2008 Peking University
Campus Programming Contest

Problem Set Solution Sketch

Problem A. Escort of Dr. Who How

An *evolving graph* is a dynamic directed graph whose topology changes over time. A *journey* is the counterpart in an evolving graph of a path in a directed graph. It is defined as a sequence of edge-time pairs $\langle(e_1, \tau_1), (e_2, \tau_2), \ldots, (e_l, \tau_l)\rangle$ where for each $1 \le i \le l$, τ_i is the traversal time of edge e_i, and along which one can travel from the starting vertex of e_1 to the ending vertex of e_l in agreement with the timing constraints of every edge.

A *fastest* journey $J = \langle(a_1, \tau_1), (a_2, \tau_2), \ldots, (a_l, \tau_l)\rangle$ from s to t is a journey that minimizes $\tau_l + c_l - \tau_1$. An obversation that is crucial to solving the problem is that J can be "slided" over time subject to the timing contraints of its constituent edges so that it remains fastest. That is to say, if J is a fastest journey, then the journey $J+\epsilon = \langle(a_1, \tau_1+\epsilon), (a_2, \tau_2+\epsilon), \ldots, (a_l, \tau_l+\epsilon)\rangle$ where ϵ is some real number is also a fastest journey as long as $b_i \le \tau_i + \epsilon < \tau_i + c_i + \epsilon \le e_i$ for each $1 \le i \le l$.

Deriving from the above observation, we can impose an additional constraint on a fastest journey J that $\tau_i = b_i$ for some $1 \le i \le l$ without affecting the optimality of J.

We resolve the trouble of lacking the a priori knownledge of which edge in J satisfies the aforementioned constraint by enumerating each edge. Let a_i be that edge. J must arrive at x_i no later than time b_i and departure from x_i no earlier than time e_i. In order that $\tau_l + c_l - \tau_1$ is minimized, J must be the concatenation of a journey from s to x_i which departures from s as late as possible and another one from x_i to t which arrives at t as early as possible. We can identify such two journeys using a variant of Dijkstra's algorithm for the single-source shortest paths problem.

Problem B. Full Steiner Topologies

First we derive a closed-form formula to compute T_n, the number of different full Steiner topologies, for any n. Consider the gadget shown in Figure 1. Let T be full Steiner topology with $n-1$ leaves. To add a new leaf w to T, we can pick an arbitrary edge e connecting two vertices u and v, create a new Steiner vertex s in the middle, and connect w to s. It is obvious that there is a one-to-one correspondence between each pair (T, e) and the resulting full Steiner topology. From this we have following recurrence relation:

$$T_n = \begin{cases} 1 & n = 3, \\ (2n-5)T_{n-1} & n > 3. \end{cases}$$

The above recurrence relation can be solved in closed-form:

$$T_n = (2n-5)!! = \frac{(2n-5)!}{2^{n-3}(n-3)!}.$$

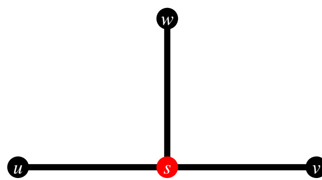

Figure 1: A gadget for adding a leaf to an existing full Steiner topology

We evaluate T_n via its natural logarithm

$$\ln T_n = \ln(2n-5)! - \ln(n-3)! - (n-3)\ln 2.$$

Stirling's approximation states that

$$n! \approx \sqrt{2\pi n}\left(\frac{n}{e}\right)^n.$$

More precisely,

$$n! = \sqrt{2\pi n}\left(\frac{n}{e}\right)^n e^{\lambda_n}$$

where $\frac{1}{12n+1} < \lambda_n < \frac{1}{12n}$. Taking the natural logarithm of both sides, we have

$$\ln n! = \left(n+\frac{1}{2}\right)\ln n - n + \frac{\ln 2\pi}{2} + \lambda_n.$$

We use $\hat{\lambda}_n = \frac{1}{12n}$ to approximate λ_n, then our approximation of $\ln n!$ is subject to an error bounded by $\frac{1}{12n(12n+1)}$. Subsequently, our approximation of $\ln T_n$ is subject to an error bounded by $\frac{1}{12(n-3)(12n-35)}$, which is less than $\ln\frac{1}{0.9995} \approx 5.001\cdot 10^{-4}$ for $n \geq 7$, and thus is a sufficiently accurate approximation. For small values of n, we can hard-code the values of T_n to avoid any trouble with precision.

Problem C. Illuminated Planet

The solution is best explained by Figure 2. The plane passing through the space probe and the centers of the planet and the star intersects the surfaces of the planet and the star at two circles. The two sectors shaded in red and green highlight the arc illuminated by the star and the one visible from the space probe on the circle representing the planet, respectively. A positive portion of the illuminated part of the planet's surface is visible from the space probe if and only if the two arcs properly overlap each other.

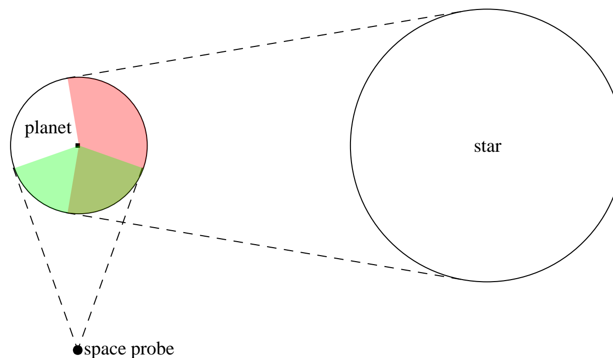

Figure 2: Illustration of the solution to Problem C

Problem D. Polygon Division

This problem is based on the well-known problem of counting divisions of regular polygons into triangles, the answers to which are the Catalan numbers. We solve this problem using a dynamic programming approach.

Denote by T_n the number of divisions of a regular n-gon into triangles and/or quadrangles for any $n \geq 3$. We number the vertices of the n-gon 1 through n in either clockwise or counterclockwise order. We try to find out a recurrence relation of T_n by considering the side connecting vertices 1 and n in the divisions. For the sake of notational simplicity, we denote by e_{ij} a side or diagonal connecting two vertices i and j ($1 \leq i < j \leq n$) and define $T_2 = 1$.

If e_{1n} forms part of a triangle, there is some vertex i ($2 \leq i \leq n-1$) such that e_{1i} and e_{in} divide the n-gon into a i-gon, the triangle formed by vertices 1, i and n, and a $(n-i+1)$-gon. We can then further divide the i-gon and the $(n-i+1)$-gon. In this case, there are

$$\sum_{i=2}^{n-1} T_i T_{n-i+1}$$

different divisions.

If e_{1n} forms part of a quadrangle, there are two vertices i and j ($2 \leq i < j \leq n-1$) such that e_{1i}, e_{ij}, e_{jn} divide the n-gon into a i-gon, a $(j-i+1)$-gon, a $(n-j+1)$-gon and the quadrangle formed by vertices 1, i, j and n. We can then further divide the three polygons other than the quadrangle containing e_{1n}. In this case, there are

$$\sum_{i=2}^{n-2} \sum_{j=i+1}^{n-1} T_i T_{j-i+1} T_{n-j+1}$$

different divisions.

Adding the numbers of divisions in both cases, we come to the following recurrence relation:

$$T_n = \sum_{i=2}^{n-1} T_i T_{n-i+1} + \sum_{i=2}^{n-2} \sum_{j=i+1}^{n-1} T_i T_{j-i+1} T_{n-j+1}.$$

This recurrence relation immediately leads to an $O(n^3)$-time algorithm to compute T_i for all $3 \leq i \leq n$. We can rewrite it as

$$U_n = \sum_{i=2}^{n-1} T_i T_{n-i+1}$$

$$T_n = U_n + \sum_{i=2}^{n-2} T_i U_{n-i+1}$$

to obtain an $O(n^2)$-time algorithm.

Problem E. Tower of Hanoi

If the order of equisized disks is not required to be preserved, the problem can be simplified a little. The optimal solution will always transfer all disks of the same size altogether from one

peg to another, reversing their order. Denote by T_n the number of moves needed. We can derive the following recurrence relation:

$$T_n = \begin{cases} a_1 & n = 1, \\ 2T_{n-1} + a_n & n > 1. \end{cases}$$

Solving the recurrence relation gives

$$T_n = 2^{n-1}a_1 + 2^{n-2}a_2 + \cdots + 2^1 a_{n-1} + a_n.$$

Observe that all disks of any size other than n are transferred an even number of times and subsequently have the order preserved. And if $a_n = 1$, T_n is readily the answer to the unsimplified problem.

The only difficulty that we may encounter is the case that $a_n > 1$. Examined from an alternative perspective, the optimal solution must first transfer all but the bottom disk to another disk to the auxiliary peg, then the bottom disk to the target peg, and lastly the other disks to the target peg. Consequently, all non-bottom disks must be moved an even number of times, which enables us to ignore the ordering constraint on them. Thus, that solution is identical to the solution to the instance with $n' = n + 1$ and $\{a'_1, a'_2, \ldots, a'_{n'}\} = \{a_1, a_2, \ldots, a_{n-1}, a_n - 1, 1\}$, the number of moves needed by which is given in the previous paragraph.

Problem F. PopKart

Each kart can be represented by a point (v, w) in a Cartesian plane. The points corresponding to karts of class 1 form a "skyline" when joined using line segments in increasing order of the v-coordinates. Such a "skyline" can be found as follows.

We sort the points in increasing order first by their v- then by their w-coordinates. The last point in this order must be rightmost point in the "skyline". We denote it by (v_k^1, w_k^2). The second rightmost point in the "skyline", which we denote by (v_{k-1}^1, w_{k-1}^1), must the last point in sorted order such that $v_{k-1}^1 < v_k^1$ and $w_{k-1}^1 > w_k^1$. Similarly, the third rightmost point (v_{k-2}^1, w_{k-2}^1) must be the last point in sorted order such that $v_{k-2}^1 < v_{k-1}^1$ and $w_{k-2}^1 > w_{k-1}^1$. After locating all k points in the "skyline", we have found the k karts of class 1. We then remove them from the collection. The "skyline" karts of the remaining ones are those of class 2. We repeat this process until all karts are classified.

To find the immediately next rightmost point in the "skyline" efficiently, we use the segment tree data structure. Each node in the segment tree covers a range of points in sorted order and store the minimum and maximum v- and the maximum w-coordinates of the points in the range. These satellite data help refine the search process in the segment tree. After the desired point is found, it is removed from the segment tree, satellite data of affected nodes are also modified accordingly. Segment tree operations takes $O(\log n)$ each, which contributes a $O(n \log n)$-time solution.

Problem G. Pumping Lemma

A string $w = xyz$ satisfying the given requirements exists iff there is a loop or cycle of state transitions in the state chart which is reachable from the start state s and from which some

final state is reachable. To locate such a loop or cycle, we regard the state chart as an edge-labeled directed graph. We perform a depth-first search starting from s. We try to find an edge connecting some vertex v which is reachable from s and from which some final state is reachable to itself or an ancestor w of v in the depth-first search tree. If such an edge edge cannot be found, w does not exist. Otherwise, there exist either a loop $v \rightsquigarrow v$ or a cycle $v \rightsquigarrow w \rightsquigarrow v$ and subsequently a walk $s \rightsquigarrow v \rightsquigarrow v \rightsquigarrow f$ where f is some final state. We collect the labels along $s \rightsquigarrow v$, $v \rightsquigarrow v$ and $v \rightsquigarrow f$ to form x, y and z, respectively.

Problem H. Subimage Recognition

We enumerate the columns of pixels that are removed from image B, then check whether image A is a subimg of the image B' resulting from the removal of those columns.

Problem I. Typographical Ligatures

We scan the text sequentially, recognize the use of each glyph following the leftmost longest rule and count them en passant.

Problem J. Zen Puzzle Garden

The basic strategy is depth-first state space search. We build up a solution by finding raking paths one by one. There are several optimizations which allow the search process to complete in reasonable time:

1. Existing raking paths may divide the unraked squares into several disjoint connected blocks. If any of them cannot be raked abiding to the game rules, the current branch of the search tree can then be pruned. Considering the blocks in increasing order of their sizes generally increases the chance for early pruning to take place.

2. Some special cases can be handled more efficiently. For example, if some connected block of squares is unreachable from outside the sand, there is no hope that the game can be completed.

3. We can avoid repeated exploration of a branch of the search tree by appropriately ordering the candidate raking paths and using hash tables to store previous results.