アルゴリズム・データ構造1 第15回 まとめ

名城大学理工学部情報工学科
山本修身

この講義で扱った内容

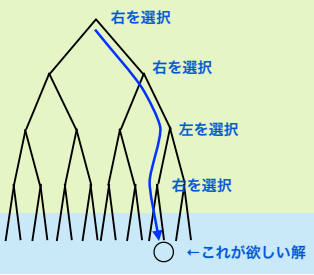
青字は特に重要な項目

- 二分法（バイナリーサーチ）
- ニュートン法
- 挟み撃ち法
- 木の探索—深さ優先探索
- 木の探索—幅優先探索
- ハッシング—チェイン法
- ハッシング—オープンアドレッシング
- ソーティング—セレクションソート
- ソーティング—クイックソート
- ソーティング—マージソート
- ソーティング—プライオリティーキューとヒープソート
- ソーティング—インサーションソート
- 計算量—オーダによる計算量の表現
- 計算不能な問題，NP完全問題，NP困難な問題

２分法について

- 以下のような木構造（これを2分木という）があるとする．この木構造において，それぞれのレベルで2つうちのどちらかを選択すると，最終的に欲しい解に辿り着くことができる場合，この探索方法を**2分探索法（バイナリーサーチ：binary search）**と呼ぶ．

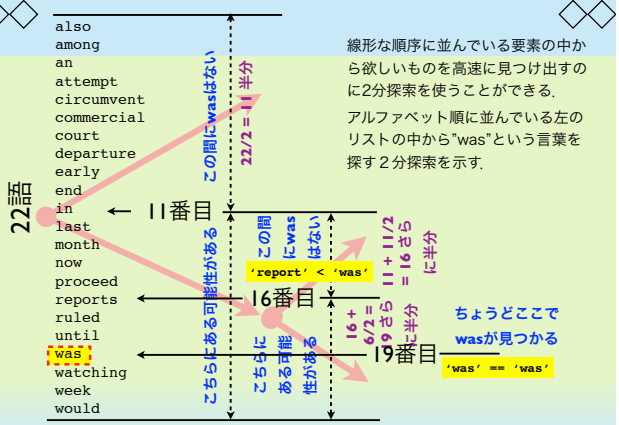
２分法はソートされているデータから効率的に目的のデータを見つけ出す方法である．



2分探索の例（1）

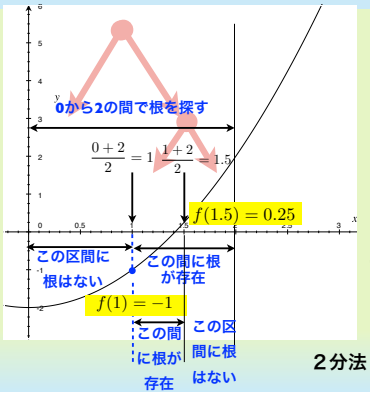
線形な順序に並んでいる要素の中から欲しいものを高速に見つけ出すのに2分探索を使うことができる．

アルファベット順に並んでいる左のリストの中から"was"という言葉を探す２分探索を示す．



２分探索の例（2）

- ２の平方根を計算してみる． $f(x) = x^2 - 2$ の根を求めれば良い



始めの状態：[0.0--2.0]
↓
次の状態：[1.0--2.0]
↓
その次の状態：[1.0-1.5]
↓
⋮

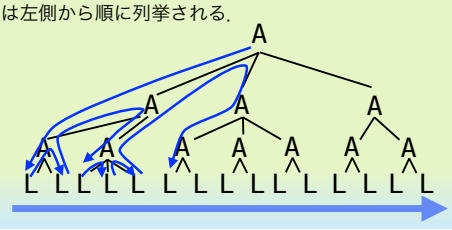
真の答えに近づく

近似解

２分法 （bisection method）

深さ優先探索

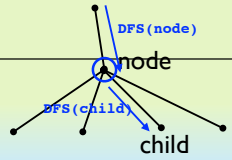
- 深さ優先探索は葉に出会うまで，深く進み，深く進めなくなったら，一段階もどって，別の方向に深く進む．深く進むことを優先する探索法である．
- もし，木が無限に深い場合には止まらなくなる．したがって，この方法は使えない．
- 葉は左側から順に列挙される．



与えられた木のノードを順に走査する方法：深さ優先探索 (4)

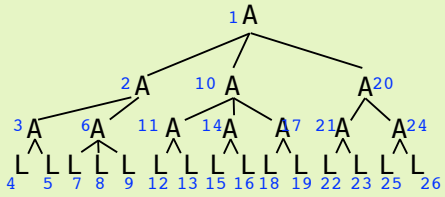
- 深さ優先探索を行うためのプログラム

```
function DFS(node){
    nodeについての処理;
    for (nodeのすべての子供childについて) {
        DFS(child)
    }
}
```



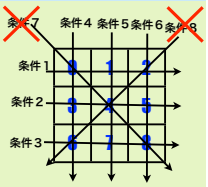
与えられた木のノードを順に走査する方法：深さ優先探索 (3)

- 葉ではないノードの順位も含めて以下のような順番で探索が行われる．



第８回 課題１

一般の**魔方陣 (magic square)** は右図のように縦，横，斜めの和がすべて等しいという条件の成り立つ0から8までの数の並びである．このパズルを変形して，右図の斜めの条件を除外して，縦および横の条件が成り立っている並びのことを**準魔方陣（semi-magic square)**と呼ぶ．4x4の準魔方陣（すなわち，0から15までの数のならび）のうち，左上の数がnであるものの総数を返す関数 kadai1(n) を作れ．



ヒント： なるべく早い段階で枝刈りをするようなプログラムにすると，少ない計算量で数えることができる．

課題1解答例 (1)


```
function kadai(n){
    function check(pat, i, j, k, l){
        var sum = pat[i] + pat[j] + pat[k] + pat[l]
        if (sum != 30) return true
        else return false
    }
    function magic(pat, rest){
        var llen = pat.length
        if (llen >= 16) return 1
        if (llen == 4 && check(pat, 0, 1, 2, 3)) return 0
        if (llen == 8 && check(pat, 4, 5, 6, 7)) return 0
        if (llen == 12 && check(pat, 8, 9, 10, 11)) return 0
        if (llen == 13 && check(pat, 0, 4, 8, 12)) return 0
        if (llen == 14 && check(pat, 1, 5, 9, 13)) return 0
        if (llen == 15 && check(pat, 2, 6, 10, 14)) return 0
        var sum = 0
        for (var i = 0; i < rest.length; i++){
            var ppat = pat.slice(0)
            var restx = rest.slice(0)
            ppat.push(rest[i])
            restx.splice(i, 1)
            sum += magic(ppat, restx)
        }
        return sum
    }
    var rest = []
    for (var i = 0; i < 16; i++)
        if (i != n) rest.push(i)
    return magic([n], rest)
}
```

例題をそのままプログラムにしたもの．時間がかなりかかることになる．

以下のように実行する

```
puts(kadai(0))
```

↓

34344

引数に0から15のどれをいれても同じ値になる

課題1解答例 (2)


```
function kadai(n){
    var pat = new Array(16)
    var used = new Array(16)
    function check(pat, i, j, k, l){
        var sum = pat[i] + pat[j] + pat[k] + pat[l]
        if (sum != 30) return true
        else return false
    }
    function magic(i){
        if (i >= 16) return 1
        if (i == 4 && check(pat, 0, 1, 2, 3)) return 0
        if (i == 8 && check(pat, 4, 5, 6, 7)) return 0
        if (i == 12 && check(pat, 8, 9, 10, 11)) return 0
        if (i == 13 && check(pat, 0, 4, 8, 12)) return 0
        if (i == 14 && check(pat, 1, 5, 9, 13)) return 0
        if (i == 15 && check(pat, 2, 6, 10, 14)) return 0
        var sum = 0
        for (var k = 0; k < 16; k++){
            if (used[k] === false){
                used[k] = true
                pat[i] = k
                sum += magic(i + 1)
                used[k] = false
            }
        }
        return sum
    }
```

```
    for (var i = 0; i < 16; i++){
        used[i] = false
    }
    used[n] = true
    pat[0] = n
    return magic(1)
}
```

magicが配列を渡しながら再帰するのは無駄なので，配列は外部で定義しておく．特に現在どの数が使われているのかを配列usedに覚えさせて，適宜その値を変える．magic(i)はpatの0からi - 1番目までが決まって，iつぎに決定するという意味．

課題1解答例 (3)

そもそも束縛が存在することから決めるべき数値は左上の3x3の領域（青い部分）の値である．その部分の値が決まれば，それ以外の場所は自動的にあたいが決まる．xの場所の値が決まると青以外の場所も一部決定される．

0	1	2x	
3	4	5x	
6x	7x	8x	

0の位置は外部から指定されるので，他の8箇所の値を決めればすべて決まることになる．これによって木構造がかなり単純化される．ただし，用いる値は0〜15であり，"x"のついている場所については，2つ以上の数が使用済になる．

課題1解答例 (4)

プログラムは多少複雑になる．

```
function kadai(n){
    var pat = new Array(9)
    var used = new Array(16)
    function idx2(i){
     if (i == 2) return [0, 1]
     else if (i == 5) return [3, 4]
     else if (i == 6) return [0, 3]
     else if (i == 7) return [1, 4]
    }
    function magic(i){
     if (i == 0 || i == 1 || i == 3 || i == 4){
         var sum = 0
         for (var k = 0; k < 16; k++){
          if (used[k] === false){
              used[k] = true
              pat[i] = k
              sum += magic(i + 1)
              used[k] = false
          }
         }
         return sum
     } else if (i == 2 || i == 5 || i == 6 || i == 7 ){
         var [ii1 , ii2] = idx2(i)
         var d = 30 - (pat[ii1] + pat[ii2])
         var sum = 0
         var [st, end] = [d - 15, 15]
         if (d <= 15 && d >= 0)
             [st, end] = [0, d]
         for (var k = st; k <= end; k++){
             var el = d - k
             if ((k != el) && (used[k] === false) &&
                 (used[el] === false)){
                 used[k] = true;
                 used[el] = true;
                 pat[i] = k
                 sum += magic(i + 1)
                 used[k] = false
                 used[el] = false
             }
         }
         return sum
    }
```


課題1解答例 (5)

ブラウザ上で実行すると1秒かからないで結果がでる．

```
var t1 = new Date()
var res = kadai(0)
var t2 = new Date()
puts(res)
puts("Time = " + (t2 - t1) + " ms")
```

```
34344
Time = 314 ms
```

```
} else if (i == 8){
    var sum = 0
    for (var k = 0; k < 16; k++){
        if (used[k] === true) continue
        used[k] = true
        var dd1 = 30 - (pat[2] + pat[5] + k)
        var dd2 = 30 - (pat[6] + pat[7] + k)
        if (dd1 < 0 || dd2 < 0 || dd1 == dd2 || k == dd1 || k == dd2
            || dd1 > 15 || dd2 > 15){
            used[k] = false
            continue
        }
        if (used[dd1] === true){
            used[k] = false
            continue
        }
        used[dd1] = true
        if (used[dd2] === true){
            used[k] = false
            used[dd1] = false
            continue
        }
                used[dd2] = true
                sum += 1
                used[dd1] = false
                used[dd2] = false
                used[k] = false
            }
            return sum
        }
    }
    for (var i = 0; i  < 16; i++)
        used[i] = false
    used[n] = true
    pat[0] = n
    return magic(1)
}
```


第８回 課題２

3組の夫婦がいる．この3組の夫婦は今ボートを使って川の左岸から右岸へ渡りたい．このボートは二人乗りで３人以上は乗れない．さらに複雑な事情として，3組の夫婦のそれぞれの夫は非常に嫉妬深く，自分がいないときに他の夫と自分の妻が一緒に居ることを許さない．このような状況で全員が左岸から右岸に渡るにはどうしたら良いか．ただし，ボートは無人で移動できないとする．幅優先探索を用いて最も回数の少ない渡り方を返す関数 kadai2() を作れ．ただし，返す値は， 夫1, 妻1，夫2, 妻2，夫3, 妻3をそれぞれ0, 1, 2, 3, 4, 5 として，それぞれの時点でボートに乗るメンバーのリストとして，

kadai2() = [[1, 3], [1], [0, 2], ...]

のように返せ．答は複数存在する可能性があるが，そのうちの一つを返せば良い．もちろんボートが無人で移動することはない．

課題２解答例 (1)

```
function kadai2(){
    var init_state = [0, 1, 1, 1, 1, 1, 1]
    var final_state = [1, 0, 0, 0, 0, 0, 0]
    var move_list = [[0], [1], [2], [3], [4], [5],
        [0, 1], [0, 2], [0, 4],
        [1, 3], [1, 5],
        [2, 3], [2, 4],
        [3, 5],
        [4, 5]]
    var tabulist = []
    function move_state(state, move){
        state1 = state.slice(0)
        var [m1, m2] = [1, 0]
        if (state[0] == 1) [m1, m2] = [0, 1]
        for (var i = 0; i < move.length; i++)
            if (state[move[i] + 1] != m1) return null
        for (var i = 0; i < move.length; i++)
            state1[move[i] + 1] = m2
        state1[0] = m1
        return state1
    }
    function eq_state(state1, state2){
        for (var i = 0; i < 7; i++)
            if (state1[i] != state2[i]) return false
        return true
    }
```

状態は６人が舟がどちらの岸にあるか（左岸：0, 右岸: 1）とそれぞれ人々がどちらの岸にいるか（左岸が1で右岸が0）で表現できる．

夫婦1：夫 (0)，妻 (1)
夫婦2：夫 (2)，妻 (3)
夫婦3：夫 (4)，妻 (5)

課題２解答例 (2)

```
    function add_to_tabulist(state){
        for (var i = 0; i < tabulist.length; i++){
            if (eq_state(state, tabulist[i])) return false
        }
        tabulist.push(state)
        return true
    }
    function check_state(state){
        if (state[1] != state[2]){
            if (state[2] == state[3] || state[2] == state[5]) return false
        }
        if (state[3] != state[4]){
            if (state[4] == state[1] || state[4] == state[5]) return false
        }
        if (state[5] != state[6]){
            if (state[6] == state[1] || state[6] == state[3]) return false
        }
        return true
    }
```


課題２解答例 (3)

最後に幅優先探索で問題を解く．11ステップで解くことができる．

```
add_to_tabulist(init_state)
var queue = [[init_state, null, null]]
while (queue.length > 0){
    var node = queue.shift()
    var [state, parent, mv] = node
    if (eq_state(state, final_state)) break
    for (var i = 0; i < move_list.length; i++){
        var state1 = move_state(state, move_list[i])
        if (state1 == null) continue
        if (! check_state(state1)) continue
        if (! add_to_tabulist(state1)) continue
        queue.push([state1, node, move_list[i]])
    }
}
var res = []
while (node != null){
    var [state, parent, mv] = node
    res.unshift(mv)
    node = parent
}
res.shift()
return res
}
```

```
res = kadai2()
for (var i = 0; i < res.length; i++)
  puts(res[i])
```

```
0,1 夫妻１で右岸へ
0   夫１が左岸へ戻る
3,5 妻１と妻３が右岸へ
1   妻１が左岸へ戻る
2,4 夫２と夫４が右岸へ
2,3 夫婦２が左岸へ戻る
0,2 夫１と夫２が右岸へ
5   妻３が左岸へ戻る
1,3 妻１と妻２が右岸へ
1   妻１が左岸へ戻る
1,5 妻１と妻３が右岸へ
```

夫婦1：夫 (0)，妻 (1)
夫婦2：夫 (2)，妻 (3)
夫婦3：夫 (4)，妻 (5)

課題２解答例 (4)

① ① ② ② ③ ③		
② ② ③ ③	→	① ①
① ② ② ③ ③	←	①
① ② ③	→	① ② ③
① ① ② ③	←	② ③
① ①	→	② ② ③ ③
① ① ② ②	←	③ ③
① ②	→	① ② ③ ③
① ② ③	←	① ② ③
③	→	① ① ② ② ③
① ③	←	① ② ② ③
	→	① ① ② ② ③ ③

チェイン法 (1)

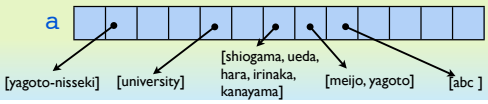
- この問題点はテーブルにそのまま要素を書くのではなくて，その中の配列に書くことによって解決する．

```
var strs = ["meijo", "university", "abc",
    "shiogama","yagoto", "ueda", "hara",
    "irinaka", "yagoto-nisseki", "kanayama"]

function make_table2(){
   var a = []
   for (var i = 0; i < strs.length; i++){
     var v = h(strs[i])
     if (a[v] === undefined) a[v] = [strs[i]]
     else a[v].push(strs[i])
   }
   return a
}
puts(make_table2().join("|"))
```

|yagoto-nisseki|||university||shiogama,ueda,hara,irinaka,kanayama|meijo,yagoto|abc

meijo : 7
university : 4
abc : 8
shiogama : 6
yagoto : 7
ueda : 6
hara : 6
irinaka : 6
yagoto-nisseki : l
kanayama : 6



チェイン法 (2)

```
function search(s){
   var v = h(s)
   if (a[v] === undefined)
     puts(s + " is not in the table.")
   else {
      var lst = a[v]
      for (var i = 0; i < lst.length; i++){
        if (lst[i] == s){
           puts(s + " is in the table")
           return;
        }
      }
      puts(s + " is not in the table.")
   }
}
var a = make_table2()
search("shiogama")
search("irinaka")
search("motoyama")
```

セパレートチェイニング法の場合には，ハッシュ値を計算してから，さらにリストの中身を探さないといけない場合がある．その分時間がかかる．

shiogama is in the table
irinaka is in the table
motoyama is not in the table.

オープンアドレッシング (1)

- チェイン法と並んでもう１つのハッシュアルゴリズムとしてオープンアドレッシング法ある．
- オープンアドレッシング法では衝突が起こった場合，適当なアルゴリズムによって別の場所にデータを入れる．

```
function add_data(table, data){
   var v = h(data)
   while (table[v] !== undefined){
     v = (v + 1) % 31
   }
   table[v] = data
}
```

線形探索

table

```
0-->
1-->
2-->meijo
3-->
4-->
5-->ueda
6-->yagoto-nisseki
7-->abc
8-->yagoto
9-->hara
10-->
11-->university
12-->kanayama
13-->
14-->
15-->
16-->shiogama
17-->
18-->
19-->
20-->
21-->
22-->
23-->irinaka
```

この方法はクラスターが出来易い

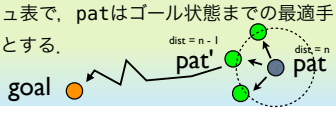
主クラスター化

meijo : 2
university : 11
abc : 7
shiogama : 16
yagoto : 7
ueda : 5
hara : 8
irinaka : 23
yagoto-nisseki : 6
kanayama : 11

衝突

第11回の課題

本日解説したハッシュ表を用いた8パズルの探索で，任意の状態patからゴール状態へ至る最適な手順をハッシュ表を用いて計算するプログラムを作れ．ハッシュ表を用いればあるパターンpatから１ステップで移動可能なパターンpat'でゴール状態までの距離が１少なくなるものが必ず存在する．そのような状態をゴール状態に達するまで順に調べていけば，最適な動かし方がわかる．kadai(hash_table, pat) がコマの移動を表す文字列 "rulull..." を返すようにせよ．ただし，hash_tableは講義で説明したハッシュ表で，patはゴール状態までの最適手順が知りたいパターンとする．



解答例 (1)

これまで説明した関数を組み合わせて用いれば作ることができる．手順は問題で示した通りである．

1.与えられた状態sから最終状態Fまでの最適ステップ数dを調べる．

2.一つ動かしてみてステップ d − 1になる方向を探す（これは必ずあるはず）．

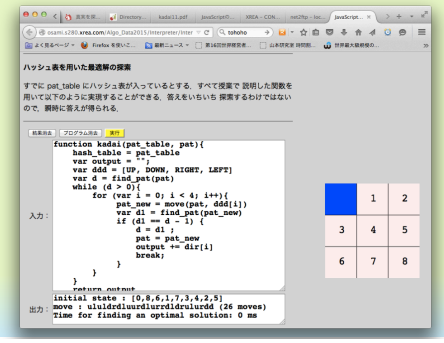
3.そちらの方向にsを移動させ，それを記録する．

```
function kadai(pat_table, pat){
    hash_table = pat_table
    var output = "";
    var ddd = [UP, DOWN, RIGHT, LEFT]
    var d = find_pat(pat)
    while (d > 0){
        for (var i = 0; i < 4; i++){
            pat_new = move(pat, ddd[i])
            var d1 = find_pat(pat_new)
            if (d1 == d - 1) {
                d = d1 ;
                pat = pat_new
                output += dir[i]
                break;
            }
        }
    }
    return output
}
```

解答例 (2)

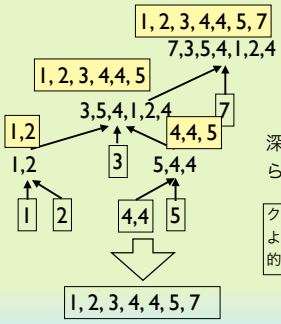
実際に8パズルの実行環境（ハッシュ表の計算結果を付加したもの）の上で実行した結果を以下に示す．探索時間は0〜1ms程度（おそらく多くの場合1ms未満）となっている．

http://osami.s280.xrea.com/Algo_Data2015/Interpreter/Interpreter5-hash.html



クイックソートアルゴリズム (1)

- 以前示したように，与えられた配列を分割して木構造を作りながらソートするアルゴリズムがクイックソートである．



深さ優先探索で葉を探索して得られる順に列挙すれば良い．

クイックソートではキーの選択方法によっていくつかの種類が存在するが本質的ではない

クイックソートアルゴリズム (2)


```
function qsort(lst){
   if (lst.length <= 1) return lst
   var key = lst[0]
   var lt = []
   var eq = []
   var gt = []
   for (var i = 0; i < lst.length; i++){
     var ele = lst[i]
     if (ele < key) lt.push(ele)
     else if (ele > key) gt.push(ele)
     else eq.push(ele)
   }
   return qsort(lt).concat(eq).concat(qsort(gt))
}

m = [7,3,5,4,1,2,4]
puts(m)
puts(qsort(m))
```

プログラムは再帰を用いて左のようになる．

プログラムを実行すると以下のようになる．

```
7,3,5,4,1,2,4
1,2,3,4,4,5,7
```

クイックソートアルゴリズム (3)

このプログラムの実行時間を測ってみる．

```
lst = []
for (var i = 0; i < 10000; i++)
   lst.push(Math.random())
t1 = new Date()
res = qsort(lst)
t2 = new Date()
puts((t2 - t1) + "ms")
puts(res[0] + ":" + res[5000] + ":" + res[9999])
```

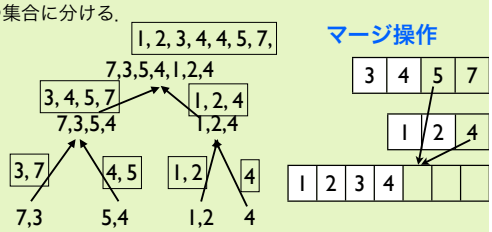
プログラムを実行すると以下のようになる．

```
14ms
0.00007199321912798595:0.5064526987194872:0.9998958819354733
```

セレクションソートの約500倍速くソートできた．なぜ，こんなに速いのかは，次回の講義で述べる

マージソート (1)

もう一つの深さ優先探索を用いたソートアルゴリズムとしてマージソートがある．クイックソートはキーよりも大きな集合，キーに等しい集合，キーよりも小さな集合に分けたが，マージソートはほぼ大きさの等しい2つ集合に分ける．



半分ずつに分解した後，木を遡るときに「マージ」を行う．マージは２つの小さなソート列から１つの大きなソート列を作る作業である．

マージソート (2)


```
function merge(lst1, lst2){
  var lst = []
  while (true){
    if (lst1.length == 0) return lst.concat(lst2)
    else if (lst2.length == 0) return lst.concat(lst1)
    if (lst1[0] < lst2[0]) lst.push(lst1.shift())
    else lst.push(lst2.shift())
  }
}

function msort(lst){
  var n = lst.length
  if (n < 2) return lst
  else {
    var n2 = Math.floor(n / 2)
    return merge(
        msort(lst.slice(0, n2)), msort(lst.slice(n2, n)))
  }
}
```

マージソートのプログラムは以下のとおり．

```
7,3,5,4,1,2,4
1,2,3,4,4,5,7
```

```
m = [7,3,5,4,1,2,4]
puts(m)
puts(msort(m))
```

マージソート (3)

- マージソートの実行時間を測ってみる.

```
lst = []
for (var i = 0; i < 10000; i++)
   lst.push(Math.random())
t1 = new Date()
res = msort(lst)
t2 = new Date()
puts((t2 - t1) + "ms")
puts(res[0] + ":" + res[5000] + ":" + res[9999])
```

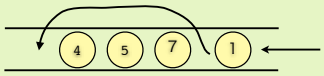
- 上のプログラムを実行すると以下のようになる.

```
59ms
0.00000194169570333579:0.49702765282789085:0.9999521926597846
```

クイックソートに比べると遅いが，セレクションソートの約120倍速くソートできた.

プライオリティーキュー (1)

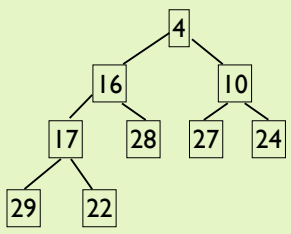
ここでソーティングから少々離れて，特殊なキューの構造について考えてみる．プライオリティーキュー（優先順位付きキュー）はデータを取り出すとき，最初に入れられたものではなく，一番小さな値を持つものが常に取り出される.

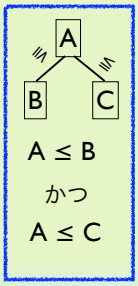


このようなキューを実現するデータ構造はフィボナッチヒープなどいくつか知られている．ここでは，２分ヒープによる実現を示す.

プライオリティーキュー (2)

- ２分ヒープは，２分木構造をしている．この２分木では，親ノードの値が必ず子供のノードよりも小さい.

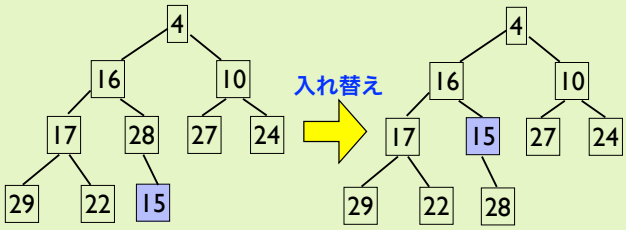




木のノードはいつも上から詰まって，一番末端は左から詰まっていくとする.

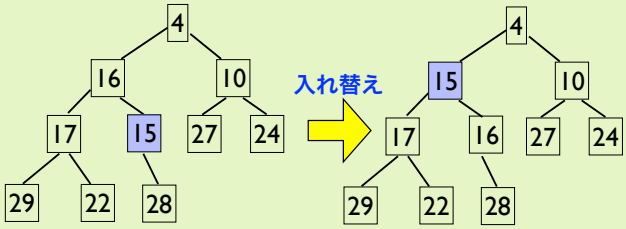
プライオリティーキュー (3)

データの挿入：この性質を保ちつつデータを付加するには，一番したの枝に要素を入れてから入れ替えによって，この性質が成り立つよう変形する.



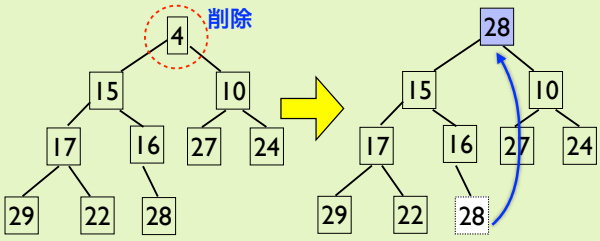
プライオリティーキュー (4)

データの挿入： 16と15の大小関係に問題があるので，さらに入れ替える．最大で根 (root) まで入れ替え作業を行えば，矛盾は解消する



プライオリティーキュー (5)

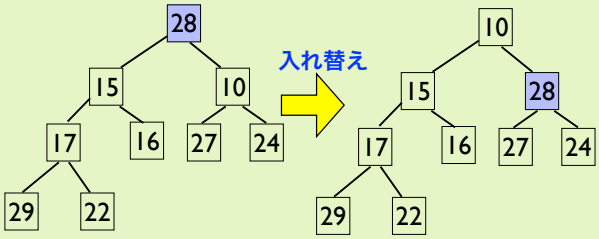
最小データの削除：一番値の小さなデータを削除するには，根の要素を削除すればよい（それが最小であることは明らか）.



削除しただけでは穴があいてしまうので最後の要素を根に持って行く

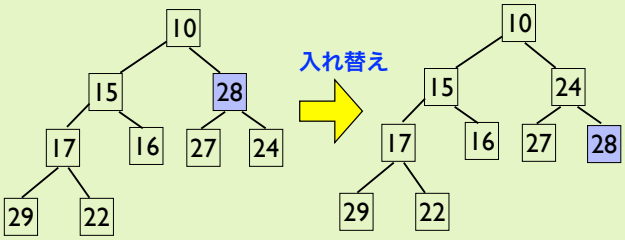
プライオリティーキュー (6)

- キューの条件を満たすために入れ替えを行う



プライオリティーキュー (7)

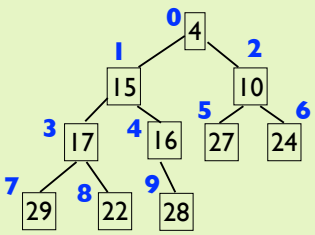
さらに入れ替えを行って，全体的に条件が満たされるようにする．



プライオリティーキュー (6)

- ２分ヒープは配列によって表現することができる

ノード i の親ノードは，Math.floor((i - 1) / 2)で計算することができる．子供のノードは，2i + 1と2i + 2になっている．



配列による表現

0	1	2	3	4	5	6	7	8	9
4	15	10	17	16	27	24	29	22	28

プライオリティーキューのプログラム (4)

２分ヒープによるプライオリティーキューを使うとソートができる．これをヒープソート (heap sort) と呼ぶ．かなり速くソートできる．

```
function heap_sort(lst){
  heap = []
  for (var i = 0; i < lst.length; i++)
   add_element(lst[i])
  var ans = []
  while (heap.length > 0){
    ans.push(delete_element())
  }
  return ans
}
```

```
0.0000421288169178968 3:0.9999583648667706
time = 26 ms
```

```
lst = []
for (var i = 0; i < 10000; i++)
  lst.push(Math.random())
t1 = new Date()
ans = heap_sort(lst)
t2 = new Date()
puts(ans[0] + ":" + ans[lst.length - 1])
puts("time = " + (t2 -t1) + " ms")
```

期末試験に向けての注意

- ここで採り上げたデータ構造やアルゴリズムを良く理解しておくこと．
- アルゴリズムやデータ構造と実際のプログラムとの対応がわかるようにしておくこと．
- **オーダの記法の意味を理解しておくこと．**
- 木探索の方法について理解しておくこと
- **2分ヒープ（プライオリティキュー）の管理方法と表現方法について理解しておくこと．**
- **ハッシュテーブルの管理方法について理解しておくこと．**
- いくつかのソートアルゴリズムについて理解しておくこと．

アンケートの自由設定項目

- 設問１３： JavaScriptのプログラミングは難しかった
- 設問１４： JavaScriptに限らずプログラミングに興味を持っている．