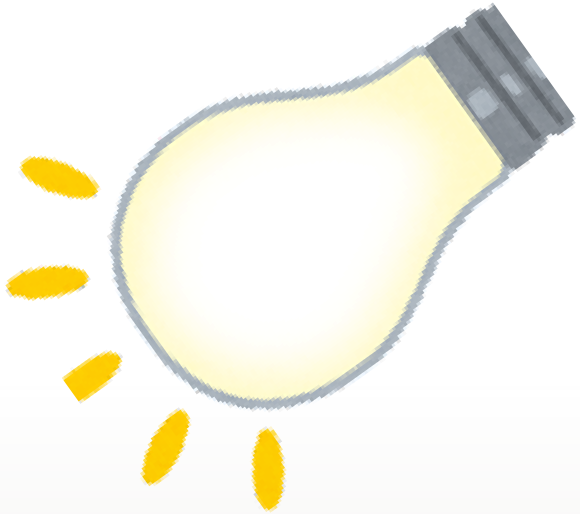


白色光 (White Light)

JOIG 2022/2023 春合宿 競技2-2

解説：大野栞 / 萩原千晴

出典：いらすとや

問題概要

- 赤(R)、緑(G)、青(B)のライトが計N個並んでいる
- １～K個の連続したライトを選び、１回の操作で消せる
- R, G, Bの順に点灯し、Bで終了するようにしたい
- 操作の最小回数を出力

小課題

- **小課題１**　N≦16，K=1　…10点
- **小課題２**　N≦200000，K=1　…26点
- **小課題３**　N≦200000，K≦100　…26点
- **小課題４**　K≦N≦200000　…38点

小課題

◆小課題１	N≦16，K=1	…10点
◆小課題２	N≦200000，K=1	…26点
◆小課題３	N≦200000，K≦100	…26点
◆小課題４	K≦N≦200000	…38点

小課題１

- N≦16　　...O(2^N)でも間に合う

→ **bit 全探索**

- 二進法で、各ライトを消すか消さないか管理
- 全通り調べて、消すライトの最小数を出力

（∵ K= １）

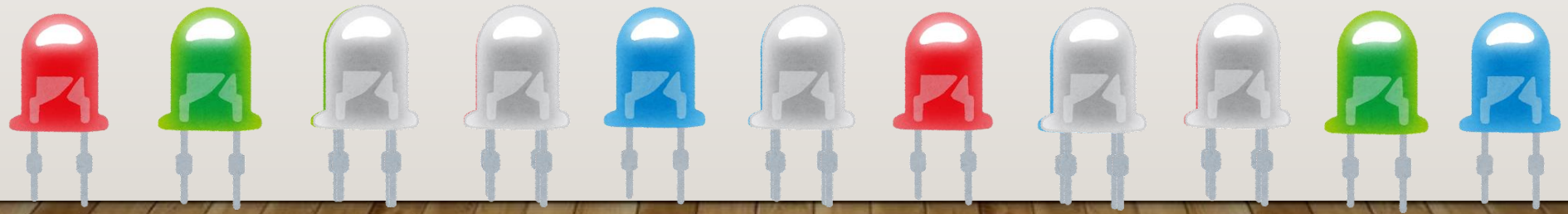
小課題

◆小課題１	N≦16，K=1	…10点
◆小課題２	N≦200000，K=1	…26点
◆小課題３	N≦200000，K≦100	…26点
◆小課題４	K≦N≦200000	…38点

小課題２

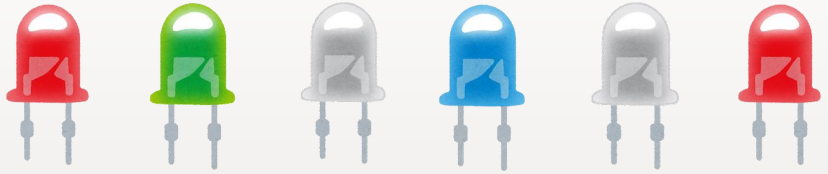
- K=1　→　消すライトの数をなるべく減らすだけ
- 左から順に見ていき、RGBの順になっている限り消さない

→　**貪欲法　…O(N)**



小課題２＜貪欲法＞

- Bで終わらなければいけない



← ダメ

- RGBRGB…Bとなっているなら、点灯しているライトの数は３の倍数

→　(ライトの総数)ー(消したライトの数) を３で割った余り

の分だけ後ろのライトを消灯すれば、 RGBRGB…Bとなる

★出力だけ調整すればOK★

小課題２＜DP編＞

以下

文字列 RGBRGB...R　　：文字列R

文字列 RGBRGB...RG　　：文字列G

文字列 RGBRGB...RGB　　：文字列B

と呼ぶ（長さは任意）

小課題２＜アルゴリズム一例＞

dp[i][0]

… S[0]~S[i-1]の部分列を、文字列Rにするために消す文字数の最小値

dp[i][1]

… S[0]~S[i-1]の部分列を、文字列Gに… 略）

dp[i][2]

… S[0]~S[i-1]の部分列を、文字列Bに… 略）

※ dp[i][j]＝INF
…不可能を示唆

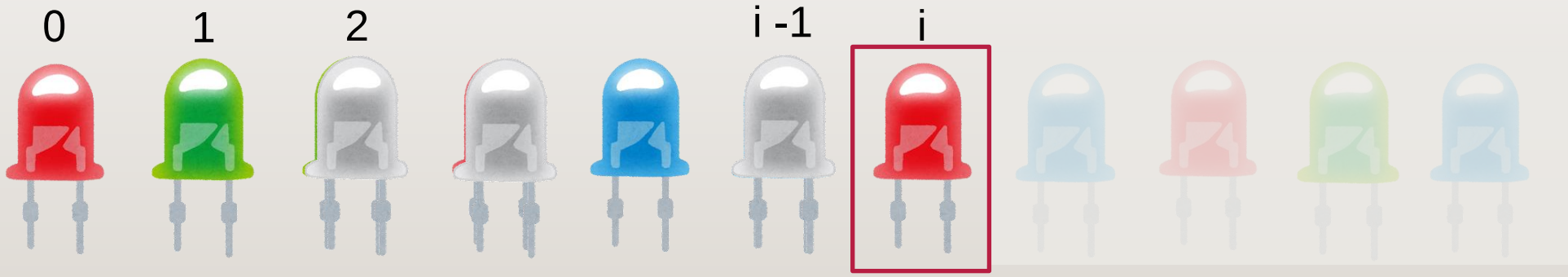
※ dp[0][2]＝0
(∵一つも消灯しないのもOK)

小課題２＜アルゴリズム一例＞

例えば　S[i]がRの時

① S[0]~S[i-1]の部分文字列Bに、S[i]を加えて文字列Rにする

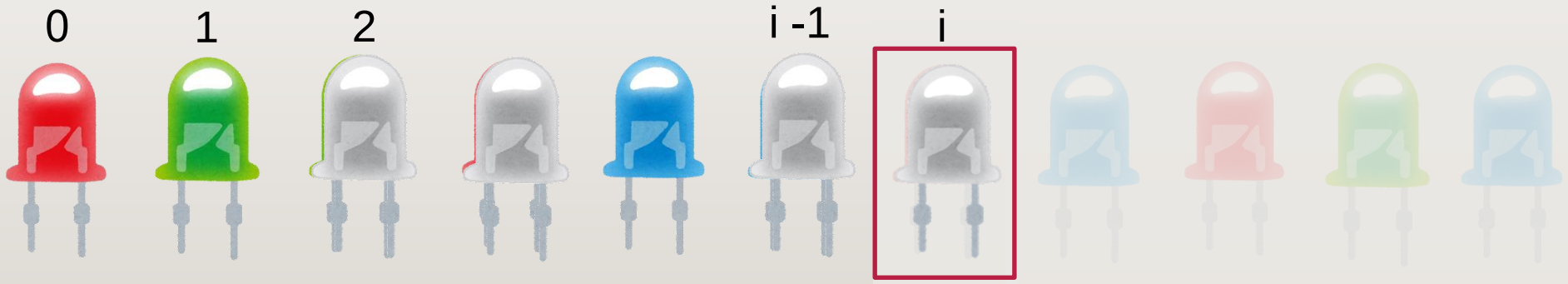
➡ dp[i+1][0] = min{ dp[i+1][0] , dp[i][2] }



小課題２＜アルゴリズム一例＞

② S[i]は消灯して、S[0]~S[i-1]の部分文字列Bをそのまま使う
(文字列G,Rでもやる)

→ dp[i+1][2] = min{ dp[i+1][2] , dp[i][2] +1 }



小課題

◆小課題１	N≦16，K=1	…10点
◆小課題２	N≦200000，K=1	…26点
◆小課題３	**N≦200000，K≦100**	**…26点**
◆小課題４	K≦N≦200000	…38点

小課題３

O(N * K)で間に合いそう！

→ K文字前までさかのぼりながらDPできる

☆小課題２のDPをそのまま使える☆

①はそのままで…

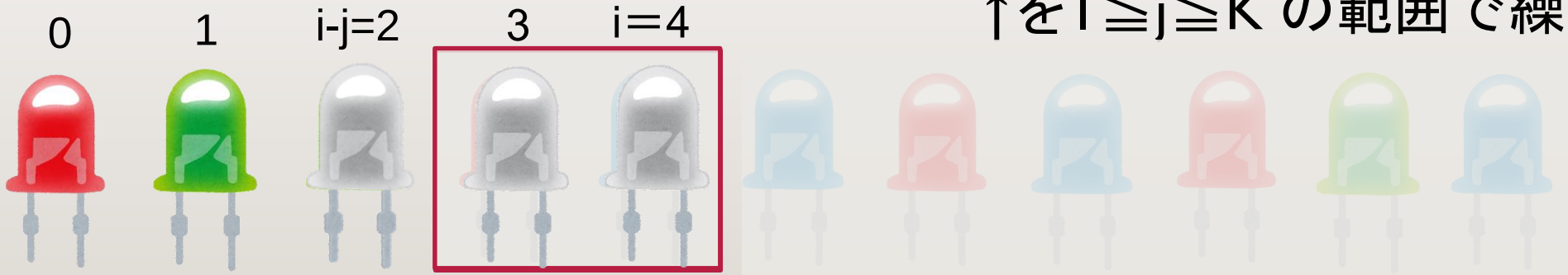
小課題３

j 文字分さかのぼる

② S[i-j+1]~S[i]は消灯して、S[0]~S[i-j]の部分文字列Bをそのまま使う
(文字列B,Rでもやる)

➡ dp[i+1][2] = min{ dp[i+1][2] , dp[i-j+1][2] +1 }

↑を1≦j≦K の範囲で繰り返す



小課題

◆小課題１	N≦16，K=1	…10点
◆小課題２	N≦200000，K=1	…26点
◆小課題３	N≦200000，K≦100	…26点
◆小課題４	K≦N≦200000	…38点

小課題４

O(N * K)はTLE

→ K回の for 文をなくしたい

for(j=1; j≦K; j++) dp[i+1][2] = min{ dp[i+1][2] , **dp[i-j+1][2] +1** }

は結局、

dp[i+1][2] = min{ dp[i+1][2] , **min{ dp[i-j+1][2] | 1≦j≦K }** +1 }

小課題４

スライド最小値
という有名問題

min{ dp[i-j+1][2] | 1≦j≦K }　を**O(1)** / O(logN)で得たい！

★　deque（que[0]~que[2]）で記憶しておく

i からK以上離れた位置のデータは使えない　＝位置のデータが必要

➡ 更新したdp[i][j]は、{ i , dp値} のペアにしてque[j]に後ろから追加

i1< i2 , dp[i1] ≧ dp[i2] となったら、 dp[i1] のデータは捨てる

➡ queは、i が昇順、dp値が降順になる　➡ que[j]の最後尾が、求める最小値

小課題４（別解）

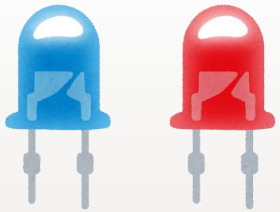
min{ dp[i-j+1][1] | 1≦j≦K }　をO(1) / **O(logN)**で得たい！

区間の最小値を求める　⇒　**セグメント木！**

（O(logN)でデータを更新、O(logN)で任意の区間の最小値を求める）

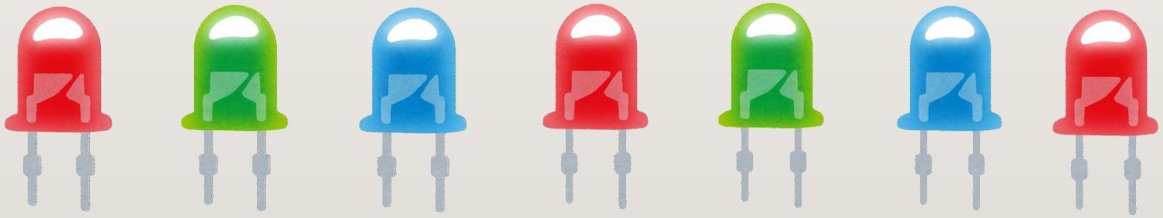
得点分布

100 点



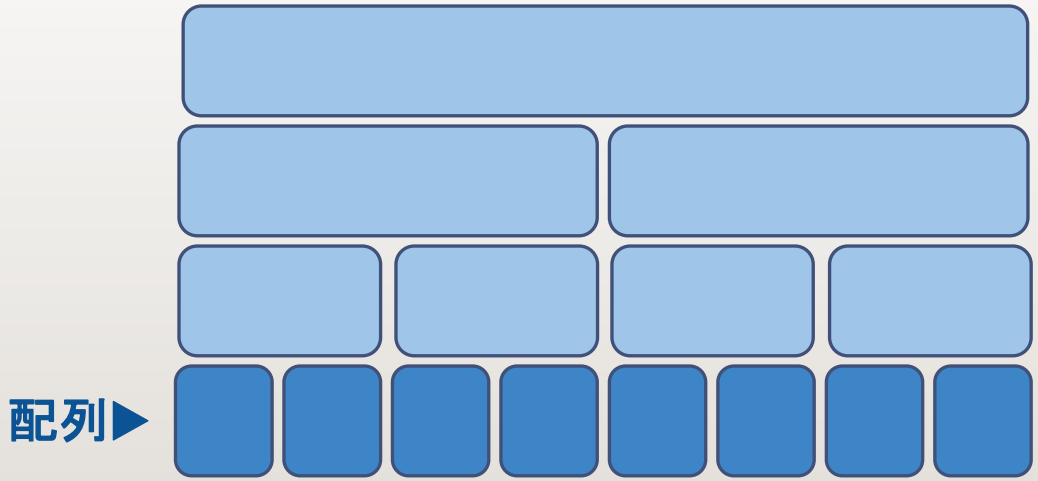
62 点

36 点



おまけ：セグメント木って何？？

→ 配列を高速に処理できる便利なデータ構造！

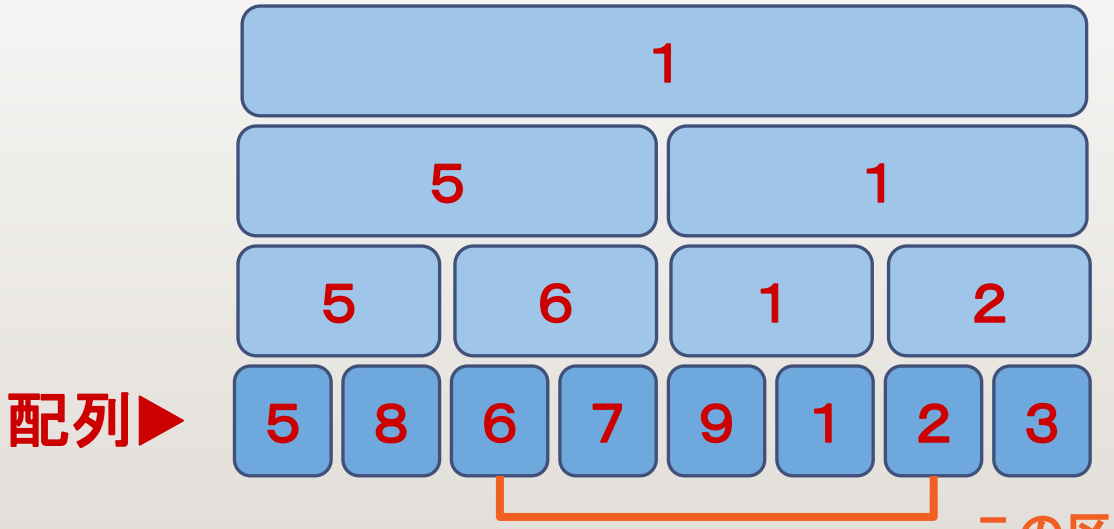


左のように
半々ずつを管理していく木を作る

**ある区間での最小値を求めたり
要素の総和を求めたりすること
が
O(log N) の計算量でできる！**

おまけ：セグメント木って何？？

具体的にやってみよう！

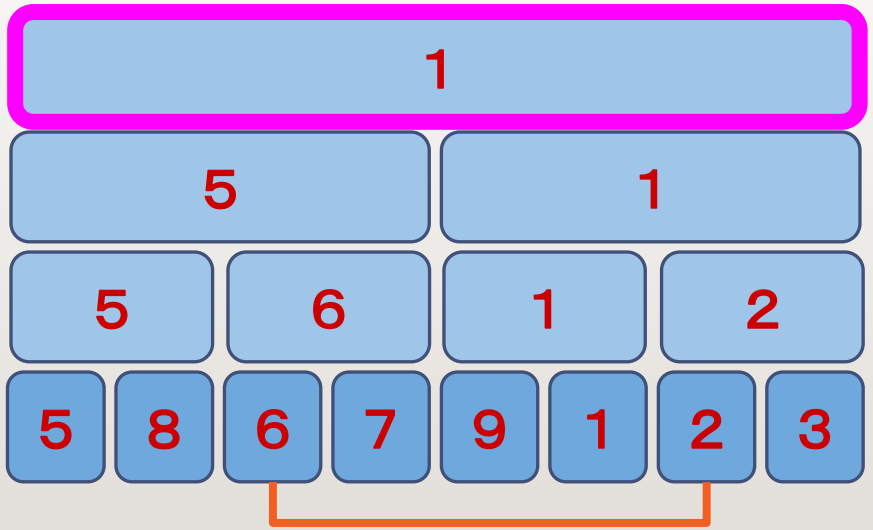


区間最小値を求める場合はこんな感じ

区間最小値クエリ：
RMQ（Range Minimum Query）

おまけ：セグメント木って何？？

具体的にやってみよう！

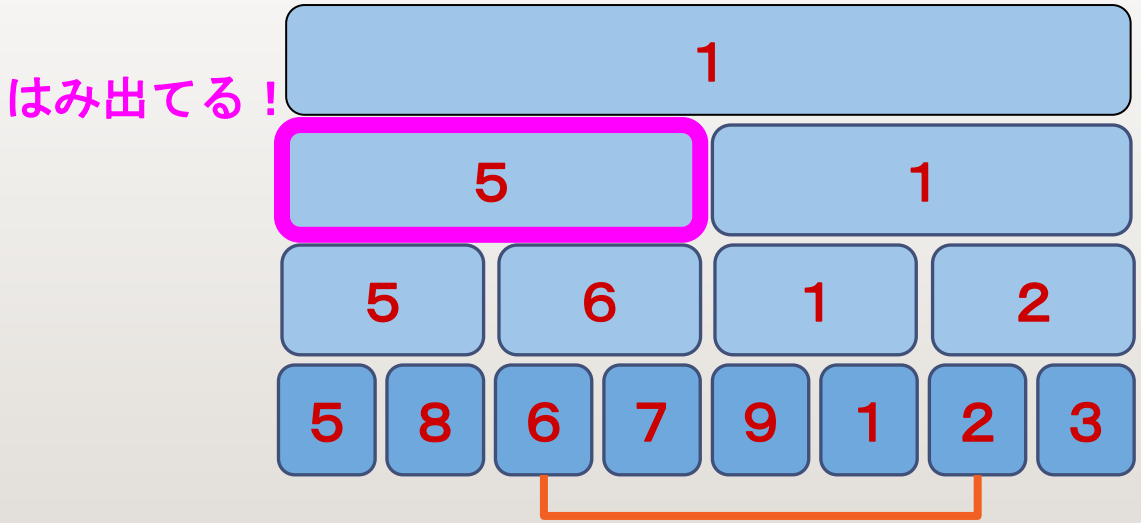


はみ出てる！

根から見てゆき、
各頂点が管理する区間と
今考えている区間を見比べる

おまけ：セグメント木って何？？

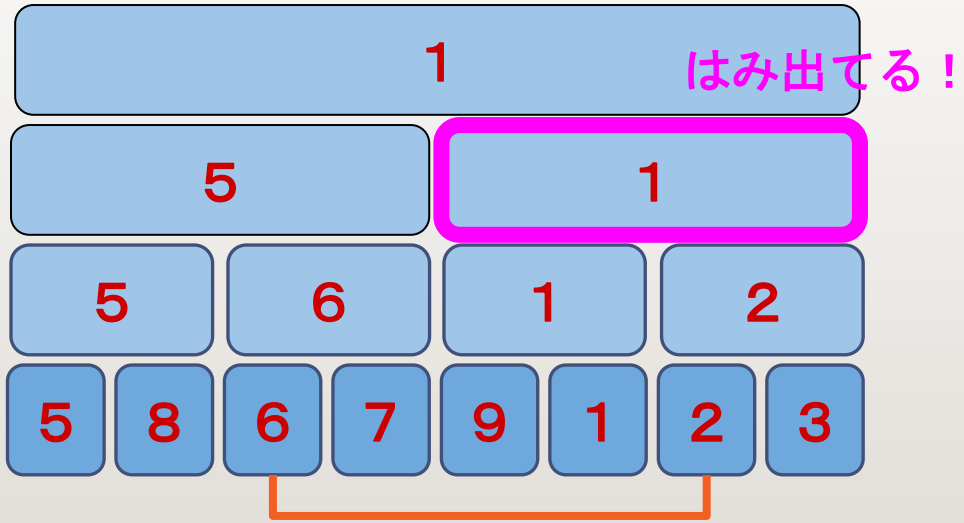
具体的にやってみよう！



根から見てゆき、
各頂点が管理する区間と
今考えている区間を見比べる

おまけ：セグメント木って何？？

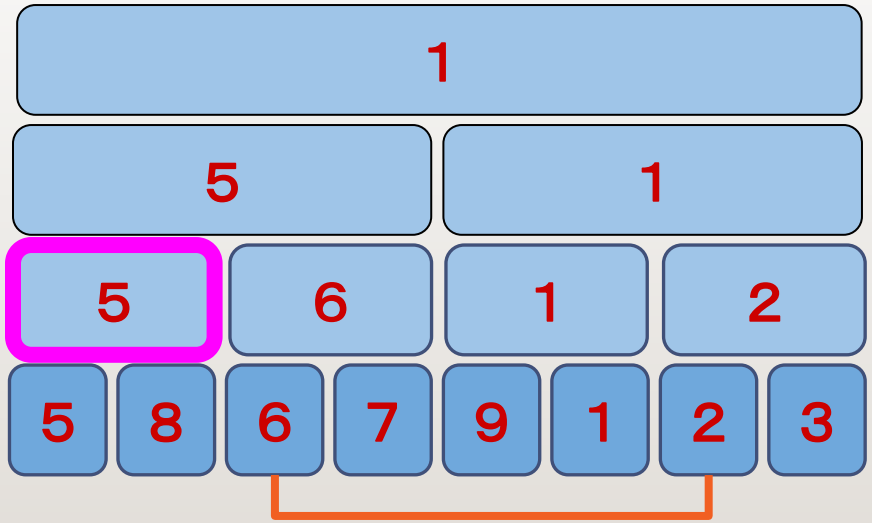
具体的にやってみよう！



根から見てゆき、
各頂点が管理する区間と
今考えている区間を見比べる

おまけ：セグメント木って何？？

具体的にやってみよう！

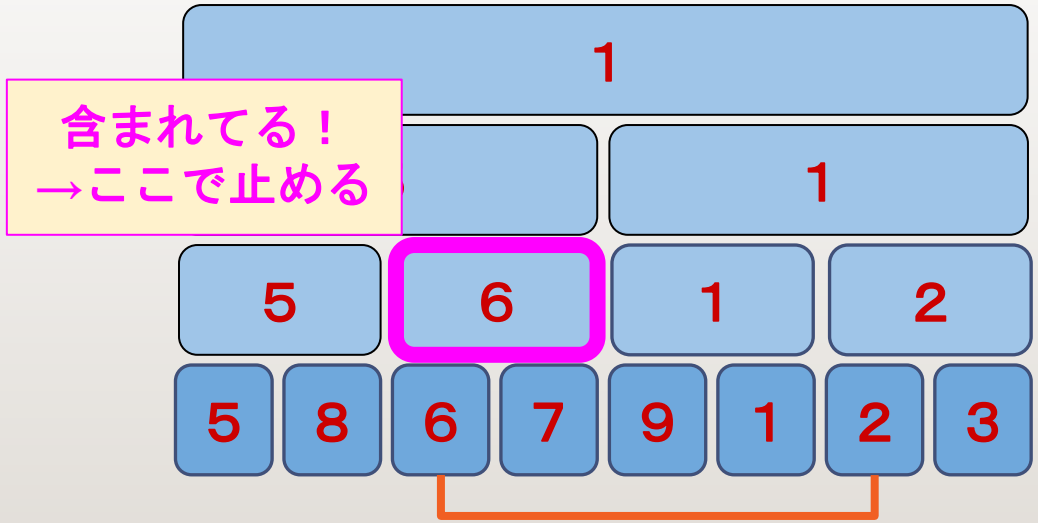


入ってない！
→打ち切り

根から見てゆき、
各頂点が管理する区間と
今考えている区間を見比べる

おまけ：セグメント木って何？？

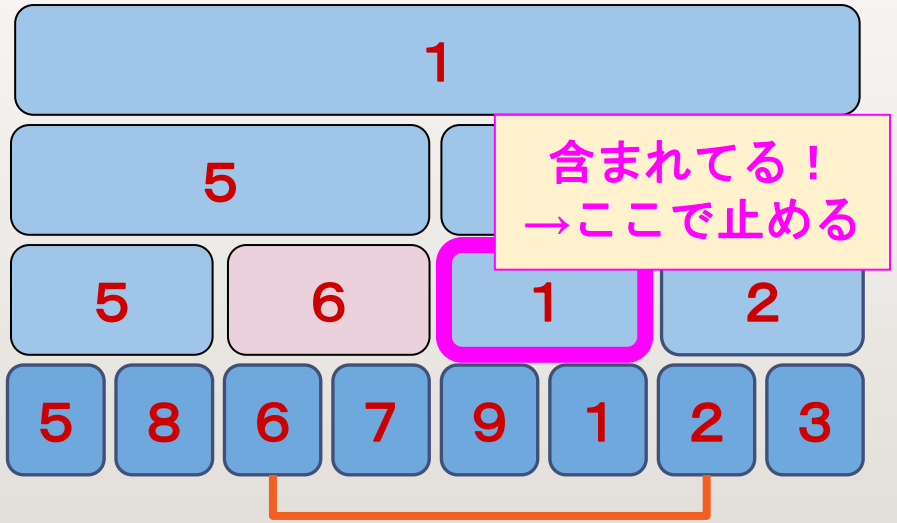
具体的にやってみよう！



根から見てゆき、
各頂点が管理する区間と
今考えている区間を見比べる

おまけ：セグメント木って何？？

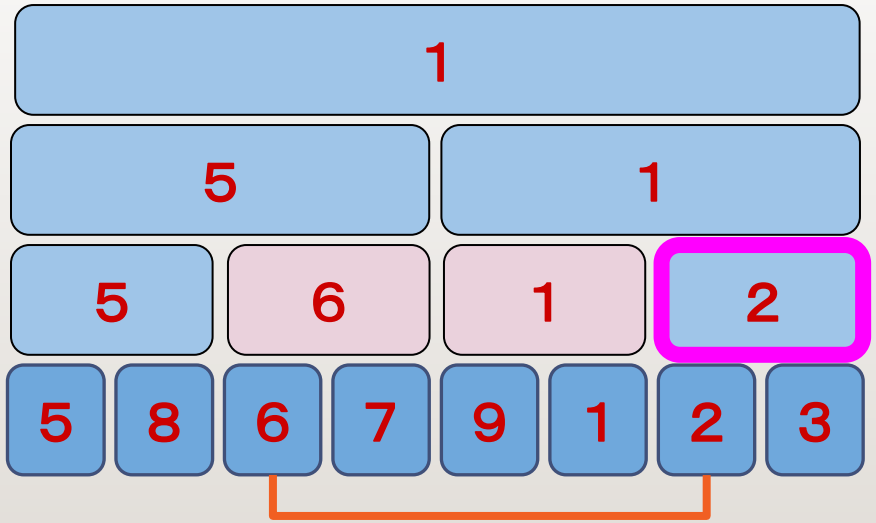
具体的にやってみよう！



根から見てゆき、
各頂点が管理する区間と
今考えている区間を見比べる

おまけ：セグメント木って何？？

具体的にやってみよう！

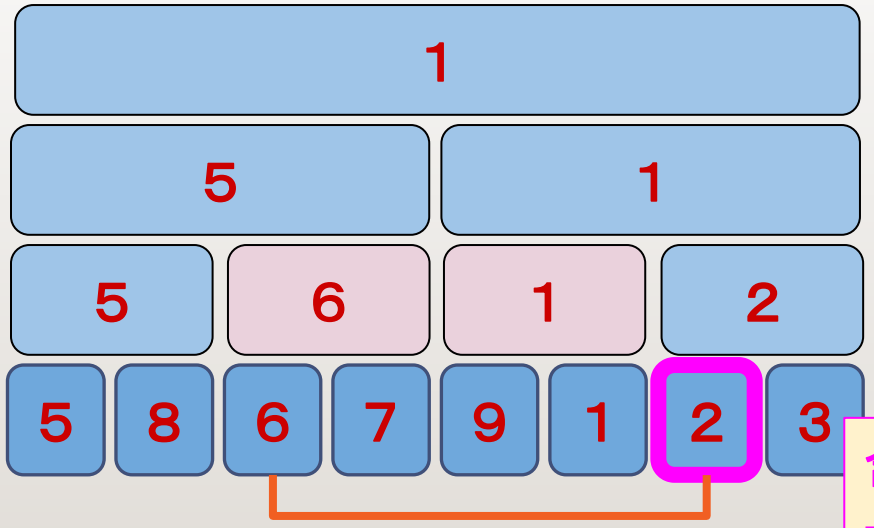


はみ出てる！

根から見てゆき、
各頂点が管理する区間と
今考えている区間を見比べる

おまけ：セグメント木って何？？

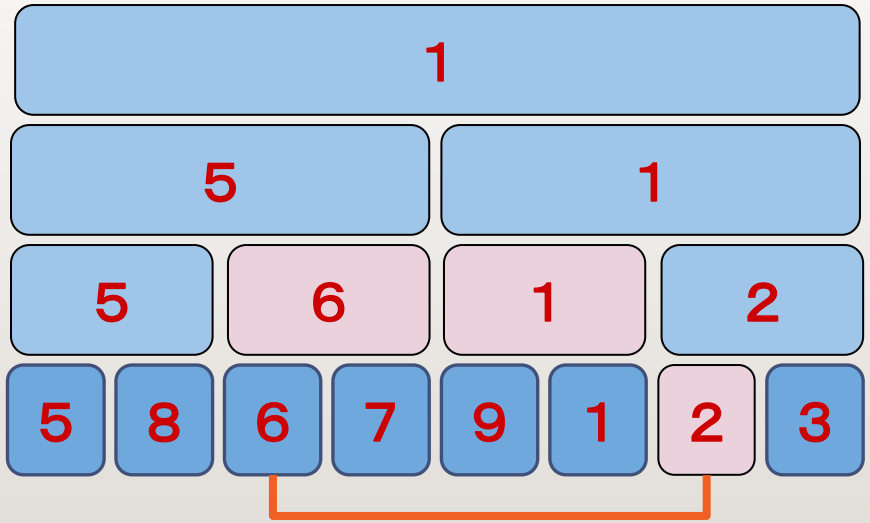
具体的にやってみよう！



根から見てゆき、
各頂点が管理する区間と
今考えている区間を見比べる

おまけ：セグメント木って何？？

具体的にやってみよう！



この頂点を見ればOK！

おまけ：セグメント木って何？？

具体的にやってみよう！

すごく大きなデータでも
細かく見ずに済む！

おまけ：セグメント木って何？？

RMQの実装

値をI箇所更新する時
→ 葉を更新し、親を辿りつつ更新していく

区間最小値を求める時
→ 根から辿り再帰的に見ていく

おまけ：セグメント木って何？？

RM

値

一

区

→ 根から辿り再帰的に見ていく

さらに先の実装については割愛します
ぜひ調べてみましょう！