培正喇沙 編程挑戰賽
LA SALLE – PUI CHING PROGRAMMING CHALLENGE

第五屆 · 2021

2021 年 8 月 8 日 (星期日)

香港培正中學

題解

題號		名稱	作者	編製	難易度
A		Allowance Exhaustion	葉景輝	葉景輝	★★★★★
B		Boat Assignment	鍾懷哲	鍾懷哲	★★★★☆
C		Copy of the String	吳有孚	吳有孚	★★☆☆☆
D		Double Queue	袁樂勤	袁樂勤	★★☆☆☆
E		Escape the Cube	李呈禧	李呈禧	★★☆☆☆
F		Furthest Travel	黃梓駿	黃梓駿	★★★★★★
G		Gold Medal Bout	招朗軒	鄭希哲	★★★☆☆
H		How to Get Rice	李卓傑	招朗軒	★★★☆☆
I		Inno Per Gli Sconfitti	黃敏恆	黃敏恆	★☆☆☆☆
J		Just Skip It	袁樂勤	衛家熙	★★★☆☆
K		Kario Mart	李卓傑	李卓傑	★★★★★★
L		Lockout	李啟峰	李啟峰	★★★☆☆

難易度是指作者團隊認為能在比賽中解決該題的隊伍比例

由 1 星至 6 星為: >75%, 50-75%, 25-50%, 5-25%, 1-5%, <1%

Assume there exists a heaviest cycle of length T.

Let $e(v, w)$ be the heaviest edge in the cycle.

If the cycle has visited any node u (excluding v and w) more than 3 times, it must be in the following form:

- $s \to \cdots \to u$
- 3 or more smaller cycles $u \to \cdots \to u$, one after another
- $u \to \cdots \to s$

This can be refactored into:

- $s \to \cdots \to u$
- (0 or more) even-length smaller cycles $u \to \cdots \to u$, one after another
- (0 or more) odd-length smaller cycles $u \to \cdots \to u$, one after another
- $u \to \cdots \to s$

We can always "collapse" even-length smaller cycles / pairs of odd-length smaller cycles by removing them from the big cycle and reallocating the "edge quota" to traverse e back and forth. This creates another cycle that is no worse than the original one.

(Considering only the direction $v \to w$, WLOG)

If e is in $s \to \cdots \to u$ (analogous to $u \to \cdots \to s$), then the collapsed cycle will be:

- $s \to \cdots \to v \to w \to v \to w \to \cdots \to v \to w \to \cdots \to u$
- At most 1 odd-length smaller cycle
- $u \to \cdots \to s$

If e is in an even-length smaller cycle, the collapsed cycle will be:

- $s \to \cdots \to u$
- 1 even-length smaller cycle: $u \to \cdots \to v \to w \to v \to w \to \cdots \to v \to w \to \cdots \to u$
- At most 1 odd-length smaller cycle
- $u \to \cdots \to s$

If e is in an odd-length smaller cycle, the collapsed cycle will be:

- $s \to \cdots \to u$
- 1 odd-length smaller cycle: $u \to \cdots \to v \to w \to v \to w \to \cdots \to v \to w \to \cdots \to u$
- At most 1 odd-length smaller cycle
- $u \to \cdots \to s$

In both cases, u appears ≤ 3 times in the collapsed cycle.

We can thus infer that, for any heaviest cycle that has visited any node (except for v and w) more than 3 times, we can always construct a cycle that is no worse than the mentioned cycle that has only visited the node ≤ 3 times. We call this operation "reduction".

Whenever the heaviest cycle visits a node (excluding v and w) more than 3 times, we can always apply reduction to form another cycle with the node appearing at most 3 times. Therefore, a heaviest cycle with every node (excluding v and w) appearing at most 3 times must exist:

$$s \to \cdots \to v \to w \to v \to w \to \cdots \to v \to w \to \cdots \to s$$

Whereas the parts $s \to \cdots \to v$ and $w \to \cdots \to s$ visits each node (excluding v and w) at most 3 times.

Note that if both $v \to w$ and $w \to v$ appears elsewhere in the cycle, the cycle can be reformed and these two edges can be attached back to central sequence of $v \to w \to v \to w \ldots \to v \to w$. The proof is left as an exercise for the reader.

Hence, there is at most one $v \to w$ or $w \to v$ that we have to account for in the parts $s \to \cdots \to v$ and $w \to \cdots \to s$.

Therefore, by exhausting all cycles satisfying the above property, we get to know whether there is such a cycle, or if there is, the weight of the heaviest cycle.

Solution:

$$path[u][e] = \max\{\text{weight of path} \mid \text{path starts at s} \wedge \text{ends at u, using exactly e edges}\}$$

For each edge, we assume the edge to be edge (v, w) described above, and use $w[u][e]$ to construct the answer. The dimensions of $path$ only need to be at most $[|V|][|V| \cdot 3 + 2]$ from the proof above.

Answer $= \max_{e(v,w)} \left\{ \max_{0 \leq i,j \leq |V| \cdot 3+2} \{path[v][i] + weight_{v,w} \cdot (T - i - j) + path[w][j]\} \right\}$

Where $i + j < T$

Which can be optimised to:

$$\max_{e(v,w)} \left\{ \max_{0 \leq i,j \leq |V| \cdot 3+2} \left\{ \left(path[v][i] + weight_{v,w} \cdot (T - i)\right) + \left(path[w][j] - weight_{v,w} \cdot j\right) \right\} \right\}$$

Then to:

$$\max_{e(v,w)} \left\{ \max_{0 \leq i \leq |V| \cdot 3+2} \left\{ path[v][i] + weight_{v,w} \cdot (T - i) + \max_{0 \leq j < T-i} \left\{ \left(path[w][j] - weight_{v,w} \cdot j\right) \right\} \right\} \right\}$$

Since $\max_{0 \leq j < T-i} \left\{ \left(path[w][j] - weight_{v,w} \cdot j\right) \right\}$ is independent of i, we can precompute it.

Overall time complexity: $O(VE + VE) = O(VE)$

B - Boat Arrangement

Firstly, we can observe that in order to use the minimum number of boats, we should sort the boats from largest to smallest in terms of maximum load: let's assume there is a set of boats S{ A[i] , ... , A[j] } which is the current optimal arrangement under the given condition. if A[x] > A[i], it is always better to replace A[i] with A[x], so the greedy algorithm works.

Secondly, we can come up with a slow solution in which we try all permutations of the animals and try to assign them to boats one by one, and the exhaustion will give us the minimum number of boats required. Observe that N is small and we can optimize the permutation exhaustion with bitmask Dynamic Programming, where each state contains both the {minimum number of boats required, minimum weight in the last boat}, then the answer would be in the dp[1 << (N-1)] if the arrangement is possible under the given condition.

Bonus

Try to solve the question with flow and see how large the constraints can be :))

C - Copy of the String

Firstly, let's calculate the coins we need if we are not using the first operation. It is actually very simple — just sum up the difference of each character in the strings. More formally, for the string S[1...n] and T[1...n], we need to calculate the sum of abs(s[i] - t[i]), where 1 <= i <= n.

Secondly, we can observe that the first operation will be used at most once, since we can already obtain any permutation with it. Therefore, we can actually sort both string S and T if we are using the first operation. After sorting the strings, we can calculate the coins we need the same way as the time we do not sort them. We can guarantee calculating the minimum possible coins, since the smallest character in string S is matching the smallest character in string T, all the way till the largest. The sum of the differences will be minimal, and we cannot do any better by swapping any characters.

Finally, we just need to compare the two possible cases and the smaller one is our answer.

Comments

This task is rated by the author as 1-star difficulty at proposal stage, and increased to 2-star after discussions. In the actual contest, most of the teams managed to solve it early in the contest, which met the author's expectation.

D - Double Queue

The crucial observation for the task is (although very intuitive), to achieve the minimum time, there are only 2 possible ways:

1. Queue and buy at Alice's shop first, then queue and buy at Bob's shop
2. Queue and buy at Bob's shop first, then queue and buy at Alice's shop

If Dan was queuing at Alice's shop initially, but before buying, he left to buy from Bob's shop first, then the overall time used would be not less than queuing at Bob's shop initially (i.e. the 2nd way).

The only thing left is to calculate the time used by both ways, and that can be done by simple simulation. Suppose Dan queued at Alice's shop initially, he would arrive Bob's shop at time (Qa + 1) * Sa + W. So, only B[i] <= (Qa + 1) * Sa + W needed to be considered.

Maintain a pointer denoting the time after serving each B[i]. The pointer is at Qb * Sb at first. For each B[i], update the pointer to max(pointer + Sb, B[i] + Sb). Update the pointer once more with the time Dan enters the queue.

Calculate the time if Dan queues at Bob's shop first as well, and output the minimum of the 2 ways.

Comments

This task is rated by the author as 1-star difficulty at proposal stage, and increased to 2-star after preparation stage. In the actual contest, a majority of the teams encountered much more difficulties than expected.

It is suggested to come up with your own test cases to test your solution because the given samples are often misleading. Coming up with strong and inspiring test cases is a useful skill to get a deeper understanding of tasks.

E – Escape the cube

While the problem statement seems complicated (it's intentional), it is easy to see that the possible coordinates of a particular room actually rotate among a few values. Instead of simulating the whole thing, we can find out the permutation vector of the coordinates of the room and output the corresponding coordinates after taking mod of the number of moves.

The proof of why it rotates among a few values is trivial and is left to readers as an exercise. Even if you do not prove it, it should be easy to see the pattern during the contest by simulation.

Note: the author personally thinks the Cube is quite an interesting movie. You can try it out if you are into science fiction horror movies :)

F - Furthest Travel

Observation

We first find the diameter of the given tree. If the diameter consists of even number of edges (i.e. odd number of vertices), call the centre vertex as *critical point*. Otherwise (i.e. even number of vertices), call the centre edge *critical point*. We root the tree with this critical point. (Note: we can see that even if multiple diameters exist, the critical point is still unique.)

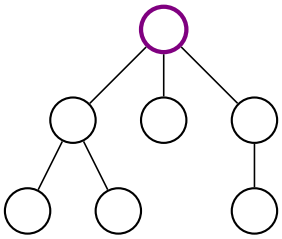


Figure 1: A vertex being the root

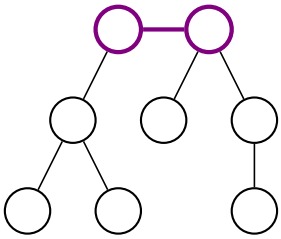


Figure 2: An edge being the "root"

We can observe that for every travel (from r_i to r_{i+1}), the path must consist of the root (critical point). Furthermore, the path must be from one subtree of a child of the root, to the other subtree. This can be proven: as without visiting the root, the longest path must be shorter than visiting the root and travel to some another subtree's deepest leaf.

Solution

As we now know that we must traverse between subtrees (of children of the root) per travel, we can now treat each of these subtrees as an unit. Denote D as the number of such subtrees (in both of the examples, $D = 3$):

We can have a vector v of $1 \times D$, where v_i states the number of possible ways that after the latest travel, the drone is located at some vertex in the i-th subtree.

Also create a $D \times D$ transition matrix M, where M_{ij} denotes the multiplication factor: how many ways to travel from some vertices in subtree i to some vertices (must be deepest leaves) in subtree j within a day. We can see that under this setting, M_ij is the number of deepest (bottomost) leaves in the subtree j. In both of the shown examples, the number of such leaves are 2, 0, 1 respectively from left to right.

Then, if we have the v of some day i, we can compute the vector v' of the day $i + 1$ by $v' = vM$. If we have the initial v (day 0), we can find the final answer by computing vM^T, where T is the given parameter in the statement.

Initializing v is a bit tricky, since the root is not under any of the subtrees. One possible hacky way of doing so is to create an additional "subtree" that represents the root vertice(s). Then the initial v of figure 1 would be $\begin{bmatrix}3 & 1 & 2 & 1\end{bmatrix}$ and that of figure 2 would be $\begin{bmatrix}3 & 1 & 2 & 2\end{bmatrix}$ (subtrees from left to right, then the root). For the transition matrix, we can still follow the definition mentioned. For instance, the matrix M for both examples would be $\begin{bmatrix}0 & 0 & 1 & 0\\ 2 & 0 & 1 & 0\\ 2 & 0 & 0 & 0\\ 0 & 0 & 0 & 0\end{bmatrix}$.

We can use fast exponent algorithm for calculating M^T, with time complexity of $O(D^3 \log T)$. Notice that the value of D is still $O(N)$ in the worst case, so such algorithm is insufficient to get an Accepted.

To optimize the algorithm, we can group the subtrees with equal number of deepest leaves. The number of distinct groups, and also the dimension M, would reduce to $O(\sqrt{N})$ as $1 + 2 + 3 + \cdots + \lceil\sqrt{N}\rceil \approx N/2$. This makes the value of D be $O(\sqrt{N})$, thus the time complexity would be $O(N^{1.5} \log T)$.

G - Gold Medal Bout

Consider a simpler case: Assume there is no **deuce** (points tied at 10-10), so that all matches are played to exactly 11 points. Then the match result must be (assume Player 1 wins):

- First ≤ 3 games: $y_i : 11$ $0 \leq y_i \leq 9$
- Next 4 games: $11 : x_i$ $0 \leq x_i \leq 9$

Hence we can exhaust all 1.3×10^7 possible cases (or 2×10^5 with slight optimization).

Deuce can then also be easily handled: since games with 10-10 ties must end with $11+k : 9+k$ (or vice versa), for each possible case, we check

- If the remaining points (P_1 and P_2 minus the seven games) are equal; and
- If one of the games is of 9 : 11 or 11 : 9

Comments

In the contest, teams generally approach this question using a constructive manner, which is expected, and 3 teams managed to solve it. There are many case handling involved. One way to construct is to first fix the number of games Player 1 lost (i.e. 4 - n); another is to observe that only at most 1 deuce is required (why?).

P.S. Thanks to Lee Wai Sze, the statement had to be updated 2 hours before the start of the contest. Congratulations!

> Today is the last day of the Tokyo 2020 Summer Olympic Games, with the closing ceremony going to take place tonight at 7PM HKT. In the last 16 days, Hong Kong Team has been continuously making history with by-far the best result of 1 Gold, 2 Silver and 2 Bronze Medals (as of 5 August 18:40), bringing pride to all Hong Kong citizens. Such credit also belongs to all other Hong Kong athletes, even without medals, since they too have paid numerous effort and fought to their best.

H - How to Get Rice

To begin with, starting at Cell 1 with $R = K = 1$ is equivalent to starting at Cell 0 with $R = K = 0$, since at Cell 0 only Operation 1 is valid. For the sake of the following representation, let's treat as if we start at Cell 0.

We try to represent the sequence of operations using a sequence of 1's and *K*'s, for instance, [1, 1, 2, 1] for Sample 1 ($N = 5$) and [1, 1, 1, 3] for Sample 2 ($N = 6$). Using this representation, the sum of the sequence must always be equal to *N*. This problem is therefore a variation of "maximizing product with sum constraint". Similar arguments can be applied here to show that *K* should not be ≥ 4, details are left as exercise.

Therefore, the optimal sequence of operations for $N \geq 5$ must be one of the following two formats: (commas omitted)

A. 112...213...3(1)
B. 112...2 for even *N*

We can show that there should be at most one 1's at the end of sequence A, otherwise putting more 2's and 3's would be more optimal. We can also show that there should be at most one 2's in Sequence A, since:

- If there are $\geq$ three 2's, we can replace [...2221...] with [...133...]
- If there are two 2's, we can replace [11221...] with [1113...1]

With the above observation, we can generalize Sequence A into groups of (N mod 3). Since *N* can be very large, modular exponentiation ("big mod") should be used here.

The only cases where Sequence B is better than Sequence A are $N = 8$, 10, and 16, which teams commonly overlook on their first attempt.

Comments

It is hard to come up with a complete proof during the contest, so a common advice is to generate the first few cases (e.g. via recursion or DP for this problem) and observe the patterns before coming up with a full solution.

I - Inno Per Gli Sconfitti

The 3 favourite schools need to have strictly more Likes than the other schools.

Therefore, the first step is to find out the number of Likes the 3 schools have.
For example, we can store them into 3 variables p, q, r.

Next, find the school with the most Likes from the **remaining** schools.
Let it be m.
This can be done in many ways:

- Set the number of Likes of the selected schools to -1. Then find the maximum in the whole array.
- Remove the schools from the array. Then find the maximum in the whole array.
- Loop over the array but ignore the indices that equals x, y or z.

Finally, for each favourite school, no Likes are needed if it is already greater than the maximum number of Likes from the remaining schools.
The answer is max(0, m - x + 1) + max(0, m - y + 1) + max(0, m - z + 1)

J - Just Skip It

For each type of advertisement, if choosing to refresh one time can reduce the expected time, we will choose to refresh the page once this type of advertisement appears. This is because after refreshing the page, an independent event happens and the same thing (i.e. choose to refresh one time) can be done to further reduce the expected time. Finally, we can observe that we just need to choose whether to refresh for each type of advertisement.

Since there are only two choices for each type of advertisement, the number of choices for this problem is only 2^4. We can exhaust each choice and calculate the corresponding expected time. For each choice, we can calculate the expected time in the following way:

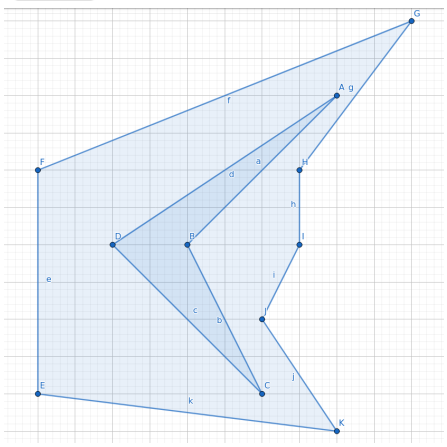
- Let N and M be the number of types of advertisements we choose to refresh and not to refresh respectively.
- Let $p_{1,i}$ and $t_{1,i}$ ($1 <= i <= N$) be the probability and the length of the i-th type of advertisement we choose to refresh respectively.
- Let $p_{0,i}$ and $t_{0,i}$ ($1 <= i <= M$) be the probability and the length of the i-th type of advertisement we choose not to refresh respectively.
- The expected time is equal to $p_{0,1} * t_{0,1} + \ldots + p_{0,M} * t_{0,M} + (p_{1,1} + \ldots + p_{1,N}) * (1 + p_{0,1} * t_{0,1} + \ldots + p_{0,M} * t_{0,M} + (p_{1,1} + \ldots + p_{1,N}) \times (1 + ...)) = (p_{0,1} * t_{0,1} + \ldots + p_{0,M} * t_{0,M} + 1) / (1 - (p_{1,1} + \ldots + p_{1,N})) - 1$

At last, we just need to find the minimum expected time among all choices.

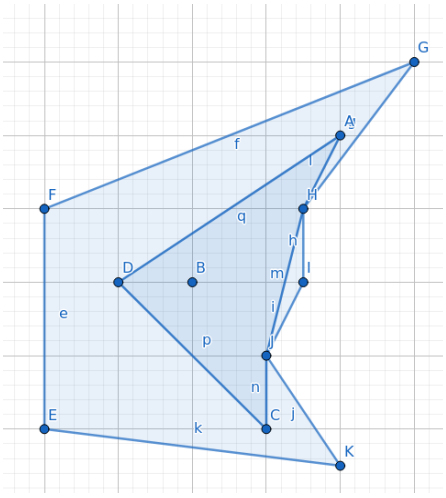
K - Solution

Consider a simplified scenario where there is no outer polygon. The answer is intuitively the perimeter of the convex hull of the inner polygon.

The solution to the general case is similar. For each anticlockwise turn of the inner polygon, consider all points on the outer polygon enclosed by the triangle formed by the three preceding points. A shortcut can be obtained by a convex hull algorithm on the enclosed points. We can then replace the turn with the shortcut obtained and repeat the process until there is no further shortcut.



For example, consider the turn A->B->C, the triangle formed by A, B, C encloses points H, I, J. By a convex hull algorithm, we obtain a shortcut A->H->J->B. We then replace B on the inner polygon with vertices H, J, obtaining a new “inner polygon” AHJCD. There is no additional shortcut. The answer is hence the perimeter of AHJCD.



The complexity of the algorithm depends on the implementation and choice of the convex hull algorithm. However, as the values of n and m are small, implementations of Õ(n^3) should pass well under the time limit.

Aside from the solution above, solutions using shortest paths algorithms on the visibility graph are also correct. While sub-cubic visibility graph algorithms exist, a naive cubic algorithm is sufficient for this task. Remember to make sure the shortest path computed encloses the inner polygon.

L – Lockout

Solution

If neither of the contestants managed to claim 9 tasks, it is impossible to tell whether the match ended in a win or a tie. We will choose to output a tieing configuration to save us from worrying about whether a line is formed before the last submission. Since there exists a valid configuration in which one contestant claims 9 tasks without anyone ever forming a line, we can simply use it (with a slight modification) for all cases. One such configuration is illustrated below:

1	1	2	2
2	2	1	1
1	1	2	2
2	2	1	1

where one cell should be assigned to the other contestant if someone claimed 9 tasks. Unclaimed tasks may be assigned arbitrarily.

Comments

The solution described above (i.e. assume no one ever forms a line) was by far the most popular during the contest. A fair number of teams constructed grids that always result in a line being formed. This is a task in which coming up with an efficient way to write the program is as important as finding a valid construction. A small number of teams dealt with every possible number of claimed tasks separately, thus taking more time and many more lines of code. An easy way to test or debug your program is to replace the task IDs with numbers 1 to 16.

The author wanted to propose a constructive algorithm task that would be solved by most, if not all, teams which achieved a Bronze/Merit or above. In the end, all but one such teams managed to solve this task during the contest.

For those who are familiar with Lockout, the change to a 4×4 grid (from 3×3) was necessary to prevent a naive exhaustion on all possible grids from passing. If you have never heard of Lockout and are intrigued by this task, you may want to watch the remaining matches in this year's Joint School OI Lockout Contest (click here for the JSOI Lockout Youtube channel) or even participate in next year's competition! (There is also a JSOI Discord server – please reach out to one of the existing members if you would like to join.)