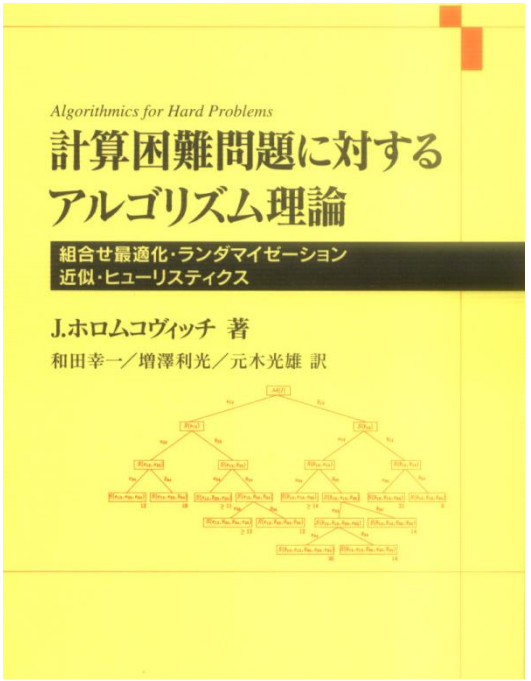
計算困難問題に対するアルゴリズム理論
(組み合わせ最適化・ランダマイゼーション・近似・ヒューリスティス)
第二章:初歩的な基礎
J. ホロムコヴィッチ(訳: 和田幸一, 増澤利光, 元木光雄)
Springer , pp.11～160 , 2012.

2014/6/21
理論輪読会
B4　森部伸一

第2章：初歩的な基礎の概要

- 本書の2章以降の内容に関して必要な初歩的な基礎をすべて記述

2.1　序論
2.2　数学の基礎
　2.2.2　線形代数
　2.2.2　組合せ、数え上げ、グラフ理論
　2.2.3　ブール関数とブール式
　2.2.4　代数と数論
　2.2.5　確率論
2.3　アルゴリズム論の基礎
　2.3.1　アルファベット、語、言語
　2.3.2　アルゴリズム問題
　2.3.3　計算量理論
　2.3.4　アルゴリズム設計技法



グラフ理論の基礎

グラフGは以下の条件を満たす対(V,E)で定義される

(1)VはGの頂点集合と呼ばれる有限集合である

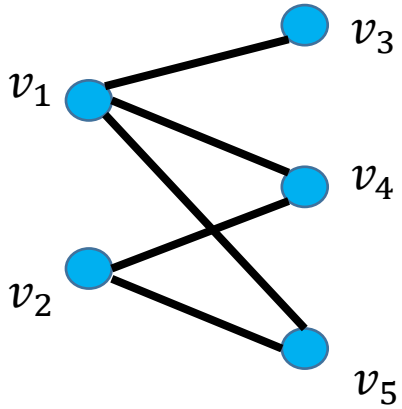
(2)EはGの辺集合と呼ばれる{{u,v}|v,u∈Vかつv≠u}の部分集合である

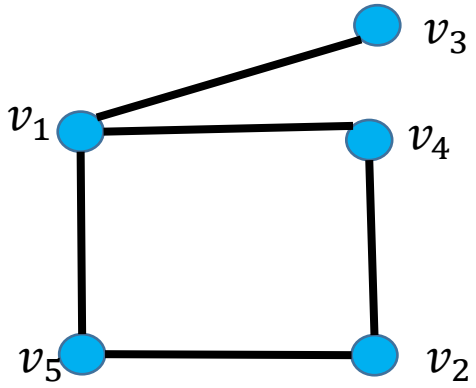
例:

V=$\{v_1, v_2, v_3, v_4, v_5\}$

E=$\{\{v_1, v_3\},\{v_1, v_4\},\{v_1, v_5\},\{v_2, v_4\},\{v_2, v_5\}\}$

のときG=(V,E)のグラフは以下のように表せる





辺を表す曲線が平面のどの点でも交差しないようなGの図的表現が存在するならGは**平面的である**という

2頂点uとv の間には辺が存在するかを表現する方法として**隣接行列**という表現がある。

$$\begin{pmatrix} 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \end{pmatrix}$$

隣接行列(有向グラフ・重み付きグラフ)

有向グラフの隣接行列

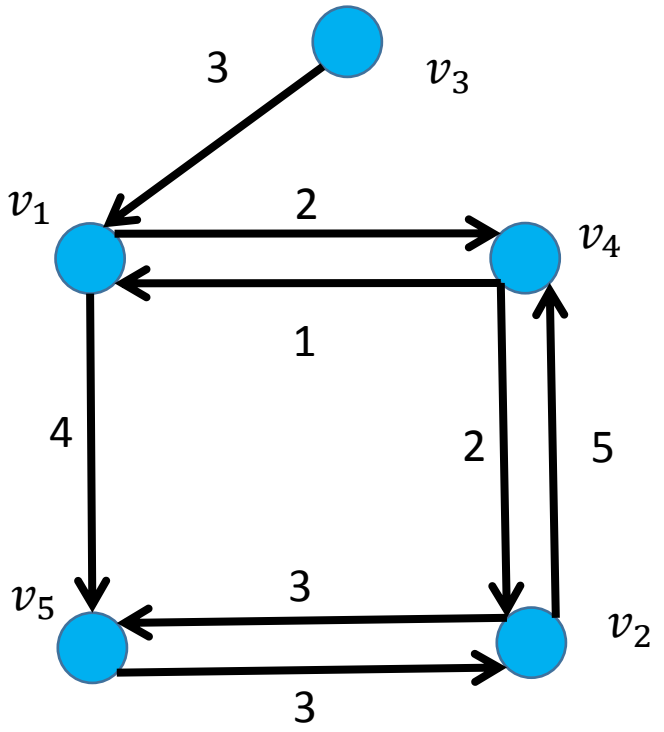
V=$\{v_1, v_2, v_3, v_4, v_5\}$
E=$\{\{v_1, v_4\},\{v_1, v_5\},\{v_2, v_4\},\{v_2, v_5\},\{v_3, v_1\},\{v_4, v_1\},\{v_4, v_2\},\{v_5, v_2\}\}$
($\{v_i, v_j\}$はiからjに到達できることを示している)
iからjに到達できるときi行目j列目を1として行列を有効グラフの隣接行列という。

$$\begin{pmatrix} 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{pmatrix}$$

重み付き(有効)グラフの隣接行列

iからjに到達できるとき、その経路に重み付け(時間など)をしてi行目j列目にその値を代入した行列を重みづきグラフの隣接行列という。

$$\begin{pmatrix} 0 & 0 & 0 & 2 & 4 \\ 0 & 0 & 0 & 5 & 3 \\ 3 & 0 & 0 & 0 & 0 \\ 1 & 2 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 & 0 \end{pmatrix}$$



ブール値と論理演算

- ブール論理はブール値「真」(=1)・「偽」(=0)から構成される基礎的な数学の体系(以降では真、偽に対して1,0で表す)

- ブール値は論理演算によって操作される
  ＜基礎的な演算＞
  否定：¬(1)=0 ($\bar{1}$によって表すこともある)
  乗法：記号∧によって表現(AND)
  加法：記号∨によって表現(OR)
  排他的論理和：記号⊕によって表現(XOR)
  含意：記号⇒によって表現
  　　(第1命題が偽または第2命題が真のときに真となる論理演算)
  等価：記号⇔によって表現

A	B	A∧B	A∨B	A⊕B	A⇒B	A⇔B
0	0	0	0	0	1	1
0	1	0	1	1	1	0
1	0	0	1	1	0	0
1	1	1	1	0	1	1

ブール変数とブール関数

ブール変数：0か1の値が与えられる任意の記号$(x_1, x_2, \cdots, x_n)$

ブール関数：n個のブール変数の集合$\{0,1\}^n$から{0,1}への任意の写像fである

$f(\alpha) = 1$となるようなブール変数の集合に対して
”αはfを充足する“という
$f(\beta) = 0$となるようなブール変数の集合に対して
”βはfを充足しない“という

右図の例の場合
fを充足するすべての入力
{(0,0,0),(0,1,1),(1,0,0),(1,1,0)}
fを充足しないすべての入力
{(0,0,1),(0,1,0),(1,0,1),(1,1,1)}

x_1	x_2	x_3	$f(x_1, x_2, x_3)$
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

ブール関数の例

ブール式

ブール式とはブール関数を定義する一般的な方法

ブール式の定義

Xをブール変数の可算集合とし、Sを単項ブール演算と2項ブール演算の集合とする。XとSの上のブール式は以下のように再帰的に定義される

(ⅰ)ブール値0,1はブール式である

(ⅱ)任意のブール変数x∈Xに対して、xはブール式である

(ⅲ)Fがブール式でψ∈Sが単項ブール演算ならψ(F)はブール式である

(ⅳ)F_1とF_2がブール式で、Δ ∈Sが2項演算ならば$(F_1 \Delta F_2)$はブール式である

(ⅴ)(ⅰ),(ⅱ),(ⅲ),(ⅳ)のみを使って構成された式のみがXとS上のブール式である

ブール式とブランチングプログラム

前のページのブール関数は以下のように表すことができる

①$((x_1 \wedge x_2) \vee (x_1 \wedge \neg x_2) \oplus x_3)$

fを充足するすべての入力 {(0,0,0),(0,1,1),(1,0,0),(1,1,0)}

②$(\neg x_1 \wedge \neg x_2 \wedge \neg x_3) \vee (\neg x_1 \wedge x_2 \wedge x_3) \vee (x_1 \wedge \neg x_2 \wedge \neg x_3) \vee (x_1 \wedge x_2 \wedge \neg x_3)$

⇒ 完全加法標準形(完全DNF)

fを充足しないすべての入力 {(0,0,1),(0,1,0),(1,0,1),(1,1,1)}

③$(x_1 \vee x_2 \vee \neg x_3) \wedge (x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_3)$

⇒ 完全乗法標準形(完全CNF)

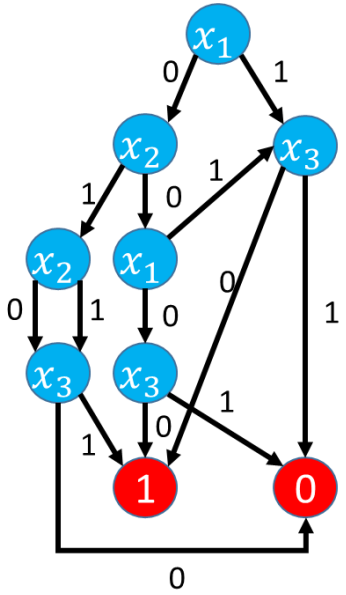
x_1	x_2	x_3	$f(x_1, x_2, x_3)$
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

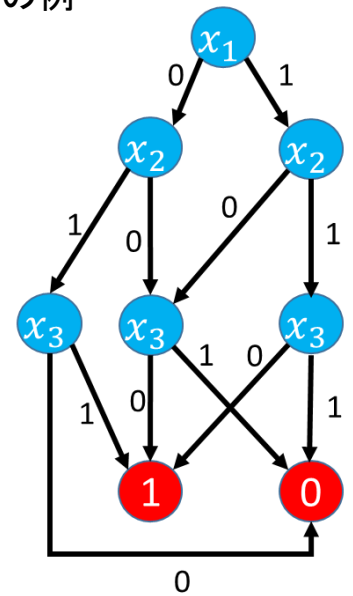
ブール関数の例

ブランチングプログラム

①入次数が0である頂点が存在(ブランチングプログラムのソース)
②出次数が0でない任意の頂点はブール変数でラベル付けされている
③出次数が0の頂点が2つ存在(ブランチングプログラムのシンク)
④出次数が0でない任意の頂点は出次数2である

任意の有効路Pに対してP上のすべての頂点が互いに異なったラベルを持つとき、**1回読みブランチングプログラム**という





形式言語理論における初等的な基礎

任意の空でない有限集合を**アルファベット**と呼ぶ。アルファベットΣの任意の要素はΣの**シンボル**と呼ばれる

$$\Sigma_{bool} = \{0,1\}$$

またアルファベットΣのシンボルの任意の有限列を**語**という。**λ**は0個のシンボルからなる唯一の語である。**Σ***はアルファベットΣ上のすべての語の集合を示す。

$$\Sigma = \{a, b\}\text{に対して}$$
$$\Sigma^* = \{\lambda, a, b, aa, ab, ba, bb, aaa, \ldots.\}$$

アルゴリズム問題　決定問題

アルゴリズム：数学・コンピューティング・言語学などに関連する分野において**問題を解くための手順を定式化した形で表現したもの。**アルゴリズムを使って問題として、決定問題や最適化問題がある。

決定問題の定義

決定問題は、３項組(L,U,Σ)である。ここでΣはアルファベットでL⊆U⊆Σ*である。
アルゴリズムAが決定問題(L,U,Σ)を解くと行くのは、任意のx⊆Uに対して
(ｉ)x∈Lのとき、A(x)=1
(ii)x∈U-Lのとき、A(x)=0　となることである

決定問題の記述は以下の入出力の振る舞いを指定する形式と等しい

問題(L,U,Σ)
入力：x∈U
出力：x∈L　ならば 「はい」
そうでなければ 「いいえ」

アルゴリズム問題　最適化問題

最適化問題の定義

最適化問題は以下の条件を満たす7項組U=(Σ_I, Σ_O,L,LI,M,$cost$,$goal$)で定義される
(ⅰ) Σ_Iはアルファベットで、Uの入力アルファベットと呼ばれる。
(ⅱ) Σ_Oはアルファベットで、Uの出力アルファベットと呼ばれる。
(ⅲ)L($\subseteq\Sigma_I^*$)は実行可能な問題インスタンスの言語
(ⅳ)$L1\subseteq L$はUの(実)問題インスタンスの言語
(ⅴ)MはLからべき集合Σ_O^*への関数で、任意の$x\subseteq L$に対して、$M(x)$はxに対する実行可能解の集合である
(ⅵ)$cost$は任意の対(u,x)に対するコスト関数である。ここで、$u\in M(x)$であり、ある$x\in L$対して、正の実数$cost(u,x)$を割り当てる
(ⅶ)$goal\in\{minimum,maximum\}$
任意のx∈L1に対して
$cost(y,x)=goal\{cost(z,x)/z\in M(x)\}$のとき実行解$y\in M(x)$はxとUに対して最適であるという。

本書では問題の定義をわかりやすくするように、Σ_I, Σ_Oは使用せずに、以下の点のみを記述
- **入力**
- **制約**
- **コスト関数**
- **ゴール**

単純ナップサップ問題(SKP)

問題の説明

正整数bで与えられる重量制限のあるナップサックと、重さが$\omega_1, \omega_2, \ldots, \omega_n$であるn個の荷物があるとき、ナップサップに荷物を詰め込み、荷物の重さの合計がb超えないように最大化するという問題。

下のようにして単純ナップサップ問題は定義

入力：正整数bとあるn∈自然数に対してn個の正整数$\omega_1, \omega_2, \ldots, \omega_n$

制約：$M(b, \omega_1, \omega_2, \ldots, \omega_n) = \{T \subseteq \{1, \ldots, n\} | \Sigma_{i \in T} \omega_i \leq b\}$
すなわち、問題インスタンス$b, \omega_1, \omega_2, \ldots, \omega_n$に対する実行可能解は、その合計が$b$を超えない荷物の任意の集合である。

コスト：各$T \in M(b, \omega_1, \omega_2, \ldots, \omega_n)$に対して$cost(T, b, \omega_1, \omega_2, \ldots, \omega_n) = \Sigma_{i \in T} \omega_i$

ゴール：$maximum$

計算量理論

計算量は、計算機やそのシステムのソフトウェアの特徴的・技術特徴に依存しないので、アルゴリズムを比較する際の指標になる。

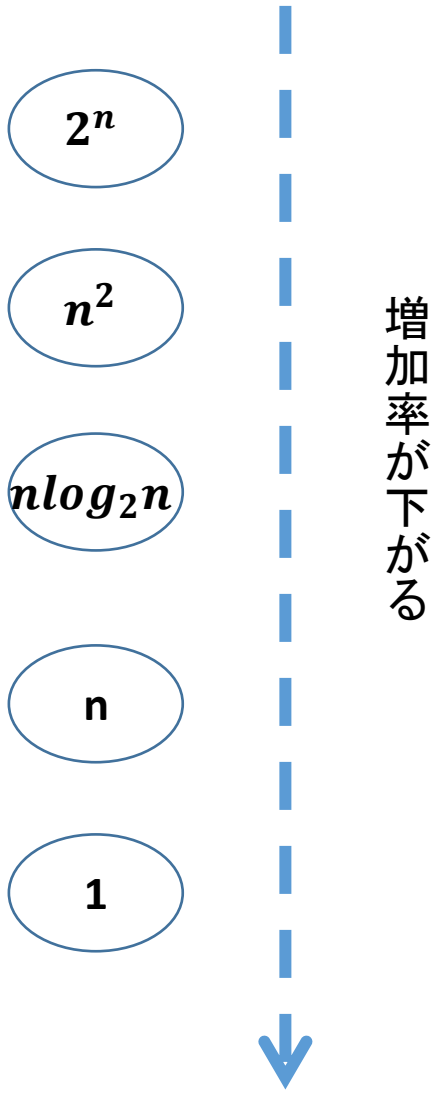
O表記法(Big-O notion)
データ件数に対して繰り返し回数の増加を、漸近的に求めた値
データ件数が非常に大きい場合の増加の度合いを最重視している

あるアルゴリズムの入力に対する計算量を以下のように定義する

$$f(n) = 2n^2 + 500$$

O表記法による計算量は以下のように表現できる

$$f(n) = O(n^2)$$



アルゴリズムによる計算量の違い(ソート)

ソートとは、整列のことで降順または昇順に並べることをいう。ソートはアルゴリズムによって計算量が異なることを示す。

バブルソート法

並べ替える対象を並べて隣り合う要素を比較して、大小の順が逆であれば、それらの要素を入れ替えるという操作を繰り返して行う方法

3,4,2,5,1 ⇒ 1,2,3,4,5

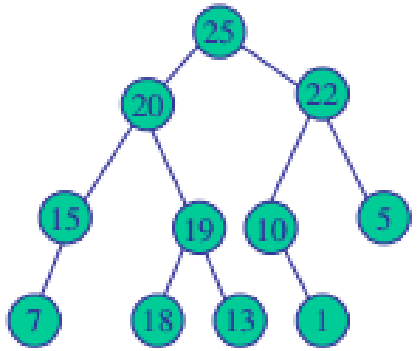
計算量：O(n^2)

ヒープソート法

二分木の構造を使うソート法

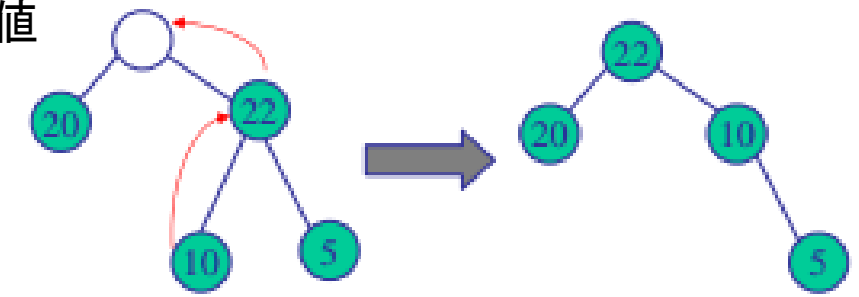
すべてのノードは下の二つのノードよりも大きいようにヒープ作る.

計算量：O(nlogn)



ヒープを再構成しながら、最大値を取り出していくことでソート

計算量：*O(nlogn)*



合計の計算量は*O(nlogn)*

アルゴリズム設計技法

2.3.4では最も重要なアルゴリズム設計技法を5つ紹介

・分割統治法

・動的計画法

・バックトラッキング法

・局所探索法

・貪欲アルゴリズム

これらの技法と新しいアイデアやアプローチを組み合わせると困難問題に対する実際的なアルゴリズムの設計の手助けになる。

動的計画法

動的計画法：対象となる問題を複数の部分問題に分割し、部分問題の計算結果を記録しながら解いていく手法である。

動的計画法の例

全対最短経路問題：与えられた重み付きグラフ(G,c)のすべての頂点に対して最短経路のコストを見つけるという問題。ただし、$G(V,E)$かつcは自然数。V=$\{v_1, v_2, \dots, v_n\}$とする

アルゴリズムのアイディア

$Cost_k(i,j) = \{v_1, v_2, \dots, v_n\}$のみの内部頂点を用いた$v_i, v_j$間の最短経路コスト。ただしk=0,1,…,nである

初期値として

$$Cost_0(i,j) = \begin{cases} c(\{v_1, v_2\}) & \{v_1, v_2\}\text{がリンクとして含まれている場合} \\ \infty & \{v_1, v_2\}\text{がリンクとして含まれていなく、} i \neq j\text{の場合} \\ 0 & i = j\text{の場合} \end{cases}$$

を入力

頂点v_kを経由した方がコストが少ないなら、その値を暫定的な最短経路として記録

$$Cost_k(i,j) = \min\{Cost_{k-1}(i,j), Cost_{k-1}(i,k) + Cost_{k-1}(k,j)\}$$

全対最短経路問題アルゴリズム (FLOYDのアルゴリズム)

右はフロイドによるアルゴリズム

Step2の部分が動的計画法
(計算結果を記録しながら最短経路を検索している)

このときの計算量は$O(n^3)$

入力：グラフG=(V,E),V={$v_1, v_2, \ldots, v_n$},n(自然数)
Step1:for i = 1 to n do
do begin Cost [i,i] :=0;
for j :=1 to n do
if $\{v_1, v_2\} \in E$ *then*
Cost[i,j]:=c($\{v_1, v_2\}$)
else if i≠j then Cost[i,j]:=∞
end
Step2:for k:=1 to n do
for i:=1 to n do
for j: = 1 to n do
Cost[i,j]:=min{Cost[i,j],Cost[i,k]+Cost[k,j]}
出力：全対最短経路

まとめ

- 計算量の多い問題に対してアルゴリズムを作るためには、数学的な基礎を抑えて、アルゴリズムを作る必要がある。
- 計算量を抑えるようなアルゴリズムが望ましい。
- 最適化問題に関しては、規範となるような最適化問題アルゴリズムがあるので、それと新しいアイデアやアプローチを組み合わせてアルゴリズムを設計するとよい。