay all of them exactly once, so we can just subtract them from the overall time limit and ignore afterwards.

This is enough to come up with a simple-enough brute solution:

- Find shortest paths between each pair of vertices in the target set by running Dijkstra P times, $O(P \cdot M \log N)$

- Check every possible visit order by following shortest paths, $O(P!)$.
- Check that again but replace the most expensive edge with a taxi jump.

$O(P!)$ is too much for the limit in the task - 15! is around 10^{12}. In this case it gives 4 points.

A good rule of thumb is that $O(n!)$ "works" for limits $n <\approx 10$, $O(2^n)$ for $n <\approx 20$; limits of 10^5 or 10^6 usually require linear or linear-logarithmic solutions.

However, TCP admits an $O(P^2 \cdot 2^P)$ dynamic programming solution. For each non-empty subset of vertices S and final vertex $v \in S$ we define the minimal cost it takes to visit S as:

$$c(S, v) = \min_{u \in S - \{v\}} c(S - \{v\}, u) + d(u, v)$$

where d is the shortest distance function computed before, and $c(\{0\}, 0) = 0$.

We need to modify this DP to take the taxi ride into account:

$$c(S, v, \text{taxi available}) = c(S, v)$$

$$c(S, v, \text{taxi used}) = \min_{u \in S - \{v\}} \min \begin{cases} c(S - \{v\}, u, \text{taxi used}) + d(u, v) \\ c(S - \{v\}, u, \text{taxi available}) + T \end{cases}$$

where T is the taxi cost. This doesn't worsen the time complexity. Memory used is bounded by the number of states in the DP, which is $O(P \cdot 2^P)$.

The final result can be found by iterating all reachable states (S, v, t) where S is equal to the entire target set and attempting to close the cycle by travelling from v to 0. If $t = \text{taxi available}$ we can also take the taxi jump as the last step instead of the $v \rightsquigarrow 0$ path.

Judging system

Demo: how to upload a solution and read the outcome.

Memory limit, time limit. There is a submission limit of 10.

The submission has to be a single file compiled to an exe. It runs isolated – no syscalls (no network, no filesystem), only the standard library.