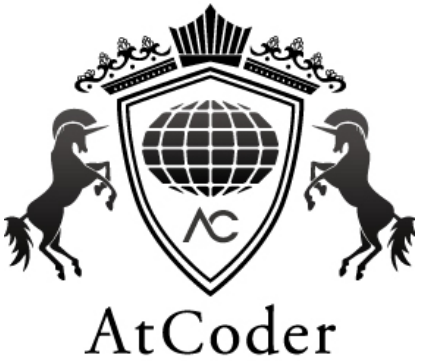
CODE FESTIVAL 2016 Final 解説



AtCoder株式会社

(English editorial is at the bottom.)

A問題 解説
「Where’s Snuke?」

CODE FESTIVAL 2016 本戦

概要と解説

- H行W列に文字列が並んでいるので、”snuke”を見つけてその場所を出力して下さい

- 問題文の指示通りに実装しましょう
- 文字列の扱いに注意しましょう

B問題 解説
「Exactly N points」

CODE FESTIVAL 2016 本戦

問題概要

- 配点が 1~N の問題が 1 問ずつある
- 解く問題の集合を選んで得点を N 点にしたい
- 解く問題のうちの配点の最大値を最小化したい
- そのような問題の集合を 1 つ出力せよ

- 制約
  - $1 \leqq N \leqq 10^7$

解法

- 配点の最大値の最小値 X を求めたい
  - X を 1 から順に試していき、1~X からいくつか選んで合計を N に出来るかを判定する
  - 初めて「可能」になった X が求めたい値

解法

- 1~X からいくつか選んで合計を N に出来るかの判定
  - 実は、1~X の和が N 以上なら必ず出来る
    - 1~Y の和がギリギリ N 以上になるような Y をとる
      - 1~(Y-1)の和 < N ≦ 1~Yの和
    - このとき、1~Y の和と N の差が Y 未満になっている
      - 0 ≦ 1~Yの和 − N < Y
    - つまり、1~Y から高々 1 つの問題を取り除けばOK
    - 例えば、N=12 の場合
      - Y=5 で、(1+2+3+4+5)-12=3 なので、1,2,4,5 と選べば良い
    - 差が 0 の場合は何も取り除かなくて良い

別解・部分点

- 別解
  - 配点の多い順に、合計が N が超えない範囲で貪欲に問題を選ぶ
    - 正当性は数学的帰納法などで証明できます
- 部分点
  - 動的計画法などを用いることで部分点を得ることが出来ます

C問題 解説
「Interpretation」

CODE FESTIVAL 2016 本戦

問題概要

- N 人の人と M 個の言語がある
- それぞれの人が話せる言語のリストが与えられる
- 通訳を挟む（または直接話す）ことで全員がコミュニケーションを取り合うことが出来るか判定せよ

- 制約
  - $2 \leqq N \leqq 10^5$
  - $1 \leqq M \leqq 10^5$
  - 話せる言語のリストのサイズの和を ΣK としたとき、$\Sigma K \leqq 10^5$

部分点解法

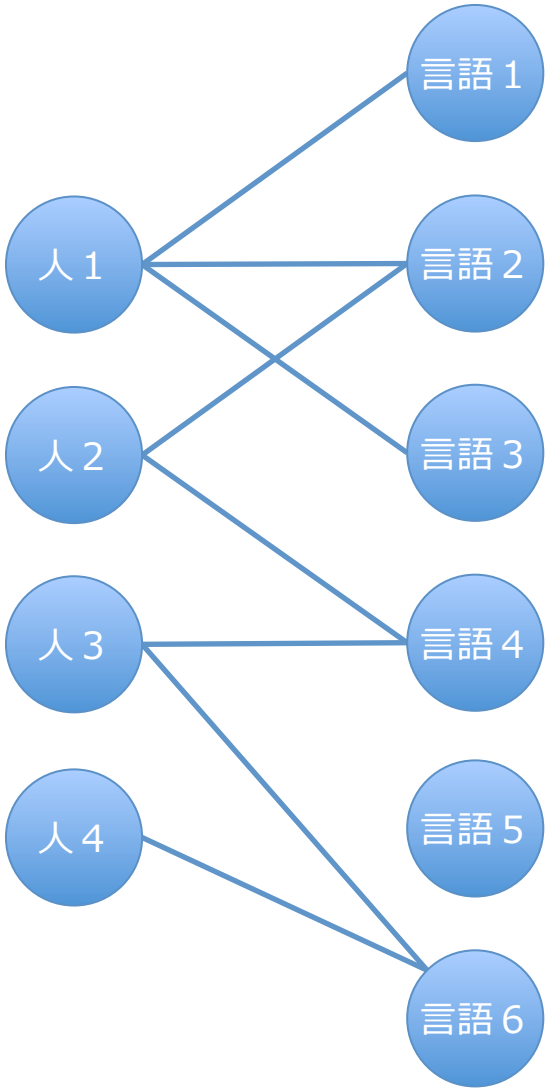
- 部分点制約
  - N, M, ΣK ≦ 1000

- 人を頂点としたグラフを作る
- 直接話すことが出来る人の頂点の間に辺を張る
  - ２人の人の話せる言語のリストに同じ言語があれば張る
- このグラフが連結であるかどうかをチェックする
  - Union Findを用いると容易に判定できる
  - DFSやBFSなどでも良い

- 辺の本数が $O(N^2)$ 本なので、満点には届かない

満点解法

- 辺の本数を減らしたい
- 人と言語の二部グラフを考える
- このグラフで、人どうしが連結になっていれば良い
  - 言語５のように、連結でない言語が存在しても良い点に注意
- 連結性判定は部分点とほぼ同じ
- 頂点が N+M 個、辺が ΣK 本のグラフになるので間に合う



D問題 解説
「Pair Cards」

CODE FESTIVAL 2016 本戦

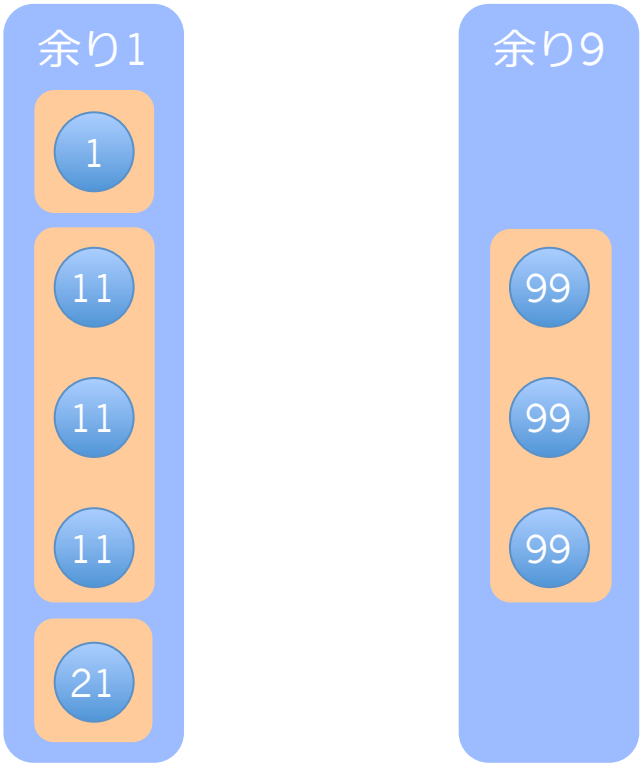
問題概要

- N 枚のカードがある
- カード i には整数 X_i が書かれている
- 「書かれた数が同じ」または「数の和が M の倍数」であるような 2 枚のカードを組にできる
- 最大で何組作ることが出来るか？

- 制約
  - $2 \leqq N \leqq 10^5$
  - $1 \leqq M \leqq 10^5$
  - $1 \leqq X_i \leqq 10^5$

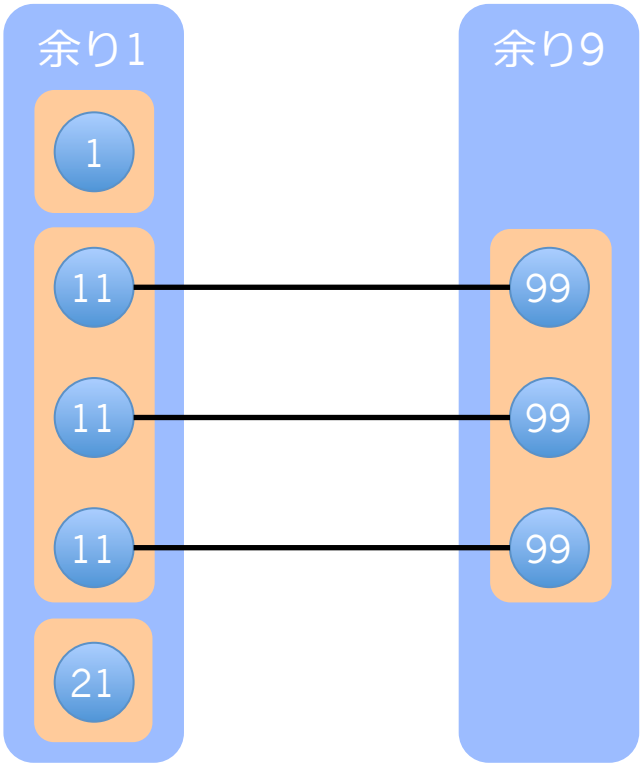
解法

- 書かれた数を M で割った余りでグループ分けする
- 例：M=10 の場合



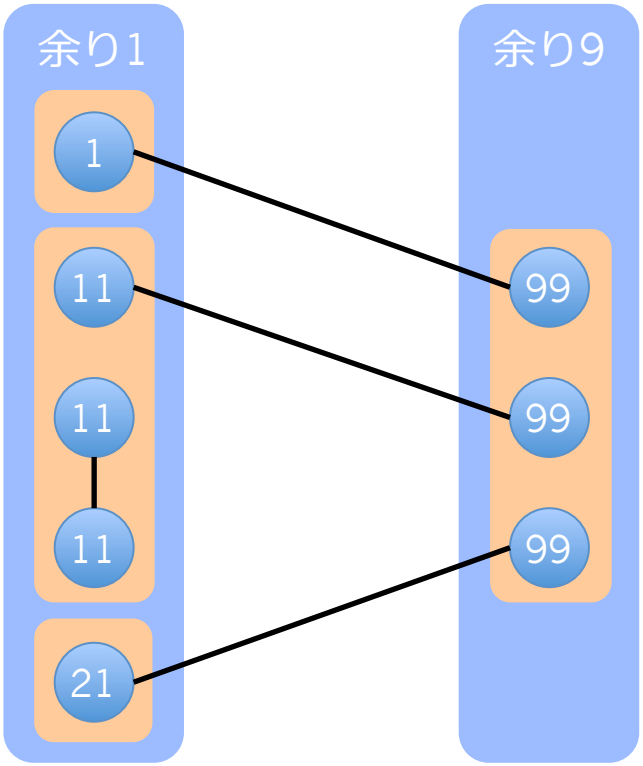
解法

- 最大マッチングを考える
- これは最大でない



解法

- これが最大



解法

- 以下のような貪欲法で最大マッチングが求まる
  - 下準備
    - 余り x のグループを S、余り M-x のグループを T とする
    - S の要素数を |S|、T の要素数を |T| と書くことにする
    - |S| < |T| の場合は、S と T をswapする
      - これで、|T| ≦ |S| となる
  - 貪欲法
    - T の要素はすべて S の要素とマッチングさせることにする
    - S の要素を |T| 個以上の要素が余る範囲で最大限マッチングさせる
      - 同じ数どうしの組を作る
    - T の要素をすべて S の余った要素とマッチングさせる
      - 和が M の倍数の組を作る
  - 正当性の証明はこの問題の解説の末尾にあります

解法

- M で割った余りが 0 または M/2 のケース
  - たとえば、M=10 のときの 10,20 や 5,15
  - このケースは特別処理をしなければいけない
  - 余りが 0（または M/2）のグループの要素は、グループ内でどれとどれをマッチングさせても良い
    - 合計が M の倍数になるため
  - ということは (グループの要素数)/2 個の組が作れる

解法

- まとめ
  - M で割った余りでグループ分けをする
  - グループ内の要素を足し合わせると M の倍数になるグループ：
    - (グループの要素数)/2
  - そうでない場合：
    - 余り x のグループと余り M-x のグループをまとめて処理する
    - 前述の貪欲法で最大マッチングを求める

貪欲法の証明

- 貪欲法の証明
  - 「T の要素はすべて S の要素とマッチングさせることにする」
    - この部分の正当性が証明できればあとは簡単
  - 「T の要素をすべて S の要素とマッチングさせるような最大マッチングが存在する」を証明すればよい

貪欲法の証明

- 「T の要素をすべて … が存在する」の証明
  - ある最大マッチングにおいて T の要素が余っている場合：
    - 余っている T の要素を x とする
    - |T| ≦ |S| より、S の要素でかつ T の要素とマッチングされていない要素が存在する（この要素を y とする）
      - y が S の別の要素とマッチングされている場合は解消する
        - （そもそも y がマッチングされていない場合は最大マッチングでないが）
    - x と y をマッチングさせることにより、x を S の要素とマッチングさせるような最大マッチングを構成することができる

貪欲法の証明

- 「T の要素をすべて … が存在する」の証明
  - ある最大マッチングにおいて T の要素どうしのマッチングが存在している場合：
    - このマッチングの要素を x, y とする
    - |T| ≦ |S| より、S の要素でかつ T の要素とマッチングされていない要素が 2 つ以上存在する
      - S の要素どうしのマッチングが存在する場合：マッチングを解消し、それらの要素を u, v とする
      - 存在しない場合：余っている S の要素が 2 つ以上存在し、それらの要素を u, v とする
        - （そもそもこの場合は最大マッチングでないが）
    - x と y のマッチングを解消し、x と u・y と v をマッチングさせることにより、x, y を S の要素とマッチングさせるような最大マッチングを構成することができる

E問題 解説
「Cookies」

CODE FESTIVAL 2016 本戦

問題概要

- ある人がクッキーを焼いている
  - はじめ、クッキーを 1 秒で 1 枚焼くことが出来る
  - クッキーが x 枚あるとき、それらを A 秒かけてすべて食べることにより、クッキーを 1 秒で x 枚焼くことができるようになる
- クッキーを N 枚以上用意するには最短で何秒かかるかを求めよ

- 制約
  - $1 \leqq N \leqq 10^{12}$
  - $0 \leqq A \leqq 10^{12}$

部分点解法

- 部分点制約
  - N,A ≦ 10^6

- DP（動的計画法）
  - dp[i] = i 枚**ちょうど**焼くときの最短時間
  - 初期化：dp[0] = 0, それ以外は ∞
  - 遷移：
    - for (i=1; i<N; i++) for (j = 0; j <= N*2; j += i)
      dp[j] = min(dp[j], dp[i]+A+j/i);
  - 答え：dp[N]~dp[2N] の最小値
    - dp[N] が必ずしも最小値とならない点に注意
    - 最後にクッキーを全部食べる個数 i は N-1 以下のはずで、N~2N の中に i の倍数があるため、2Nで十分
  - 計算量：O(N log N)
    - O(Σ(N/i | 1≦i≦N)) = O(N log N)

満点解法

- クッキーを食べる回数を決め打ちする
  - 0 回から 40 回くらいまで試せば十分
    - O(log N)

- クッキーを食べた回数を k、クッキーを連続で焼いた秒数をそれぞれ $s_1, s_2, \ldots, s_{k+1}$ とする
  - かかる時間：$A*k + (s_1+s_2+\ldots+s_{k+1})$
  - クッキーの枚数：$s_1*s_2*\ldots*s_{k+1}$

満点解法

- s の積を N 以上にするときの、s の総和の最小値を求めたい
- どのような s_i を取るのが最適か？
  - **なるべく均等にする**のが最適！
  - 具体的には s の最大値と最小値の差 1 以下になるようにする
    - もしそうでないなら、最大値から 1 を引いて最小値に 1 を足すことにより、総和を変えずに積を増やすことができ、より有利な配分を作ることが出来る
- どうやって最適な s を求めるか
  - s の最大値 m として考えられる最小値を二分探索する
  - さらに、s のうち m-1 に出来る個数の最大値を全探索で調べる

満点解法

- 計算量：$O(\log^3 N)$
- かけ算のオーバーフローに注意

Problem E
Cookies

CODE FESTIVAL 2016 Final

Problem

- A person is baking cookies
  - Initially, one cookie can be baked per second
  - When there are x cookies, by eating all of them spending A seconds, x cookies can be baked per second afterwards
- How many seconds does it take to prepare at least N cookies?

- Constraints
  - $1 \leqq N \leqq 10^{12}$
  - $0 \leqq A \leqq 10^{12}$

Solution for Partial Score

- Additional constraints for partial score
  - $N,A \leqq 10^6$

- Dynamic Programming (DP)
  - dp[i] = the minimum time it takes to prepare **exactly** i cookies
  - Initialization: dp[0] = 0, ∞ for the other values
  - Transition:
    - for (i=1; i<N; i++) for (j = 0; j <= N*2; j += i)
      dp[j] = min(dp[j], dp[i]+A+j/i);
  - Answer: the minimum value among dp[N] through dp[2N]
    - Note that dp[N] is not necessarily minimum among them
  - Time complexity: O(N log N)
    - $O(\Sigma(N/i \mid 1 \leqq i \leqq N)) = O(N \log N)$

Solution for Full Credit

- Decide the number of times to eat cookies first
  - Iterating from 0 times to around 40 times is enough
    - In the optimal strategy, cookies are eaten O(log N) times

- Let the number of times cookies are eaten be k, and the consecutive number of seconds cookies are baked be s_1, s_2, ..., s_{k+1} , respectively
  - The total time taken: $A*k + (s_1+s_2+...+s_{k+1})$
  - The number of cookies prepared: $s_1*s_2*\cdots*s_{k+1}$

Solution for Full Credit

- We must find the minimum possible sum of s when the product of s is at least N
- What are the optimal values of s_i?
  - They should be as **uniform** as possible!
  - That is, the maximum and minimum values among s should differ by at most 1
    - Otherwise, by subtracting 1 from the maximum and adding 1 to the minimum, the product can be increased while keeping the sum, leading to a more efficient distribution
- How to find optimal s
  - Perform binary search on the maximum value among s (let it be m)
  - Then, use brute-force on the maximum possible number of occurrences of m-1 among s

Solution for Full Credit

- The time complexity: $O(\log^3 N)$
- Watch out for overflow when multiplying numbers

F問題 解説
「Road of the King」

CODE FESTIVAL 2016 本戦

問題概要

- N 個の頂点がある
- 始点が 1 であるような長さ M のパスを考える
- パスに沿って有向辺を張ったグラフが強連結になるようなパスは何通り考えられるか？

- 制約
  - 2 ≦ N ≦ 300
  - 1 ≦ M ≦ 300

考察

- 強連結になる条件を考える
  1. パスはすべての頂点を通らなければならない
  2. パスの終点から頂点 1 へ到達可能でなければならない
- 上記は必要十分条件になっている
  - 必要性は自明
  - 十分性の証明の流れ
    - 1. より、頂点 1 からすべての頂点に到達可能
    - 1. より、すべての頂点からパスの終点に到達可能であり、2. よりすべての頂点から頂点 1 に到達可能であることが言える
    - よって、すべての頂点は頂点 1 を経由することにより他のすべての頂点に到達可能であるため、強連結であることが言える

解法

- DP（動的計画法）で解くことを考える
- DPの状態はどう取ればよいか？
  - dp[パスの長さ][使った頂点の個数][強連結性に関する状態]
    - 遷移としては、パスの末尾を１つずつ伸ばしていくイメージ
  - これくらいあれば十分そう
  - 「強連結性に関する状態」に何を取れば良いのかが問題

解法

- 強連結性に関する状態についての考察
  - パスの末尾を１伸ばすとき
    - 頂点 1 に繋げば強連結になる
    - 頂点 1 に到達可能な頂点に繋いでも強連結になる
      - 頂点 1 に到達可能な頂点は頂点 1 を含む強連結成分内の頂点であることが、先程の強連結になる条件の証明と同様に証明できる
    - それ以外の頂点に繋いだ場合は？
      - 実は強連結性に関して全く進展がない
  - ということは、**頂点 1 を含む強連結成分のサイズ**さえ持っておけば良い！

解法

- 正しいDP
  - dp[i][j][k] = パスの長さが i で、j 個の頂点がパスに含まれていて、頂点 1 を含む強連結成分のサイズが k である場合の数
  - 初期化：dp[0][1][1] = 1
  - 遷移：
    - dp[i+1][j+1][k] += dp[i][j][k]*(N-j)
      - パスに含まれない頂点をパスの末尾につける
    - dp[i+1][j][k] += dp[i][j][k]*(j-k)
      - 頂点 1 を含む強連結成分に含まれない頂点をパスの末尾につける
    - dp[i+1][j][j] += dp[i][j][k]*k
      - 頂点 1 を含む強連結成分の含まれる頂点をパスの末尾につける
  - 答え：dp[M][N][N]
  - 計算量：$O(MN^2)$

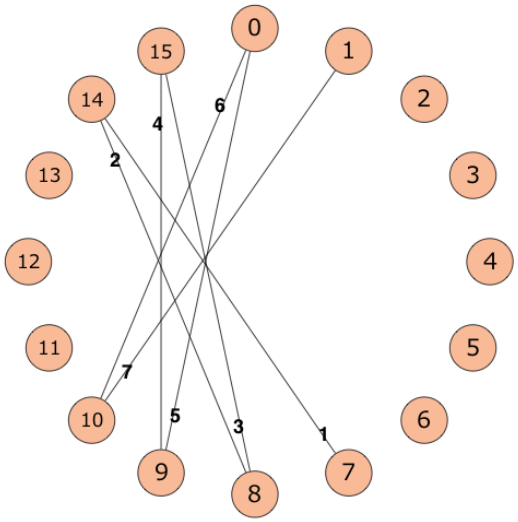
G問題 解説
「Zigzag MST」

CODE FESTIVAL 2016 本戦

問題概要

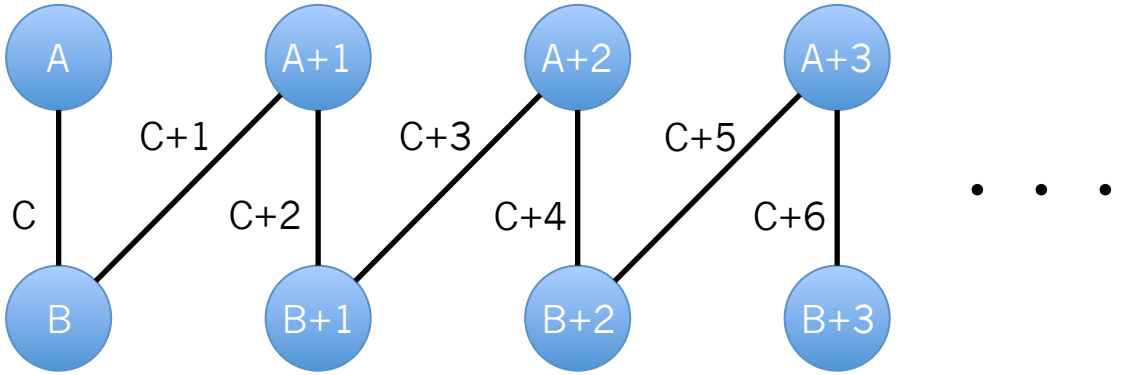
- N 個の頂点がある
- Q 個の辺追加クエリが与えられる
  - クエリ i : A_i, B_i, C_i が与えられるので以下のような辺を張る
    - 頂点 A_i と頂点 B_i を繋ぐ重み C_i の辺
    - 頂点 B_i と頂点 A_i+1 を繋ぐ重み C_i+1 の辺
    - 頂点 A_i+1 と頂点 B_i+1 を繋ぐ重み C_i+2 の辺
    - 頂点 B_i+1 と頂点 A_i+2 を繋ぐ重み C_i+3 の辺
    - …
- このグラフの最小全域木(MST)を求めよ

- 制約
  - $2 \leqq N \leqq 200{,}000$
  - $1 \leqq Q \leqq 200{,}000$
  - $1 \leqq C_i \leqq 10^9$



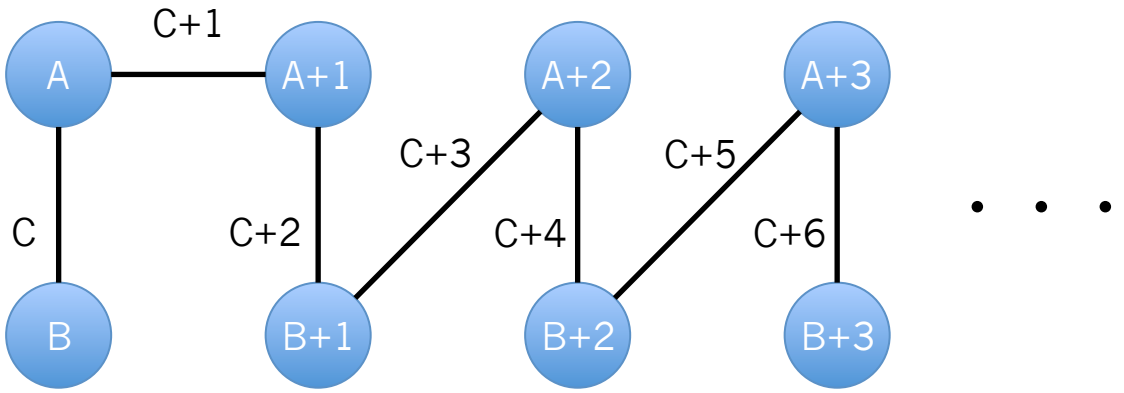
解法

- 辺の本数が多すぎてそのままではMSTが求められない
- 辺の追加クエリについて考察する
  - クラスカル法でMSTを求めることを考える
    - 下図だと、辺は左から順に試されていく



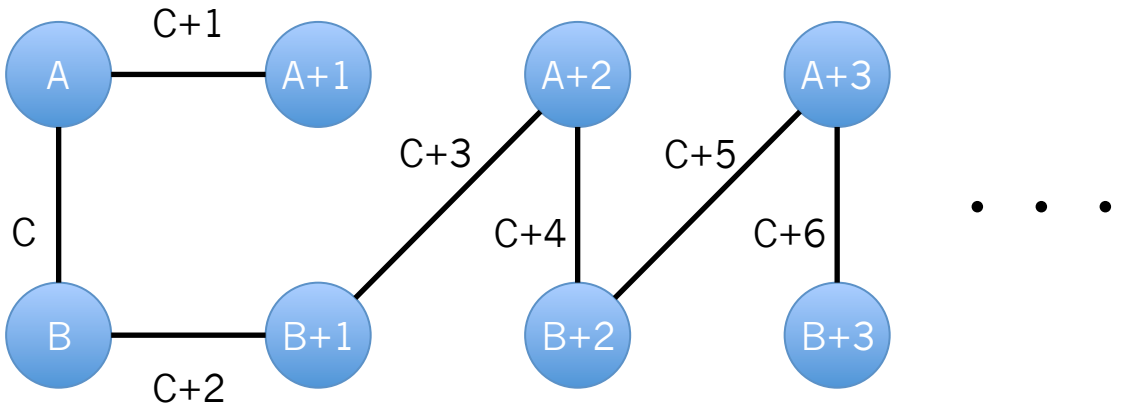
解法

- 辺の追加クエリについて考察する
  - クラスカル法でMSTを求めることを考える
  - 実は、下図のように辺を繋ぎ変えてもMSTの重みは同じ！
    - 重み C+1 の辺を試すときには、すでに重み C の辺は試されており、頂点 A と頂点 B は連結になっているはず



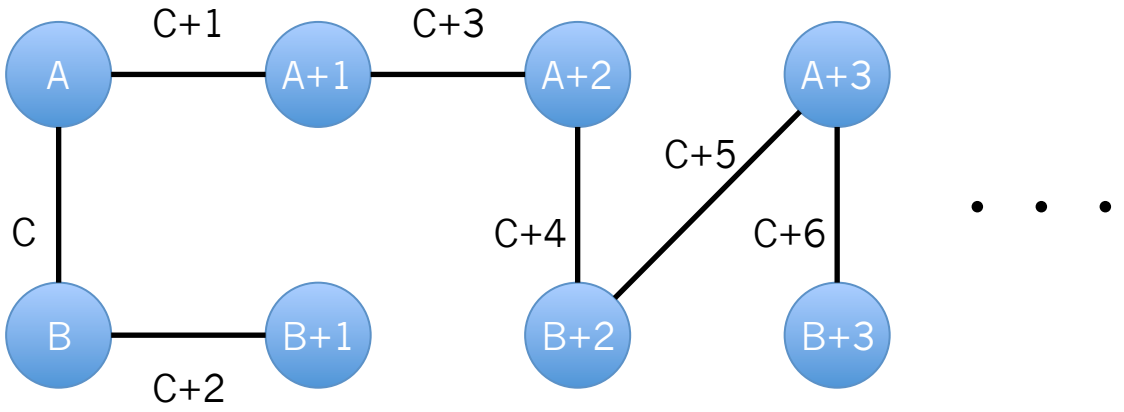
解法

- 辺の追加クエリについて考察する
  - 同様に繋ぎ変えていく



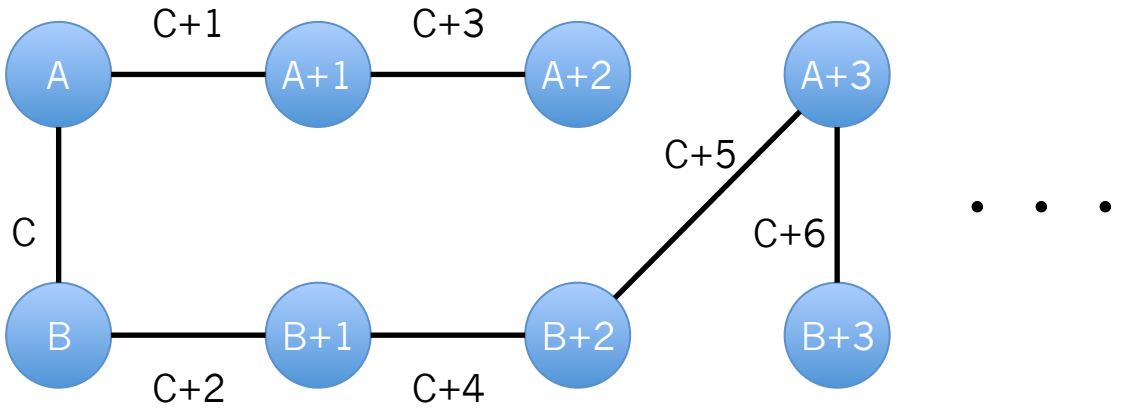
解法

- 辺の追加クエリについて考察する
  - 同様に繋ぎ変えていく



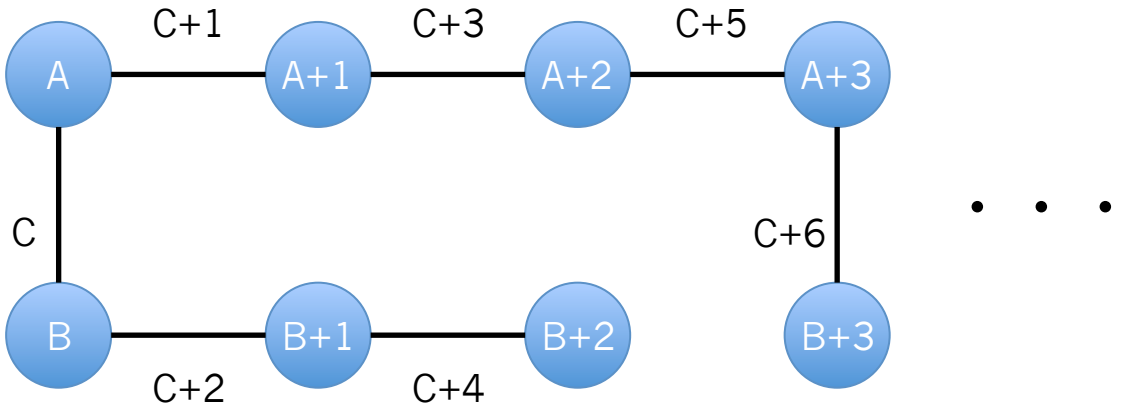
解法

- 辺の追加クエリについて考察する
  - 同様に繋ぎ変えていく



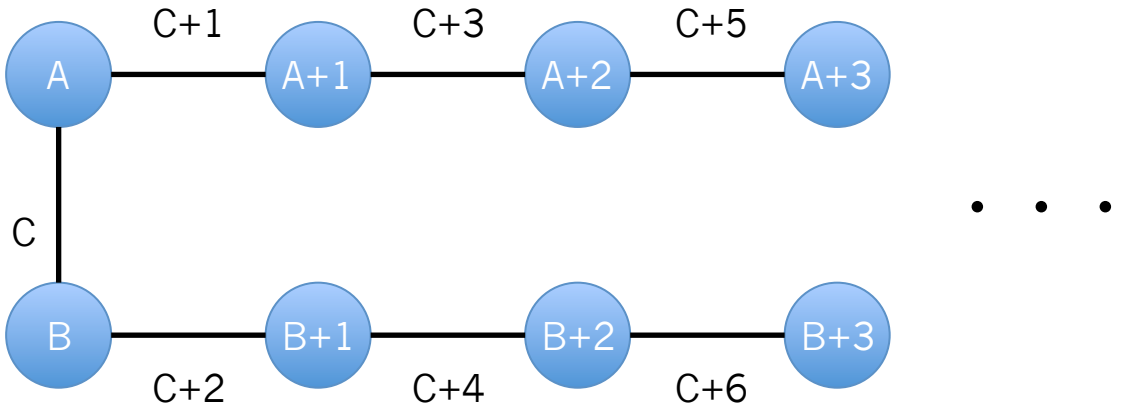
解法

- 辺の追加クエリについて考察する
  - 同様に繋ぎ変えていく



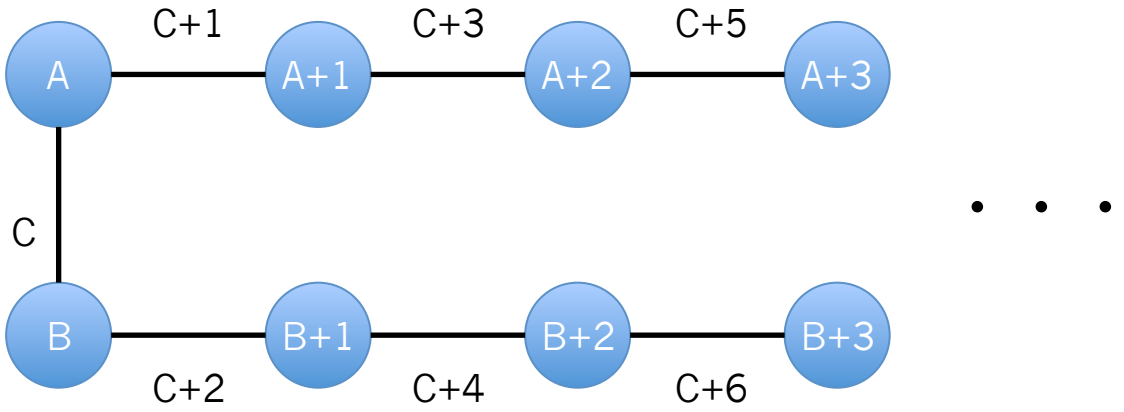
解法

- 辺の追加クエリについて考察する
  - 同様に繋ぎ変えていく



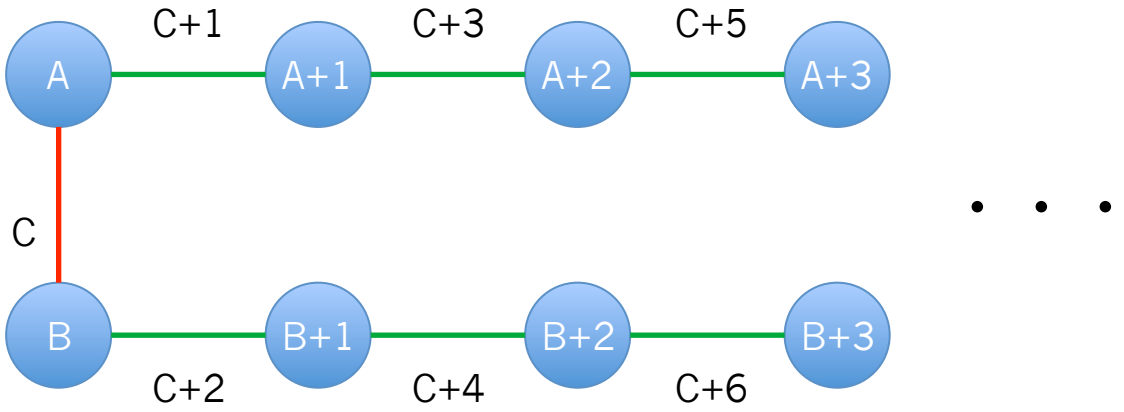
解法

- 辺の追加クエリは以下のように分解できる
  - 頂点 A と頂点 B を繋ぐ重み C の辺
  - 頂点 A+i と頂点 A+1+i を繋ぐ重み C+1+2i の辺
  - 頂点 B+i と頂点 B+1+i を繋ぐ重み C+2+2i の辺



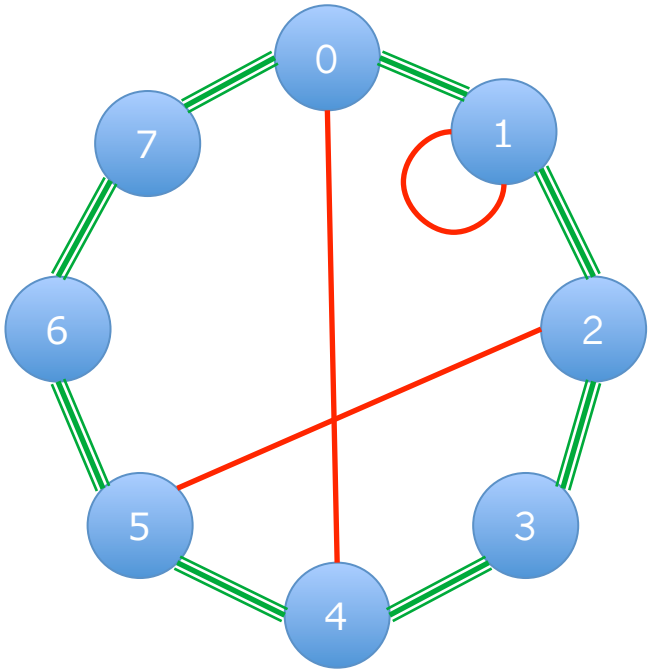
解法

- 辺の追加クエリは以下のように分解できる
  - 頂点 A と頂点 B を繋ぐ重み C の辺
  - 頂点 A+i と頂点 A+1+i を繋ぐ重み C+1+2i の辺
  - 頂点 B+i と頂点 B+1+i を繋ぐ重み C+2+2i の辺
- 説明のため、辺をタイプごとに色分けしておきます



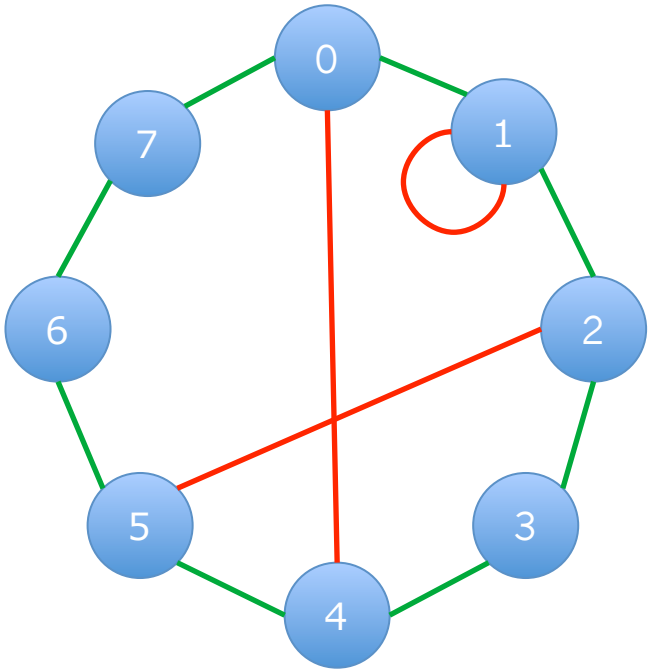
解法

- 辺追加クエリを処理し終わった後、グラフには図のようなイメージで辺が張られる
  - 緑の部分には無限本の緑辺が張られている



解法

- 緑の部分にある無限本の緑辺のうち、重みが最小のもの以外を削除してもMSTは変わらない
- これで、辺の本数が Q+N 本になったため、あとはクラスカル法などでMSTを求めれば良い



解法

- 緑の部分の緑辺のうち最小のものを求める方法
  - 緑辺：頂点 S+i と頂点 S+1+i を繋ぐ重み X+2i の辺
  - 緑辺は以下のような手順で張られていくものだと考えると考えやすくなる
    - はじめに頂点 S と頂点 S+1 を繋ぐ重み X の辺を追加する
    - そこから、各辺について、時計回りに１つ進んだ場所に重みを２増やした辺を張っていく
  - アルゴリズム
    - c[i] = 頂点 i と頂点 i+1 を繋ぐ辺の重みの最小値、を求める

    1. c[i] を ∞ で初期化する
    2. 各(S,X) の組について c[S] = min(c[S], X) と更新する
    3. 0~N-1 の i について、順番に c[i+1] = min(c[i+1], c[i]+2) と更新していくことを２周繰り返す
       - ２周繰り返すのは、N-1 と 0 の間をまたぐものに対応するため

解法

- 計算量
  - 緑の部分の緑辺の最小値を求める：O(Q+N)
  - MSTを求める：O((Q+N) log (Q+N))

H問題 解説
「Tokaido」

CODE FESTIVAL 2016 本戦

問題概要

- ０以上の整数の点数が設定された N 個のマスが左右に一列に並んだマス目を使って２人ゲームをする
  - ２人は駒を用意し、マス１とマス２にそれぞれ置く
  - 各ターンは、相手の駒より左に駒があるプレイヤーの手番となり、今のマスより右でかつ相手の駒がないマスに動かさなければならない
  - 駒が動かせなくなったら終了
  - 得点は「自分の駒を置いたことのあるマスの点数の和」
  - ２人とも相手との得点差を最大にしようとする
- 右端のマスの点数はまだ決まっておらず、いくつか候補が考えられるので、それぞれについてゲームの結果を求めよ

- 制約
  - $3 \leqq N \leqq 200{,}000$
  - $0 \leqq$ 右端以外のマスの点数の総和 $\leqq 10^6$
  - $1 \leqq$ 右端のマスの点数の候補の個数 $\leqq 200{,}000$
  - $0 \leqq$ 右端のマスの点数の候補 $\leqq 10^9$

考察

- 駒を動かす戦略を考える
- 自分の駒のすぐ次のマスに相手の駒がない場合、そこに動かすのが最適
  - マスの点数は負でないため取って損はせず、次も自分が駒を動かせる
- ということは、駒の動かし方を以下のように変えてもゲームの結果は変わらない
  - 「自分の駒を動かした後、相手の駒を勝手に自分の駒のすぐ左まで１マスずつ動かしてくる」
  - 手番が終わった後は、常に駒が隣接したマスに置かれた状態になる

部分点解法

- 部分点制約
  - 右端のマスの点数の候補が 1 つだけ

- DP（動的計画法）
  - 状態
    - dp[i] = 自分の駒が i-1 番目のマス、相手の駒が i 番目のマスにある状態から始めたときの「自分の得点ー相手の得点」の最大値
      - ただし、A[i-1] と A[i] は得点に含まないことにします
      - dp[N] = 0 で、dp[2]+A[1]-A[2] が答え
  - 遷移
    - dp[i] = max(A[j]-sum(A[i+1]~A[j-1])-dp[j] | i<j≦N)
      - ただし、A[i] で i 番目のマスの点数を表すこととする
- これではまだ計算量が悪く、間に合わない

部分点解法

- max(A[j]-sum(A[i+1]~A[j-1])-dp[j] | i<j≦N)
  - これを高速に求めたい
- dp[i+1] の式に注目する
  - max(A[j]-sum(A[i**+1**+1]~A[j-1])-dp[j] | i**+1**<j≦N)
    - dp[i] の式にかなり似ている
- 結果を再利用することを考える
  - max(A[j]-sum(A[i+1]~A[j-1])-dp[j] | i<j≦N)

    = **max(**max(A[j]**-A[i+1]**-sum(A[i+1+1]~A[j-1])-dp[j] | i+1<j≦N),
    A[i+1]-dp[i+1])
  - すなわち、
  - dp[i] = max(dp[i+1]-A[i+1], A[i+1]-dp[i+1])
- これで計算量は O(N) となり、間に合う

満点解法

- dpの漸化式に注目すると、
  - dp[i] = max(dp[i+1]-A[i+1], A[i+1]-dp[i+1])
    = |dp[i+1] - A[i+1]|
  - dp[i+1] から A[i+1] を引いて絶対値を取るだけのシンプルな式になっている
  - 以下のようなゲームを考えると直感的に理解しやすい
    - はじめ、１つの駒が 2 番目のマスに置いてあり、２人の得点の初期値はそれぞれ A[1] と A[2] である
    - 「主導権」をどちらかのプレイヤーが持っている
      - 元のゲームで左に駒があるプレイヤーに相当する
    - 各ターンで、主導権を持ったプレイヤーは「現状維持」か「交代」を選ぶ
      - 現状維持：駒を１マス進め、そのマスの点数を相手に加算する
      - 交代：駒を１マス進め、そのマスの点数を得て、主導権を相手に渡す

満点解法

- もはや答えは配列を使わず計算できる
  - x = A[N];
    for (i=N-1; i >= 3; i--) x = abs(x-A[i]);
    ans = x+A[1]-A[2];
  - 以降ではこれを「アルゴリズムX」と呼ぶことにする
  - A[N] を変化させたときの ans を高速に求められるようにしなければならない
  - +A[1]-A[2] の部分はもはや本質ではないため、以降では無視して考えることにする

満点解法

- A[N] が非常に大きい場合
  - A[N] が 10^9 などの場合、abs(x-A[i]) は常に x-A[i] である
  - このとき答えは A[N]−sum(A[3]~A[N-1]) となる
    - 具体的には、sum(A[3]~A[N-1]) ≦ A[N] の場合にこうなる
  - あとは、A[N] ≦ sum(A[3]~A[N-1]) の場合が解ければ良い
    - 制約より sum(A[3]~A[N-1]) ≦ 10^6 である点に注意せよ

満点解法

- A[N] が小さい場合
  - 以下のようなDPテーブルを考える
    - dp[j][k] = アルゴリズムXのforで i が j まで回ったときの x の値が k であったときの最終的な x の値
    - dp[3][k] = k で、dp[N][A[N]] が答え
    - dp[j+1][k] = dp[j][abs(k-A[j])]
  - 例：N=6, A[3]=5, A[4]=3, A[5]=4

k	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
dp[3][k]	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
dp[4][k]	5	4	3	2	1	0	1	2	3	4	5	6	7	8	9	10
dp[5][k]	2	3	4	5	4	3	2	1	0	1	2	3	4	5	6	7
dp[6][k]	4	5	4	3	2	3	4	5	4	3	2	1	0	1	2	3

満点解法

- dp[N] を高速に計算したい
  - DPテーブルの性質を考える
  - dp[j+1] は、dp[j] を A[j] だけずらし、dp[j][1]~dp[j][A[j]] をリバースしたものを先頭につけたものになっている！

k	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
dp[3][k]	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
dp[4][k]	5	4	3	2	1	0	1	2	3	4	5	6	7	8	9	10
dp[5][k]	2	3	4	5	4	3	2	1	0	1	2	3	4	5	6	7
dp[6][k]	4	5	4	3	2	3	4	5	4	3	2	1	0	1	2	3

満点解法

- dp[N] を高速に計算したい
  - DPテーブルの性質を考える
  - dp[j+1] は、dp[j] を A[j] だけずらし、dp[j][1]~dp[j][A[j]] をリバースしたものを先頭につけたものになっている！

k	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
dp[3][k]	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
dp[4][k]	5	4	3	2	1	0	1	2	3	4	5	6	7	8	9	10
dp[5][k]	2	3	4	5	4	3	2	1	0	1	2	3	4	5	6	7
dp[6][k]	4	5	4	3	2	3	4	5	4	3	2	1	0	1	2	3

満点解法

- dp[N] を高速に計算したい
  - DPテーブルの性質を考える
  - dp[j+1] は、dp[j] を A[j] だけずらし、dp[j][1]~dp[j][A[j]] をリバースしたものを先頭につけたものになっている！

k	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
dp[3][k]	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
dp[4][k]	5	4	3	2	1	0	1	2	3	4	5	6	7	8	9	10
dp[5][k]	2	3	4	5	4	3	2	1	0	1	2	3	4	5	6	7
dp[6][k]	4	5	4	3	2	3	4	5	4	3	2	1	0	1	2	3

満点解法

- dp[N] を高速に計算したい
  - DPテーブルの性質を考える
  - dp[j+1] は、dp[j] を A[j] だけずらし、dp[j][1]~dp[j][A[j]] をリバースしたものを先頭につけたものになっている
  - つまり、dp[j] の先頭に dp[j][1]~dp[j][A[j]] を逆順にpushしていく、ということを繰り返していけば dp[N] が得られる
    - 先頭へのpushがしたいので、deque を使うと良い
  - dp[j] から dp[j+1] を得るときの計算量は O(A[j]) であり、dp[N] を得るときの計算量は O(sum(A[3]~A[N-1])) となる
    - 制約より、sum(A[3]~A[N-1]) ≦ 10^6 なので、間に合う

I問題 解説
「Reverse Grid」

CODE FESTIVAL 2016 本戦

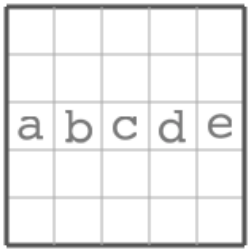
問題概要

- H*W のマス目に文字がかかれている
- 行のリバースと列のリバースを何度か行うことによって作ることの出来る盤面は何通り考えられるか
- mod 10^9+7 で求めよ

- 制約
  - $1 \leqq H,W \leqq 200$

解法

- 行数が奇数の場合
- 真ん中の行は列リバースによって変化しない
- 行リバースで変化するかどうかしか関係ない
  - 変化する場合：答え＝他の行に関する場合の数＊2
  - 変化しない場合：答え＝他の行に関する場合の数



- 列数が奇数の場合も同様に処理すれば良い
- これで、**HもWも偶数の場合**を解けば良いことになった

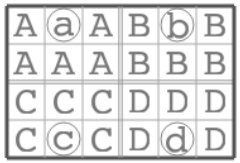
解法

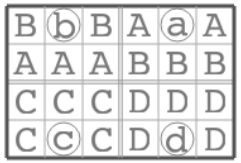
- 行と列を半分に区切り、下の図で同じ番号の付いているようなマスを**グループ**にして考える
- 行リバース・列リバース操作を行っても、あるグループにある文字が別のグループのマスへ行ったりすることはない

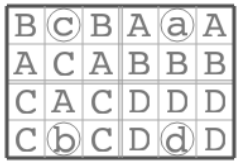
1	2	3	3	2	1
4	5	6	6	5	1
4	5	6	6	5	4
1	2	3	3	2	1

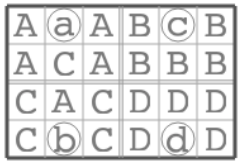
解法

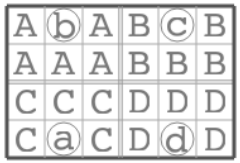
- あるグループ内の配置を、他のグループの状態を変えずに変えるような操作列を考えてみる
  - 例えば、
    - 1行目をリバース
    - 2列目をリバース
    - 1行目をリバース
    - 2列目をリバース
  - と操作を行うと、３つの文字の配置を巡回させることが出来る











解法

- あるグループ内の配置を、他のグループの状態を変えずに変えるような操作列を考えてみる
  - ３つの文字の配置を巡回させることが出来ると何が出来るか？
  - **偶置換**全体を作ることが出来る
    - 偶置換とは、偶数回のswapで作ることのできる置換
    - 奇数回のswapで作ることの出来る置換は奇置換という
    - 偶置換の個数＝奇置換の個数＝置換の個数/2
  - 逆に他のグループの状態を変えない限り偶置換しか作ることが出来ないことも証明できる

解法

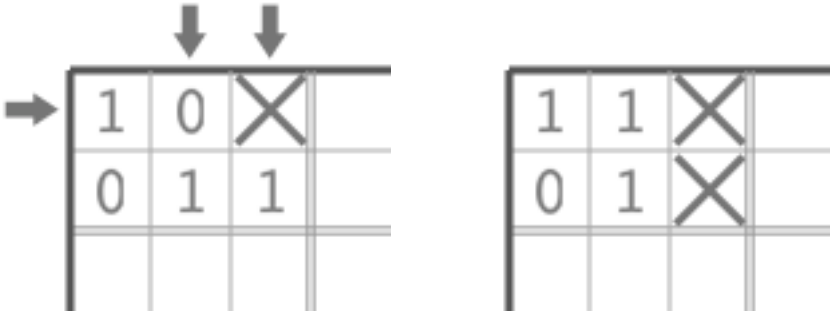
- 奇置換を作るにはどうすればいいか？
  - 行リバースと列リバースのうち、片方だけをする
  - ただしその場合、リバースした行（または列）にあるグループも同時に奇置換になる

解法

- グループ内の文字の分布ごとに場合分けする
  - “abcd” のようにすべての文字が異なる場合
    - 偶置換のみ作れる（6!/2 = 12 通り）
  - “aabc” のように同じ文字が 2 つ以上ある場合
    - すべての並べ替えが作れる（場合の数は分布により異なる）
      - ‘a’ どうしを余計にswapされていると考えると奇置換が作れるため、偶置換から出来る文字のパターンの集合と奇置換から出来る文字のパターンの集合が等しい
- 文字のパターンに偶置換か奇置換かが関係あるグループと関係ないグループが存在する

解法

- 偶置換か奇置換かが関係あるグループに関して、どれが偶置換でどれが奇置換なのかというパターンを考えたとき、実現可能なパターンは何通りあるか？
- 実現可能な例と実現不可能な例
  - 図はマス目の左上1/4の部分だけを取り出している
    - バツは偶置換か奇置換かが関係ないグループを表す
    - ０は偶置換にしたいグループを表す
    - １は奇置換にしたいグループを表す
      - ちなみに、今求めたいものはこの０と１のパターンの個数
  - 左は矢印の付いた行・列をリバースすることにより実現可能
  - 右は実現不可能



解法

- 行と列をそれぞれ頂点とした二部グラフを考える
- バツでないマスに対応する行と列の頂点を結ぶ辺を張る
- 各頂点について、０か１かを決める
  - リバースするかどうかに対応している
- 各辺について頂点の数の和を２で割った余りを求める
  - 偶置換が０、奇置換が１に対応している
- 各辺の０/１のパターンが何通りあるかを求めればよい

解法

- 各辺の０/１のパターンが何通りあるか
- 連結成分ごとに考える
  - 連結成分の全域木 T を考える
  - T の辺の０/１のパターンが決まれば他の辺のパターンは一意に定まる
    - (A+B)+(B+C) = A+C+2B ≡ A+C (mod 2)
  - T の辺の０/１のパターンは全通り実現可能
    - １つの頂点の０/１を決めると残りの頂点の０/１が一意に定まる
  - よって、頂点数 S の連結成分についてのパターン数は $2^{(S-1)}$
- グラフ全体のパターン数は $2^{(\text{頂点数}-\text{連結成分数})}$

解法

- まとめ
  - H,W が奇数なら真ん中の行/列について処理する
  - 各グループ内での文字の配置の場合の数をかけ合わせる
  - 二部グラフを作り、$2^{(H+W-連結成分数)}$ を答えにかける

- 計算量は O(H*W) であり、時間制限には余裕がある

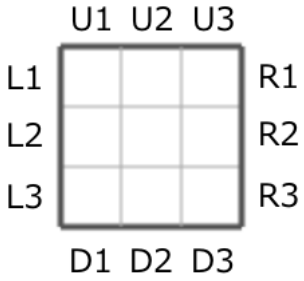
J問題 解説
「Neue Spiel」

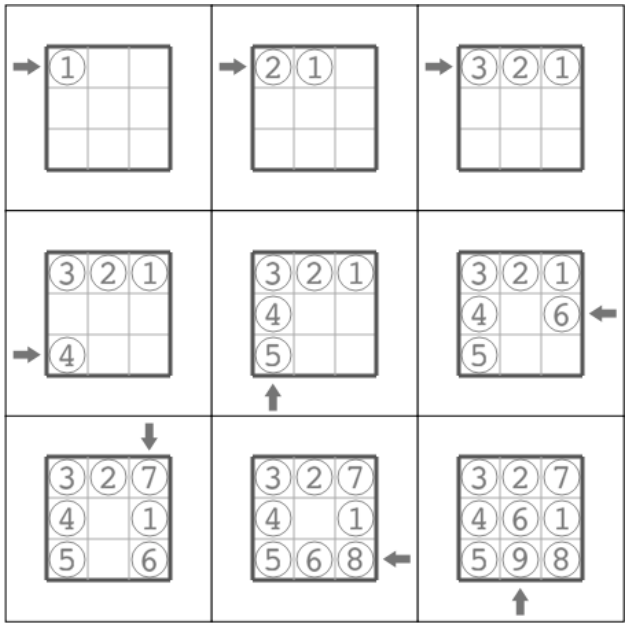
CODE FESTIVAL 2016 本戦

問題概要

- N*N のマス目がある
- 上下左右の辺から、場所ごとに決まった枚数のタイルを押し込む
- すべてのマスが埋まるような押し込み順を構成せよ

- 制約
  - 1 ≦ N ≦ 300
  - （制限時間は4秒）

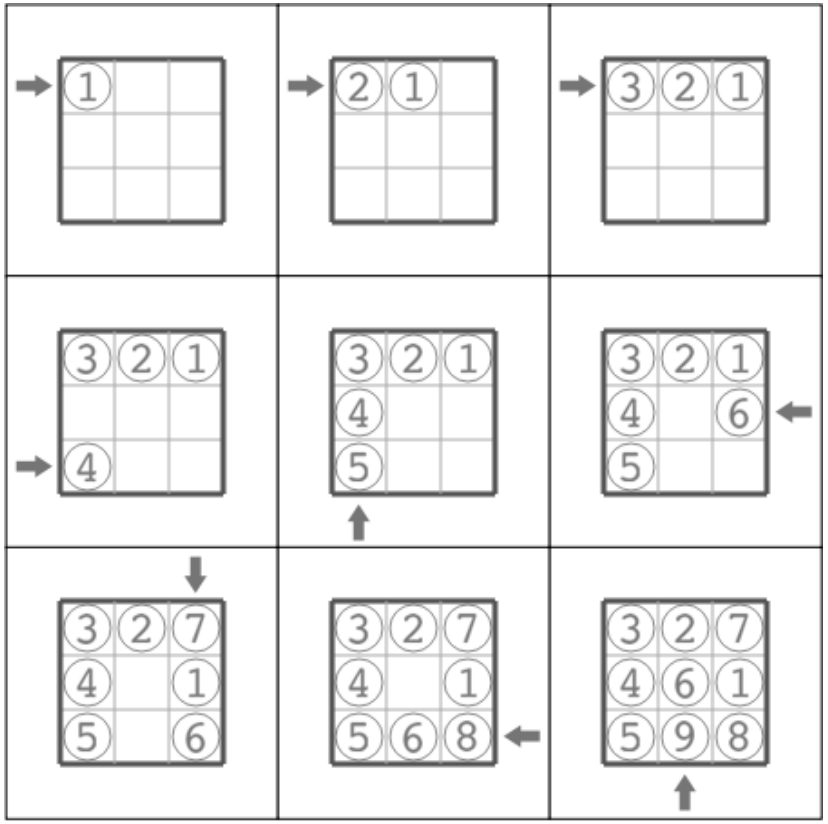




考察

- 「押し込む」操作
  - 図で i と書かれたタイルは、i 回目の操作で押し込んだタイルを表している

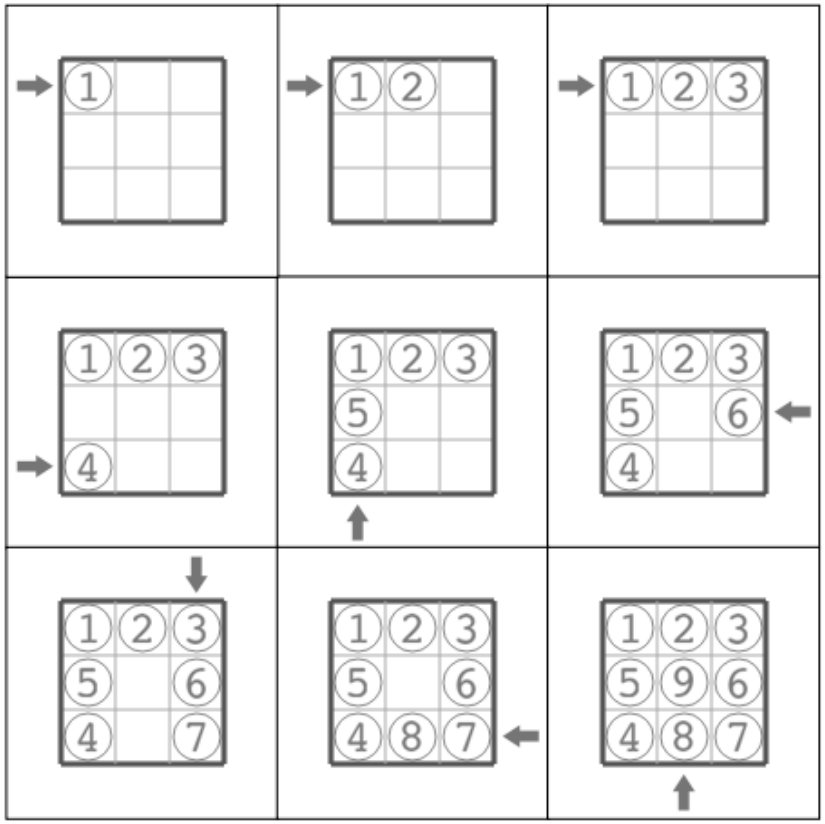
- 複雑すぎる



考察

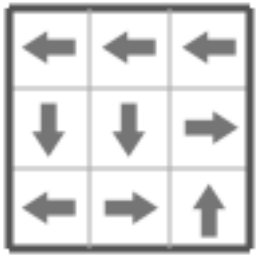
- 「押し込む」操作、ではなく
- **「空いているマスのうち最も手前のマスに置く」**操作、と言い換える

- 少し見通しが良くなった



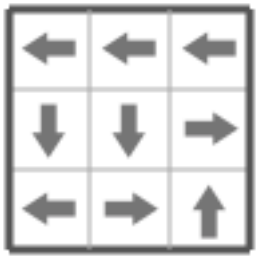
可能性判定

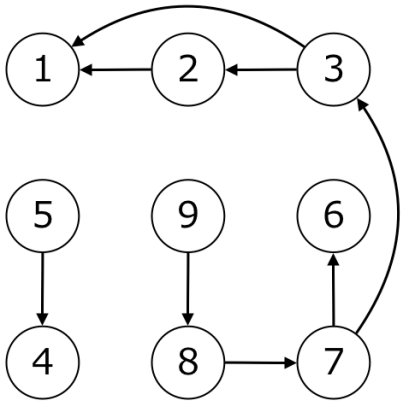
- 各マスに「どっちからタイルが来るか」を割り当てる
  - 図では矢印で表している
- 各行・列で、その向きに対応する矢印の個数が入力どおりになっているような、**矢印の割当て**をする
  - 少なくとも、割当てが出来なければ不可能
  - では、割当てが出来れば必ず出来る？



可能性判定

- 割当てが出来れば必ず出来る？
- 操作手順を復元することを考える
  - 置くことが出来るマスから埋めていく
    - 置くことが出来る＝そのマスより手前がすべてすでに埋まっている
  - 「このマスよりこのマスの方を先に埋めなければいけない」という**依存関係の有向グラフ**を考える
  - そのグラフがDAGになっていれば、トポロジカル順の逆に埋めていけば良い
- このケースは可能

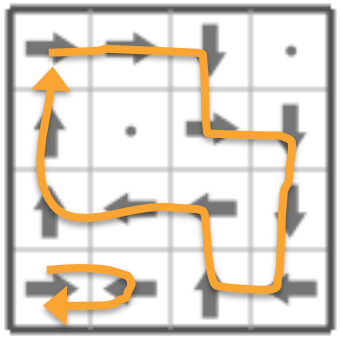


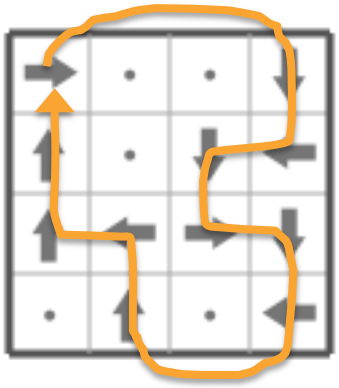


可能性判定

- 割当てが出来れば必ず出来る？
  - 依存関係のグラフにサイクルがあると**デッドロック**が起きる

- これらケースではデッドロックが起きている！

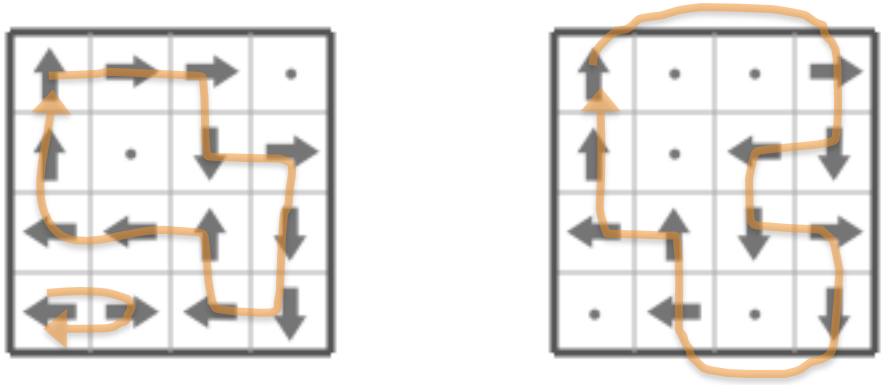




可能性判定

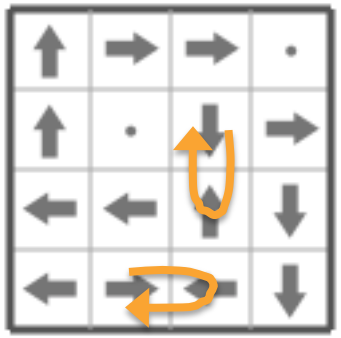
- 割当てが出来れば必ず出来る？
  - デッドロックが起きることがある

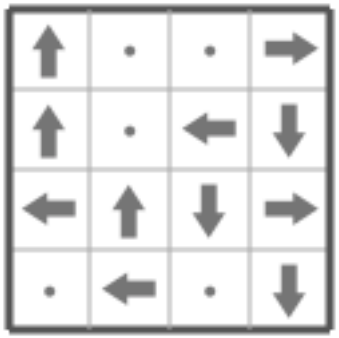
- 実は**デッドロックは必ず解消できる**
  - サイクルに含まれる矢印を順方向に１つずつ進めていく
  - 矢印の個数の制約を満たしたままデッドロックを解消できた



可能性判定

- 割当てが出来れば必ず出来る？
  - 実は、**必ず出来る！**
  - デッドロックは必ず解消できる
- ただし、デッドロックを解消した後に新たなデッドロックが発生することはある
  - 繰り返し解消していけば有限回の解消で完全になくなる
    - **「矢印からそれが指す辺までの距離」の和**は、解消操作を行うたびに減少していくため





部分点解法

- 部分点制約
  - N ≦ 40

- 「矢印からそれが指す辺までの距離」の和が**最小**となる矢印の割当てを求める
  - このような割当てにはデッドロックが存在しない
    - デッドロックがあるとすると「最小」に矛盾する

- そのような割当ての求め方
  - **最小費用流**

部分点解法

- 最小費用流のグラフ
  - Source → 差込口 (容量：差し込む個数、コスト：0)
  - 差込口 → 対応する行/列のマス (容量：1、コスト：距離)
  - マス → Sink (容量：1、コスト：0)
- 残余グラフから割当てを復元し、トポロジカルソートをする

- 計算量
  - 頂点数 $|V| = 4N + N*N + 2 \leqq 1762$
  - 辺数 $|E| = (4N + 4N*N + N*N)*2 \leqq 16320$
  - 流量 $k = N*N \leqq 1600$
  - $O(k|E| \log |V|)$ 、$k|E| \log |V| \leqq 3*10^8$ となる
    - ギリギリですが、多少遅い実装でも間に合います

満点解法

- $O(N^3)$解法があります

- 全体の流れとしては、矢印の割当てをし、デッドロックを解消しながら手順を構成するという流れ
  - 矢印の割当て：$O(N^2 \log N)$
  - 手順の構成：$O(N^3)$

満点解法

- 矢印の割当て
  - 上下左右を割り当てる代わりに縦/横(上下/左右)を割り当てる
    - 割り当てた後、縦のマスを上下に、横のマスを左右に振り分ける
- 縦/横の割当て
  - 各行と各列を頂点とし、各マスに対応する行・列の間に辺を張る二部グラフを考える
  - 辺をいくつか選び、各頂点の次数を目標の値にする
  - 行側の頂点を順番に見ていき、列側の頂点のうち残り容量の多い順に繋いでいく、という貪欲法で解ける
  - 計算量は悪くなりますが、最大流でも間に合います

満点解法

- 手順の復元
  - 基本的にはDFSと同じ
  - ただし、DFS中にデッドロック（サイクル）を検出した場合は、その場で解消し、DFSを再開する
  - 次のスライドにC++風の擬似コードを載せます
- 計算量
  - DFSの計算量にデッドロック解消分の計算量が乗る
  - DFSは $O(N^2)$
  - デッドロックの解消には O(デッドロックに含まれるマスの個数) の計算量がかかる
  - Σデッドロックに含まれるマスの個数 ≦ マスの個数*N
    - 各マスはデッドロックの解消のたびに１マス以上前進する
  - よってデッドロックの解消は合計で $O(N^3)$

満点解法

```
dir[N][N] //各マスに割当てられている矢印の向き
used[N][N] //そのマスを埋めたかどうか(初期値はfalse)
state[N][N] //そのマスがDFSの再帰のスタックに入っているかどうか(初期値はfalse)
bool dfs(i,j) { //i行目j列目、戻り値はデッドロックを検出したかどうか
  if (used[i][j]) return false;
  if (state[i][j]) {
    state[i][j] = false;
    return true;
  }
  state[i][j] = true;
  cur_dir = dir[i][j];
  for ((i',j') : (i,j)からcur_dirの向きに見たときに途中にあるマス) {
     if (!dfs(i',j')) {
       dir[i'][j'] = cur_dir;
       if (state[i][j]) {
         state[i][j] = false;
         return true;
       } else {
         return dfs(i,j);
       }
     }
  }
  state[i][j] = false;
  used[i][j] = true;
  output(i,j); //(i,j)をdir[i][j]の向きで入れる旨を出力
  return false;
}
```

Problem A
Where's Snuke?

CODE FESTIVAL 2016 Final

Problem and Solution

- Given a rectangular array of strings with H rows and W columns, find “snuke” and report its location

- Do as told
- Make sure how your language deals with strings

Problem B
Exactly N points

CODE FESTIVAL 2016 Final

Problem

- There are N problems assigned 1, 2, ..., N points
- Select a set of problems to score exactly N points...
- while minimizing the maximum score of a solved problem
- Find any such set

- Constraints
  - $1 \leqq N \leqq 10^7$

Solution

- We want to find the minimized “maximum score of a solved problem”
  - For each integer X starting from 1, test if it is possible to select a subset of {1, 2, ⋯, X} totaling N
  - The value of X when the result is “possible” for the first time, is the answer

Solution

- Testing if it is possible to select a subset of {1, 2, ⋯, X} totaling N
  - Actually, it is always possible if the sum of 1 through X is N or above
    - Let Y be a number such that the sum of 1 through Y is “barely” N or above
      - (the sum of 1 ~ Y-1) < N ≦ (the sum of 1 ~ Y)
    - Then, the sum of 1 ~ Y and N differs by less than Y
      - 0 ≦ (the sum of 1 ~ Y) – N < Y
    - Thus, we can remove at most one element from 1 ~ Y to make the sum equal to N
    - For instance, when N=12:
      - Since Y=5 and (1+2+3+4+5)-12=3, we can select 1,2,4,5
    - If the difference is 0, do not remove anything

Another Solution / Partial Score

- Another solution
  - Choose problems in descending order of assigned score, while keeping the total score at most N
    - validated by mathematical induction etc.

- Partial Score
  - can be obtained using Dynamic Programming etc.

Problem C
Interpretation

CODE FESTIVAL 2016 Final

Problem

- There are N persons and M languages
- For each person, the list of languages spoken by that person is given
- Determine if any person can communicate with any other person by speaking directly or using other persons as interpreters

- Constraints
  - $2 \leqq N \leqq 10^5$
  - $1 \leqq M \leqq 10^5$
  - Let the sum of the lists of languages be ΣK, then $\Sigma K \leqq 10^5$

Solution for Partial Score

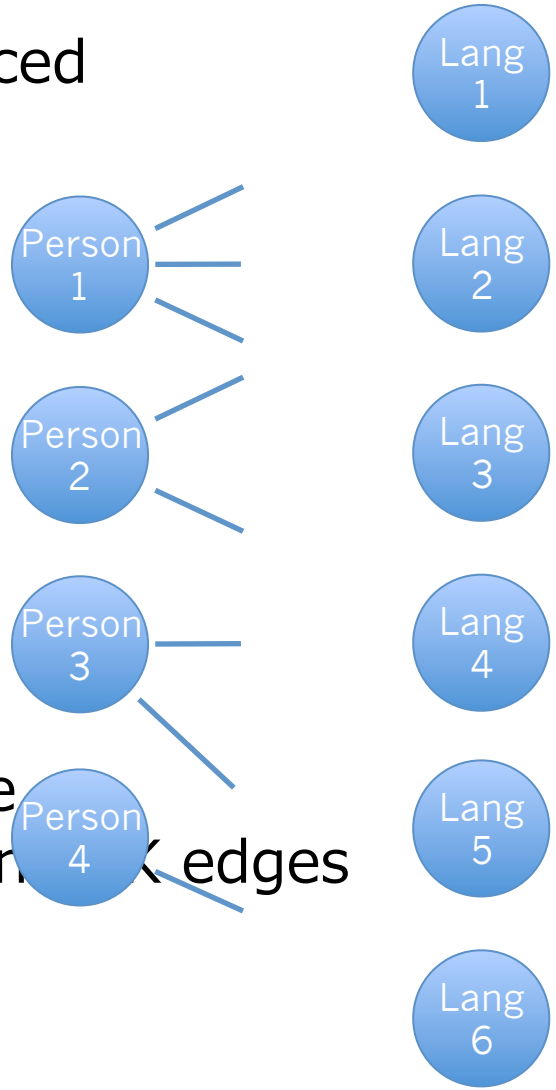
- Additional constraints for partial score
  - $N, M, \Sigma K \leqq 1000$

- Consider a graph with a vertex for each person
- Connect each pair of persons who can speak directly, with an edge
  - For each pair, connect them if there is a language spoken by both
- Check if this graph is connected
  - An easy way is using Union Find
  - DFS, BFS and such can also be used

- Not enough for full credit, since there are $O(N^2)$ edges

Solution for Full Credit

- The number of edges must be reduced
- Consider a bipartite graph with persons and languages
- What we want to know is if all persons are connected in this graph
  - It is not necessary for the languages to be connected, like language 5 in the graph to the right
- Checking connectivity is similar to that for partial score
- This solution will run within the time since there are only N+M vertices an K edges



Problem D
Pair Cards

CODE FESTIVAL 2016 Final

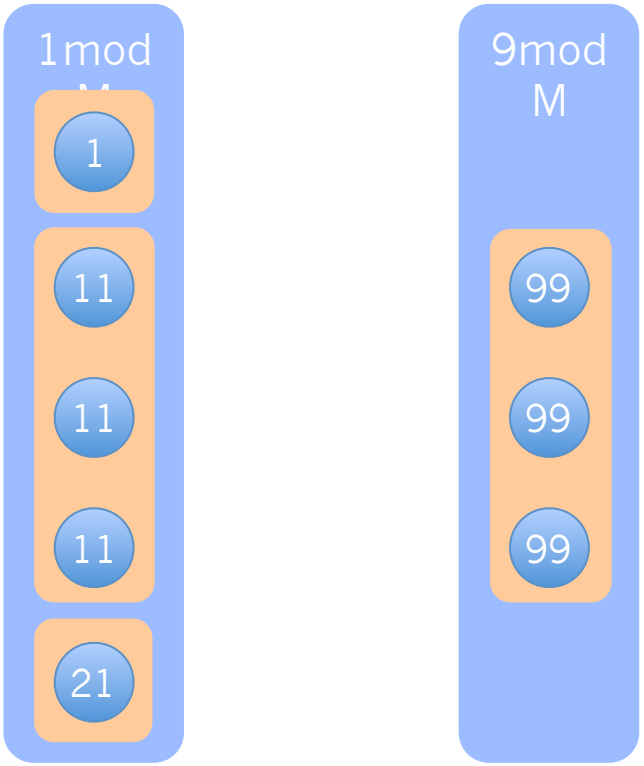
Problem

- There are N cards
- The i-th card has an integer X_i on it
- Two cards can be paired if the same integer is written on them, or the sum of the integers is a multiple of M
- How many pairs can be created at most?

- Constraints
  - $2 \leqq N \leqq 10^5$
  - $1 \leqq M \leqq 10^5$
  - $1 \leqq X_i \leqq 10^5$

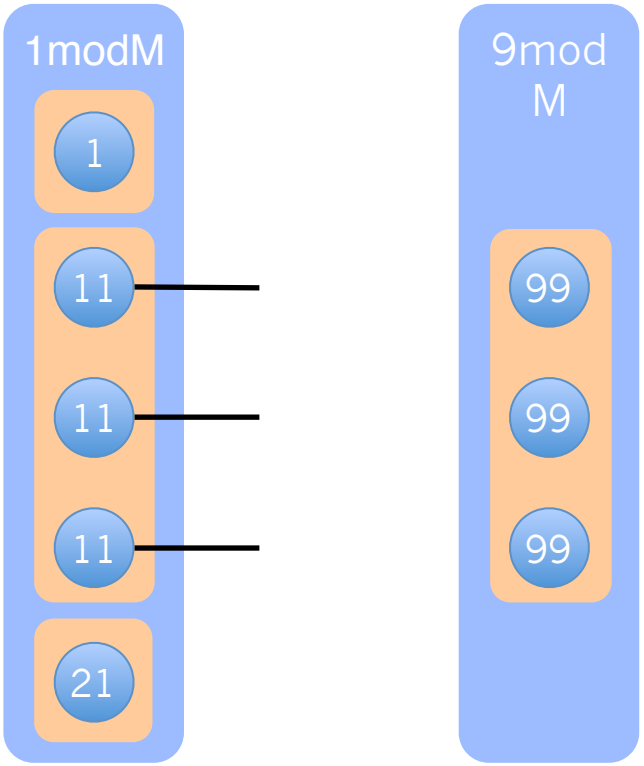
Solution

- Classify the written integers modulo M
- Example where M=10:



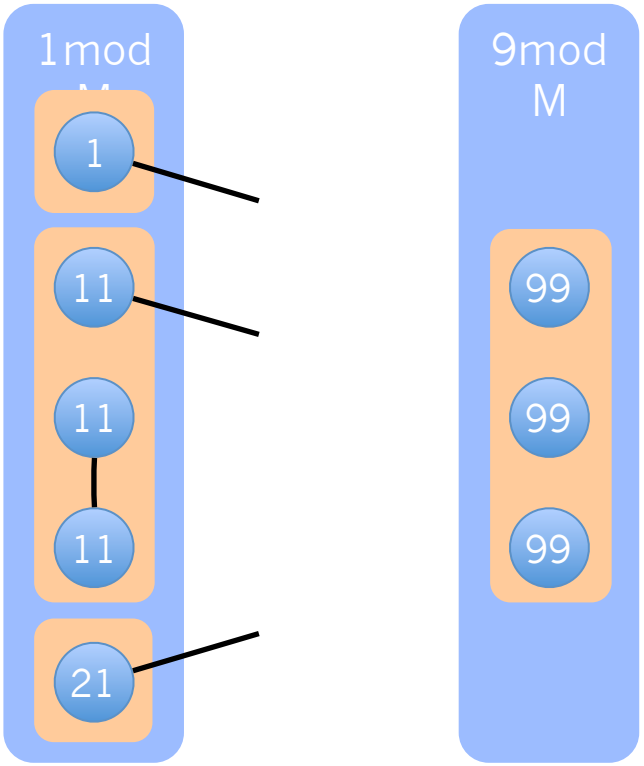
Solution

- Consider the maximum matching
- A non-maximum matching:



Solution

- A maximum matching:



Solution

- The following greedy algorithm finds a maximum matching
  - Preparation
    - Let the group of “x modulo M” be S, and “M-x modulo M” be T
    - Let |S| and |T| denote the number of elements in S and T
    - If |S| < |T|, swap S and T
      - which guarantees |T| ≦ |S|
  - The algorithm
    - match every element in T with a element in S
    - create as many pairs as possible among S, as long as at least |T| elements remains in S
      - create each pair from two equal numbers
    - Then, match all elements in T with the remaining elements in S
      - create pairs whose sums are multiples of M
  - This algorithm is validated at the end of this slide show

Solution

- The groups of “0 mod M” and “M/2 mod M”
  - For instance, groups [10, 20] and [5, 15] where M=10
  - These groups are exceptions
  - Among each of the groups “0 mod M” and “M/2 mod M”, any two elements can be paired
    - The sum will always be a multiple of M
  - Thus, (# of elements)/2 pairs can be created for each group

Solution

- Summary
  - Classify the integers modulo M
  - For the groups “0 mod M” and “M/2 mod M”:
    - (# of elements)/2 pairs can be created
  - For other groups:
    - Deal with two groups “x mod M” and “M-x mod M” at a time
    - Find a maximum matching using the greedy algorithm

Validation of the Greedy Algorithm

- Validation of the greedy algorithm
  - "match every element in T with a element in S"
    - if this part is validated, the rest is easy
  - what we must prove is: "there exists a maximum matching where every element in T is matched with a element in S"

Validation of the Greedy Algorithm

- The proof of “there exists a maximum... in S”
  - If a element is left unmatched in some maximum matching:
    - Let the unmatched element in T be x
    - Since |T| ≦ |S|, there exists a element in S that is not matched to a element in T (let this element in S be y)
      - If y is matched with another element in S, cancel that matching
        - » (in the first place, however, the matching is not maximum if y is left unmatched)
    - By matching x with y, we can construct a maximum matching where x is matched with a element in S

Validation of the Greedy Algorithm

- The proof of “there exists a maximum... in S”
  - If two elements in T is paired in some maximum matching:
    - Let the elements in the pair be x and y
    - Since |T| ≦ |S|, there exists at least two elements in S that are not matched with elements in T
      - If two elements in S is paired: cancel that pairing, and let these elements be u and v
      - If no two elements in S is paired: there exists at least two elements left unmatched, and let these elements be u and v
        - » (in the first place, however, the matching is not maximum in such a case)
    - By canceling the paring of x and y and then matching x with u and y with v, we can construct a maximum matching where x and y are matched with elements in S

Problem E
Cookies

CODE FESTIVAL 2016 Final

Problem

- A person is baking cookies
  - Initially, one cookie can be baked per second
  - When there are x cookies, by eating all of them spending A seconds, x cookies can be baked per second afterwards
- How many seconds does it take to prepare at least N cookies?

- Constraints
  - $1 \leqq N \leqq 10^{12}$
  - $0 \leqq A \leqq 10^{12}$

Solution for Partial Score

- Additional constraints for partial score
  - $N,A \leqq 10^6$

- Dynamic Programming (DP)
  - dp[i] = the minimum time it takes to prepare **exactly** i cookies
  - Initialization: dp[0] = 0, ∞ for the other values
  - Transition:
    - for (i=1; i<N; i++) for (j = i; j <= N*2; j += i)
      dp[j] = max(dp[j], dp[i]+A+j/i);
  - Answer: the minimum value among dp[N] through dp[2N]
    - Note that dp[N] is not necessarily minimum among them
  - Time complexity: O(N log N)
    - $O(\Sigma(N/i \mid 1 \leqq i \leqq N)) = O(N \log N)$

Solution for Full Credit

- Decide the number of times to eat cookies first
  - Iterating from 0 times to around 40 times is enough
    - In the optimal strategy, cookies are eaten O(log N) times

- Let the number of times cookies are eaten be k, and the consecutive number of seconds cookies are baked be s_1, s_2, ..., s_{k+1} , respectively
  - The total time taken: $A*k + (s_1+s_2+...+s_{k+1})$
  - The number of cookies prepared: $s_1*s_2*\cdots*s_{k+1}$

Solution for Full Credit

- We must find the minimum possible sum of s when the product of s is at least N
- What are the optimal values of s_i?
  - They should be as **uniform** as possible!
  - That is, the maximum and minimum values among s should differ by at most 1
    - Otherwise, by subtracting 1 from the maximum and adding 1 to the minimum, the product can be increased while keeping the sum, leading to a more efficient distribution
- How to find optimal s
  - Perform binary search on the maximum value among s (let it be m)
  - Then, use brute-force on the maximum possible number of occurrences of m-1 among s

Solution for Full Credit

- The time complexity: $O(\log^3 N)$
- Watch out for overflow when multiplying numbers

Problem F
Road of the King

CODE FESTIVAL 2016 Final

Problem

- There are N vertices
- Consider paths of length M starting from vertex 1
- How many paths p are there such that when edges are spanned along p, the resulting graph is strongly connected?

- Constraints
  - $2 \leqq N \leqq 300$
  - $1 \leqq M \leqq 300$

Observation

- Consider the condition for strong connectivity
  1. The path must visit every vertex
  2. Vertex 1 must be reachable from the vertex at the end of the path
- These two conditions are both necessary and sufficient
  - Necessity is obvious
  - The sketch of the proof of sufficiency
    - From condition 1., every vertex is reachable from vertex 1
    - From 1., the vertex at the end of the path is reachable from every vertex, and from 2., vertex 1 is reachable from every vertex
    - Thus, every vertex is reachable from every other vertex via 1, leading to strong connectivity

Solution

- We will apply Dynamic Programing (DP)
- What should be the state in DP?
  - dp[length of the path][number of the used vertices][state on the strong connectivity]
    - Imagine the transition as appending one vertex at the end of the path at a time
  - This amount of information should be enough
  - The problem is what this “state on the strong connectivity” actually is

Solution

- Observation on “state on the strong connectivity”
  - When a vertex is appended at the end of the path:
    - If appended to vertex 1, it will become strongly connected
    - If appended to a vertex from which vertex 1 is reachable, it will become strongly connected as well
      - Similarly to the previous proof on strong connectivity, it can be proved that a vertex from which vertex 1 is reachable belongs to the strongly connected component containing vertex 1
    - If appended to any other vertex?
      - Actually, the strong connectivity will not develop at all
  - which means we only have to maintain **the size of the strongly connected component containing vertex 1**!

Solution

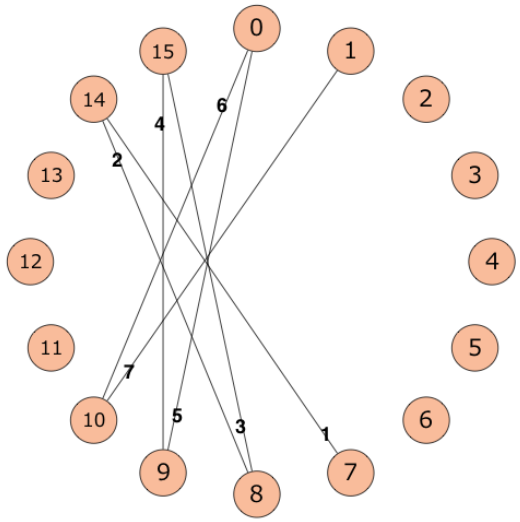
- The complete DP
  - dp[i][j][k] = (the number of paths of length i, consisting of j vertices, such that the size of the strongly connected component containing vertex 1 is k)
  - Initialization: dp[0][1][1] = 1
  - Transition:
    - dp[i+1][j+1][k] += dp[i][j][k]*(N-j)
      - Append a vertex not contained in the path at the end of the path
    - dp[i+1][j][k] += dp[i][j][k]*(j-k)
      - Append a vertex not contained in the strongly connected component containing vertex 1 at the end of the path
    - dp[i+1][j][j] += dp[i][j][k]*k
      - Append a vertex contained in the strongly connected component containing vertex 1 at the end of the path
  - Answer: dp[M][N][N]
  - Time complexity: $O(MN^2)$

Problem G
Zigzag MST

CODE FESTIVAL 2016 Final

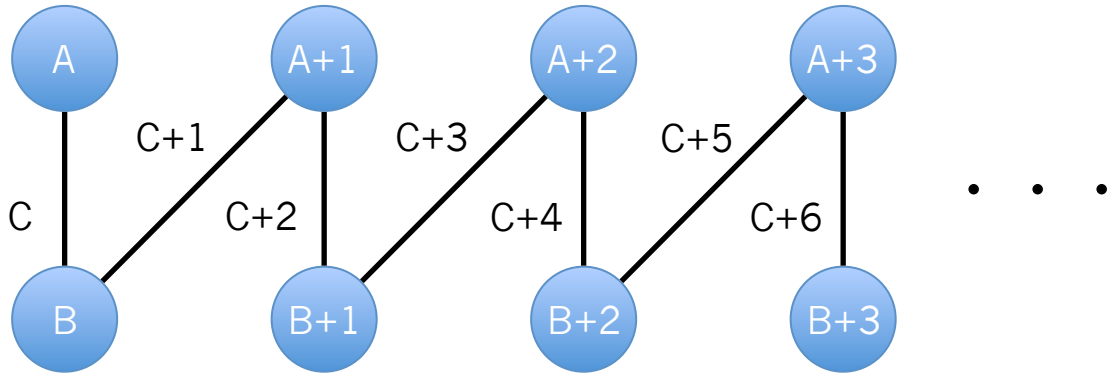
Problem

- There are N vertices
- Q queries will be processed
  - The i-th query: given A_i, B_i, C_i:
    - connect vertex A_i and B_i with an edge of cost C_i
    - connect vertex B_i and A_i+1 with an edge of cost C_i+1
    - connect vertex A_i+1 and B_i+1 with an edge of cost C_i+2
    - connect vertex B_i+1 and A_i+2 with an edge of cost C_i+3
    - …
- After all the queries are processed, find the weight of the minimum spanning tree (MST) of the graph.
- Constraints
  - $2 \leqq N \leqq 200{,}000$
  - $1 \leqq Q \leqq 200{,}000$
  - $1 \leqq C_i \leqq 10^9$



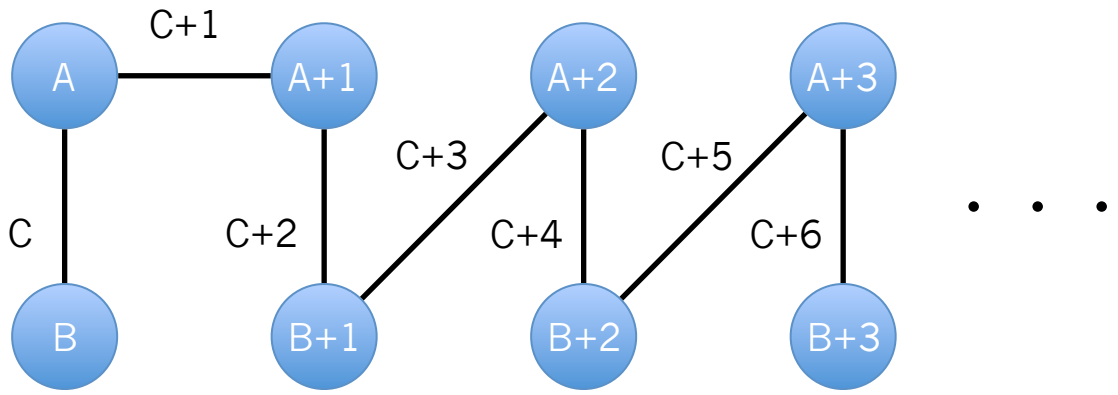
Solution

- Too many edges to directly find the MST
- We will examine the query
  - Let us apply Kruskal's algorithm to find the MST
    - On the graph below, the edges are taken into account from left to right



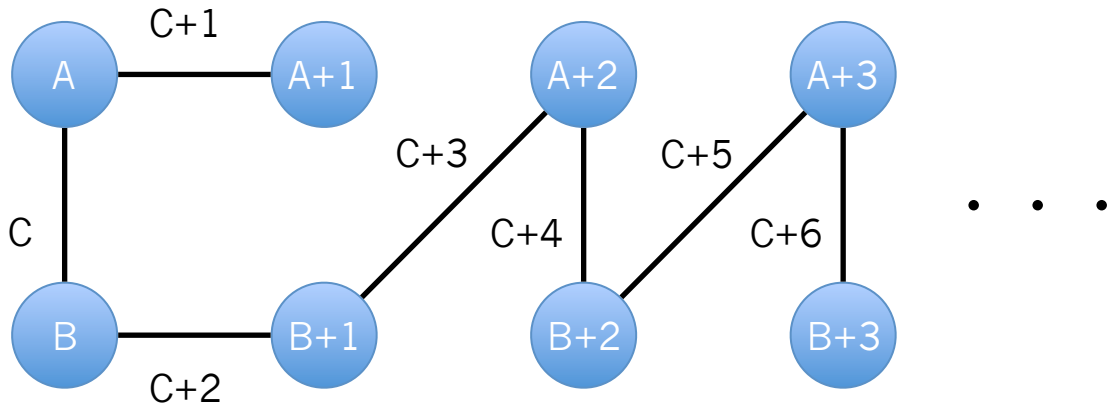
Solution

- We will examine the query
  - Let us apply Kruskal's algorithm to find the MST
  - Actually, we can relocate an edge as below without affecting the weight of MST!
    - When the edge with cost C+1 is taken into account, the edge with cost C must already be taken into account and vertices A and B must already be connected



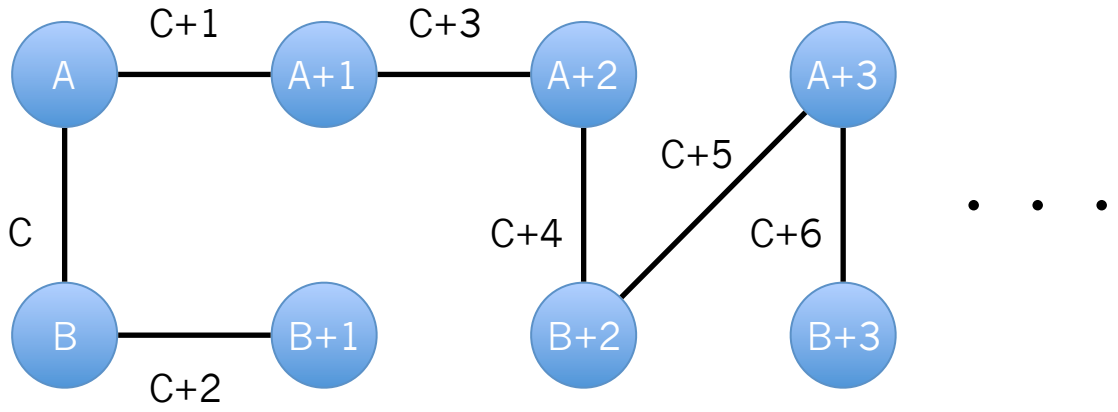
Solution

- We will examine the query
  - Relocating edges in the same way



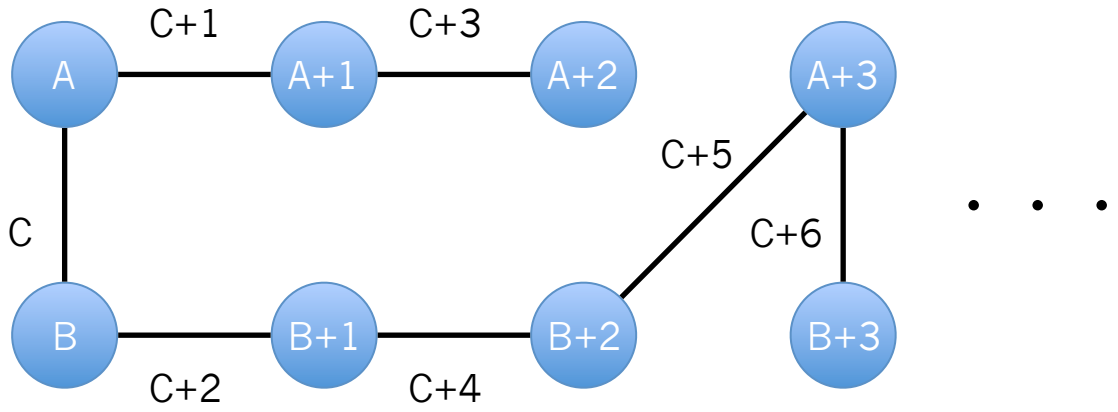
Solution

- We will examine the query
  - Relocating edges in the same way



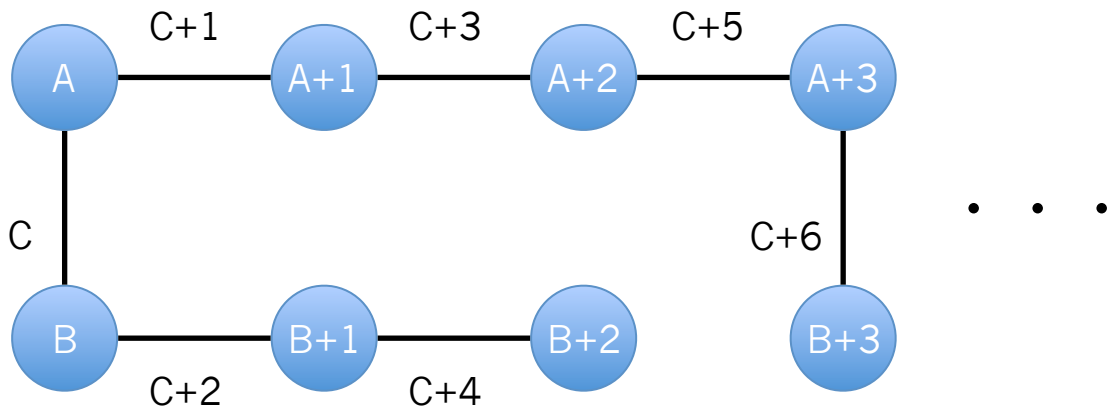
Solution

- We will examine the query
  - Relocating edges in the same way



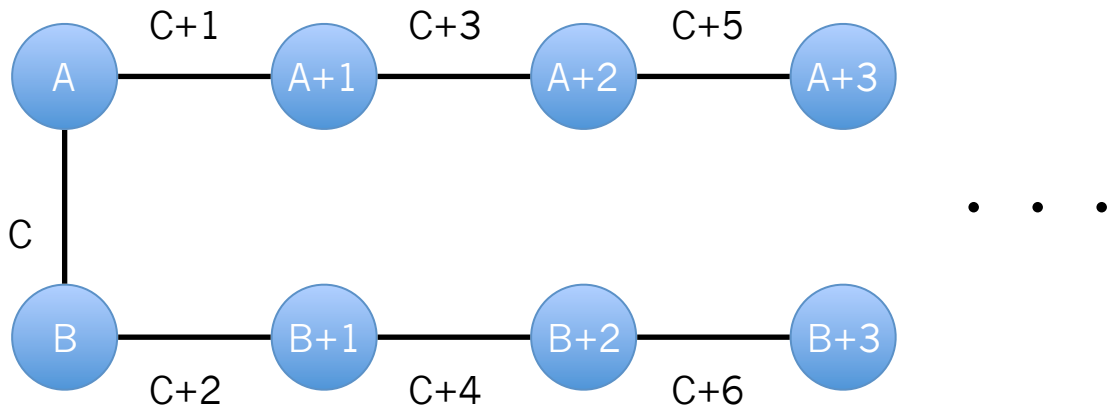
Solution

- We will examine the query
  - Relocating edges in the same way



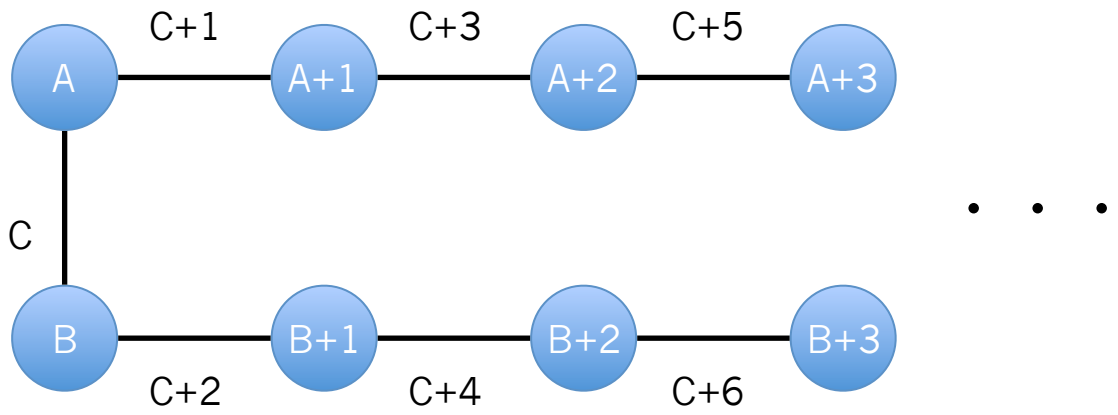
Solution

- We will examine the query
  - Relocating edges in the same way



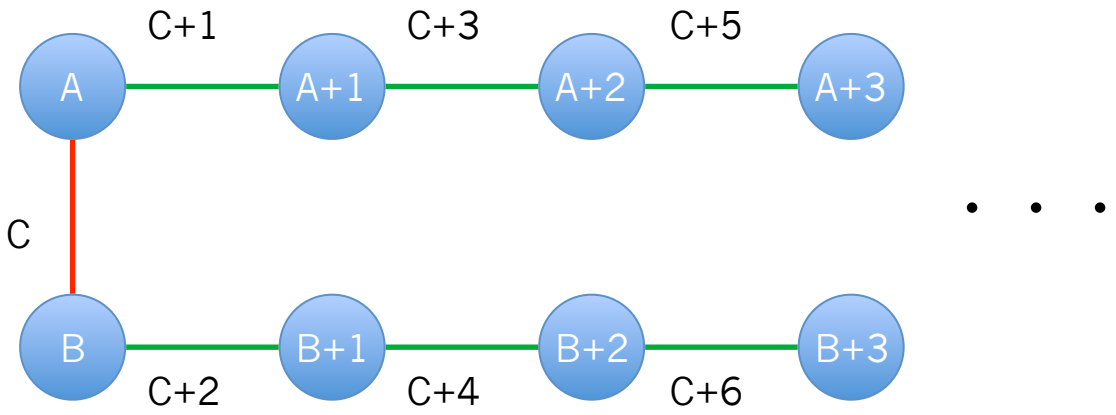
Solution

- After relocation, the edges can be classified into:
  - An edge of cost C connecting vertices A and B
  - Edges of cost C+1+2i connecting vertices A+i and A+1+i
  - Edges of cost C+2+2i connecting vertices B+i and B+1+i



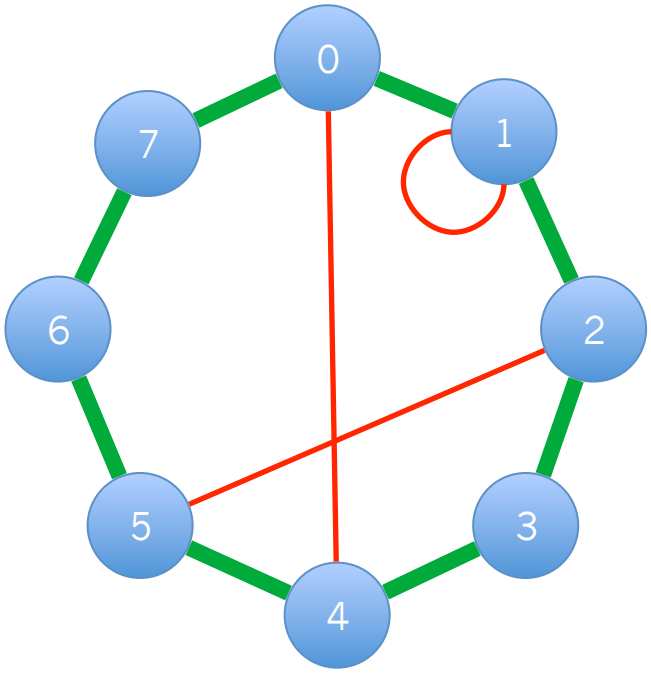
Solution

- After relocation, the edges can be classified into:
  - An edge of cost C connecting vertices A and B
  - Edges of cost C+1+2i connecting vertices A+i and A+1+i
  - Edges of cost C+2+2i connecting vertices B+i and B+1+i
- These types of edges are colored differently for illustrative purposes



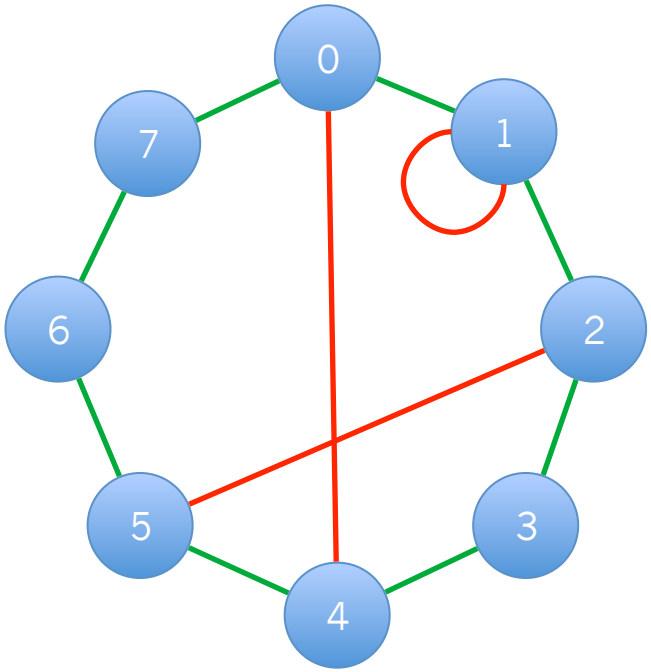
Solution

- After all the queries are processed and the edges are relocated, the graph looks as below:
  - There are infinitely many green edges where shown in green



Solution

- Among the green edges where shown in green, we can remove all but the one with the minimum weight, without affecting the weight of the MST
- Now there are only Q+N edges and we can simply find the MST



Solution

- How to find the green edge with the minimum weight where shown in green?
  - Green edges: edges with cost X+2i connecting vertices S+i and S+1+i
  - For simplicity, let us assume that green edges are spanned as follows:
    - First, connect vertices S and S+1 with an edge of cost X
    - From there, proceed clockwise spanning edges, while increasing the cost of an edge by 2 after each spanning
  - The algorithm
    - Output: c[i] = the minimum cost of an edge connecting vertices i and i+1
    1. Initialize each c[i] to ∞
    2. For each pair (S,X), perform an update: c[S] = min(c[S], X)
    3. For each i from 0 through N-1, perform an update: c[i+1] = min(c[i+1], c[i]+2). Execute this loop twice.
       - We are executing the loop twice to reflect the connection between N-1 and 0

Solution

- The time complexity
  - Finding the green edge with the minimum weight where shown in green: O(Q+N) in total
  - Finding MST afterwards: O((Q+N) log (Q+N))

Problem H
Tokaido

CODE FESTIVAL 2016 Final

Problem

- Two persons are playing a game on N squares arranged in a row from left to right, where each square is assigned an integer score
  - Each player places his piece onto square 1 and 2, respectively
  - In each turn, the player with his piece to the left of the opponent's, moves his piece. The destination must be to the right of the current position, and must not coincide with the position of the opponent's piece
  - When the pieces cannot be moved any more, the game ends
  - Each player's score is the sum of the scores of the squares where the player has placed his piece
  - Each player will plays to maximize (the player's score) - (the opponent's score)
- The score of the rightmost square is not yet determined. Given several candidates for that score, find the expected outcome of the game for each candidate
- Constraints
  - $3 \leqq N \leqq 200{,}000$
  - $0 \leqq$ (The total score assigned to non-rightmost squares) $\leqq 10^6$
  - $1 \leqq$ (The number of candidates for the score of the rightmost square) $\leqq 200{,}000$
  - $0 \leqq$ (Each candidate for the score of the rightmost square) $\leqq 10^9$

Observation

- Consider the strategy for moving the piece
- If the next square to the current position is vacant, the optimal choice is to move there
  - Since the score of a square is non-negative, you lose nothing and the next turn will also be yours
- which means the outcome of the game will be the same after modifying the rule on moving the pieces as follows:
  - “After moving his piece, the player must repeatedly move the opponent’s piece one square, until it is next to the player’s piece”
  - After each turn, the pieces are always adjacent

Solution for Partial Score

- Additional constraints for partial score
  - there is only one candidate for the score of the rightmost square

- Dynamic Programming (DP)
  - States
    - dp[i] = Starting from the situation where the player's and opponent's piece are on the (i-1)-th and i-th squares, respectively, the maximum value of "(the player's score) - (the opponent's score)"
      - Here, A[i-1] and A[i] are excluded from each player's score
      - Initialize with dp[N] = 0, and the answer will be dp[2]+A[1]-A[2]
  - Transitions
    - dp[i] = max(A[j]-sum(A[i+1]~A[j-1])-dp[j] | i<j≦N)
      - Here, A[i] denotes the score of the i-th square
- The time complexity is still too much for the time limit

Solution for Partial Score

- max(A[j]-sum(A[i+1]~A[j-1])-dp[j] | i<j≦N)
  - This needs to be calculated faster
- Observe the formula for dp[i+1]
  - max(A[j]-sum(A[i**+1**+1]~A[j-1])-dp[j] | i**+1**<j≦N)
    - This is considerably similar to the formula for dp[i]
- Let us reuse the result of the calculation
  - max(A[j]-sum(A[i+1]~A[j-1])-dp[j] | i<j≦N)
    = **max(**max(A[j]**-A[i+1]**-sum(A[i+1+1]~A[j-1])-dp[j] | i+1<j≦N),
    A[i+1]-dp[i+1])
  - That is,
  - dp[i] = max(dp[i+1]-A[i+1], A[i+1]-dp[i+1])
- The time complexity is now O(N), which is enough

Solution for Full Credit

- Examining the recurrence relation of dp:
  - dp[i] = max(dp[i+1]-A[i+1], A[i+1]-dp[i+1])
    = |dp[i+1] - A[i+1]|
  - This is a very simple formula, just subtracting A[i+1] from dp[i+1] and taking the absolute value
  - The following game will help intuitive understanding:
    - Initially, a piece is placed on the second square, and the two player's score are A[1] and A[2], respectively
    - The "initiative" is hold by either player
      - It corresponds to the player whose piece is to the left of the opponent's in the original game
    - In each turn, the player holding the initiative chooses "keep" or "change"
      - Keep: move the piece one square. The opponent gains the score of the arriving square
      - Change: move the piece one square. The player gains the score of the arriving square, then gives the initiative to the opponent

Solution for Full Credit

- DP arrays are no longer needed for the calculation
  - x = A[N];
    for (i=N-1; i >= 3; i--) x = abs(x-A[i]);
    ans = x+A[1]-A[2];
  - We will refer to this procedure as “Algorithm X”
  - We needs to efficiently find the value of ans when the value of A[N] is changed
  - The part “+A[1]-A[2]” is no longer essential, and will be ignored from now on

Solution for Full Credit

- When A[N] is extremely large
  - When A[N] has a large value such as 10^9, abs(x-A[i]) will always equals x-A[i]
  - In which case the answer will be A[N]–sum(A[3]~A[N-1])
    - Specifically, this happens when sum(A[3]~A[N-1]) ≦ A[N]
  - What remains is the case where A[N] ≦ sum(A[3]~A[N-1])
    - Note that sum(A[3]~A[N-1]) ≦ 10^6 holds from the constraints

Solution for Full Credit

- When A[N] is small
  - Consider the following DP table:
    - dp[j][k] = the eventual value of x in Algorithm X, assuming the value of x was k when the iteration for i=j is done
    - Initialized with dp[3][k] = k. The answer will be dp[N][A[N]]
    - dp[j+1][k] = dp[j][abs(k-A[j])]
  - Example: N=6, A[3]=5, A[4]=3, A[5]=4

k	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
dp[3][k]	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*	*13*	*14*	*15*
dp[4][k]	*5*	*4*	*3*	*2*	*1*	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*
dp[5][k]	*2*	*3*	*4*	*5*	*4*	*3*	*2*	*1*	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*
dp[6][k]	*4*	*5*	*4*	*3*	*2*	*3*	*4*	*5*	*4*	*3*	*2*	*1*	*0*	*1*	*2*	*3*

Solution for Full Credit

- Calculate dp[N] efficiently
  - Observe the properties of the DP table
  - dp[j+1] can be obtained by first shifting dp[j] by A[j] and then appending dp[j][1]~dp[j][A[j]] in reverse order at the beginning of it!

k	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
dp[3][k]	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
dp[4][k]	5	4	3	2	1	0	1	2	3	4	5	6	7	8	9	10
dp[5][k]	2	3	4	5	4	3	2	1	0	1	2	3	4	5	6	7
dp[6][k]	4	5	4	3	2	3	4	5	4	3	2	1	0	1	2	3

Solution for Full Credit

- Calculate dp[N] efficiently
  - Observe the properties of the DP table
  - dp[j+1] can be obtained by first shifting dp[j] by A[j] and then appending dp[j][1]~dp[j][A[j]] in reverse order at the beginning of it!

k	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
dp[3][k]	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
dp[4][k]	5	4	3	2	1	0	1	2	3	4	5	6	7	8	9	10
dp[5][k]	2	3	4	5	4	3	2	1	0	1	2	3	4	5	6	7
dp[6][k]	4	5	4	3	2	3	4	5	4	3	2	1	0	1	2	3

Solution for Full Credit

- Calculate dp[N] efficiently
  - Observe the properties of the DP table
  - dp[j+1] can be obtained by first shifting dp[j] by A[j] and then appending dp[j][1]~dp[j][A[j]] in reverse order at the beginning of it!

k	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
dp[3][k]	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
dp[4][k]	5	4	3	2	1	0	1	2	3	4	5	6	7	8	9	10
dp[5][k]	2	3	4	5	4	3	2	1	0	1	2	3	4	5	6	7
dp[6][k]	4	5	4	3	2	3	4	5	4	3	2	1	0	1	2	3

Solution for Full Credit

- Calculate dp[N] efficiently
  - Observe the properties of the DP table
  - dp[j+1] can be obtained by first shifting dp[j] by A[j] and then appending dp[j][1]~dp[j][A[j]] in reverse order at the beginning of it
  - That is, dp[N] can be obtained by repeatedly pushing dp[j][1]~dp[j][A[j]] in reverse order at the beginning of dp[j]
    - We will use deque to push elements at the beginning of the sequence
  - It takes O(A[j]) time to obtain dp[j+1] from dp[j], thus it takes O(sum(A[3]~A[N-1])) time in total to obtain dp[N]
    - From the constraints, sum(A[3]~A[N-1]) ≦ 10^6, which will suffice

Problem I
Reverse Grid

CODE FESTIVAL 2016 Final

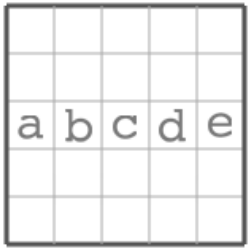
Problem

- There is an HxW grid containing a character in each square
- How many placement of characters can be obtained by reversing rows and columns in any order, any number of times?
- Answer modulo 10^9+7

- Constraints
  - $1 \leqq H,W \leqq 200$

Solution

- When the number of rows is odd:
- The middle row is not affected by column-reverse
- It is only affected by row-reverse
  - If row-reverse can actually alter the middle row: the answer is (the number of placement for other rows) * 2
  - Otherwise: the answer is (the number of placement for other rows)



- The same goes for odd columns
- Now we only have to cc **both H and W are even**

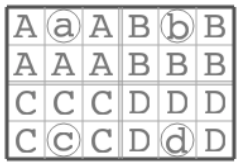
Solution

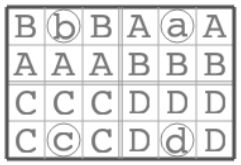
- Divide the rows and columns in half, and consider squares assigned the same number in the figure below, as a **group**
- Row-reverse and column-reverse will not transfer a character to a square belonging to a different group from the current one

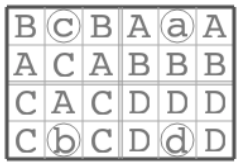
1	2	3	3	2	1
4	5	6	6	5	1
4	5	6	6	5	4
1	2	3	3	2	1

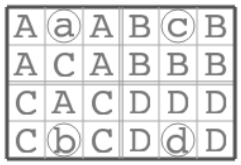
Solution

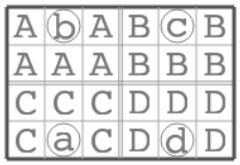
- Let us think of a sequence that will change the placement of characters within some group, without affecting the other groups
  - For instance:
    - Reverse the first row
    - Reverse the second column
    - Reverse the first row again
    - Reverse the second column again
  - In this way, we can perform a circular shift of three characters











Solution

- Let us think of a sequence that will change the placement of characters within some group, without affecting the other groups
  - What can be done from circular shifts of three characters?
  - We can obtain all the **even permutations**
    - An even permutation is an permutation obtained by even number of swaps
    - An odd permutation is similarly defined
    - It is known that (# of even permutations) = (# of odd permutations) = (# of all permutations) / 2
  - Conversely, it can also be proven that only even permutations can be obtained unless it is allowed to affect other groups

Solution

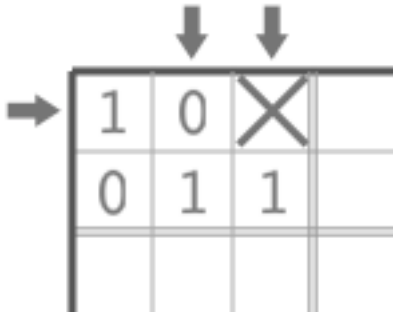
- How to obtain odd permutations?
  - Perform exactly one of row-reverse and column-reverse
  - In that case, however, the groups in the reversed row/column will also produce only odd permutations

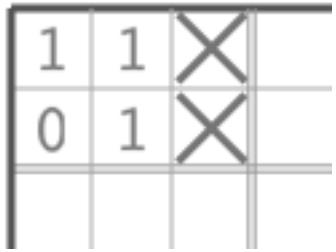
Solution

- Classify groups according to distribution of characters
  - If all four characters are distinct, as in “abcd”:
    - Only even permutations will be produced（6!/2 = 12 permutations)
  - If there are duplicate characters, as in “aabc”:
    - All permutations can be obtained (the number of permutation depends on the distribution)
      - Since it is possible to “waste” a swap on two identical characters, the even permutations and the odd permutations produce the same set of possible placement of characters
- Depending on the distribution of characters, some groups are affected by the parity of permutations, while the others are not.

Solution

- For groups affected by the parity of permutations, how many patterns of parity on each of those groups are obtainable?
- Examples of valid and invalid patterns
  - The figures show the upper left part of the grid
    - A cross represents a group not affected by the parity
    - 0 represents a group to be even
    - 1 represents a group to be odd
      - What we are finding now is the number of the binary patterns
  - The one to the left is obtainable by reversing the rows/columns indicated by arrows
  - The one to the right is unobtainable





Solution

- Consider a bipartite graph with a vertex for each row and each column
- For each square that is not a “cross” (in the previous figure), connect the corresponding row vertex and column vertex by an edge
- For each vertex, choose 0 or 1
  - It corresponds to whether the row/column is reversed
- For each edge, find the sum of the numbers on the vertices it connects, modulo 2
  - 0 corresponds to even permutations, and 1 corresponds to odd permutations
- We want to find the number of binary patterns on the edges

Solution

- How to find the number of binary patterns?
- Deal with each connected component separately
  - Consider a spanning tree T for the connected component
  - If the binary pattern on the edges of T is determined, the pattern on the remaining edges will be uniquely determined
    - (A+B)+(B+C) = A+C+2B ≡ A+C (mod 2)
  - Any binary pattern on the edges of T is obtainable
    - When a value for one vertex is determined, the values for the other vertices will be uniquely determined
  - Thus, the number of obtainable patterns within a connected component with S vertices is $2^{(S-1)}$
- The number of obtainable patterns for the whole graph is $2^{((\#\text{ of vertices}) - (\#\text{ of connected components}))}$

Solution

- Summary
  - If H and/or W are odd, process the middle row/column
  - Multiply the number of the obtainable placements of characters within each group
  - Construct a bipartite graph, then multiply $2^{(H+W-(\#\text{ of connected components}))}$ to the product computed above

- The time complexity is O(H*W), which is more than enough

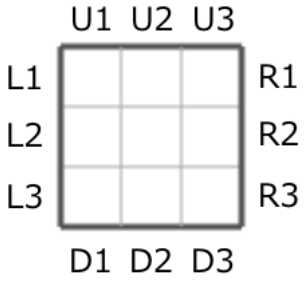
Problem J
Neue Spiel

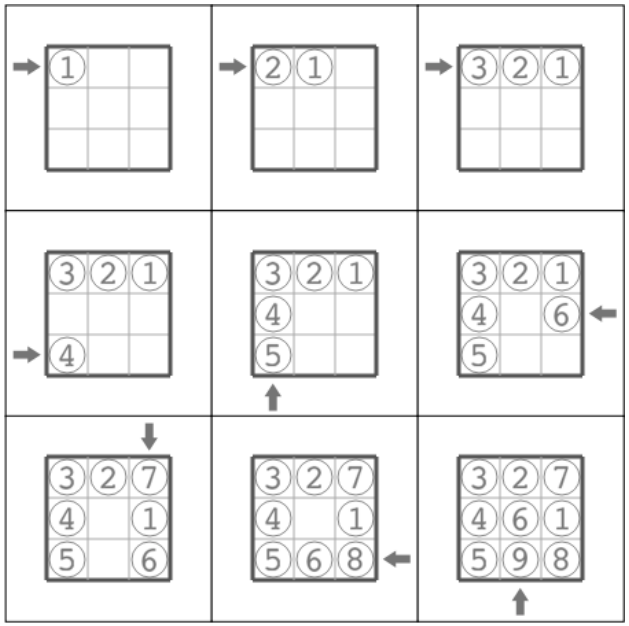
CODE FESTIVAL 2016 Final

Problem

- We have an NxN grid
- From each of the 4N sockets around the grid, we push a fixed number of panels into the grid
- In what order should panels be pushed into the grid to fill all the squares?

- Constraints
  - 1 ≦ N ≦ 300
  - (4 seconds per case)

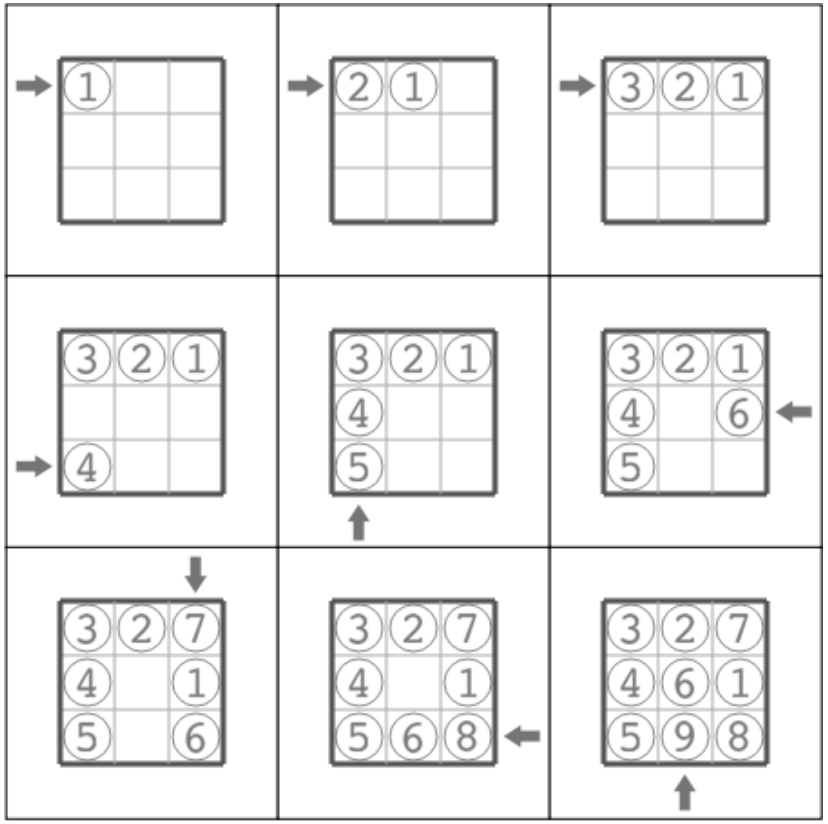




Observation

- The operation of pushing a tile from a socket
  - In the figure, a circle with a number i represents the tile inserted by the i-th operation

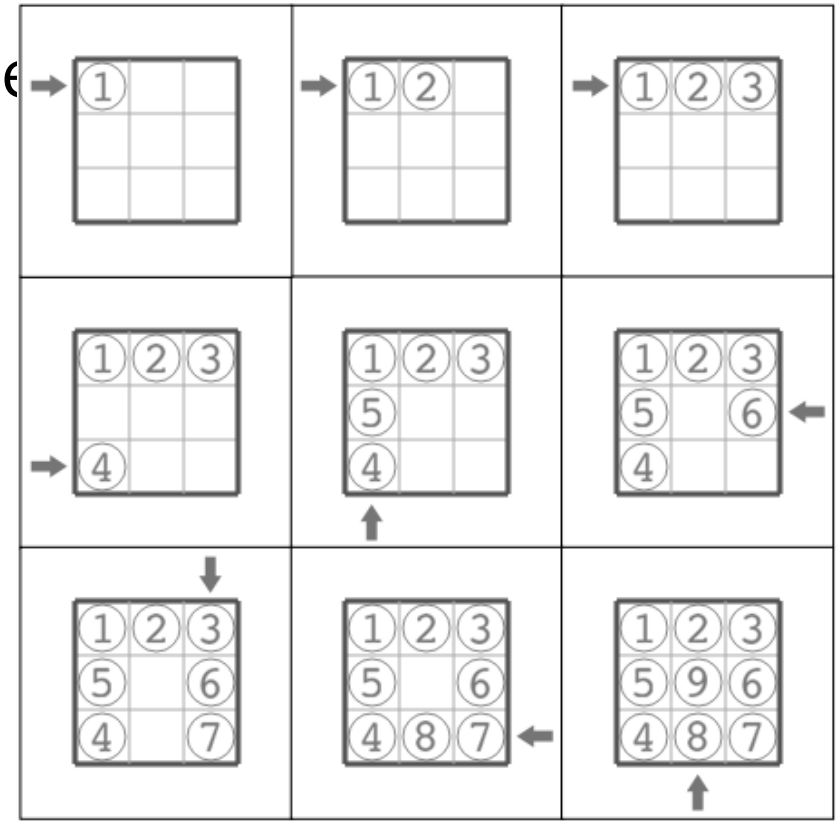
- Too complicated!



Observation

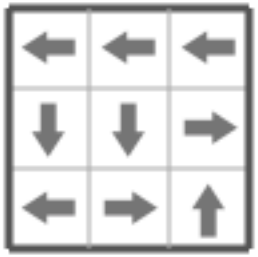
- Instead of pushing a tile from a socket...
- We can simply **directly put a tile into the nearest vacant square from the socket**

- Now it looks manageable



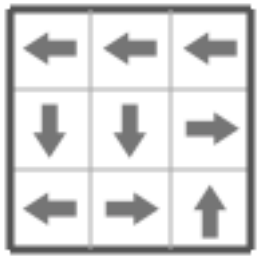
Determining Possibility

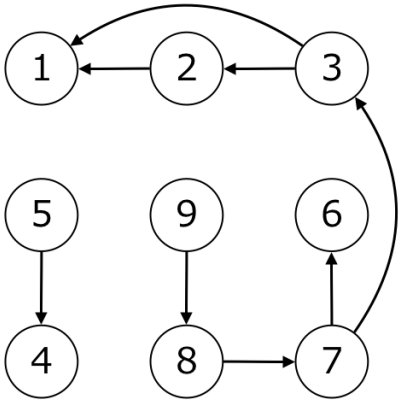
- To each square, we allocate a direction from which a tile comes
  - Indicated by an arrow in the figure
- We must **allocate arrows** to each square so that for each row and each column, the number of arrows in each direction is consistent with the input
  - If we cannot allocate arrows, the answer is “Impossible”
  - If we can, is it always possible?



Determining Possibility

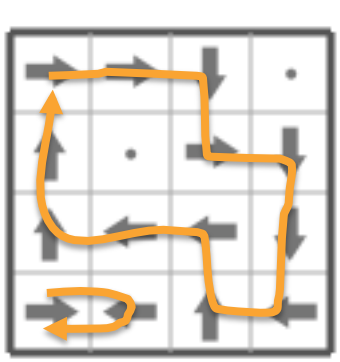
- If we can allocate arrows, is it always possible?
- Let us try to restore the sequence of operations
  - Repeatedly fill the “ready” squares
    - a square is ready when all closer squares are occupied
  - Consider the relationships “square A must be filled before square B” as a **directed graph**
  - If that graph is a DAG, we can fill all the squares in reverse topological order
- Possible in this case:

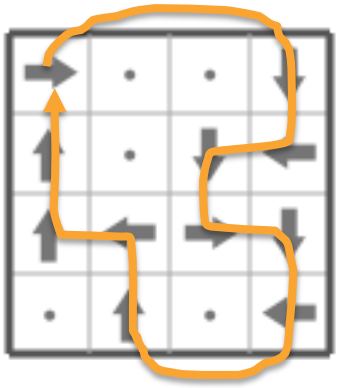




Determining Possibility

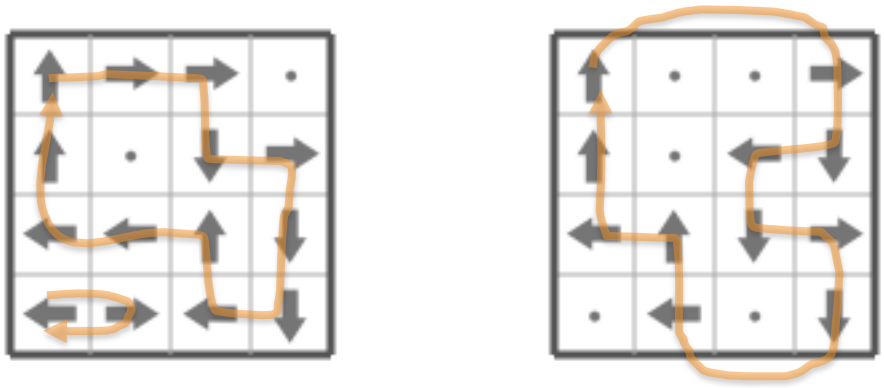
- If we can allocate arrows, is it always possible?
  - If there are cycles in the relational graph, **deadlocks** occur
- Examples of deadlocks:





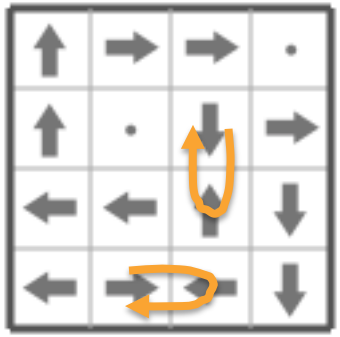
Determining Possibility

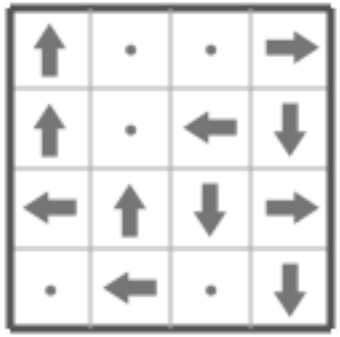
- If we can allocate arrows, is it always possible?
  - Deadlocks may occur
- Actually, we can always **eliminate deadlocks**
  - Shift the arrows along the cycle by one square
  - The deadlocks are eliminated while keeping the constraints on the numbers of arrows



Determining Possibility

- If we can allocate arrows, is it always possible?
  - **Yes!**
  - We can always eliminate deadlocks
- However, when deadlocks are eliminated, new deadlocks may occur
  - They will disappear after a finite number of elimination
    - This is because **the sum of the distances from the arrows to the sockets** decreases after each elimination





Solution for Partial Score

- Additional constraints for the partial score
  - $N \leqq 40$
- Find the allocation of arrows where the sum of “the distance from each arrow to the corresponding socket” is **minimum** possible
  - There is no deadlock in such allocation
    - If there are deadlocks, the total distance is not minimized
- Such allocation is found by...
  - **Minimum Cost Flow**

Solution for Partial Score

- Constructing the graph
  - Source → Sockets (Capacity: # of tiles to insert, Cost: 0)
  - Sockets → Squares in the row/column (Capacity: 1, Cost: the distance)
  - Squares → Sink (Capacity: 1, Cost: 0)
- Restore the allocation from the residual graph, then perform topological sort

- The time complexity
  - # of vertices $|V| = 4N + N*N + 2 \leqq 1762$
  - # of edges $|E| = (4N + 4N*N + N*N)*2 \leqq 16320$
  - The value of flow $k = N*N \leqq 1600$
  - $O(k|E| \log |V|)$, where $k|E| \log |V| \leqq 3*10^8$
    - Close, but there is room for slower implementation to pass

Solution for Full Credit

- There is an $O(N^3)$ solution
- In summary, we first allocate arrows to the squares, then construct the procedure of insertion while eliminating deadlocks
  - Allocating arrows: $O(N^2 \log N)$
  - Constructing the procedure : $O(N^3)$

Solution for Full Credit

- Allocating arrows
  - Instead of directly allocating the four directions, we first allocate "V" (vertical) or "H" (horizontal)
    - Then, allocate Up/Down to V and Left/Right to H
- Allocating V/H
  - Consider the bipartite graph where there is a vertex for each row and each column, and there is a edge for each square connecting the corresponding row and column
  - Choose a set of edges so that the degree of each vertex is equal to the specified value
  - Solved by scanning the "row" vertices from top to bottom, greedily connecting them to the "column" vertices in decreasing order of remaining capacity
  - Although slower, Maximum Flow will also run in time

Solving the Full Task

- Restoring the procedure
  - Similar to DFS
  - However, when a deadlock (cycle) is detected during DFS, eliminate it on the spot and resume DFS
  - C++-esque pseudocode is on the next slide
- The time complexity
  - The time complexity of eliminating deadlocks on top of DFS
  - $O(N^2)$ for DFS itself
  - Eliminating a deadlock takes O(# of squares in the deadlock)
  - Σ(# of squares in the deadlock) $\leqq$ (# of all the squares)*N
    - Each time a deadlock is eliminated, each square advances at least one square
  - Thus, $O(N^3)$ in total for eliminating deadlocks

Solving the Full Task

```
dir[N][N] // direction allocated to the square
used[N][N] // whether the square is filled (false initially)
state[N][N] // whether the square is in the stack of DFS (false initially)
bool dfs(i,j) { // i: row, j: column, returns whether a deadlock is detected
  if (used[i][j]) return false;
  if (state[i][j]) {
    state[i][j] = false;
    return true;
  }
  state[i][j] = true;
  cur_dir = dir[i][j];
  for ((i',j') : squares to the cur_dir of (i,j)) {
     if (!dfs(i',j')) {
       dir[i'][j'] = cur_dir;
       if (state[i][j]) {
         state[i][j] = false;
         return true;
       } else {
         return dfs(i,j);
       }
     }
  }
  state[i][j] = false;
  used[i][j] = true;
  output(i,j); // reports that the direction of (i,j) is dir[i][j]
  return false;
}
```