EPFL

Solutions to the Graded Programming Homework

Problem B - `lesc.cpp`

We model the sewage system as a rooted tree T with vertices $0, \dots, n$ and root 0, where vertices u and v are connected if there is a pipe between them. Given T, we define the satisfaction function $s : \{0, \dots, n\} \to \mathbb{Z}$ as $s(u) = \sum_{v \in T_u} a_v$, where T_v denotes the subtree of T rooted at v.

Note that $s(u) = a_u + \sum_{v \in \text{children}(u)} s(v)$, so if we have already computed the value $s(v)$ for every children of u, it is easy to compute the value of $s(u)$. Hence, we can compute $s(u)$ for every $u \in \{0, \dots, n\}$ by starting a post-order DFS from the root vertex 0, which takes $O(n)$ time.

If we do not remove any pipe, then the total satisfaction remains $s(0) = s(0) - 0$. If we remove the pipe connecting a vertex u to its parent, the total satisfaction becomes $s(0) - s(u)$. If we close the plant, the total satisfaction becomes $0 = s(0) - s(0)$. To solve the problem, we need to find the best out of these three possibilities, which is achieved by returning the value $s(0) - \min\{0, \min_{u \in \{0,\dots,n\}} s(u)\}$. This part also takes $O(n)$ time.

Problem C - `stable_bridge.cpp`

Given the costs $c_0, \dots, c_n$ and the budget X, we define the function $f : \{1, \dots, n-1\} \to \mathbb{N}$ such that $f(k)$ is the minimum cost of a bridge where consecutive piers are within k meters of each other (and the bridge must have a pier at meter 0 and a pier at meter $n-1$). Then, our goal is to find the smallest $k^* \in \{1, \dots, n-1\}$ such that $f(k^*) \le X$ if it exists.

Note that $f(k) \le f(k')$ for any $k \ge k'$, since larger k gives us more flexibility for deciding where to place the piers. Therefore, for any $k \in \{1, \dots, n-1\}$, if $f(k) \le X$ then we know that $k^* \le k$, and if instead $f(k) > X$ then we know that $k^* > k$. This observation suggest that instead of trying all possible $n-1$ values of k until we find k^*, we use binary search: after trying a certain value k, we only need to try either those below k or those above k, but not both.

Given some k, we want to compute $f(k)$. For $i = 0, \dots, n-1$, let $\text{dp}_k(i)$ be the minimum cost to build a bridge that has a pier at meter 0 and at meter i, and where every two consecutive piers are within k meters of each other. Then, observe that $f(k) = \text{dp}_k(n-1)$. Moreover, we have the base case $\text{dp}_k(0) = c_0$ and the recurrence relation $\text{dp}_k(i) = c_i + \min_{j:\, \max\{0, i-k\} \le j < i} \text{dp}_k(j)$ for every $i = 1, \dots, n-1$. A trivial implementation will take $O(nk)$ time to compute $\text{dp}_k(n-1)$, but we can use some data structures to compute $\min_{j:\, \max\{0, i-k\} \le j < i} \text{dp}_k(j)$ in time faster than $O(k)$.

One option is to maintain the last k values $\text{dp}_k(i-k), \dots, \text{dp}_k(i-1)$ in a binary search tree where insertion, deletion, and finding the minimum element take time $O(\log k)$. This is what `stable_bridge.cpp` does: when computing $\text{dp}_k(i)$, it fetches the minimum value among $\text{dp}_k(i-k), \dots, \text{dp}_k(i-1)$ from the data structure; then, it removes the value $\text{dp}_k(i-k)$ and adds the newly computed value $\text{dp}_k(i)$. This implementation takes $O(n \log k)$ time to compute $f(k)$.

The time complexity is $O(\log(|\text{search space for } k|) \cdot (\text{time to compute } f(k)))$, so we get a running time of $O(n \log^2 n)$.