ボードゲーム

解説担当：西本将樹（maspy）

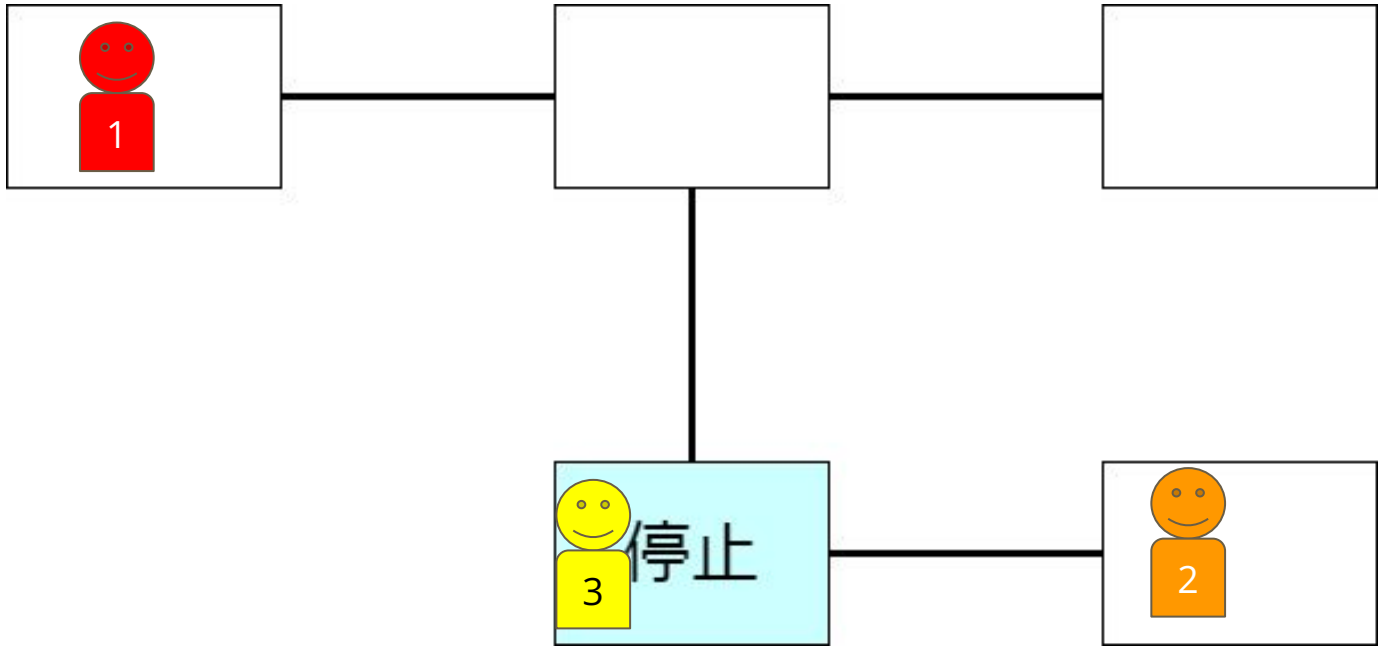
問題概要

- K 個のコマがある
- 動かし方のルールがある（問題文をよく読みましょう）
- 目標を達成するまでの最小操作回数を求めよ

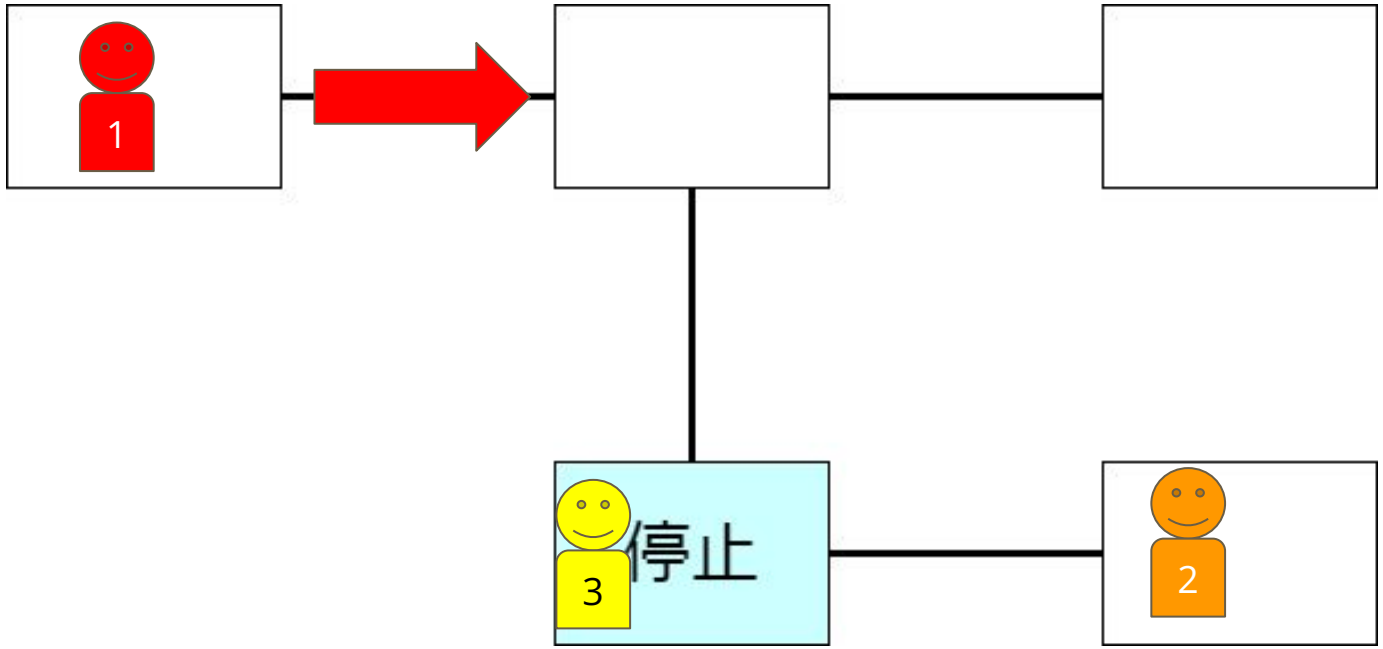
K 人ゲームと言っているが，勝負要素等はなし．

1 人が動かすと思ってもよい．

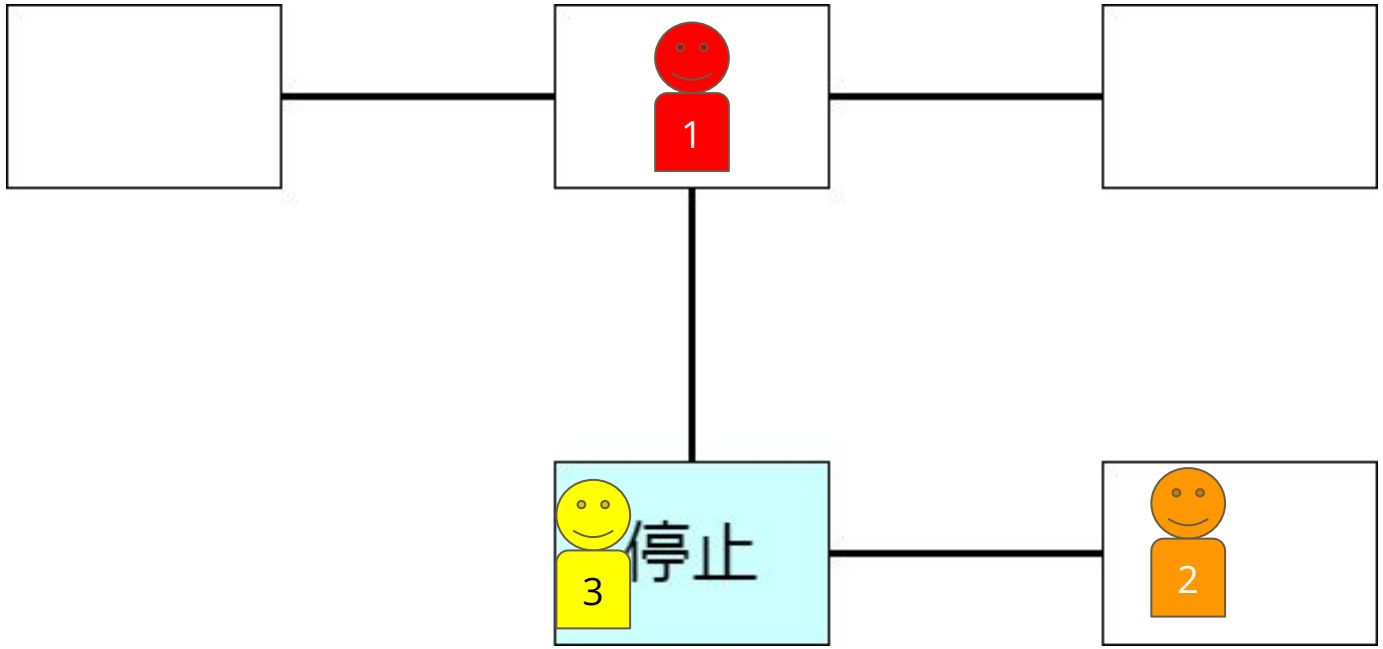
問題概要（動かし方のルール）



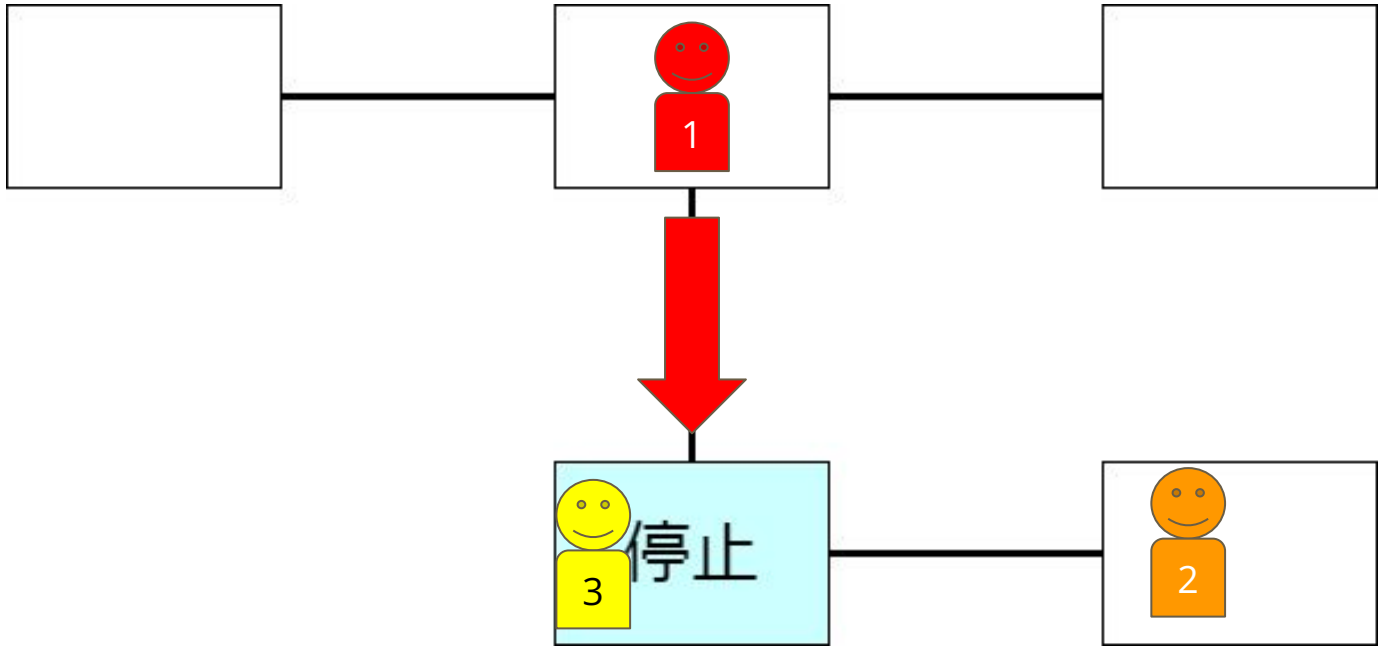
問題概要（動かし方のルール）



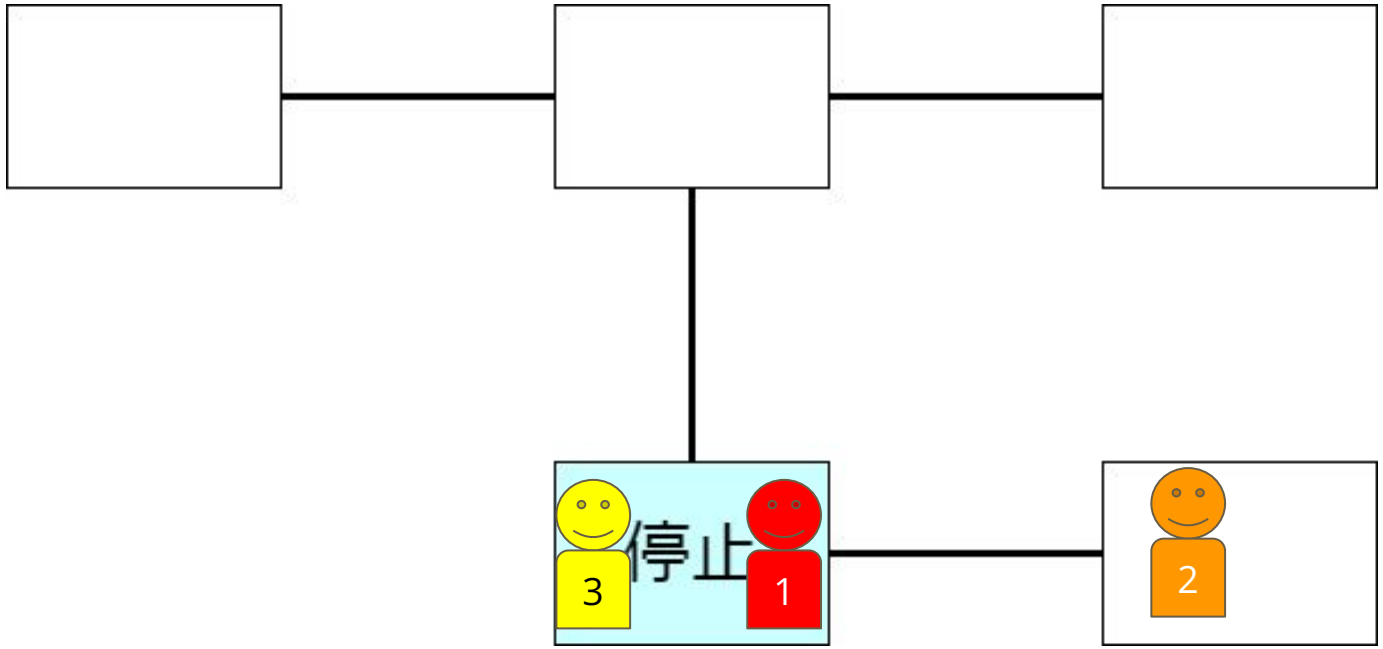
問題概要（動かし方のルール）



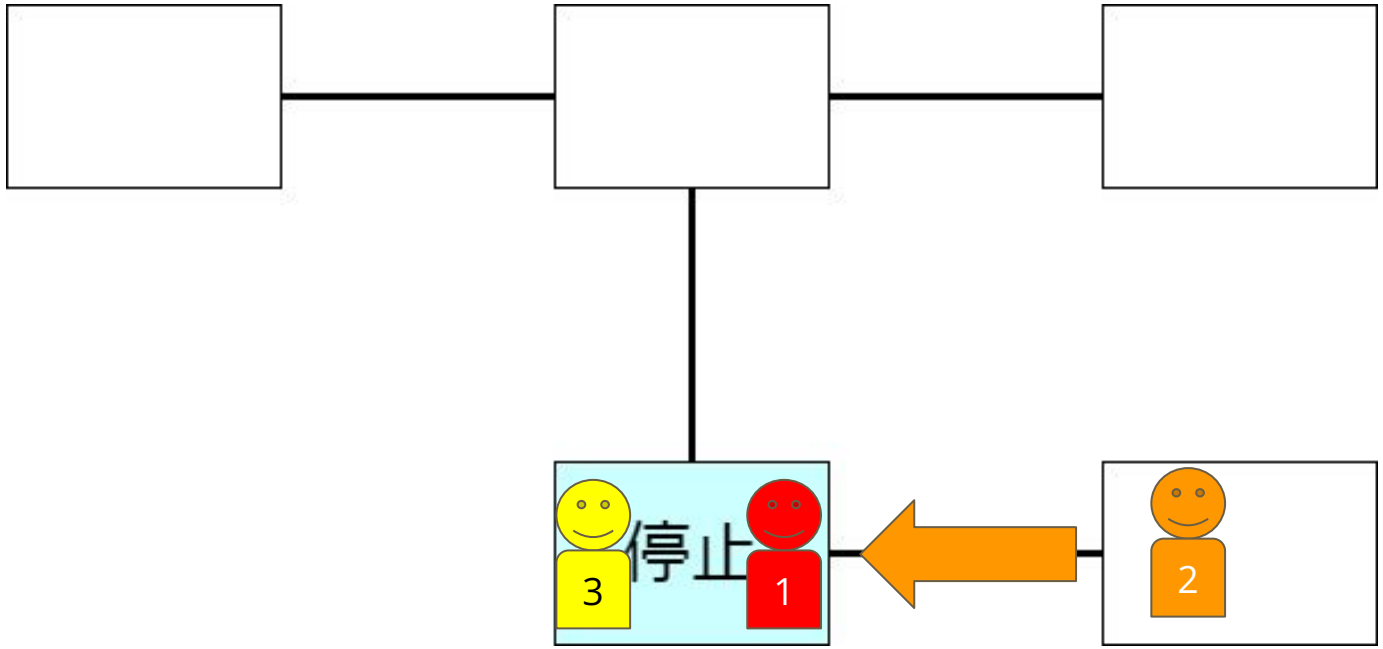
問題概要（動かし方のルール）



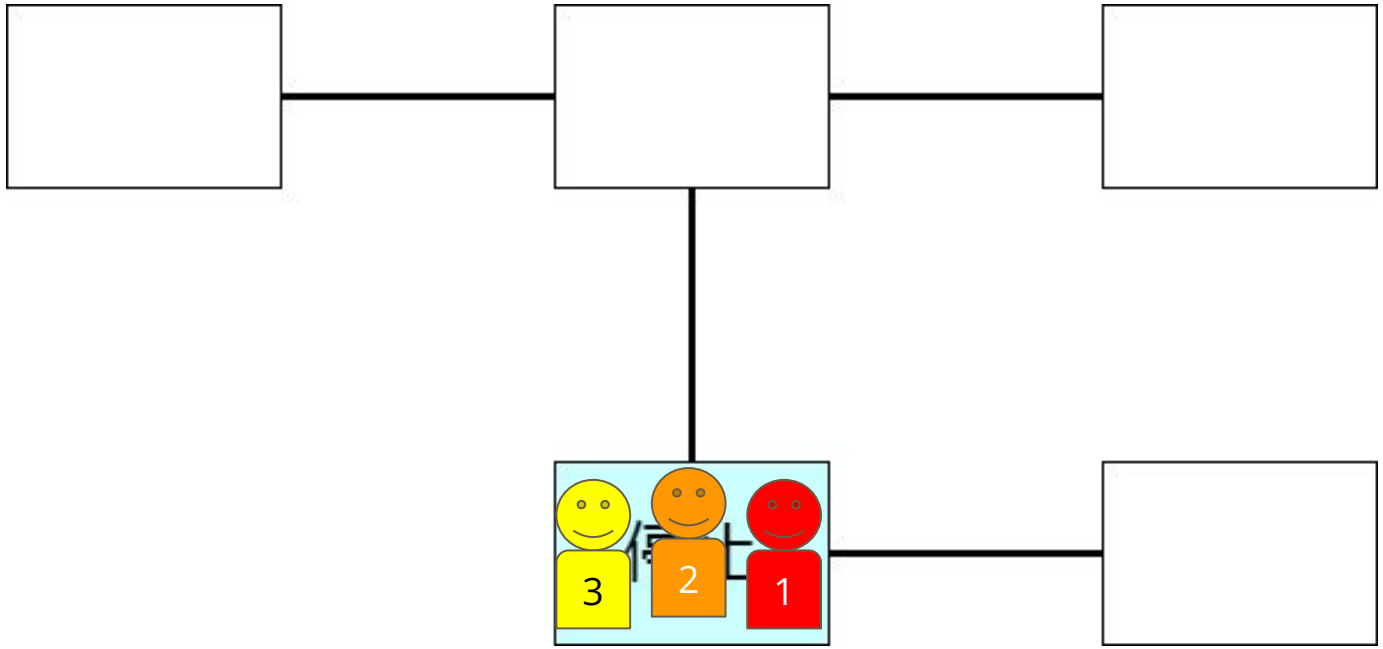
問題概要（動かし方のルール）



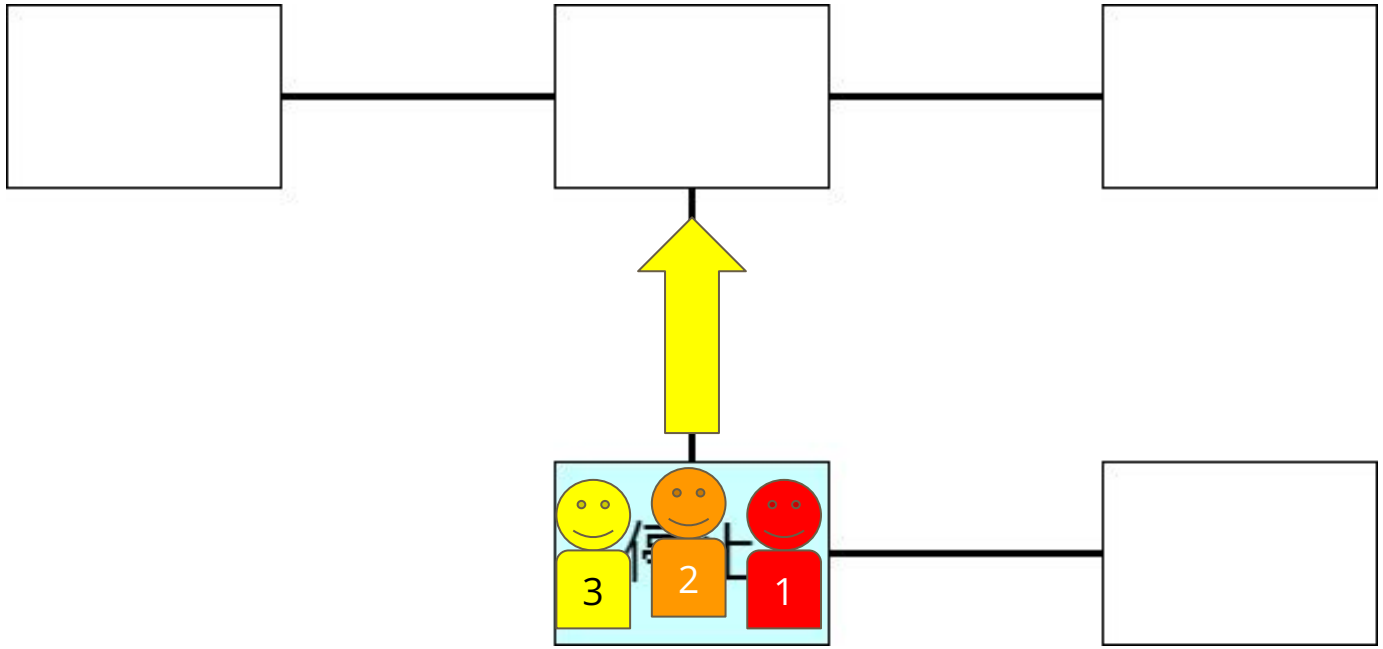
問題概要（動かし方のルール）



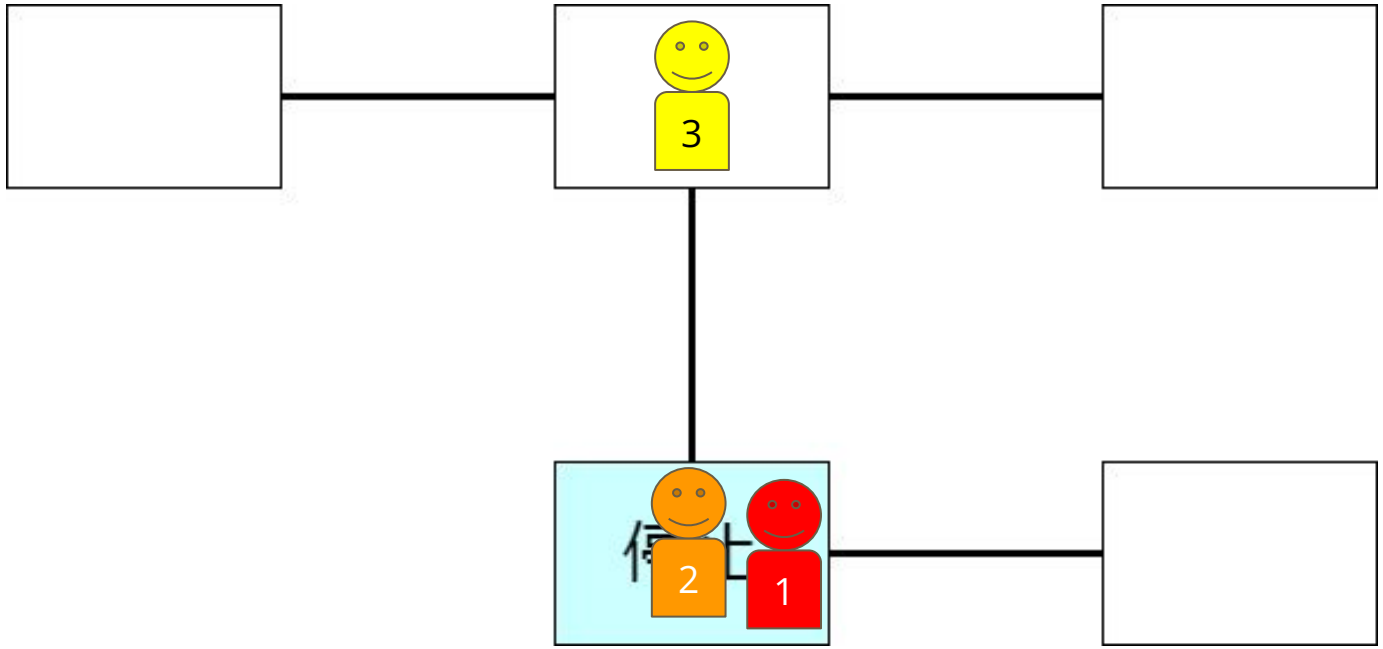
問題概要（動かし方のルール）



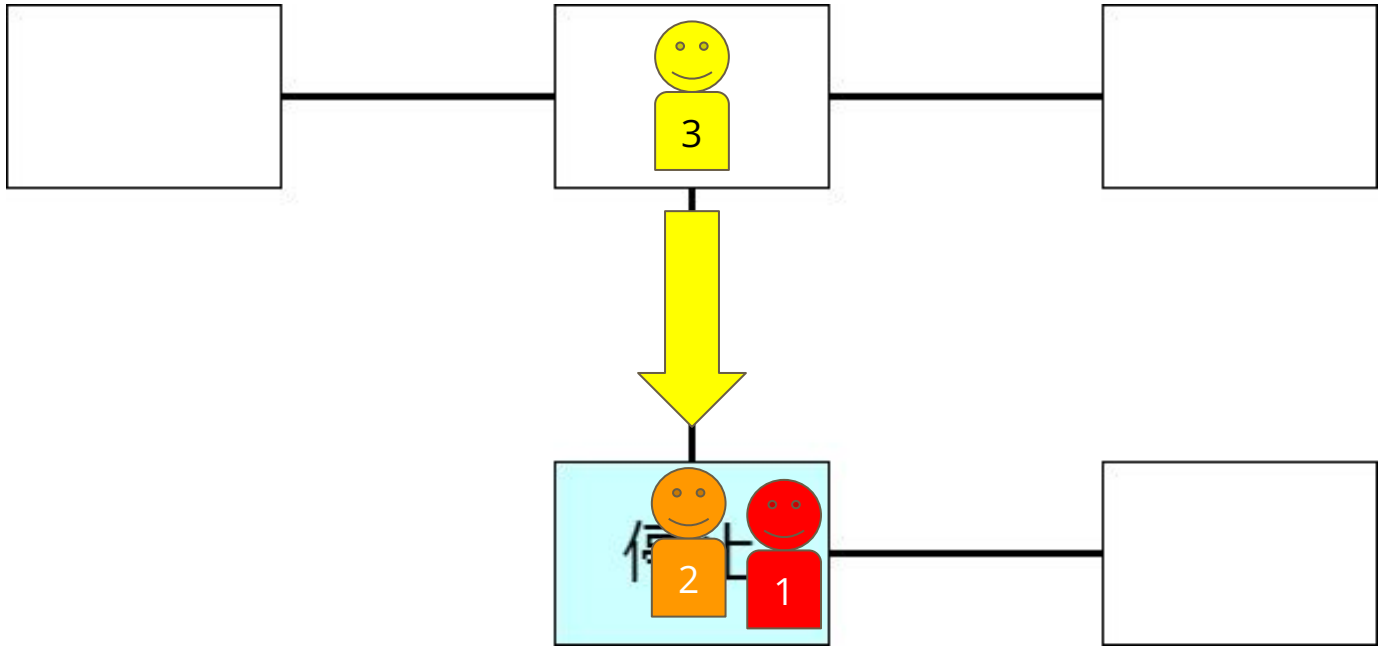
問題概要（動かし方のルール）



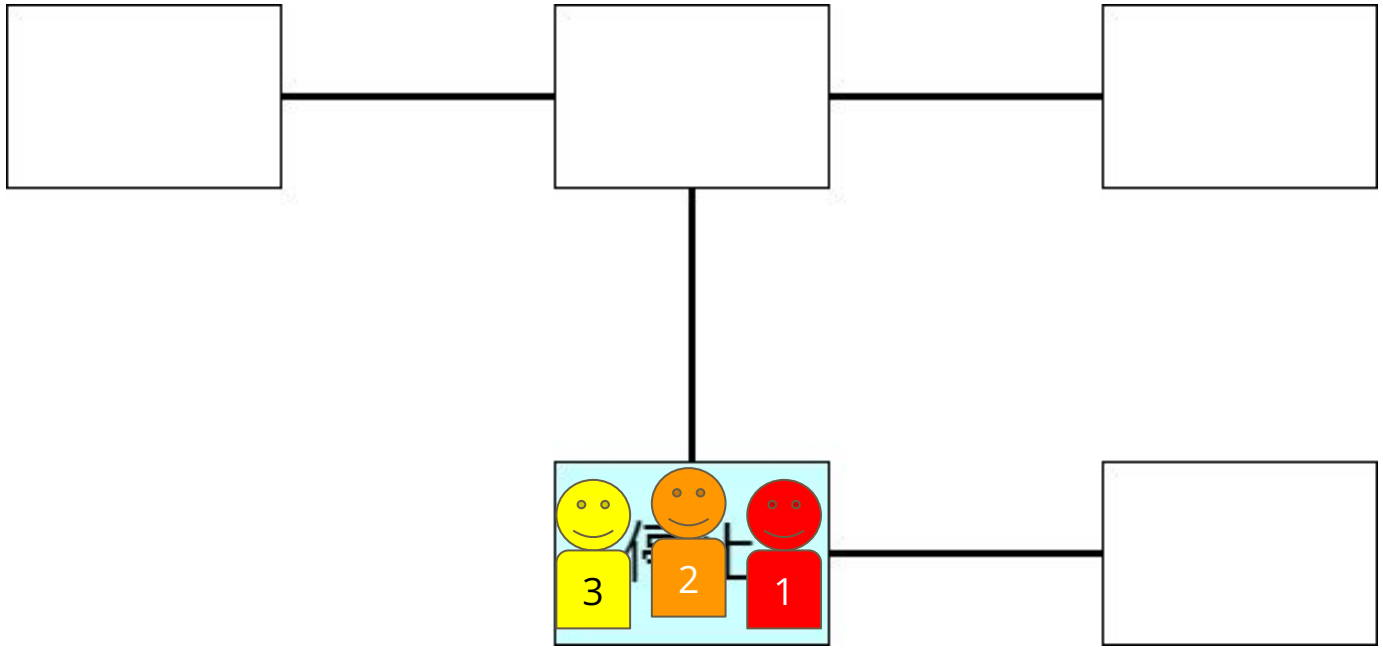
問題概要（動かし方のルール）



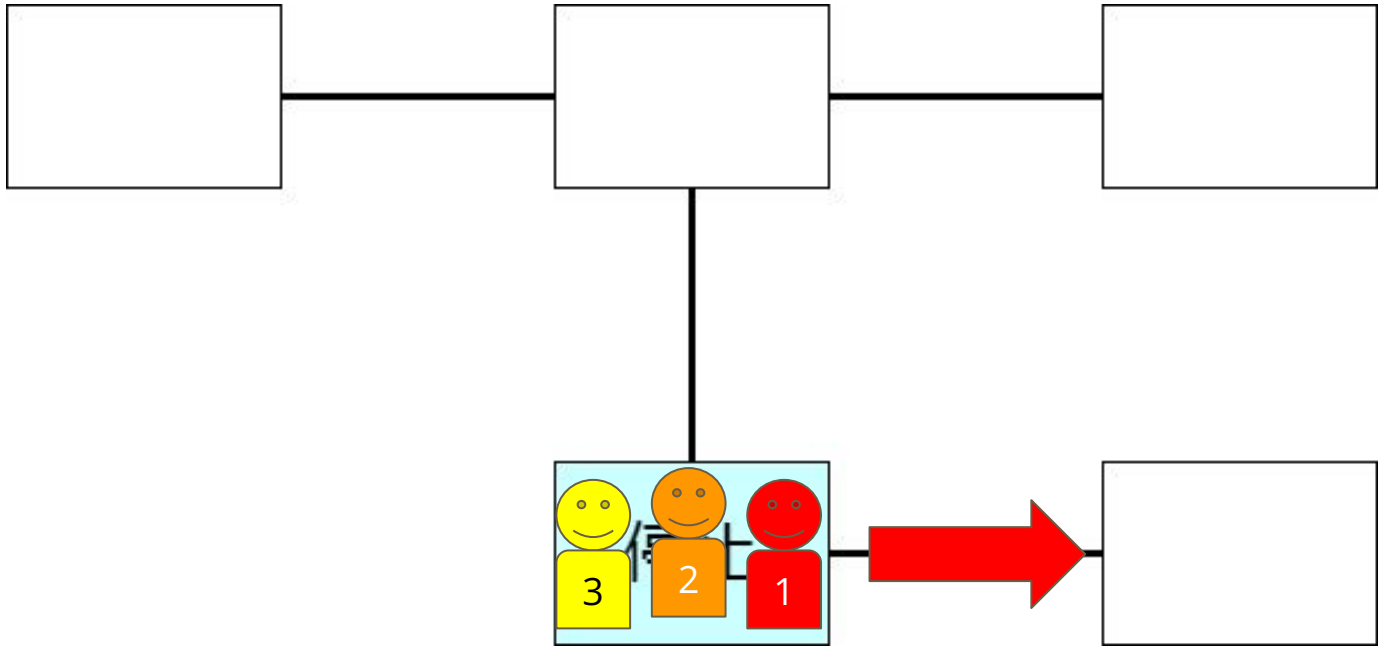
問題概要（動かし方のルール）



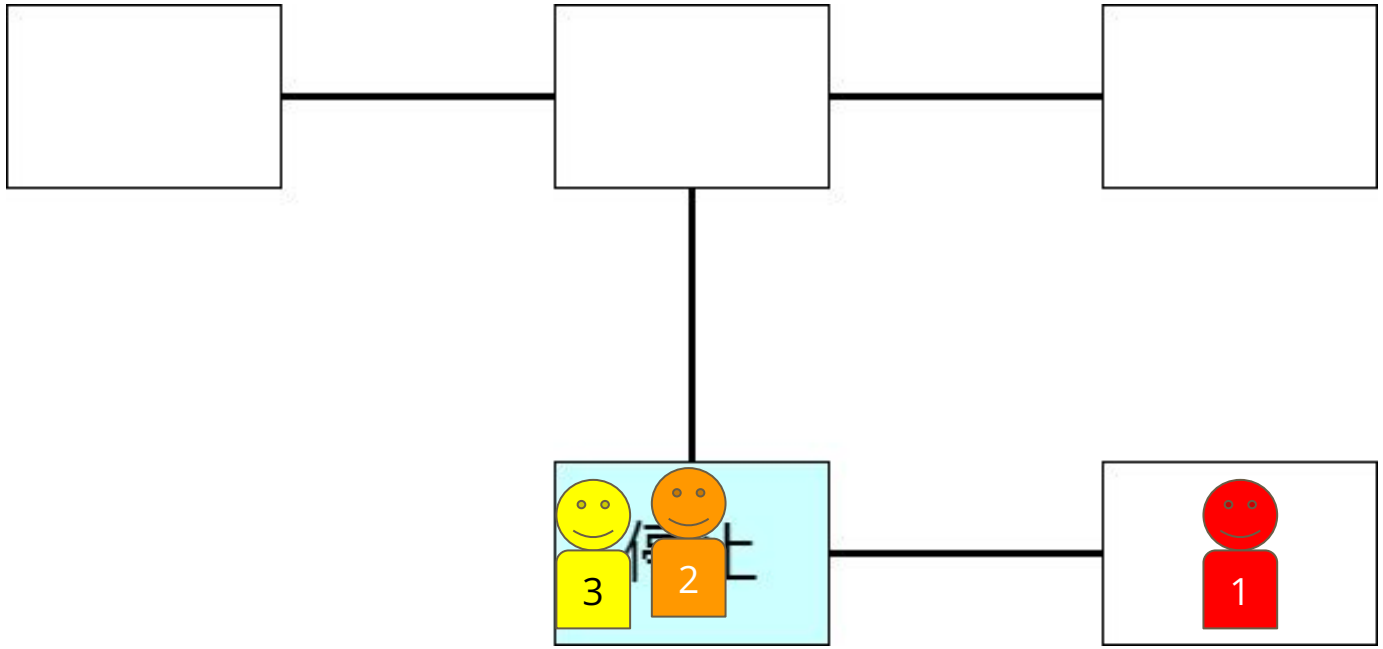
問題概要（動かし方のルール）



問題概要（動かし方のルール）



問題概要（動かし方のルール）



小課題1：停止マスが0個
（JOI 3pts, JOIG 7pts）

小課題1：停止マスが0個

- 単にプレイヤー 1 が動き続ける問題です.
- すべての辺重みが 1 であるような，単一始点最短路問題
- 幅優先探索（BFS）で解ける
- 計算量は O(N+M)

小課題1：停止マスが0個

最短路を求める手法としては次が有名です

- (01) - BFS
- Dijkstra
- Warshall Floyd
- Bellman Ford, SPFA

計算量は？負辺はあってもよい？負閉路は検出可能？

小課題2：停止マスが1個
（JOI 7pts, JOIG 12pts）

小課題2：停止マスが1個

- プレイヤ 1 は次のどちらか
  - マス T まで停止マスを踏まずに進む
  - マス T まで停止マスを 1 回踏んで進む
- 他のプレイヤ
  - 停止マス（唯一）まで進む

小課題2：停止マスが1個

- プレイヤ 1 は次の**どちらか　→　最後に考える**
  - マス T まで停止マスを踏まずに進む　→　**計算可能**
  - マス T まで停止マスを 1 回踏んで進む　→　**計算可能**
- 他のプレイヤ
  - 停止マス（唯一）まで進む　→　**計算可能**

小課題2：停止マスが1個

【解法】

- プレイヤ 1：停止マスを 0 回・1 回踏んで T に行くときの移動回数を計算する

- 他のプレイヤ：停止するまでの移動回数を計算する

小課題2：停止マスが1個

【解法】

- プレイヤ 1：停止マスを 0 回・1 回踏んで T に行くときの移動回数を計算する

→ **2N 状態の最短路問題**

- 他のプレイヤ：停止するまでの移動回数を計算する

→ **停止マスを始点として最短路**

小課題2：停止マスが1個

【実装の注意】

停止マスの扱いに注意しよう

- プレイヤ 1 の場合：到達する瞬間は数えない
- 他のプレイヤの場合：停止マスから距離 0 だとターン終了まで 2 回かかる

小課題3：停止マスが2個
（JOI 7pts, JOIG 12pts）

小課題3：停止マスが2個

- プレイヤ 1 は次のどれか
  - マス T まで停止マスを踏まずに進む
  - マス T まで停止マスを 1 回踏んで進む
  - マス T まで停止マスを 2 回踏んで進む
- 他のプレイヤ
  - 停止マスのどちらかまで進む
  - 2 回目のターンは 1 or 2 手で終わらせる

小課題3：停止マスが2個

- プレイヤ 1 は次の**どれか** → **最後に決める**
  - マス T まで停止マスを踏まずに進む → **計算可能**
  - マス T まで停止マスを 1 回踏んで進む → **計算可能**
  - マス T まで停止マスを 2 回踏んで進む → **計算可能**
- 他のプレイヤ
  - 停止マスのどちらかまで進む → **計算可能**
  - 2 回目のターンは 1 or 2 手で終わらせる → **計算可能**

小課題3：停止マスが2個

答えを求めるには

- プレイヤ 1：停止マスを 0 回・1 回・2回踏んで T に行くときの移動回数を計算する

→ **3N 状態の最短路問題**

- 他のプレイヤ：停止するまでの移動回数を計算する，2 回目のターン終了にかかる時間を計算する

→ **最短路問題 2 回 / 停止マスが隣接しているかチェック**

小課題4 : N, M, K は 3000 以下（JOI 19pts, JOIG 23pts）

小課題4：N, M, K は 3000 以下

これまでの解法の通り，プレイヤ 1 が停止マスを踏む回数 n を固定したときのことを計算することを考えましょう.

- **プレイヤ 1 について計算したいもの**
  - 停止マスを n 回踏んでマス T に到達するときの移動回数
- **他のプレイヤについて計算したいもの**
  - 自分のターンを n 回消化するためにかかる移動回数
- n = 0, 1, ..., N すべてに対してこれが求まれば解ける

小課題4：N, M, K は 3000 以下

- **プレイヤ 1 について計算したいもの**
  - 停止マスを n 回踏んでマス T に到達するときの移動回数

→ これは **N^2 状態の最短路問題**

dist[k][T] := マス T に到達し，到達時点で停止を k 回行っているときの最小移動回数（T での停止は含めない）

小課題4：N, M, K は 3000 以下

- **他のプレイヤについて計算したいもの**
  - 自分のターンを n 回消化するためにかかる移動回数

この部分について，2 つの解法を紹介します．「2 つめ」は小課題 5のときに扱います．

小課題4 : N, M, K は 3000 以下

次の N^2 状態の最短路問題と考えることができます.

- マス v で自分のターンを始めて, k ターン消化を目標としている状態 (v, k)
- v, w が隣接しているとき, (v,k) から (w,k) や (w,k-1) に進める
- (v,k) から (any, 0) への距離は？

小課題4 : N, M, K は 3000 以下

次の N^2 状態の最短路問題と考えることができます.

- マス v で自分のターンを始めて, k ターン消化を目標としている状態 (v, k)
- v, w が隣接しているとき, (v,k) から (w,k) や (w,k-1) に進める
- (v,k) から (any, 0) への距離は？

→ **(any,0) から逆向きに最短路を求めればよい**

小課題1～4まとめ（JOI 36pts, JOIG 54pts）

ここまで解けていれば，問題に登場するすべての要素を正しく理解して計算に組み込めているといってよいでしょう．

考えるべき状態は基本的なものばかりで，計算も BFS だけと基本的ですが，それでもすべての実装を正確にこなすのは結構大変だと思います．

少しだけ解説者なりに，**実装のためのアドバイス**をします．

実装のためのアドバイス

まず細かい計算式を正確に作るためには，変数の意味を正確に定義して，それに沿って考えることが大切です．

dist[k][T] := 停止マス k 回でマス T に居るときの移動回数

dist[k][T] := マス T に到達し，到達時点で停止を k 回行っているときの最小移動回数（T での停止は含めない）

実装のためのアドバイス

自分が定義した dist[k][T] が，マス T で停止マスを踏むタイミングを考慮しているのか等を明確に答えられることを確認してから実装に臨みましょう．ソースコードにコメントを入れておくことも有効だと思います．

```
// dist[k][v]
// - v につくまでにターンエンドを k 回やった
// - v でのターンエンドは含まない
vc<vc<int>> dist(N, vc<int>(N, infty<int>));
```

実装のためのアドバイス

実装ミスを減らすことも大事ですが，実装ミスをしたときのリカバリも超重要です．以下の 2 つを手早く実装できるようにしておくととても便利です．

愚直な解法

- なるべく問題文で言われている通りに実装する
- 計算量は気にしない

ランダムケース生成

- 制約を満たす小さいケースを適当に作る

実装のためのアドバイス

【愚直な解法】例えばコマの場所・次に動かせるコマの番号を状態とした最短路問題を解けばよいです．とにかく全部の状態を考えて最短路問題，というのはかなりよく使います．

```
using T = pair<vc<int>, int>; // コマの場所・次に動かせるコマ
map<T, int> dist;
deque<T> que;
auto upd = [&](vc<int> X, int k, int x) -> void {
  T t = {X, k};
  if (!dist.count(t)) { chmin(ANS[X[0]], x), dist[t] = x, que.push_back(t); }
};
upd(X, 0, 0);
while (!que.empty()){
```

実装のためのアドバイス

【ランダムケース生成】私は Python をよく使います．何らかの言語の乱数の扱いに慣れておきましょう．自分の実装が扱えているつもりのケースで
あれば厳密に問題の制約に
合わせる必要もありません．
画像のコードでは多重辺も
作っています．

```
N = randint(2, 8)
M = randint(N - 1, N + 3)
K = randint(2, 4)
print(N, M, K)

for v in range(2, N + 1):
    print(randint(1, v - 1), v)
for _ in range(M - (N - 1)):
    a = randint(1, N - 1)
    b = randint(a + 1, N)
    print(a, b)

print("".join(random.choice("01")for _ in range(N)))
print(*[randint(1, N) for _ in range(K)])
```

小課題５：K = 2（JOI 24pts, JOIG 26pts）

小課題５以降では小課題４までの解法の計算量を改善することが問われています.

小課題5：K = 2

- 小課題4での解法
- プレイヤ 1 について計算したいもの
  - 停止マスを n 回踏んでマス T に到達するときの移動回数

→ **計算したいものが N^2 通りもあるので無理**

n の範囲を絞り込んだりするのも難しい（n が巨大な解と n が小さな解の比較も明らかではない）

→ プレイヤ 1 のコスト・他プレイヤのコストを個別に求めようとすると厳しい

小課題5：K = 2

【解法の大筋】

- 他のプレイヤの移動回数は，まあまあ簡単な n の関数になる.
- →「**総コスト** = プレイヤ 1 の移動回数 + 他のプレイヤの移動回数」を直接扱える

小課題5：K = 2

【他のプレイヤの移動回数について】

2 回目以降のターンの過ごし方にはどういうものがあるか？

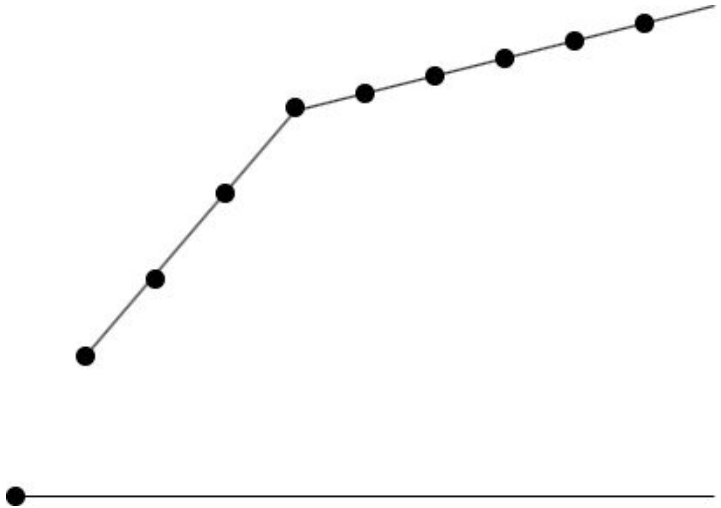
- 1 手かけてターンを終了する
- 2 手かけてターンを終了する
- 将来 1 手かけてターンを終了できるようになることを目指して移動

小課題5：K = 2

【他のプレイヤの移動回数について】

だいたいこんな形の関数になる

- 0 回のときは 0
- 途中まで傾き 2
- 途中から傾き 1



小課題5：K = 2

【他のプレイヤの移動回数について】

- 途中まで傾き 2 → 最短路問題（簡単）
- 途中から傾き 1

→ dp[v] := 頂点 v からスタートしてどこかでコストが f(n)=n+c にできるような c の最小値

→ これも最短路問題みたいに計算できる

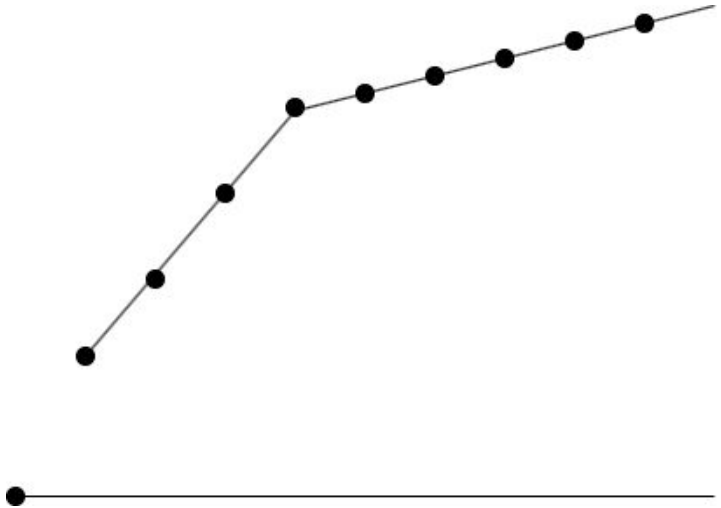
小課題5：K = 2

【他のプレイヤの移動回数について】

左の方と右の方で挙動が違う

→ 結局プレイヤ 1 の停止回数を

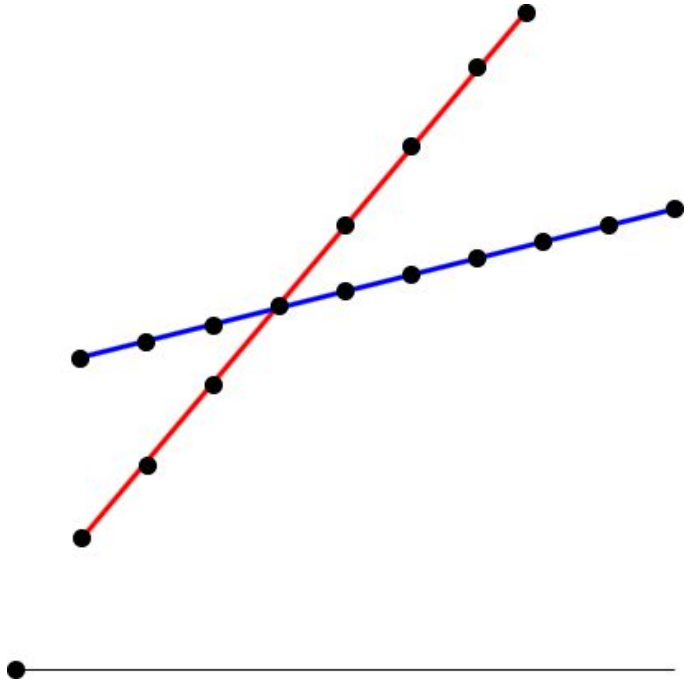
正確に把握する必要がないか？



小課題5 : K = 2

【他のプレイヤの移動回数について】

- 最小移動回数は 2 つの min
- min じゃない方も好きなよう
  に選べることにしてしまって
  解けばよい



小課題5：K = 2

【解法】

- プレイヤ 2 の最小移動回数の関数を表す 2 つの 1 次関数を求める.
- それぞれの 1 次関数を使うと決めてしまって 2 回解く.
- 両プレイヤの移動回数の総和が，2N 状態の最短路問題で解けるようになる.

小課題5：K = 2

【解法】

1 次関数 an + b を使うとき，v から w への移動について，

- 間にプレイヤ 2 のターンがはさまるとき：コスト 1 + a
- プレイヤ 1 のターンが継続するとき：コスト 1

として最短路を計算すればよい．

（S[v] が 1 かつ v が X[0] ではないときコスト a 発生）

小課題5：K = 2

【計算量】

- Dijkstra → O((N+M)\log N)
- コストが 2 種類しかないことを使って O(N+M) 時間で解くこともできる
  - 最後の移動コストが c_1 のものを入れておく que_1
  - 最後の移動コストが c_2 のものを入れておく que_2
  - 2 つの que の先頭のうち小さいものを取り出す

小課題5：K = 2

【実装注意点】

- 停止 0 回とその他は区別する必要があります.
- 初期位置が停止マスの隣であるとき，次のような関数になります.

停止回数	0	1	2	3	4
移動回数	0	1	3	5	7

小課題６：K <= 100
（JOI 9pts, JOIG 7pts）

小課題6 : K <= 100

小課題 5 の解法がそのまま K<=100 に一般化可能です.

プレイヤ 2, 3, 4, ... の移動回数の総和は,

- 各プレイヤは傾き 1, 2 の 1 次関数を持っている.
- 各プレイヤが使う 1 次関数をひとつずつ選んで足す.

として形で考えることできます.

小課題6 : K <= 100

プレイヤ 1 以外の戦略は，傾き K-1, K, ..., 2(K-1) の 1 次関数によって表すことができます．

これらの 1 次関数に対して小課題5と同様にして答を計算できます．

O(K(N+M)) 時間で解けます．

小課題7 : N, M, K <= 30000 (JOI 23pts, JOIG 12pts)
小課題8 : N, M, K <= 50000 (JOI 9pts, JOIG 4pts)

最後の 2 つの小課題は制約が類似しており，必要な考察に大きな違いはありません.

小課題7 : N, M, K <= 30000
小課題8 : N, M, K <= 50000

- 小課題4での解法
- プレイヤ 1 について計算したいもの
  - 停止マスを n 回踏んでマス T に到達するときの移動回数

→ 計算したいものが N^2 通りもあるので無理

n の範囲を絞り込んだりするのも難しい（n が巨大な解と n が小さな解の比較も明らかではない）

小課題7 : N, M, K <= 30000
小課題8 : N, M, K <= 50000

- 小課題4での解法
- プレイヤ 1 について計算したいもの
  - 停止マスを n 回踏んでマス T に到達するときの移動回数

→ 計算したいものが N^2 通りもあるので無理

n の範囲を絞り込んだりするのも難しい（n が巨大な解と n が小さな解の比較も明らかではない）

→ **しかし K が大きければ n の範囲を絞り込める**

小課題7 : N, M, K <= 30000
小課題8 : N, M, K <= 50000

- 停止マス 1 回あたり，最低でもコスト K-1 がかかる.
- 停止マスを n 回余分に使った場合，プレイヤ 1 の移動回数で n(K-1) 回得しないと損失回復できない.
- 停止マスの回数は最小回数から見て +N/(K-1) 以下であるときに限定できる.

O(N^2/K) 状態の最短路問題 → K が大きければ高速.

小課題7 : N, M, K <= 30000
小課題8 : N, M, K <= 50000

【解法まとめ】

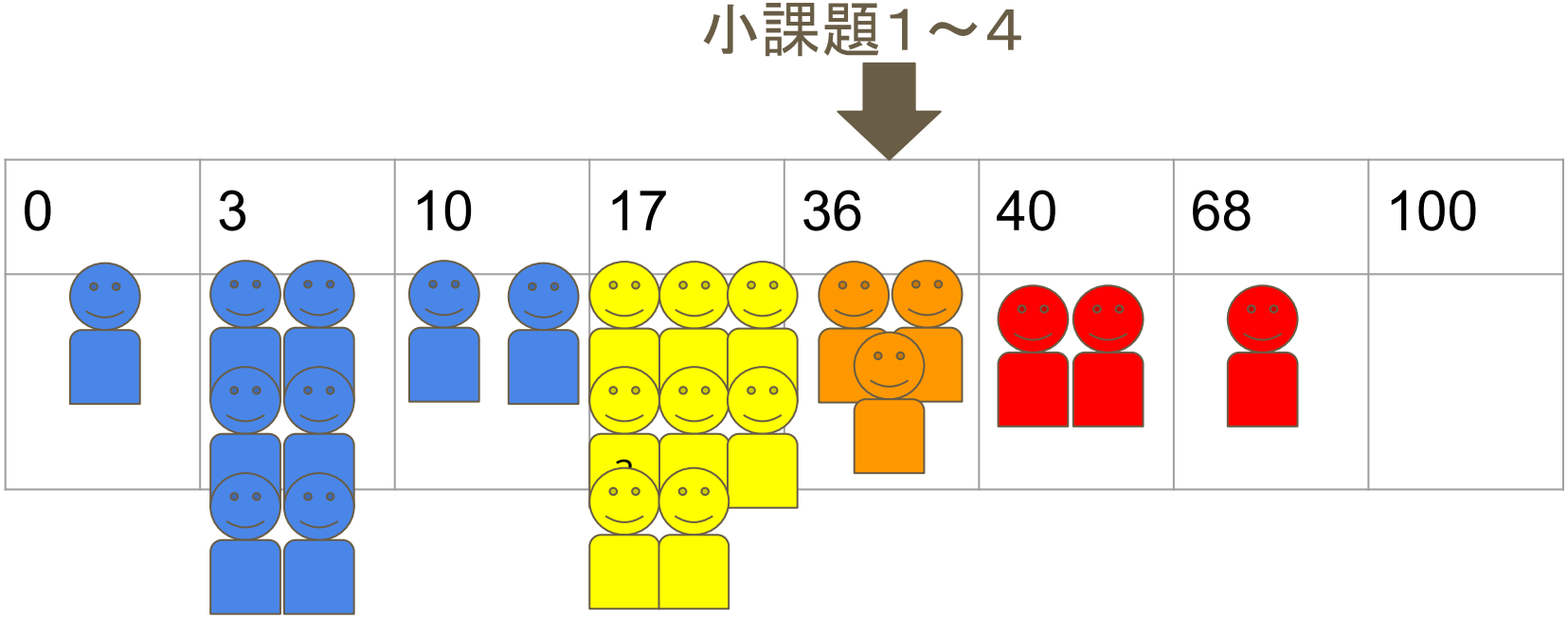
- それぞれの他プレイヤのコスト関数を求める
- それらを足し合わせたコスト関数を求める（累積和などを使う）
- K が小さい（ or コスト関数の傾きの種類数が少ない）場合
  → 小課題6の解法を使う
- K が大きい（or コスト関数の傾きの種類数が多い）
  → 停止マスの回数も状態とした最短路問題
  → 適切に枝狩りをすると O(N^2/K) 状態
- 適切に使い分けると，O((N+M)^{1.5}) 時間で解ける.

小課題7 : N, M, K <= 30000
小課題8 : N, M, K <= 50000

ある程度高速な実装をすれば小課題8まで通ります. TLE してしまった場合には以下のような高速化手法をいくつか試すとよいでしょう.

- 2 種の解法のしきい値を調整
- Dijkstra 法を避けて線形時間の最短路アルゴリズムを使う
- 枝狩りの計算式をより丁寧にやる
- 最後の解法を O(N) メモリで実装する
- ...

得点分布（JOI, 23人）



得点分布（JOIG, 9人）

小課題1～4