POJ Founder Monthly Contest – 2008.06.29 Solution Sketches

POJ Administrative Staff

July 10, 2008

Contents

Problem A Professor Ben

Let the prime factorization of N be

$$N = \prod_{i=1}^{m} p_i^{k_i}, \tag{A.1}$$

where $p_1, p_2, \ldots, p_m$ are distinct prime numbers. Denote by $L(n)$ the number of factors of any number n. For any factor n of N whose prime factorization is

$$n = \prod_{i=1}^{m} p_i^{l_i}, \tag{A.2}$$

we have

$$L(n) = \prod_{i=1}^{m} (l_i + 1). \tag{A.3}$$

Summing the cube of the right-hand side of Equation A.3 over all possible values of $l_1, l_2, \ldots, l_m$, we have

$$\begin{aligned} \sum_{l_1, l_2, \ldots, l_m} \left[\prod_{i=1}^{m} (l_i + 1) \right]^3 &= \prod_{i=1}^{m} \sum_{l_i=1}^{k_i} (l_i + 1)^3 \\ &= \prod_{i=1}^{m} \left[\frac{(k_i + 1)(k_i + 2)}{2} \right]^2 . \end{aligned} \tag{A.4}$$

Problem B Sheryl's Circuit I

If "1" does not appear in the desired signal, the answer is trivially zero; otherwise, the optimal strategy must be to toggle the least number of bits to achieve the first "1" and flip one bit upon each change in the output.

We use a dynamic programming approach to determine the least number of bits to toggle so that the output is "1". Label the gates in a top-down order similar to that of numbering elements of a binary heap in array representation. The topmost gate is gate 0. For any gate i, its two input-feeding gates, if they exist, are labelled $2i + 1$ and $2i + 2$, respectively. Denote by O_i the least number of "1" bits needed to make gate i output "1". It follows from the definitions of OR and AND gates that

$$O_i = \begin{cases} 1 & \text{gate } i \text{ is an OR gate,} \\ 2 & \text{gate } i \text{ is an AND gate,} \end{cases} \tag{B.1}$$

if gate i is in the bottom level, and

$$O_i = \begin{cases} \min\{O_{2i+1}, O_{2i+2}\} & \text{gate } i \text{ is an OR gate,} \\ O_{2i+1} + O_{2i+2} & \text{gate } i \text{ is an AND gate,} \end{cases} \tag{B.2}$$

otherwise.

Problem C Sheryl's Circuit II

We do not have to care about the number of falling edges for it must be exactly one less than that of rising edges. We use a dynamic programming approach that works bottom up as we did with Problem B.

Denote by Z_i, O_i and E_i the numbers of "0" bits, "1" bits and rising edges in gate i's output over a complete cycle. Here a cycle is defined as the period over which the raw input bits directly or indirectly fed to a gate steps from all "0"s to all "1"s and then jump back to all "0"s. For any gate i in the bottom level, the values of $\langle Z_i, O_i, E_i \rangle$ is obvious:

$$\begin{pmatrix} Z_i \\ O_i \\ E_i \end{pmatrix} = \begin{cases} \begin{pmatrix} 1 \\ 3 \\ 1 \end{pmatrix} & \text{gate } i \text{ is an OR gate,} \\ \begin{pmatrix} 3 \\ 1 \\ 1 \end{pmatrix} & \text{gate } i \text{ is an AND gate.} \end{cases} \tag{C.1}$$

For any other gate i, the values of Z_i and O_i remains easy to determine, but the formula for E_i is a little bit tricky. Let's first consider an OR gate i. Observe that gate i effectively functions as a transfer gate, assuming gate $2i + 2$'s output as its own, when gate $2i + 1$ outputs "0". During those periods when gate $2i+1$'s output stays at "0", all rising edges in gate $2i + 2$'s output are transferred to gate i's output. Furthermore, in no other situation can rising edges appear in gate i's output. Consequently, we derive that for any OR gate i not at the bottom level,

$$E_i = Z_{2i+1} E_{2i+2}. \tag{C.2}$$

Similarly, we can derive that for any AND gate i not at the bottom level,

$$E_i = O_{2i+1} E_{2i+2}. \tag{C.3}$$

Now the formulæ for any gate i not at the bottom level are complete:

$$\begin{pmatrix} Z_i \\ O_i \\ E_i \end{pmatrix} = \begin{cases} \begin{pmatrix} Z_{2i+1} Z_{2i+2} \\ O_{2i+1} O_{2i+2} + O_{2i+1} Z_{2i+2} + Z_{2i+1} O_{2i+2} \\ Z_{2i+1} E_{2i+2} \end{pmatrix} & \text{gate } i \text{ is an OR gate,} \\ \begin{pmatrix} Z_{2i+1} Z_{2i+2} + O_{2i+1} Z_{2i+2} + Z_{2i+1} O_{2i+2} \\ O_{2i+1} O_{2i+2} \\ O_{2i+1} E_{2i+2} \end{pmatrix} & \text{gate } i \text{ is an AND gate.} \end{cases} \tag{C.4}$$

Problem D Distribution of Toys

Given a set of the numbers $\{1, 2, \ldots, n\}$, we desire to count all such partitions into m possibly empty subsets that the cardinality of every subset in them is at least k. We again use a dynamic programming approach.

Assume that $k \geq 1$. Denote by $T_{n,m}$ the number of valid partitions of $\{1, 2, \ldots, n\}$ into m subsets. We classify these partitions by whether the removal of n renders them invalid. The partitions invalidated by removing n must have n in a subset of cardinality exactly k. Hence, their number is $\binom{n-1}{k-1}T_{n-k,m-1}$. The remaining partitions, which retains validity regardless of the removal of n, amounts to $mT_{n-1,m}$. Adding up both parts and complemented with boundary conditions, the complete recurrence relation is:

$$T_{n,m} = \begin{cases} 1 & n = 0 \text{ and } m = 0, \\ 0 & n < mk, \\ \binom{n-1}{k-1}T_{n-k,m-1} + mT_{n-1,m} & \text{otherwise.} \end{cases} \tag{D.1}$$

The cases where $k = 0$ have to be specially dealt with. In these cases, the partitions can be arbitrary as long as they contain exactly m sets, including one or more empty sets. Their number is given by $\sum_{i=0}^{m} \left\{ {n \atop i} \right\}$, where $\left\{ {n \atop i} \right\}$ denotes a Stirling number of the second kind.

Problem E Bridge Across Islands

The technique of rotating calipers is particularly suitable for computing the distance between two non-overlapping convex polygons.

Given two convex polygons P and Q as shown in Figure E.1, we start by placing two horizontal lines touching the topmost vertex of P and the bottommost vertex of Q, respectively, as shown in Figure E.2. Next, we rotate the two lines in the counterclockwise direction, ensuring that they always touch the borders of the polygons. We claim that there exists some instance when the two lines touch two points p and q on the borders of P and Q such that the distance between them is the distance between P and Q, as shown in Figure E.3.

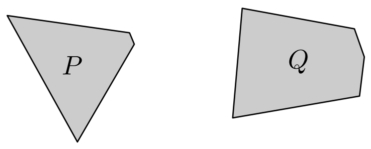

Figure E.1: Convex polygons P and Q

Problem F Reset Sequence

We perform breadth-first search over subsets of states of the given chip's control logic to find the shortest reset sequence. We define a directed graph as described below for the search procedure to proceed on.

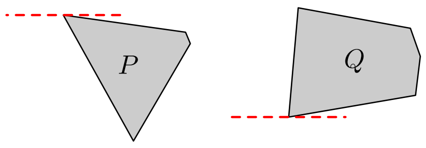

Figure E.2: Initial positions of two rotating lines

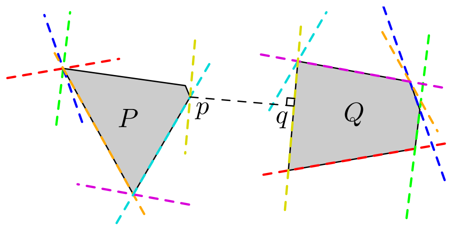

Figure E.3: Various positions of the rotating lines

Let the set of states be N. Denote by $\delta_c(x)$ the state the chip transits to when it receives command c in state x. We define a directed graph $G = (2^N, A)$, where 2^N is the power set of N. For each command c and each subset $X = \{x_1, x_2, \dots, x_k\}$ of N, we create an arc labelled with c in G which goes from X to $\{\delta_c(x) : x \in X\}$. Now we can use breadth-first search to identify the shortest path from N to $\{0\}$ in G. The concatenation of labels on the path is the shortest reset sequence.

Problem G Maclaurin Series of Reciprocals

Given a power series

$$f(x) = a_0 + a_1x + a_2x^2 + \cdots + a_{n-1}x^{n-1} + \mathcal{O}(x^n), \tag{G.1}$$

we want to determine its reciprocal also written as a power series.

Let

$$g(x) = b_0 + b_1x + a_2x^2 + \cdots + b_{n-1}x^{n-1} + \mathcal{O}(x^n) \tag{G.2}$$

be the reciprocal of $f(x)$, i.e.,

$$f(x)g(x) = 1. \tag{G.3}$$

We expand the product $f(x)g(x)$ and collect the coefficients by terms, which gives

$$\begin{aligned} f(x)g(x) &= \left[\sum_{k=0}^{n-1} a_kx^k + \mathcal{O}(x^n)\right]\left[\sum_{k=0}^{n-1} b_kx^k + \mathcal{O}(x^n)\right] \\ &= \sum_{k=0}^{n-1}\left(\sum_{i=0}^{k} a_ib_{k-i}\right)x^k + \mathcal{O}(x^n). \end{aligned} \tag{G.4}$$

Comparing the right-hand sides of Equations G.3 and G.4, we have the following linear system

$$\begin{pmatrix} a_0 & & & \\ a_1 & a_0 & & \\ \vdots & \vdots & \ddots & \\ a_{n-1} & a_{n-2} & \cdots & a_0 \end{pmatrix} \begin{pmatrix} b_0 \\ b_1 \\ \vdots \\ b_{n-1} \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix}, \tag{G.5}$$

which we can solve for $b_0, b_1, \ldots, b_{n-1}$ and thus $g(x)$.

Problem H Minimum Weighted Perfect Fractional b-Matching

We reduce the minimum weighted perfect fractional b-matching problem to the minimum cost flow problem. We illustrate the reduction using the sample test case.

Figure H.1 shows graph $G = (V, E)$ given by the sample test case. Every edge, as depicted in the general form in Figure H.2, is interpreted as connecting two vertices i and j with capacity u and weight c. The label b_i is the balance of vertex i.

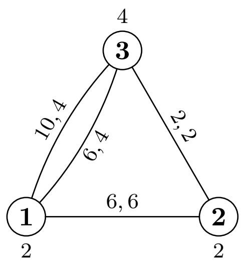

Figure H.1: Graph G in the sample test case

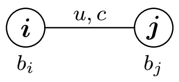

Figure H.2: Labels around an edge

We define a flow network $G' = (N, A)$ as follows. For each vertex i in G, we create two corresponding vertices i' with supply b_i and i'' with demand b_i in G'. For each edge e connecting vertices i and j with capacity u and weight c in G, we create two arcs running from vertices i' to j'' and from i'' to j', respectively, and both with capacity u and cost c. The resulting flow network G'' is depicted in Figure H.3.

Let x^* be an optimal solution to the minimum cost flow problem on G', and $x^*_{i'j''}$ denote the flow from node i' to node j'' in x^*. Observe that x^* provides a lower bound on the minimum weight perfect b-matching x on graph G. We define another flow x' on G' where for any pair of nodes $\langle i', j'' \rangle$,

$$x'_{i'j''} = \frac{x^*_{i'j''} + x^*_{j'i''}}{2}. \tag{H.1}$$

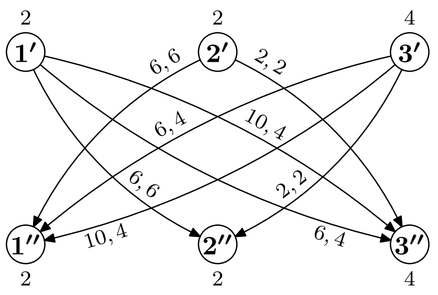

Figure H.3: Flow network G' corresponding to graph G

We can prove that x' is also an optimal flow. Consequently, if for each edge connecting vertices i and j in G, we let

$$x_{ij} = \frac{x^*_{i'j''} + x^*_{j'i''}}{2}, \tag{H.2}$$

the resulting solution x will be optimal.