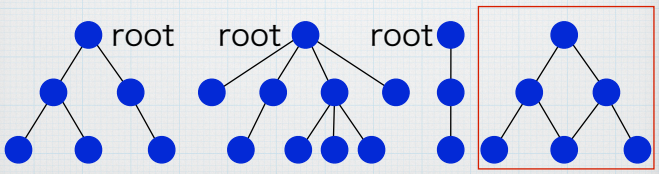
ヒープソート
heap sort

1

木 tree

木 tree =
　いくつかの**節点 node** ＋ それらを結ぶ**枝 branch**
ただし，
- すべての node は 1 つ以上の他の node と branch で結ばれている
- 2 つの node を結ぶ経路 path は 1 通りしかない

ルート root =
　起点（根，頂上）となる node：1 つ決める！



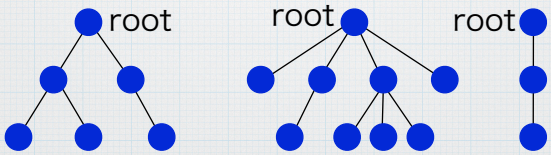
2

木 tree

ある node に対して，その node の
　子＝下に分岐する branch の先のnode
　親＝分岐元の node

葉 leaf＝子を持たない node

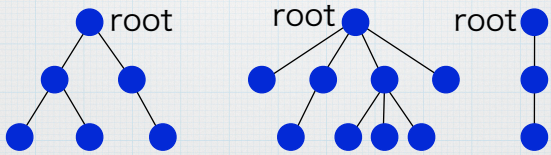
部分木 subtree =
　ある node を root と見なし，その node よりも下（子, 孫, ひ孫,...）の node と branch の集合



3

木 tree

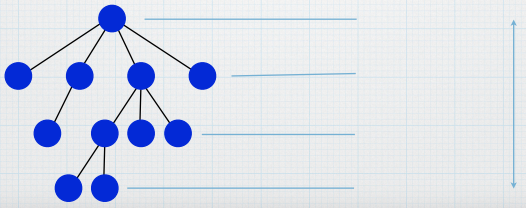
経路 path＝
　2つの node を結ぶ branch の列
経路 path の**長さ length**＝
　path に含まれる branch の個数



4

木 tree

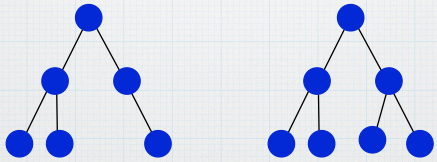
node の **深さ depth**＝
　その node と root とを結ぶ path の length
　（ただし，root の depth は 0 だと思う）
tree の **高さ height**＝
　その木のすべての node の depth の最大値



5

二分木 binary tree

二分木 binary tree =
　各 node から下向きに出る branch が2本以下である tree
高さ m の**完全二分木 complete binary tree** =
　深さ m の leaf が 2^m 個であるような binary tree



6

二分木 binary tree

［注］高さ m の完全二分木の node の総数は，

$1+2+2^2+\ldots+2^m = 2^{m+1}-1$ 個

↓

各 node にデータを置いていくとき，任意の個数のデータを表現できない

↓

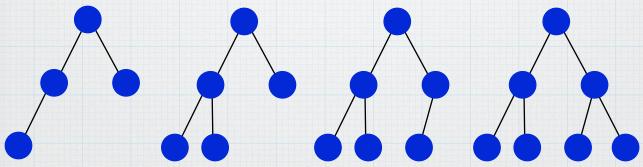
擬完全二分木（この授業だけの仮の名称）を使う

7

擬完全二分木（仮称）

高さ m の**擬完全二分木** ＝

- 深さ m-1 までは完全二分木
- 深さ m では，残りのデータを左から順に詰める（たまたま，完全二分木になることもある。）



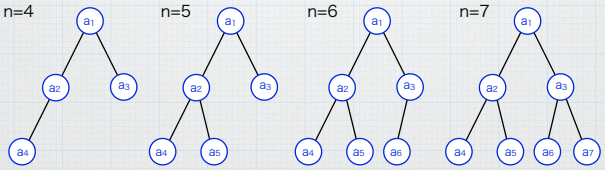
8

擬完全二分木へのデータ配置

n 個のデータ $a_1, a_2, \ldots, a_n$ を，擬完全二分木上に，以下のように配置する：

- a_1 を root に置く
- a_2, a_3 を深さ1の node に左から順に置く
- a_4, a_5, a_6, a_7 を深さ２の node に左から順に置く

（以下同様）



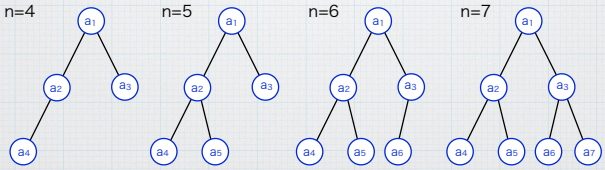
9

データの，木の中での位置

- k 番目のデータ a_k は，擬完全二分木上の，深さ d で，左から $k-(2^d-1)$ 番目に位置する。ただし，d は，$2^d-1 < k \leqq 2^{d+1}-1$ を満たす整数。
- a_k の位置するノードの
  - 左の子は a_{2k}，右の子は a_{2k+1}
  - 親は，$a_{[k/2]}$

課題8-1：これらの性質を証明せよ

- a_k が子を持つ⇔ $2k \leqq n$ （ a_k が左の子のみを持つ⇔ $2k = n$）
- a_k が子を２つ持つ⇔ $2k+1 \leqq n$



10

課題

- 擬完全二分木上の，k 番目のデータの深さ dep(k) および左から数えた位置 pos(k) を与える関数を JS で書け。また，k=1,2,…,100 に対して，dep(k) と pos(k) の値を，

  (0,1) (1,1) (1,2) (2,1) ［以下略］

  のように出力せよ。［ファイル名：8-2.html］
- 擬完全二分木上の，深さ d の，左から数えて p 番目のデータが何番目のデータであるかを与える関数 dnum(d, p) を JS で書け。また，深さ 5 まで，dnum(d, p) の値を

  1

  2 3

  4 5 6 7

  ［以下略］

  のように出力せよ。［ファイル名：8-3.html］

11

課題で使うかもしれない関数

- Math.log(x) = x の自然対数（e=2.71828… を底とする対数）
- Math.floor(x) = x を超えない最大の整数 = [x]
- Math.pow(d,n) = d^n

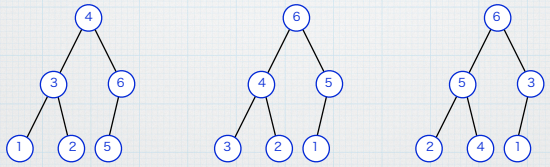
12

ヒープ heap

ヒープ とは，各 node にデータ（node の**キー key** という）を持つ擬完全二分木で，次の**ヒープ条件**を満たすもの：

ヒープ条件：

常に，親のもつデータ ≧ 子のもつデータ



［注］heap ならば，root (=a_1)が最大値

13

課題

- 配列 a[1], a[2], …, a[n] で表現された擬完全二分木がヒープ条件を満たす場合には “yes” を返し，満たさない場合には “no” を返す関数 isheap(a, n) を JS で書け。（option: ヒープ条件を満たさない箇所をすべて出力しつつ判断をすすめよ。）
- さらに，次の配列（擬完全二分木）がヒープ条件を満たすか否かを，isheap を用いて判定せよ。
  - b=[null,10, 9, 4, 8, 7, 2, 3, 5, 6, 1] （b[0] は使わない, n=10）
  - c=[null,10, 3, 9, 4, 7, 6, 8, 5, 2, 1] （c[0] は使わない, n=10）
  - d=[null,いつものデータ] （d[0] は使わない, n=8）

［ファイル名：8-4.html］

14

ヒープソート heap sort

考え方

- $a_1, a_2, \ldots, a_n$ を heap 化する。
- a_1(=root) が $a_1, a_2, \ldots, a_n$ の最大値なので，a_1 と a_n を swap する。→ a_n が最大値となる。
- $a_1, a_2, \ldots, a_{n-1}$ が heap でなくなれば，それを heap 化する。
- a_1(=root) が $a_1, a_2, \ldots, a_{n-1}$ の最大値なので，a_1 と a_{n-1} を swap する。→ a_{n-1} が最大値の次に大きなデータとなる。

（以下繰り返す）

- a_1, a_2 が heap でなくなれば，それを heap 化する。
- $a_1 \geqq a_2$ なので，a_1 と a_2 を swap する。→ $a_1 \leqq a_2$ となる。

↓

- $a_1 \leqq a_2 \leqq \ldots \leqq a_n$ となる。

15

ヒープソート heap sort

考え方をまとめると

```
for(k=n; k≧2; k=k-1){
  a1, a2,..., ak を heap 化する;     ←具体的には？
  swap(a1,ak);
}
```

16

ヒープ化の方法（帰納的）

考え方

$a_1, a_2, \ldots, a_m$ のうち，a_k を root とする subtree を heap 化するには，

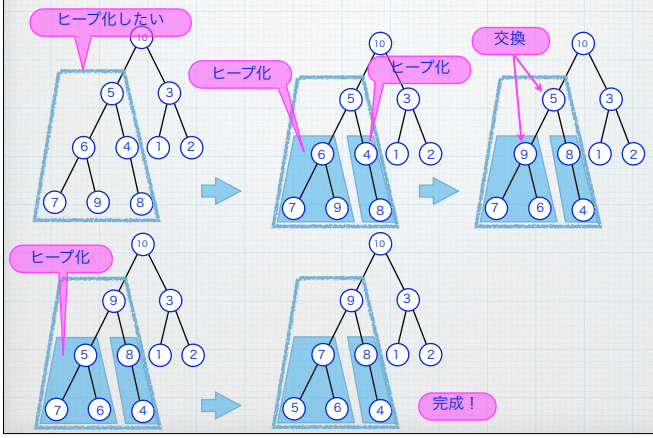
- a_k の左の子を root とする subtree をheap 化する［再帰］
- a_k の右の子を root とする subtree をheap 化する［再帰］
- a_k の左右の子 a_{2k}, a_{2k+1} を較べて，大きい方を a_p（p=2k または p=2k+1）とする
- $a_p \leqq a_k$ ならば，完成！
- $a_p > a_k$ ならば，swap(a_p, a_k) し，新たに a_p を root とする subtree を heap 化する［再帰］

［終了条件］

- a_k が子をもたない → 何もしない。
- a_k が左の子のみを持つ → a_k と左の子を比較し，大きい方を a_k，小さい方を左の子とする。

17

ヒープ化の方法（帰納的）



18

ヒープ化の方法（帰納的）

アルゴリズム 1/2

```
/* a1,..., am のうち，ak を root とする subtree を heap 化 */
make_heap(a, m, k){
  left←2*k;          /* aleft = 「ak の左の子」 */
  right←2*k+1;       /* aright = 「ak の右の子」 */
  /* 終了条件ここから */
  if(left > m){ return; }      /* ak が子を持たない場合 */
  if(left=m){                  /* ak が左の子のみを持つ場合 */
    if(aleft > ak){ swap(aleft, ak); }
    return;
  }
  /* 終了条件ここまで */
```


ヒープ化の方法（帰納的）

アルゴリズム 2/2

```
  /* これ以降，ak が２つの子を持つ場合 */
  make_heap(a, m, left);   /* 左の子の subtree を heap 化 */
  make_heap(a, m, right); /* 右の子の subtree を heap 化 */
  if(aleft>aright){ p←left; } else { p←right; }   /* 下記注１ */
  if(ap ≦ ak){ return; }       /* 子 ≦ 親 ならば終わり */
  swap(ap, ak);                /* 親 ≦ 子 ならば，入れ替え：下記注２ */
  make_heap(a, m, p);      /* 入れ替えによって壊れた heap を修復 */
  return;
}
```

注１：$a_{left} = a_{right}$ の場合，p = left? or p = right?
注２：$a_p = a_k$ のとき，swap する？しない？

ヒープソート（帰納的）

アルゴリズム

```
for(k=n; k≧2; k=k-1){
  make_heap(a, k, 1);
  swap(a1,ak);
}
```

課題8-5：何時ものデータ（誕生日と学籍番号）をヒープソート（帰納的）せよ。

課題

次の操作を JS で書き，ファイルを提出せよ。

1. いつものデータを用意する。
2. a[1], a[2],…, a[8] を出力する。
3. a[1], a[2],…, a[8] を（帰納的）ヒープソートする。（option：実行中に，途中段階を出力せよ。）
4. a[1], a[2],…, a[8] を出力する。

* 提出する際のファイル名＝heap-rec.html

ヒープ化の方法（非帰納的）

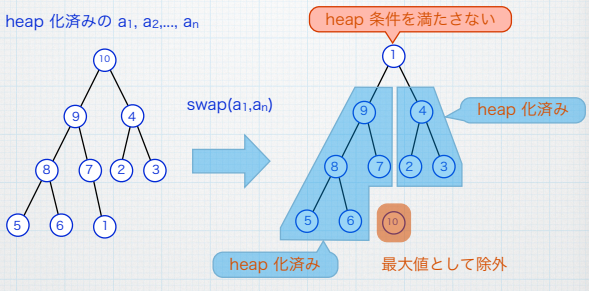
ヒープソートの考え方（復習）

```
for(k=n; k≧2; k=k-1){
    a1, a2,..., ak を heap 化する;
    swap(a1,ak);
}
```

↓

```
a1, a2,..., an を heap 化する;     ← a1, a2,..., an はバラバラ  ←別扱いする！
swap(a1,an);
a1, a2,..., an-1 を heap 化する;   ← a2,..., an-1 はすでにヒープ化されている。
swap(a1,an-1);                      a1 だけがヒープ条件を崩している。
a1, a2,..., an-2 を heap 化する;   ← a2,..., an-2 はすでにヒープ化されている。
swap(a1,an-2);                      a1 だけがヒープ条件を崩している。
    ・・・
a1, a2 を heap 化する;             ← a2 はすでにヒープ化されている。
swap(a1,a2);                        a1 だけがヒープ条件を崩している。
```

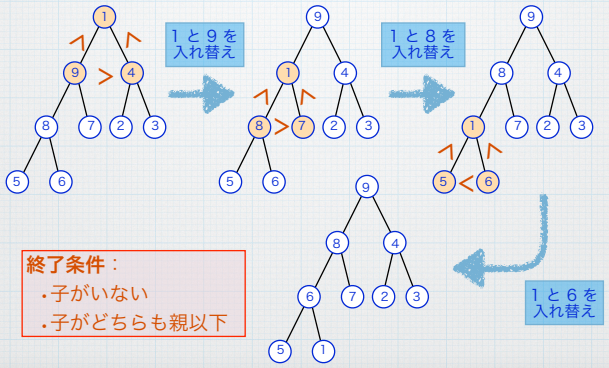

ヒープ化の方法（非帰納的）



このような特殊な状況下ではもっと簡単にヒープ化できるのでは？

ヒープ化の方法（非帰納的）

考え方　『「親 < (どちらかの子)」であれば，親と「大きいほうの」子を入れ替える』という操作を繰り返し，a_1 を下方に送っていく。

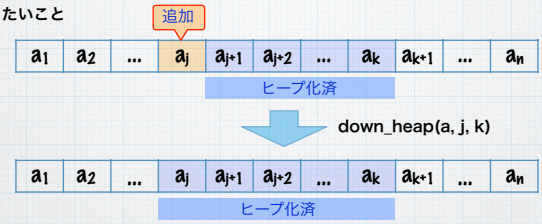


終了条件：
- 子がいない
- 子がどちらも親以下

25

ダウンヒープ down heap

やりたいこと



考え方
- a_j の子のうち，大きい方を a_i とする。
- $a_j < a_i$ ならば，a_i と a_j の位置を交換する。
- これを，子がなくなるか，あるいは，どちらの子も a_j 以下になるまで繰り返す。

2 j > k

26

ダウンヒープ down heap

アルゴリズム

```
down_heap(a, j ,k){
  v ← aj;
  while ( 2 j ≦ k){
    i ← 2 j ;
    if (i < k and ai < ai+1){
      i ← i+1;
    }
    if (v < ai){
      aj ← ai;
      j ← i;
    }
    else {
      break;
    }
  }
  aj ← v;
}
```

説明

27

ヒープの初期化

やりたいこと

最初に与えられた $a_1, a_2, \ldots, a_n$ はバラバラ。
これを down_heap を用いてヒープ化する。

考え方
- $a_1, a_2, \ldots, a_n$ のうち，leaf の部分（$= a_{[n/2]+1}, a_{[n/2]+2}, \ldots, a_n$）は既にヒープ化されていると考えられる。
- そこで，それに $a_{[n/2]}, a_{[n/2]-1}, \ldots, a_1$ を順番に前に付け加えて，付け加える毎に down_heap を行う。

アルゴリズム

```
/* 初期化：a1, a2, ..., an をヒープ化する */
for ( i = [n/2]; i ≧ 1; i = i-1){
    down_heap(a, i, n);
 }
```

28

ヒープソート（非帰納的）

アルゴリズム（非帰納的ヒープソート）

```
/* 初期化：a1, a2, ..., an をヒープ化する */
for ( i = [n/2]; i ≧ 1; i = i-1){
    down_heap(a, i, n);
 }

/* 「a1 を最大値として末尾に移動し，heap を修復」を繰り返す */
for ( k = n; k ≧ 2; k = k-1 ){
    swap( a1, ak );
    down_heap(a, 1, k-1);
 }
```

課題8-6：何時ものデータ（誕生日と学籍番号）をヒープソート（非帰納的）せよ。

29

課題

次の操作を JS で書き，ファイルを提出せよ。

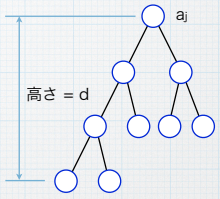
1. いつものデータを用意する。
2. a[1], a[2],…, a[8] を出力する。
3. a[1], a[2],…, a[8] を（非帰納的）ヒープソートする。（option：実行中に，途中段階を出力せよ。）
4. a[1], a[2],…, a[8] を出力する。

* 提出する際のファイル名＝heap-nonrec.html

30

ヒープソート（非帰納的）の計算量

down_heap(a, j, k) の計算量



aj を root とする subtree の高さを d とすると，比較・swap 共に，それぞれ高々 d 回しか起こらない。

よって，down_heap の計算量は，最大でも O(d)。

31

ヒープソート（非帰納的）の計算量

簡単のため，完全二分木を考え，$n = 2^{d+1}-1$ とする。
（このとき木の高さは d。また，$d = \log_2 (n+1) - 1 = O(\log_2 n)$。）

初期化：$a_1, a_2, \ldots, a_n$ のヒープ化の計算量

```
for ( i = [n/2]; i ≧ 1; i = i-1){
    down_heap(a, i, n);
}
```

a_i の深さ	i の個数	a_i を root とする subtree の高さ	計算量のオーダー
d-1	2^{d-1}	1	$2^{d-1}\times 1$
d-2	2^{d-2}	2	$2^{d-2}\times 2$
・・・	・・・	・・・	・・・
2	2^2	d-2	$2^2\times(d-2)$
1	2	d-1	$2\times(d-1)$
0	1	d	$1\times d$

32

ヒープソート（非帰納的）の計算量

初期化：$a_1, a_2, \ldots, a_n$ のヒープ化の計算量（続き）

よって，その計算量は，

$$\begin{aligned}\sum_{k=1}^{d} 2^{d-k}k &= \sum_{k=1}^{d} 2^d 2^{-k}k \\ &= 2^d \sum_{k=1}^{d} k\left(\frac{1}{2}\right)^k \\ &= 2^{d+1} - d - 2 \\ &< 2^{d+1} - 1 \\ &= n\end{aligned}$$

故に，初期化：$a_1, a_2, \ldots, a_n$ のヒープ化の計算量のオーダーは O(n)

33

ヒープソート（非帰納的）の計算量

「『a_1 を最大値として末尾に移動し，heap を修復』を繰り返す」の計算量

```
for ( k = n; k ≧ 2; k = k-1 ){
    swap( a1, ak );
    down_heap(a, 1, k-1);
}
```

「swap(a_1,a_k)」の計算量の合計
n-1 回起こりうるので，O(n)

「down_heap(a,1,k-1)」の計算量の合計（概数）

a_k の深さ	k の個数	a_1 を root とする tree の高さ	計算量のオーダー
d	2^d	d	$2^d\times d$
d-1	2^{d-1}	d-1	$2^{d-1}\times(d-1)$
d-2	2^{d-2}	d-2	$2^{d-2}\times(d-2)$
・・・	・・・	・・・	・・・
2	2^2	2	$2^2\times 2$
1	2	1	2×1

34

ヒープソート（非帰納的）の計算量

「down_heap(a,1,k-1)」の計算量の合計（続き）

よって，その計算量は，

$$\begin{aligned}\sum_{k=1}^{d} 2^k k &= (d-1)2^{d+1} + 2 \\ &< d(2^{d+1} - 1) \\ &= d\,n \\ &= O(n \log n)\end{aligned}$$

以上，まとめて，「『a_1 を最大値として末尾に移動し，heap を修復』を繰り返す」の計算量のオーダーは，O(n) + O(n log n) = O(n log n)

以上，すべて纏めると，ヒープソート（非帰納的）の計算量のオーダーは，

O(n) + O(n log n) = O(n log n)

35