# 第 14 回情報オリンピック本選
# 問題 4 – 舞踏会(Ball)

解説：城下慎也(@phidnight)

# 問題概要

- N 人の貴族が一列に並ぶ。
- 既に M 人の貴族については、初期状態で並ぶ位置が決まっている。
- 「先頭の 3 人のうち、踊りのうまさが最大の人と最小の人が抜けて、残った 1 人が列の末尾に移動する」操作が、残り 1 人になるまで行われる。
- 残った 1 人の踊りのうまさとして考えられる最大値を求めよ。

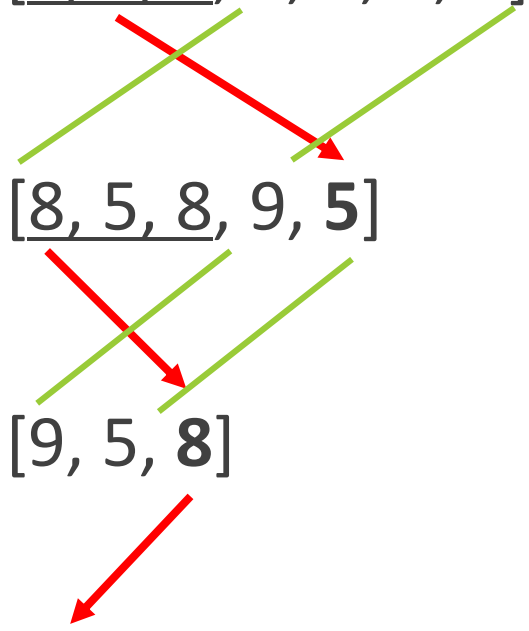
- $3 \leqq N \leqq 99{,}999$

# 例 (順番が決まる前)

- [?, 5, ?, ?, 5, 8, ?] (6, 2, 5, 9 が入る)

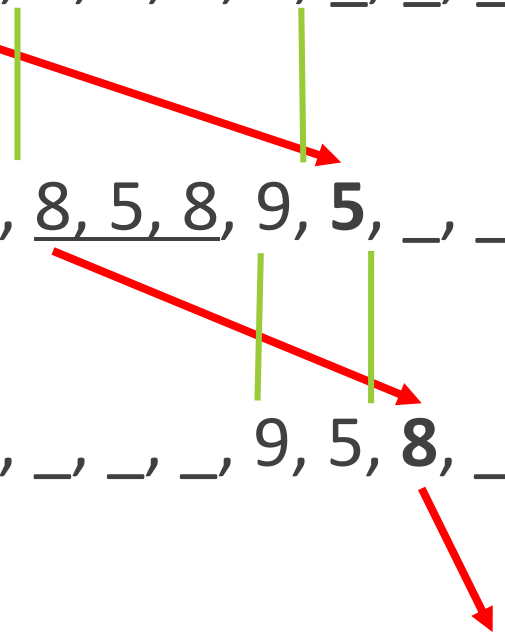
# 例 (順番が決まった後)

- [2, 5, 6, 8, 5, 8, 9]
- [8, 5, 8, 9, **5**]
- [9, 5, **8**]
- [**8**]



# 処理の変更

- [2, 5, 6, 8, 5, 8, 9, _, _, _]
- [_, _, _, 8, 5, 8, 9, **5**, _, _]
- [_, _, _, _, _, _, 9, 5, **8**, _]
- [_, _, _, _, _, _, _, _, _, **8**]



- 列の更新をする際に、毎回配列全体をスライドさせていると毎回 O(N) かかってしまうが、左図のように最初に大きな配列を用意して次の処理をさせるようにすると列の更新についての計算量が O(1) になります。
- このように処理を変更することによって列が決まった後のシミュレーションのコストが O(N) にできます。

# 部分点解法 1

- すべての並び順を試します。
- 考えられる並び順は O(N!) 通りあり、これらをシミュレートして、得られた解の最大値を出力する方針で正解を得ることができます。

- 計算量は O(N!) となり、小課題 1 に正解することができます。

# 方針の転換 1

- どの数字を使って、どの数字を使っていないかを保存すると大変です。
- もしも、登場する数が 2 種類 (high, low) しかなかった場合はどうなるか考えてみます。

# high, low しかないとき

- high, low しかない場合でも、以降の項に持っていく値を定めることができます。

  [high,high,high] → high (high のみなら、次は high)

  [high,high,low] → high (high 2 つ low 1 つなら、次は high)

  [high,low,low] → low (high 1 つ low 2 つなら、次は low)

  [low,low,low] → low (low のみなら、次は low)

- この場合、最初に置く組み合わせは $2^N$ 通りしかないので、すべての組み合わせを試しても計算量は $O(N* 2^N)$ に抑えることができます。
- この性質をうまく利用できないか考えてみます。

# 方針の転換 2

- 実際に登場する数は 2 種類とは限らず、もっと多い場合があります。
- もしも、答えが最大値を求めるという最適化問題ではなく、ある整数 K に対して「K 以上にできますか」という判定問題だった場合を考えてみます。
- こちらの問題の場合、K 以上である数が何であるかの違い (例 : K+1 か K+2 かの違い) は考えなくて良くなります (両者を入れ替えても同じ、K 未満同士についても同様)。
- この場合、K 以上である数を high、K 未満である数を low であるとして考えることで、先ほどの探索を行えば $O(N* 2^N)$ で判定することができます。

# 部分点解法 2

- 答えの候補は高々 N 個しかありません。
- 各候補について、その値を先ほどの K とおいて、high, low の全部の配置を試す方針ならば、全体で $O(N^2 * 2^N)$ で判定できます。
- high を持って行くことのできた K の中で最大のものが答えとなります。

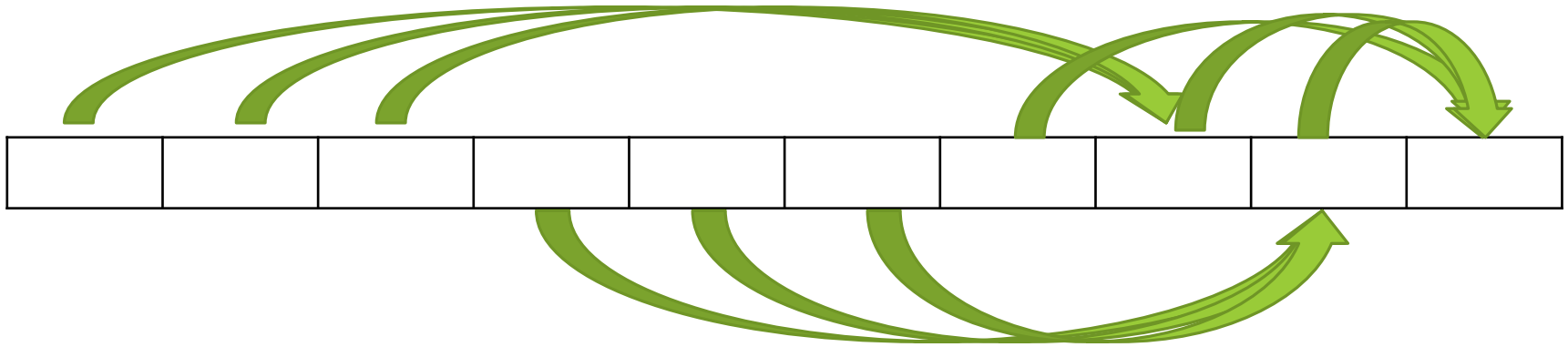
# 多項式にしたい

- high, low に分ける方針でも、すべての並べ方を試す方針だと、指数時間かかってしまいます。

- 多項式の計算量にするために、操作の性質を調べます。

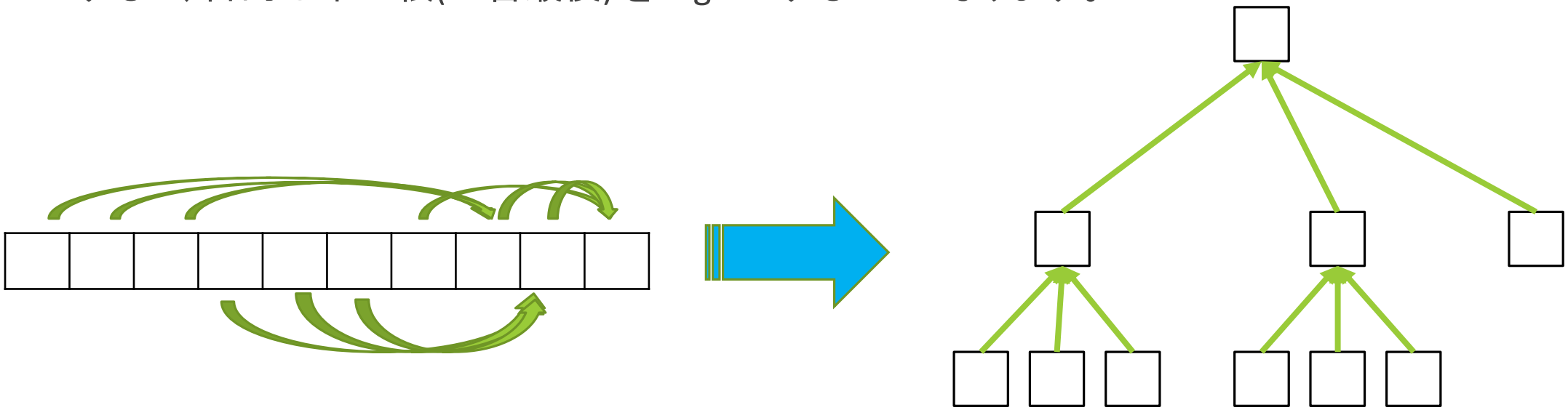
# 考察

- 配列の各要素について、配列のどの場所が関与するのかは、配列に入る値によらず一定です。



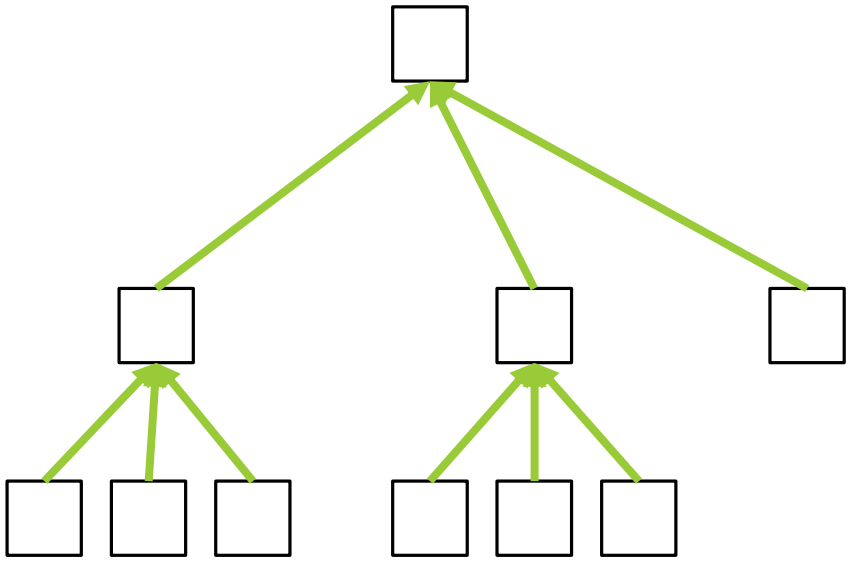
# 考察

- 先ほどの関係性は、以下のように木構造に変換することができます。
- 以降は、この木構造について話を進めることにします。
- すると、目的は木の根(一番最後) を high にすることになります。



# high の節約

- 各ノードについて、そのノードを high にするときに、空いたマスに埋める high の数が少ないほど、他の場所に多く high を回すことができます。
- high は節約した方が良いので、「各ノードについて、そのノードを high にするために必要な high の個数」を計算することを考えます。



# 葉ノードについて

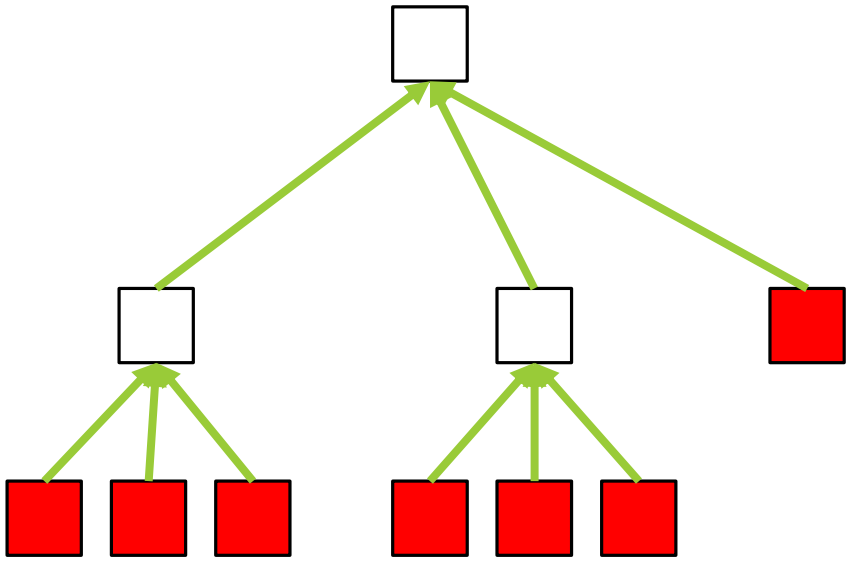
- 葉ノードについて、そのノードを high にしたい場合は、

  埋まっていない→ 1 個必要

  埋まっていて、high→ 0 個必要

  埋まっていて、low→INF (high にできない)

となります。



# 葉ノード以外について

- 葉ノード以外については、そのノードに影響を与える 2 つのノードのうち、2 つが high であれば high になります。一方で high が 1 個以下なら必ず low になります。
- よって、3 つのノードそれぞれについて必要な個数を a 個,b 個,c 個

とおくと、

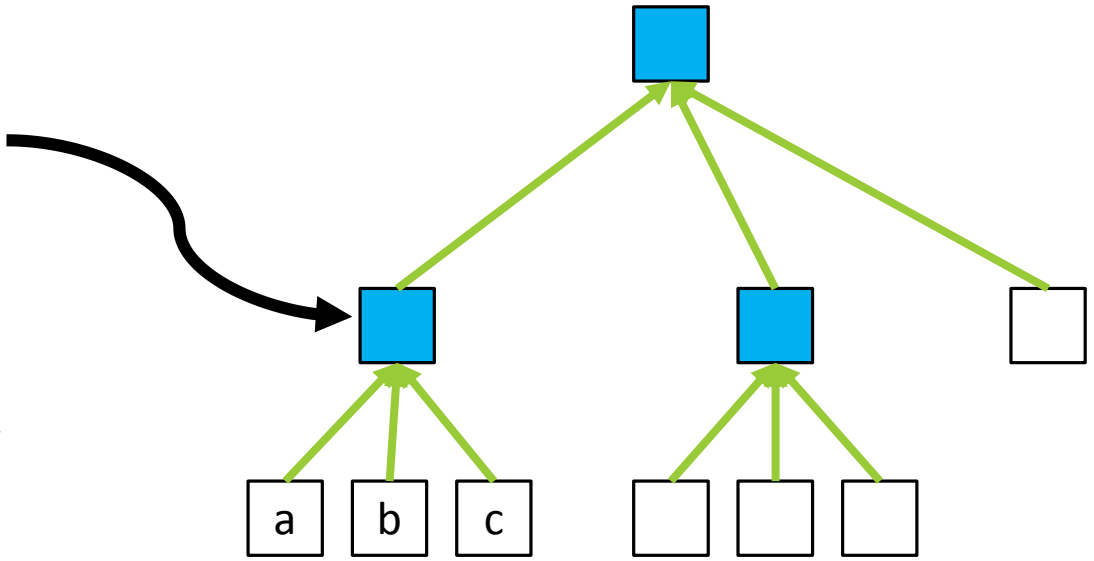
a + b + c – max{a,b,c} (小さい方 2 つ)

となります。

- このようにすることで、動的計画法で、

根ノードを high にするのに必要な high の個数

の最小値がわかります。



# 部分点解法 3

- 先ほどの動的計画法は、O(N) で実行することができます。
- 根ノードの値が、自由における high の個数以下ならば、最初に定めた K 以上にすることができることがわかります。
- すべての K について試すことにして、実現可能だった K の中で最大のものを出力することで正解を得ることができます。

- 計算量は $O(N^2)$ となります。

# 考察

- 例えば、ある K についてダメだったとして、K+1 で実現できるかどうかを考えてみます。
- 初期状態については状況がより悪くなっていて (少なくとも良くなってはいない)、手持ちの high もより少なく (あるいは変化なく) なっています。
- このことから、「ある K についてダメだったら、その K より大きい値はいずれもダメ」とわかります。
- 同様に考えてみると、「ある K について成立したら、その K より小さい値はいずれも成立」とわかります。

- 以上 2 つをまとめると、「ある X が存在して、K が X≧K なら成立し、X＜K なら成立しない」ということができます。この X が求める答えとなります。
- このような X を求める方法とは…?

# 二分探索

- 二分探索は、先ほどの X みたいに、「ある値以上か未満かで評価値が変わる関数の境界値を効率的に求める手法」です。
- 具体的には X の範囲が [a,b] (a 以上 b 以下) にあるとわかっている場合に、c=(a+b)/2 (端数切り捨て)として、

    c ≧ X と分かった→[a,c] について再帰的に試す

    c ＜ X と分かった→[c+1,b] について再帰的に試す

を繰り返して幅を縮める手法です。

※今回、値は全て整数値であると仮定しています。

# 二分探索の例

- X=58 (ここは非公開とします), X が [1,100] に収まっていることがわかっているとします。
- a=1, b=100 より c=50 で c ＜ X なので区間が [51,100] に縮まります。
- a=51, b=100 より c=75 で c ≧ X なので区間が [51,75] に縮まります。
- a=51, b=75 より c=63 で c ≧ X なので区間が [51,63] に縮まります。
- a=51, b=63 より c=57 で c ＜ X なので区間が [58,63] に縮まります。
- a=58, b=63 より c=60 で c ≧ X なので区間が [58,60] に縮まります。
- a=58, b=60 より c=59 で c ≧ X なので区間が [58,59] に縮まります。
- a=58, b=59 より c=58 で c ≧ X なので区間が [58,58] に縮まります。
- こうして X = 58 だとわかります。

# 満点解法

- 今回の場合、解の候補は 1 以上 1,000,000,000 以下であることが制約上わかるので、二分探索が有限回で終了し正解を得ることができます。
- 区間の長さが毎回半分ずつに狭まっているので、反復回数が log(1,000,000,000) 回程度(2 を底とする)になります。
- 各 c について X との比較結果を算出するのに、さっきの動的計画法分の計算量 O(N) がかかるので、全体で O(N*log(値の範囲)) の計算量となります。
- log(値の範囲) という値は 30 くらいと小さく、満点を得ることができます。

- 範囲が大きくても、解の候補が高々 N 個しかないので、O(NlogN) にできます。

# おまけ

# 乱択アルゴリズム

- 「N! の並び替えのうち無作為に 1 つ選んでシミュレーション」を時間のぎりぎりまで繰り返して最大スコアを出力する。
- この手法だと一体どのくらいの得点が望めるのでしょうか?

# 実はそこそこ強い

- 適当に並べ替える操作は、「答えとなる X に対しての high, low の割り当て方のうち無作為に並べ替えたものの 1 つを実験する操作」と等しいので、1 回あたり $1/2^{N}$ の確率で正解を得ることができると見積もることができます。
- 実際はそれよりもはるかに少なく、$1/\ {}_{N}C_{(\frac{N}{2})}$ 以上の確率で正しい割り当てをしていることになります。
- subtask 2 までの範囲ならば有効です。

- このように乱択アルゴリズムが時として有用な場合があるので、いろいろ試してみると、思わぬ発見があるかもしれません。

# 得点分布