Garden

Tags	On the full solution	On the subtasks
	Dynamic programming in a tree	Dijkstra

Analysis

Interestingly, there is also probably a solution with $O(N\ log_2\ N)$ complexity with Small-to-large ideas, but it would be very unpleasant to write it ☺. Also, another interesting thing is that there is a solution for a chain, using Dijkstra, but I don't feel the need to describe it here. It makes sense to think that $c_i \leq c_j$ for $(i < j)$, because otherwise, we can irrigate more gardens for less cost. We can precalculate for each vertex the minimum amount of electricity needed to cover all vertices at x distance from each vertex for each $(0 \leq x < N)$. When I use the term "power x", I refer to a pump, working for x minutes. When I use "cover", I mean a vertex being irrigated. suggest reading the whole analysis because I will skip explaining some details if explained in earlier subtasks.

Solution for a chain

The task screams for dynamic programming, at least I tried to make the tests good enough to prevent other solutions from gaining good points (I hope ≈ 600 line test generator is long enough). Interestingly, the solution for a chain and a tree will begin from the same core idea, which will be optimized.

Solution for $N \leq 75$

I haven't predicted a solution for the following N, but I'm ready to be surprised ☺.

Solution for $N \leq 500$

For convenience, let's renumber the vertices according to their position in the chain, with the leftmost being 1 and the rightmost being N. We will choose the water, released from the pumps, as we choose the water for vertex 1, then for vertex 2, …, and last for vertex N. Let's consider the DP with state `dp[vertex][balance]`. All upcoming DPs will have the same state, although they will have different recurrent dependencies/optimizations. If $balance > 0$, then water is incoming from the previous vertex, which will cover all vertex right from the current vertex, being at distance at most $balance - 1$ from the current vertex, including the current vertex itself. If $balance \leq 0$, then the previous vertex borrowed water from the current vertex, so all vertices left from the current vertex, being at distance at most $-balance$ from the current vertex, stay unirrigated. If $balance = 0$, then all vertices up the previous vertex will be irrigated, but the current vertex won't be irrigated. Then the answer to the task will be `dp[1][0]`. We can choose the water, released from the pump of the current vertex and look at how the balance changes. I will leave this as your exercise ☺.

Achieved complexity: $O(N^3)$

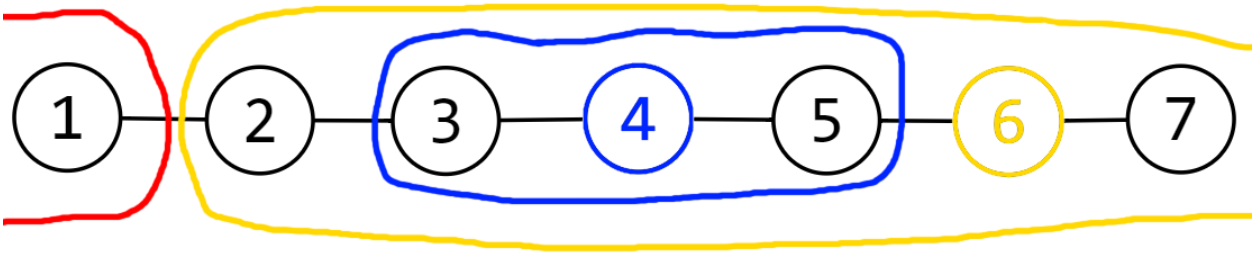
Garden

Implementation: `garden_23p.cpp`

Solution for $N \leq 2\,000$

Several optimizations can be made here. One of them is the use of a segment tree or Dijkstra with a different state of the DP, which as I said, I will not explain in the analysis. Let's optimize the current state. One option is to do fun interval minimums, which frankly I haven't looked at so I can save myself the headaches. Instead, one could apply the standard trick of making the balance parameter an inequality. One can easily prove that $dp[vertex][balance] \geq dp[vertex][balance - 1]$. Because of this, let $dp[vertex][balance]$ be the minimum cost if water of strength x enters the current vertex for $x \leq balance$. For this purpose, we can consider 3 options:

- To cover the necessary vertices with the next vertex: $dp[vertex + 1][balance - 1]$.
- For $balance < 0$, to cover the balance with the minimum cost for electricity: $dp[vertex + 1][-balance] + c[balance + 1]$, if $balance + 1 \leq t[vertex]$.
- To cover the balance with more than the minimum possible cost: $dp[vertex][balance - 1]$. In this way we will consider all cases with the power of the water being $-balance + 2$, $-balance + 3$, $-balance + 4$, …, $t[vertex]$. It's never optimal to use power $\leq -balance$. To prove this, look at the picture below:



- Red – covered vertices to the moment.
- Blue – covered from vertex 4.
- Yellow – covered from a vertex from the right, in this case, 6.

In this case, the current vertex is 4 and $balance = -2$. The picture illustrates the case when we cover the vertices with power being equal to 2. As we don't cover all the vertices left from 4, some vertex from the right must cover it (in this case, 6). In this way the water from 4 is meaningless.

In this way, we get 36 points. If any contestant solves the current subtask with this idea, he will hardly remain with 36 points, he will probably reach a full solution.

Achieved complexity: $O(N^2)$

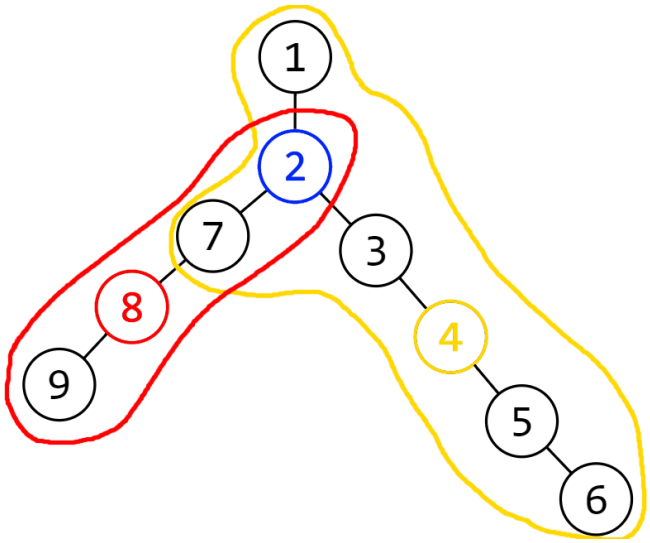
Implementation: `garden_36p.cpp`

Solution for a tree

Let's jump right into the next subtasks, without a parachute. I hope you land smoothly ☺.

Solution for $N \leq 75$

We will use the DP with state `dp[vertex][balance]`. The state in a tree will mean that the parent of the current vertex has lent or borrowed water from it. Let's look at how the recurrent dependency changes. We consider the same cases – whether the pump in the current vertex works or not. I will leave the easier cases for your exercise ☺. The case when $balance < 0$ stands out. Then there are two cases – to cover the vertices the parent has borrowed from the current vertex, or hand over the water to some vertex in the subtree of the current vertex. The case when we hand over the borrowed water to the subtree is more interesting. It's always optimal to hand over the water to only one child of the current vertex. To consider the case where we cover the water from ≥ 2 vertices, look at the illustration below:



- Blue – the current vertex (in this case 2).
- Red – The covered from the vertex, from which less water comes out (in this case 8).
- Yellow – The covered from the vertex, from which more water comes out (in this case 4).

As you may notice from the picture, the current vertex is 2 and $balance = -1$. The child from which more water comes out will cover all the vertices outside of the subtree of 2 which will be covered by the child, from which less water comes out. Because of this, we won't need to hand over the borrowed water to more than one vertex. Nothing prevents water coming out of the subtree of more than one child – sometimes it will be necessary to cover the subtree of the child itself. Because of this, we will choose the power of the water for the current vertex/its children. The details will be for your exercise ☺.

The solution, probably for a very logical reason that I didn't take the time to think about, doesn't pass the tests for $O(N^3)$ complexity, but it passes the tests provided for $O(N^4)$ complexity.

Garden

Achieved complexity: $O(N^4)$?

Implementation: `garden_40p.cpp`

Solution for $N \leq 500$

Many optimizations can be made to solve this subtask, but I don't feel the need to describe them here. If you are curious, you may check the difference between the previous and the current implementations and see the optimization.

Achieved complexity: $O(N^3)$

Implementation: `garden_65p.cpp`

Solution for $N \leq 2\,000$

To optimize the DP even further, we will use the same optimization we used for a chain. The cases we need to consider are identical to the solution for $N \leq 75$.

Achieved complexity: $O(N^2)$

Implementation: `garden_100p.cpp`

P.P: For implementation reasons, in most of my solutions, I've split the DP into two functions: `plus` and `minus`, respectively for the cases when $balance > 0$ and $balance \leq 0$. The parameter in the `minus` function is equal to $-balance$.

P.P №2: The task is old, relatively a year and a half old, having undergone numerous transformations in the statement, affecting the meaning and plot. If you are curious about what they are, you can text me and I can show you ☺.

Author: Boris Mihov