ICPC輪講第二回の残り

理学部情報科学科4年 小島晃司

今回の輪講内容

- 教科書『算法芸術与信息学競賽』(劉汝佳, 黄亮)の1.4節(pp.79~112)の前半
  - Disjoint set
  - Heap
- 教科書の例題解説が中心

例題7-The race(1/5)

- 同じ方向に移動するn台の宇宙船があり(nは高々250,000)、各々の宇宙船に対して、その位置xと速度vが与えられている(各xは高々1,000,000、各vは高々100)。このとき
  - 宇宙船同士がすれ違う総数
  - 始めm回分の、すれ違った各2台の宇宙船の番号(mは高々10,000)。但し、同じ時間ですれ違った場合はすれ違った位置xの小さい方を先に出力し、同じ場所で3台以上の宇宙船が同時にすれ違うことはないと仮定する。

を求めよ。

(CEOI 2003)

例題7-The race(2/5)

- 解法
  - すれ違った総数の求め方
    - $\#\{(i,j) | x_i < x_j, v_i > v_j\}$を求めれば良い
      - 単純にやると$O(n^2)$で無理
    - Merge sortをうまく利用することでO(nlogn)でとくことができる
      - xに関してソートされた(x,v)のリストをvに関してマージソートする
      - 2つのsort済みリストL_1,L_2をMergeする際L_1からある要素(x,v)が取り出されたとすると、そのときにL_2に残っている要素の総数が $\#\{j | (x_j, v_j) \in L_2, x < x_j, v > v_j\}$ となる。
        - L_1,L_2先頭要素が等しいときはL_2の方の要素を先に取り出すことに注意

例題7-The race(3/5)

- あとは以下の式に従えばソート終了とともにすれ違いの総数が計算できる
  - $\#\{(i,j)|x_i<x_j, v_i>v_j\}=$

    $\#\{(i,j)|(x_i,v_i)\in L_1, (x_j,v_j)\in L_1, x_i<x_j, v_i>v_j\}+$

    $\#\{(i,j)|(x_i,v_i)\in L_2, (x_j,v_j)\in L_2, x_i<x_j, v_i>v_j\}+$

    $\#\{(i,j)|(x_i,v_i)\in L_1, (x_j,v_j)\in L_2, x_i<x_j, v_i>v_j\}$

  - $\#\{(i,j)|(x_i,v_i)\in L_1, (x_j,v_j)\in L_2, x_i<x_j, v_i>v_j\}=$

    $$\sum_{(x,v)\in L_1} \#\{j|(x_j,v_j)\in L_2, x<x_j, v>v_j\}$$

例題7-The race(4/5)

- vの範囲が狭い(v < 100)ことを利用する手法
  - ビンソートを利用
    - num[0],num[1], ... ,num[99]を用意しておく
  - xに関して降順にソートされた各(x,v)を先頭から順にビンに放りこんでいく際、以下の関係式が各段階で成立する

$$\sum_{k=0}^{v-1} num[k] = \#\{ j | x < x_j, v > v_j \}$$

  - あとは以下の式に従えば計算することができる

$$\#\{(i, j) | x_i < x_j, v_i > v_j\} = \sum_{(x,v)} \#\{ j | x < x_j, v > v_j \}$$

  - 計算量はO(nv)
    - 区分木を利用すればO((n+v)logv)で計算可能

例題7-The race(5/5)

- 具体的なすれ違い方の求め方
  - ヒープを利用すれば良い
    - 各宇宙船に対して、「すぐ前にいる宇宙船に追い付くまでの時間」を計算したものをヒープに入れておく
    - 以下の処理をm回繰り返す
      - ヒープから最小値を取り出す
      - 最小値を達成するすれ違いi,jに対して、以下の変更を施し、ヒープの更新を行う
        - iのすぐ前にいる宇宙船を、jのすぐ前にいた宇宙船に変更
        - jのすぐ前にいる宇宙船をiに変更
        - iのすぐ後ろにいた宇宙船のすぐ前にいる宇宙船をjに変更
    - 計算量はO(n+mlogn)

例題8-Eat or Not to Eat(1/5)

- 周期的に出るミルクの量が変化する牛たちがいる(周期は1~10日で、各牛たちによって異なる。また、牛たちの数nは高々100,000で、一日に出るミルクの量は高々200)。このとき、以下の規則に従って牛たちを間引いていったとき、最終的に生き残る牛たちと、最後に牛が間引かれるまでに掛かった日数を求めよ。
  - その日に一番ミルクが出なかった牛を一頭間引く
  - 最もミルクが出なかった牛が複数頭いた場合はどの牛も間引かれることはない

(OIBH Reminiscene Programing Contest)

例題8-Eat or Not to Eat(2/5)

- 単純にシミュレートしようとすると、1日シミュレートするのにO(n)かかる
  - 周期が1~10日の牛たちが全て揃っていて全頭生き残る場合、過去と同じ状態に戻るまでに最低でも1~10の最小公倍数である2520日かかるので、時間的に無理
  - → 周期が短く、バリエーションが少ない(高々10)であることを利用する
    - 周期が同じ牛たちを纏めて取り扱う

例題8-Eat or Not to Eat(3/5)

- 解法
  - 始めに周期によって牛たちを分類し、併合する
    - 併合することで得られるグループの総数kは高々10
    - 周期tに属するグループはt個のヒープによって構成されている
      - 各位相での牛たちの出すミルクの量を格納
  - あとはこのデータを利用、保守しつつシミュレーションを行っていけば良い
    - 併合した各グループから、最低値を達成する牛に関する部分情報が得られるので、この情報から全体でどの牛が間引かれるのかが計算できる
      - → 一日のシミュレーションにかかる時間がO(k)
    - 牛が間引かれたときに必要となるデータの修正がt回のヒープ要素削除で実現できるので高速(O(klogn))
  - 全体の計算量はO(Tk+nklogn)
    - Tはシミュレーションに掛かった日数

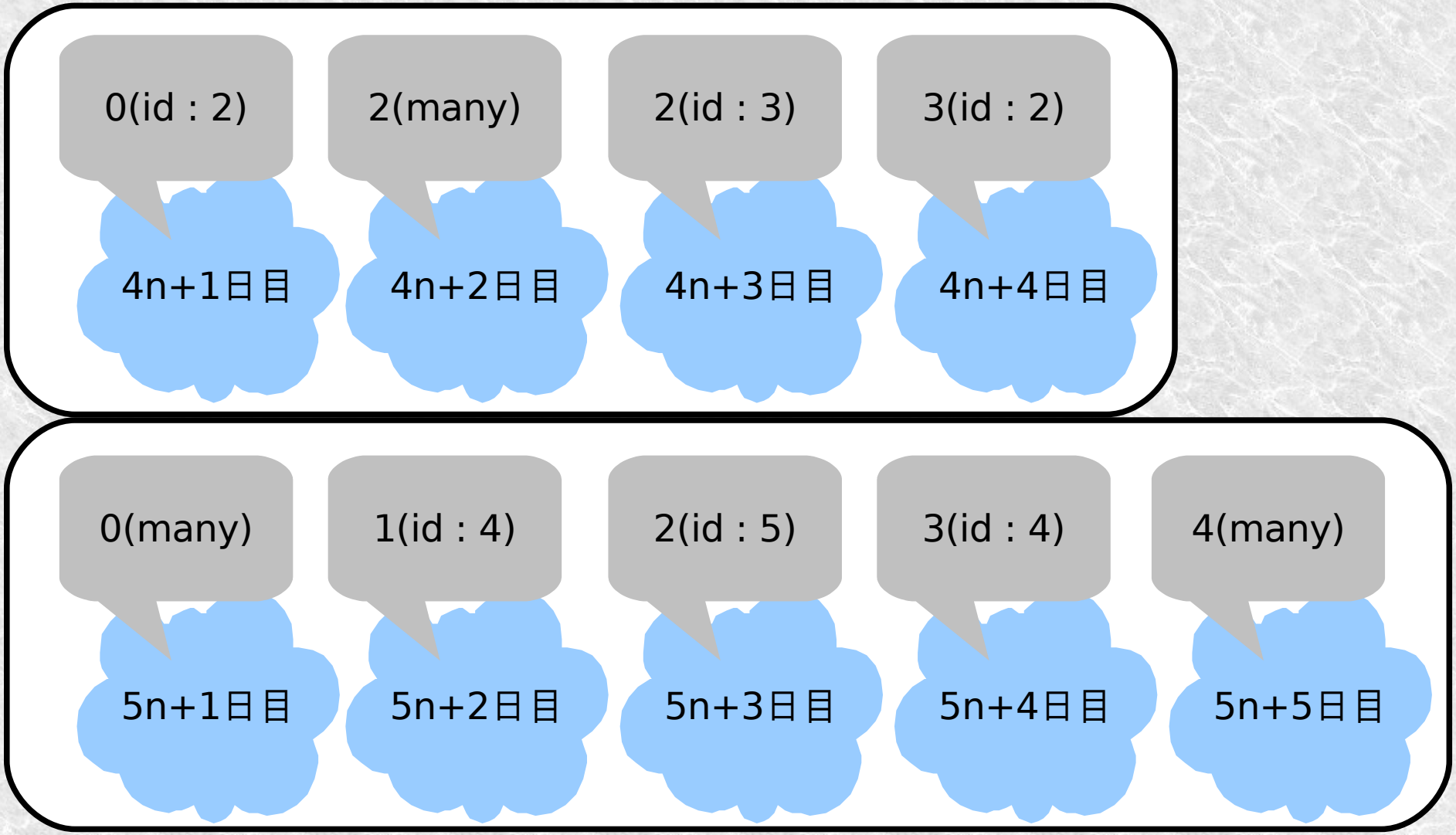
例題8-Eat or Not to Eat(4/5)

- ex.

	1日目	2日目	3日目	4日目	5日目
1	2	9	8	4	-
2	0	2	9	3	-
3	9	2	2	9	-
4	4	8	8	4	-
5	0	1	2	3	4
6	0	2	9	8	4

例題8-Eat or Not to Eat(5/5)

- 併合後の状態



例題9-The lightest language(1/5)

- k(<=26)文字のアルファベット集合A_kに対して文字列Sを$S \in A_k^*$、言語Lを$L \in 2^{|S|}$として定義する。各$a \in A_k$に対して重みが与えられたとき、S、Lの重みを以下のように定義する。

$$w(S) = \sum_{i=0}^{length(S)-1} w(S[i]), \quad w(L) = \sum_{S \in L} w(S)$$

Lの大きさn(<=10,000)が与えられたとき、任意の2要素が接頭語の関係にならないLの中での重みの最小値を求めよ。

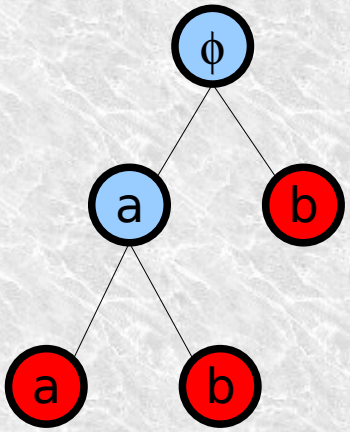
(Polish Olympiad in Informatics 1998)

例題9-The lightest language(2/5)

- ex.
  - A_k={a,b},w(a)=2,w(b)=5
  - N=3
    - 条件を満たすもののうち最小を達成するLは L={aa,ab,b}で、そのときw(L)=16
    - L={a,aa,b}だとw(L)=11だが、aがaaの接頭語になっているので不可

例題9-The lightest language(3/5)

- 条件を満たす言語は、言語中の文字列に関してTrieを構成するとリーフノードのみが選択されたものになっている
  - トップダウンにノードを「展開」して行くことで計算ができる



例題9-The lightest language(4/5)

- 解法
  - 以下のような貪欲アルゴリズムで解く事が可能
    - 始めに各単一アルファベットからなる文字列集合{[a],[b],[c],...}を持っておき、以下の処理を繰り返す
      - 集合の中で最小の重みを持つ文字列sをとりだして、新たに[s+a],[s+b],[s+c],...を集合に格納する
      - 集合変更前と変更後とで、集合中で最も重みの小さいn個の文字列をとってきてその総和を比べ、その値が変更後でより小さくなっているならば処理を続行し、そうでなければ変更前の状態を最適値として計算を終了する

例題9-The lightest language(5/5)

- どのようにして実現するか?
  - ヒープを使えばとりあえずよさそうだが...
    - 単純にやるとヒープの大きさがn^2程度になる(らしい)ので、メモリが辛い
      - 枝刈りを行って無駄なノードをヒープに入れないようにする
        - 前状態でn番目に小さかった値よりも重みの大きいノードを入れることを禁止
        - 解が改善されるのならばこのようなノードが最小のn個に含まれることはない
    - 単純にやると「最小のn個の総和値」や、上で述べた「n番目に小さいノード」の計算に時間がかかる
      - ヒープをもうひとつ用意する
        - 最大値が取り出せるヒープ
        - 操作の各段階で大きさが常にn個になるようにしておく
          - 各状態での最適解を意味する

ヤング盤(1/13)

- 以下のような性質を持つ二次元テーブルをヤング盤と呼ぶ
  - 任意の要素xに対して、xが左端及び下端に位置していない限り、左と下のセルには有意な要素が入っていてその値はx以下になっている
- 左下には全要素中最低の値が入っている
  - ➔ ヒープに似たような性質を持っているといえる
- 数学的には組合せ論などと関係の深い概念
  - 「分割」と対応

ヤング盤(2/13)

- ヤング盤の例

8					
8	8	9			
4	4	8	9		
2	2	4	8	8	9

ヤング盤(3/13)

- ヤング盤に関する(どうでもいい)諸定理
  - 要素数nのヤング盤の形状の総数a(n)は以下の漸化式で与えられる
    - a(1) = 1,a(2) = 2
    - a(n) = a(n – 1) + (n – 1)a(n – 2)
  - n個の要素が入るヤング盤の形状が与えられた時、これに1~nの数を入れる方法は以下の式で与えられる。

$$\frac{n!}{\prod(fook(t)+1)}$$

(fook(t)は、セルtに対するフック数(そのセルの右と上に有るセルの総数)を意味する)

ヤング盤(4/13)

- ヤング盤に対する操作
  - 挿入(bumping)
    - 最下行の左から右へ要素を順に見て行き、挿入したい要素xよりも大きい要素yが現れたならば、yの有ったセルにxを入れて上の行に対して再帰的にyの挿入操作を行う。このような要素が現れずに走査が終了してしまったときはその行の一番右にxを挿入する。
    - m×nのヤング盤に対して、計算量はO(mn)
      - 数学的側面が強い

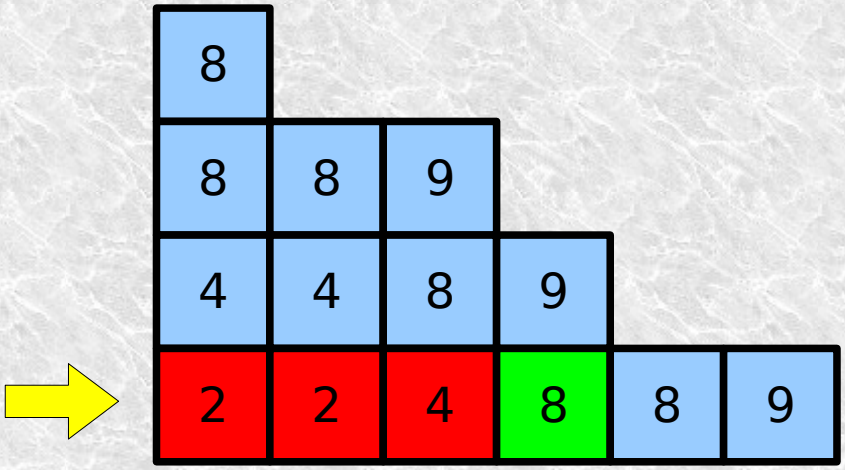
ヤング盤(5/13)

- ex.
  - 下のヤング盤に7を挿入

8					
8	8	9			
4	4	8	9		
2	2	4	8	8	9

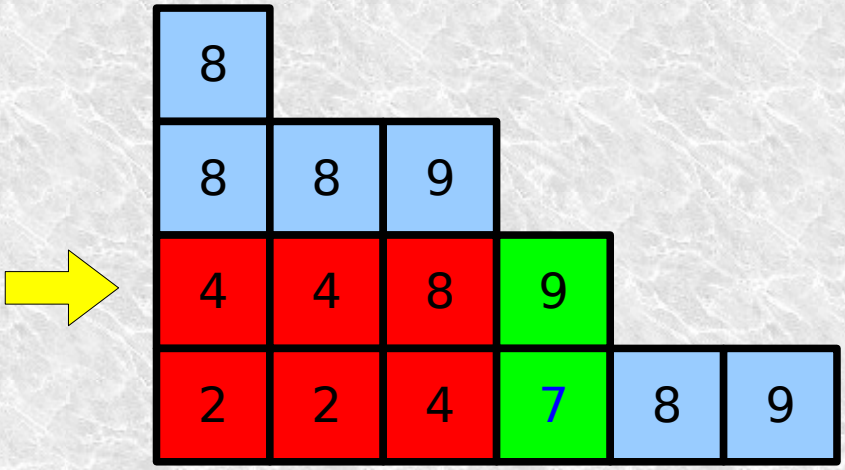
ヤング盤(6/13)

- 途中状態1
  - insert(7)



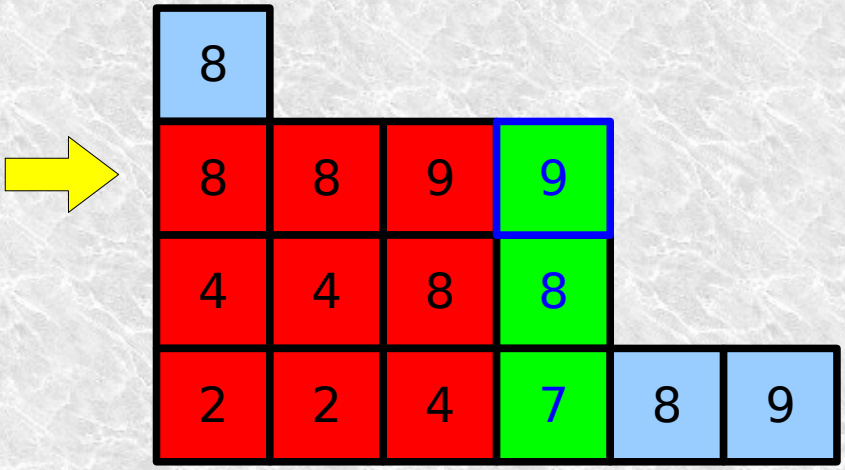
ヤング盤(7/13)

- 途中状態2
  - insert(8)



ヤング盤(8/13)

- 最終状態
  - insert(9)



ヤング盤(9/13)

- 要素探索
  - 最下列の右端から開始して、現在位置の要素yと、探索している要素xに関して以下の処理を繰り返す。
    - y<xならば左に移動する。左端に達していたのならば存在しないとして探索終了
    - x<yならば上に移動。上端に達していたのならば存在しないとして探索終了
    - x=yならば存在するとして探索終了
  - m×nのヤング盤に対して、計算量はO(m+n)

ヤング盤(10/13)

- ex.
  - 下のヤング盤で7を探索

8					
7	8	9			
4	4	8	9		
2	2	4	8	8	9

ヤング盤(11/13)

- 途中状態1

8					
7	8	9			
4	4	8	9		
2	2	4	8	8	9

ヤング盤(12/13)

- 途中状態2

8					
7	8	9			
4	4	8	9		
2	2	4	8	8	9

ヤング盤(13/13)

- 最終状態

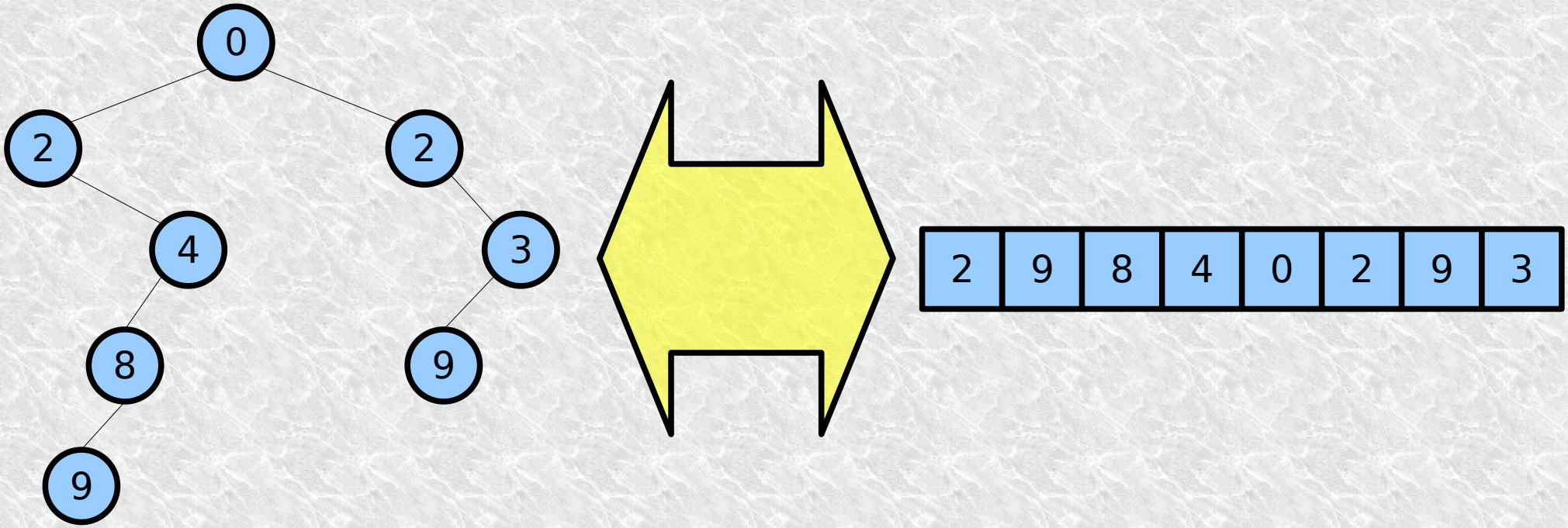
8					
7	8	9			
4	4	8	9		
2	2	4	8	8	9

カルテシアン木(1/14)

- 与えられた数列Aに対して、以下の性質を満たす二分木をAに対するカルテシアン木と呼ぶ
  - 任意のノードtに対して、その左部分木は元の数列でのtの左側部分列からなり、右部分木はtの右側部分列からなる
  - 任意のノードtに対して、tはtを含む部分木の中で最小の要素となっている(親ノードが常に子ノードよりも小さい)
- ルートが元数列の中で最小要素
  - → ヒープに似たような性質を持っているといえる
- LCA問題やRMQ問題と関係が深い(後述)

カルテシアン木(2/14)

- カルテシアン木の例



カルテシアン木(3/14)

- 与えられた数列(要素数n)からカルテシアン木を構築する方法
  - 単純にやると$O(n^2)$かかってしまう
    - 各段階で順次最小値を探して行く
  - O(n)でできるアルゴリズムが存在する

カルテシアン木(4/14)

- O(n)アルゴリズム
  - 空の木から初めて、k=1 to nで以下の処理を繰り返せば良い
    - 木の右経路(ルートから、常に右の子を手繰ることで得られる経路)$p=p_1p_2...p_n$に対して、$p_i<x<p_{i+1}$なるiを探して来て、
      - p_iの右の子をxにする
      - xの左の子をp_{i+1}にする
    - p_nから手繰っていって比較してゆけば、全てのノードは全体で高々1度の挿入操作と($p_i>x$となる)比較処理を受ける。$p_i<x$となる比較処理はループの回数であるn回発生するだけであるから、全体の計算量が確かに線形になることがわかる。

カルテシアン木(5/14)

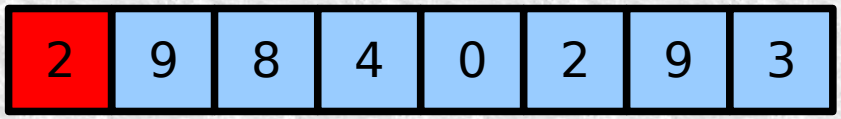
- カルテシアン木の構成
  - 初期状態

(null)

2	9	8	4	0	2	9	3

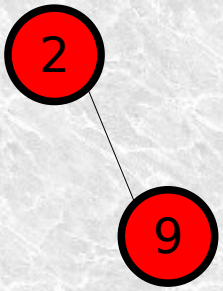
カルテシアン木(6/14)

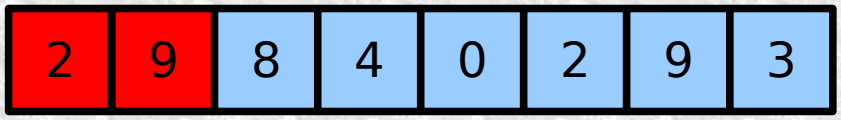
- カルテシアン木の構成
  - 途中状態1



カルテシアン木(7/14)

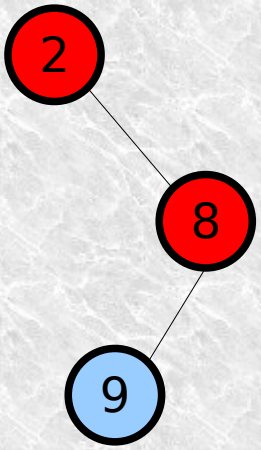
- カルテシアン木の構成
  - 途中状態2

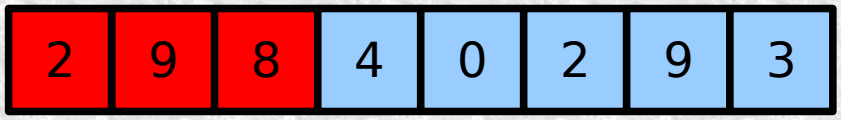




カルテシアン木(8/14)

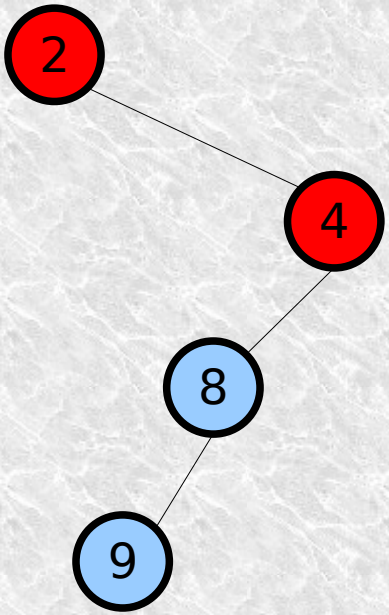
- カルテシアン木の構成
  - 途中状態3

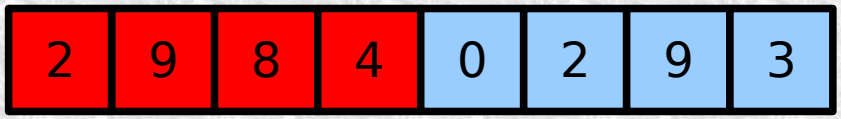




カルテシアン木(9/14)

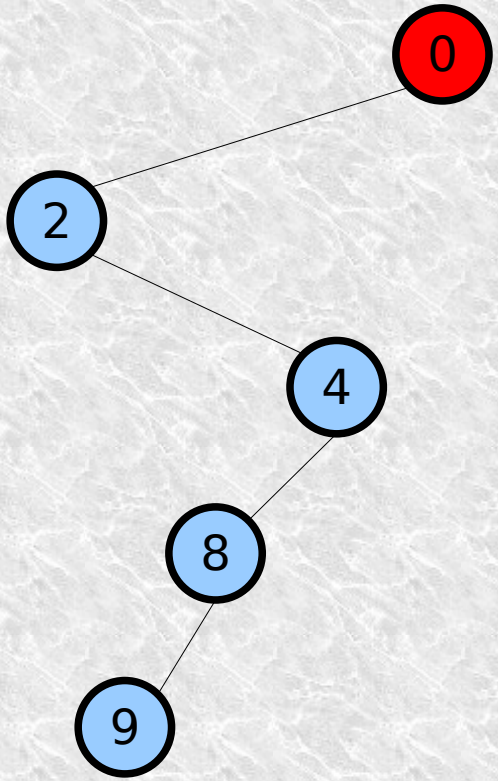
- カルテシアン木の構成
  - 途中状態4

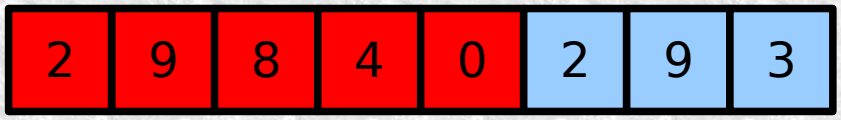




カルテシアン木(10/14)

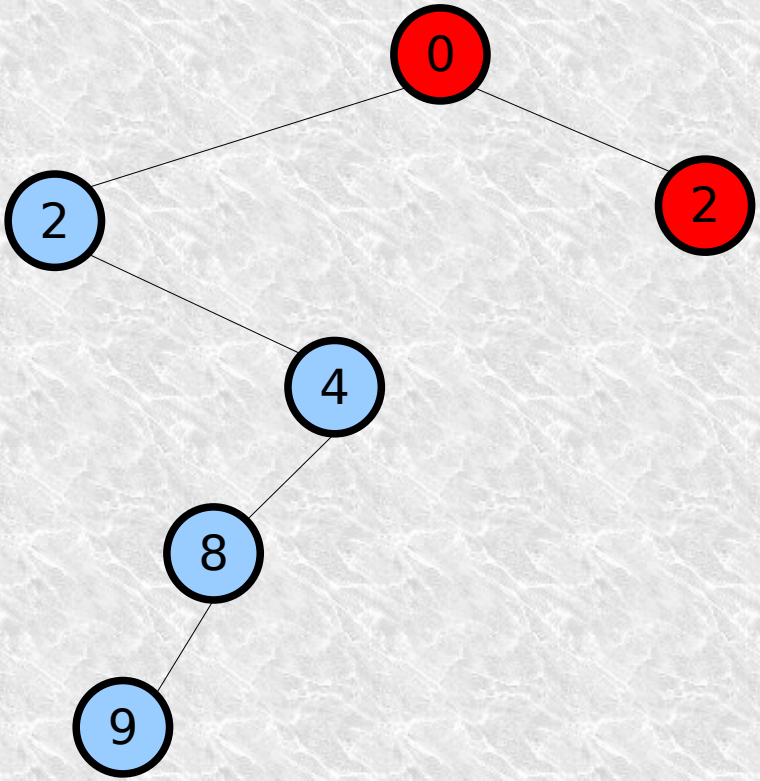
- カルテシアン木の構成
  - 途中状態5

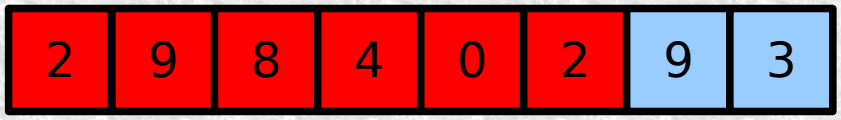




カルテシアン木(11/14)

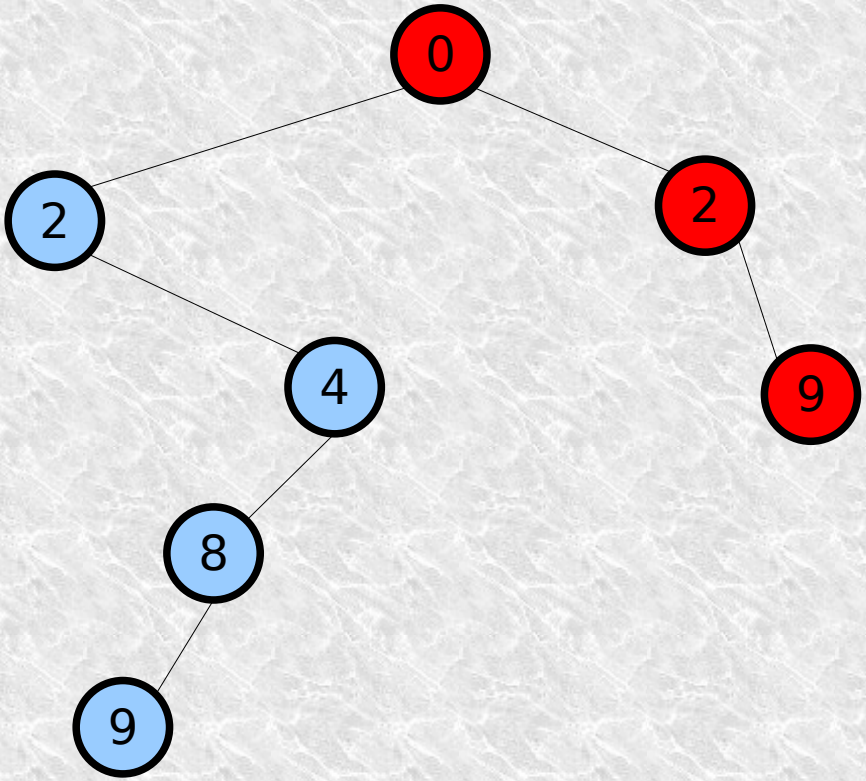
- カルテシアン木の構成
  - 途中状態6

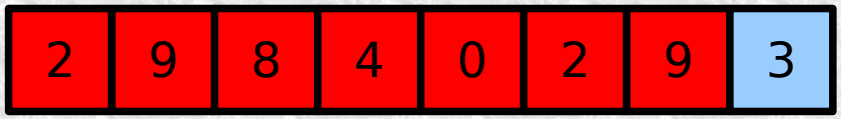




カルテシアン木(12/14)

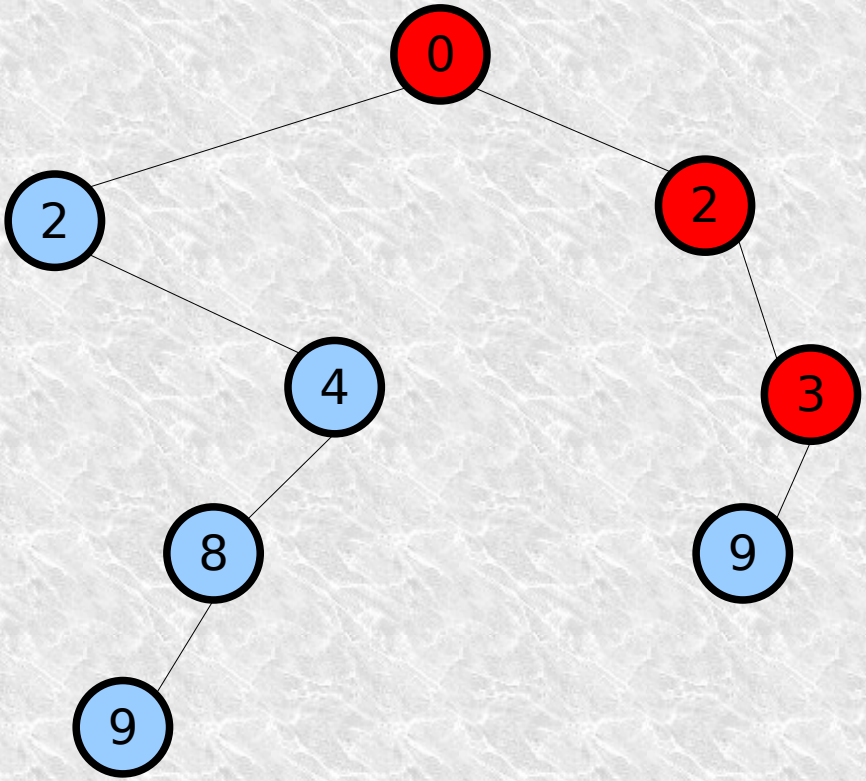
- カルテシアン木の構成
  - 途中状態7





カルテシアン木(13/14)

- カルテシアン木の構成
  - 最終状態



2	9	8	4	0	2	9	3

カルテシアン木(14/14)

- カルテシアン木の応用
  - カルテシアン木のLCA問題は、元の数列でのQMA問題になる
    - 性質から考えてみれば自明
    - ➔ 一般のQMA問題に対してO(n)-O(1)アルゴリズムを与えることになる