10　いくつかの補足

”Combinatorial Optimization”を読む上で知っておいた方が良いが教科書では省略されている内容をこの章に記す．

10.1　ヒープ

ヒープは木を使ったデータ構造の一種であり，最小値（もしくは最大値）を高速に求められるという特徴がある．ヒープにもいくつかの種類があるが，ここでは二分ヒープについて説明する．

定義 10.1 二分木（binary tree）
グラフ理論において，根付き木であって各頂点について子が 2 個以下であるようなものを根付き二分木，あるいは単に二分木という．根付き二分木の各頂点にデータを保存するデータ構造のことも二分木という．

二分木は次の図のように，各頂点になんらかのデータと親と左右の子へのポインタをもたせることで実装される．

定義 10.2 二分ヒープ（binary heap）
二分木のデータ構造であって，次の 2 つの条件をみたすようにしたものを最大二分ヒープという．
(1) 各頂点に保存されている値は子の値以上である．
(2) 各頂点は高さが低い順に，同じ高さでは左から順にデータが埋まっている．
(1) の条件を *max-heap property*，(2) の条件を *shape property* という．(1) の「以上」の部分を「以下」にしたものを最小二分ヒープという．

二分ヒープのことを単にヒープということもある．二分ヒープは次のように親と子の添字を計算することが簡単にできるため，配列を用いて実装することが可能である．

```
1: function PARENT(i)                          ▷ 親の添字を返す関数
2:     return ⌈i/2⌉
3: function LEFT(i)                            ▷ 左の子の添字を返す関数
4:     return 2i
5: function RIGHT(i)
6:     return 2i + 1                           ▷ 右の子の添字を返す関数
```

次の主張は，二分ヒープの高さが $\Theta(\log n)$ であることによって成立する．

主張 10.1
最大二分ヒープにおいて，最大値の取得の計算量は $O(1)$，要素の追加・削除の計算量は $O(\log n)$ でできる．

また，未整列のリストの要素をすべてヒープに追加して，その後すべて取り出すことで，整列したリストを得ることができる．このソートの方法を**ヒープソート**という．

主張 10.2
ヒープソートの計算量は $O(n \log n)$．

10.2　数理計画問題

一般に，様々な問題を次の形へ書き換えることができると知られている．

入力：集合 $\boldsymbol{S} \subseteq \mathbb{R}^n$ と関数 $f : \mathbb{R}^n \to \mathbb{R}$ が与えられる．
目的：$\boldsymbol{S}$ のなかで f を最小（あるいは最大）とする元を見つける．

上で現れる関数 f を**目的関数**，集合 $\boldsymbol{S}$ のことを**実行可能集合**あるいは**実行可能領域**，その元 $\boldsymbol{x} \in \boldsymbol{S}$ を**実行可能解**という．また，集合 $\boldsymbol{S}$ を定義するときに現れる条件を，**制約条件**という．そして，実行可能解のなかで目的関数を最小（あるいは最大）とするものを**最適解**という．このような問題を総称して，**数理計画問題**という．

線形計画問題

数理計画問題の中でも制約条件がいくつかの 1 次不等式や等式であり，なおかつ目的関数が 1 次関数である問題を**線形計画問題**という．どのような線形計画問題であっても，次の形へと変形できることが知られている．

入力：実行可能集合 $S = \{\boldsymbol{x} \mid A\boldsymbol{x} = \boldsymbol{b},\ \boldsymbol{x} \leq 0\}$ と目的関数 $f(\boldsymbol{x}) = \boldsymbol{c}^{\mathrm{T}}\boldsymbol{x}$ が与えられる．
目的：$\boldsymbol{S}$ のなかで f を最小とする元を見つける．