

データ構造とアルゴリズム IB

九州大学大学院システム情報科学研究院
情報学部門
横尾 真

E-mail: yokoo@inf.kyushu-u.ac.jp
<http://agent.inf.kyushu-u.ac.jp/~yokoo/>

自己紹介

- 1986年東京大学大学院工学系研究科 電気工学専門課程 修士課程修了
- 同年 日本電信電話株式会社 (NTT) 入社
- NTT情報通信処理研究所 (神奈川県横須賀市), NTTコミュニケーション科学基礎研究所 (京都府相楽郡) 等に勤務
- 人工知能, マルチエージェントシステムに関する研究に従事
- 1995年 博士 (工学), 東京大学工学系研究科 電子情報工学専攻
- 2004年4月より九州大学システム情報科学研究院 教授, 2012年より主幹教授

223

成績に関して

- 小テスト, 期末試験, (レポート?) の成績で判断
- 出席は取るが, 試験の成績が良ければ, 出席率は問わない (小テストは受験するように!)
- 不合格ぎりぎりぐらいの場合は出席率も考慮するかも知れない

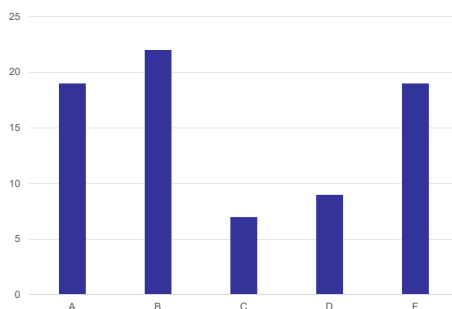
224

GPA制度(2007年度より導入)

- A: 90-100点 : 4: 特に優れている
- B: 80-89点 : 3: 優れている
- C: 70-79点 : 2: 普通である
- D: 60-69点 : 1: 一応の学修成果があり, 単位は認める
- F: 59点以下 : 0: 不合格

225

2017年度前期の成績



226

予定

- 6/8: 第1回: ヒープソート
- 6/15: 第2回: クイックソート
- 6/22: 第3回: 線形時間ソーティング
- 6/29: 第4回: ハッシュ表
- 7/6: 第5回: 2分探索木
- 7/13: 小テスト (範囲は1-3回まで)
- 7/20: 休講
- 7/27: 夏学期期末試験

227

講義の目的

- 目的: データ構造とアルゴリズムの基礎を身につける
- 電気情報工学科としての最低限の教養
 - 身につけていないと困る (アルゴリズムとデータ構造II, 卒論, 大学院進学/就職後)

228

講義について

- パワーポイントのスライドを用いる
- 教科書 (近代科学社, コルメン, ライザーソン, リベスト, シュタイン著, アルゴリズムイントロダクション 第一巻, 第三版) に準じて講義を進める
 - 旧版を持っているなら買い換えなくてもよい
- スライドはホームページで後日公開する
 - google等で“横尾 九州大学”
- 詳細なノートを取る必要はない (講義の内容に集中!)

229

自習方法

- 講義中で紹介したアルゴリズムに関して動作を良く理解し, 使えるようになること
 - 小さな例題で, 手を動かしてトレースする
 - トランプのカード等を使うのも良い
 - 自分で計算機に実装して動かす
 - プログラムする言語は好きなもの/慣れているものを選べば良い
 - 実装自体は簡単 (インタラクションなし)
 - フリーの処理系は多い (GCC/G++, JAVA)

230

ソーティングと順序統計量

入力: n 個の数の列 $\langle a_1, a_2, \dots, a_n \rangle$

出力: $a'_1 \leq a'_2 \leq \dots \leq a'_n$ であるような入力列の置換 $\langle a'_1, a'_2, \dots, a'_n \rangle$

- 実際には, 入力は数だけでなくデータの集合 (レコード) の場合が多い
- レコード = (キー, 付属データ)
- キーの順序にレコードをソートする
- レコード自身でなく, そのポインタを並び替える

231

順序統計量

- n 個の数の集合に対する i 番目の順序統計量 = その集合で i 番目に小さい数
- 入力をソートしてから i 番目を出力 $\Rightarrow O(n \lg n)$ 時間
- 実は, ソートなしで $O(n)$ 時間で求めることができる
 - ソートの後に説明

232

今までのソーティングアルゴリズム

1. 挿入ソート: $O(n^2)$ 時間だが, 入力サイズが小さいときには高速. in-place (入力の配列以外のメモリが不要)
2. マージソート: $\Theta(n \lg n)$ 時間だが, 実行には一時的な配列が必要

233

新しいソーティングアルゴリズム

1. ヒープソート: $O(n \lg n)$ 時間, in-place.
2. クイックソート: 最悪 $O(n^2)$ 時間だが平均実行時間は $\Theta(n \lg n)$. 実用上は高速. in-place.

234

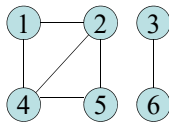
ヒープソート

- $O(n \lg n)$ 時間アルゴリズム, in-place
- ヒープ (heap) と呼ばれるデータ構造を用いる
- ヒープはプライオリティキュー (priority queue) を効率よく実現する

235

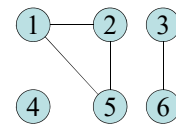
B.4 グラフ

- 無向グラフ (undirected graph)
- $G = (V, E)$
 - V : G の頂点集合 (vertex set)
 - E : G の辺集合 (edge set)



236

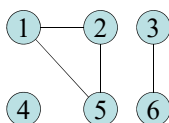
- 辺集合 E は頂点の非順序 (unordered) 集合
- 1つの辺は (u,v) で表現される
 - (u,v) と (v,u) は同一の辺を意味する



237

用語の定義

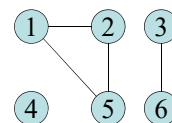
- (u,v) が $G = (V, E)$ の辺であるとき, 頂点 v は頂点 u に隣接 (adjacent) しているという.
 - 無向グラフでは隣接関係は対称的
- 頂点 v の次数 (degree)
 - = v に接続している辺の数



238

経路 (path)

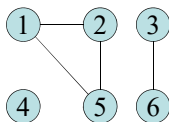
- $G = (V, E)$ の頂点列 $\langle v_0, v_1, v_2, \dots, v_k \rangle$ が頂点 v_0 から v_k までの長さ k の経路とは
 - $(v_{i-1}, v_i) \in E$ ($i=1, 2, \dots, k$)
- 経路の長さ = 経路上の辺の数
- u から v への経路 p が存在するとき, v は経路 p を経由して u から到達可能 (reachable) という



239

閉路 (cycle)

- 経路 $\langle v_0, v_1, v_2, \dots, v_k \rangle$ が閉路であるとは
 - $v_0 = v_k$
 - 少なくとも1つの辺を含む



240

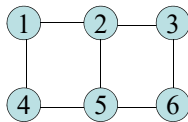
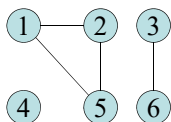
グラフに関する例題 (ラムゼーの定理)

- 人間同士の関係を, 知り合いかそうでないかに分類する.
- AがBの知り合いなら, BはAの知り合いであることを仮定する.
- 任意に選ばれた6人の人に関して, 以下のいずれかが必ず成立する
 - 全員が知り合い同士の3人がいる (6人からうまく3人を選ぶと, お互いに知り合い)
 - 全員が知り合いでない3人がいる (6人からうまく3人を選ぶと, だれも知り合いでない)

241

ラムゼーの定理

- グラフの用語で言い換えると,
 - 6個の頂点を持つ任意の無向グラフに関して, 次のどちらかが必ず成立する
 - 互いに辺で結ばれた3つの頂点が存在 (完全グラフもしくはクリークと呼ばれる)
 - お互いの間に辺が存在しない3つの頂点が存在 (独立集合と呼ばれる)



242

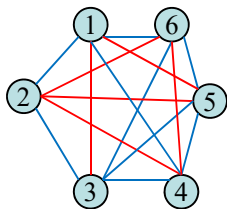
課題

- ラムゼーの定理を証明せよ
- 小学生でも分かる問題の記述:
 - 赤と青の色鉛筆を使って, 六角形を書いて, さらにすべての頂点が結ばれるように線を引きましょう. 線を引くときには, 赤, 青の鉛筆のどちらを使っても構いません. この場合, 赤の線だけの三角形, および青の線だけの三角形が全く作られないようにすることはできません. この理由はなぜでしょうか?

243

ラムゼーの定理の証明

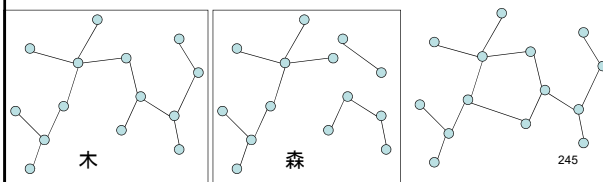
- 知り合い同士を青いエッジ, 知らない同士を赤いエッジで結ぶ.
- あるノードからは, ちょうど5本のエッジが出ている.
- かつ, 青か赤のどちらかは少なくとも3本存在.
- 青が3本以上の場合, その先のノード n_1, n_2, n_3 間に, 少なくとも1つ青があれば, 青のクリークが存在
- そうでなければ, n_1, n_2, n_3 が独立集合 (互いに知り合い同士でない)
- 赤が3本の場合も同様.



244

B.5 木

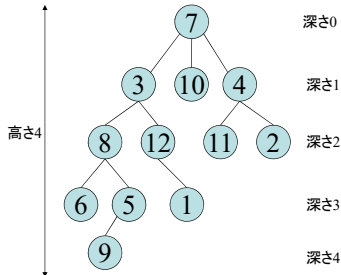
- 木 (tree): 閉路を持たない連結無向グラフ
- 森 (forest): 閉路を持たない無向グラフ (連結でなくてもいい)



245

根付き木

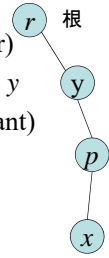
- 根付き木 (rooted tree): 唯一の他と区別される頂点 (根, root) を持つ木



246

先祖, 子孫

- 根付き木 T 上の節点 x の先祖 (ancestor)
 - \Leftrightarrow 根 r から x に至る経路上の任意の節点 y
- y が x の先祖 $\Leftrightarrow x$ が y の子孫 (descendant)
- x を根とする部分木 (subtree rooted at x)
 - $\Leftrightarrow x$ を根とし, x の子孫からなる部分木
- p は x の親 (parent), x は p の子 (child)
- 同一の親を持つ2節点: 兄弟 (sibling)
- 子を持たない節点: 外部節点 (external node) または葉 (leaf)
- 葉でない節点: 内部節点 (internal node)



247

クイズ: この人は誰?

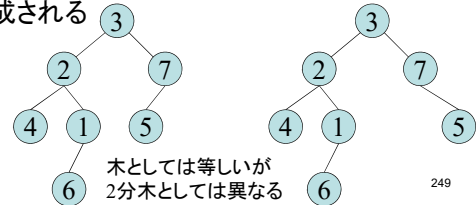
- 僕には兄弟姉妹はいないけど, この人の父親は, 僕のお父さんの子供だ.
- この人は誰?
- 僕を x , x の父親を px , この人を y , この人の父親を py とすると, $py = \text{child}(px) = x$.
- よって $\text{parent}(y) = x$. この人は僕の子供.

248

2分木 (binary tree)

定義: 木 T が2分木とは

- T は節点を全く持っていない (空), または
- T は根, 左部分木 (left subtree) と呼ばれる2分木, 右部分木 (right subtree) と呼ばれる2分木の3つの節点集合 (共通要素を持たない) から構成される



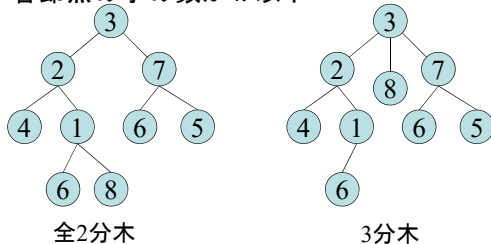
249

- 全2分木 (full binary tree)

\Leftrightarrow 各節点が葉または度数 (子供の数) が2である木

- k 分木 (k -ary tree)

\Leftrightarrow 各節点の子の数が k 以下



全2分木

3分木

250

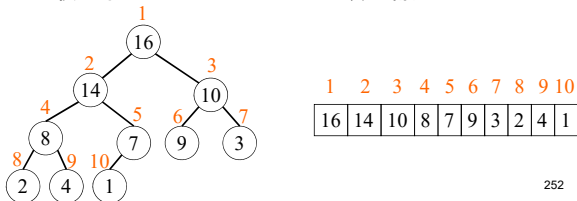
木の重要性

- 大量のデータの整理には ほぼ必ず木構造が用いられる (図書の分類, 住所, yahoo!カテゴリ, etc.)
- 木の深さを d とすると, 木の葉節点の数は $O(2^d)$
- うまく木を使えば, 大量のデータを対数オーダーで処理できる!

251

6.1 ヒープ

- ヒープ: 完全2分木とみなせる配列
- 木の各節点は配列の要素に対応
- 木は最下位レベル以外の全てのレベルの点が完全に詰まっている
- 最下位のレベルは左から順に詰まっている



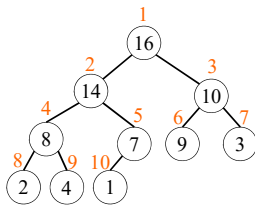
252

ヒープを表すクラス

```
class HEAP {
public:
    int length; // 配列 A に格納できる最大要素数
    int heap_size; // ヒープに格納されている要素の数
    data *A; // 要素を格納する配列へのポインタ
};
```

253

- length: 配列 A に格納できる最大要素数
- heap_size: 格納されているヒープの要素数
- heap_size ≤ length
- 木の根: A[1]
- 節点の添え字が i のとき
 - 親 PARENT(i) = ⌊i/2⌋
 - 左の子 LEFT(i) = 2i
 - 右の子 RIGHT(i) = 2i + 1
- 木の高さは Θ(lg n)



254

ヒープ条件 (Heap Property)

- 根以外の任意の節点 i に対して

$$A[\text{PARENT}(i)] \geq A[i]$$
- つまり, 節点の値はその親の値以下
- ヒープの最大要素は根に格納される

255

ヒープの操作

- HEAPIFY: ヒープ条件を保持する (根節点が子供より大きいとは限らないが, 両側の部分木ではヒープ条件が満たされていることを仮定). $O(\lg n)$
- BUILD_HEAP: 入力の配列からヒープを構成する. 線形時間.
- EXTRACT_MAX: ヒープの最大値を取り除き, 残りがヒープ条件を満たすようにする. $O(\lg n)$ 時間.
- HEAPSORT: 配列をソートする. $O(n \lg n)$ 時間.
 - BUILD_HEAPとEXTRACT_MAXから構成される.
- INSERT: ヒープに値を追加する. $O(\lg n)$ 時間.

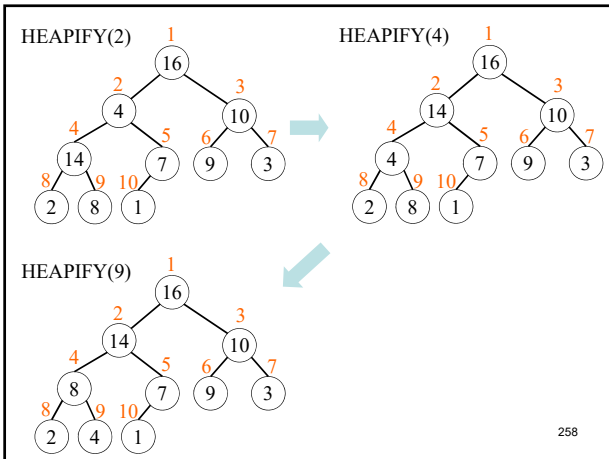
256

6.2 ヒープ条件の保持

- HEAPIFY(i): クラスヒープのメンバ関数, A[i] を根とする部分木がヒープになるようにする. ただし LEFT(i) と RIGHT(i) を根とする2分木はヒープと仮定.

```
void HEAPIFY(int i) {
    int l, r, largest;
    data tmp;
    l = LEFT(i); r = RIGHT(i);
    if (l <= heap_size && A[l] > A[i]) largest = l; // A[i] と左の子で
    else largest = i; // 大きい方を largest に
    if (r <= heap_size && A[r] > A[largest]) // 右の子の方が大きい
        largest = r;
    if (largest != i) {
        tmp = A[i]; A[i] = A[largest]; A[largest] = tmp;
        // A[i] を子供と入れ替える
        HEAPIFY(largest);
    }
```

257



6.3 ヒープの構成

- HEAPIFYでは左右の部分木がヒープである必要がある
- 全体をヒープにするには、2分木の葉の方からヒープにしていけばいい

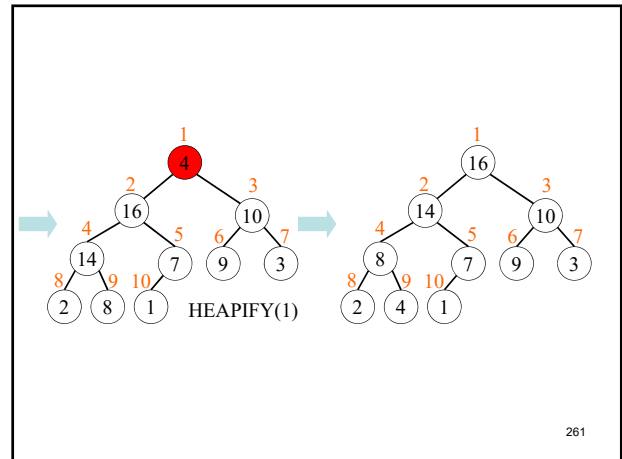
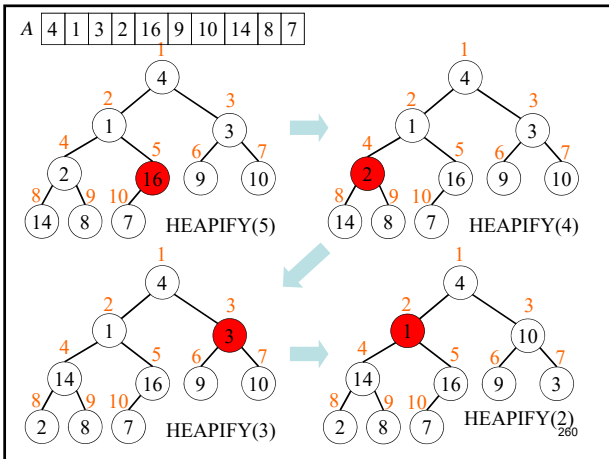
```

void BUILD_HEAP(int n, data D[])
{
    int i;
    heap_size = n;
    A=D;

    for (i = n/2; i >= 1; i--) {
        HEAPIFY(i);
    }
}

```

259



課題

- 配列 A=<1,7,3,6,5,2,4>に対する BUILD_HEAPの動作を示せ

262

HEAPIFYの実行時間

- 節点 i を根とする、サイズ n の部分木に対する HEAPIFY の実行時間 $T(n)$
 - 部分木のサイズは $2n/3$ 以下
 - $T(n) \leq T(2n/3) + \Theta(1)$
 - $T(n) = O(\lg n)$
- 高さ h の節点における HEAPIFY の実行時間は $O(h)$

263

BUILD_HEAPの計算量の解析

- $O(\lg n)$ 時間のHEAPIFYが $O(n)$ 回
 $\Rightarrow O(n \lg n)$ 時間 (注: これはタイトではない)
- $O(n)$ が示せる.

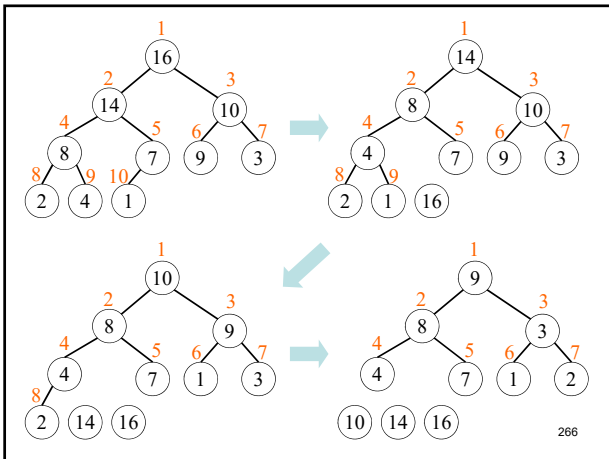
264

6.4 ヒープソート

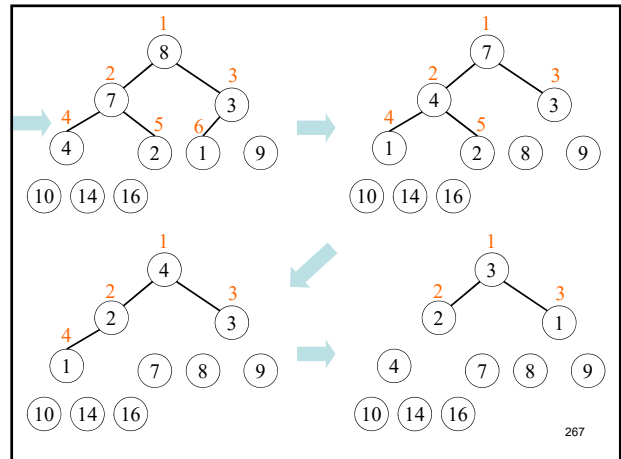
- まずヒープを作る
- すると最大要素が $A[1]$ に入る
- $A[1]$ と $A[n]$ を交換すると, 最大要素が $A[n]$ に入る
- ヒープのサイズを1つ減らしてヒープを維持する

```
void HEAPSORT(int n, data D[])
{int i;
 data tmp;
 BUILD_HEAP(n,D);
 for (i = n; i >= 2; i--) {
 tmp = A[1]; A[1] = A[i]; A[i] = tmp;
 // 根と最後の要素を交換
 heap_size = heap_size - 1;
 HEAPIFY(1);
 }
}
```

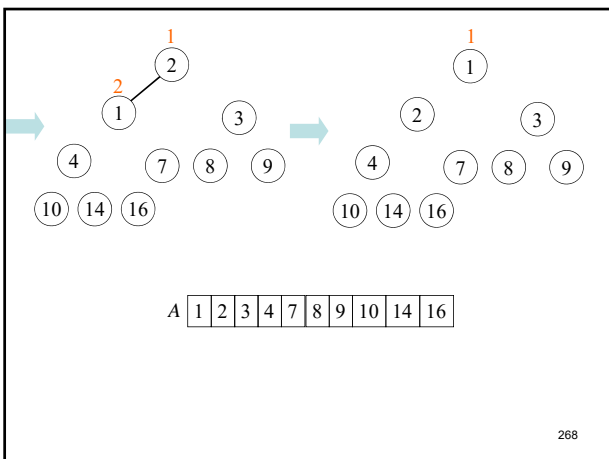
265



266



267



268

課題

- BUILD_HEAPで配列<7,6,4,1,5,2,3>が得られた後のHEAP_SORTの動作を示せ

269

計算量

- BUILD_HEAP: $O(n)$ 時間
 - HEAPIFY: 合計 $O(n \lg n)$
- 全体で $O(n \lg n)$ 時間

270

6.5 プライオリティキュー

- 要素の集合 S を保持するためのデータ構造
- 各要素はキーと呼ばれる値を持つ
- 次の操作をサポートする
 - INSERT(S, x): S に要素 x を追加する
 - MAXIMUM(S): 最大のキーを持つ S の要素を返す
 - EXTRACT_MAX(S): 最大のキーを持つ S の要素を削除し、その値を返す

271

応用: 計算機のジョブ割り当て

- 実行中のジョブと優先順位をプライオリティキューに保持
- ジョブが終了または割り込み発生時に、一時中断しているジョブの中から最大の優先順位のジョブを選び実行 (EXTRACT-MAX)
- 新しいジョブはプライオリティキューに挿入される (INSERT)

272

ヒープによるプライオリティキューの実現

- EXTRACT_MAX(): $A[1]$ を返してHEAPIFY

```
data EXTRACT_MAX() //  $O(\lg n)$  時間
{
    data MAX;
    if (heap_size < 1) {
        cout << "ERROR ヒープのアンダーフロー" << endl;
        exit(1);
    }
    MAX = A[1];
    A[1] = A[heap_size];
    heap_size = heap_size - 1;
    HEAPIFY(1);
    return MAX;
}
```

273

```
void INSERT(data key) //  $O(\lg n)$  時間
{
    int i;

    heap_size = heap_size + 1;
    if (heap_size > length) {
        cout << "ERROR ヒープのオーバーフロー" << endl;
        exit(1);
    }

    i = heap_size;
    while (i > 1 && A[PARENT(i)] < key) {
        A[i] = A[PARENT(i)];
        i = PARENT(i);
    }
    A[i] = key;
}
```

274

課題

- 4, 1, 2, 13, 9, 10の順に、これらの優先度を持つジョブが到達した後のヒープの状態を示せ
- 優先度の高いジョブが二つ処理された後の状態を示せ

275