-
-

Permutation Game

You are given a **connected** graph G of m vertices and e edges. The i-th edge of the graph G is (u_i, v_i).

You are also given a permutation P of length n, where $m \leq n$.

Define the **score** of the permutation P as the number of indices i such that $P_i = i$.

Two players, Alice and Bob will play a game with the permutation. The game will last for at most 10^{100} turns. In each turn, the following happens:

1. Alice decides to end the game, and the game stops.
2. Alice chooses distinct indices $i_0, i_1, \ldots, i_{m-1}$. Note that we do **not** require $i_0 < i_1 < \ldots < i_{m-1}$.
3. Bob choose an index $0 \leq j < e$ of the edges of the graph and swaps $P_{i_{u_j}}$ and $P_{i_{v_j}}$.

Alice wishes to maximize the score of the permutation while Bob wishes to minimize the score of the permutation.

This is an interactive problem.

You will play as Alice and the grader will play as Bob.

You will need to determine the maximum score of the permutation if both players play optimally and then play the game with Bob to achieve at least that maximum score after some turns.

Iteraction Format

You will have the implement the function `Alice` with the following function signature:

```
int alice(int M, int E, vector<int> U, vector<int> V, int N, vector<int> P)
```

Here the variables are defined as follows:

- M is the number of vertices of the graph G
- E is the number of edges of the graph G
- The vectors U and V describes the edges of G. The i-th edge of graph G connects vertices $U_i and$V_i$
- N is the length of the permutation.
- The vector P describes the given permutation.

Constraints:

- $2 \leq M \leq 400$
- $M - 1 \leq E \leq 400$
- $0 \leq U_i, V_i < M$
- $M \leq N \leq 400$
- $0 \leq P_i < N$
- The graph is simple and connected
- All elements of P are distinct

The function `Alice` should return the final score of the game if Alice and Bob plays optimally.

To make moves, the function `Alice` can call a function `Bob` with the following function signature:

```
int Bob(vector<int> I)
```

Here the vector I denotes the indices that Alice chooses to perform her operation on.

It should satisfy the following constraints:

- I contains m elements
- $0 \leq I_j < n$
- All elements of I are distinct

The function `Bob` will return a single integer j that satisfies $0 \leq j < m$.

This means that Bob swaps $P_{I_{U_j}}$ and $P_{I_{V_j}}$.

To get full score, Alice should call the function `Bob` until the score of the permutation becomes at least the optimal score.

Note that Alice's strategy should work no matter what moves Bob makes, including if Bob makes "unoptimal" moves.

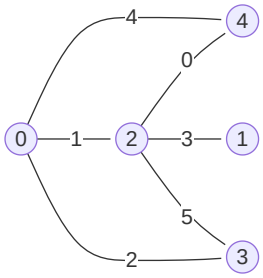
Here, we only require that the final score is at least the optimal score because Bob is allowed to make unoptimal moves.

Sample

Let the original call to `Alice` have:

- $M = 5$
- $E = 6$
- $U = [4, 0, 3, 4, 4, 2]$
- $V = [2, 2, 0, 0, 4, 3]$
- $N = 10$
- $P = [8, 2, 7, 6, 1, 5, 0, 9, 3, 4]$

The graph is as follows:



Given the constraints above, we can prove that the final score under optimal play is 1.

Alice will make the following 4 moves:

Argument of I to `Bob`	Return Value of `Bob`	P after the swap by Bob
$[3, 1, 5, 2, 0]$	5	$P = [8, 2, 5, 6, 1, 7, 0, 9, 3, 4]$
$[9, 3, 7, 2, 1]$	0	$P = [8, 9, 5, 6, 1, 7, 0, 2, 3, 4]$
$[5, 6, 7, 8, 9]$	1	$P = [8, 9, 5, 6, 1, 2, 0, 7, 3, 4]$
$[7, 5, 2, 3, 6]$	5	$P = [8, 9, 2, 6, 1, 5, 0, 7, 3, 4]$

Note that the moves of Alice and Bob above are not necessarily making the optimal moves. The moves shown are purely for the sake of demonstration.

After Alice has performed all the moves above, the score of the permutation is 2.

The function `Alice` will finally return 1, the maximum score achieveable under optimal play from both parties.

Note that even though Alice has achieved a score of 2 by playing with Bob, you will get 0 points if the return value of `Alice` was 2 instead of 1.

Subtasks and Scoring Suggestions

Subtasks will be based on graph types. The following are the suggested graph types for the subtasks:

- $m = 2$ (very easy)
- $e = m - 1$ and $u_i = i$ and $v_i = i + 1$ (medium)
- $m = 3$ and $e = m$ and $u_i = i$ and $v_i = (i + 1) \bmod m$ (hard)
- $m = 4$ and $e = m$ and $u_i = i$ and $v_i = (i + 1) \bmod m$ (hard)
- $e = m$ and $u_i = i$ and $v_i = (i + 1) \bmod m$ (very hard)
- $e = m - 1$ (medium)
- $e = m$ (very hard)
- $e > m$ (very easy)
- No restriction on G (very hard)

Then in each subtask, there are two scoreable parts:

1. The function `Alice` returns the correct value.
2. The participant will make calls to `Bob` until the score of the permutation is at least the optimal score. Suppose the participant makes $f(n)$ calls. Then there can be partial scoring based on $f(n)$. Currently, we have a solution with $f(n) \leq 3n$. We have proven that $f(n) \geq 3n - \epsilon$ below. I expect there to be a subtask where $f(n)$ is big enough to allow any reasonable solution to pass.

It is possible to have partial scoring also on how close to the optimal score the participant is able to get in part 2, but I that it would be hard to kill silly solution in these types of problems. And the metric for such a scoring would be quite hard to write down.

Also, the participant will not have to play as Bob. This is because I believe constructing the strategy for Alice is much harder than constructing the strategy for Bob. Requiring participants to implement Bob's strategy would make the implementation much more tedious, which is undesirable as the entire solution is quite tedious already, especially if a full score requires $f(n) \leq 3n$. Yet, removing the construction of Alice's moves would make the problem reward guesswork too much.

Ideally, the limits should be set such that code that takes $O(n)$ to make a move per turn will pass, it does not make sense to force the contestant to code an algorithm that only requires $O(m)$ per move.

Motivation of the Problem

This problem was created when brainstorming last minute ideas for a codeforces round.

The original idea was for Alice to specify 3 distinct indices a, b and c and for Bob to choose two of them and swap.

The solution here was very unexpected and interesting.

To make the problem harder, we generalized the problem to allow for arbitrary connected graphs G.

Other Remarks

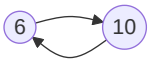
What if it is not required that G is connected?

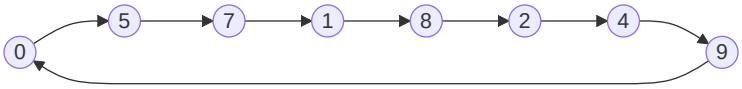
We did not manage to make much progress on this problem.

Solution

First, we note that we can view a permutation P of length n as the permutation graph G_P where the vertices are labelled from 1 to n and there is a directed edge $i \to P_i$ for all i. This graph is the union of directed cycles. For the rest of this editorial, we will view the permutation as a multiset of its cycle sizes.

For example, $P = [5, 8, 4, 3, 9, 7, 10, 1, 2, 0, 6]$



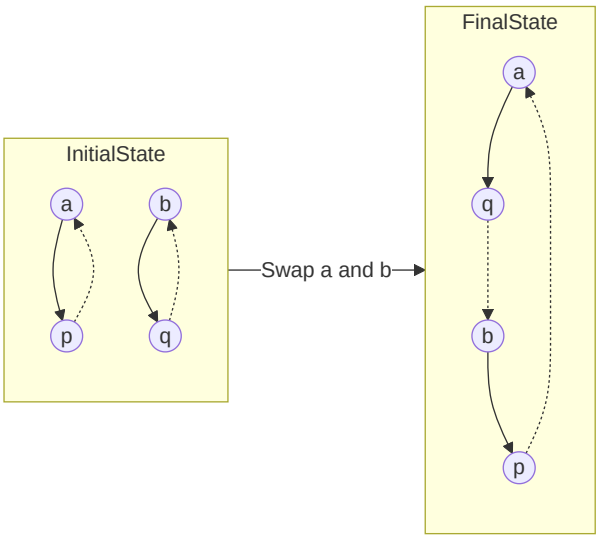


The cycle set of this permutation is $\{1, 2, 8\}$.

At this stage, we should note that the score of a permutation is the number of self-loops. That is, the number of 1s in the cycle set.

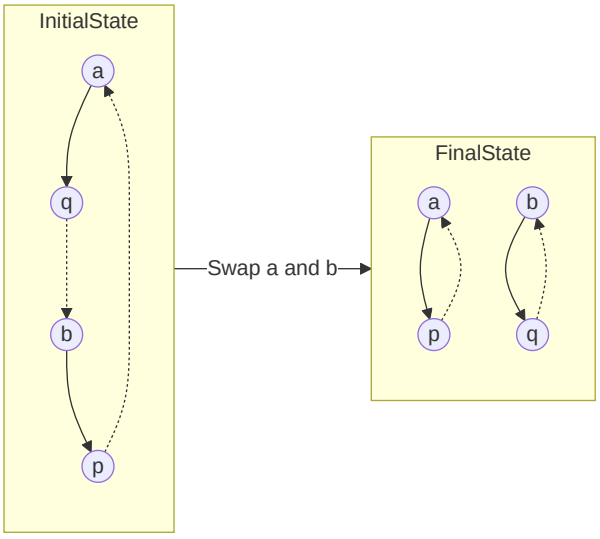
Now, let us analyze how swapping elements a and b changes the cycle set.

Suppose that a and b are from different cycles of sizes s_a and s_b. Then this operation merges both cycles into a large cycle of size $s_a + s_b$.



In our cycle set, this is equivalent to deleting one occurence of s_a and s_b respectively and then adding one occurence of $s_a + s_b$.

Suppose that a and b are from the same cycle of size s and they are d edges apart. Then this operation split the cycle into two smaller cycles of size d and $s - d$.



In our cycle set, this is equivalent to deleting one occurence of s and then adding one occurence of d and $s - d$ respectively.

Now, at this stage, it would be helpful to analyze how the score of the permutation changes for each swap.

In the case where a and b are from different cycles, it is not possible for $s_a + s_b = 1$. So the score of the permutation will decrease by one for each s_a and s_b that is equal to 1 respectively.

In the case where s_a and s_b are from the same cycle, it is not possible for $s = 1$. So the score of the permutation will increase by one for each d and $s - d$ that is equal to 1 respectively. Note that $d = 1$ or $s - d = 1$ only when a and b are adjacent to each other.

The only time when Bob is forced to pick a and b such that $d = s - d = 1$ is when $m = 2$. Otherwise, there is an edge connecting another vertex c to either a or b. WLOG c connects to a. Then Bob can pick to swap a and c instead so that the score of the permutation does not increase.

So let us get the pesky $m = 2$ case out of the way first.

Case 2: $m = 2$

Alice can make the score of the permutation n is at most n moves. This case is trivial because Bob has no choice in which edge to select as there is only 1 edge. So Alice is allowed to arbitrarily swap any pair of characters in 1 move. ■

Now, we can assume that $m > 2$ now.

We will also split the remaining graphs into the following 4 types:

- A: Graphs where there exists a vertex with degree at least 3
- L: Line graphs
- C1: Cycle graphs with odd number of vertices
- C0: Cycle graphs with even number of vertices

We will start proving the upper bounds for all graph types.

Upper Bounds

In this section, Bob is banned from choosing a and b such that they form a 2-cycle. The upper bounds in this variant will not be lower than the original problem.

By banning such moves from Bob, we can assume that the score will increase by at most one after each move.

So if we can prove that for all permutation with score k, Bob can always find a move that does not increase the cost of the permutation, then Alice will not have a strategy to increase the score of the permutation past k.

Lemma 1: If $k \geq n - m + 1$, then Bob can find an edge that does not increase the score.

Proof of lemma 1:

Since $k \geq n - m + 1$, one of $i_1, i_2, \ldots, i_m$ must be a self-loop. Suppose i_u is a self-loop. Then there must be an edge (u, v) in G and we can swap i_u and i_v. Since i_u is a self-loop, i_u and i_v are in different cycles and the cost of the permutation cannot increase (and it will actually decrease). ■

Because of the lemma 1, if the initial score of the permutation is already greater than $n - m + 1$, Alice should immediately end the game. Otherwise, she could try to increase the score to $n - m + 1$.

This is the only upper bound proof that is shared between graph types. We will now proceed with case work.

We will denote s_{initial} and s_{final} as the initial score and the final score so that we can give an explicit formula for s_{final} in terms of s_{initial} for each case. Very surprisingly, s_{final} only depends on s_{initial} for most of the graph types.

Using these variables, lemma 1 gives us the inequality

$$s_{\text{final}} \leq \begin{cases} s_{\text{initial}}, & \text{if } s_{\text{initial}} \geq n - m + 1 \\ n - m + 1, & \text{otherwise} \end{cases}$$

Case A: Graphs where there exists a vertex with degree at least 3

Bob has a strategy so that the score of the permutation in each move never increases.

Suppose the indices $(i_1, i_2, \ldots, i_m)$ that are chosen by Alice are not all in the same cycle. Then since G is connected, we are able to pick some (u, v) such that u and v belong to different cycles. Then Bob will swap i_u and i_v. This move cannot increase the score.

Otherwise, the indices $(i_1, i_2, \ldots, i_m)$ all belong to the same cycle. Since some vertex u has $\deg(u) \geq 3$, and there are only at most two vertices that adjacent to u in the permutation graph. Pick v as any vertex that is not adjacent to u and swap i_u and i_v.

$$s_{\text{final}} \leq s_{\text{initial}}$$

Case L: Line Graphs

The inequality implied by lemma 1 is actually the tightest possible.

$$s_{\text{final}} \leq \begin{cases} s_{\text{initial}}, & \text{if } s_{\text{initial}} \geq n - m + 1 \\ n - m + 1, & \text{otherwise} \end{cases}$$

Case C1: Cycle graph with odd number of vertices

WLOG the edges of G are $\{(0,1),(1,2),\ldots,(m-2,m-1),(m-1,0)\}$

Let O be the number of odd cycles in the cycle set initially.

Firstly, we will show that Bob has a strategy to never increase the number of odd cycles in the cycle set. This is important because self-loops are odd cycles themselves. So if Bob always ensures that the number of odd cycles in the cycle never increases, we have the bound $s_{\text{final}} \leq O$.

Suppose the indices $(i_1, i_2, \ldots, i_m)$ Alice chose such that they are not all in the same cycle. Then since G is connected, we are able to pick some (u, v) such that u and v belong to different cycles. Then Bob will swap i_u and i_v. Since this move merges two cycles, it cannot increase the number of odd cycles.

Otherwise, the indices $(i_1, i_2, \ldots, i_m)$ all belong to the same cycle. We will delete a cycle of size s and add cycles of sizes d and $s - d$. The only case where the number of odd cycles increase is when s is even and d is odd. We will prove that when all indices Alice chooses belongs to an even cycle, we can split the graph with even d.

The proof is a simple parity argument. Since the cycle in the permutation is even, we can bicolor it with black and white. Now the vertices of G are also bicolored based on which vertex it is assign to in $(i_1, i_2, \ldots, i_m)$. We want to choose a pair which has even d, which corresponds to adjacent vertices with the same color. But it is impossible for all adjacent vertices to have different colors, or else we can bicolor C_n, where n is odd. But this is clearly false.

Now let S be the minimum number such that the sum of sizes of S odd cycles and all even cycles in the cycle set is at least m. We actually have a stronger bound of $s_{\text{final}} \leq O - 2\lfloor \frac{S}{2} \rfloor$.

As we have seen previously, if $(i_1, i_2, \ldots, i_m)$ belong to different cycles, Bob will not increase the score. If $(i_1, i_2, \ldots, i_m)$ belongs to a single even cycle, then Bob can split it into two smaller even cycles. So the only way that Alice can force Bob to increase the score is when $(i_1, i_2, \ldots, i_m)$ belongs to a single odd cycle.

Now, before Alice is able to play a turn where $(i_1, i_2, \ldots, i_m)$ belongs to a single odd cycle, she will only be able to play moves of the previous two forms.

Let's see what Alice can do with the previous two moves:

1. Join an even and odd cycle
2. Join two even cycles
3. Join two odd cycles
4. Split an even cycle into two even cycles

Moves (2) and (4) will not change S.
For move (1), S cannot decrease. We can show by splitting into 3 cases.

- Case 1: the odd cycle is part of the S cycles. No change is made to the choice of S cycles, S remains the same
- Case 2: the odd cycle was not part of the S cycles, and after joining it becomes larger than the original S-th largest odd cycle. Let the size of the even cycle that was joined be e. The sum of sizes of even cycles decreases by e, and the sum of sizes of the $S-1$ largest odd cycles increases by at most e, hence the new value of S cannot be smaller than its original value.
- Case 3: the odd cycle was not part of the S cycles, and after joining it still is not. The sum of sizes of the S odd cycles remains the same, but the sum of sizes of all even cycles decreases. S will either stay the same or increase

For move (3), S will decrease by at most 2, and the total number of odd cycles will decrease by 2.

When $(i_1, i_2, \ldots, i_m)$ belongs to a single odd cycle, $S \leq 1$.
For S to reach its final value, move (3) has to be done at least $\lfloor \frac{S}{2} \rfloor$ times, which will decrease the number of odd cycles by $2\lfloor \frac{S}{2} \rfloor$, hence resulting in the final score of $O - 2\lfloor \frac{S}{2} \rfloor$.

$$s_{\text{final}} \leq \begin{cases} s_{\text{initial}}, & \text{if } s_{\text{initial}} \geq n-m-1 \\ O - 2\lfloor \frac{S}{2} \rfloor, & \text{otherwise} \end{cases}$$

Case C0: Cycle graph with even number of vertices

We will show that if $k = n - m - 1$, then Bob has a strategy to not increase the score of the permutation.

If Alice picks $(i_1, i_2, \ldots, i_m)$, then i_k and $i_{k \bmod m+1}$ must be adjacent to each other. If they were not, then Bob will choose those pair and the score will not increase.

This means that for the score to increase, there must be a cycle of size m. But we have that $k = n - m - 1$, so there are exactly $m + 1$ vertices that are not self-loops. If out of those $m + 1$ vertices, m are used to form a cycle of size m, then the remaining vertex will only be a self-loop, which contradicts $k = n - m - 1$. So a cycle of size $m + 1$ existing is not possible in the first place.

$$s_{\text{final}} \leq \begin{cases} s_{\text{initial}}, & \text{if } s_{\text{initial}} \geq n-m+1 \\ n-m+1, & \text{if } s_{\text{initial}} = n-m \\ n-m-1, & \text{otherwise} \end{cases}$$

Lower Bounds

Time to prove that all inequalities in the previous section are tight. We will explicitly construct a possible strategy for Alice, where each strategy uses at most $3n$ moves.

Case A: Graphs where there exists a vertex with degree at least 3

Alice simply ends the game on the first move.

Therefore, $s_{\text{final}} \geq s_{\text{initial}}$ in at most $3n$ moves.

Case big: $s_{\text{initial}} \geq n - m + 1$

Alice simply ends the game on the first move.

Therefore, $s_{\text{final}} \geq s_{\text{initial}}$ in at most $3n$ moves.

Case L: G is a line and $s_{\text{initial}} < n - m + 1$

Suppose the current non self-loop cycles are $(c_{1,1}, c_{1,2}, \ldots, c_{1,s_1}), (c_{2,1}, c_{2,2}, \ldots, c_{2,s_2}), \ldots, (c_{k,1}, c_{k,2}, \ldots, c_{k,s_k})$. Then Alice will pick the first m elements of these in order. i.e. $c_{1,1}, c_{1,2}, \ldots, c_{1,s_1}, c_{2,1}, c_{2,2}, \ldots, c_{2,s_2}, \ldots, c_{i,1}, c_{i,2}, \ldots, c_{i,j}$.

Then each operation Bob can do would correspond to swapping adjacent elements or swapping elements from different cycles. Let C denote the current number of cycles.

In the former case, the score increases by one and C increases by one too. In the latter case, C decreases by one.

Since in the final state where the score is $n - m + 1$, we have $C \geq n - m + 2$, it is clear that the number of moves required is $C_{\text{initial}} - C_{\text{final}} + 2$.

$$\begin{aligned} C_{\text{initial}} - C_{\text{final}} + 2 \cdot (\text{score}_{\text{final}} - \text{score}_{\text{initial}}) &\leq n - (n - m + 2) + 2 \cdot (n - m + 1 - 0) && (1) \\ &= m - 2 + 2(n - m + 1) && (2) \\ &= 2n - m \leq 3n && (3) \end{aligned}$$

Therefore, $s_{\text{final}} \geq n - m + 1$ in at most $3n$ moves.

Case C0 edge case: G is an even cycle and $s_{\text{initial}} = n - m$

Suppose the non self-loop cycles are $(c_{1,1}, c_{1,2}, \ldots, c_{1,s_1}), (c_{2,1}, c_{2,2}, \ldots, c_{2,s_2}), \ldots, (c_{k,1}, c_{k,2}, \ldots, c_{k,s_k})$.

In this case, $s_1 + s_2 + \ldots + s_k = m$. So in each move when Alice picks the first m elements of these in order, she actually picks all of the non-self-loops, that is $c_{1,1}, c_{1,2}, \ldots, c_{1,s_1}, c_{2,1}, c_{2,2}, \ldots, c_{2,s_2}, \ldots, c_{k,1}, c_{k,2}, \ldots, c_{k,s_k}$.

Then again, each operation Bob can do would correspond to swapping adjacent elements or swapping elements from different cycles.

From the previous analysis, the number of moves required is $\leq 3n$.

Therefore, $s_{\text{final}} \geq n - m + 1$ in at most $3n$ moves.

Case C1: G is an odd cycle and $s_{\text{initial}} < n - m + 1$

Again, define S as the minimum number such that the sum of sizes of S odd cycles and all even cycles in the cycle set is at least m.

The following is a brief overview of our strategy:

Phase 1: Make a cycle of size at least m.
Phase 2: Produce a self-loop from each odd cycle.

Do note that in any phase of our algorithm it is possible that there will already be at least $O - \lfloor \frac{S}{2} \rfloor$ counts of 1. In that case, we will terminate our algorithm.

Furthermore, let T be the minimum number of cycles such that S of these cycles are odd and the sum of cycles is at least m. Also, let X be the number of self-loops.

For phase 1, let the cycles $(c_{1,1}, c_{1,2}, \ldots, c_{1,s_1}), (c_{2,1}, c_{2,2}, \ldots, c_{2,s_2}), \ldots, (c_{k,1}, c_{k,2}, \ldots, c_{k,s_k})$ be the T cycles such that out of S of them are odd and the sum of cycle sizes at least m. As above, Alice will pick the first m elements of these in order. i.e.
$c_{1,1}, c_{1,2}, \ldots, c_{1,s_1}, c_{2,1}, c_{2,2}, \ldots, c_{2,s_2}, \ldots, c_{i,1}, c_{i,2}, \ldots, c_{i,j}$.

Let us analyze the effect of each possible case on $O - \lfloor \frac{S}{2} \rfloor$, T and X.

- If Bob merges two even cycles, O and S remains unchained. T decreases by 1. X remains unchanged.
- If Bob merges one even and one odd cycle, O and S remains unchained while T decreases by 1. X remains unchanged.
- If Bob merges two odd cycles, O decreases by 2 while S decreases by 2. Therefore, $O - 2\lfloor \frac{S}{2} \rfloor$ is still constant. T decreases by 1. X remains unchanged.
- If Bob splits an even cycle of size s into $[1, s-1]$. O increases by one while S increases by at most 1. $O - 2\lfloor \frac{S}{2} \rfloor$ will not decrease. T increases by at most 1. X increases by 1.
- If Bob splits an odd cycle of size s into $[1, s-1]$. O decreases by one while S decreases by at most 2. $O - 2\lfloor \frac{S}{2} \rfloor$ will not decrease. T increases by at most 1. X increases by 1.

In all cases, $O - 2\lfloor \frac{S}{2} \rfloor$ will not decrease.

In the first 3 cases, T decreases by 1 while X remains unchanged.

In the last 2 cases, T increases by at most 1 while X increases by 1.

In both cases, $\Delta T - 2\Delta X$ decreases by at least 1.

Therefore, the numeber of moves for phase 1 is bounded by $T_{\text{final}} - T_{\text{initial}} + 2 \cdot (X_{\text{final}} - X_{\text{initial}})$.

Since $T \leq m$, the number of moves is bounded by $2 \cdot (X_{\text{final}} - X_{\text{initial}}) + m$.

For phase 2, observe that if we have a cycle of size $s > m$, then we can force Bob to split it into cycles of sizes $[1, s-1]$ or $[2, s-2]$.

It is sufficient to find a sequence of **distinct** integers $x_1, x_2, \ldots, x_m$ between 1 and s (inclusive) such that $|x_i - x_{i \bmod n+1}| = 1$ or $|x_i - x_{i \bmod n+1}| = 2$.

The construction $x = [1, 3, 5, \ldots, m, m-1, m-3, \ldots, 2]$ works.

Therefore, if we have a odd cycle of size $s \geq m$, we can produce a self-loop from it using at most $\frac{s-m}{2} + 1 \leq m$ moves.

Now, we will handle the small odd cycles.

Since we have gone through phase 1, we definitely will have a cycle of size $e \geq m$. If this cycle was odd size, we would have used the above process to obtain an even cycle of size at least $m - 1$ too. Therefore, we can be sure that we would have an even cycle of size at least $m - 1$ at this stage.

For each odd cycle of size $s \geq 3$, we will first join the odd cycle with our even cycle it using 1 move. Then using the move above, we will produce a self-loop along with an even cycle of size $e' \geq m - 1$ using $\frac{e+s-e'}{2} + 1$ additional moves.

Therefore, the number of moves in total is bounded by $\frac{n}{2} + 2(X_{\text{final}} - X_{\text{initial}})$.

Therefore, the total number of moves is $2(X_{\text{final}} - X_{\text{initial}}) + \frac{n}{2} + m$. Now, $X_{\text{final}} - X_{\text{initial}}$ is at most the number of odd cycles of size greater than 1. This is at most $\frac{n}{3}$.

So the total number of moves is bounded by $2 \cdot \frac{n}{3} + \frac{n}{2} + m \leq 3n$.

$$s_{\text{final}} \geq O - 2\lfloor \frac{S}{2} \rfloor$$

Case C0: G is an even cycle and $s_{\text{initial}} < n - m + 1$

Observe that if we have a cycle of size $s \geq m + 2$, then we can force Bob to split it into cycles of sizes $[1, s-1]$ or $[m, s-m]$.

It is sufficient to find a sequence of **distinct** integers $x_1, x_2, \ldots, x_m$ between 1 and s (inclusive) such that $|x_i - x_{i \bmod n+1}| = 1$ or $|x_i - x_{i \bmod n+1}| = s - m$.

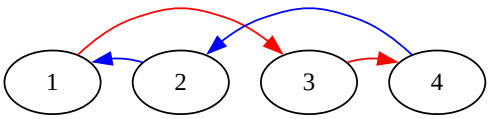
Although since m is small, it is possible to write dp. There is a simple $O(m)$ construction.

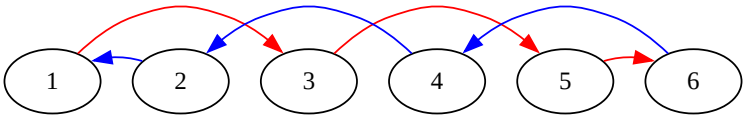
The following is pseudocode to construct $x_1, x_2, \ldots, x_m$:

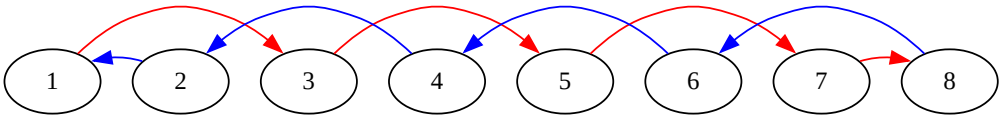
```
x=[]
curr=1

x.add(curr)

for i in 1..m/2-1:
    if (curr mod (2(s-m)-2) == 1): curr += s-m
    else: curr++
    x.add(curr)

curr++
x.add(curr)

curr -= s-m
x.add(curr)

for i in 1..m/2-2:
    if (curr mid (2(s-m)-2) == 2): curr -= s-m
    else: curr--
    x.add(curr)
```

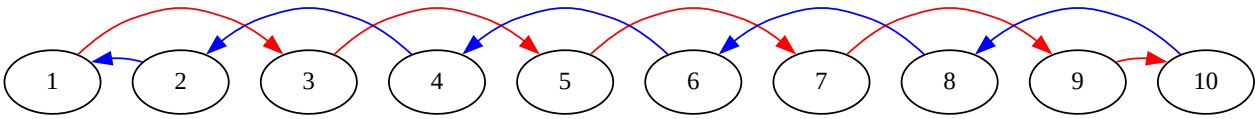
Below are illustrations of its output.

For $s - m = 2$:

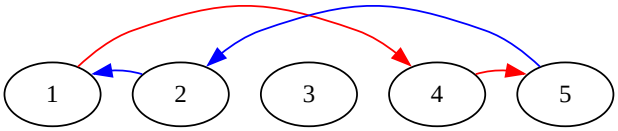


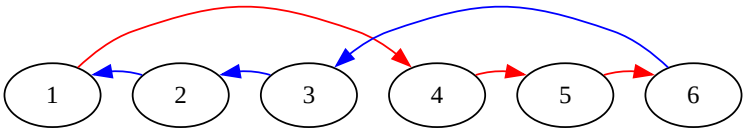


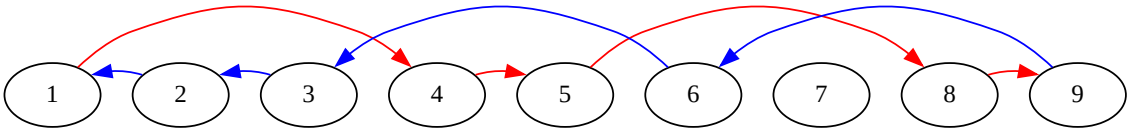


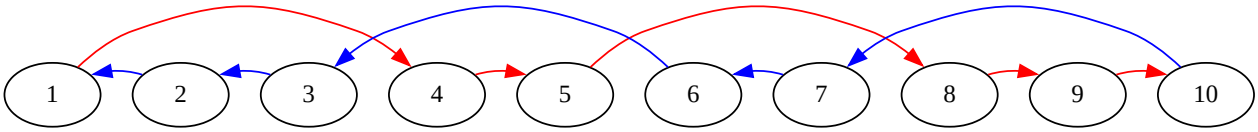


For $s - m = 3$:

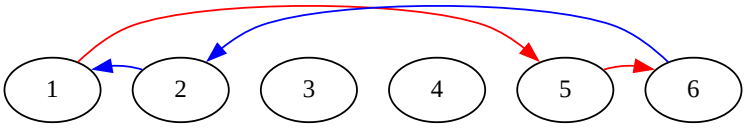


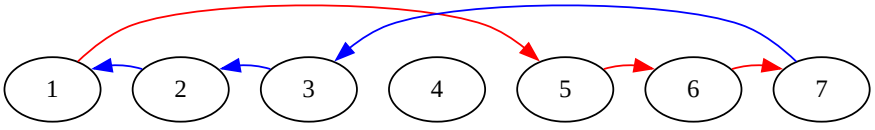


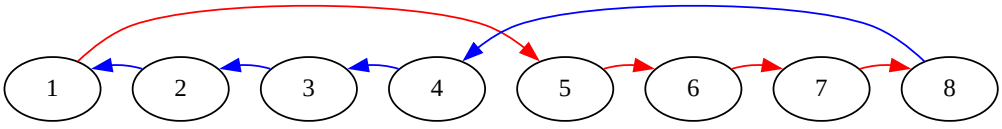


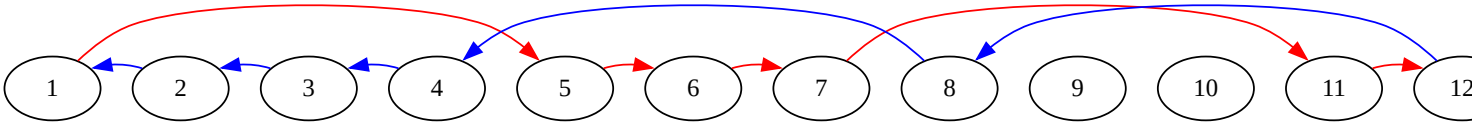


For $s - m = 4$:









The following is a brief overview of our strategy:

Phase 1: Make the cycle set $T \cup \{2, \ldots, 2, 1, \ldots, 1\}$, where $\min(T) \geq 3$ and $\sum T \leq m + 1$.
Phase 2: Make the cycle set $\{s, 1, \ldots, 1\}$, where $s \leq 2m - 1$.
Phase 3: Perform more operations until there are at least $n - m - 1$ counts of 1.

Do note that in any phase of our algorithm it is possible that there will already be at least $n - m - 1$ counts of 1. In that case, we will terminate our algorithm.

For phase 1, Alice will use the following algorithm:

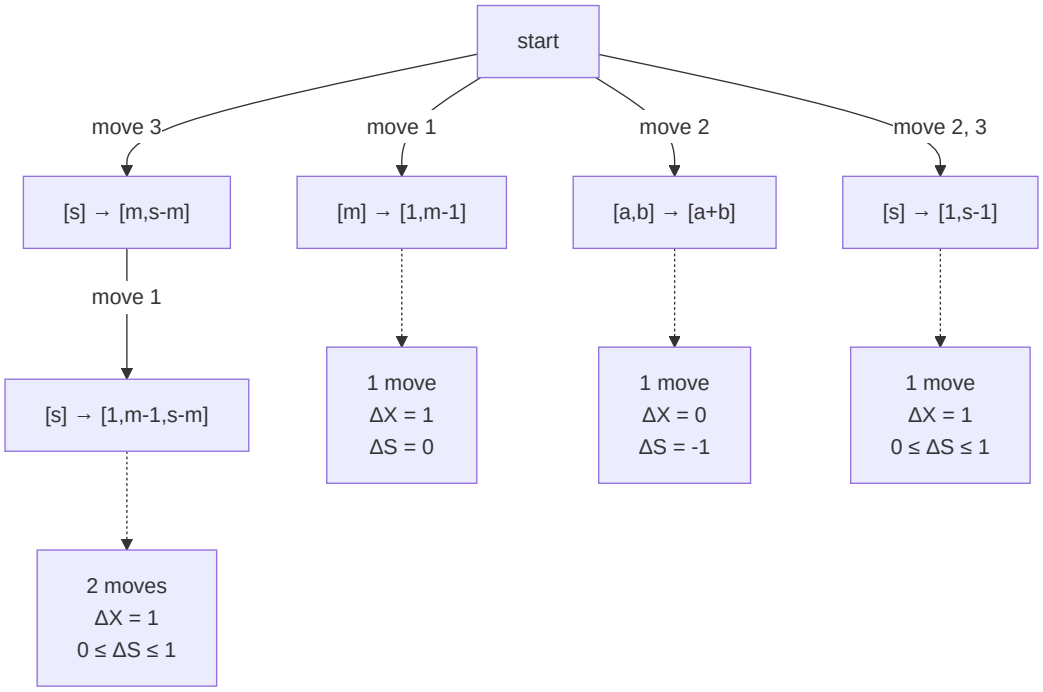
1. If there is a cycle of size exactly m, Alice will force Bob to do $[m] \to [1, m - 1]$.
2. If there isn't a cycle of size at least $m + 2$, Alice will try to make a cycle of size at least $m + 2$ by combining cycles of size at least 3 together. Note that Alice can only choose a set of cycles whose sum is at least m and then force Bob to merge two of these cycles or split one of the cycles of size s into $[1, s - 1]$.
3. If there is a cycle of size at least $s \geq m + 2$, Alice can use the above construction to force Bob to split it into $[1, s - 1]$ or $[m, s - m]$.
4. Otherwise, we have completed phase 1 and the cycle set is of the form $T \cup \{2, \ldots, 2, 1, \ldots, 1\}$, where $\min(T) \geq 3$ and $\sum T \leq m + 1$.

Let us bound the number of moves Alice will make for phase 1.

Let S be the minimum number of cycles of length at least 3 so that their sum is at least $m+2$. If S is undefined by this definition, we will define it as 1 as convention (it will make sense in our analysis).

Furthermore, let X be the number of self-loops.

The possible cases are given by:



In all cases, $3X - S$ increases by at least the number of moves.

In the worst case, the number of moves for phase 1 is at most by $3 \cdot (X_{\text{final}} - X_{\text{initial}}) - (S_{\text{final}} - S_{\text{initial}})$.

Since $1 \leq S \leq \lceil \frac{m+2}{3} \rceil$, it is clear that this is at most $3 \cdot (X_{\text{final}} - X_{\text{initial}}) + \lceil \frac{m+2}{3} \rceil - 1$.

For phase 2, Alice will use the following algorithm:

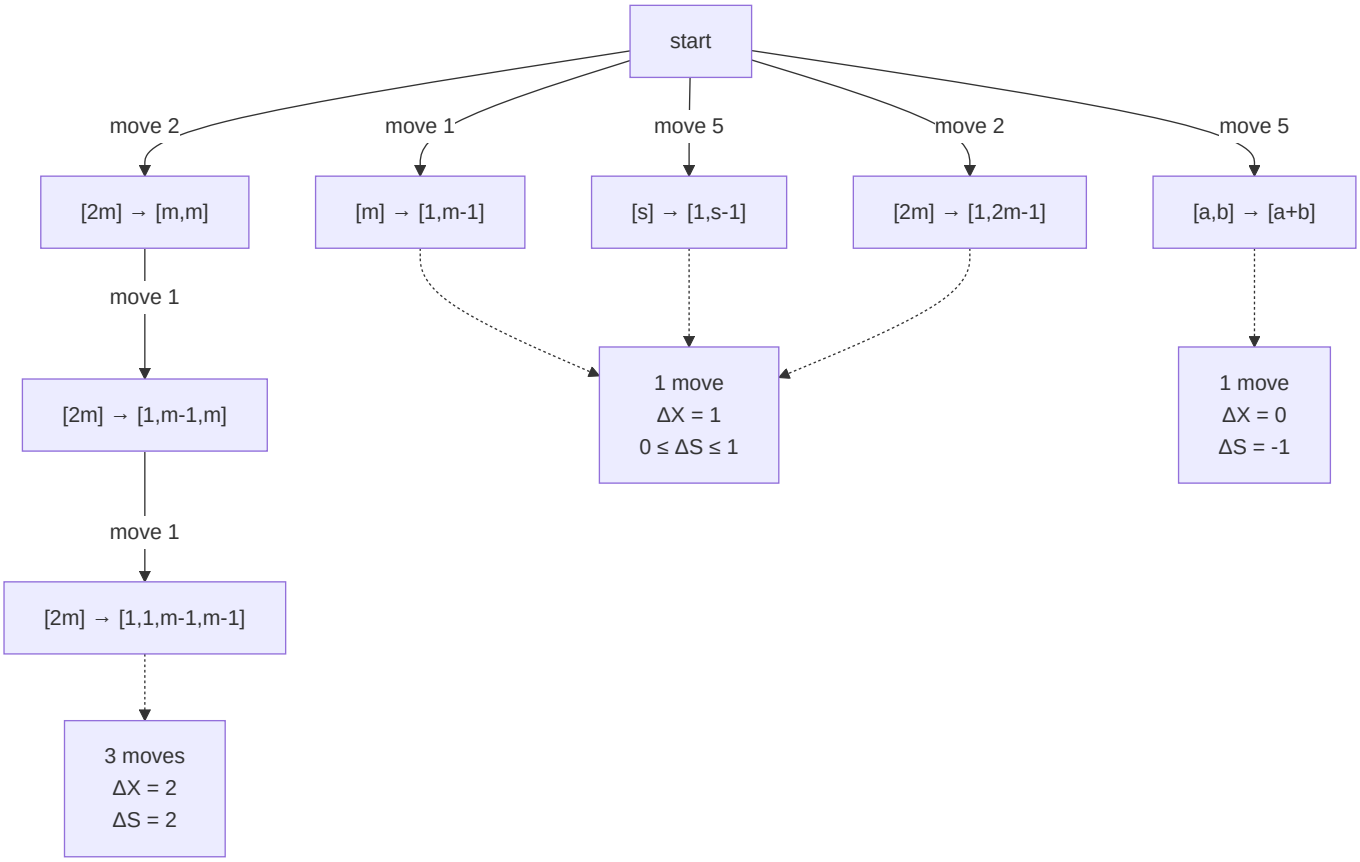
1. If there is a cycle of size exactly m, Alice will force Bob to split m into $[1, m-1]$.
2. If there is a cycle of size exactly $2m$, Alice will force Bob to split $2m$ to either $[1, 2m-1]$ or $[m, m]$.
3. If $[2m-1, 2]$ is a subset of the cycle set, Alice will force Bob to split $2m-1$ to either $[1, 2m-2]$ or $[m-1, m]$.
4. If $[m-1, m-1, 2]$ is a subset of the cycle set, Alice can force Bob to change $[m-1, m-1]$ into $[2m-2]$ or $[1, m-2, m-1]$.
5. If there isn't a cycle of size at least $2m-1$, Alice will try to make a cycle of size at least $2m-1$ by combining cycles of size at least 2 together. Note that Alice can only choose a set of cycles whose sum is at least m and then force Bob to merge two of these cycles or split one of the cycles of size s into $[1, s-1]$.
6. Otherwise, we have completed phase 2 and the cycle set is of the form $\{s, 1, \ldots, 1\}$, where $s \leq 2m-1$.

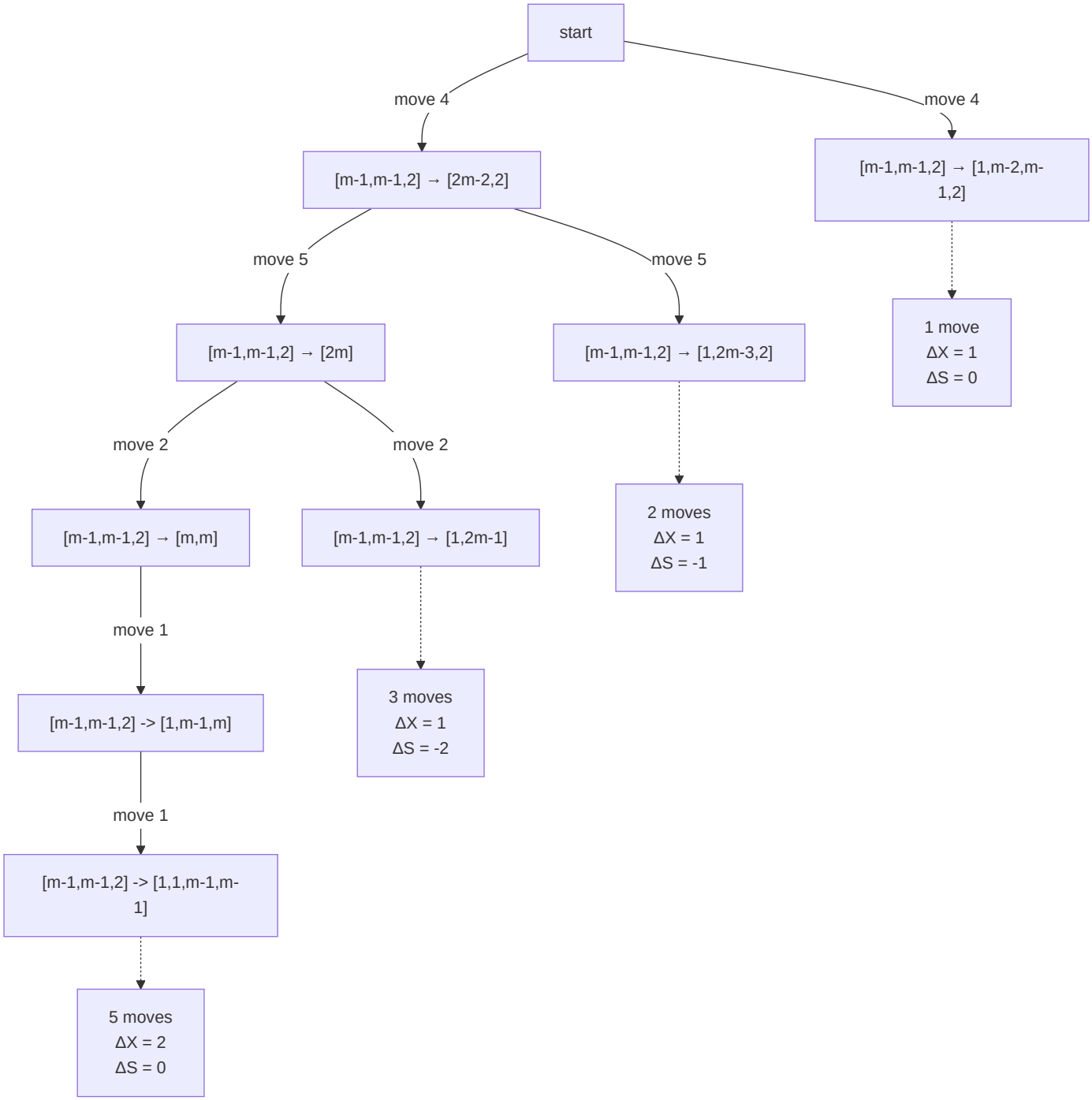
Do note that the maximum cycle size will never exceed $2m$ following our algorithm.

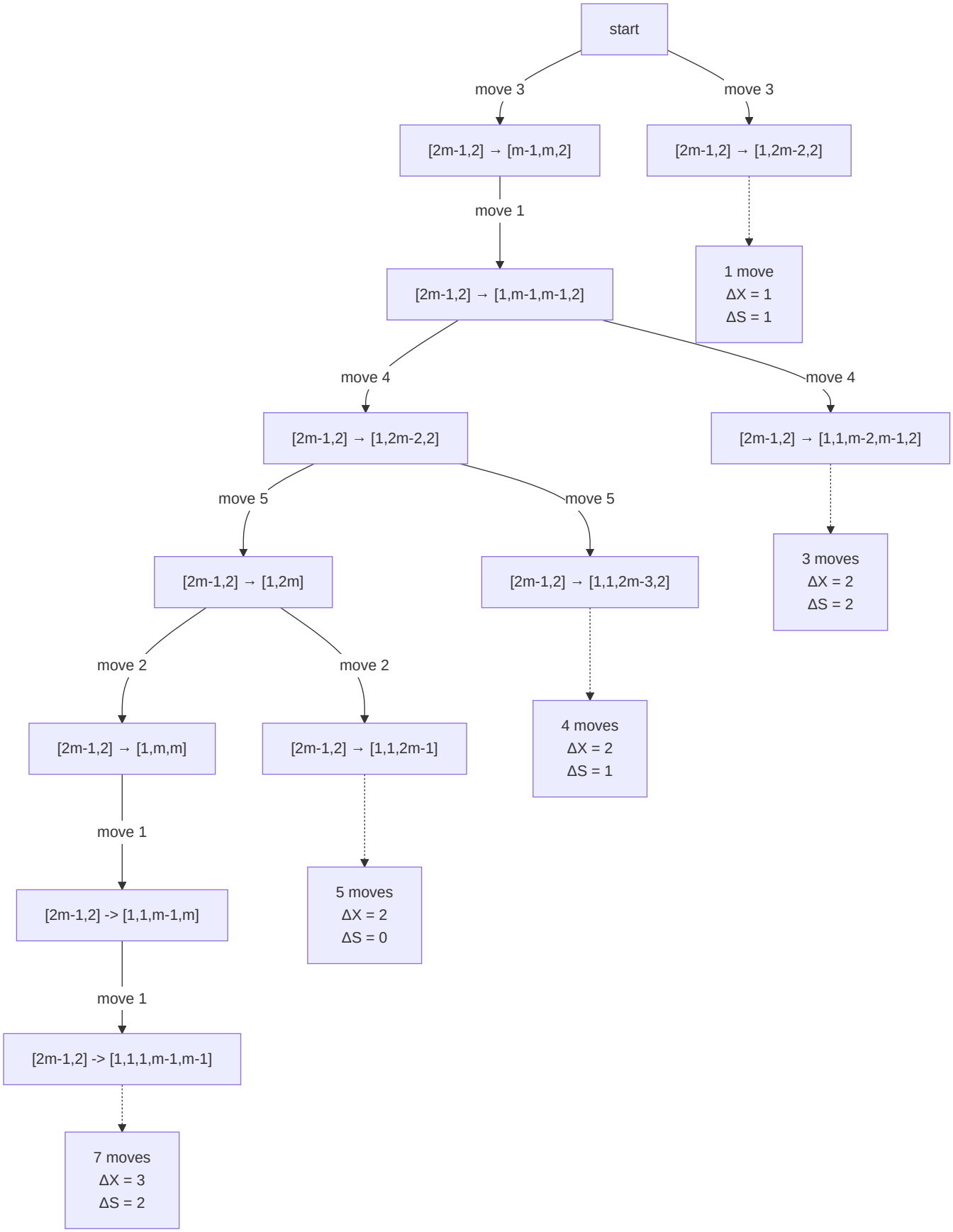
Let S be the minimum number of cycles of length at least 2 so that their sum is at least $2m-1$. If S is undefined by this definition, we will define it as 1 as convention (it will make sense in our analysis).

Furthermore, let X be the number of self-loops.

The possible cases are given by:


start
move 2
move 1
move 5
move 2
move 5
[2m] → [m,m]
[m] → [1,m-1]
[s] → [1,s-1]
[2m] → [1,2m-1]
[a,b] → [a+b]
move 1
[2m] → [1,m-1,m]
move 1
[2m] → [1,1,m-1,m-1]
3 moves
ΔX = 2
ΔS = 2
1 move
ΔX = 1
0 ≤ ΔS ≤ 1
1 move
ΔX = 0
ΔS = -1


start
move 4
move 4
[m-1,m-1,2] → [2m-2,2]
[m-1,m-1,2] → [1,m-2,m-1,2]
move 5
move 5
[m-1,m-1,2] → [2m]
[m-1,m-1,2] → [1,2m-3,2]
1 move
ΔX = 1
ΔS = 0
move 2
move 2
[m-1,m-1,2] → [m,m]
[m-1,m-1,2] → [1,2m-1]
2 moves
ΔX = 1
ΔS = -1
move 1
3 moves
ΔX = 1
ΔS = -2
[m-1,m-1,2] -> [1,m-1,m]
move 1
[m-1,m-1,2] -> [1,1,m-1,m-1]
5 moves
ΔX = 2
ΔS = 0



In all cases, $3X - S$ increases by at least the number of moves.

In the worst case, the number of moves for phase 2 is at most $3 \cdot (X_{\text{final}} - X_{\text{initial}}) - (S_{\text{final}} - S_{\text{initial}})$.

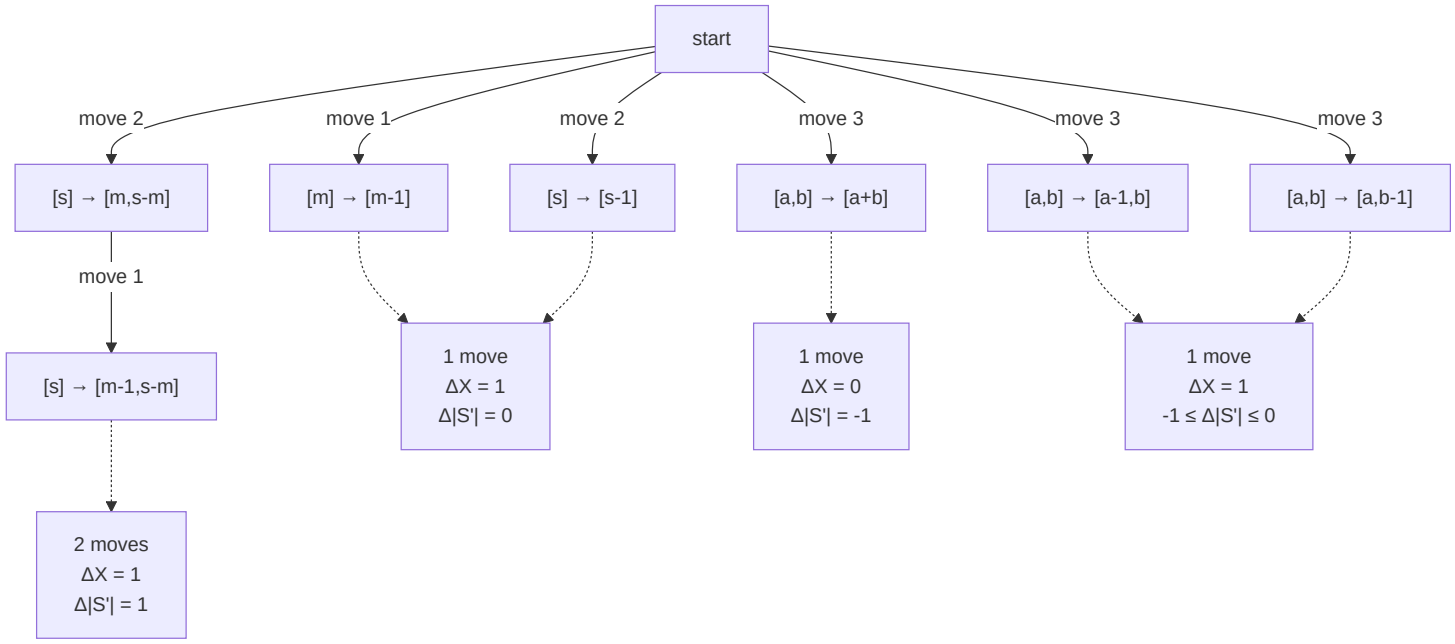
Since $1 \leq S \leq \lceil \frac{2m-1}{2} \rceil$, it is clear that this is at most $3 \cdot (X_{\text{final}} - X_{\text{initial}}) + \lceil \frac{2m-1}{2} \rceil - 1$

For phase 3, Alice will use the following algorithm:

Let S' be the cycle set after removing all the 1s.

1. If m is in S', split it into $[m-1]$.
2. If $S' = [s]$ where $s \geq m+2$, then Alice will force Bob to split it into $[s-1]$ or $[m, s-m]$.
3. If $S' = \{a, b\}$ where $a+b \geq m+2$, try to combine $a+b$. This can result in $S' = \{a+b\}$ or $S' = \{a-1, b\}$ or $S' = \{a, b-1\}$. Note that in the second case, it is possible that $a-1=1$, so S' would actually be only $\{b\}$.
4. Otherwise, the sum of S' is at most $m+1$. And we have reached the maximum score possible.

The possible cases are given by:



In all cases, $3X - |S'|$ increases by at least the number of moves.

In the worst case, the number of moves for phase 3 is at most $3 \cdot (X_{\text{final}} - X_{\text{initial}}) - (|S'|_{\text{final}} - |S'|_{\text{initial}})$.

Since $1 \leq |S'| \leq 2$, it is clear that this is at most $3 \cdot (X_{\text{final}} - X_{\text{initial}}) + 1$.

In summary, this strategy uses at most:

- $3 \cdot (X_{\text{final}} - X_{\text{initial}}) + \lceil \frac{m+2}{3} \rceil - 1$ moves in phase 1.
- $3 \cdot (X_{\text{final}} - X_{\text{initial}}) + \lceil \frac{2m-1}{2} \rceil - 1$ moves in phase 2.
- $3 \cdot (X_{\text{final}} - X_{\text{initial}}) + 1$ moves in phase 3.

Therefore, in total we use at most $3 \cdot (X_{\text{final}} - X_{\text{initial}}) + \lceil \frac{m+2}{3} \rceil + \lceil \frac{2m-1}{2} \rceil - 1 \leq 3 \cdot (X_{\text{final}} - X_{\text{initial}}) + 3(m-1)$ moves.

Note that $0 \leq X \leq n-m+1$, so $(X_{\text{final}} - X_{\text{initial}}) + 3(m-1) \leq 3(n-m+1) + 3(m-1) = 3n$, as required.

Therefore, $s_{\text{final}} \geq n-m+1$ in at most $3n$ moves.

Summary

The following are the final scores for each case

2: $m = 2$

$$s_{\text{final}} = n$$

A: Graphs where there exists a vertex with degree at least 3

$$s_{\text{final}} = s_{\text{initial}}$$

L: Line graphs

$$s_{\text{final}} = \begin{cases} s_{\text{initial}}, & \text{if } s_{\text{initial}} \geq n - m + 1 \\ n - m + 1, & \text{otherwise} \end{cases}$$

C1: Cycle graphs with odd number of vertices

Let O be the number of odd cycles in the cycle set initially and S be the minimum number such that the sum of sizes of S odd cycles and all even cycles in the cycle set is at least m.

$$s_{\text{final}} = \begin{cases} s_{\text{initial}}, & \text{if } s_{\text{initial}} \geq n - m - 1 \\ O - 2\lfloor \frac{S}{2} \rfloor, & \text{otherwise} \end{cases}$$

C0: Cycle graphs with even number of vertices

$$s_{\text{final}} = \begin{cases} s_{\text{initial}}, & \text{if } s_{\text{initial}} \geq n - m + 1 \\ n - m + 1, & \text{if } s_{\text{initial}} = n - m \\ n - m - 1, & \text{otherwise} \end{cases}$$

Proof that at least $3n$ moves are required for Alice to get the maximum score

We will consider the case where G is an even cycle of size m.

Firstly, our only restriction on Bob is that he will never make a move that creates a self-loop whenever possible. That means the only way for Alice to create self-loops is to split a cycle of size m into cycles of sizes $m - 1$ and 1.

Other than the above restriction, we can pretend that Alice is allowed to force Bob to perform any moves that she likes.

Now, consider the case where the original cycle set contains km cycles of size $m - 1$ and one cycle of size $m + 1$. Then we will show that the minimum number of moves to reach a state where the cycle set contains $(m - 1)mk$ self-loops and one cycle of size $m + 1$ is $(3m - 4)mk$.

Note that it is possible that there might exist an optimal sequence of moves using O that produces $(m - 1)mk$ self-loops but the rest of the vertices may not be arranged into a single cycle of size $m + 1$.

But Alice can use m extra moves to combine the rest of the cycles into one cycle of size $m + 1$. Therefore, we have that $O + m \geq (3m - 4)mk$.

The total size of all cycles is $n = (m - 1)mk + m + 1$.

Therefore, $\frac{O}{n} \geq \frac{(3m-4)mk-m}{(m-1)mk+m+1}$.

It is clear that $\frac{O}{n} \geq 3 - \epsilon$ for any $\epsilon > 0$ by choosing sufficiently large m and k.

The following is the proof that we need at least $(3m - 4)mk$ moves.

So we consider the cycle set C, the following is the move that Alice can perform:

- Choose different $i, j \in C$. Then delete both i and j from C and add $i + j$ into C.
- Choose $i + j \in C$ such that $i, j \neq 1$ then delete $i + j$ from C and add both i and j into C.
- Delete m from C and add both 1 and $m - 1$ into C.

We will write the above in the following notation:

- $[i, j] \to [i + j]$.
- $[i + j] \to [i, j]$, $i, j \neq 1$.
- $[m] \to [1, m - 1]$.

Note that we want to transform $[m - 1] \cdot km \to [1] \cdot (m - 1)mk$.

Since the only move that produces self-loops is $[m] \to [m - 1, 1]$, we need to perform that move $(m - 1)mk$ times.

So it is sufficient to show that we need at least $(2m - 3)mk$ moves to perform $[m - 1] \cdot m^2k \to [m] \cdot (m - 1)mk$.

Now, let us not care about the restriction that $i, j \neq 1$ when we do the move $[i + j] \to [i, j]$. So the valid moves is simply:

- $[i, j] \to [i + j]$.

- $[i+j] \to [i,j]$.

Suppose we did some operations of the form $a_i \to b_i$ to get the overall effect of $[m-1] \cdot km \to [m] \cdot (m-1)mk$.

For two multisets S and T with the same sum, define $F(S,T)$ as the minimum number of "cuts" you have to do both S and T until they become the same multiset. Here, a "cut" means $[i+j] \to [i,j]$.

Then, a lower bound on the minimum of operations is clearly $F(\bigcup a, \bigcup b)$. An operation $[i,j] \to [i+j]$ corresponds to a cut in b and an operation $[i+j] \to [i,j]$ corresponds to a cut in a.

For example, suppose that we wants to do $[3,5] \to [1,1,6]$ and we performed the following moves:

- $[3,5] \to [3,2,3]$ ($a_1 = [5]$, $b_1 = [2,3]$)
- $[3,2,3] \to [2,6]$ ($a_2 = [3,3]$, $b_2 = [6]$)
- $[2,6] \to [1,1,6]$ ($a_3 = [2]$, $b_3 = [1,1]$)

Then, $\bigcup a = [5,3,3,2]$ and $\bigcup b = [2,3,6,1,1]$.

Lemma 2: $F([n-1] \cdot cn, [n] \cdot c(n-1)) = c(2n-3)$.

Lemma 3: $F(S,T) = F(S \cup \{x\}, T \cup \{x\})$.

From lemmas 2 and 3 (proven below), it is clear that $F(\bigcup a, \bigcup b) = k(2n-3)$. Therefore, we need at least $(2m-3)k$ moves to perform $[m-1] \cdot m^2k \to [m] \cdot (m-1)mk$ and therefore need at least $(3m-4)k$ moves to perform $[m-1] \cdot km \to [1] \cdot (m-1)mk$.

Although for the purposes of our prove, we only needed to show $F([n-1] \cdot cn, [n] \cdot c(n-1)) \geq c(2n-3)$, the other direction was trivial.

Proof of lemma 2:

First, let us show that $F([n-1] \cdot cn, [n] \cdot c(n-1)) \geq c(2n-3)$.

For each of the $c(n-1)$ occurence of $[n]$ on $[n] \cdot c(n-1)$, it must be cut at least once, or else it cannot possibly match with anything after performing some cuts to $[n-1] \cdot cn$ as all elements will have size of at most $n-1$, since they can only be formed by cutting $[n-1]$.

Now, the total elements of the resulting multisets must be same. After performing a cut on each of the $c(n-1)$ occurences of n on $[n] \cdot c(n-1)$, it will be split into $c(2n-2)$ elements.

For $[n-1] \cdot cn$ to have at $c(2n-2)$ elements, we must perform at least $c(n-2)$ cuts on it.

Therefore, we must perform at least $c(2n-3)$ cuts.

It is also not too difficult to find a construction using exactly $c(2n-3)$ cuts, proving $F([n-1] \cdot cn, [n] \cdot c(n-1)) \leq c(2n-3)$.

For each of the $c(n-1)$ occurence of n in $[n] \cdot c(n-1)$, we will cut it into $n-1$ and 1 using $c(n-1)$ cuts.

Then for only c occurences of $n-1$ in $[n-1] \cdot cn$, we will use $n-2$ cuts each to cut it into $[1] \cdot (n-1)$, using $c(n-2)$ cuts total.

Therefore, we have cut both arguments into $[n-1] \cdot c(n-1) + [1] \cdot (n-1)$ using a total of exactly $c(2n-3)$ cuts. ■

Proof of lemma 3:

It is clear that $F(S,T) \geq F(S \cup \{x\}, T \cup \{x\})$. Because any cutting scheme of (S,T) works for $(S \cup \{x\}, T \cup \{x\})$ by not cutting x.

Now, to prove that $F(S,T) \leq F(S \cup \{x\}, T \cup \{x\})$, we will proceed by contradiction.

Suppose we have a cutting scheme $(S \cup \{x_1\}, T \cup \{x_2\})$ that uses the least moves over all x. Here, we label the different x as x_1 and x_2 for clarity.

If some parts of x_1 gets matched with x_2, in the above cutting scheme, where the matching parts sum to y, we can find a cutting scheme $(S \cup \{x_1 - y\}, T \cup \{x_2 - y\})$ that uses strictly less moves, contradicting our assumption that our cutting scheme uses the least moves. So $y = 0$ must hold.

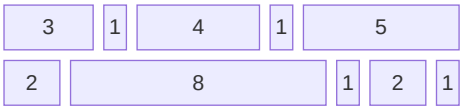
Therefore, in our cutting scheme, x_1 and x_2 must be matched with some cuts of T and S respectively.

Suppose x_1 and x_2 is cut into p and q pieces respectively. Then the matching parts of T and S are split p and q parts respectively. We will denote these parts as $t_1, t_2, \ldots, t_p$ and $s_1, s_2, \ldots, s_q$.

x_1 and x_2 has $p-1$ and $q-1$ respectively. But we can just perform at most $p-1$ and $q-1$ cuts on s and t respectively and none on x instead so that we can conclude that our cutting scheme must use as many cuts as $F(S,T)$.

Here is how we will perform our cuts on s and t. We simply line them up. If there is a cut on one side, then we will also cut the other side.

For example, if $s = [3, 1, 4, 1, 5]$ and $t = [2, 8, 1, 2, 1]$. Then we arrange them as follows:



Then we will make cuts to both arrays so that they become $[2, 1, 1, 4, 1, 1, 1, 2, 1]$. ■