# AtCoder Grand Contest 004 解説

writer : sugim48

## A : Divide a Cuboid

$A \times B \times C$ 個のブロックからなる直方体を 2 つの直方体へ分割し，2 つの直方体に含まれるブロックの個数の差を最小化する問題です．分割の方法をすべて試すと TLE してしまうので，より速い方法を考えなければなりません．

まず，$A$，$B$，$C$ のどれかが偶数の場合を考えます．例えば，$A$ が偶数であるとして，$A = 2a$ とおきます．この場合，$a \times B \times C$ の直方体 2 つへ分割すると，ブロックの個数の差は 0 になります．よって，答えは 0 です．

次に，$A$，$B$，$C$ がすべて奇数の場合を考えます．例えば，$A$ の方向に分割することにして，$A = 2a + 1$ とおきます．この場合，$a \times B \times C$ の直方体と $(a + 1) \times B \times C$ の直方体へ分割すると，ブロックの個数の差は $B \times C$ で最小になります．同様に，$B$ の方向に分割する場合の最小値は $C \times A$ であり，$C$ の方向に分割する場合の最小値は $A \times B$ です．よって，答えは $\min\{B \times C, C \times A, A \times B\}$ です．

## B : Colorful Slimes

魔法を唱える回数を $k$ に固定してみます．すると，最終的に色 $i$ のスライムが欲しい場合，色 $i$，$i-1$，...，$i-k$ のスライムのどれかを適切なタイミングで捕まえればよいです．ただし，色 1 のひとつ前は色 $N$ であるとします．例えば，$K = 2$ で最終的に色 3 のスライムが欲しい場合，次の 3 通りの方法があります．

- 魔法を唱える → 魔法を唱える → スライム 3 を捕まえる
- 魔法を唱える → スライム 2 を捕まえる → 魔法を唱える
- スライム 1 を捕まえる → 魔法を唱える → 魔法を唱える

どのタイミングでスライムを捕まえるかは自由なので，最も $a_i$ が小さいタイミングで捕まえればよいです．よって，最終的に色 $i$ のスライムを手に入れるためには，$\min\{a_i, a_{i-1}, \ldots, a_{i-k}\}$ 秒が掛かります．これを $b_i(k)$ 秒とおくと，全色のスライムを手に入れるためには，合計で $k \times x + \sum_i b_i(k)$ 秒が掛かります．

以上の考察より，$k$ は 0 から $N-1$ まで試せば十分であることが分かります．$k$ を 0 から $N-1$ まで全探索し，$k \times x + \sum_i b_i(k)$ の最小値を求めれば，それが答えです．しかし，各 $b_i(k)$ を計算するのに $\mathrm{O}(N)$ 時間掛けてしまうと，全体で $\mathrm{O}(N^3)$ 時間となり TLE してしまいます．ここで，$b_i(k) = \min\{b_i(k-1), a_{i-k}\}$ であることを用いると，各 $b_i(k)$ を $\mathrm{O}(1)$ 時間で計算できます．すると，全体で $\mathrm{O}(N^2)$ 時間となり，間に合います．

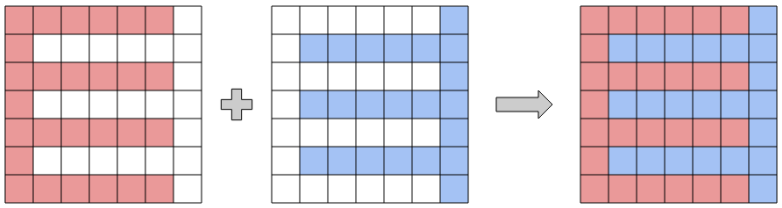
## C：AND Grid

与えられた紫色のマスの配置に対して，臨機応変に赤いマス目と青いマス目のペアを構成するのは大変です．ここでは，どのような紫色のマスの配置に対しても，機械的に赤いマス目と青いマス目のペアを構成できる方法を考えてみます．

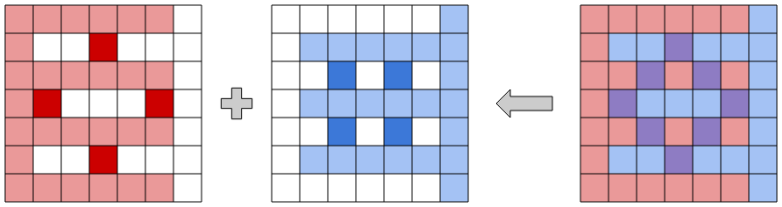
次のような条件を満たす赤いマス目と青いマス目のペアを準備しておくと，嬉しいことが分かります．

- 赤いマス目と青いマス目を重ねると，各マスは赤または青のどちらかである．
- 赤いマスも青いマスも上下左右に連結である．
- 赤いマス目において，最も外側のマス以外のマスをひとつ選んで赤く塗っても，赤いマスは上下左右に連結のままである．青いマス目においても同様．

例えば，次図のようなマス目のペアが条件を満たします．

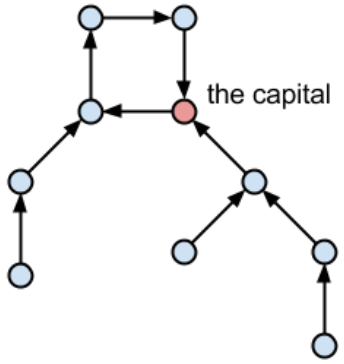


このようなマス目のペアを準備しておくと，どのような紫色のマスの配置に対しても，紫色の各マスに対応するマスを赤または青で塗ることで，機械的に赤いマス目と青いマス目のペアを構成できます（次図）．

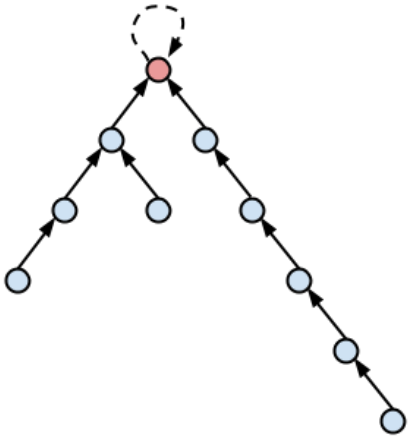


## D : Teleporter

各 $1 \leq i \leq N$ について，頂点 $i$ から頂点 $a_i$ へ有向辺を張ったグラフを考えます（次図）.「どの町から出発しても，テレポーターを何回か使うことで首都へ辿り着ける」という制約より，グラフは連結です．グラフの辺の行き先をいくつか変えて，「どの頂点から出発しても，辺をちょうど $K$ 回辿ると，最終的に首都にいる」という条件が成り立つようにするのが目標です．

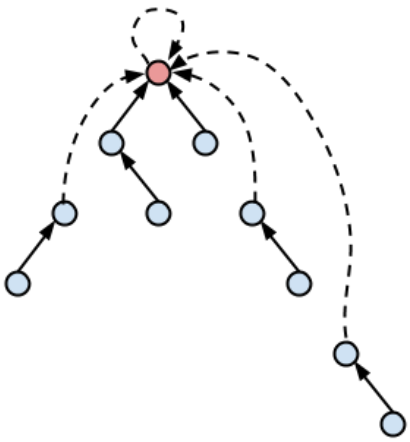



まず，首都から出る辺は首都へ戻る自己ループでなければならないことを示します．条件が成り立つようにグラフを変形できたとき，仮に $a_1 \neq 1$ であったとします．条件が成り立っているので，首都から辺をちょうど $K$ 回辿ると，最終的に首都にいます．そのためには，頂点 $a_1$ から辺をちょうど $K-1$ 回辿ると，最終的に首都にいなければなりません．よって，頂点 $a_1$ から辺をちょうど $K$ 回辿ると，最終的に頂点 $a_1$ $(\neq 1)$ にいます．これは条件に矛盾します．以上より，首都から出る辺は首都へ戻る自己ループでなければなりません．首都から出る辺を首都へ戻る自己ループに変えると，グラフは首都を根とする根付き木のようになります（次図）.



首都に自己ループがあるので，条件は「どの頂点から出発しても，辺を $K$ 回以下辿れば，首都

へ辿り着ける」と言い換えられます．また，辺の行き先を変えるときは，できるだけ早く首都へ辿り着きたいので，辺の行き先を首都へ変えるのが最善であることが分かります．以上の考察に従うと，例えば $K = 2$ の場合，次図のように辺の行き先を変えればよいです．



ここまで来ると，問題は次のように言い換えられます．

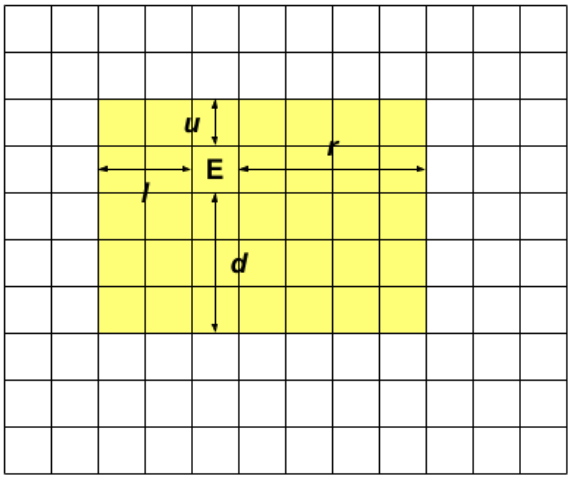
> 首都を根とした根付き木から，できるだけ少ない本数の辺を取り除き，分断されたそれぞれの木が次の条件のどちらかを満たすようにせよ．
>
> - 根が首都であり，高さが $K$ 以下である．
> - 根が首都ではなく，高さが $K - 1$ 以下である．

この問題は次のようにして解くことができます．各頂点の高さを計算するのと同じ要領で，根付き木を深さ優先探索します．ただし，頂点 $v$ を見ているとき，高さが $K - 1$ に達しており，かつ，$v$ の親が首都でなければ，頂点 $v$ を親から分断します．このとき，親へ伝える高さを 0 へリセットします．深さ優先探索が終わった後，分断した回数が答えです．時間計算量は $\mathrm{O}(N)$ です．

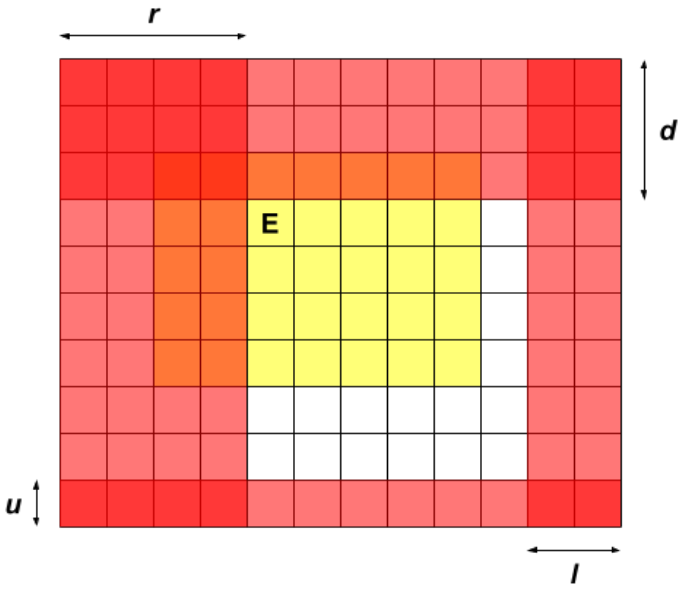
## E : Salvage Robots

ロボットを動かす代わりに，壁と出口を動かすと考えます．例えば，すべてのロボットを左向きに 1 マスだけ動かすというのは，壁と出口を右向きに 1 マスだけ動かすのと同じです．すると，問題設定は「壁と出口の動かし方を工夫し，壁が触れる前に出口が触れるロボットの個数を最大化せよ」となります．

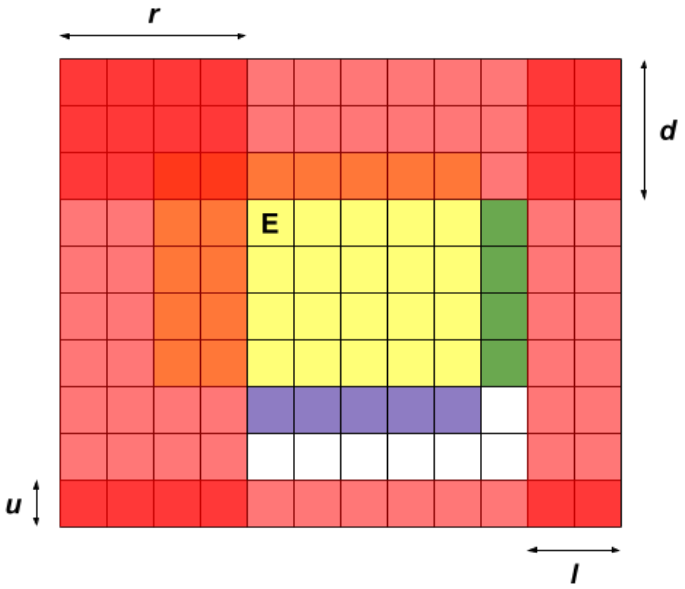
今，出口が初期位置から左向きへ最大 $l$ マス，上向きへ最大 $u$ マス，右向きへ最大 $r$ マス，下向きへ最大 $d$ マス離れたことがあるとします（次図）．このとき，出口はできるだけ多くのロボットを救出するように工夫して動いてきたと仮定します．




出口が動くのと同時に壁も動くので，次図の赤い長方形内にロボットは生き残っていません．（ただし，壁に触れて爆発する前に出口が触れて救出された可能性はあります．）また，黄色の長方形内にもロボットは生き残っていないことが分かります．というのも，出口が黄色の長方形内で動く分には $l$，$u$，$r$，$d$ は新たに大きくならないので，赤い長方形の範囲も大きくなりません．よって，黄色の長方形内にロボットが生き残っていれば，必ず救出しに行けます．出口はできるだけ多くのロボットを救出するように工夫して動いてきたと仮定したので，黄色の長方形内にはロボットは生き残っていないはずです．以上より，色のないマスのロボットはすべて生き残っており，色のあるマスのロボットはすべて消えていると確定できます．



さらに出口が動いていくと，いずれ $l$，$u$，$r$，$d$ のどれかが $+1$ されます．例えば，$r$ が $+1$ されると，次図の緑の長方形内のロボットすべてが新たに救出できます．また，$d$ が $+1$ されると，次図の紫の長方形内のロボットすべてが新たに救出できます．$l$ または $u$ が $+1$ されると，新たに救出できるロボットはありません．

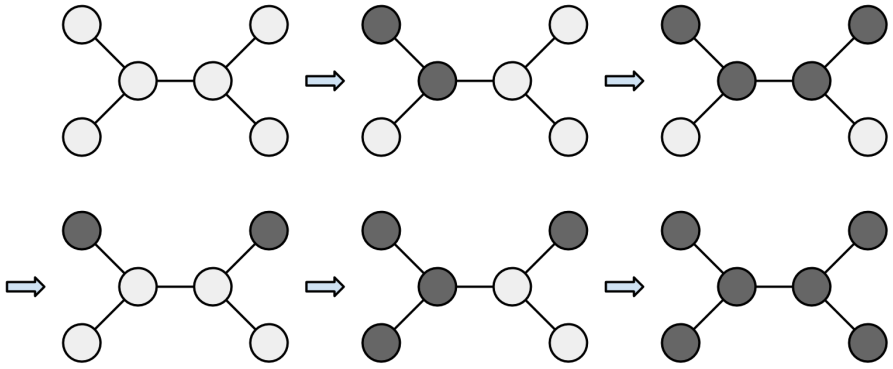



$dp[l][u][r][d]$ を，「出口が初期位置から左向きへ最大 $l$ マス，上向きへ最大 $u$ マス，右向きへ最大 $r$ マス，下向きへ最大 $d$ マス離れたことがあるとき，今までに救出したロボットの個数の最大値」と定義します．すると，以上の考察により，$dp[l][u][r][d]$ の値を用いて，$dp[l+1][u][r][d]$，$dp[l][u+1][r][d]$，$dp[l][u][r+1][d]$，$dp[l][u][r][d+1]$ の各値を更新していくことができます．この DP テーブルを埋めた後，最も大きい黄色の長方形に対応する $dp$ の値が答えです．時間計算量は $\mathrm{O}(H^2W^2)$ です．

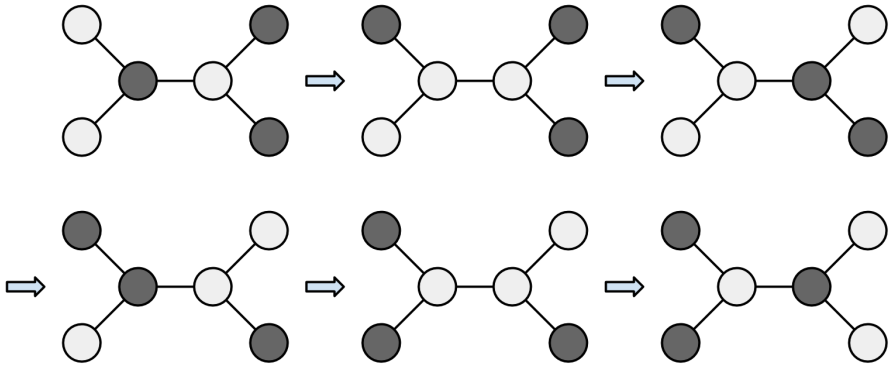
## F : Namori

まずは，$M = N - 1$ の場合，すなわちグラフが木の場合を考えます．今，可能な操作は 白-白 $\leftrightarrow$ 黒-黒 ですが，このままでは考察が難しいです．そのため，次のように見方を変えることにします．

まず，木の頂点たちを白黒交互に塗り分けたものを用意し，これを市松模様と呼ぶことにします．以降，木の各頂点の色を見るときには，この市松模様との XOR を取ったものを代わりに見ることにします．すると，白-白 $\leftrightarrow$ 黒-黒 の操作だったものは，白-黒 $\leftrightarrow$ 黒-白 の操作に見えることになります．サンプル 1 を例として，見え方がどう変わるかを次図に示します．



市松模様との XOR を取る前



市松模様との XOR を取った後

白い頂点を空きマス，黒い頂点を駒のあるマスとして捉え直すと，問題は次のように言い換えられます．

> 木の頂点のうちいくつかに駒が乗っている．駒は隣り合う頂点へ動かすことができるが，駒同士を重ねることはできない．駒の初期配置と最終配置が与えられるので，初期配置から最終配置へ変えられるか判定せよ．変えられるならば，必要な操作回数の最小値を求めよ．

かなり見通しが良くなりました．

言い換え後の問題は，次の最小費用流問題として定式化できます（証明は後述）．まず，初期配置において駒がある各頂点に，流出量 1 の source をそれぞれ設置します．また，初期配置において駒がある各頂点に，流入量 1 の sink をそれぞれ設置します．さらに，各辺を容量 $\infty$，コスト 1 に設定します．とりあえず，source の個数と sink の個数が異なる場合，答えは当然 Impossible です．source の個数と sink の個数が等しい場合，実際にフローを流して得られる最小コストが答えになります．

しかし，この問題では $N \leq 10^5$ とグラフが大きいため，実際にフローを流すことはできません．そこで，最小コストを次のようにして計算します．ある辺 $e = (u, v)$ で木を二分割したとします．

頂点 $u$ 側の部分木を見たとき，この部分木内の (source の個数) − (sink の個数) を $f_e$ とおきます．すると，頂点 $u$ から $v$ への流量は $f_e$ に一致し，辺 $e$ でのコストは $|f_e|$ となります．よって，最小コストは $\sum_e |f_e|$ として計算できます．この時間計算量は $\mathrm{O}(N)$ です．以上で，グラフが木の場合を解くことができました．

次に，$M = N$ の場合を考えます．この場合，グラフにはちょうど 1 つだけ閉路が含まれます．この閉路が偶閉路か奇閉路かで場合分けをします．

まずは，偶閉路の場合を考えます．偶閉路の場合は，木の場合と同様にして，グラフを白黒交互に塗り分けられるので，市松模様との XOR を取ることができます．よって，駒を動かす問題設定へ言い換えられ，さらに最小費用流問題として定式化することができます．しかし，この最小コストは木の場合と同じ方法では計算できません．

最小コストは次のようにして計算できます．まず，偶閉路に含まれる辺を適当にひとつ選びます．この辺を $e_0 = (u, v)$ とします．頂点 $u$ から $v$ への流量 $x$ を決めた後，辺 $e_0$ をグラフから取り去ることを考えます．すると，残ったグラフは木になるので，木の場合と同様にして最小コストを求めることができます．こうして求まるグラフ全体のコストの総和を $F(x)$ とおくと，実は $F(x)$ は $x$ に関して下に凸であることが示せます．よって，$x$ について三分探索を行うことで，$\min F(x)$ を $\mathrm{O}(N \log N)$ 時間で求めることができます．

$F(x)$ は $x$ に関して下に凸であることを示します．$x$ を決めたとき，辺 $e$ でのコストを $f_e(x)$ と書くことにします．まず，辺 $e$ が閉路に含まれない場合，$f_e(x)$ は定数関数なので無視します．辺 $e$ が閉路に含まれる場合，$f_e(x) = |x - a_e|$ ($a_e$ は整数定数) という形になっています．これは $x$ に関して下に凸の関数です．よって，$f_e(x)$ の総和である $F(x)$ も $x$ に関して下に凸です．

次に，奇閉路の場合を考えます．奇閉路の場合は，そもそもグラフを白黒交互に塗り分けられません．しかし，できるだけ白黒交互になるように塗り分けたとします．このとき，同じ色が隣り合うような頂点のペアがちょうど 1 つ存在します．これを $e_0 = (u, v)$ とおきます．すると，辺 $e_0$ 以外の辺での操作は 白-黒 $\leftrightarrow$ 黒-白 ですが，辺 $e_0$ での操作だけは 白-白 $\leftrightarrow$ 黒-黒 となります．言い換えると，頂点 $u$，$v$ に同時に駒を置くか，同時に駒を取り去ることができます．

辺 $e_0$ での例外的な操作により，駒を増やしたり減らしたりすることができるようになりました．そのため，初期配置と終了配置で駒の個数が異なっても，初期配置から最終配置へ変えられる可能性があります．ただし，駒の増減は 2 個刻みなので，初期配置と終了配置で駒の個数の偶奇が異なる場合，答えは Impossible です．初期配置と終了配置で駒の個数の偶奇が一致する場合，適切に駒を増減することで駒の個数を揃えられます．また，駒を増やしたり減らしたりする操作の回数は一意に決まります．駒を増やす操作を $k$ 回行う場合，頂点 $u$，$v$ に source を $k$ 個ずつ設置すればよいです．駒を減らす操作を $k$ 回行う場合，頂点 $u$，$v$ に sink を $k$ 個ずつ設置すればよいです．その後，辺 $e_0$ をグラフから取り去ると，残ったグラフは木になるので，木の場合と同様にして最

小コストを求めることができます．この最小コストに $|k|$ を足せば，答えが求まります．時間計算量は $\mathrm{O}(N)$ です．

TODO：最小費用流問題として定式化できることの証明

# AtCoder Grand Contest 004 Editorial

writer : sugim48

## A : Divide a Cuboid

In this editorial "box" means "rectangular parallelepiped".

In this task, you are asked to divide an $A \times B \times C$ box into two boxes and minimize the difference between the number of blocks in the two boxes.

If at least one of $A$,$B$,$C$ is even, for example, when $A = 2a$, you can divide the box into two boxes of size $a \times B \times C$, thus the answer is 0.

Suppose that all of $A$,$B$,$C$ are odd. If $A = 2a+1$ and you want to divide the box by a plane orthogonal to this direction, it is optimal to divide the box into two boxes of sizes $a \times B \times C$ and $(a+1) \times B \times C$, and the difference is $B \times C$. Similarly, you can get the differences of $C \times A$ and $A \times B$ using other directions. Therefore, when all of all of $A$,$B$,$C$ are odd, the answer is $\min\{B \times C, C \times A, A \times B\}$.

## B : Colorful Slimes

Suppose that Snuke casts the spell $k$ times in total. In order to get a slime of color $i$, he needs to capture one of slimes of colors $i$,$i-1$,...,$i-k$ (indices are modulo $N$). For example, if $K = 2$ and you need a slime of color 3, there are 3 ways to get it:

- Cast a spell → Cast a spell → Capture a slime of color 3
- Cast a spell → Capture a slime of color 2 → Cast a spell
- Capture a slime of color 1 → Cast a spell → Cast a spell

Thus, in order to get a slime of color $i$, he needs $\min\{a_i, a_{i-1}, \ldots, a_{i-k}\}$ seconds (except for the time spent for casting spells). Let $b_i(k)$ be this value. Then, he needs $k \times x + \sum_i b_i(k)$ seconds to get all types of slimes.

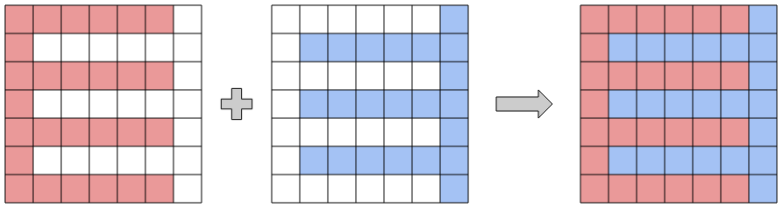
If you try all possible values of $k$ from 0 to $N-1$ and compute the minimum of $k \times x + \sum_i b_i(k)$, you can get the answer. The straightforward implementation of this algorithm is $O(N^3)$. If you use the fact that $b_i(k) = \min\{b_i(k-1), a_{i-k}\}$, you can compute $b_i(k)$ in $O(1)$ for each pair $(i, k)$, and thus the solution works in $O(N^2)$.

## C : AND Grid

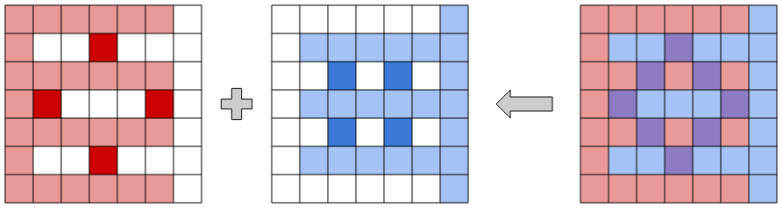
First, we construct a pair of a red grid and a blue grid with the following properties:

- If you overlaid the red grid on the blue grid, each cell becomes either red or blue. In other words, blue cells are the complement of the red cells.
- Red cells are 4-connected. Blue cells are also 4-connected.
- In the red grid, if you choose an arbitrary subset of white cells and paint them red, the grid remains 4-connected. The same holds for blue grid.

For example, the following pair satisfies the conditions.

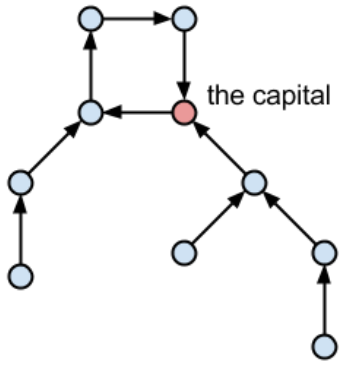


By using this pair, you can construct a solution for arbitrary set of purple cells as in the following diagram:



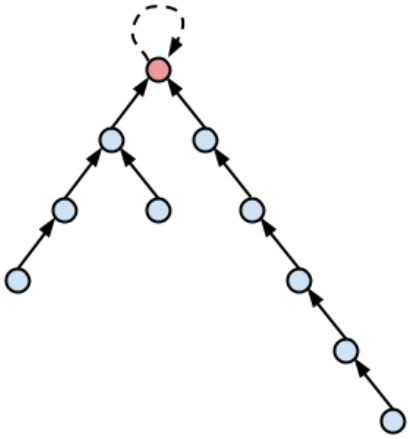
## D : Teleporter

Consider a graph with $N$ vertices. For each $1 \leq i \leq N$, there is a directed edge from the vertex $i$ to the vertex $a_i$. By the condition "one can get to the capital from any town by using the Teleporters some number of times.", (if we ignore the orientations) the graph is connected. In this task, you are asked to change the heads of the smallest possible number of edges and satisfy the condition "Starting from any town, one will be at the capital after using the Teleporters exactly $K$ times in total."



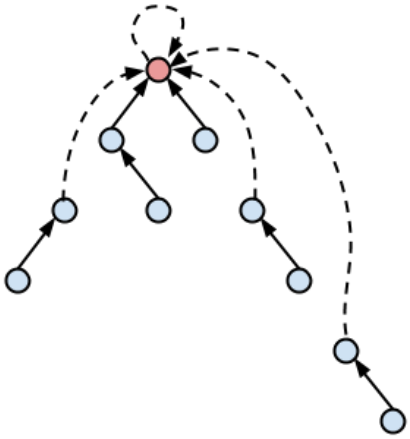

First, we prove that after we change the graph, the edge from the vertex 1 should go to the vertex 1 itself. When you follow the edges $k$ times from the vertex 1, you should be at the vertex 1 (from the condition in the statement). Thus, there must be a cycle containing the vertex 1, and the length of this cycle must be a divisor of $K$. Suppose that there is an edge $1 \to r$. Since $r$ is in the cycle mentioned above, if you follow the edges $k$ times from the vertex $r$, you will reach the vertex $r$. From the statement, you must be at 1m thus we proved that $1 = r$.

If we change the tail of the edge from vertex 1 to 1, the graph will be a tree rooted at the vertex 1:



In this rooted tree, the condition can be restated as "Starting from any town, one will be at the capital after using the Teleporters *at most* $K$ times in total." Also, when we change the tails, we can see that it is optimal to change it to the capital.

When $K = 2$, the following diagram shows an example of optimal solution:



Now the problem can be restated as follows.

> You are given a rooted tree. You want to remove the minimum number of edges such that in each connected component, one of the following is satisfied:
>
> - The root is the capital and the depth is at most $K$.
> - The root is not the capital and the depth is at most $K - 1$.

This problem can be solved as follows. Run a dfs from the root. When we call $dfs(x)$, we basically compute the depth of the subtree rooted at the vertex $x$. However, when we call $dfs(x)$, if there is a child of $x$ (call it $y$) such that $dfs(y) = K - 1$ and $x$ is not the root, we

cut the edge between $x$ and $y$. The answer is the number of cuts. The time complexity of this solution is $O(N)$.

Another solution (easier but a bit more complicated to implement) is as follows. First, choose all the vertices in decreasing order of depth, and mark them unvisited, and repeat the following.

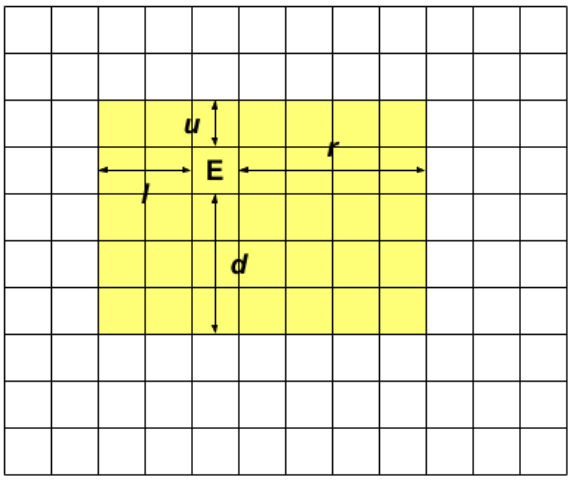
Choose a deepest (unvisited) vertex of the tree. Let's call it $x$. If $depth(x) \leq K$, we are done. Otherwise, we must cut at least one edge from the vertices $x, parent(x), ..., parent^{K-1}(x)$ (to their parent), and we can easily prove that it is optimal to choose an edge from the vertex $parent^{K-1}(x)$. After you cut an edge from the vertex $y$, you should mark all descendants of $y$ as visited.

If you simulate this, you'll get an $O(N)$ solution.

## E : Salvage Robots

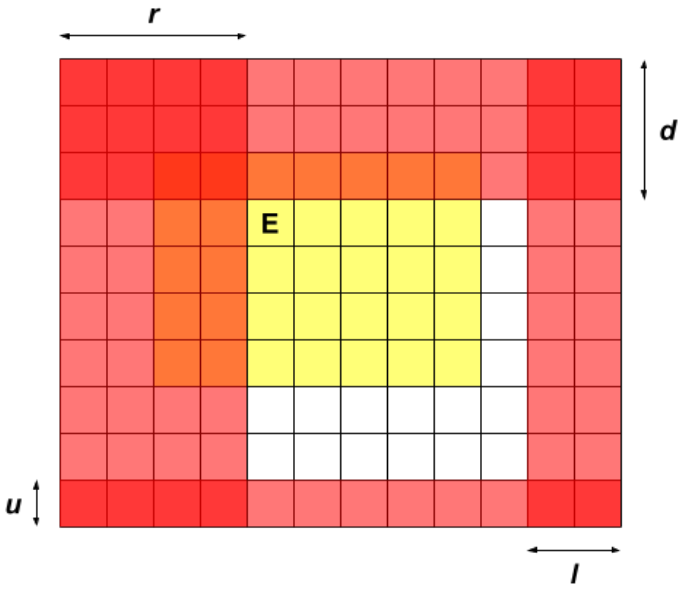
Instead of moving robots, we assume that we move the boundaty of the grid and the exit. For example, moving all robots to the left by a unit cell is equivalent to moving the boundary and the exit to the right by a unit cell. In this task, you are asked to maximize the number of robots that touch the exit before go out of the boundary (by finding the optimal movement of the boundary and the exit).

Suppose that the leftmost cell reached by the exit so far is $l$ units to the left of the initial position of the exit. Similarly, define $u$, $r$, and $d$ for other directions. In the diagram below, the yellow rectangle shows the bounding box of the cells reached by the exit.




When we know the values of $l$, $u$, $r$, and $d$, we can determine the set of cells that have

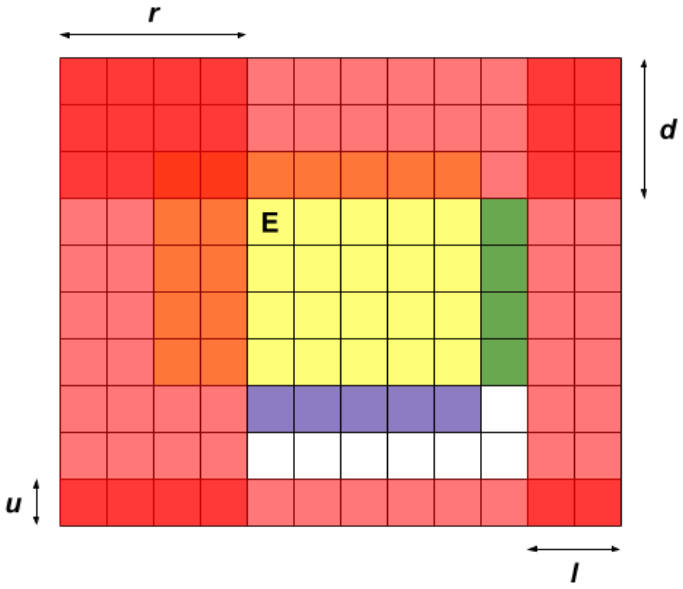
been out of the boundary. For example, the rightmost $l$ columns have no robots because it already went out of the boundary. In the diagram below, no robots are remaining in red cells. (However, it is possible that some of them are rescued before explosion).




Notice that the set of red cells only depends on the *bounding box* of the yellow cells. Thus, you don't need to know the exact set of cells reached by the exit, instead the bounding box is enough. If there is at least one robot in the yellow region in the diagram above, we can rescue it without extending the red cells. Thus, it is enough to consider the case where there is no robot in colored cells. Also, since neither of the grid and the boundary have reached the white cells, we know that all robots that were initially at the white cells still remain.

Therefore, we can define the current state by the four integers $l$, $u$, $r$, and $d$. Define $dp[l][u][r][d]$ as the maximum number of robots you could have saved so far, when the position of the yellow bounding box is defined by $l, u, r, d$, and the exit has reached all cells in the bonding box.

Let's see how to update the $dp$ array. When the exit goes out of the bounding box, one of the values $l$,$u$,$r$,$d$ will be incremented. For example, if we increment $r$, we can save all robots in the green rectangle. Similarly, if we increment $d$, we can save all robots in the purple rectangle. If we increment $l$ or $u$, no new robots can be saved.




In summary, you can update the values of $dp[l+1][u][r][d]$,$dp[l][u+1][r][d]$,$dp[l][u][r+1][d]$,$dp[l][u][r][d+1]$ using the value of $dp[l][u][r][d]$. The answer is the value of $dp$ corresponding to the entire rectangle. The time complexity of this algorithm is $O(H^2W^2)$.

## F : Namori

This task may look overwhelming - it is 2200 points, and probably you won't get any progress for a while. However, if you notice a single clever observation, the task suddenly becomes much more tractable.

We'll write a hint first. If you haven't solved this task by yourself and get stuck, we'll recommend you to try again after reading the hint. Can you solve it?

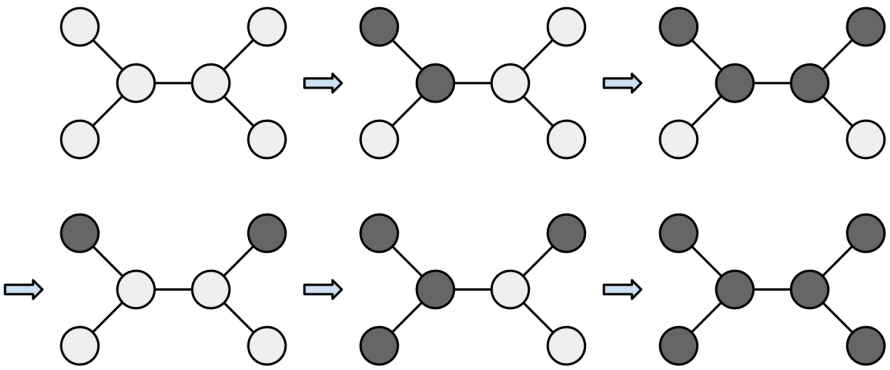
Hint: Trees are always bipartite. Let's color the given tree red and blue. Now, flip all colors in red vertices - black in red vertices becomes white, and vice versa. What does the operation mean after this transformation?

*This page is intentionally left (almost) blank.*

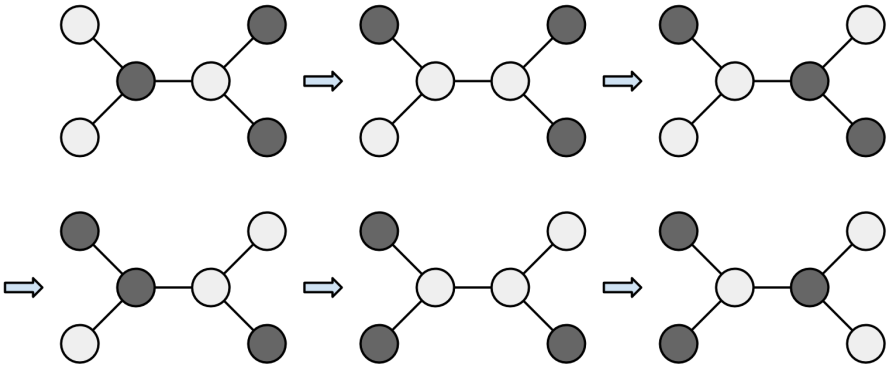
First, consider the case $M = N - 1$, that is, the graph is a tree. Currently you can convert white-white $\leftrightarrow$ black-black, but this looks difficult. Thus, we transform the graph in the following way.

Let's color the vertices of the tree red and blue, and flip all colors in red vertices. Then, the operation white-white $\leftrightarrow$ black-black becomes white-black $\leftrightarrow$ black-white.

As an example, we use Sample Input 1.



Before the transformation



After the transformation

Let's regard white vertices as empty vertices and black vertices as vertices with tokens. Then, the problem can be restated as follows:

> You are given a tree. Some vertices may contain tokens. In each operation, you can "slide" a token in a vertex into adjacent empty vertex. You are given the initial state and the final state. (State can be represented by a set of vertices with tokens). Determine if you can convert from the initial state to the final state by repeating the operations, and if this is possible compute the minimum number of operations.

Now the problem looks much easier!

We first explain an intuitive solution, and then give a formal proof. Let $G$ be the input graph, $S$ be the initial state, and $T$ be the final state. (Also, assume that the graph is transformed as we described above).

## Tree Case

In this case $G$ is a tree.

First, it is obvious that the number of tokens never changes. Thus, when the number of tokens in $S$ and $T$ are different, the answer is -1. Otherwise, it is always possible.

Consider an edge $e$ in $G$. If we cut the graph $G$ by the edge $e$, we get two connected components. Let's call those components $G_1$ and $G_2$. Let $x$ be the number of tokens in $G_1$ in $S$, and let $y$ be the number of tokens in $G_1$ in $T$. Then, $|x - y|$ tokens must pass through the edge $e$.

For each edge, compute the number of tokens that pass through this edge. The answer is the sum of these values for all edges.

## Even Cycle Case

When $M = N$, the graph contains exactly one cycle. We handle two cases separately depending on the parity of the length of the cycle. In this section suppose that the cycle length is even.

In this case, since the graph is bipartite, we can transform the colors in the same way and it is sufficient to solve "slide tokens" version.

Choose an arbitrary edge in the cycle and call it $e_0 = (u, v)$. Let $x$ be the number of tokens that pass through this edge in $u \to v$ direction. (If the tokens pass through the other direction, this number is negative).

Remove this edge from the graph. If we know the value of $x$, by a similar observation as in the tree case, we can compute the number of tokens that pass through each edge, and these values are of the form $x + (constant)$.

Thus, the total number of operations can be represented as the sum of functions of the form $|x + (constant)|$. Since each term of this function is convex, this function is also convex, and we can compute the minimum using ternary search. (It is also possible to find the minimum by computing the median of the constants.)

## Odd Cycle Case

Choose an arbitrary edge in the cycle (call it $e_0 = (u, v)$), and remove it from the graph. After the removal the graph becomes a tree, thus it is almost sufficient to solve "slide tokens"

version on a tree. The only difference is that, there are two distinct vertices in the tree $A, B$ and the following additional operations are possible:

- If both $A$ and $B$ are empty, put tokens to both vertices.
- If both $A$ and $B$ contain tokens, remove tokens from both vertices.

Now we can change the number of tokens using these exceptional operations. However, these operations don't change the parity of the number of tokens. When the parity of the number of tokens in $S$ and $T$ are different, the answer is -1.

Otherwise, without loss of generality (since the operations are revertable), we can assume that the number of tokens from $S$ to $T$ is non-decreasing.

Suppose that the number of tokens increases by $2k$ from $S$ to $T$. In this case, we need perform the operation "Put tokens on $A$ and $B$" $k$ times (and this costs $k$ operations). Then, imagine that we add $k$ more tokens to each of $A$ and $B$, and compute the minimum cost in the same way as the tree case. The answer is this minimum cost plus $k$.

The time complexity of this algorithm is $\mathrm{O}(N)$.

## Formal Proof

Now the formal proof of solutions described above. In the solution above we proved that we need *at least* certain number of operations, but we didn't prove why that number of operations is enough.

Let $G$ be an arbitrary graph, and $S$ and $T$ be two states. Consider a "slide tokens" problem on this graph.

First, for each edge in $G$, we assign an integer with direction (this is the number of tokens that pass through this edge). The integer assigned to $u \to v$ is the negative of the integer assigned to $v \to u$, and for each vertex $v$ in $G$, the following condition must be satisfied:

(The number of tokens in $v$ in $S$) + $\sum value(w \to v)$ = (The number of tokens in $v$ in $T$)

We want to prove that if we can assign integers to each edge with the condition above, we can convert from $S$ to $T$ with at most (the sum of absolute values of assigned integers) numbers of operations.

Let $K$ be the number of tokens (in $S$ and $T$). First, when an integer $x > 0$ is assigned to the edge $u \to v$, we add $x$ arrows from $u$ to $v$. By merging these arrows, you get $K$ paths. Each path starts from a token in $S$ and ends at a token in $T$. Thus, each token in $S$ is now assigned to a token in $T$ (and this relation is bijective), and the total length of paths connecting each pair is the required number of operations. In general, there are multiple optimal matchings between tokens in $S$ and $T$. In this case, suppose that we want to maximize the number of

matchings to itself (that is, a token at vertex $v$ in $S$ is assigned to a token at vertex $v$ in $T$).

Consider a pair of matching. A token at vertex $p$ in $S$ is assigned to a token at vertex $q$ in $T$. When $p \neq q$, we can prove that the vertex $p$ in $T$ is empty and the vertex $q$ in $S$ is empty. In this case, we can move a token from $p$ to $q$ in $dist(p, q)$ steps. It is possible that there is another token in the path between $p$ and $q$. However, when a vertex $r$ contains a token, we can first move a token at $r$ to $q$ and then a token at $p$ to $r$. We can do this in a similar when even when there are multiple tokens on the path between $p$ and $q$.

Now, the proof for the odd cycle case. Instead of adding a token to $A$ and $B$, add a new vertex $v$ with a token and add an edge from $v$ to $A$ with length 0. This way, the problem is similar to the "slide tokens" problem except for the restriction that when a token is moved into $A$, you must move a token into "B" at the same time. This is trivial when $A$ and $B$ are empty before the operation. If they are not empty, for example, if you want to move a token at $v$ to $w$ through $A$ but $A$ contains a token, you can first move a token at $A$ to $w$ with cost $dist(A, w)$ and empty the vertex $A$. Do the same thing in case the vertex $B$ is non-empty. Now, we can finally move tokens into vertices $A$ and $B$, and we get a valid sequence of operations.