Unless stated otherwise, for all questions in this homework involving dynamic programming, you need to provide a solution with explicit memoization.

19 (100 PTS.) Superstring theory.

You are given a DFA $M = (Q, \Sigma, \delta, s, A)$ with n states, where $|\Sigma| = O(1)$.

For two strings $x, y \in \Sigma^*$, the string x is a ***superstring*** of y, if one can delete characters from x and get y.

Let w be a given input string with m characters.

19.A. (25 PTS.) Let $q, q' \in Q$ be two states of M. Prove, that the shortest string w'', such that $\delta^*(q, w'') = q'$ is of length at most $n - 1$.

Solution:

Lemma 7.1. *Let q, q' be two states of Q, and let $w = w_1 w_2 \ldots w_t$ be the shortest string, such that $\delta^*(q, w) = q'$. We have that $t \leq n - 1$.*

Proof: For $i = 0, \ldots, t$, let $q_i = \delta^*(q, w_1 w_2 \ldots w_i$, and observe that $q_0, q_1, \ldots, q_t$ are all unique. Indeed, if $q_i = q_j$, then the following string

$$w' = w_1 \ldots w_i w_{j+1} \ldots w_t$$

is shorter, and, as can be easily verified, as the property that $\delta^*(q, w') = \delta^*(q, w) = q'$. A contradiction.

Since $q_0, q_1, \ldots, q_t$ are all unique, and there are at most $n = |Q|$ states, it follows that $t \leq n - 1$, as claimed. ■

19.B. (25 PTS.) Prove, that if there is a superstring x of w, such that x is accepted by M, then the shortest such superstring is of length at most $(n + 1)(m + 1)$, where $n = |Q|$ and $m = |w|$.

Solution:

Let $w = w_1 \ldots w_m$, and let $w' = s_0 w_1 s_1 w_2 \ldots s_{n-1} w_m s_m$ be the shortest superstring of w that is accepted by M, where $s_i \in \Sigma^*$. Consider the states $q_i = \delta^*(s, s_0 w_1 \ldots s_{i-1} w_i)$ and $q'_i = \delta^*(s, s_0 w_1 \ldots s_{i-1} w_i s_{i+1})$. By optimality, we have that s_{i+1} must be the shortest string that leads from q_{i-1} to q_i. By the above lemma, this implies that $|s_{i+1}| \leq n$. We conclude as such that

$$|w'| \leq (m + 1)n + m \leq (m + 1)(n + 1),$$

as claimed.

19.C. (50 PTS.) Describe an algorithm, as efficient as possible, that computes the shortest superstring z of w, such that z is accepted by M. (One can solve this problem using dynamic programming, but this is definitely not the only way.)

Solution:

Let us start with the dynamic programming solution. Let $f(q,i,j)$ be true if there is a superstring w' of $w_1 w_2 \ldots w_i$ of length $i+j$, such that $\delta^*(s,w') = q$. For a state $q \in Q$, and $c \in \Sigma$, let $\delta^{-1}(q,c) = \{q' \in Q \mid \delta(q',c) = q\}$ be the inverse transition function. Similarly, let $\delta^{-1}(q) = \bigcup_{c\in\Sigma} \delta^{-1}(q,c)$. We have that

$$f(q,i,j) = \bigvee_{c\in\Sigma} \bigvee_{q'\in\delta^{-1}(q,c)} f(q',i,j-1) \quad \vee \bigvee_{q'\in\delta^{-1}(q,w_i)} f(q',i-1,j),$$

Here, the base case is

$$f(q,i,j) = \begin{cases} 1 & q = s, i = 0. j = 0 \\ 0 & q \neq s, i = 0. j = 0 \\ 0 & i < 0 \text{ or } j < 0. \end{cases}$$

Clearly, assuming all the previous values had been computed, computing $f(q,i,j)$ takes $O(n)$ time.

We are interested in computing $f(q,i,j)$, for all $q \in Q$, $i = 0,\ldots,m$, and $j = 0,\ldots,(n+1)(m+1)$ (here we are using part (B)). As such, using memoization, this results in dynamic programming solution with running time $O(n \cdot m \cdot nm \cdot n) = O(n^3m^2)$.

Once we computed all these values, it is now easy to find the minimum j, such that $f(q,n,j)$ is true and $q \in A$.

The running time can be improved by being more careful about the details. But we present an alternative solution which might be conceptually nicer.

Solution:

Using the above notations, let use define the state space

$$S = \{(q,i,j) \mid q \in Q, 0 \le i \le m, 0 \le j \le (n+1)(m+1)\}.$$

The size of $|S|$ is $O(n^2m^2)$.

We now build a directed graph on S. We put an edge between (q,i,j) and $(q',i+1,j)$ if $\delta(q,w_{i+1}) = q'$, for all q,q',i,j. Similarly, we put an edge between (q,i,j) and $(q',i,j+1)$, if there is a character $c \in \Sigma$ such that $\delta(q,c) = q'$.

Since $|\Sigma| = O(1)$, it follows that this graph can be build in $O(1)$ time per vertex. As such, the resulting directed graph $G = (S,E)$ can be computed in $O(n^2m^2)$ time.

Next we do a **BFS** on G, starting from $(s,0,0)$. This breaks the graph into layers $L_0, L_1, \ldots$. By scanning, we can find the first layer L_ν, such that L_ν contains a state (q,n,j) and $q \in A$. Observe that $\nu = n+j$ in this case. Since **BFS** takes time proportional to the size of the graph, it follows that the running time of the algorithm is $O(n^2m^2)$.

Solution:

Faster solution.

Compute for any start state $q \in Q$, the minimal length of a string that one has to use to get to any other state of Q. This can be done by **DFS** in $O(|Q||\Sigma|) = O(n)$ time. Doing

this for all states takes $O(n^2)$ time. Let $L(q, q')$ be $n \times n$ table with the length of this string from q to q', over all $(q, q') \in Q \times Q$.

Now, we define the recursive function $f(q, i)$ as the length of the shortest superstring $\tau_{q,i}$ that includes the first i characters of w, such that $\delta^*(s, \tau_{q,i}) = q$ (if no such strings exists, then this quantity is infinity).

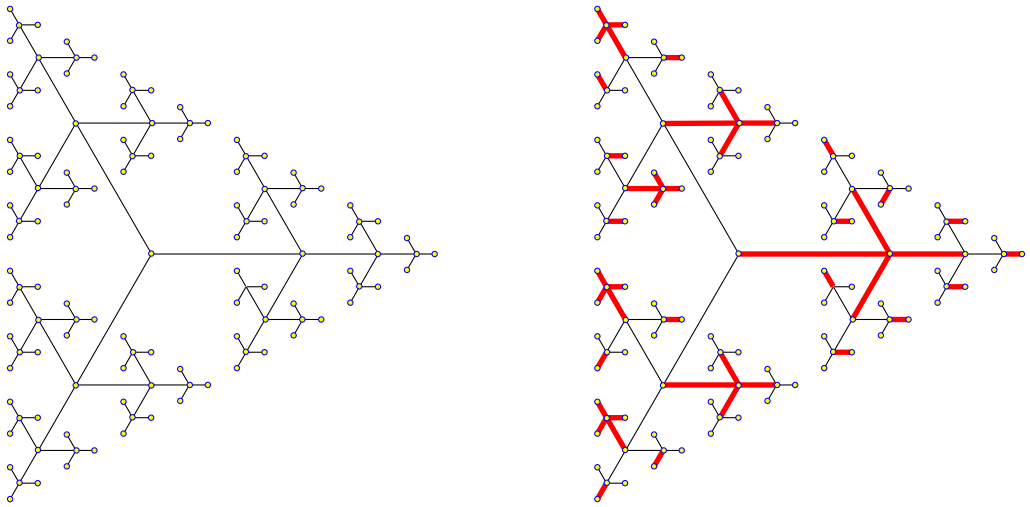
We now have that

$$f(q, i) = \min_{q' \in Q} \Big(f(q', i-1) + 1 + L(\delta(q', w_i).q) \Big).$$

This recursive function has $n \times (m+1)$ different recursive calls. Each recursive call takes $O(n)$ time to handle. It follows that using memoization this would take $O(n^2 m)$ time.

20 (100 PTS.) Stars in a tree.

A *star*, in a tree, is a vertex together with *all* the edges adjacent to it. A collection of stars is ***independent*** if no two stars shares a vertex. The *mass* of an independent set of stars S is the total number of edges in the stars of S.



Describe an algorithm, as efficient as possible, that computes the maximum mass of any set of independent stars in the given tree T. Here the input tree T has n vertices.

Solution:

Pick an arbitrary vertex r of T, and root T at r.

We define a recursive solution for a vertex u of T, which is appropriate for the subtree of u. The recursive solution has a state s, which specifies what state u has to be in the solution:

- $s = C$: `center`. The vertex u can used in any solution for T_u that uses u as a center.
- $s = U$: `unused`. The vertex u can not be used in any star of a solution for T_u.
- $s = E$: `edge`. The vertex u can be used in any solution for T_u, as long as u is an endpoint of an edge of a star (but not the center of the star).

Let $f(u, s)$ the optimal solution for T_u where u is of state s. We now just need to go through the cases.

- `center`: $f(u, C) = d(u) - 1 + \max_{u' \in \text{children}(u)} f(u'.U)$.
- `unused`. Let us define
$$g(u') = \max(f(u', U), f(u', E)).$$
Clearly we have that $f(c, U) = \sum_{u' \in \text{children}(u)} g(u')$.
- `edge`: For $f(u, E)$ we have to pick the child that is the center of the star that uses u. All the other children have to be either `unused` or in an `edge` state.
$$f(c, E) = 1 + \max_{u' \in \text{children}(u)} (f(c, U) - g(u') + f(u', C)).$$
The desired optimal solution is $\max(f(r, C), f(r, E), f(r, U))$. Using memoization and dynamic programming, it follows that the value of the optimal solution can be computed in $O(n)$ time.

21 (100 PTS.) Exploring Narnia.

Three travelers had decided to travel to Narnia. Since they are all anti-social, they do not want to travel together. Instead, the first traveler would move from location p_1 to location p_2, and so on till arriving to location p_n (here, $P = p_1, \ldots, p_n$). A location is just a point in the plane (i.e., $p_i \in \mathbb{R}^2$). Similarly, the second traveler is going to move along $Q = q_1, \ldots, q_n$, and the third traveler is going to move along $R = r_1, \ldots, r_n$. Every day, a traveler might decide to stay in its current location, or move to the next location (the traveling between two consecutive locations takes less than a day).

By the evening of each day, the travelers arrive to their desired locations for the day, which can be thought of as a configuration $(p, q, r) \in P \times Q \times R$.

A configuration (p, q, r) is *ℓ-legal* if
$$\Delta(p, q, r) = \max\big(\|p - q\|, \|p - r\|, \|q - r\|\big) \leq \ell,$$
where $\|p - q\|$ is the Euclidean distance between the points p and q (this distance can be computed in constant time). (Intuitively, the travelers do not want to be too far from each other, so that if one of them is hurt, the others can quickly come over and help.)

21.A. (50 PTS.) Given the point sequences P, Q, R, and a parameter $\ell > 0$, describe an algorithm, as fast as possible, that decides if there is a motion planning schedule from (p_1, q_1, r_1) to (p_n, q_n, r_n) such that all the configurations used are ℓ-legal. Such a schedule is an *ℓ-feasible schedule*.

Here, it is legal for several travelers to move in the same day, but they are not allowed to move back to a previous location they already used. Furthermore, a traveler can move, in one day, only one location forward in their sequence.

Solution:

We build a graph over the state space $P \times Q \times R$. Here, we put an edge between (p_i, q_j, r_k) and $(p_{i'}, q_{j'}, r_{k'})$ if $i' \in \{i, i+1\}$ or $j' \in \{j, j+1\}$ or $k' \in k, k+1$ and $(i, j, k) \neq (i', j', k')$, for all $i, j, k \in \{1, \ldots, n\}$. Let G be the resulting graph.

A state (p_i, q_j, r_k) is *illegal* if $\Delta(p_i, q_j, r_k) > \ell$. We remove any edge adjacent to an illegal state. Clearly, the desired motion is any path in the resulting graph from (p_1, q_1, r_1) to (p_n, q_r, r_n), and this can be computed in $O(n^3)$ time using **DFS** or **BFS**.

21.B. (50 PTS.) Given P, Q, R, as above, describe an algorithm, as fast as possible, that computes the minimum ℓ for which there is an ℓ-feasible schedule.

Solution:

We compute the same graph G as above, except that for any edge in the graph $(p_i, q_j, r_k) \to (p_{i'}, q_{j'}, r_{k'})$ we label it with

$$\max\Big(\Delta(p_i, q_j, r_k),\ \Delta(p_{i'}, q_{j'}, r_{k'})\Big).$$

(We no longer remove edges.).

Clearly, we are looking for the path from (p_1, q_1, r_1) to (p_n, q_r, r_n) such that the maximum label on an edge of the path is minimized. One can easily do that by computing all the $3n^2$ possible values (i.e., all the pairwise distances in $P \times Q$, $P \times R$, and $Q \times R$). Then use binary search using the algorithm from (A) as a subroutine. The resulting running time is $O(n^2 \log n + n^3 \log n)$.

We can speed it up. We do not really need to search for the optimal value. Instead, we can compute the value of the best solution arriving to a configuration. Formally, we define the recursive function

$$g(i,j,k) = \max\Bigg(\Delta(p_i, q_j, r_k),\quad \min_{i'=i-1,i}\ \min_{j'=j-1,j}\ \min_{k'=k-1,k} g(i',j',k')\Bigg).$$

With the boundary cases being:

$$g(1,1,1) = \Delta(p_1, q_1, r_1) \qquad \text{and} \qquad g(i,j,k) = +\infty \text{ if } i = 0 \text{ or } j = 0 \text{ or } k = 0.$$

This is just a 3d variant of edit distance, and $g(n,n,n)$ can be computed in $O(n^3)$ time using the regular dynamic programming/memoization techniques.