Stones – solution

Ivan Žufić, Daniel Paleka

September 4th, 2021

This problem is a twist on the standard Nim game, with the catch that the pile to take stones from is adversarially chosen by the opponent.

However, the solution has nothing to do with the Sprague-Grundy theory and bitwise operations. Instead, we will only use the standard approach of separating the states into winning and losing states.

Definition 0.1. *A state is the set $(a_1, \ldots, a_N)$ of piles, together with the index i of the pile the opponent selected.*

Let's write down the inductive definition of losing and winning state in the context of this game:

1. The states with all piles empty are losing;
2. A state is winning if and only if the turn player can make a move that reaches a losing state;
3. A nonempty state is losing if and only if the turn player must make a move that reaches a winning state.

The above defines a valid partition of all states by induction on the number of stones.

0.1 The first two subtasks

For the first subtask, we can explicitly enumerate all $(M+1)^N \cdot N$ states (some are equivalent, but $8^7 \cdot 7$ is small enough) and brute force the described induction using dynamic programming.

A good strategy for solving this problem is: implement the first subtask, get the points on the evaluator, and then try to find some pattern.

The first thing that is easy to see from the printed moves is that the turn player should always take everything or leave exactly one stone in the chosen pile. Using this observation, we can implement a dynamic programming in time complexity $O(3^N)$.

0.2 The full solution

We can find a characterization of the losing states.

Let A be the number of piles containing exactly one stone.
Let B be the number of piles containing strictly more than one stone.
Let H be the number of stones in the currently selected pile.

If $B = 0$, we win if and only if A is odd. It's easy to prove this by induction.

Otherwise, let $K \geq 1$. Partition the post-pile-selection states into three types:

- Let *type-W* states be such that A is even;
- Let *type-L* states be such that A is odd and $H = 1$.
- Let *type-O* states be other states, i.e. A is even and $H > 1$.

Claim. Every *type-W* state is winning, and every *type-L* state is losing.

Proof. We have two cases.

- First case: we're in a *type-W* state.

  If $H \neq 1$ and $B = 1$, then we can remove all stones from the current pile, which is clearly a winning move, since A is even.

  If $H = 1$, we can remove the only stone in that pile and select any other pile with exactly one stone. This is possible, since A is even. This leads to a *type-L* state with smaller total number of stones.

  Otherwise, if $H \neq 1$ and $B > 1$, then remove $H - 1$ stones from the pile, and force the next player to select the last stone from current pile. This also leads to a *type-L* state with smaller total number of stones.

- Second case: we're in a *type-L* state.

  Note that $H = 1$ and A is even, so the next state will have $B > 0$, odd A, and it will have smaller total number of stones.

Every *type-W* state leads to a *type-L* state or some losing state with $B = 0$, and every *type-L* state leads to a *type-W* state, and in every transition total number of stones is smaller, so *type-W* are winning states, and *type-L* are losing positons.

□

Remark. *Type-O* states are winning, but that is not important at all, since it is guaranteed that the starting state is winning regardless of the first selected pile.

To solve the problem, implement the moves used to prove the Claim. It is enough to take $O(N)$ time per move, that is, $O(NM)$ time in total.

Tortoise – solution

Josip Klepec

September 4th, 2021

1 Setup

Definition 1.1 (Area)**.** *An* Area *is a maximal continuous subsequence of locations with at least one candy and no playgrounds.*

Lemma 1.1. *Tom buys candies in the order of increasing location.*

Proof. We can easily prove that it is never optimal to first buy a candy from a later Area and then return to buy a candy from a previous Area.

Then we can prove that in each Area we will buy candies in order. This is left as an exercise, as the proof is very similar to the proof of the the next Lemma. □

Lemma 1.2. *In each Area, Tom will first take some candies to the left playground (possibly zero) and then some candies to the right playground.*

Proof. Assume he takes a candy to the right playground, then takes the next one to the left.

Let $a < b \leq c < d$ be the locations of the playground, the first and the second cady, and the right playground.

- If $d - c \leq b - a$, we can take both candies to right playground and achieve better result (less time spent) and arrive at the same time to the second candy.
- Else, we can take the first candy to the left and second candy to the right because

  $$(d - b) + (d - a) + (c - a) > (b - a) + (d - a) + (d - c).$$

  (we spend less time in total); and

  $$(d - b) + (d - c) > (b - a) + (c - a)$$

  (we will arrive earlier to the second candy).

Doing this exchange a finite number of times we can achieve that we take some prefix of candies to the left playground and some suffix to the right. □

Lemma 1.3. *In each Area with a right playground, if Tom takes at least one candy, he will take at least one to the right playground.*

Proof. If he takes candies only to the left playground, he can just take the last one to the right playground. □

Corollary 1.4. *In each Area with a right playground, Tom will take at least one candy.*

Definition 1.2 (Left cycle)**.** *Tom takes a candy, brings it to the left playground on his left and comes back.*

Definition 1.3 (Right cycle)**.** *Tom is located at the right playground of some Area. He goes back to pick up the candy and brings it to the playground.*

Definition 1.4 (Candy walk)**.** *Tom buys a candy. Walks to the right playground and drops it there.*

Lemma 1.5. *If a candy is closer to the right playground, it will not be involved in a Left cycle and vice versa.*

Proof. Obviously only one of those things can happen. With exchanges we can keep the solution valid with less time spent in this Area. □

Lemma 1.6. *If a candy is at the same distance to both playgrounds, it can be shown that it doesn't matter which cycle we do with it. We can always do left for example.*

2 Greedy

Greedy algorithm works as follows.

For a candy i we use x_i to denote its position and w_i for twice the distance to its closest playground. We order the candies such that $w_1 \leq w_2 \leq \dots$

We consider candies and its cycle in order. If adding that cycle keeps solution valid, we add that cycle. Multiset of cycles is valid if we can manage to buy everything in time and in each Area with right playground we can do one Candy walk (we can carry some candy to the right playground after left cycles and before right cycles).

3 Proof

Lemma 3.1. *Tom makes exactly one candy walk in any area which has a playground to its right and has at least one candy.*

We say a set of candies S is valid if Tom can pick up this set of candies, each candy in one cycle and in addition one candy from each non-candiless area with a playground to its right.

Let S_G/S_O be the set of candies picked up with a cycle by the greedy/optimal algorithm, respectively. For a candy i we use x_i to denote its position and w_i for twice the distance to its closest playground. We order the candies such that $w_1 \leq w_2 \leq \dots$

Let $Y = S_G \setminus S_O$ and let $i = \min Y$. Then it is enough to find a valid set S'_O with $|S'_O| = |S_O|$ and $\min S_G \setminus S'_O > i$. Let $X = S_O \setminus S_G$. Observe that, by our greedy algorithm, we have

$$\min X > \min Y = i. \tag{1}$$

We consider several cases:

- There is no playground to the right of x_i.

  Let j be leftmost candy in X. We have $w_j \geq w_i$ by (1) and set $O' = O \setminus \{j\} \cup \{i\}$.

  This is trivial case.

- Cycle i is a Left cycle and Candy walk in optimal algorithm is to the left or at the same candy as i.

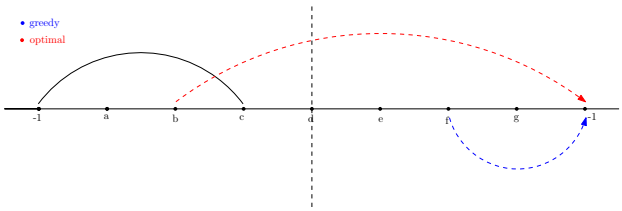

Figure 1

In this example $x_i = c$

If there exist a cycle right of i and left of Candy walk in greedy in X (locations c, d, e, f), then we remove leftmost such j cycle and it becomes a new candy walk.

Else let j be the leftmost candy in X. New Candy walk will be the same as in greedy (location f).

In both cases we have $w_j \geq w_i$ by (1) and set $O' = O \setminus \{j\} \cup \{i\}$.

- Cycle i is a Left cycle and Candy walk in optimal algorithm is to the right.

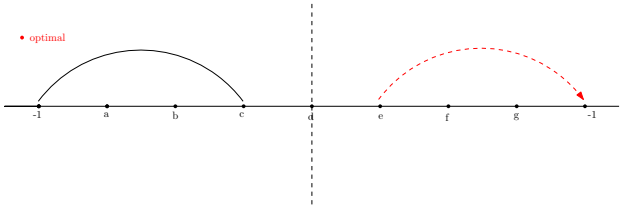

Figure 2

Let j be the leftmost candy in X.

If it is left of i then Candy walk will stay the same (we removed longer cycle that happened before).

Else, j is in some Area. If it is Left cycle, Candy walk in that Area can stay the same (because we added smaller cycle than we added and both of them happen before). But if it is the Right cycle, then that Area can use the same Candy walk as greedy. Because prefix is the same.

Again we set $O' = O \setminus \{j\} \cup \{i\}$.

- Cycle i is a Right cycle.

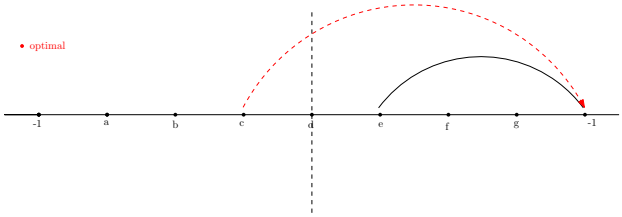

Figure 3

Candy walk in optimal solution cannot be to the right of i because that cycle would then also be in greedy (greedy takes some suffix of Right cycles).

Let j be the rightmost left of i u X if such exists. If the candy walk was at x_i and needs to be moved to the left then we can set candy walk to any available candy in this Area. Such will exist because we will remove either the last left cycle in this Area or some bigger cycle. And candy will be available for a candy walk because greedy algorithm is also valid.

Otherwise let j be the leftmost in cycle in X. This case is similar to the previous one.

4 The algorithmic parts

4.1 First subtask

This can be achieved with dynamic programming and bitmasks (with state $mask, time, position$). Or just using the assumption that Tom will take candies in order and then using solely bitmasks.

4.2 Second subtask

This can be achieved with dynamic programming keeping time, position, position of last taken candy and a boolean saying whether we have a candy in hand.

4.3 Third subtask

This can be achieved with dynamic programming. We are using result of Lemma 1.2. Having boolean did we start taking candy to the right playground in current Area.

4.4 The full solution

We simulate the greedy algorithm described in 2. To get full points one should use some kinda of data Structure to check validity when adding a cycle.

For the fourth subtask, it is enough to implement this validity check in linear time. And for full points we can keep how many seconds before Wilco we get to every shop.

Wells – solution

Domagoj Bradač

September 4th, 2021

1 Notation

We denote the given tree by $T = (V, E)$. We write $P(u, v)$ for the unique path from u to v in T and $d(u, v)$ for the number of edges in $P(u, v)$. The diameter of T is a longest (simple) path in T. We denote a fixed diameter of T by D and imagine the edges not on D to be directed away from D. Given a vertex $v \in T$, we write $r(v)$ for the closest vertex of v on D (if $v \in D$, then $r(v) = v$). We write T_v for the subtree rooted at v following the directed edges away from the diameter. We denote by $h(v)$ the height of T_v and by b_v some vertex on the "bottom" of T_v, that is, $b_v \in T_v$ and $d(v, b_v) = h(v)$.

We define a *partial colouring* of T as follows: we say that a vertex is coloured red if it is marked, blue if it is not marked and uncoloured if we have not yet decided whether or not it is marked. Given a partial colouring c, a path of length $k-1$ is said to be invalid with respect to c if it contains more than one red vertex or all its vertices are coloured blue, otherwise it is *valid*. A (partial) colouring is invalid if it contains an invalid path of length $k-1$, otherwise it is valid. A colouring is *complete* if all vertices are coloured (red or blue). A partial colouring c is an *extension* of a partial colouring c' if $c(v) = c'(v)$ for any vertex v which is coloured under c'.

2 Observations

Recall the following fact about a diameter of a tree.

Fact 2.1. *Suppose z_ℓ, z_r are endpoints of a diameter of T. Then for any vertex $v \in T$:*

$$\max_{u \in T} d(v, u) = \max\{d(v, z_\ell), d(v, z_r)\}.$$

Let $D = (d_1, d_2, \dots d_L)$ be a diameter of T with endpoints $z_\ell = d_1$ and $z_r = d_L$. If $L < k$, then there is no path of length $k-1$ in T, so the answer is YES and the number of colourings 2^n. Now, assume $L \geq k$. Then, there are k ways to colour the diameter, for any $1 \leq i \leq k$, we colour the vertices $d_i, d_{i+k}, d_{i+2k}, \dots$ red and the other vertices of D blue.

We say a path P is *crossing* if both its endpoints are not in D and it contains a vertex in D.

Claim 2.2. *Let c be an arbitrary complete colouring of T. Then c is valid if and only all non-crossing paths are valid with respect to c.*

Proof. Suppose this is not the case, and consider a crossing path P of length $k-1$ with endpoints u, v. Assume without loss of generality that $z_l, r(u), r(v), z_r$ appear in this order on D, with possibly some of these

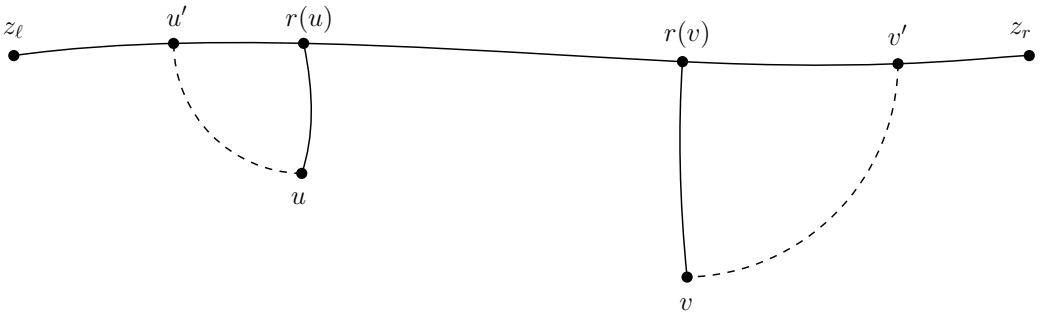

Figure 1

vertices being identical. Since D is a diameter, there exists a vertex $v' \in D$ such that $d(u, v') = d(u, v) = k$ and $r(v)$ lies between $r(u)$ and v'. Analogously, there exists a vertex $u' \in D$ such that $d(u', v) = d(u, v) = k$ and $r(u)$ lies between u' and $r(v)$ (see Figure 1). Note that

$$d(u', v') = d(u, v') + d(u', v) - d(u, v) = k + k - k = k.$$

For vertices x, y let $f_c(x, y)$ denote the number of red vertices in $P(x, y)$ under the colouring c and note that a path $P(x, y)$ of length $k - 1$ is valid if and only if $f_c(x, y) = 1$. Then:

$$f_c(u, v) = f_c(u, v') + f_c(u', v) - f_c(u', v').$$

Under our assumptions, the paths $P(u, v'), P(u', v)$ and $P(u', v')$ are valid, implying $f_c(u, v) = 1$. In other words, $P(u, v)$ is valid as well. □

Consider a vertex v not on the diameter such that

$$h(v) + \max\{d(v, z_l), d(v, z_r)\} < k - 1. \tag{1}$$

As D is a diameter, this implies there is no path of length $k - 1$ containing v. Hence, v can be coloured arbitrarily without changing the validity of the colouring. We delete all vertices satisfying (1) and multiply the number of colourings by 2 for each deleted vertex. From now on, we assume there are no vertices satisfying (1).

Claim 2.3. *Let c be a partial colouring and let v be a vertex such that there exists a path of length at least $k - 1$ containing v and a vertex coloured red under c. Then in any proper extension of c, the colour of v is the same. In other words, if such a path exists, the colour of v is uniquely determined.*

Proof. We need to show that the colour of v is uniquely determined by the partial colouring c. This follows immediately. Let P be a path of length at least $k - 1$ containing v and a red vertex under c. Observe that every k^{th} vertex on P must be red. Since there is a red vertex u on P, it follows that v is red if and only if $d(u, v) \equiv 0 \pmod{k}$. □

Claim 2.4. *Suppose there is a path of length $k - 1$ not touching D. Then, there are at most two ways to colour D which can be extended to a complete valid colouring.*

Proof. Let $P = P(u, v)$ be a path of length $k - 1$ not touching D. As D is a diameter, this implies:

$$d(u, z_l), d(u, z_r), d(v, z_l), d(v, z_r) \geq k - 1.$$

As P does not touch D, we have $r(u) = r(v) = r$. Assume that $d(r,u) \geq d(r,v)$, implying $d(r,u) \geq (k-1)/2$. Observe that on $P(u, z_l)$ every k^{th} vertex must be red and the same holds for $P(u, z_r)$. It is easy to see that this can only happen if r is red or k is odd and r is in the middle of two consecutive red vertices on D. □

Definition 2.5. Let U be a tree rooted at a vertex r and let t be a nonnegative integer. Define the depth of a vertex $v \in U$ as $d(r,v)$. A complete colouring of U is a *t-covering* of U if:

- every red vertex is at depth at most t; and
- for every vertex u such that $d(r,u) \geq t$, there is exactly one red vertex on $P(r,u)$.

Definition 2.6. We call a vertex $v \notin D$ *pinned* if

$$h(v) + \min\{d(v, z_\ell), d(v, z_r)\} \geq k - 1, \tag{2}$$

and *loose*, otherwise.

Claim 2.7. *Let $v \notin D$ be a loose vertex. Without loss of generality, assume $h(v) + d(v, z_\ell) < k - 1$ and $h(v) + d(v, z_r) \geq k - 1$. Let c be a valid partial colouring in which the following vertices are coloured:*

- *all vertices in D,*
- *all vertices on the path $P(v, r(v))$ except v.*

Consider extensions of c obtained by colouring the vertices of T_v in some way. Then

a) if there is a red vertex in c on the path $P(v, z_r)$, there is a unique valid extension,

b) otherwise an extension is valid if and only if it forms a t-covering of T_v with $t = k - 1 - d(v, z_r)$.

Proof. We consider the two cases in order.

a) Then for any $u \in T_v$, we have $d(u, z_\ell) + h(u) < k - 1$, implying $d(u, z_r) + h(u) \geq k - 1$. By assumption there is a red veretx on $P(u, z_r)$ so by Claim 2.3, the colour of u is uniquely determined.

b) Note that any non-crossing path of length $k - 1$ with one endpoint in T_v has the other endpoint in $P(r(v), z_r)$. This follows directly from $d(u, z_\ell) < k - 1$ and the fact that D is a diameter. Now consider a path $P(u, w)$ of length $k - 1$ where $u \in T_v$ and $w \in P(r(v), z_r)$. By assumption there are no red vertices in $P(v, z_r) \setminus \{v\}$. Hence, for c' to be valid there must be exactly one red vertex in $P(u, v)$. In other words, c' forms a t-covering of T_v. It is easy to see that the converse also holds: if an extension of c obtained by colouring T_v forms a t-covering of T_v, then it is valid. □

Let us summarize what we have shown so far. We may delete all vertices satisfying (1). Given a colouring of D, consider the vertices $v \in T \setminus D$ which are loose but their parent is on D or is pinned. Then, either the colouring of T_v is uniquely determined or there is an integer t such that the colouring of T_v is any t-covering of T_v. The colours of all other vertices are uniquely determined by the colouring of D. Now, consider a colouring c satisfying these properties and suppose P is a path of length $k - 1$ invalid under c. By Claim 2.2, P is non-crossing. Recall that we first colour D and then extend this colouring to other vertices in a way that paths touching D are valid, so by construction, P does not intersect D. By Claim 2.4, if such a path P exists there are at most two ways to colour D and these two ways can be easily determined. Consider one such colouring and extend it to all pinned vertices in a unique way and to all loose vertices as described by Claim 2.7. By construction, any path invalid path under such a colouring is disjoint from D and contains only pinned vertices.

3 The algorithm

First, remove all vertices satisfying (1) and multiply the number of colourings by 2 for each deleted vertex. For any vertex $v \notin D$ such that v is loose but its parent is on D or is pinned, using dynamic programming calculate the number x_v of t-coverings of T_v with the appropriately chosen value of t. Then, for a colouring of D for which the colouring of T_v is not uniquely determined (as given by Claim 2.7), multiply the number of ways to extend this colouring by x. Finally, if there is a path of length $k-1$ not touching D, then there are at most two colourings of D and for each of them we can easily check whether they can be extended to a complete colouring.

The algorithm can be implemented to run in time $\mathcal{O}(n)$. Let us elaborate. Finding the diameter and determining which vertices satisfy (1) and which are pinned can easily be done in linear time. Computing the number of t-coverings is done in linear time using dynamic programming. If there is a path of length $k-1$ not touching D there are at most two colourings of D we need to consider. The validity and number of extensions of each of these can be done in linear time. Finally, assume there is no path of length $k-1$ disjoint from D. Consider a vertex $v \in T \setminus D$ which is loose but its parent is on D or pinned, that is, v is a vertex for which we need to calculate the number of t-coverings, denoted by x_v, for some $t = t(v)$. As v is loose, either $h(v) + d(v, z_\ell) < k-1$ or $h(v) + d(v, z_r) < k-1$ and assume the former holds, the other case being analogous. By Claim 2.7, for any colouring of D for which all the vertices on $P(v, z_r)$ (which is uniquely determined) are blue, we need to multiply by x the number of ways to extend this colouring of D. In other words, if we enumerate the k possible colourings of D in the natural way, we arrive at the following problem. We are given at most n cyclic intervals I_j of residues modulo k, where each interval I_j, ends with 0 (if $d(v, z_\ell) < k-1$) or starts with $d(z_\ell, z_r) \bmod k$ (if $d(v, z_r) < k-1$), and values x_j for each of these intervals. For each residue $y \in [0, k-1)$, we need to calculate the product

$$\prod_{j \mid\, y \in I_j} x_j.$$

This can be done in time $\mathcal{O}(n)$, without calculating any modular inverses, by considering separately the intervals starting with 0 and those ending with $d(u_\ell, u_r) \bmod k$ and calculating prefix products in linear time.

4 Subtasks

In this last section, we describe possible partial solutions to this problem. Observing some of the facts mentioned in Section 2 can lead to various solutions. It seems difficult to anticipate all possible variations and here we describe only some of the possibilites. All of the described solutions can be implemented with or without counting the number of valid colourings which yields 60% of points for the subtask.

Note that whenever a t-covering is required in the above solution, one can replace it with a 0-covering, that is, only colour the root of the subtree. It is easier to show this is valid (if there exists a valid colouring at all) and it is simpler to check as it is only a single colouring. This way a solution worth 60% of points can be obtained (for any subtask).

4.1 $\mathcal{O}(N^2K)$ – 30 points

We represent the colouring of D by K Boolean variables $x_0, x_1, \ldots, x_{K-1}$, where vertex d_i is marked if and only if the value of $x_{i \bmod K}$ is true. We label each vertex d_i with $i \bmod K$ and extend this labelling to all pinned vertices as follows. Let v be a pinned vertex. If there exists a vertex u on D whose distance from v is divisible by K, then give v the label of u. Otherwise, label v with -1 (meaning that it cannot possibly be

marked). We know that exactly one of $x_0, x_1, \ldots, x_{K-1}$ is true, and that x_i can be true if and only if i occurs exactly once on every path of length $K-1$ consisting only of pinned vertices/vertices on D. Finally, for each $i \in \{0, 1, \ldots, K-1\}$ with this property (call such an i *good*), we calculate the corresponding number of colourings as previously described. This approach has complexity $\mathcal{O}(N^2K)$ because there are $\mathcal{O}(N^2)$ paths of length $K-1$ and naively updating the set of good indices takes $\mathcal{O}(K)$ time.

4.2 $\mathcal{O}(N^2) - 20$ points

The approach in this case is basically the same as in the previous subtask, except that we more efficiently maintain the set of good indices while traversing the paths of length $K-1$. We traverse the tree starting at every possible vertex and keep a list of indices that do not occur exactly once on the current path but are not yet marked as bad. When we reach the end of a path of length $K-1$, we mark all vertices in the list as bad and clear the list. The list can be maintained by keeping for each index the number of its occurrences on the current path and its index in the list (or -1 if it is not an element of the list). In this way, the operations of adding elements to the list and removing elements from the list can be implemented to run in constant time.

4.3 $\mathcal{O}(N \log N) - 20$ points

Consider one of the K possible colourings of D. Additionally, colour red any vertex which is at distance divisible by k from any red vertex on D. Finally, check whether this colouring is valid. Observe that this way each vertex is coloured red at most twice across all K colourings of D. Suppose we fixed a colouring of D, extended it as described, yielding M red vertices in total. Using the auxiliary tree formed by these vertices (the tree containing the M vertices and all pairwise lowest common ancestors), which has size $O(M)$, it is possible to check the vailidity of the given colouring in time $\mathcal{O}(M \log M)$. Hence, this algorithm can be implemented to run in time $\mathcal{O}(N \log N)$. However, the implementation of this approach is very difficult.