Diamonds – Analysis

Problem: Cheng Zhong, Analysis: Iliyan Yordanov

We are given a directed weighted graph. The condition that each vertex has at least one outgoing edge means there are no dead ends – there is always an edge to continue along. Unlike classic graph problems, which maximize the total sum of weights, here the comparison is based on lexicographic order. We have to find the sum of the first K edges on a lexicographically largest path (comparing the weights).

At first glance, this might seem easier: a straightforward simulation appears sufficient; we begin the traversal at the starting vertex of the edge with the largest weight, and in each round, we choose the currently accessible edge with the largest weight. However, the situation is more complex because multiple edges may have the same weight, leading to frequent ties for the maximum. Therefore, if we rely solely on simulation, each decision may involve multiple tied maximums, resulting in exponential time complexity.

Subtask 1

The small constraints allow a direct `BFS`-like approach for the solution. We perform a simulation where round 0 begins with all vertices active, and in each round, we select the currently accessible edges with the largest weight and store their endpoints for the next round. The answer is obtained by storing the sum of the weights for each round.

Code: `diamonds6.cpp`
Time complexity: $O(N + M^K)$. Memory complexity: $O(N + M^K)$.

Subtask 2

It is easy to see that the exponential behavior of the previous approach is because in each round we unnecessarily repeat the same vertices multiple times. We can use a boolean array to flag the vertices that have already been added to the next round or alternatively store each round's vertices in a boolean array (instead of an integer vector).

Code: `diamonds5.cpp`
Time complexity: $O(K(N + M))$.
Memory complexity: $O(M + KN)$ or $O(M + N)$ if we only store previous and current round.

Subtask 3

This subtask is a major step toward the full solution because K is large. We can see that there is an optimal construction for the lexicographically largest path, since after sufficiently many rounds, some edges (and vertices) must repeat because they belong to the lexicographically largest sequence of weights among all possibilities.

Lemma 1: There exists a lexicographically largest path that begins with a simple path (without duplicating vertices) and then repeats a simple cycle.
go to proof

If we know the optimal construction, it becomes a straightforward task to compute the answer. The sequence of weights starts with some numbers and then it becomes periodic. For greater K, we have to find the sum of the terms of a periodic sequence.

There are two approaches for the task – one is by doing smarter simulation round by round and the other is by computing a dp to find the optimal decisions. Both approaches require a bound on the number of iterations. One can experimentally or intuitively find that this number of iterations is linear. In fact, two paths have identical sequences of weights for all rounds if and only if they have identical sequences of weights for the first $2N$ rounds. We will discuss this in detail in the next subtask.

The first approach is a continuation of the idea in the previous subtasks. Let's consider we have made sufficiently many rounds (at least $2N$). This means the non-optimal paths considered initially have already been eliminated, and we are now repeating the cycles of the possible optimal constructions. However, it is not straightforward, since the optimal paths can still be continued non-optimally for a couple of rounds (and then eliminated), after that these non-optimal continuations reappear, and so on. We can see that the only certain thing is that the starting vertices of all remaining paths need to be part of an optimal construction, otherwise they would have been eliminated already. So for each (r, v), where r is the round and v is the vertex, it is enough to store a $(r-1, u)$ it came from (if there are multiple options, it is enough to store one). Then, after $2N$ rounds, we pick an arbitrary vertex, recover the path, and examine the starting part, which must be an optimal construction (a path and then a cycle).

The other approach is to use dynamic programming $dp[r][v] =$ the optimal sequence of weights starting from vertex v and having r rounds (which is also the length of the sequence). Let $w(v, u)$ denote the weight of edge (v, u). We have the following recursive relation:

$$dp[r][v] = \max_{u:\text{ for all edges }(v,u)} \text{lexicographically } \{w(v,u), dp[r-1][u]\}.$$

As the constraints are smaller, we can compute the dp for sufficiently many rounds R

(at least $2N$) fast enough without optimizations. We pick the optimal sequence from $dp[R][0], dp[R][1], ..., dp[R][N-1]$. There are different ways how exactly we can find the answer. We can store additional information so we can find the vertices of the edges that form the optimal sequence and then find the optimal construction from the vertices.

Another way, used in the implementations, is to use only the sequence of weights. For this R needs to be at least $3N$. We know the first elements must come from the path of an optimal construction, so to identify the repeating cycle we temporarily ignore the first N elements. As the length of the repeating cycle in the optimal construction is at most N, we can find any valid repeating cycle in the sequence of the last $2N$ elements.

The only drawback with the second approach is the memory if we store directly the sequences in each state. We have to be more efficient here and store the sequences as linked lists so $dp[r][v]$ is only the head element of the sequence, and that element is linked to some optimal (for v) $dp[r-1][u]$.

	First approach	**Second approach**
Code	`diamonds_iliyan_n2_mem.cpp`	`diamonds4.cpp`
Time complexity	$O(NM)$	$O(N^2M)$
Memory complexity	$O(M+N^2)$	$O(M+N^2)$

Subtask 4

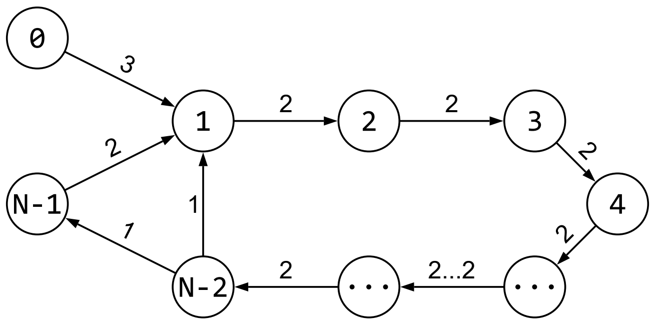
The constraints for this subtask make the graph consist of a single cycle or multiple disconnected cycles. This subtask is given so that competitors can think more about the cycles and the needed rounds for comparison. We can find the optimal traversal by comparisons:

- For two traversals in the same cycle, we only need to compare their results of the first $length(cycle)$ rounds;
- For two traversals in different cycles, we only need to compare their results of the first $lcm(length(cycle_1), length(cycle_2))$ rounds.

Although not necessary for this subtask, we can use the following lemma, which shows that only a linear number of comparisons are needed for the second case.

Lemma 2: If two periodic sequences with periods p and q have the same first $p+q$ terms, then they are identical.
go to proof

There is an even stronger bound: it is enough to compare the first $p+q-\gcd(p,q)$ terms but this is not needed for the task and does not change the complexity. The lemma tells us that we need approximately $2N$ rounds to compare any two cycles. In the general case when the optimal construction starts with a path this would require approximately $3N$ rounds (additional N to enter the cycle). But it is easy to see that still $2N$ rounds are enough because the cycle and the path have to be disjoint (they cannot share vertices). This is a tight bound as illustrated in the following example, where $2N - 3$ rounds are required to compare the two possible optimal constructions (there are also such tests with a modified version):



Code: `diamonds9.cpp`
Time complexity: $O(N^2)$. Memory complexity: $O(N)$.

Subtask 5

The task becomes much simpler when all $d[i]$ are distinct. There is only one maximum choice for each round, so the only challenge is handling large K. It is easy to see that the optimal construction is first a simple path and then repeating a simple cycle. So we follow the maximum weight for each round until we find a cycle, and then we have to find the sum of the terms of a periodic sequence. This subtask allows contestants to observe the optimal construction more easily.

Code: `diamonds_iliyan_unique_d.cpp`
Time complexity: $O(N + M)$. Memory complexity: $O(N + M)$.

Subtask 6

This subtask is similar to the previous one as the optimal construction is much easier to see than in the general case. Let edge (u, v) be the one with weight 2. We should use the weight 2 as often as possible. We have to start with (u, v) and then return as quickly as possible (using the fewest number of weight-1 edges) to vertex u. If this is not possible, the answer is simply $2 + (K - 1) = K + 1$. Otherwise we have to compute how many times we will use weight 2 and the answer is $K+$ this number. We can use BFS to find the shortest path of 1-s from v to u or determine that no such path exists.

Code: `diamonds8.cpp`
Time complexity: $O(N + M)$. Memory complexity: $O(N + M)$.

Subtask 7

We now revisit the two approaches and discuss the different ways to optimize them to fit the small memory constraint. There are two ways to optimize the first approach, which relied on smarter round-by-round simulation. We can use a `bitset` memory optimization. Instead of storing, for each (r, v), a $(r-1, u)$ it came from, we can use one `bitset` per round to record the vertices that are present. If we know the vertices in consecutive rounds it is easy to find a path that starts from $(0, v)$ and goes to (R, u) for some vertices v and u, and a final round R. If we are at (i, x) and we have an edge (x, y) in the original graph, we have to check if vertex y was present in the next round $i+1$, i.e. if $(i+1, y)$ existed during the simulation. Once we find a path, we can compute the answer as before.

The second way to optimize the first approach uses the earlier idea of finding the answer using only the sequence of weights. We can store the maximum weight that is used for each round and construct the optimal sequence of weights without knowing the exact path. Then, as described earlier, we can find the cycle by examining the last $2N$ elements and then compute the final answer. (we still need at least $3N$ rounds of simulation)

For the second approach, we will first optimize the time complexity. We recall the recursive relation:

$$dp[r][v] = \max_{u:\text{ for all edges }(v,u)} \text{lexicographically } \{w(v,u), dp[r-1][u]\}.$$

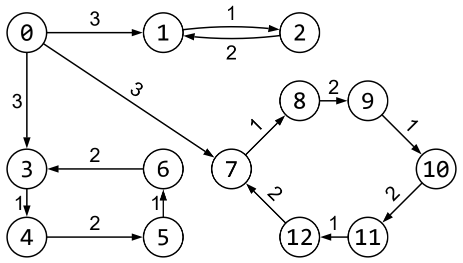
The bottleneck is the "max lexicographically" comparison, which we previously did in linear time, but this can be improved. We will keep a sorted list of $dp[r-1][v]$ (for fixed $r-1$) and we need to obtain a sorted list of $dp[r][v]$. For this we can store in $dp[r-1][v]$ "ranks" which are numbers that show the relative order of the weight sequences. To obtain the sorted list of $dp[r][v]$, for each v we use $w(v,u)$ as a primary key and $dp[r-1][u]$ as a secondary key, where u is the argmax in the relation above for v. Then we sort these pairs lexicographically and find the new ranks for $dp[r][v]$. We can even do this in linear time as we can do a `radix sort` over the pairs. It is also possible not to fix the number of iterations and terminate when the ranks stop changing.

Now we have good time complexity and we will find the optimal construction in the following way. Suppose we have all $dp[R][v]$ for some number of rounds R, and we construct a new graph with the edges (v, u) for each v, where u is the argmax in the `dp` relation for $dp[R][v]$. We claim that if R is at least $2N$ and we start from the optimal v (with the highest rank $dp[R][v]$), and follow these edges, we will find an optimal construction. This is because all paths with such length starting from v must follow an optimal construction so even if there are ties for argmax for v, all of them will be part of optimal constructions. A similar argument holds for the other vertices we visit along the path since non-optimal possibilities would be eliminated for large enough R. Thus, we directly obtain the optimal construction, from which the answer follows as before. For this solution, $R = N$ is also sufficient, since the `dp` takes a more global view and removes non-optimal possibilities from the beginning, and needs additional iterations to propagate that information.

	First approach	**Second approach**
Code	`diamonds_encho_bitset.cpp`, `diamonds7.cpp`	`diamonds1.cpp`, `diamonds_iliyan.cpp`
Time complexity	$O(NM)$	$O(NM + N^2 \log N)$ or $O(NM + N^2)$
Memory complexity	$O\left(M + \frac{N^2}{64}\right)$ for the `bitset` and $O(N + M)$ for the other solutions	$O(N + M)$

Final thoughts

During testing, we found a particular variation of the first approach. Instead of reasoning about the number of rounds, we can terminate if we repeat the same set of vertices during a round (which we can find using `hashing`). Then the cycle is the sequence of weights between the two repetitions. It turns out exponentially many rounds may be required for the repetition to happen so we still require a bound for the rounds. The best type of test we came up with to battle such cheat solutions is the following:



The base model is we have some constant k and then we make cycles of lengths $k, 2k, 3k, 5k, 7k, 11k, \dots$ that repeat $1, 2, \dots, k$ as weights (in the example $k = 2$ and the cycles are of lengths 2, 4 and 6). In this way we need the LCM of the lengths (a huge value), as the number of rounds required to repeat the same set of vertices. Also if we terminate early, it is not simple to compute the answer looking at the sequence of weights, since we still have a non-trivial repeating cycle (the simplest correct way is to have a full solution like `diamonds7.cpp`). Unfortunately, it turned out we had missed adding these tests in the official test data so a few contestants passed by terminating early and computing the answer in a wrong way.

The original proposal for this task was even harder, but we believe the current variant is more appropriate for EJOI. In fact, it turned out to be slightly harder for contestants than we initially thought.

Proof of Lemma 1

Lemma 1: There exists a lexicographically largest path that begins with a simple path and then repeats a simple cycle.

Suppose a lexicographically largest path is: $L := (v_0, v_1), (v_1, v_2), \dots$. As the path is infinite and the different vertices are only N, there has to be a repeated vertex. Let the first repeated vertex be v_j, with its previous occurrence at v_i $(0 \le i < j)$. This also means that $v_0, v_1, \dots, v_{i-1}$ is a simple path and $v_i, v_{i+1}, \dots, v_{j-1}, v_i$ $(v_i = v_j)$ is a simple cycle. Let $P := (v_0, v_1), (v_1, v_2), \dots, (v_{i-1}, v_i)$ and $C := (v_i, v_{i+1}), (v_{i+1}, v_{i+2}) \dots, (v_{j-2}, v_{j-1}), (v_{j-1}, v_i)$.

We will prove the following claim by induction: for each $r \ge 1$, the sequence of weights in L starts with the sequence of weights of $\{P, C^{(r)}\}$, where $C^{(r)}$ denotes $\underbrace{C, \dots, C}_{r}$.

Base case $r = 1$. We know that the path L starts with $\{P, C\}$ so the claim holds.

Induction step. Let the claim be true for some $r \ge 1$. We will prove it for $r + 1$.
The claim is true for r, so the weights of L start with the weights of $\{P, C^{(r)}\}$, and we will denote the edges after that with S. We will also denote $P' := \{C, S\}$.

The sequence of weights of L equals that of $\{P, C^{(r-1)}, P'\}$. Since $\{P, C^{(r-1)}, S\}$ is a valid path (with one fewer cycle), P' must be lexicographically larger than or equal to S. If they are equal, then we are ready as this would mean that the weights of S are starting with the weights of C (since $P' = \{C, S\}$), so P' is starting with weights $C^{(2)}$ and L is starting with weights $\{P, C^{(r+1)}\}$.

Let's assume they are not equal. So there must be an edge in P' with larger weight than the corresponding one in S. Let the first such edge be e. If e is in the cycle C, then the cycle will be lexicographically larger than the corresponding part of S, so a path $\{P, C^{(r+1)}\}$ will be lexicographically larger than L, which is a contradiction.

The only remaining case is when e is after the cycle C. This would mean that the sequence of weights in the cycle C is the same as the corresponding sequence of weights in S, which means that the weights of S are starting with the weights of C, and L is starting with weights $\{P, C^{(r+1)}\}$.

Thus, the claim follows by induction. This means that the sequence of weights in L is the same as the sequence of weights of $\{P, C, C, \dots\}$. Since $\{P, C, C, \dots\}$ is a valid path, it is a lexicographically largest path that begins with a simple path and then repeats a simple cycle. $\square$

Proof of Lemma 2

Lemma 2: If two periodic sequences with periods p and q have the same first $p+q$ terms, then they are identical.

Without loss of generality, assume $p \leq q$. Let the sequences be $P_0, P_1, \ldots$ and $Q_0, Q_1, \ldots$

We will prove the following claim by induction: for each $r \geq 0$, the sequences have the same first $p+q+rp$ terms.

Base case $r = 0$. As the sequences have the same first $p+q$ terms, the claim holds.

Induction step. Let the claim be true for some $r \geq 0$. We will prove it for $r+1$.
Let's assume the contrary and let $p+q+rp \leq i < p+q+(r+1)p$ be an index such that $P_i \neq Q_i$. We have the following:

- As P has a period p, we have $P_i = P_{i-p}$.
- As $i-p < p+q+rp$, we have $P_{i-p} = Q_{i-p}$.
- As Q has a period q, we have $Q_{i-p} = Q_{i-p-q}$.
- So, $P_i = P_{i-p} = Q_{i-p} = Q_{i-p-q}$.

Similarly, $Q_i = Q_{i-q} = P_{i-q} = P_{i-p-q}$ (since $p \leq q$, we have $i-q \leq i-p < p+q+rp$). But $0 \leq i-p-q < p+q+rp$, so $P_{i-p-q} = Q_{i-p-q}$ by the induction step and this means $P_i = Q_i$, which is a contradiction.

Thus, the claim follows by induction. This means that the sequences must be identical, since we can pick an arbitrarily large r. $\square$