Problem Analysis – Common Contest 1
ICPC Training Camp powered by Huawei

contributors: zhouyuyang, skeydec, jiry, scape, shanquan2

Guojie Luo

gluo@pku.edu.cn

Problem A: Final Exam

- allocate M minutes (in real number) to prepare for n exams
- get $f_i(t) = \max(0, \min(d_i, a_i t^2 + b_i t + c_i))$ points for t minutes in the i-th exam
- maximize the total points for the n exams
- i.e., maximize $\sum_{i=1}^{n} f_i(t_i)$ subject to $\sum_{i=1}^{n} t_i = M$ and $t_i \geq 0$

- Input value ranges
  - $1 \leq n \leq 100000, \;\; 0 < M < 10^8,$
  - $|a_i| \leq 10, \;\; |b_i| \leq 5000, \;\; 0 \leq c_i \leq d_i \leq 5000,$
  - at most 18 exams have $a_i > 0$

Problem A: Final Exam

Solution

- maximize $\sum_{i=1}^{n} f_i(t_i)$ subject to $\sum_{i=1}^{n} t_i = M$ and $t_i \geq 0$
  - $f_i(t) = \max(0, \min(d_i, a_i t^2 + b_i t + c_i))$
- Assume K is the number of exams with a_i>0
- Lemma 1: The optimal answer does not change when we change the function to $f_i(t) = c_i + \max(0, \min(d_i - c_i, a_i t^2 + b_i t))$
  - Thus, we can simply ignore ci at the beginning and add them back into the final answer at the end.
- Lemma 2: Let the ``full score'' of an exam be the tight upper bound of the points that Rikka can get, by assuming she has unlimited time. And Ti is the minimal amount of minutes that she spends to get the full score for the i-th exam. There is no need for her to spend more than Ti minutes on the i-th exam.
- Lemma 3: If she spends x minutes on the i-th exam and x>0, then fi(x)>0.
- Lemma 4: If there are only exams with ai > 0, there is always an optimal solution with at most one exam that she spends time on but does not get the full score.

Problem A: Final Exam Solution

- Lemma 4: Suppose she spends time in exam i and j, but none of them get full scores. Suppose she spends S minutes in them, and let g(x)=fi(x)+fj(S-x)=a'y^2+b'y+c, we can find a'>0。
  - So we can always take the maximum of g in two endpoints.
  - But in the hypothesis, the maximun is not taken in two endpoints, which lead to conflict.
- Lemma 5: There is always optimal solution, so that for all exams she spends time but do not get full score, then they should have a equal (2 ai ti + bi)
- Proof 5 (by contradiction)
  - Suppose she spends time in exam i and j, and 2 ai ti + bi > 2 aj tj + bj, but none of them get full scores.
  - We can always find Δ>0, so that if we change the review time to ti+ Δ and t_j-Δ $; both of them spend time, but none of them get full score.
  - We can found that the difference between them is (2 ai ti + bi – 2 aj tj – bj) Δ + (ai+aj)/2 $Δ^2$.
  - Because we have 2 ai ti + bi – 2 aj tj – bj > 0, we can always find 0 < ε < Δ, which makes (2 ai ti + bi – 2 aj tj – bj) ε + (ai+aj)/2 $Δ^2$ > 0.
  - So we can always adjust to make the answer better, which lead to conflict.
- Corollary 5.1: For all exams she get full score, it will always have a 2 ai ti + bi greater than those who doesn't get full score.
- Corollary 5.2: For all exams she doesn’t spend time, it will always have a 2 ai ti + bi greater than those who spends time.

Problem A: Final Exam Solution

- Note that for any optimal solution, there may be two exams i,j, which satisfy ai>0, aj<0, and they all took time and didn't get full scores. For example:
  - 2 4
  - 1 0 0 1000
  - -2 12 0 1000
- You can easily find that you can get the highest score 20 if you spend 2 minutes in both exams, and there is no other ways to get at least 20.
- Because we have K ≤ 18, we can enumerate the subset with ai>0 and full scores.
- If she didn‘t spend time in any other exams with ai>0, we can obtain the optimal solution if she can only spend time in the exams with ai ≤ 0. According to Lemma 5, we can show that the optimal solution can be regarded as a piecewise quadratic function with no more than 2n segments. Simply look up the table and we can use binary search to solve it.
- In another case, we can always find exactly one exam with ai>0, where she spends time on but does not get the full score. So we can firstly select the only exam, and update the answer. After enumerating the subset with ai>0 and full scores, we can get 2^{K-1} quadratic function f(x) = ai x^2 + bi x + ci, where they always have the same ai.

Problem A: Final Exam Solution

- Lemma 6: If there are n quadratic functions f1(x),f2(x),…,fn(x) with definition ***R***, then max{f1(x),f2(x),…,fn(x)} can be regarded as a piecewise quadratic function with no more than 2n segments.
- Proof 6
  - suggest max{f1(x),f2(x),…,fn(x)} has m segments, and in each segments [li,ri] we have max{f1(x),f2(x),…,fn(x)}=f {id_i}(x). Also we can ensure that the id_i in adjacent segments is different, otherwise we can simple merge them. So we can generate a sequence (may be infinity) according to she segment.
  - Notice that we don't have a subsequence a b a b in our sequence. So if we found a subsequence like a … b a, we know that b will not occur in the following sequence anymore.
  - So if we have 2n-1 sequences, we can find at least n-1 pairs with different right endpoint, which ensures that we can use only one id in the following sequence. So the length of the sequence is always shorter than 2n.

Problem A: Final Exam

Solution

- Lemma 7: If there are n quadratic functions f1(x), f2(x),…,fn(x) where the definitions of them are continuous subintervals of ***R***, then max{f1(x),f2(x),…,fn(x)} can be regarded as a piecewise quadratic function with no more than 4n segments.
- Proof 7
  - Similarly, we also use the sequence of index id to proof it. We don't have a subsequence a b a c a b in our sequence. So if we found an index a occurs 3 times in sequence, then we can always find a subsequence like a … b a … c a, we know that b will not occur in the following sequence anymore
  - We can add a 0 before the sequence. If we found an index a such that there are 3 pairs of adjacent elements in the form of x a we can always find an index b and exclude it in the following sequence.
  - Because b occurs at most twice, we can simply erase them in the sequence, and then merge the adjacent same elements. The length of the sequence decreases at most 4. So after 4n operations, we ensure that we can use only one id in the following sequence. So the length of the sequence is always shorter than 4n.
  - Because of this, we can use divide and conquer to solve it and merge them using the method like mergesort. And the time complexity of a single query is O(K 2^K+n).
- We have to solve K different problems, so we can solve the whole problem in O(K^2 2^K + nK +n log n)

Problem B: Travel around China

- Cities and expressways form a $n \times m$ grid graph with node costs
- Compute $\sum_{p=1}^{n} \sum_{x=1}^{n} \sum_{q=1}^{m} \sum_{y=1}^{m} [(p, q) \neq (x, y)] \cdot \text{distance}\big((p, q), (x, y)\big)$
- Input value ranges
  - $n = 3$ and $1 \leq m \leq 1.5 \times 10^5$
  - node cost $1 \leq a_{i,j} \leq 10^9$

Problem B: Travel around China — Solution

- Divide-and-conquer can solve a simpler problem when n=2
  - The shortest path will stay inside the bounding box of $[1, L]\times[2, R]$
    - $L = \min\{q, y\}$ and $R = \max\{q, y\}$ at the beginning
  - Use column "mid" to divide the problem, and merge using paths via (1,mid) or (2,mid)
  - Time complexity $O(m \log^2 m)$
- Can we extend the idea for the case n=3?
  - The sample input tells us a detour may be needed
- Divide-and-conquer-like approach for n=3
  - Not necessarily stay in the bounding box $[1,L]\times[3,R]$
  - Compute dist((1,L), (3,L)) and dist((1,R), (3,R)) using shortest path algorithm for the detours
  - Use column "mid" to divide the problem, and merge using (1,mid), (2,mid), and (3,mid)
  - Time complexity $O(m \log^2 m)$

Problem C: Wandering

Statement

- starting from $(0,0)$, walk in n random steps
  - each step walks by (x_i, y_i), uniformly drawn inside a circle $\{(x,y) \mid x^2 + y^2 \leq R_i^2\}$
  - after n steps, move to location $(\sum_{i=1}^{n} x_i, \sum_{i=1}^{n} y_i)$
- compute the expectation $E[(\sum_{i=1}^{n} x_i)^2 + (\sum_{i=1}^{n} y_i)^2]$
  - square of the expected distance to the origin

Problem C: Wandering

- Random variables (x_i, y_i) for the n steps $(1 \leq i \leq n)$
- $E[(\sum_{i=1}^{n} x_i)^2 + (\sum_{i=1}^{n} y_i)^2] = \sum_{i=1}^{n} \sum_{j=1}^{n} (E(x_i x_j) + E(y_i y_j))$
- Because $E[x_i] = E[y_i] = 0$ and (x_i, y_i) and (x_j, y_j) are independent for $i \neq j$
- $E[(\sum_{i=1}^{n} x_i)^2 + (\sum_{i=1}^{n} y_i)^2] = \sum_{i=1}^{n} E(x_i^2 + y_i^2) = \frac{\int_0^{R_i} 2\pi r \cdot r^2 dr}{\int_0^{R_i} 2\pi r dr} = \frac{R_i^2}{2}$

Problem D: Insects

Statement

- n insects in the field
  - each has a type and a level
- Initially every insect has a “seed” buff
- Player can eliminate an insect (and possibly add a new insect)
  - If the eliminated insect has the “seed” buff
    - player can add a new insect with arbitrary type and level L
    - L is the highest level among the remaining insects with the same type as the eliminated one
    - If no remaining insects has the same type, no new insect can be added
    - new insect does not have the “seed” buff
- Compute the maximum sum of levels for all remaining insects in the field by eliminating at most K insects

Problem D: Insects

Solution

- Lemma: in the optimal case,
  - 1) every elimination adds a new insect
  - 2) every new insect has the same level L
- Proof sketch
  - 1) we can always skip the elimination, if no new insect can be added
  - 2) for any optimal case, we can always construct another one, such that all new insects have level L
    - i) if a new insect with the highest level L is added at the k-th step, we can always add a level-L insect at the 1-st step
    - ii) for the remaining elimination, we set the type of the previously added insect as the type of the one to be eliminated
- Algorithm
  - Enumerate the type (out of n types) for the elimination at the first step
  - When we add k new insects, the contribution to the sum of levels is a linear function of k
  - Use some data structure to find the optimal sequence for eliminating at most 1, 2, …, K insects in $O(\log^2 n)$
  - Overall time complexity $O(n \log^2 n)$

Problem E: Minimum Spanning Tree Statement

- A graph has n nodes and m edges
  - The edge weights are a permutation of {1, 2, …, m}
- The minimum spanning tree consist of the first n-1 edges
- Find the number of edge weight permutations satisfy the condition above
- Input value ranges
  - $2 \le n \le 20$
  - $n-1 \le m \le 100$

Problem E: Minimum Spanning Tree Solution

- A combinatorial counting problem
- Count smartly
  - ► Enumerate the (n-1)! relative order for the first n-1 edges
    - Instead of enumerating C(m,n-1) configurations
  - ► Using the cycle property to count the cases of the non-tree edges
    - For any cycle in the graph, if an edge weight is larger than the others, then this edge cannot belong to an MST.
    - For each non-tree edge it must larger than the largest edge on the tree path.
    - We can count it by using some combinatorial method.
    - Then we can optimize enumeration of tree edges DP.
- Time complexity $O(m\ 2^n)$

Problem F: Flow

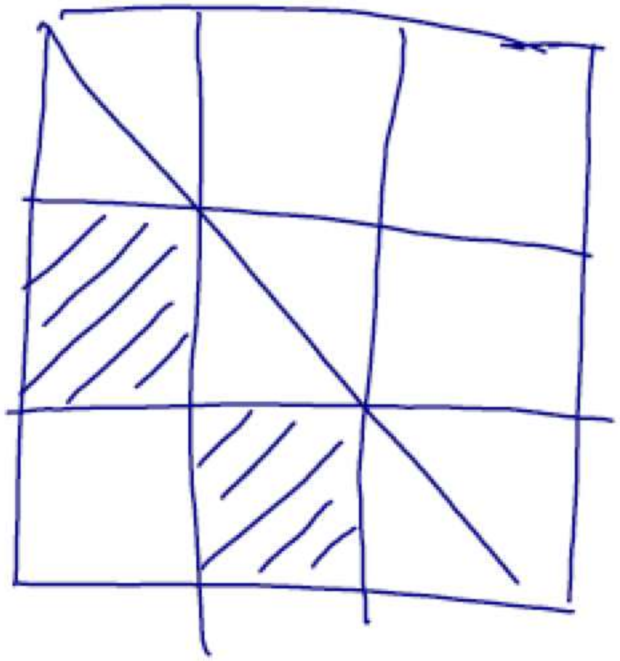
Statement

- Given an undirected graph with edge weights
- For each vertex i, add an edge (i, i%n+1, INF) to the graph
- Let c(i,j) be the minimal cut between vertex i and j
- Calculate the sum of c(i,j) for all pairs of vertices

- n,m <= 20000

Problem F: Flow — Solution 1/2

- Using Gomory-Hu tree
  - Capable of computing all-pair mincut in $O(n)$ online queries of $c(i,j)$; instead of $O(n^2)$ queries
- How to quickly calculate $c(i,j)$?
- A special graph
  - The newly added ∞ edges (with a huge weight 10^9) form a cycle
  - Every mincut must cut the cycle into two halves
  - Exactly two ∞ edges in $c(i,j)$ are cut
- Let $A[i,j]$ be the cost for cutting range $[i,j]$ from the rest
  - Each edge contributes to two sub-rectangles in A
  - Calculating $c(i,j)$ = sub-rectangle minimal query in A



Problem F: Flow

Solution 2/2

- An converted problem
  - A n×n array with initial value 0
  - **Firstly** add sub-rectangles
  - **Then** online query minimal sub-rectangles online
- divide-and-conquer on the x-axis + segment tree to maintain the y axis (scanning line)
  - Operation: range add, query on the range maximal and historic maximal
  - Time: $O(n\log^2 n)$
  - Memory: $O(n\log^2 n)$

Problem G: Revenue Statement

- Seller sets a pricing $p = (p_1, p_2, \cdots, p_n)$
- Buyer has a valuation profile $v = (v_1, v_2, \cdots, v_n)$
  - The utility of buying the i-th item is $v_i - p_i$
  - Buyer will purchase a single item with the maximal **positive utility**; or buys nothing
  - If there is a tie, Buyer will buy the one with the lowest price
- Seller knows v is a random vector
  - v_i follows a known marginal distribution F_i
    - v_i is one of the k values: $v_i^{(1)}, v_i^{(2)}, \ldots, v_i^{(k)}$
    - $\Pr\left(v_i = v_i^{(j)}\right) = p_i^{(j)}$ such that $\sum_{j=1}^{k} p_i^{(j)} = 1$
  - But the correlation among $\{v_i\}$ and the joint distribution F are **unknown**
- Help Seller to estimate the minimal expected revenue

Problem G: Revenue Solution 1/2

- Basic idea
  - ▶ The joint distribution F of $\{v_i\}$ is unknown
  - ▶ and we need to estimate the minimal expected revenue

  ⇒ construct a "worst-case" joint distribution

  The valuations of all items are "perfectly coupled"

  i.e., a single q~U[0,1] random variable determines the whole valuation profile

  construct a correlation to minimize the revenue

Problem G: Revenue Solution 2/2

- Sort all entities of $\{(v_i^{(j)}, p_i^{(j)})\}$ with descending utilities
- For any prefix length K of this sequence
  - Insert the K-th entity in a min-price heap
  - Let $S_{i,K}$ be the sum of probability of the i-th item at length K
  - Let T_K be $\max_i\{S_{i,K}\}$ at length K
  - consume $T_k – T_{k-1}$ probability of the heap top; and pop it when its $p_i^{(j)}$ becomes zero
    - It is guaranteed that we can construct a legal correlation with this item having the maximal utility at the current case
  - Process until $T_k=1$

Problem H: Segment Statement

- An array A of length n
- Query: find the longest “loose” segment S in A, such that
  - min(S) + max(S) > len(S)
- Operation: change the array in m turns
  - In each turn, k pairs of elements are swapped
  - Perform query on the resulting A at the end of each turn

- Require: Linear time
  - $1 \leq n \leq 10^6$
  - $1 \leq m \leq 30$
  - $1 \leq k \leq 10^6$

Problem H: Segment Solution

- Cartesian Tree
  - Capable of maintaining a list of (index, weight)
  - An in-order traversal gives the node list with ordered indices
  - The weight satisfy the heap property (with min-weight at the root)
- For each subtree, maintain
  - T.len = length of the longest loose segment
  - T.size = size of the subtree
  - T.max = maximal value in this subtree
- DP on the Cartesian tree
  - Merge the information of two lists A,B into the information of A+[a]+B
    - a is guaranteed to be smaller than min(min A, min B)
  - Suppose A.max < B.max
    - The opposite can be derived similarly
  - If (a+B.max–1)>B.len, ans = max(A.ans, 1+B.len+min(a+B.max-1-(B.len+1), A.len))
  - Else ans = max(A.ans, B.ans)
    - Proof omitted

Problem I: Color

Statement

- Given a complete graph with n vertices
  - Each edge has one of the m colors
  - No two adjacent edges have the same color
- Check whether we can extend it to a complete graph with $m + 1$ edges
- Input value ranges
  - $1 \leq n \leq 200$
  - $1 \leq m \leq 200$
  - $n \geq m + 1$

Problem I: Color

Solution

- Easy cases
  - Initial graph contains two adjacent edges with the same color: “NO”
  - m is an even number, i.e., (m+1) is an odd number: “NO”
- Now we have m family of sets (defined by the coloring)
  - Each family contains (m+1)/2 sets and each set contains 0, 1 or 2 vertices (denoted as 0-set, 1-set, 2-set respectively).
  - With the first m points given, we can add 1, …, n into the sets.
  - If here some family contains greater than (m+1)/2 sets, the answer is “NO”
  - Otherwise we will show that the answer is "YES".
- To show this, it suffices to show that we can add n+1.
  - Consider that we already have n(n-1)/2 “2-sets” and n(m+1-n) “1-sets”.
  - Construct a network flow as follows (next slide)

Problem I: Color Solution

- Construct a network flow as follows
  - The graph is a bipartite graph with a source S and a sink T.
  - The points on the left represents the families and the points on the right represents 0-set and each 1-set we've already have.
  - We give each family A an edge (S,A) with capacity 1, give each 1-set {i} an edge (i,T) with capacity 1 and give the 0-set an edge (0,T) with capacity n-m.
  - If family A contains an 1-set or a 0-set a, draw an edge (A,a) with capacity 1.
  - We can construct a maximal flow by given each edge in A equal flow.
  - Thus, we can have an integer maximal flow.
  - Then we can extend the graph to m+1 points.

Problem J: Horses

Statement

- m horses in a queue a (we call a queue "string")
- n subspecies of horses (we call a horse "character")
  - Some are friend subspecies
  - Two adjacent horses belong to friend subspecies may swap in the queue
- The "equivalence" (we use the symbol ~) of strings
  - one can be generated from the other by legal swapping
- Find the "minimal-commuter" b of string a
  - (commuter) ba ~ ab
  - (minimal) b=cd does not exist, such that ca ~ ac and da ~ ad
  - (minimal) b is lexicographically the least

Problem J: Horses

Solution

- Two kinds of minimal commuters of a
  - Case 1: Character that can swap with all characters in a
  - Case 2: Ignore characters not in a, the string consists of a single connected component in the complement of the “friend” graph.
- We may assume the characters in a are connected in the complement graph.
  - Then we can show that if a and b commutes, then the shorter one can be a prefix of the longer one. By induction we have a=c^i and b=c^j. Here in case 2 we only need to calculate minimal c.
  - Cyclicly move the biggest character to the front and calculate the minimal representation by toposort.
  - Cyclicly move the biggest character to the front again and calculate the minimal representation by toposort.
  - Then use KMP to calculate the minimal period.
  - At last, move c' cyclicly back to get c.

Thank you!