ICPC輪講第二回

理学部情報科学科4年 小島晃司

今回の輪講内容

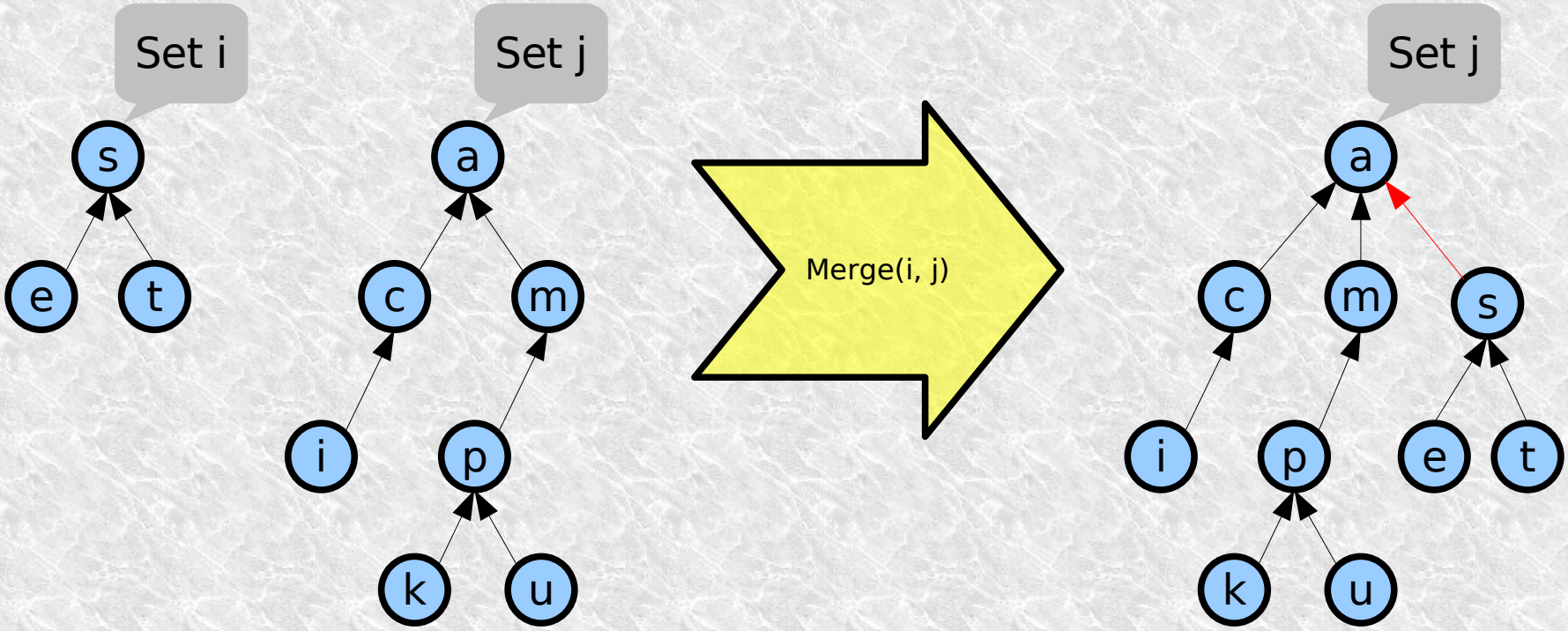
- 教科書『算法芸術与信息学競賽』(劉汝佳, 黄亮)の1.4節(pp.79~112)の前半
  - Disjoint set
  - Heap
- 教科書の例題解説が中心

Disjoint set(1/4)

- Merge-Find set, Union-Find setなどとも呼ばれる
- 互いに共通要素のない、幾つかの要素集合に対して以下の操作を行うことができる
  - 指定した2集合の併合(Merge)
  - 指定した要素の帰属集合の取得(Find)
- 親へのポインタを持つ木による実現が効率的
  - Merge操作はO(1)
  - うまくやると、n個の要素を持っているときにm回のFind操作を$O(m\alpha(m,n))$で行うことができる

Disjoint set(2/4)

- Merge操作
  - 一方の木のルートノードを他方の木のルートノードの子ノードにする
  - 各集合の要素数を覚えておいて、小さい方の木を大きい方の木にMergeする

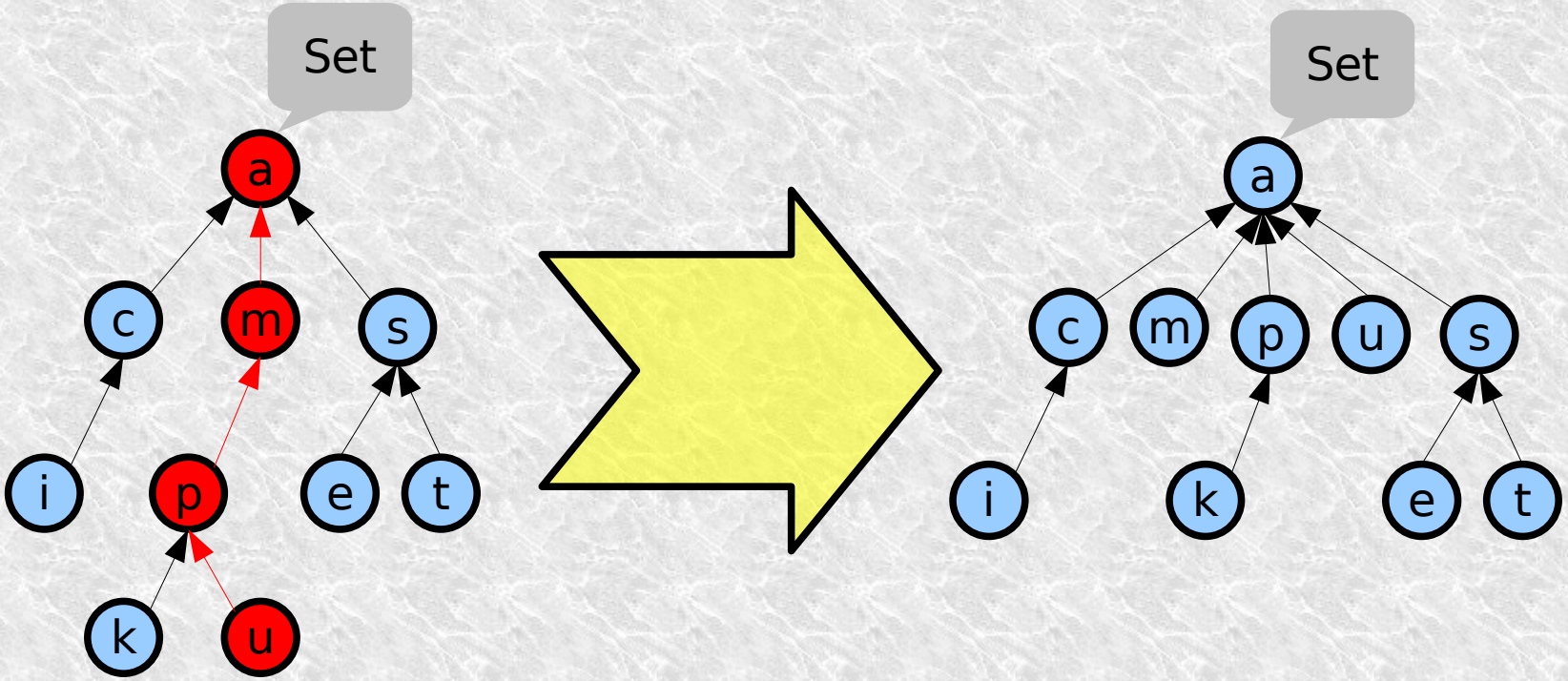


Disjoint set(3/4)

- Find操作
  - 指定された要素の親を辿っていって、ルートに到達したときにそこに貼られたラベルを返してやれば良い。
  - 木の深さを浅くするために、Find実行時に経路圧縮処理も行う
    - 辿った全てのノードをルートノードの直下の子ノードに変更

Disjoint set(4/4)

- ex. Find(u)実行時の効果



例題1-Word equations(1/5)

“0”,”1”から成る幾つかの文字列変数(高々26個)に対して

- 各変数の文字列長
- “0”,”1”と各変数による、文字列に関する1つの等号関係式(式中の文字列は10,000以上の長さにならない)

が与えられる。このときに、この制限を満たす解の総数を答えよ

(Polish Olympiad in Informatics 1998)

例題1-Word equations(2/5)

- ex.
  - (|a|, |b|, |c|, |d|, |e|) = (4, 2, 4, 4, 2)
  - 1bad1 = acbe
    - 解のひとつとしては

      a = 1001

      b = 00

      c = 0010

      d = 0000

      e = 01

    などがある。このとき等式の文字列は”100100100001”になる

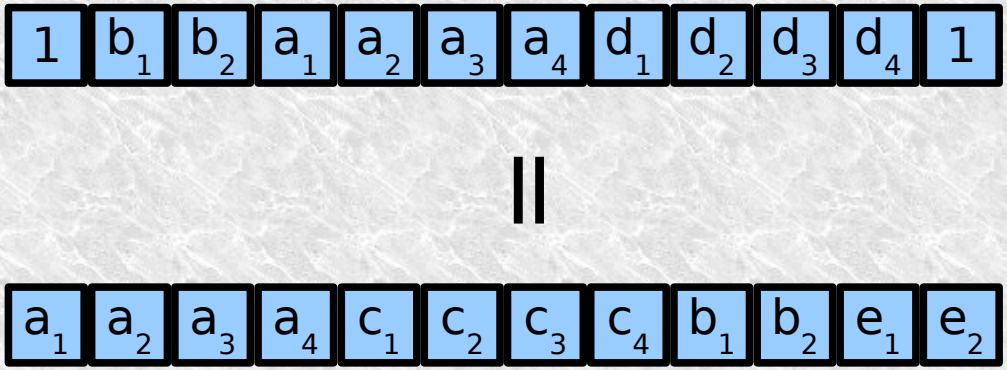
例題1-Word equations(3/5)

- 解法
  - ”0”,”1”と各文字列変数の各桁の文字を要素としたDisjoint setを利用すれば良い。
  - 統合関係式に於いて各桁を照合して、対応している要素が帰属している集合同士をMerge
    - 最後に残ったDisjoint setにおいて、” 0”, “1”が含まれていない集合の総数pに対して2^pを返してやれば良い。
  - 要素数をn,桁数をmとすると大体O(n+m)で計算可能
    - 初期状態の構築にO(n)
    - Findをm回行うのにO(mα(m,n)) ≈O(m)

例題1-Word equations(4/5)

- ex. の場合
  - 右の関係から、
    - 1とa_1
    - b_1とa_2
    - b_2とa_3

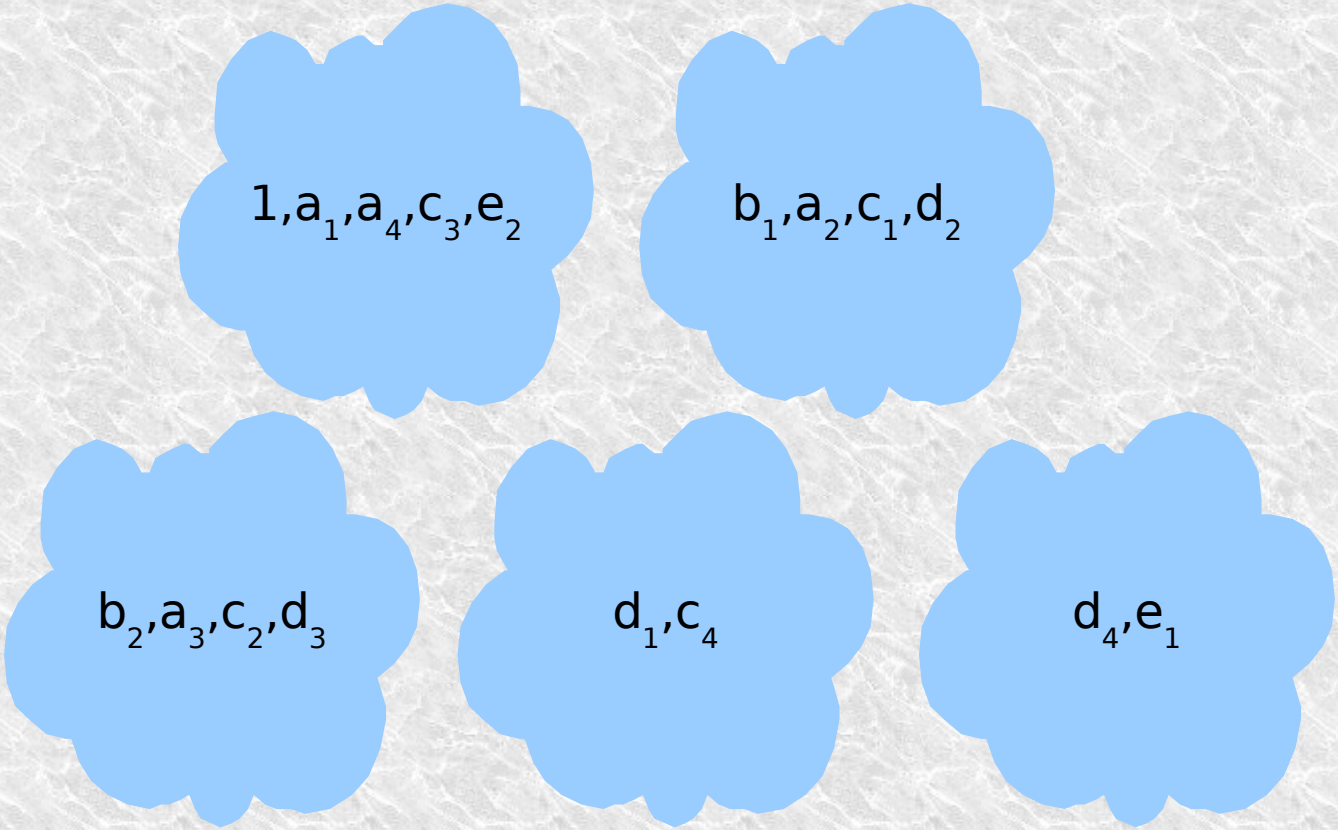
    …
    - 1とe_2



の、計12回のMerge操作を行えば良い。

例題1-Word equations(5/5)

- 最終的には以下のようになるので、答えは $2^4 = 16$



例題2-Gang(1/2)

- n人のギャングに対して、友好関係、敵対関係の情報を与えたときに、できるだけ多くの派閥を形成しようとすると幾つの派閥ができることになるのかを求めよ。但し友好関係、敵対関係に関して以下のような性質が成立しているとする
  - 自分と友好関係にある相手と友好関係にある者とは、推移的に友好関係が発生する
  - 自分と敵対関係にある相手と敵対関係にある者と友好関係が発生する

(Baltic Olympiad in Informatics 2003)

例題2-Gang(2/2)

- 解法
  - 例題1と同様、Disjoint setを利用することで簡単に解ける。詳細は省略
    - 友好関係にあるもの同士をMergeして行けば良い。
    - Disjoint setとは別に、敵対関係にある相手に関する情報を保持したテーブルが必要になる

例題3-銀河英雄伝説(1/15)

- 横一列に並んだ艦隊(戦艦数は高々30,000で、1,2,...と名前付けされているとする)に対して以下で定義する命令MとCを与えていったときに(命令の総数は高々500,000)得られる答えを求めよ。
  - Ｍｉｊ

    iが編成されている隊列をjが編成されている隊列の最後尾に移動、合流させる命令
  - Ｃｉｊ

    クエリ命令。この命令が出された段階で、まずiとjが編成されている隊列が同じかどうかを調べ、同じであればiとjの間に存在している戦艦の総数を返す

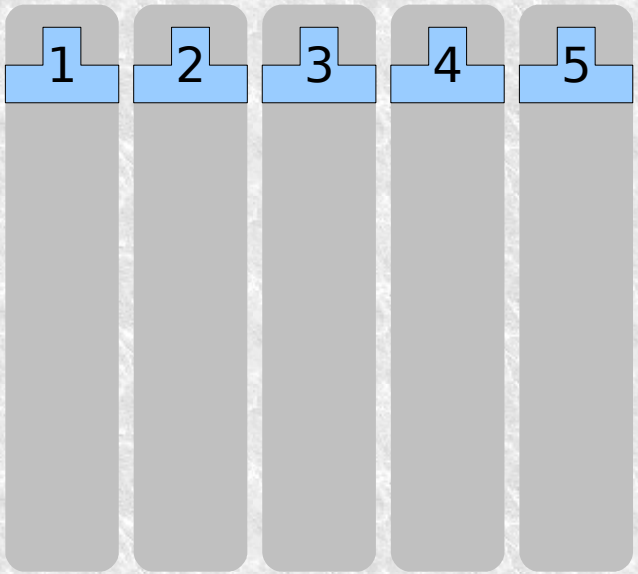
(NOI 2002, Galaxy)

例題3-銀河英雄伝説(2/15)

- ex.
  - 戦艦数n = 5としたときに、以下のような命令列を考える
    1. C 1 2
    2. M 1 2
    3. C 1 2
    4. M 4 5
    5. M 1 3
    6. C 3 5
    7. M 2 4
    8. C 3 5

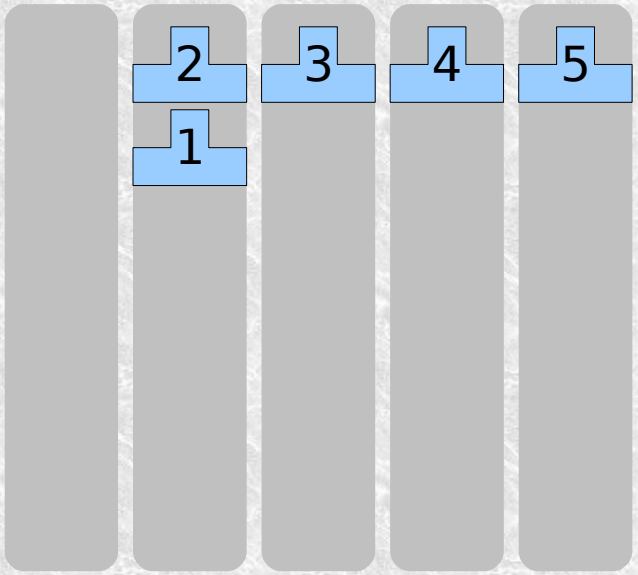
例題3-銀河英雄伝説(3/15)

- 1のクエリ(Ｃ１２)命令実行時の状態
  - 下図から1と2は異なる列にいることがわかる



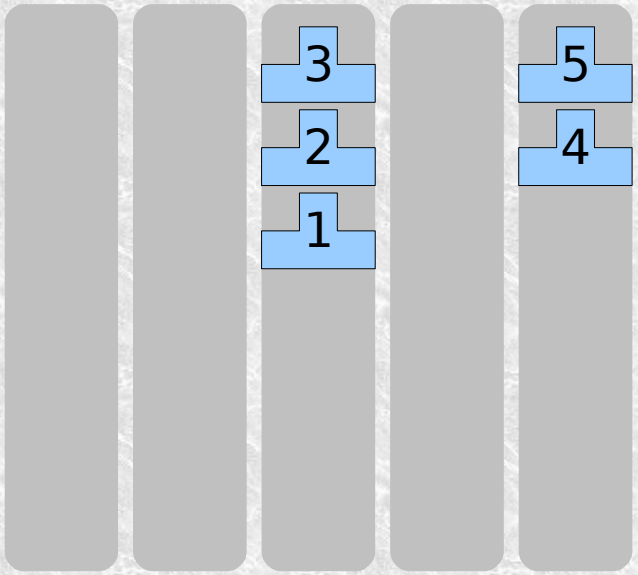
例題3-銀河英雄伝説(4/15)

- 3のクエリ(Ｃ１２)命令実行時の状態
  - 下図から1と2は同じ列にいて、間には0個の戦艦がいることがわかる



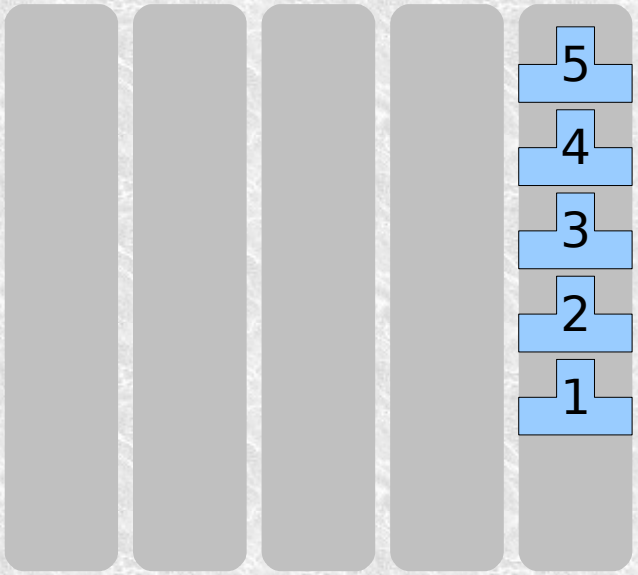
例題3-銀河英雄伝説(5/15)

- 6のクエリ(Ｃ３５)命令実行時の状態
  - 下図から3と5は異なる列にいることがわかる



例題3-銀河英雄伝説(6/15)

- 8のクエリ(Ｃ３５)命令実行時の状態
  - 下図から3と5は同じ列にいて、間には1個の戦艦がいることがわかる



例題3-銀河英雄伝説(7/15)

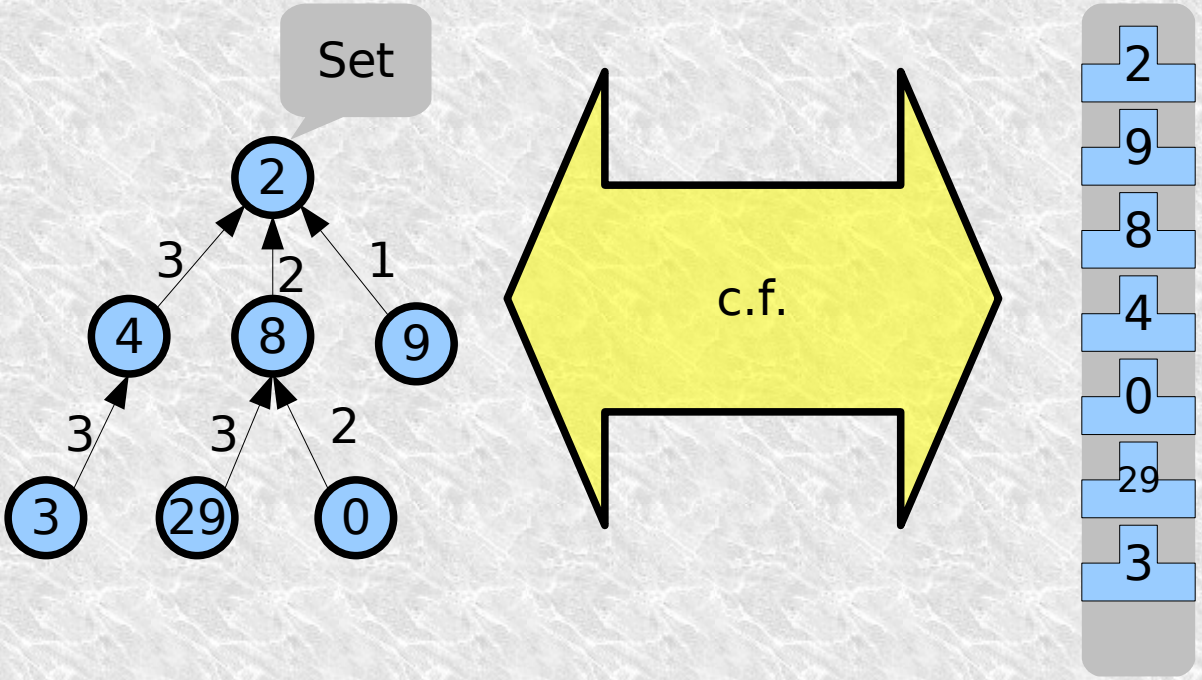
- 解法
  - 同じ隊列にいるかどうかだけを判定するならば単純にDisjoint setを利用してやれば良いが、それだけでは間にある戦艦数を求めることはできない
    - Disjoint setの変形版を利用

例題3-銀河英雄伝説(8/15)

- 従来のDisjoint set
  - 親へのポインタを保持した森
  - 各集合の要素数も保持していると仮定する
- 今回の拡張
  - 親へのポインタに「距離」を表す整数値を付加してやる
    - 自分と親の間の戦艦数を意味する
    - この値が正しい値を保持するようにMerge/Find時の処理を拡張する

例題3-銀河英雄伝説(9/15)

- 拡張されたDisjoint set

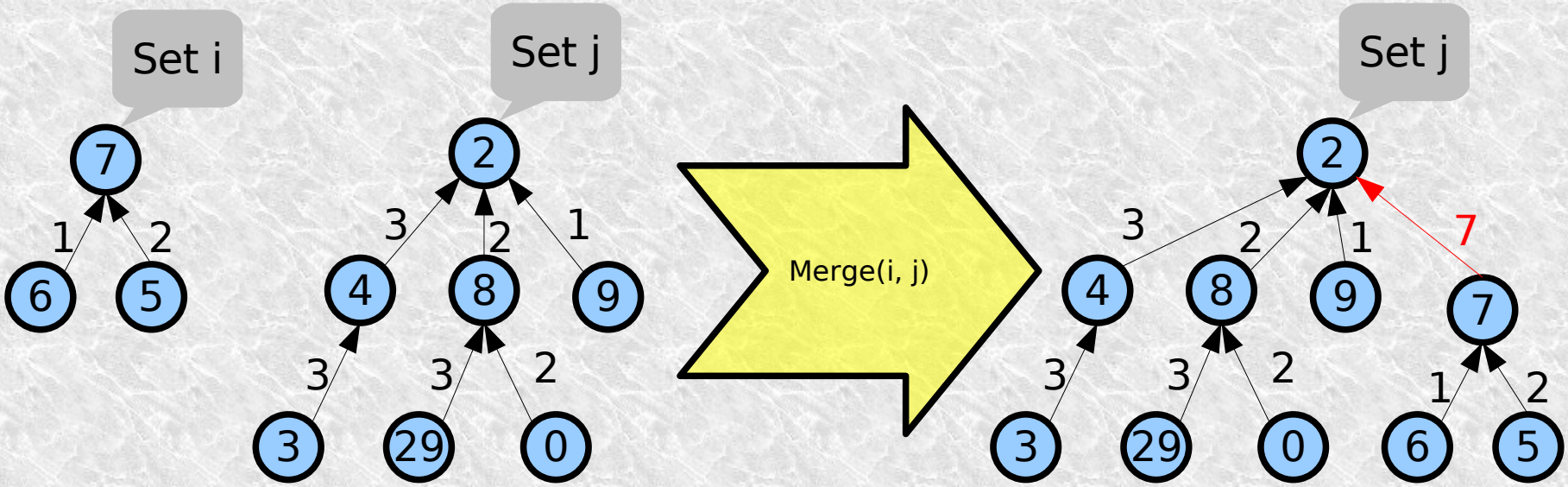


例題3-銀河英雄伝説(10/15)

- Mergeの拡張
  - 新しく付加される、一方のルートノードから他方のルートノードへのポインタの値を正しく設定すれば良い
    - ある隊列が別の隊列の最後尾に付けたときに、元々先頭だった戦艦と新しくできた隊列の先頭との距離のことだから、合流先の隊列の元々の総戦艦数であることがわかる
    - 親子関係が隊列に於ける前後関係を意味するために、Ｍｉｊに対しては必ずiの属する木の方をjの属する木に付加するようにしなくてはならない。

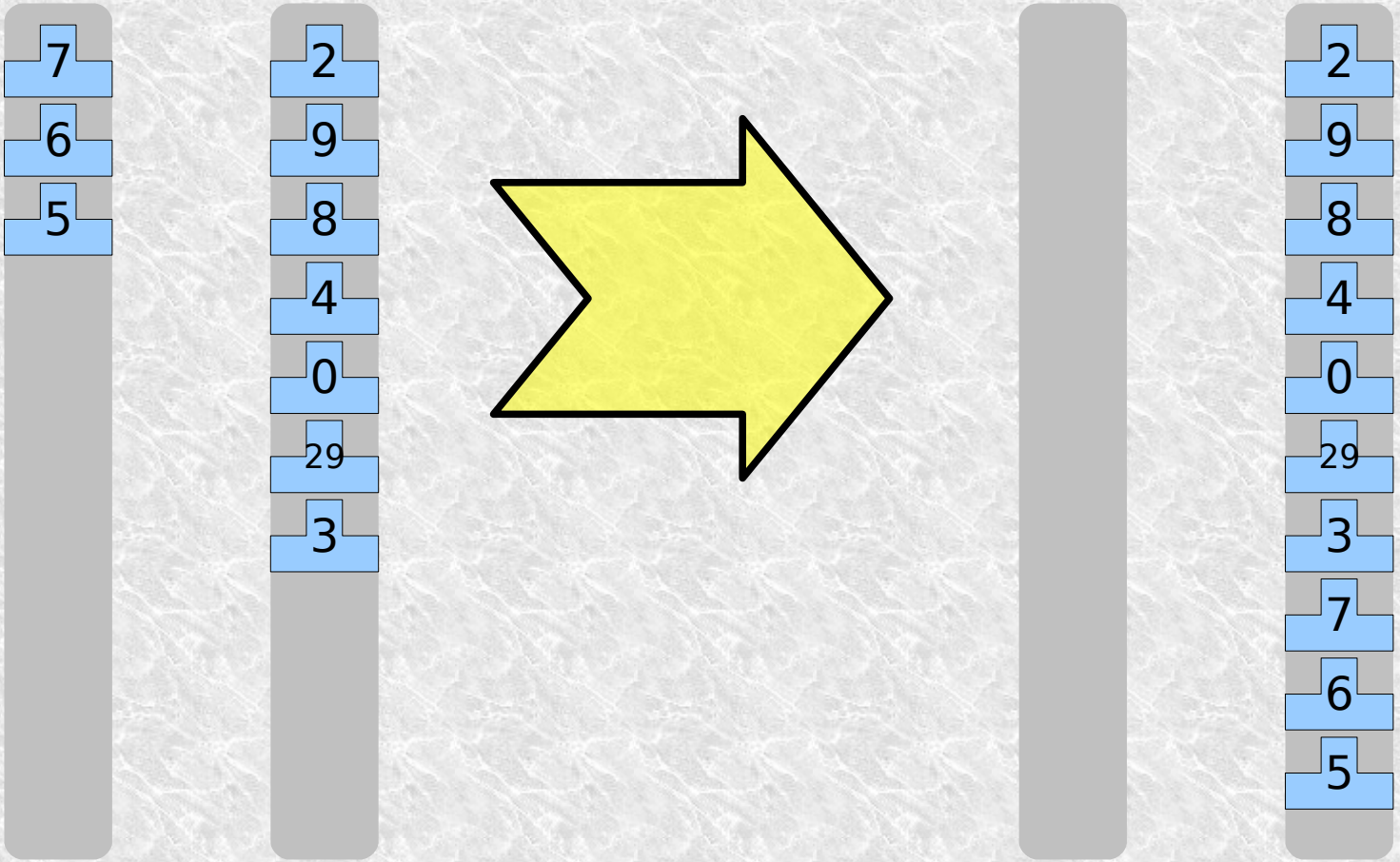
例題3-銀河英雄伝説(11/15)

- 拡張されたMerge操作



例題3-銀河英雄伝説(12/15)

- c.f.

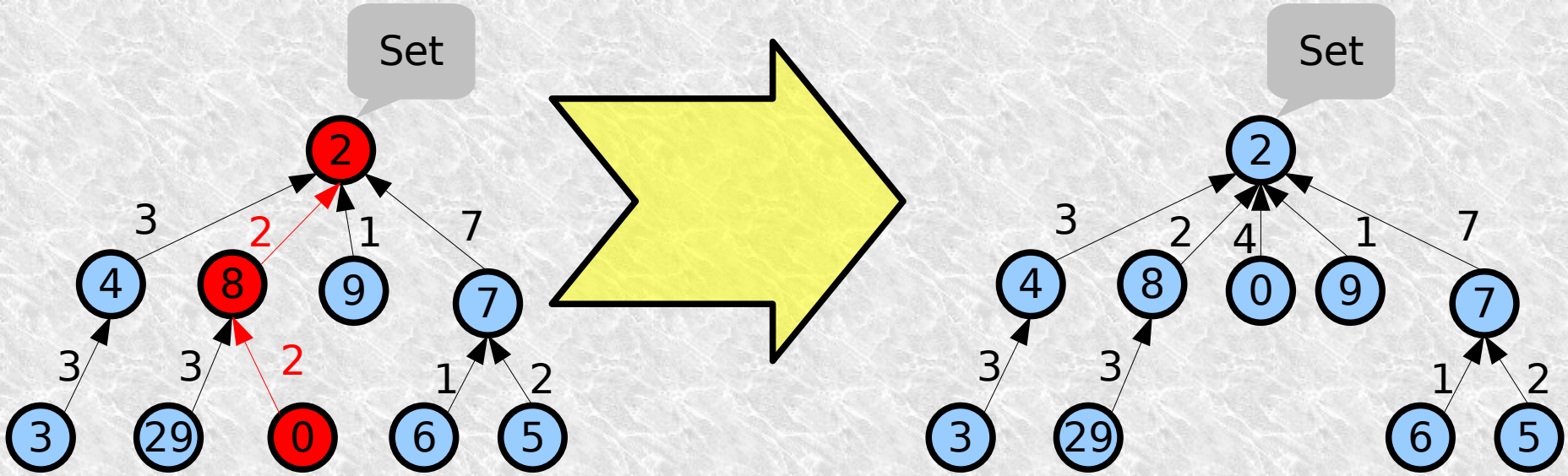


例題3-銀河英雄伝説(13/15)

- Find操作の拡張
  - Find自体の操作を変更する必要はない
  - 経路圧縮の際に貼りかえるポインタの値を適宜変更してやる必要がある
    - 始めにFindで要求された要素からルートノードへの距離がわかる(探索した経路の全体長)ので、そこからボトムアップに計算可能
      - new_d[parent] := new_d[child] – old_d[parent]

例題3-銀河英雄伝説(14/15)

- 拡張された経路圧縮操作
  - ex. Find(0)実行時の効果



例題3-銀河英雄伝説(15/15)

- 同一隊列間の2戦艦間の距離の算出
  - このような拡張されたDisjoint setを利用すれば、同一の隊列に含まれているとわかった2つの戦艦の間にいくつ別の戦艦が含まれているかを求めることは容易
    - Find時に辿った2つの各経路の全体長d_i, d_jに対して

      |d_i – d_j| - 1

      を計算すれば良い。
- 計算量
  - このDisjoint setでは小さい木をMergeさせるという処理ができないので、全体の計算量は戦艦数n,命令数mとしてO(n + mlogn)

例題4-Monkey(1/4)

- n匹の猿が木から以下のような状態でぶら下がっているとする(nは高々200,000で、猿達は1,2,...と名前付けされているとする)。
  - 猿1のみが木に尻尾を巻き付けていて、残りの猿達は直接木にぶら下がってはいない
  - (当然)各猿には手が二本あり、各々の手は高々1匹の猿の尻尾を握ることができる

このとき、時間0から初めて各1単位時間毎にいづれかの猿の一方の手が離されるという現象がm単位時間(mは高々400,000)続いたときに、各々の猿が木から離れて地面に落下する時刻を求めよ

(Polish Olympiad in Informatics 2003)

例題4-Monkey(2/4)

- ex.(n, m) = (3, 2)とする
  - 右のような状態から以下のようにして手が離されて行くとする

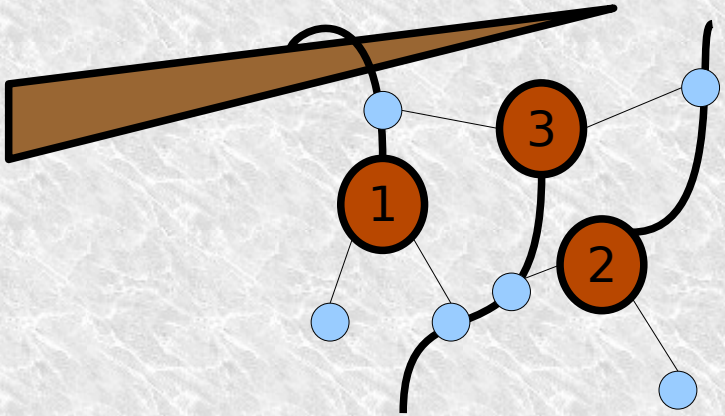
    0s. 1の右手

    1s. 3の左手

    → 1.落ちない

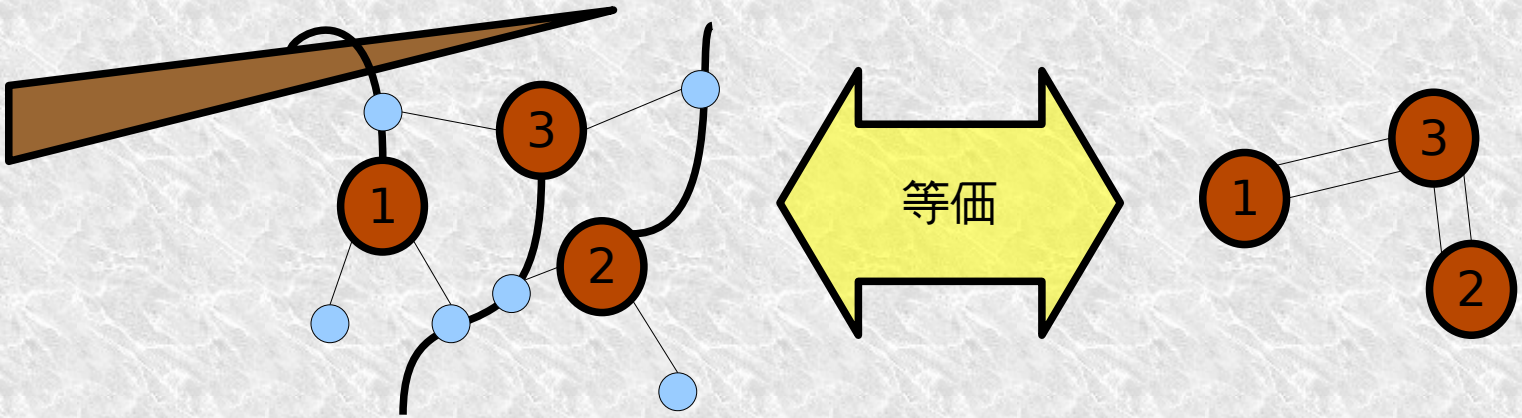
    2.1s時に落ちる

    3.1s時に落ちる



例題4-Monkey(3/4)

- 猿1に繋がらなくなった猿は、その瞬間に地面に落下する
  - 猿達をノードにした無向グラフに対して、段々枝を切っていったときに、各ノードがノード1の連結成分でなくなった瞬間を求めることと同値だが...
    - 単純にやるとO(nm)で終わらない(グラフは単純でも木構造でも無い)



例題4-Monkey(4/4)

- 解法:Disjoint Setを利用
  - 時間に関して順方向に見て行くとSplit操作なのだから、逆方向に見て行けば良い
    - 終状態を求める
      - 始状態の構成にO(n),シミュレーションでO(m)だから、O(n+m)で可能
    - 終状態でのグラフの連結成分分解を行う
      - 計算量はO(n)
      - ➔ 得られた連結成分集合をDisjointSetの初期状態とする
    - 時間を遡って、離された状態を復元してゆく
      - 切られた枝の両端点に対応する集合をMergeして行けば良い
      - 各ノードに対して、初めて1が含まれる集合にMergeされた時刻を覚えてやれば、それが答えになる
      - m回Find/Mergeするから、計算量は大体$O(m\alpha(m,n)) \approx O(m)$

例題5-蝸牛(1/9)

- n匹の蝸牛がいて、各々1,2,3,...nの移動速度で移動できるとする(nは高々1,000,000)。これらの蝸牛に対して以下の二種類の操作を行うことを考える(操作回数は高々n回)
  - 井戸の中に適当な蝸牛を一匹投げ込み、井戸の蓋を閉じる
  - 井土の蓋を開け、一番始めに井戸から這い上がってきた(井戸にいる中で一番速い)蝸牛を取りだし、蓋を閉じる。一度取り出された蝸牛が再び井戸に入れられることはないとする

このとき、各蝸牛が井戸から取り出された順序を答えよ。

(経典問題)

例題5-蝸牛(2/9)

- 要するに、ヒープのpush/pop処理のシミュレーション問題。但しpushされる値は1~nで、同じ値は高々1回しかpushされない
  - ✗ 単純にヒープで実装するとO(mlogm)で時間が足りない(はず)。
  - ➔ Disjoint setを利用することでO(n + m)で解ける

例題5-蝸牛(3/9)

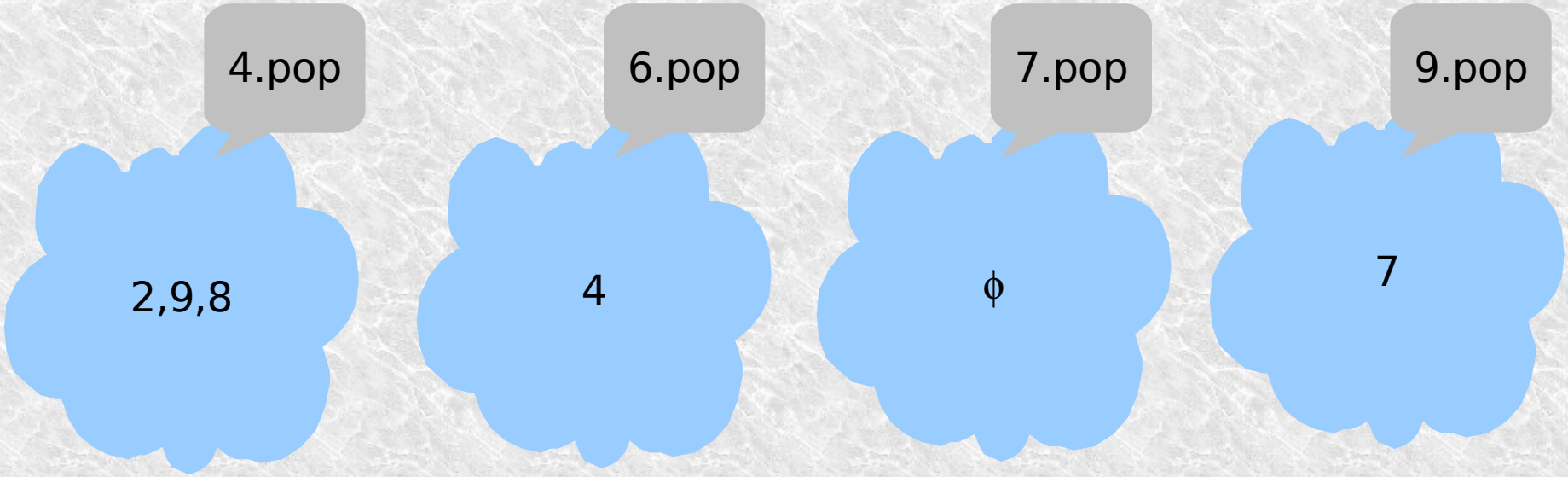
- 解法
  - 操作列をpop操作で区切ってpush操作列の集合を求める
    - 各操作列にはその直後に来るpop操作を関連づけておく
    - 計算量はO(m)
    - → Disjoint setの初期状態になる
  - k = n downto 1で以下の操作を繰り返す
    - push k操作を含む操作列を探す
      - → 再帰的に、その時点でヒープに存在しうる値は全てk以下であることがわかっているので、その直後にあるpop操作では必ずkがpopされる
    - 見付かった操作列を、その隣りにある操作列とMerge
    - n回Find/Mergeするので計算量は$O(n\alpha(n,m)) \approx O(n)$

例題5-蝸牛(4/9)

- ex.(n, m) = (9, 9)とする
  - 以下のような操作を考える
    1. push 2
    2. push 9
    3. push 8
    4. pop
    5. push 4
    6. pop
    7. pop
    8. push 7
    9. pop

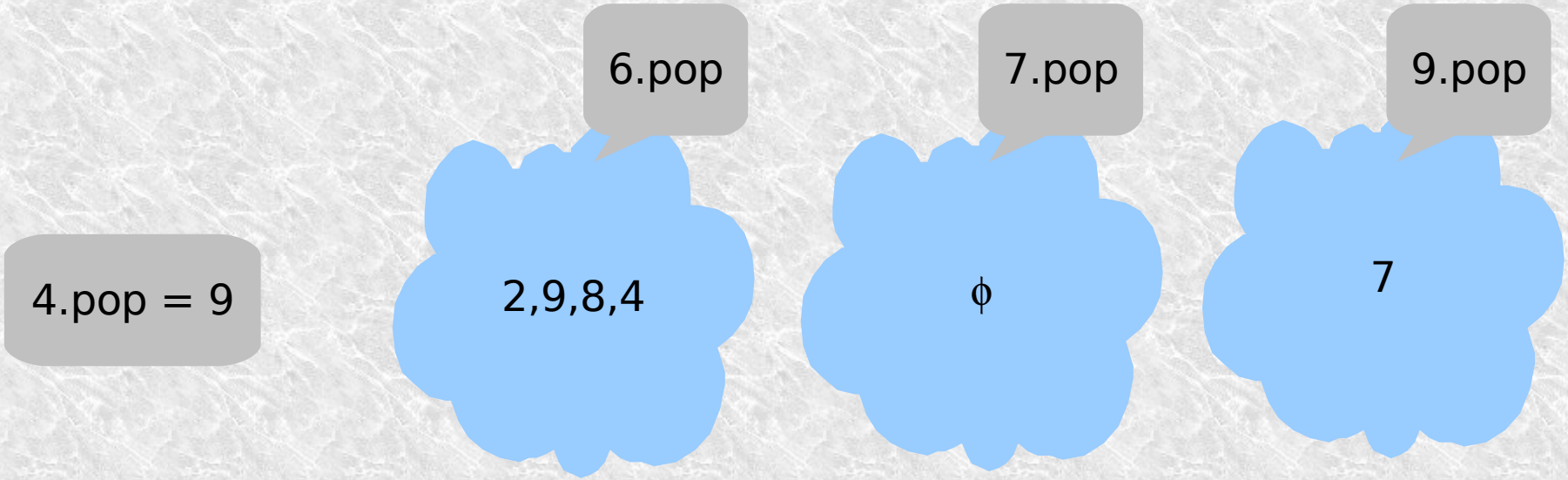
例題5-蝸牛(5/9)

- 初期状態



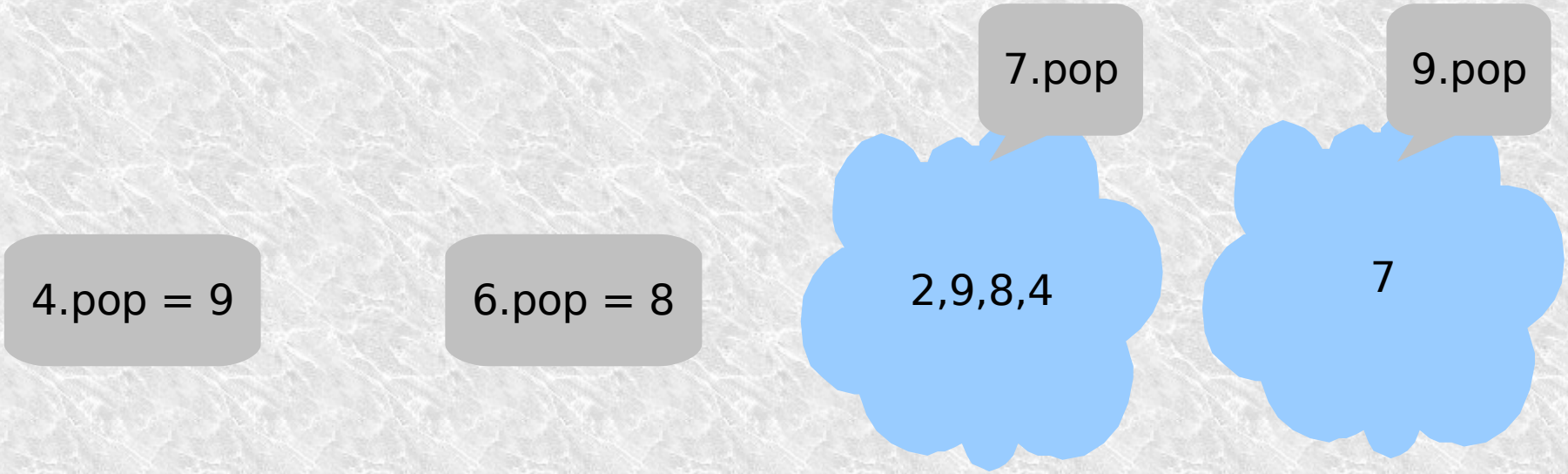
例題5-蝸牛(6/9)

- k = 9の時の状態



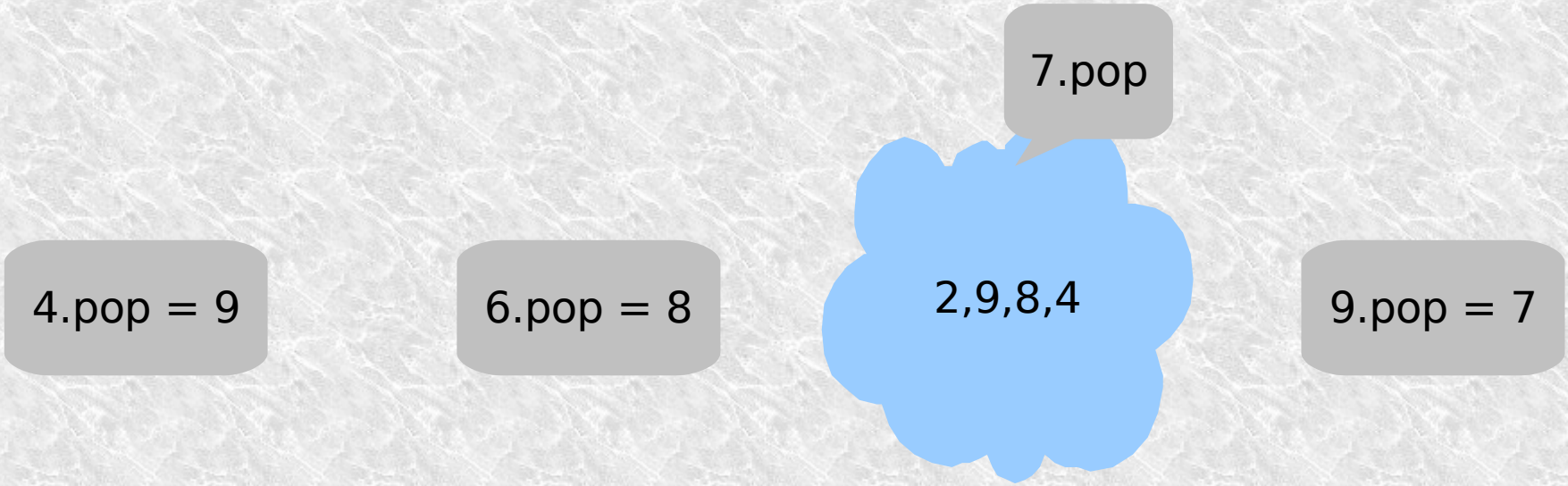
例題5-蝸牛(7/9)

- k = 8の時の状態



例題5-蝸牛(8/9)

- k = 7の時の状態



例題5-蝸牛(9/9)

- k = 4の時の状態

4.pop = 9

6.pop = 8

7.pop = 4

9.pop = 7

Heap

- popすると最小値が取り出せるデータ構造
  - pushにO(logn)
  - popにO(logn)

詳細は省略

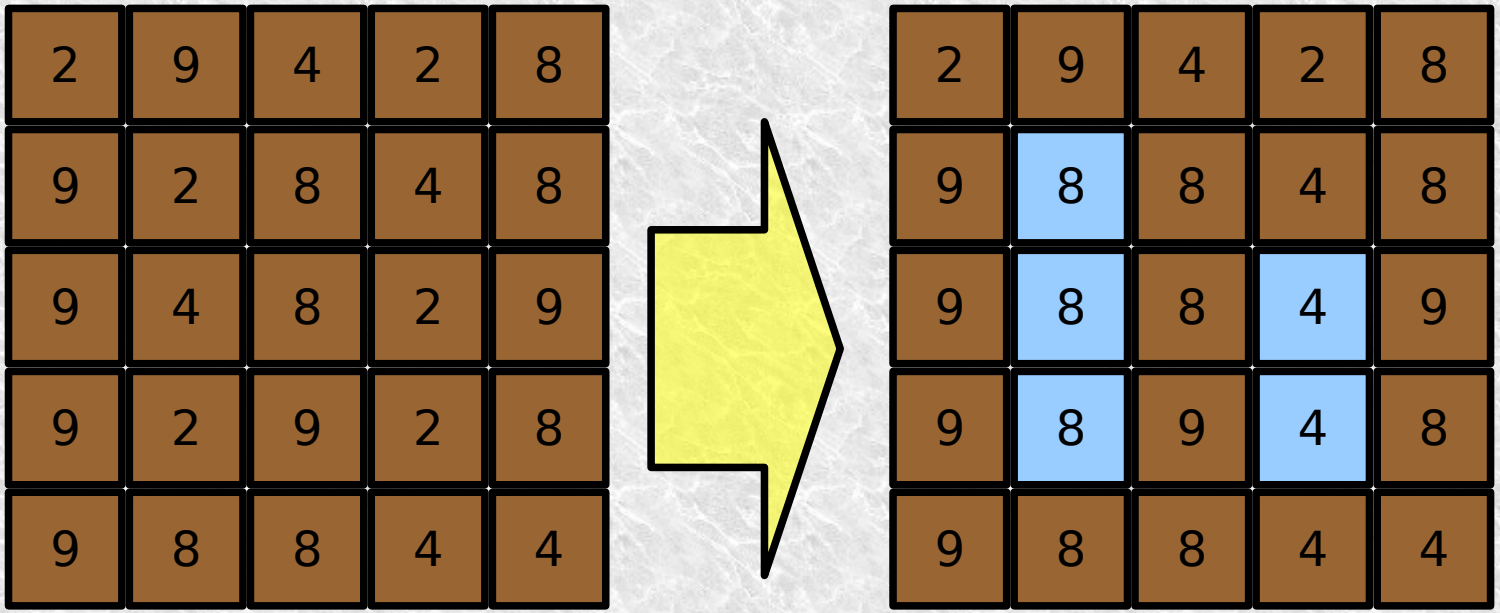
例題6-Water(1/13)

- 各格子毎に整数値の高さを持つ、凸凹した土地がある。この土地に十分な量の雨が降り注いだときに、各格子に溜まる水の量を求めよ。但し土地の端からは水はあふれ出てしまうものとする。

(Polish Olympiad in Informatics 1999)

例題6-Water(2/13)

- ex.

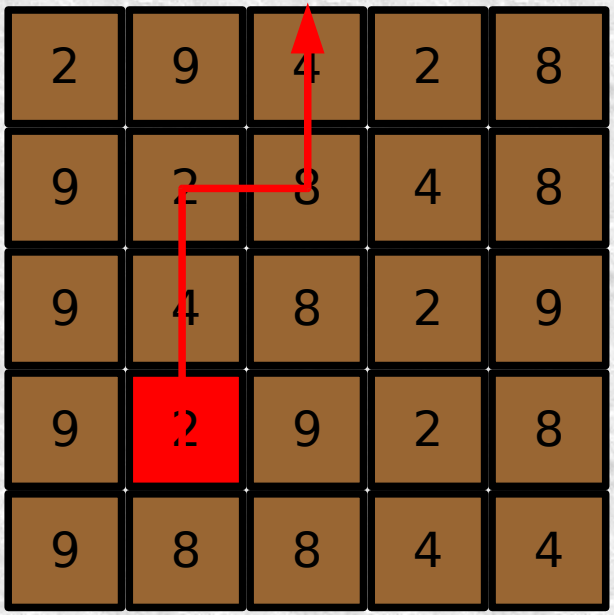


例題6-Water(3/13)

- 解法
  - ある格子に溜まる水の量は、その格子から土地の端の格子に至る任意のパスpに対して、パス中の格子点高さの最大値の最小値で与えられる。

$$water(x,y)=min_p\{max\{h(a,b)|(a,b)\in p\}\}$$

    - 「周り」の格子点情報からその「内部」に関する格子点情報を逐次的に求めることができる



例題6-Water(4/13)

- 具体的な手順
  - 始めに、土地の一番外側の格子点情報をヒープに入れておく
    - 一番外側の土地に水が溜まることはない
  - 以下の操作をヒープが空になるまで繰り返す
    - ヒープに入っている格子の中から高さが最小になるものを取り出す
    - その格子点に隣接している全格子に対して、その格子の高さが計算済みでなければ以下の式によってその高さを更新、確定してヒープに格納する
      - $h(neighbor) := max\{h(neighbor), h(popped)\}$
- 計算量は、m×nの土地に対してO(mnlogmn)

例題6-Water(5/13)

- ex. の場合
  - 初期状態

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

例題6-Water(6/13)

- 途中状態1

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

例題6-Water(7/13)

- 途中状態2

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

例題6-Water(8/13)

- 途中状態3

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

例題6-Water(9/13)

- 途中状態4

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

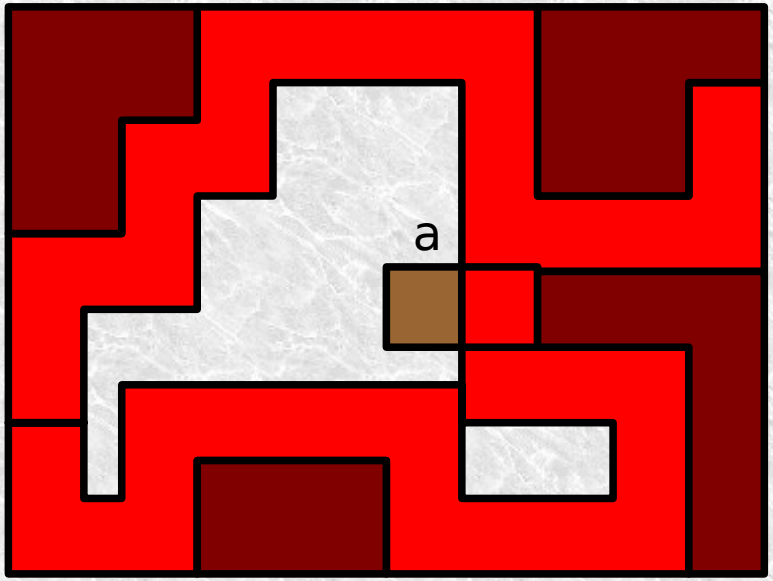
例題6-Water(10/13)

- 最終状態

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

例題6-Water(11/13)

- なぜこの方法で上手く行くのか?
  - ある格子点(x,y)が取り出されたときに、計算済みでない任意の(x,y)の隣接点aに関して、その点は現在ヒープに入っているノードで囲まれていることがいえる
    - ヒープの初期状態と、ヒープへの格納の仕方から自明



例題6-Water(12/13)

- すると、aに溜まる水の高さを達成する、aから端格子へのパスpは、必ずヒープ中の格子の何れかを通ることになる
- ヒープに格納されている値は確定済なので、ヒープ中の点(x',y')を通る任意のパスp'の中での最適値はh(x', y')以上になる
  - 最適のp'は、(x',y')での最適値を達成するp''に対して

    p' = a→…→(x',y')→p''

    と記述することができ、これはp''を包含している

例題6-Water(13/13)

- ヒープに格納されているものの中で最小のものが(x,y)だから、パス

  p = a→(x,y)→p''

のときに最小値h(x,y)を達成し、そのときの値は

$$max\{h(a), h(x, y)\}$$

となることがいえる

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)

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

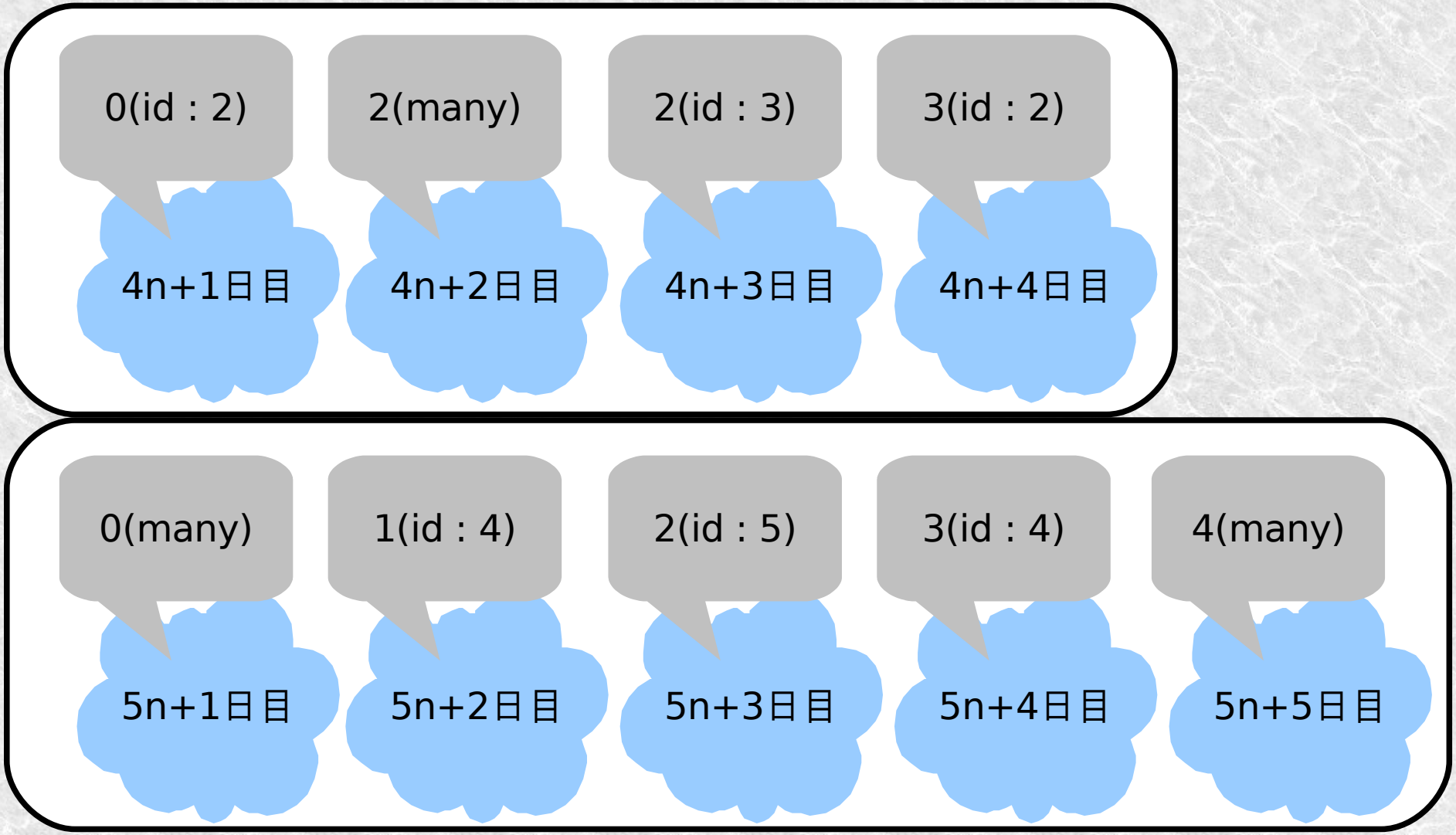
例題8-Eat or Not to Eat(4/5)

- 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。

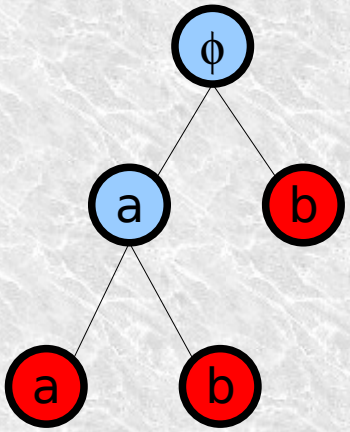
(Polish Olympiad in Informatics 1998)

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