アルゴリズムとデータ構造

テキスト

山本真基

２０２１年4月

機械的な手順で解決可能な問題を解く際には，その問題にあったアルゴリズムとデータ構造が必要である．アルゴリズムとは「問題」を解くための機械的手順のことである．データ構造とは（アルゴリズムが扱う）データを保持する「形式」のことである．ここでは，アルゴリズムとは何か，データ構造とは何か，から始め，以下の目次にあるよう，探索・整列・走査といった基本的なアルゴリズム及びそれに付随するデータ構造を学習する．特に，アルゴリズムの正当性の証明と時間計算量の解析について解説する．

以降で使われる表記

- $\mathbb{N}, \mathbb{Z}, \mathbb{Q}, \mathbb{R}$ をそれぞれ，自然数，整数，有理数，実数の集合とする．($\mathbb{Q}^+, \mathbb{R}^+$ をそれぞれ，正の有理数，正の実数，の集合とする．) また，$\mathbb{N}_0 = \mathbb{N} \cup \{0\}$, $\mathbb{Z}_n = \{0, 1, 2, \ldots, n-1\}$ とする．更に, 任意の $n \in \mathbb{N}$ について, $[n] = \{1, 2, \ldots, n\}$ とする．
- 対数関数 log, ln について，底が 2 であるとき log，底が e であるとき ln，と表記する．(よって，任意の $x \in \mathbb{R}^+$ について $2^{\log x} = x$, $\mathrm{e}^{\ln x} = x$.)

新出用語

本書で学習する新しい用語は，新出時に**太字で**書かれる．更に，それらは，(たいてい) 以下のように「定義」として丸枠で囲われる．

定義 1.1

単一コスト RAM（Random Access Memory）モデルとは，以下の操作が単一コスト（１ステップ）で行えるという「計算モデル」である．

- データへのアクセス（読み取りと書き込み）
- 二つの値の四則演算
- 二つの値の大小比較

アルゴリズムが解く問題

本書で学習する「アルゴリズムが解く」問題は，すべて以下のように問題の名前を明記して丸枠で囲われる．

探索問題（searching）

- 入力：整数 $x \in \mathbb{Z}$ と整数の列 $a_1, a_2, \ldots, a_n \in \mathbb{Z}$
- 質問：x が $a_1, a_2, \ldots, a_n$ に存在するか？

アルゴリズム

本書で学習するアルゴリズムは，すべて以下のようにシアン色で塗られる．

入力：整数 $x \in \mathbb{Z}$ と整数の列 $a_1, a_2, \ldots, a_n \in \mathbb{Z}$

1. それぞれの $i \in [n]$ について（順次）以下を繰り返す．
   (a) $a_i = x$ であれば `YES` を出力して終了する．
2. `NO` を出力する．

命題，定理，補題，系

本書で学習する命題，定理，補題，系は，すべて以下のようにマゼンタ色で塗られる．

定理 1.1. 図 1.2 のアルゴリズム A は探索問題を解く．

本文中の問

本書にある問は，すべて以下のように灰色で塗られる．

問 1.1（決定問題）．素数性判定問題，充足可能性問題とはどんな問題であるか調べ，探索問題や到達可能性問題にならって問題の定義をそれぞれ示しなさい．（数学記号を適当に導入して，「入力」と「質問」を明確に示すこと．）

本文中の実装

本書にある実装は，すべて以下のようにイエロー色で塗られる．

実装 2.1. 以下で示されたプログラムをもとに，線形探索のアルゴリズムの実装を完成させなさい．（コメントアウトされた部分に，図 2.1 のステップ 2, 3 を実装する．）更に，何度か実行させて，プログラムの動作確認をしなさい．

線形探索のプログラム

```
#include <iostream>
using namespace std;

#define NUMBER        5

int main()
{
    int x, a[NUMBER];

    for (int i=0; i<NUMBER; i++) {
        cin >> a[i];
    }
    for (int i=0; i<NUMBER; i++) {
        cout << a[i] << " ";
    }
    cout << endl;

    cout << "探索する整数：";
    cin >> x;

    /*
    線形探索のアルゴリズム
    */

    return 0;
}
```

目次

第 1 章

アルゴリズムとは

アルゴリズムとは,「問題」を解くための機械的手順である．では，問題とは何か．問題には以下のように大きく二つの種類がある．

- 決定問題（decision problem）：YES/NO や 1/0 や「はい・いいえ」など，2 値のどちらかで答える問題.
- 関数問題（function problem）：自然数や集合や数列など,「値」で答える問題.

決定問題

探索問題（searching）

- 入力：整数 $x \in \mathbb{Z}$ と整数の列 $a_1, a_2, \ldots, a_n \in \mathbb{Z}$
- 質問：x が $a_1, a_2, \ldots, a_n$ に存在するか？

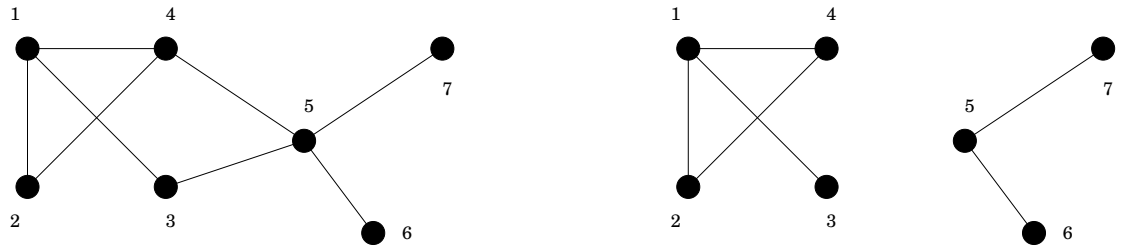
例 1.1 (探索問題)**.**

- $x = 7$, $(a_1, \ldots, a_5) = (7, 3, 5, 7, 11)$ の場合は YES.
- $x = 1$, $(a_1, \ldots, a_5) = (7, 3, 5, 2, 11)$ の場合は NO.

到達可能性問題（reachability）

- 入力：グラフ $G = (V, E)$, $s, t \in V$
- 質問：s から t への経路が存在するか？

例 1.2 (到達可能性問題)**.** 以下の図において，$V = \{1, 2, \ldots, 7\}$, $s = 1$, $t = 7$ のとき，入力グラフ G が左図 (a) の場合は YES，右図 (b) の場合は NO.



(a) 到達可能　　(b) 到達不可能

図 1.1: 到達可能性問題の例

関数問題

最大公約数問題（greatest common divisor）

- 入力：自然数 $x, y \in \mathbb{N}$
- 解：x, y の最大公約数

例 1.3 (最大公約数問題)**.** $(x, y) = (12, 123)$ の場合，解は 3.

整列問題（sorting）

- 入力：整数の列 $a_1, a_2, \ldots, a_n \in \mathbb{Z}$
- 解：$a_1, a_2, \ldots, a_n$ の昇順，つまり，$a_{i_1}, a_{i_2}, \ldots, a_{i_n}$ s.t. $a_{i_1} \leq a_{i_2} \leq \cdots \leq a_{i_n}$

例 1.4 (整列問題)**.** $(a_1, \ldots, a_7) = (0, 7, -1, 11, 2, 3, 5)$ の場合，解は $-1, 0, 2, 3, 5, 7, 11$.

他にも，決定問題として，素数性判定問題，充足可能性問題などが，関数問題として，足し算・掛け算などの四則演算，最短経路探索問題，巡回セールスマン問題などがある．このように，（おおざっぱに言って）コンピュータで解くことのできる問題をアルゴリズムが解く「問題」とする．

問 1.1 (決定問題)**.** 素数性判定問題，充足可能性問題とはどんな問題であるか調べ，探索問題や到達可能性問題にならって問題の定義をそれぞれ示しなさい．（数学記号を適当に導入して，「入力」と「質問」を明確に示すこと．）

問 1.2 (関数問題)**.** 最短経路探索問題，巡回セールスマン問題とはどんな問題であるか調べ，最大公約数問題や整列問題にならって問題の定義をそれぞれ示しなさい．（数学記号を適当に導入して，「入力」と「解」を明確に示すこと．）

アルゴリズムとデータ構造の「理論」では，提示されたアルゴリズムに対して，以下を「数学的に」証明することが議論の中心となる．

- アルゴリズムの正当性（どのような入力にも正しく動作する．）
- アルゴリズムの計算量（計算時間や計算領域など．）

1.1　探索問題を例に

アルゴリズムの擬似コード

以下の図 1.2 が，探索問題を解く線形探索と呼ばれるアルゴリズムである．

入力：整数 $x \in \mathbb{Z}$ と整数の列 $a_1, a_2, \ldots, a_n \in \mathbb{Z}$

1. それぞれの $i \in [n]$ について（順次）以下を繰り返す．
   (a) $a_i = x$ であれば `YES` を出力して終了する．
2. `NO` を出力する．

図 1.2: 線形探索のアルゴリズム

このようなアルゴリズムの記述を**擬似コード**という．これを C++ 言語で記述すると図 1.3 のようになる．

アルゴリズムの正当性

定理 1.1. 図 1.2 のアルゴリズム A は探索問題を解く．

```
#include <iostream>
using namespace std;

#define n 100

int main()
{
        int a[n] = {321,2,3,-5,...};
        int x = -123;

        for (int i=0; i<n; i++) {
                if (a[i]==x) {
                        cout << YES << endl;
                        return 0;
                }
        }
        cout << NO << endl;
        return 0;
}
```

図 1.3: 線形探索の C++ コード

証明. 整数 $x \in \mathbb{Z}$ と整数の列 $a_1, a_2, \ldots, a_n \in \mathbb{Z}$ を任意とする．定理を示すためには以下のことを示せばよい．

$$\begin{array}{ll} x \in \{a_1, \ldots, a_n\} & \text{のとき} \quad A(x, (a_1, \ldots, a_n)) = \texttt{YES} \\ x \notin \{a_1, \ldots, a_n\} & \text{のとき} \quad A(x, (a_1, \ldots, a_n)) = \texttt{NO} \end{array}$$

まず，一つ目を示す．この場合，$\forall i \in [i_0 - 1][a_i \neq x]$ かつ $a_{i_0} = x$ を満たす $i_0 \in [n]$ が存在する．このとき，アルゴリズムより，$i = i_0$ において YES を出力して終了するのは明らかである．よって，$A(x, (a_1, \ldots, a_n)) = \texttt{YES}$ である．

次に，二つ目を示す．このとき，$\forall i \in [n][a_i \neq x]$ であることから，アルゴリズムより，ステップ 1-(a) の条件（$a_i = x$）は成り立たないので，ステップ 1-(a) で YES を出力して終了することはない．ステップ 1 はいずれ終了するので，ステップ 2 で NO を出力して終了するのは明らかである．よって，$A(x, (a_1, \ldots, a_n)) = \texttt{NO}$ である． ■

アルゴリズムの計算時間

定義 1.1

単一コスト RAM（Random Access Memory）モデル とは，以下の操作が単一コスト（１ステップ）で行えるという「計算モデル」である．

- データへのアクセス（読み取りと書き込み）
- 二つの値の四則演算
- 二つの値の大小比較

以降，アルゴリズムの計算時間の解析には，それを単純化するため（計算時間を解析するにあたり「本質的」でない部分の解析を省略するため），単一コスト RAM モデルを仮定する．

定義 1.2

入力 x に対して，x の長さを $|x|$ と表記する．

各問題において，入力の長さ（入力長）をどのように考えればよいだろうか．例えば，探索問題の入力 $(3.14, (1.23, -1, 0, 456, 98.7))$ の入力長はいくらだろうか．

例 1.5. 下記の問題に対する入力長は，それぞれ以下のようにするのが慣習である．

探索問題	:	$\|(x, (a_1, \ldots, a_n))\|$	$=$	n
到達可能性問題	:	$\|(G = (V, E), s, t)\|$	$=$	$\|V\| + \|E\|$
最大公約数問題	:	$\|(x, y)\|$	$=$	$\log x + \log y$
整列問題	:	$\|(a_1, \ldots, a_n)\|$	$=$	n

定義 1.3

アルゴリズム A の計算時間が $f(n)$ であるとは，$|x| = n$ である任意の入力 x に対して，$A(x)$ が（単一コスト RAM モデルで）$f(n)$「ステップ」以内で終了することである．これを，**最悪時間計算量**という．

定理 1.2. 図 1.2 のアルゴリズム A の計算時間は $3n + 1$ である．

証明. 最悪時間を見積もるので，$x \notin \{a_1, \ldots, a_n\}$，つまり，$\forall i \in [n][a_i \neq x]$ である入力 $x, (a_1, \ldots, a_n)$ の場合を考えればよい．

問 1.3. この理由を説明しなさい.

アルゴリズムの各ステップの計算時間を見積もる．ステップ 1 の「それぞれの $i \in [n]$ について（順次）以下を繰り返す」に時間を要する箇所は，i のインクリメント操作（$i+1$ の演算）と $i \leq n$ の判定をする部分である．単一コスト RAM モデルの仮定より，計算時間はそれぞれ 1 となる．「$a_i = x$ であれば YES を出力して終了する」に時間を要する箇所は，$a_i = x$ の判定をする部分である．（$\forall i \in [n][a_i \neq x]$ であるので，YES の出力にかかる時間を考える必要はない.）単一コスト RAM モデルの仮定より，計算時間は 1 となる．これらの計算時間は合計で 3 となり，それぞれが n 回行われる．（よって，ステップ 1 にかかる計算時間の合計は $3n$ となる.）また，ステップ 2 の「NO を出力する」は，単一コスト RAM モデルの仮定より，計算時間は 1 となる．よって，アルゴリズム全体の計算時間は $3n + 1$. ■

問 1.4. 上の証明では，$\forall i \in [n][a_i \neq x]$ である入力 $x, (a_1, \ldots, a_n)$ の場合の計算時間を見積もった.（証明の中の問で，その場合を考えれば十分であることを確認した.）では，実際に，$\exists i \in [n][a_i = x]$ である入力 $x, (a_1, \ldots, a_n)$ の場合の計算時間を見積もり，上の問で示したことの正しさを確認しなさい.（計算時間の導出を示すこと.）

1.2　最大公約数問題を例に

以下の図 1.4 が，最大公約数問題を解くユークリッドの互除法と呼ばれるアルゴリズムである.

入力：自然数 x, y

1. $y > 0$ である限り以下を繰り返す.
   (a) $r = x \bmod y$ とする.
   (b) $x = y$, $y = r$ とする.
2. x を出力する.

図 1.4: ユークリッドの互除法のアルゴリズム

これを C++ 言語で記述すると図 1.5 のようになる.

```
#include <iostream>
using namespace std;

int main()
{
        int x = 123, y = 12, r;

        while (y > 0) {
                r = x % y;
                x = y;
                y = r;
        }
        cout << x << endl;
        return 0;
}
```

図 1.5: ユークリッドの互除法の C++ コード

定理 1.3. 図 1.4 のアルゴリズム A は最大公約数問題を解く.

証明. 略. ■

問 1.5. この定理を証明しなさい.

定理 1.4. $x \geq y$ とする．図 1.4 のアルゴリズム A の計算時間は $8\log y + 9$ である．

証明. まず，アルゴリズムのステップ 1 の繰り返し回数を見積もる．その回数を $k+1$ とする．任意の $i \in [k+1]$ について，ステップ 1 の第 i 回目の繰り返し後の r を r_i とする．（$r_{-1} = x, r_0 = y$ とする．）このとき，ある整数 $q_1, q_2, \ldots, q_{k+1}$ が存在して，以下の等式が成り立つ．（左の列は繰り返し回数を表す．）

$$\begin{array}{rcccccc}
1 & : & x & = & q_1 y & + & r_1 \\
2 & : & y & = & q_2 r_1 & + & r_2 \\
3 & : & r_1 & = & q_3 r_2 & + & r_3 \\
\vdots & \vdots & & & \vdots & & \\
k-1 & : & r_{k-3} & = & q_{k-1} r_{k-2} & + & r_{k-1} \\
k & : & r_{k-2} & = & q_k r_{k-1} & + & r_k \\
k+1 & : & r_{k-1} & = & q_{k+1} r_k & + & r_{k+1}
\end{array}$$

問 1.6. r_{k+1} はいくつか．

主張 1.1. 任意の $i \in \{-1, 0, 1, 2, \ldots, k-1\}$ について，

$$r_{i+2} \leq \frac{r_i}{2}.$$

証明. アルゴリズムより，（第 $i+2$ 回目の繰り返しにおいて）ある整数 q が存在して，

$$r_i = q r_{i+1} + r_{i+2}$$

$r_{i+1} \leq r_i/2$ の場合，余りの定義より，

$$r_{i+2} < r_{i+1} \leq \frac{r_i}{2}.$$

$r_{i+1} > r_i/2$ の場合，$q = 1$ より，

$$r_i = r_{i+1} + r_{i+2}$$

よって，

$$r_{i+2} = r_i - r_{i+1} < r_i - \frac{r_i}{2} = \frac{r_i}{2}.$$

■

k が偶数であるとする．この主張より，（$r_{k+1} = 0$ より $r_k \geq 1$ であるから）

$$1 \leq r_k \leq \frac{r_{k-2}}{2} \leq \frac{r_{k-2\cdot 2}}{2^2} \leq \frac{r_{k-2\cdot 3}}{2^3} \leq \cdots \leq \frac{r_{k-2\cdot i}}{2^i} \leq \cdots \leq \frac{r_0}{2^{k/2}} = \frac{y}{2^{k/2}}.$$

よって，$2^{k/2} \leq y$，つまり，$k \leq 2\log y$．（ステップ 1 の繰り返し回数は高々 $2\log y + 1$．）

問 1.7. k が奇数の場合の繰り返し回数は高々いくつになるか．

ステップ 1 の 1 回の繰り返しでは，繰り返しの条件 $y > 0$ の判定と，$r = x \bmod y$，$x = y$，$y = r$ の三つの操作が行われる．（よって，1 回の繰り返しにつき 4 つの操作が行われる．）単一コスト RAM モデルの仮定より，計算時間はそれぞれ 1 となる．また，ステップ 2 の「x を出力する」は，単一コスト RAM モデルの仮定より，計算時間は 1 となる．よって，アルゴリズム全体の計算時間は，（k が奇数の場合の方が繰り返し回数は多いので）高々，

$$4 \cdot (2\log y + 2) + 1 \;=\; 8\log y + 9.$$

■

問 1.8. 自然数は二進数で扱われるとする．アルゴリズムの入力を $(x, y) \in \mathbb{N} \times \mathbb{N}$ とした場合，入力長 $|(x, y)|$ の（およその）値を示しなさい．

次章以降，アルゴリズムはすべて擬似コードで記述される．（C＋＋コードは省略する．）また，計算時間の解析には，単一コスト RAM モデルを仮定する．

第 2 章

データ構造とは

データ構造とは，アルゴリズムが扱うデータ（の集まり）を保持する「形式」のことである．データ構造は（アルゴリズムを）実装する際のメモリの扱いに関係する．

2.1 配列

最も基本的なデータ構造が配列である．探索問題を解く線形探索のアルゴリズムを配列を明示的に用いた擬似コードとして記述すると以下のようになる．（図 2.1 のステップ 1 が，図 1.2 に示されたアルゴリズムに追記された．）これ以降，擬似コード中の配列は，配

入力：整数 $x \in \mathbb{Z}$ と整数の列 $a_1, a_2, \ldots, a_n \in \mathbb{Z}$

1. a を大きさ n の配列として，それぞれの $i \in [n]$ について $a[i] = a_i$ とする．
2. それぞれの $i \in [n]$ について（順次）以下を繰り返す．
   (a) $a[i] = x$ であれば `YES` を出力して終了する．
3. `NO` を出力する．

図 2.1: 線形探索のアルゴリズム

列名を a，配列の大きさを n としたとき，配列の番号は，**1 から n**となっているものとする．つまり，$a[1], a[2], \ldots, a[n]$ に値が入っているものとする．（C++ 言語では，それが `a[0],a[1],...,a[n-1]` となる．）

実装 2.1. 以下で示されたプログラムをもとに，線形探索のアルゴリズムの実装を完成

させなさい．（コメントアウトされた部分に，図 2.1 のステップ 2, 3 を実装する．）更に，何度か実行させて，プログラムの動作確認をしなさい．

線形探索のプログラム

```
#include <iostream>
using namespace std;

#define NUMBER        5

int main()
{
    int x, a[NUMBER];

    for (int i=0; i<NUMBER; i++) {
        cin >> a[i];
    }
    for (int i=0; i<NUMBER; i++) {
        cout << a[i] << " ";
    }
    cout << endl;

    cout << "探索する整数：";
    cin >> x;

    /*
    線形探索のアルゴリズム
    */

    return 0;
}
```

実装 2.2. 上で実装したプログラムの線形探索のアルゴリズムの直後（`return 0;` の直前）に以下のプログラムを追記しなさい．更に，一度，実行させて，プログラムの動作確認，特に，追記した部分の出力結果を確認しなさい．

メモリ番地

```
    for (int i=0; i<NUMBER; i++) {
        cout << &a[i] << " ";
    }
    cout << endl;

    cout << sizeof(int) << "ずつ増える！" << endl;
```

以上の実装及び実行結果から分かるよう，配列は連続したメモリが確保される．

2.2 連結リスト

前節では，アルゴリズムの中で，（図 2.1 のステップ 2 以降）データが追加されたり削除されたりすることはなかった．一方で，アルゴリズムによっては，アルゴリズムの途中で，データの追加・削除が生じるものもある．このような場合に便利なデータ構造が連結リストである．配列と連結リストの大きな違いは以下である．

- 配列：連続したメモリが確保される．次の値は「隣の」メモリ領域に保持される．
- 連結リスト：必ずしも連続したメモリが確保されるわけでなく，次の値は「ポインタ」で指し示されたメモリ領域に保持される．

これらの違いは以下のような図で示される．配列は連続したメモリが確保されるが，連結

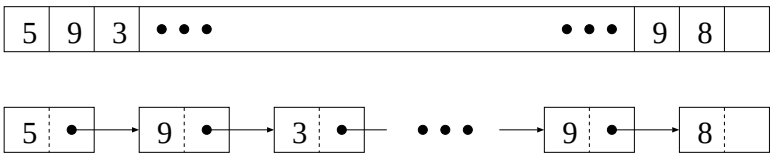


図 2.2: 配列（上段）と連結リスト（下段）

リストは必ずしもそうでなく，各値が各メモリ領域（上の図の長方形の各ブロック）に保持され，それらは「ポインタ」（上の図の矢印）で連結される．（連結リスト全体として確保されたメモリは一般に断片的になる．）

以上で示された違いから，配列と連結リストには以下のようなメリットとデメリットがある[*1]．表の一つ目の項目について，配列 a の i 番目の値を参照するには $a[i]$ とすれば

	配列	連結リスト
指定されたメモリ番地へのアクセスが容易にできる	○	×
データの追加・削除が容易にできる	×	○

表 2.1: 配列と連結リストのメリット・デメリット

よい．一方，連結リストであれば，ポインタを i 回に渡って順々に辿っていかざるを得ない．（先頭のデータ参照を 1 回と数えたとする．）

[*1] これら二つ以外にも，ポインタのためのメモリ領域が必要になるというのが，連結リストのデメリット（配列のメリット）である．

問 2.1 (メモリへのアクセス)**.** 配列と連結リストで，双方ともに，全く同じ n 個のデータを保持しているものとする．データ参照にかかる時間に最も顕著に差が出るのは，何番目のデータを参照する場合であるか．また，その理由を説明しなさい．

表の二つ目の項目について，配列の i 番目と $i+1$ 番目の間に値を追加する場合は，$i+1$ 番目以降の値を一つ一つ隣（次）へ移動させる必要がある．一方，連結リストであれば，追加する値のためのメモリ領域 p を確保して[*2]，（i 番目に辿り着いた上で）i 番目が p を，p が $i+1$ 番目を指し示すようにポインタを変更するだけでよい． また，配列の i

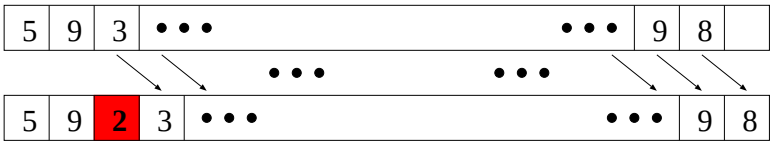


図 2.3: 配列のデータ追加

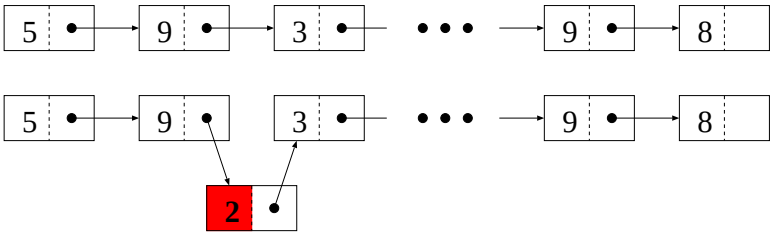


図 2.4: 連結リストのデータ追加

番目を削除する場合は，（追加と同様にして）$i+1$ 番目以降の値を一つ一つ隣（手前）へ移動させる必要がある．一方，連結リストであれば，$i-1$ 番目に辿り着いた上で，$i-1$ 番目が $i+1$ 番目を指し示すようにポインタを変更するだけでよい[*3]．

問 2.2 (データの追加・削除)**.** 配列と連結リストで，双方ともに，全く同じ n 個のデータを保持しているものとする．データの追加にかかる時間に最も顕著に差が出るのは，何番目のデータを追加する場合であるか．また，その理由を説明しなさい．

[*2] C++ では，new 演算子を用いる．
[*3] C++ では，delete 演算子を用いて，i 番目のメモリ領域を解放しておく．

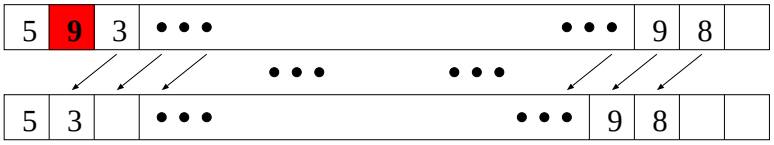


図 2.5: 配列のデータ削除

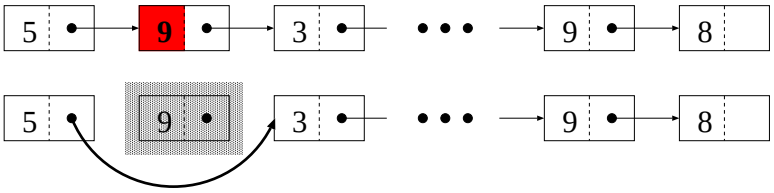


図 2.6: 連結リストのデータ削除

問 2.3 (データの追加・削除)**.** 配列と連結リストで，双方ともに，全く同じ n 個のデータを保持しているものとする．データの削除にかかる時間に最も顕著に差が出るのは，何番目のデータを削除する場合であるか．また，その理由を説明しなさい．

注 2.1. 以降，連結リストの実装はライブラリ関数を利用する[*4]．この関数はポインタを用いて実装されている．

実装 2.3. 以下で示されたプログラムをもとに，線形探索のアルゴリズムの実装を完成させなさい．（コメントアウトされた部分に，図 2.1 のステップ 2, 3 を実装する．）更に，何度か実行させて，プログラムの動作確認をしなさい．

[*4] `#include <list>` で利用可能となる．

リストを用いた線形探索

```
#include <iostream>
#include <list>
using namespace std;

#define NUMBER        5

int main()
{
    int x;
    list<int> intList;

    for (int i=0; i<NUMBER; i++) {
        cin >> x;
        intList.push_back(x);
    }
    for (auto itr=intList.begin(); itr!=intList.end(); ++itr) {
        cout << *itr << " ";
    }
    cout << endl;

    cout << "探索する整数：";
    cin >> x;

    /*
    線形探索のアルゴリズム
    */

    return 0;
}
```

実装 2.4. 上で実装したプログラムの線形探索のアルゴリズムの直後（return 0; の直前）に以下のプログラムを追記しなさい．更に，何度か実行させて，プログラムの動作確認，特に，追記した部分の出力結果を確認しなさい．

メモリ番地

```
    for (auto itr=intList.begin(); itr!=intList.end(); ++itr) {
        cout << &(*itr) << " ";
    }
    cout << endl;
```

以上の実装及び実行結果から分かるよう，連結リストは，必ずしも連続したメモリが確保されるわけではない．

実装 2.5. これまでのプログラム，及び，以下で示されたプログラムを**利用して**，連結リストからデータを追加したり削除したりするプログラムを実装しなさい．（追加・削除による変更が分かるよう，データの追加・削除の前後で，全データを出力させるコードを追記すること．）更に，何度か実行させて，プログラムの動作確認をしなさい．

連結リストのデータ追加・削除（insert, erase）

```
int op, num;
auto itr = intList.begin();

cout << "operation number? 1:add, 2:delete ";
cin >> op;

switch (op) {
    case 1:
        cout << "where ";
        cin >> num;
        cout << "what ";
        cin >> x;
        while (--num>0) ++itr;
        intList.insert(itr, x);
        break;

    case 2:
        cout << "where ";
        cin >> num;
        while (--num>0) ++itr;
        intList.erase(itr);
        break;

    default:
        cout << "Invalid operation number" << endl;
        break;
}
```

なお，上述のプログラムでは，連結リストのデータ追加は insert 関数で，データ削除は erase 関数でなされる．（双方ともに list のメンバ関数である．）

2.3 キューとスタック

キューとスタックも，連結リストと同様，データの追加・削除（取り出し）が可能なデータ構造である．ただし，連結リストよりも限定的な追加・削除（取り出し）になる[*5]．また，キューとスタックでは，（限定的な）追加・削除の方式が対照的である．先に入れたものが先に出る（FIFO: First In First Out）データ構造がキューであり，それとは逆

[*5] 連結リストは，リスト中の任意の場所に追加・削除ができる．

に，後に入れたものが先に出る（LIFO: Last In First Out）データ構造がスタックである．データの追加・削除を意味する用語は以下のように異なる[*6]．

	キュー	スタック
追加	エンキュー（enqueue）	プッシュ（push）
削除	デキュー（dequeue）	ポップ（pop）

表 2.2: キューとスタックの追加・削除の用語

キューでは，データを追加する場合，既に存在する（横一列に並んだ）データの末尾に追加される．データを削除する場合，データの先頭（front）が削除される．このようにし

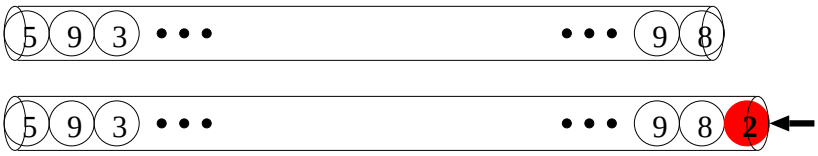


図 2.7: キューのエンキュー

て，先入れ先出し（FIFO）が実現される．

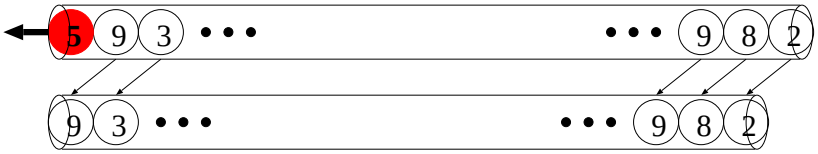
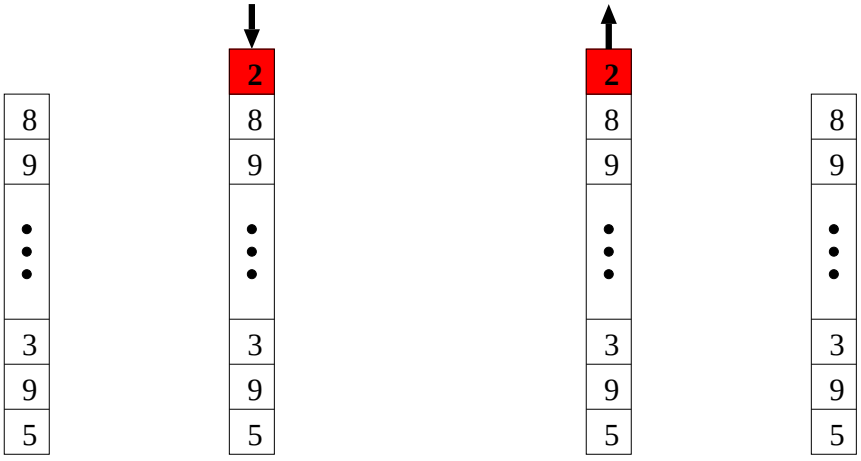


図 2.8: キューのデキュー

問 2.4. あるデータ x がエンキューされた直後，キューに n 個のデータが保持されているものとする．データ x がデキューされるには，何回のデキュー操作が必要となるか．

これに対して，スタックでは，データを追加する場合，既に存在する（縦一列に積まれた）データの頂上（top）に追加される．データを削除する場合，データの頂上が削除される．このようにして，後入れ先出し（LIFO）が実現される．

[*6] ただし，ライブラリ関数のメンバ関数は，双方ともに，追加は `push`，削除は `pop` と，同じ名前にされている．



(a) スタックのプッシュ　　(b) スタックのポップ

図 2.9: スタックのプッシュとポップ

問 2.5. あるデータ x がプッシュされた直後，スタックに n 個のデータが保持されているものとする．データ x がポップされるには，何回のポップ操作が必要となるか．

注 2.2. 以降，キューとスタックの実装はライブラリ関数を利用する[*7]．この関数は配列を用いて実装可能である．

実装 2.6. 以下で示されたプログラムをもとに，キューとスタックからデータを追加したり削除したりするプログラムをそれぞれ実装しなさい．ただし，キューとスタックでは，以下のプログラム中の `xxx`, `yyy` がそれぞれで異なる．（そのままではコンパイルエラーになる．）

	キュー	スタック
`xxx`	`queue`	`stack`
`yyy`	`front`	`top`

各実装に対して，一度ずつ実行させて，プログラムの動作確認をしなさい．

[*7] それぞれ，`#include <queue>`, `#include <stack>` で利用可能となる．

キュー・スタックのデータ追加・削除（push(), pop()）

```
#include <iostream>
#include <xxx>
using namespace std;

#define NUMBER        5

int main()
{
    int a;
    xxx<int> w;

    for (int i=0; i<NUMBER; i++) {
        cin >> a;
        w.push(a);
    }

    while (!w.empty()) {
        cout << w.yyy() << " ";
        w.pop();
    }
    cout << endl;

    return 0;
}
```

なお，上述のプログラムでは，キュー・スタックのデータ追加は push 関数で，データ削除は pop 関数でなされる．（双方ともに queue, list のメンバ関数である．）

第 3 章

オーダー表記

オーダー表記 $O(\cdot)$ の記号 “ O ” は，ランダウの記号（Landau symbol）と呼ばれる．

定義 3.1

$f, g : \mathbb{Z} \to \mathbb{Z}$ を関数とする．$f(n) = O(g(n))$ であるとは，

$$\exists c \in \mathbb{Z}, \exists n_0 \in \mathbb{Z}, \forall n \geq n_0 [f(n) \leq c \cdot g(n)].$$

直感的に，$f(n) = O(g(n))$ とは，$\lim_{n\to\infty} f(n)/g(n)$ が定数（n によらない整数）「以下」になることである．（つまり，$f(n) \leq c \cdot g(n)$.）

例 3.1. 以下は正しいオーダー表記である．

- $1 = O(1)$，$123 = O(1)$
- $1 = O(n)$，$n = O(n)$，$123n + 12345 = O(n)$
- $100n = O(n \log n)$
- $n = O(n^{1.1})$
- $n = O(n^2)$
- $100n^3 + n^2 + n + 2^{100} = O(n^3)$
- $n^{100} = O(2^n)$

一方，以下は正しいオーダー表記でない．

- $n = O(1)$
- $n \log n = O(n)$
- $1.001^n = O(n^{100})$

以降，特に断りがない限り，対数関数 log の底は 2 とする．

注 3.1. オーダー表記を用いると，定理 1.2 より線形探索は $O(n)$ で，定理 1.4 よりユークリッドの互除法は（$x \geq y$ の場合）$O(\log y)$ で，アルゴリズムが終了するということである．

問 3.1. 以下の極限を求めなさい．

- $\lim_{n\to\infty} \dfrac{n \log n}{n^{1.01}}$
- $\lim_{n\to\infty} \dfrac{n^{100}}{1.01^n}$

問 3.2. 以下のうち正しいものを選び出しなさい．

1. $2^{100} = O(1)$
2. $\log n = O(1)$
3. $n + \sqrt{n} + \log n = O(n)$
4. $n \log n - 100n = O(n)$
5. $n\sqrt{n} + n^{1.1} \log n = O(n^2)$
6. $n^{1.001} = O(n \log n)$
7. $n = O(2^{\log n})$
8. $n^3 = O(2^{2 \log n})$
9. $n^{100} = O(1.001^n)$
10. $n^{1.001} = O(2^{\sqrt{\log n}})$
11. $n^{\log n} = O(1.001^{\sqrt{n}})$
12. $1.001^n = O(2^{\sqrt{n}})$

定義 3.2

n を入力長とする．アルゴリズム A が**多項式時間**で終了するとは，ある（正の）定数 $k = O(1)$ が存在して，A の計算時間が $O(n^k)$ であることである．また，アルゴリズム A が**指数時間**で終了するとは，ある（正の）定数 $a = O(1)$ が存在して，A の計算時間が $O(a^n)$ であることである．

例 3.2. n を入力長とする．以下で示された計算時間は多項式時間である．

- 100
- $2n^2 \log n + n(\log n)^{10} + 5$
- $n^{1000} + 2^{100}$

以下で示された計算時間は（多項式時間ではない）指数時間である．

- 2^n
- $100^{n/100}$
- $n^{\log n} + n^3 + 10$

注 3.2. 線形探索，ユークリッドの互除法は多項式時間アルゴリズムである．（線形探索は入力 $x, (a_1, \ldots, a_n)$ に対して $O(n)$，ユークリッドの互除法は入力 x, y に対して[*1]（$x \geq y$ の場合）$O(\log y)$.）一方，充足可能性問題や巡回セールスマン問題に対しては，指数時間アルゴリズムしか発見されていない．

問 3.3. n を入力長とする．以下で示された計算時間のうち多項式時間を選び出しなさい．

1. 2^{100}
2. $\log n$
3. $n\sqrt{n} + n^{1.1} \log n$
4. $n^{1.001}$
5. $n^{100} + 100n$
6. $n^{1.001} - 100n - 100$
7. $n^{\log n}$
8. 1.001^n

[*1] 入力長は（およそ）$\log x + \log y$ である．

第 4 章

探索

探索問題（searching）

- 入力：整数 $x \in \mathbb{Z}$ と整数の列 $a_1, a_2, \ldots, a_n \in \mathbb{Z}$
- 質問：x が $a_1, a_2, \ldots, a_n$ に存在するか？

4.1 線形探索

第 1.1 節を参照．実装については，第 2.1 節を参照．

4.2 二分探索

図 4.1 に二分探索のアルゴリズムを示す．これは，入力 $(a_1, \ldots, a_n)$ が昇順（や降順）になっていた場合の探索手法である．（そうでない場合はそのような探索はできない．）

入力：整数 $x \in \mathbb{Z}$ と整数の列 $a_1, a_2, \ldots, a_n \in \mathbb{Z}$，ただし，$a_1 \leq a_2 \leq \cdots \leq a_n$

- $s = 1, t = n$ として以下を繰り返す．
  1. $s > t$ であれば `NO` を出力して終了する．
  2. $l = \lfloor (s + t)/2 \rfloor$ とする．
  3. $a_l = x$ であれば `YES` を出力して終了する．
  4. $a_l < x$ であれば $s = l + 1$，$a_l > x$ であれば $t = l - 1$ とする．

図 4.1: 二分探索

問 4.1. YES の場合と NO の場合それぞれに対して，ある一つの整数と 7 個の整数からなる昇順列を具体的にあげ，それに対して二分探索のアルゴリズムを適用させなさい．このとき，その入力によるアルゴリズムの動作を説明すること．

定理 4.1. 図 4.1 のアルゴリズム A は探索問題を解く．また，A の計算時間は $O(\log n)$ である．

証明. まず，アルゴリズムの正当性を示す．そのためには以下のことを示せばよい．

$$x \in \{a_1, \ldots, a_n\} \quad \text{のとき} \quad A(x, (a_1, \ldots, a_n)) = \texttt{YES}$$
$$x \notin \{a_1, \ldots, a_n\} \quad \text{のとき} \quad A(x, (a_1, \ldots, a_n)) = \texttt{NO}$$

まず，二つ目を示す．このとき，$\forall i \in [n][a_i \neq x]$ であることから，アルゴリズムより，ステップ 3 の条件は成り立たないので，ステップ 3 で YES を出力して終了することはない．また，$t - s$ の値は単調に減少していくので，いずれはステップ 1 の条件が成り立って，$A(x, (a_1, \ldots, a_n)) = \texttt{NO}$ となる．

次に，一つ目を示す．$\exists i \in [n][a_i = x]$ とする．このとき，$A(x, (a_1, \ldots, a_n)) = \texttt{YES}$ となることを，$n = t - (s - 1)$（データの個数）についての帰納法により示す．そのため，A を以下のように再帰を用いた記述に書き換える．（アルゴリズムとしては全く同じである．）

入力：整数 $x \in \mathbb{Z}$ と整数の列 $a_1, a_2, \ldots, a_n \in \mathbb{Z}$，ただし，$a_1 \leq a_2 \leq \cdots \leq a_n$

- $s = 1, t = n$ として bsearch(s, t) を実行する．

bsearch(s, t)

1. $s > t$ であれば NO を出力して終了する．
2. $l = \lfloor (s + t)/2 \rfloor$ とする．
3. $a_l = x$ であれば YES を出力して終了する．
4. $a_l < x$ であれば $s = l + 1$，$a_l > x$ であれば $t = l - 1$ として，bsearch(s, t) を実行する．

図 4.2: 二分探索：再帰を用いた記述

【初期段階】 $n = 1$ のときは明らか．

問 4.2. その理由を詳細に説明しなさい．

【帰納仮定】 $k \geq 2$ である自然数 $k \in \mathbb{N}$ を任意に固定する．任意の自然数 $n \leq k-1$ について，アルゴリズムが正しく動作するとする．

問 4.3. $k \geq 2$ という条件がある理由を詳細に説明しなさい．

【帰納段階】 上の仮定を用いて，$n = k$ について，アルゴリズムが正しく動作することを示す．$\exists i \in [n][a_i = x]$ とする．$s = 1, t = n$ として，$\texttt{bsearch}(s, t)$ の動作を考える．$s \leq t$（$s = 1, t = n, n = k \geq 2$）であることからステップ 1 の条件は満たされない．$l = \lfloor (s+t)/2 \rfloor$ とする．$a_l = x$ であれば YES を出力して終了するので，アルゴリズムは正しく動作する．そうでないとき，$a_l < x$ の場合，$a_1, \ldots, a_n$ は昇順で並んでいることから，（$\exists i \in [n][a_i = x]$ が満たされているので）$\exists i \in [n] \setminus [\ell][a_i = x]$（$x$ が上位に存在する）が成り立つ．

問 4.4. $a_l > x$ の場合はどういう条件が成り立つか．条件式を示しなさい．

$a_l < x$ のとき，帰納仮定より，$\exists i \in [n] \setminus [\ell][a_i = x]$ の場合は $\texttt{bsearch}(l+1, n)$ は正しく動作する．また，$a_l > x$ のときは（$a_l < x$ のときと同様の理由で）$\texttt{bsearch}(1, l-1)$ も正しく動作する．よって，$n = k$ のときもアルゴリズムは正しく動作することがいえる．以上より，アルゴリズムの正当性が示される．

次に，計算時間を見積もる．アルゴリズムの正当性を示したときと同様，再帰を用いた記述のアルゴリズムの計算時間を見積もる．（アルゴリズムとして全く同じであることから計算時間も同じになる．）$\texttt{bsearch}(1, n)$ の計算時間を見積もればよい．このために，次のように漸化式を立てる．$\texttt{bsearch}(s, t)$（ただし，$t - (s-1) = \alpha$）の実行にかかる計算時間を $T(\alpha)$ とする．再帰を用いた記述より，ある定数 c, c' が存在して，

$$\begin{array}{rcl} T(\alpha) & \leq & T(\lfloor \alpha/2 \rfloor) + c \\ T(1) & \leq & c' \end{array} \tag{4.1}$$

問 4.5. 上の不等式のうち一つ目が成り立つ理由を説明しなさい．特に，単一コスト RAM モデルでは，$c = 10$ としておけば十分である．これら単一時間ですむ箇所をすべてあげなさい．

問 4.6. 上の不等式のうち二つ目が成り立つ理由を説明しなさい．特に，c' と c の大小関係を，アルゴリズムが `YES` を出力して終了する場合と `NO` を出力して終了する場合に分けて説明しなさい．

この漸化式は次のように解くことができる．まず, ある非負整数 k に対して $n = 2^k \geq 1$ であるとする．（$k = \log n.$）このとき，

$$T(n) \leq T(n/2) + c \tag{4.2}$$
$$T(n/2) \leq T(n/2^2) + c \tag{4.3}$$
$$T(n/2^2) \leq T(n/2^3) + c \tag{4.4}$$
$$\vdots$$

上の式において，不等式 (4.3) を不等式 (4.2) へ代入すると，

$$\begin{aligned} T(n) &\leq T(n/2) + c \\ &\leq (T(n/2^2) + c) + c \\ &= T(n/2^2) + 2c. \end{aligned}$$

更に，この不等式に不等式 (4.4) を代入すると，

$$\begin{aligned} T(n) &\leq T(n/2^2) + 2c \\ &\leq (T(n/2^3) + c) + 2c \\ &= T(n/2^3) + 3c. \end{aligned}$$

このように，順次，不等式を代入していけば，

$$\begin{aligned} T(n) &\leq T(n/2) + c \\ &\leq T(n/2^2) + 2c \\ &\leq T(n/2^3) + 3c \\ &\vdots \\ &\leq T(n/2^i) + ic. \end{aligned}$$

$n = 2^k$ より，上の不等式で $i = k$ とおけば，

$$T(n) \ \leq \ T(1) + k \cdot c.$$

更に，$k = \log n$（$\because n = 2^k$），$T(1) \leq c'$（$\because$ 不等式 (4.1)）より，

$$T(n) \ \leq \ c \log n + c' \ = \ O(\log n).$$

一般の自然数 n についても，上の式が成り立つことを帰納法により示す．

問 4.7. 任意の自然数 n について，$T(n) \leq c \log n + c'$ が成り立つことを帰納法により示しなさい．（定数 c, c' は不等式 (4.1) のものに同じ．）

よって，任意の自然数 n について，$T(n) \leq c \log n + c' = O(\log n)$. ■

4.3　二分探索の実装

実装 4.1. 以下で示されたプログラムをもとに，二分探索のアルゴリズムの実装を完成させなさい．（コメントアウトされた部分に，図 4.1 で示された擬似コードを実装する．）更に，何度か実行させて，プログラムの動作確認をしなさい．（配列 a には奇数が昇順に入っている．）

二分探索のプログラム

```
#include <iostream>
using namespace std;

#define NUMBER        1000

int main()
{
    int x, a[NUMBER];

    for (int i=0; i<NUMBER; i++) {
        a[i] = 2*i+1;
    }

    cout << "探索する整数：";
    cin >> x;

    /*
    二分探索のアルゴリズム
    */

    return 0;
}
```

実装 4.2. 以下で示されたプログラムをもとに，線形探索と二分探索のアルゴリズムの実行時間を比較しなさい．（コメントアウトされた部分に線形探索か二分探索を実装して，それぞれ別個の実行ファイルを作成する．）探索する整数 x を色々と変えてみて，二つのアルゴリズムの実行時間の差を確認しなさい．更に，NUMBER を 20000, 30000, ...

と増やしていった場合，実行時間の差の変移を確認しなさい．

実行時間を比較するプログラム

```
#include <iostream>
#include <chrono>
using namespace std;
using namespace chrono;

#define NUMBER        10000

int main()
{
    int x, a[NUMBER];

    for (int i=0; i<NUMBER; i++) {
        a[i] = 2*i+1;
    }

    cout << "探索する整数：";
    cin >> x;

    system_clock::time_point start, end;
    start = system_clock::now();

    /*
    線形探索「または」二分探索
    */

    end = system_clock::now();
    auto diff = end - start;
    cout << duration_cast<microseconds>(diff).count();
    cout << " microsec." << endl;

    return 0;
}
```

第 5 章

データ構造 2

第 2 章では，基本的なデータ構造を学んだ．ここでは，次章以降で学ぶアルゴリズム（特に，ヒープソートや BFS・DFS）に必要となるデータ構造を学ぶ．

5.1 グラフ

定義 5.1

V を**頂点**の集合としたとき，頂点対の集合 $E \subseteq V \times V$ を**辺**の集合という．このとき，$G = (V, E)$ を V 上の**有向グラフ**という．辺の集合 E において，(u, v) と (v, u) を同じ辺とみなすとき，$G = (V, E)$ を**無向グラフ**という．有向グラフの辺を**有向辺**，無向グラフの辺を**無向辺**という．

以降では，特に断らない限り，**グラフ**といった場合，無向グラフを指すものとする．

例 5.1 (グラフ)**.** 以下は，$V = \{1, 2, 3, 4, 5\}$ 上の無向グラフ $G = (V, E)$ である．

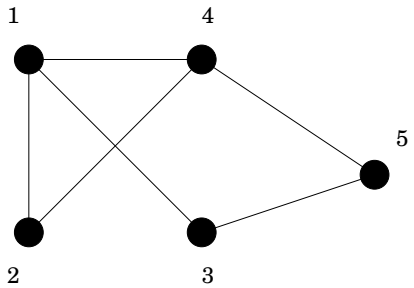


図 5.1: 無向グラフ

このグラフ G の頂点集合 V，辺集合 E は，

$$\begin{aligned} V &= \{1,2,3,4,5\} \\ E &= \{(1,2),(1,3),(1,4),(2,4),(3,5),(4,5)\} \end{aligned}$$

問 5.1. 任意のグラフ $G=(V,E)$ について，$|E| \leq |V|(|V|-1)/2$ である．この事実を示しなさい．

定義 5.2

$G=(V,E)$ をグラフとする．頂点 $u,v \in V$ について $(u,v) \in E$ であるとき，u と v は**隣接する**という．頂点 u に隣接する頂点の集合を $N(u)$ と表記する．つまり，$N(u)=\{v \in V : (u,v) \in E\}$．$u$ の**次数**とは，u の隣接頂点の個数，つまり，$|N(u)|$ のことである．

例 5.2 (隣接頂点，次数)**.** 図 5.1 のグラフ $G=(V,E)$ では，

$$\begin{aligned} N(1) &= \{2,3,4\} \\ N(2) &= \{1,4\} \\ N(3) &= \{1,5\} \\ N(4) &= \{1,2,5\} \\ N(5) &= \{3,4\} \end{aligned}$$

よって，頂点 1,4 の次数は 3，頂点 2, 3, 5 の次数は 2 である．

命題 5.1. $G=(V,E)$ をグラフとする．

$$\sum_{u \in V} |N(u)| \;=\; 2|E|.$$

問 5.2. 上の命題を示しなさい．

$G=(V,E)$ をグラフとする．G を表すデータ構造として，以下の二つがある．

- 隣接行列
- 隣接リスト

例 5.3 (隣接行列，隣接リスト)**.** 図 5.1 の（$V=\{1,2,3,4,5\}$ 上の）グラフを隣接行列

で表すと以下となる．

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

また，隣接リストで表すと以下となる．

$$\begin{array}{lcl} 1 & : & 2, 3, 4 \\ 2 & : & 1, 4 \\ 3 & : & 1, 5 \\ 4 & : & 1, 2, 5 \\ 5 & : & 3, 4 \end{array}$$

問 5.3. $G = (V, E)$ をグラフとする．G を隣接行列で保持した場合，必要なメモリ領域は $O(|V|^2)$．一方，隣接リストで保持した場合は $O(|V| + |E|)$．この事実の理由を説明しなさい．

実装 5.1. 以下で示されたプログラムをもとに，図 5.1 のグラフを隣接行列で保持したもの（二次元配列）から，隣接リストで保持するもの（連結リスト）へ変換するプログラムを完成させなさい．（コメントアウトされた 3 箇所を適切に実装する．）更に，一度，実行させて，プログラムの動作確認をしなさい．

隣接行列と隣接リスト

```
#include <iostream>
#include <list>
using namespace std;

#define N   5

int main()
{
    bool a[N][N] = {
    /* 図 5.1 のグラフ */
    };
    list<int> vList[N];

    for (int i=0; i<N; i++) {
        for (int j=0; j<N; j++) {
            if ( /* 条件 */ ) /* 連結リストの操作 */ ;
        }
    }

    for (int i=0; i<N; i++) {
        cout << i+1 << ": ";
        for (auto itr=vList[i].begin();itr!=vList[i].end();++itr) {
            cout << *itr << " ";
        }
        cout << endl;
    }

    return 0;
}
```

定義 5.3

$G = (V, E)$ をグラフとする．任意の $s, t \in V$ について，s から t への G 上の**経路（パス）**とは，以下を満たす頂点の列 $P = (v_0, v_1, v_2, \ldots, v_k)$ である．

1. $v_0 = s$
2. $v_k = t$
3. $\forall i \in [k][(v_{i-1}, v_i) \in E]$

このとき，**経路の長さ**を k とする．特に，任意の $i, j \in [k] \cup \{0\}$（$i \neq j$）について $v_i \neq v_j$ であるとき，P を**単純経路**という．

例 5.4 (経路)**.** 図 5.2 で示された太線は，図 1.1 (a) のグラフ $G = (V, E)$ 上の，頂点 1 から頂点 7 への（単純）経路である．

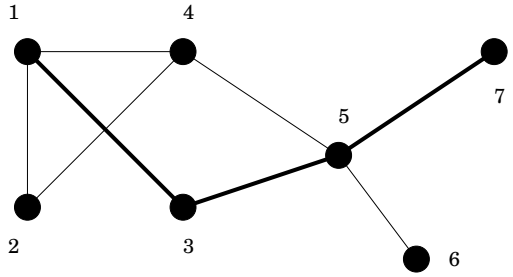


図 5.2: 経路

定義 5.4

$G = (V, E)$ をグラフとする．任意の $k \geq 3$ に対して，$P = (v_1, \ldots, v_k)$ を v_1 から v_k への経路とする．$(v_k, v_1) \in E$ であるとき，$P' = (v_1, \ldots, v_k, v_1)$ を**閉路**という．このとき，**閉路の長さ**を k とする．特に，任意の $i, j \in [k]$（$i \neq j$）について $v_i \neq v_j$ であるとき，P' を**単純閉路**という．

例 5.5 (閉路)**.** 図 5.3 で示された太線は，図 1.1 (a) のグラフ $G = (V, E)$ 上の，頂点 1 を通る（単純）閉路である．

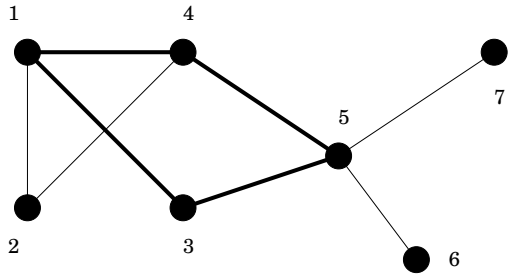


図 5.3: 閉路

以降では，特に断らない限り，経路・閉路といった場合，単純経路・単純閉路を指すものとする．

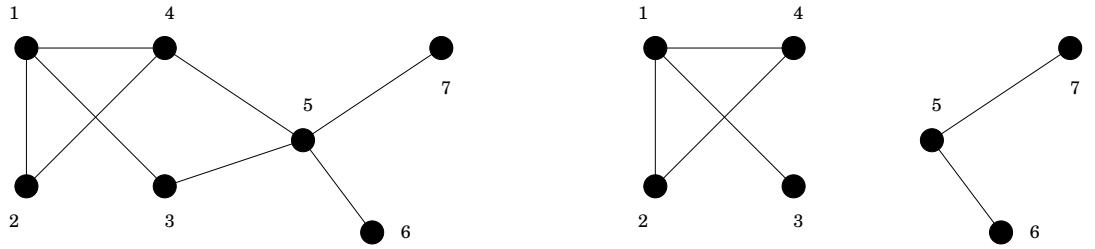
5.2 木と根付き木

定義 5.5

$G = (V, E)$ をグラフとする．頂点 u から頂点 v への経路が存在するとき，u, v は**連結している**という．任意の頂点 $u, v \in V$ について u, v が連結しているとき，G

を**連結グラフ**という．

例 5.6 (連結グラフ)**.** 以下の図の $V = \{1, 2, \ldots, 7\}$ 上のグラフにおいて，左図 (a) は連結，右図 (b) は非連結．



(a) 連結グラフ　　(b) 非連結

図 5.4: 連結・非連結

定義 5.6

$G = (V, E)$ を連結グラフとする．G に閉路が存在しないとき，G を**木**という．

例 5.7 (木)**.** 以下は，$V = \{1, 2, 3, 4, 5\}$ 上の木 $G = (V, E)$ である．

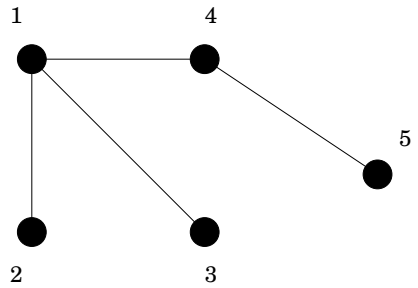


図 5.5: 木

定理 5.2. $G = (V, E)$ を木とする．このとき，

- 任意の頂点 u, v について，u から v への経路は一意である．
- どの辺 $(u, v) \in E$ を除去しても連結でなくなる．
- どの辺 $(u, v) \in (V \times V) \setminus E$ を追加しても唯一の閉路ができる．

命題 5.3. $G=(V,E)$ が木であれば $|E|=|V|-1$.

系 5.4. 任意の連結グラフ $G=(V,E)$ について, $|V|-1\leq|E|$. よって, $|V|=O(|E|)$.

定義 5.7

$G=(V,E)$ を連結グラフとする. 任意の $u,v\in V$ について, u から v への経路の長さの最小を u から v への**距離**といい, $d(u,v)$ と表す.

命題 5.5. $G=(V,E)$ を木, $r\in V$ を任意の頂点とする. このとき, 任意の $(u,v)\in E$ について, $|d(r,u)-d(r,v)|=1$

定義 5.8

$T=(V,E)$ を木, $r\in V$ を T の任意の頂点とする. T の任意の無向辺 $(u,v)\in E$ について, $d(r,u)<d(r,v)$ となるように有向辺 (u,v) を考えた場合（上の命題よりそのような向きは一意である), 「有向木」と頂点 r の対 (T,r) を r を**根**とした**根付き木**という.

T の任意の有向辺 (u,v) について, u を v の**親**, v を u の**子**という. 子をもたない頂点を T の**葉**という. r から葉までの距離の最大を T の**深さ**という.

例 5.8 (根付き木)**.** 以下の図は, 頂点 r を根とした根付き木である.（各有向辺は下向きであり矢印は省略されている.）

定義 5.9

$(T=(V,E),r)$ を根付き木とする. 任意の頂点 $v\in V$ について, $(u,v)\in E$ となる頂点 u の個数を v の**入次数**, $(v,w)\in E$ となる頂点 w の個数を v の**出次数**という.

事実 5.1. $(T=(V,E),r)$ を根付き木とする. 任意の頂点 $v\in V$ について, v の入次数は v の親の個数に, v の出次数は v の子の個数に一致する.

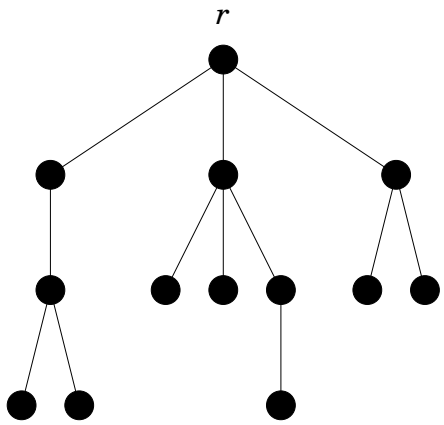
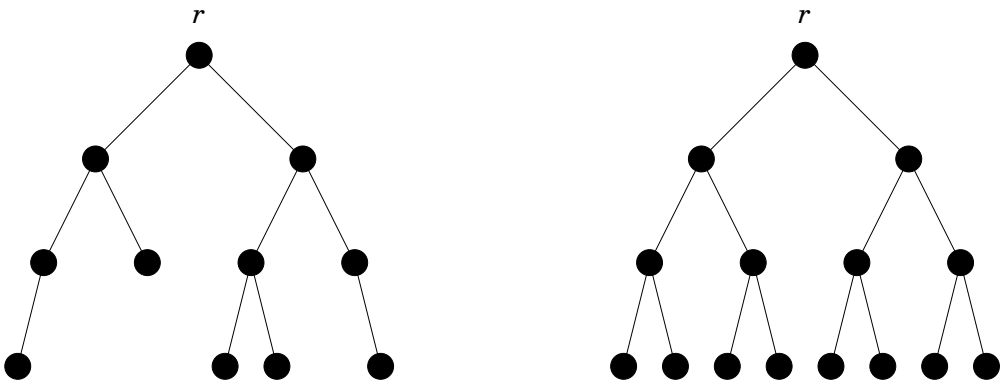


図 5.6: 根付き木

定義 5.10

(T, r) を根付き木とする．葉以外の任意の頂点の出次数が「高々」2 であるとき，T を**二分木**という．更に，任意の葉 u について $d(r, u)$ が一定であるとき，二分木 T を**完全二分木**という．

例 5.9 (二分木)**.** 以下の図 5.7 は r を根とした二分木と完全二分木である．



(a) 二分木　　(b) 完全二分木

図 5.7: 二分木と完全二分木

命題 5.6. $(T = (V, E), r)$ を完全二分木とする．$|V| = n$ のとき，T の深さは $\log(n + 1) - 1$ である．

問 5.4. 上の命題を示しなさい.

第 6 章

整列

整列問題（sorting）

- 入力：整数の列 $a_1, a_2, \ldots, a_n \in \mathbb{Z}$
- 解：$a_1, a_2, \ldots, a_n$ の昇順，つまり，$a_{i_1}, a_{i_2}, \ldots, a_{i_n}$ s.t. $a_{i_1} \leq a_{i_2} \leq \cdots \leq a_{i_n}$

以下，擬似コード中の配列は，配列名を a，配列の大きさを n としたとき，配列の番号は，**1 から n**となっているものとする．つまり，$a[1], a[2], \ldots, a[n]$ に値が入っているものとする．（C++ 言語では，`a[0],a[1],...,a[n-1]` となる．）

6.1 バブルソート

図 6.1 に，バブルソートのアルゴリズムを示す．

入力：整数の列 $a_1, \ldots, a_n \in \mathbb{Z}$

1. a を大きさ n の配列として，それぞれの $i \in [n]$ について $a[i] = a_i$ とする．
2. それぞれの $i \in [n-1]$ について（順次）以下を繰り返す．
   （a）それぞれの $j \in [n-i]$ について（順次）以下を繰り返す．
   - $a[j] > a[j+1]$ であれば $a[j]$ と $a[j+1]$ の値を入れ替える．
3. 配列 a の値を（順次）出力する．

図 6.1: バブルソート

問 6.1. 異なる 7 個の整数の列を具体的にあげ，それに対してバブルソートのアルゴリズムを適用させなさい．このとき，その入力によるアルゴリズムの動作を説明すること．

定理 6.1. 図 6.1 のアルゴリズム A は整列問題を解く．つまり，入力 $a_1, \ldots, a_n$ の A の出力 $A(a_1, \ldots, a_n)$ は，$a_1, \ldots, a_n$ の昇順である．また，A の計算時間は $O(n^2)$ である．

証明. まず，アルゴリズムの正当性を示す．

主張 6.1. 任意の $i \in [n-1]$ について次のことが成り立つ．アルゴリズムのステップ 2 の第 i 回目の繰り返しにおいて，ステップ 2-(a) の繰り返し後，$a[1], \ldots, a[n-i+1]$ の最大のものが $a[n-i+1]$ に移動する．

証明. $a[1], \ldots, a[n-i+1]$ の最大を $a[\ell]$ とおく．（$1 \leq \ell \leq n-i+1$.） ■

問 6.2. この主張の証明を完成させなさい．

主張 6.2. 任意の $i \in [n-1]$ について次のことが成り立つ．アルゴリズムのステップ 2 の第 i 回目の繰り返しにおいて，ステップ 2-(a) の繰り返し後，以下のことが成り立つ．

1. 任意の $j \in [n-i]$ について $a[j] \leq a[n-i+1]$.
2. $a[n-i+1], a[n-i+2], \ldots, a[n]$ は昇順である．

証明. $i \in [n-1]$ についての帰納法により証明する．$i = 1$ のとき，上の主張より，ステップ 2-(a) の繰り返しによって，$a[1], \ldots, a[n]$ から最大のものが $a[n]$ に移動する．これより，主張の二つの条件が成り立つことは明らかである．

問 6.3. 何がどのように明らかなのか，略された証明をそれぞれ示しなさい．

$i = k$ のとき，主張の二つの条件が成り立つとする．$i = k+1$ のとき，ステップ 2 の第 $k+1$ 回目の繰り返しを考える．上の主張より，ステップ 2-(a) の繰り返しによって，$a[1], \ldots, a[n-k]$ から最大のものが $a[n-k]$ に移動する．よって，すべての $j \in [n-k]$ について $a[j] \leq a[n-k]$.（主張の一つ目の条件が示された.）また，帰納仮定より，$a[n-k] \leq a[n-k+1]$.

問 6.4. ここで用いた帰納仮定とはどういう仮定か.

更に,（帰納仮定より）$a[n-k+1],\ldots,a[n]$ は昇順であることから，$a[n-k], a[n-k+1],\ldots,a[n]$ は昇順となる.（主張の二つ目の条件が示された.） ■

この主張に $i = n-1$ を適用すればアルゴリズムの正当性が示される.

問 6.5. なぜ，この主張に $i = n-1$ を適用すればアルゴリズムの正当性が示されるのか，その理由を説明しなさい.

次に，アルゴリズムの計算時間を見積もる．まず，ステップ 1，ステップ 3 にかかる計算時間は，それぞれ $O(n)$ である．よって，ステップ 2 全体にかかる計算時間が $O(n^2)$ であることを示せばよい．まず，それぞれの $i \in [n-1]$ について，ステップ 2-(a) の繰り返し全体にかかる計算時間は $O(n-i)$ である.

問 6.6. この事実（ステップ 2-(a) の繰り返し全体にかかる計算時間は $O(n-i)$）が成り立つ理由を説明しなさい.

つまり，ステップ 2-(a) の繰り返し全体にかかる計算時間は，ある定数 c が存在して $c(n-i)$ 以下である．よって，ステップ 2 全体にかかる計算時間は，高々,

$$\sum_{i\in[n-1]} c(n-i) \;=\; c\cdot \sum_{i\in[n-1]} i \;=\; c\cdot \frac{n(n-1)}{2} \;=\; O(n^2).$$

■

注 6.1. 選択ソート，挿入ソートは，バブルソートと本質的に同じアルゴリズムである.

6.2 クイックソート

図 6.2 に，クイックソートのアルゴリズムを示す.

注 6.2. 図 6.2 のアルゴリズムの `qsort`(a,x,y) （のステップ 2）において，$a[x]$ を**ピボット**という.

入力：整数の列 $a_1, \ldots, a_n \in \mathbb{Z}$

1. a を大きさ n の配列として，それぞれの $i \in [n]$ について $a[i] = a_i$ とする．
2. $\texttt{qsort}(a, 1, n)$ を実行する．
3. 配列 a の値を（順次）出力する．

$\texttt{qsort}(a, x, y)$

1. $x = y$ であればリターン．
2. $i = x$, $j = y$，更に，$p = a[x]$ とする．
3. $i < j$ である限り以下を繰り返す．
   (a) $a[i] < p$ かつ $i < j$ である限り $i{+}{+}$ する．
   (b) $a[j] \geq p$ かつ $i < j$ である限り $j{-}{-}$ する．
   (c) $a[i]$ と $a[j]$ の値を交換する．
4. $z = i = j$ として，以下のうち一方を実行する．
   - $z = x$ であれば，$\texttt{qsort}(a, z + 1, y)$ を実行する．
   - $z \neq x$ であれば，$\texttt{qsort}(a, x, z - 1)$, $\texttt{qsort}(a, z, y)$ を実行する．

図 6.2: クイックソート

問 6.7. 異なる 7 個の整数の列を具体的にあげ，それに対してクイックソートのアルゴリズムを適用させなさい．このとき，その入力によるアルゴリズムの動作を説明すること．

定理 6.2. 図 6.2 のアルゴリズム A は整列問題を解く．つまり，入力 $a_1, \ldots, a_n$ の A の出力 $A(a_1, \ldots, a_n)$ は，$a_1, \ldots, a_n$ の昇順である．また，A の計算時間は $O(n^2)$ である．

証明. まず，アルゴリズムの正当性を示す．

主張 6.3. $x, y \in [n]$ を任意（ただし $x \leq y$）とする．$\texttt{qsort}(a, x, y)$ により，$a[x, \ldots, y]$ がソートされる．（昇順になる．）

証明. $y-x$ の値についての帰納法により示す．$y-x=0$ のとき，つまり，$x=y$ であるとき，$a[x,\ldots,y]$ は要素が一つの配列となっている．$\texttt{qsort}(a,x,y)$ のアルゴリズムよりステップ 1 でリターンされることから，$a[x,\ldots,y]$ がソートされることは明らかである．$y-x\leq k$ のとき，$\texttt{qsort}(a,x,y)$ により $a[x,\ldots,y]$ がソートされるとする．$y-x=k+1$ のときの $\texttt{qsort}(a,x,y)$ の実行を考える．$\texttt{qsort}(a,x,y)$ のステップ 4 の実行直前において（このとき $z=i=j$），任意の $\ell<z$ について $a[\ell]<p$，更に，任意の $\ell\geq z$ について $a[\ell]\geq p$ となっている．

問 6.8. この事実（任意の $\ell<z$ について $a[\ell]<p$，任意の $\ell\geq z$ について $a[\ell]\geq p$）が成り立つ理由を説明しなさい．

ステップ 4 の後者の実行，つまり，$\texttt{qsort}(a,x,z-1)$, $\texttt{qsort}(a,z,y)$ の実行を考える．（このとき，$x+1\leq z\leq y$．前者の場合も同様にして示される．）$z-1-x\leq k$，更に，$y-z\leq k$ より帰納仮定が適用できるので[*1]，$\texttt{qsort}(a,x,z-1)$ の実行により $a[x,\ldots,z-1]$ が，$\texttt{qsort}(a,z,y)$ の実行により $a[z,\ldots,y]$ がソートされる．以上のことから，ステップ 4 の実行後，$a[x,\ldots,y]$ がソートされることが分かる． ■

この主張に $x=1$, $y=n$ を適用すればアルゴリズムの正当性が示される．

次に，アルゴリズムの計算時間を見積もる．$\texttt{qsort}(a,x,y)$ の計算時間が $O((y-x+1)^2)$ であることを示せば十分である．（$x=1,y=n$ とすればよいので．）つまり，ある定数 c が存在して $c(y-x+1)^2$ 以下であることを示す．（c の値は以降で適切に定められる．）

注 6.3. $y-x+1$ の値は，x から y までに並ぶ整数の個数を意味する．

これを $y-x$ の値についての帰納法により示す．$y-x=0$ のとき，つまり，$x=y$ であるとき，$a[x,\ldots,y]$ は要素が一つの配列となっている．$\texttt{qsort}(a,x,y)$ のアルゴリズムよりステップ 1 でリターンされることから，アルゴリズムは定数時間で終了する．よって，その定数時間より c の値を大きくとれば，（$y-x=0$ のとき）計算時間が，

$$c(y-x+1)^2 \;=\; c$$

以下となることが示される．$y-x\leq k$ のとき，$\texttt{qsort}(a,x,y)$ の計算時間が $c(y-x+1)^2$ 以下であるとする．$y-x=k+1$ のときの $\texttt{qsort}(a,x,y)$ の実行を考える．$\texttt{qsort}(a,x,y)$ のステップ 1 からステップ 3 までの計算時間が $O(y-x+1)$ であることは明らかである．

[*1] $x+1\leq z\leq y$ より，それぞれ，$(z-1)-x\leq(y-1)-x=k$, $y-z\leq y-(x+1)=k$.

問 6.9. この事実（計算時間が $O(y-x+1)$ であること）が成り立つ理由を説明しなさい.

つまり，ある定数 c' が存在して $c'(y-x+1)$ 以下である．このとき，一般性を失うことなく $c' \leq c$ であるとする．（もしそうでなければ c の値を c' より大きくとればよい.）ステップ 4 の後者の実行，つまり，$\texttt{qsort}(a, x, z-1)$, $\texttt{qsort}(a, z, y)$ の実行を考える．（このとき，$x+1 \leq z \leq y$. 前者の場合も同様にして示される.）$(z-1)-x \leq k$, $y-z \leq k$ より帰納仮定が適用できるので[*2]，$\texttt{qsort}(a, x, z-1)$ の計算時間が高々 $c(z-1-x+1)^2 = c(z-x)^2$，$\texttt{qsort}(a, z, y)$ の計算時間が高々 $c(y-z+1)^2$ である．よって，$\texttt{qsort}(a, x, y)$ 全体の計算時間は，高々，

$$\begin{aligned} & c'(y-x+1) + c(z-x)^2 + c(y-z+1)^2 \\ = \; & c'(y-x+1) + c\left((z-x)^2 + (y-z+1)^2\right). \end{aligned}$$

問 6.10. 任意の $x, y \in [n]$（$x < y$），任意の $z \in [n]$（$x+1 \leq z \leq y$）に対して，

$$(z-x)^2 + (y-z+1)^2 \;\leq\; (y-x)^2 + 1.$$

この問より，アルゴリズムの計算時間は，

$$\begin{aligned} & c'(y-x+1) + c\left((z-x)^2 + (y-z+1)^2\right) \\ \leq \; & c'(y-x+1) + c((y-x)^2+1) \\ \leq \; & c((y-x+1) + (y-x)^2 + 1) \quad (\because\; c' \leq c) \\ \leq \; & c(y-x+1)^2 \quad (\because\; y-x \geq 1) \end{aligned}$$

■

問 6.11. 上の証明では，クイックソートの「最悪の」計算時間を見積もった．どのような入力であるときに計算時間が最悪となるか説明しなさい．また，その場合に計算時間が $O(n^2)$ となる理由を説明しなさい.

注 6.4. クイックソートは，「平均的」な計算時間が $O(n \log n)$ であることが「理論的」に示されている．注 6.9 を参照.

[*2] $x+1 \leq z \leq y$ より，それぞれ，$(z-1)-x \leq (y-1)-x = k$, $y-z \leq y-(x+1) = k$.

問 6.12. クイックソートの「最良の」計算時間は $O(n \log n)$ である．どのような入力であるときに計算時間が最良（最小）となるか説明しなさい．また，その場合に計算時間が $O(n \log n)$ となる理由を説明しなさい．

6.3 マージソート

図 6.3 に，マージソートのアルゴリズムを示す．

入力：整数の列 $a_1, \ldots, a_n \in \mathbb{Z}$

1. a を大きさ n の配列として，それぞれの $i \in [n]$ について $a[i] = a_i$ とする．
2. $\texttt{msort}(a, 1, n)$ を実行する．
3. 配列 a の値を（順次）出力する．

$\texttt{msort}(a, x, y)$

1. $x = y$ であればリターン．
2. $m = \lceil (x + y)/2 \rceil$
3. $\texttt{msort}(a, x, m-1)$, $\texttt{msort}(a, m, y)$ をそれぞれ実行する．
4. b を大きさ $y - x + 1$ の配列とする．
5. $i = x$, $j = m$ とする．
6. それぞれの整数 $z : 1 \leq z \leq y - x + 1$ について（順次）以下を繰り返す．
   (a) $i = m$ または $j = y + 1$ の場合，以下のうち一方を実行してステップ 7 へ．
      - $i = m$ であれば $b[z, \ldots, y - x + 1] = a[j, \ldots, y]$
      - $j = y + 1$ であれば $b[z, \ldots, y - x + 1] = a[i, \ldots, m - 1]$

   (b) $a[i] < a[j]$ であれば，$b[z] = a[i]$, $i{+}{+}$ とする．
   (c) $a[i] \geq a[j]$ であれば，$b[z] = a[j]$, $j{+}{+}$ とする．
7. $a[x, \ldots, y] = b[1, \ldots, y - x + 1]$ とする．

図 6.3: マージソート

問 6.13. 異なる 7 個の整数の列を具体的にあげ，それに対してマージソートのアルゴリズムを適用させなさい．このとき，その入力によるアルゴリズムの動作を説明すること．

定理 6.3. 図 6.3 のアルゴリズム A は整列問題を解く．つまり，入力 $a_1, \ldots, a_n$ の A の出力 $A(a_1, \ldots, a_n)$ は，$a_1, \ldots, a_n$ の昇順である．また，A の計算時間は $O(n \log n)$ である．

証明. まず，アルゴリズムの正当性を示す．

主張 6.4. $x, y \in [n]$ を任意（ただし $x \leq y$）とする．$\texttt{msort}(a, x, y)$ により，$a[x, \ldots, y]$ がソートされる．（昇順になる．）

証明. $y - x$ の値についての帰納法により示す．$y - x = 0$ のとき，つまり，$x = y$ であるとき，$a[x, \ldots, y]$ は要素が一つの配列となっている．$\texttt{msort}(a, x, y)$ のアルゴリズムよりステップ 1 でリターンされることから，$a[x, \ldots, y]$ がソートされることは明らかである．$y - x \leq k$ のとき，$\texttt{msort}(a, x, y)$ により，$a[x, \ldots, y]$ がソートされるとする．$y - x = k + 1$ のときの $\texttt{msort}(a, x, y)$ の実行を考える．$\texttt{msort}(a, x, y)$ のステップ 3 の実行において，$m - x \leq k$, $y - m + 1 \leq k$ より帰納仮定が適用できるので，$\texttt{msort}(a, x, m - 1)$ の実行により $a[x, \ldots, m - 1]$ が，$\texttt{msort}(a, m, y)$ の実行により $a[m, \ldots, y]$ がソートされる．その状況のもと，ステップ 6 で示されたように，それぞれの先頭（$a[i]$ と $a[j]$）から小さい方を（順に）選んで並べれば，$a[x, \ldots, y]$ がソートされることが分かる． ■

この主張に $x = 1$, $y = n$ を適用すればアルゴリズムの正当性が示される．

次に，アルゴリズムの計算時間を見積もる[*3]．$\texttt{msort}(a, x, y)$ の計算時間が $O((y - x + 1) \log(y - x + 1))$ であることを示せば十分である．（$x = 1, y = n$ とすればよいので．）つまり，ある定数 c, d が存在して $c(y - x + 1) \log(y - x + 1) + d$ 以下であることを示す．（c, d の値は以降で適切に定められる．）

注 6.5. $y - x + 1$ の値は，x から y までに並ぶ整数の個数を意味する．

以降，$y - x + 1$ の値を n と表記する．この表記では，示すべきことは，$\texttt{msort}(a, x, y)$ の計算時間が $cn \log n + d$ 以下であることになる．これを n の値についての帰納法により示す．$n = 1$ のとき，つまり，$x = y$ であるとき，$a[x, \ldots, y]$ は要素が一つの配列となっている．$\texttt{msort}(a, x, y)$ のアルゴリズムよりステップ 1 でリターンされることから，

[*3] 漸化式 $T(n) \leq 2T(\lceil n/2 \rceil) + cn$ を解いて示すこともできる．

アルゴリズムは定数時間で終了する．よって，その定数時間より d の値を大きくとれば，（$n = 1$ のとき）計算時間が，

$$cn \log n + d \ = \ d$$

以下となることが示される[*4]．$n \leq k$ のとき，$\texttt{msort}(a, x, y)$ の計算時間が $cn \log n + d$ 以下であるとする．$n = k+1$ のときの $\texttt{msort}(a, x, y)$ の実行を考える．ステップ 3 を除いた計算時間（の合計）が $O(n)$ であることは明らかである．

問 6.14. この事実（計算時間が $O(n)$ であること）が成り立つ理由を説明しなさい．

つまり，ある定数 c' が存在して $c'n$ 以下である．このとき，一般性を失うことなく，c, c', d について $(c \log 1.5 - c')n \geq d$ が成り立つとする．（もしそうでなければ，c を c', d より十分に大きくとればよい．）ステップ 3 の実行において，帰納仮定より，$\texttt{msort}(a, x, m-1)$ の計算時間が $c\lfloor n/2 \rfloor \log \lfloor n/2 \rfloor + d$，$\texttt{msort}(a, m, y)$ の計算時間が $c\lceil n/2 \rceil \log \lceil n/2 \rceil + d$ である．よって，$\texttt{msort}(a, x, y)$ 全体の計算時間は，（十分大きい n について）高々，

$$\begin{aligned}
&c'n + (c\lfloor n/2 \rfloor \log \lfloor n/2 \rfloor + d) + (c\lceil n/2 \rceil \log \lceil n/2 \rceil + d) \\
&\leq c'n + c(\lfloor n/2 \rfloor + \lceil n/2 \rceil) \log(n/1.5) + 2d \quad (\because\ n \text{ が十分大きい}) \\
&= c'n + cn \log(n/1.5) + 2d.
\end{aligned}$$

問 6.15. n が十分大きいとき，$\log \lfloor n/2 \rfloor \leq \log(n/1.5)$ かつ $\log \lceil n/2 \rceil \leq \log(n/1.5)$ が成り立つ理由を説明しなさい．

よって，$\texttt{msort}(a, x, y)$ 全体の計算時間は，（高々）

$$\begin{aligned}
&c'n + cn \log(n/1.5) + 2d \\
&= c'n + cn \log n - cn \log 1.5 + 2d \\
&= cn \log n + d - ((c \log 1.5 - c')n - d) \\
&\leq cn \log n + d. \quad (\because\ (c \log 1.5 - c')n - d \geq 0)
\end{aligned}$$

■

[*4] $n = 1$ の場合も成り立つように，定数として（c だけでなく）d も必要である．

6.4　ヒープソート

ヒープの構築

定義 6.1

ラベル付き二分木とは，各頂点に値が付いた根付き二分木である．**ヒープ**とは，以下のようなラベル付き二分木である．

1. 頂点の個数を n としたとき，木の深さは $d = \lfloor \log n \rfloor$ である．$d \geq 1$ の場合，深さ $d-1$ までは完全二分木であり，深さ d の葉は木の左部分に詰められている．
2. 頂点 v のラベルを $x_v \in \mathbb{R}$ としたとき，任意の親子頂点対 (u, v) について $x_u \geq x_v$ である．

深さが d の任意のヒープを T とする．T の「頂点の名前」を次のように名付ける．根を 1 として，深さ $i \in [d]$ の頂点を左から順に $2^i, 2^i + 1, \ldots, 2^{i+1} - 1$ とする．

注 6.6. 上の定義において，頂点番号は頂点の「名前」であって，頂点に付けられた**「ラベル」のことではない．**

例 6.1 (ヒープ)**.** 以下の図は，$n = 10$ とした場合のヒープの一例である．二分木の頂点は○で表示されている．頂点の名前は○の上に，頂点のラベルは○の中に表示されている．

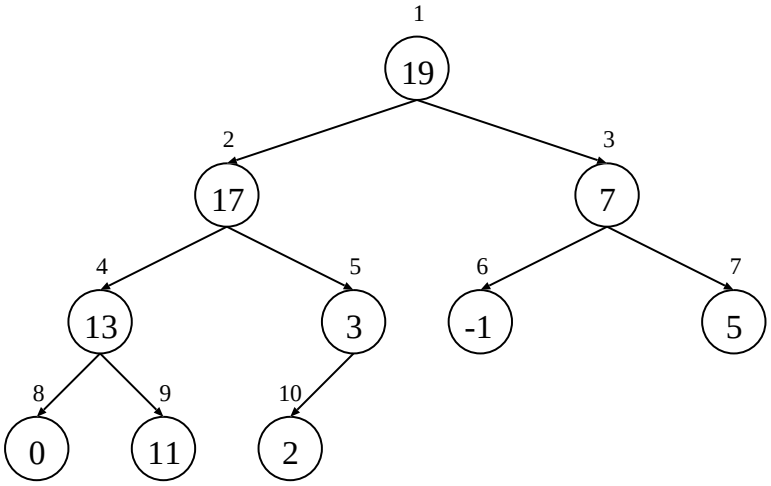


図 6.4: ヒープの例

命題 6.4. T を n 頂点のヒープとする．$i \in [n]$ を T の任意の頂点とする．$p, q \in [n]$ を i の子とする．このとき，$x_i \geq \max\{x_p, x_q\}$.

問 6.16. 上の命題を証明しなさい.

命題 6.5. T を n 頂点のヒープとする．任意の $i \in [n]$ について，頂点 i を根とした T の部分木を T_i とする．（つまり，i から下の部分.）このとき，任意の $i \in [n]$ について，T_i の頂点のラベルの最大は x_i である.

問 6.17. 上の命題を証明しなさい.

入力：整数の列 $a_1, \ldots, a_n \in \mathbb{Z}$

1. T をヒープの条件 1 を満たす n 頂点の二分木とする．（ラベルはついていない.）
2. 任意の $i \in [n]$ について（順次）以下を繰り返す.
   - T の頂点 i のラベル x_i を a_i とする．（$x_i = a_i$.）
3. 任意の $i \in \{n, n-1, n-2, \ldots, 2, 1\}$ について（降順に）以下を繰り返す.
   - `mindown`(i) を実行する.

`mindown`(i)

1. 頂点 i が葉であればリターン.
2. 頂点 i の子を $p, q \in [n]$ とする.
3. x_p, x_q の大小に応じて，以下のうちどちらか一方を実行する.
   - $x_p < x_q$： $x_i < x_q$ なら，i と q のラベルを交換して `mindown`(q) を実行する.
   - $x_p \geq x_q$： $x_i < x_p$ なら，i と p のラベルを交換して `mindown`(p) を実行する.

図 6.5: ヒープの構築

定理 6.6. 図 6.5 のアルゴリズム A によって構築される二分木 T はヒープである．また，A の計算時間は $O(n \log n)$ である．

問 6.18. 異なる１０個の整数の列を具体的にあげ，それに対してヒープの構築のアルゴリズムを適用させなさい．

問 6.19. ヒープの構築のアルゴリズムにおいて，mindown(i) のステップ 3 では，どのようなことが行われるか，簡潔に述べなさい．

証明. まず，アルゴリズムの正当性を示す．任意の $i \in [n]$ について，頂点 i を根とした T の部分木を T_i とする．

主張 6.5. $i \in [n]$ を T の任意の頂点とする．i の子を（あれば）$p, q \in [n]$ とする．T_p, T_q がヒープであれば，mindown(i) の実行後の T_i はヒープになる．

証明. T_i の木の深さ d についての帰納法により示す．$d = 0$ のとき，（i は葉であるので）mindown(i) のアルゴリズムよりステップ 1 でリターンされることから，T_i がヒープになることは明らかである．$d = k$ のとき，主張が成り立つとする．$d = k + 1$ のとき，mindown(i) の実行を考える．一般性を失うことなく，$x_p < x_q$ とする．（$x_p \geq x_q$ の場合も同様にして示される．）$x_i \geq x_q$ であれば，$x_i \geq \max\{x_p, x_q\}$ であり，（仮定より）T_p, T_q がヒープであることから，T_i は（既に）ヒープとなっている．（アルゴリズムはステップ 3 で（何もすることなく）リターンして終了する．）$x_i < x_q$ のとき，mindown(i) のアルゴリズムより，i と q のラベルが交換され mindown(q) が実行される．mindown(q) 実行後の頂点 i, p, q のラベルを，それぞれ y_i, y_p, y_q とする．（mindown(q) 実行前の頂点 i, p, q のラベルが x_i, x_p, x_q.）i と q のラベルが交換されたことから $y_i = x_q$．T_p のラベルは変更されていないことから $y_p = x_p$．これら二つ（と仮定 $x_p < x_q$）より，

$$y_i \;=\; x_q \;>\; x_p \;=\; y_p.$$

よって，$(*)$ $y_i > y_p$．また，T_q がヒープであったことから，命題 6.5（と $x_i < x_q$）より，$x_q \geq y_q$.

問 6.20. この事実（$x_q \geq y_q$）が成り立つ理由を説明しなさい.

これら二つより,

$$y_i \;=\; x_q \;\geq\; y_q$$

よって，(**) $y_i \geq y_q$. つまり，(*), (**) より，$y_i \geq \max\{y_p, y_q\}$. また,

- T_p はヒープである.（仮定より）
- T_q はヒープである.（帰納仮定より）

問 6.21. T_p, T_q がヒープである根拠が，それぞれ「仮定より」と「帰納仮定より」と異なる理由を説明しなさい.

以上より，T_i はヒープである. ■

主張 6.6. アルゴリズムのステップ 3 の実行において，任意の $i \in [n]$ について，`mindown`(i) 実行後の T_i はヒープである.

証明. i についての（降順の）帰納法により示す. $i = n$ のとき，頂点 i は T の葉であることから，`mindown`(i) のアルゴリズムよりステップ 1 でリターンされることから，T_i がヒープになることは明らかである. 任意の $i > k$ について主張が成り立つとする.（降順の帰納法.）$i = k$ の `mindown`(i) の実行を考える. 頂点 i が葉のとき，($i = 0$ のときと同様）T_i がヒープになることは明らかである. そうでないとき，一般性を失うことなく，i は二つの子をもち，それらの子を $p, q \in [n]$ とする. 帰納仮定（とアルゴリズム）より，T_p, T_q はヒープである.

問 6.22. この事実（T_p, T_q がヒープになる）が成り立つ理由を説明しなさい.

よって，主張 6.5 より，T_i はヒープとなる. ■

この主張に $i = n$ を適用すれば，アルゴリズムの正当性が示される.

次に，アルゴリズムの計算時間を見積もる. アルゴリズムのステップ 3 の計算時間が $O(n \log n)$ であることを示せば十分である. ステップ 3 において，任意の $i \in [n]$ について，`mindown`(i) を実行した場合，再帰関数 `mindown` の呼び出し回数は，（それ自身を含めて）高々 $\lfloor \log n \rfloor + 1$ である.

問 6.23. この事実（呼び出し回数が高々 $\lfloor \log n \rfloor + 1$）が成り立つ理由を説明しなさい．（なぜ，＋1 が必要となるのか．）

よって，`mindown` のアルゴリズム自体の計算時間は $O(1)$ であることから，アルゴリズム（のステップ 3）全体の計算時間は，ある定数 c に対して，高々，

$$\sum_{i \in [n]} c(\lfloor \log n \rfloor + 1) \ \leq \ cn \log n + cn \ = \ O(n \log n).$$

■

注 6.7. 計算時間の「解析」を工夫すれば，図 6.5 のアルゴリズム A の計算時間は $O(n)$ となる[*5]．（アルゴリズムを改良しなくても！）

ヒープソート

図 6.6 に，ヒープソートのアルゴリズムを示す．

入力：整数の列 $a_1, \ldots, a_n \in \mathbb{Z}$

1. 図 6.5 のアルゴリズムを実行する．（それによって構築されたヒープを T とする．）
2. 任意の $i \in \{n, n-1, n-2, \ldots, 2, 1\}$ について（降順に）以下を繰り返す．
   （a）頂点 1 と頂点 i のラベル（x_1 と x_i）を交換する．
   （b）T から頂点 i を削除して，`mindown`(1) を実行する．
3. 頂点 1 から頂点 n のラベル（$x_1, \ldots, x_n$）を（順次）出力する．

図 6.6: ヒープソート

問 6.24. 異なる１０個の整数の列を具体的にあげ，それに対してヒープソートのアルゴリズムを適用させなさい．このとき，その入力によるアルゴリズムの動作を説明すること．

[*5] アルゴリズムの計算時間は，`mindown` の呼び出しによって下に移動した頂点の移動回数となる．（上に移動した頂点を考慮する必要はない．なぜ？）深さ h の頂点の個数は高々 2^h である．よって，移動回数の合計は，高々，$\sum_{h \in [\log n - 1] \cup \{0\}} (\log n - h) 2^h = \sum_{i \in [\log n]} i \cdot (n/2^i) = O(n)$.

定理 6.7. 図 6.6 のアルゴリズム A は整列問題を解く．つまり，入力 $a_1, \ldots, a_n$ の A の出力 $A(a_1, \ldots, a_n)$ は，$a_1, \ldots, a_n$ の昇順である．また，A の計算時間は $O(n \log n)$ である．

証明. まず，アルゴリズムの正当性を示す．アルゴリズムのステップ 1 の実行後，定理 6.6 より，T は n 頂点のヒープになっている．任意の $i \in [n]$ について，アルゴリズムのステップ 2 の i についての実行前のラベル付き二分木を T_i とする．（T_i の根は 1 である．$T_n = T, T_0 = \emptyset$.）

主張 6.7. 任意の $i \in [n]$ について，T_i はヒープである．

証明. i についての（降順の）帰納法により示す．$i = n$ のときは，T がヒープであることから明らかである．帰納段階は，主張 6.5 から証明される[*6].

問 6.25. 帰納段階が，主張 6.5 からどのように示されるか説明しなさい．

■

主張 6.8. 任意の $i \in [n]$ について，頂点 i から頂点 n のラベル（$x_i, \ldots, x_n$）は昇順である．

証明. 任意の $i \in [n-1]$ について，$x_i \leq x_{i+1}$ を示せばよい．任意の $i \in [n-1]$ について，T_i の根のラベルが x_i となることから，T_i の根のラベルが T_{i+1} の根のラベル以下であることを示せばよい．これは，T_{i+1} がヒープであり，T_i の i 個のラベルは T_{i+1} の根以外の i 個のラベルと同一であることから示される． ■

この主張に $i = n$ を適用すればアルゴリズムの正当性が示される．

次に，アルゴリズムの計算時間を見積もる．まず，ステップ 1 にかかる計算時間は，定理 6.6 より，$O(n \log n)$ である．また，ステップ 2 において，それぞれの繰り返しにかかる計算時間は，`mindown`(1) にかかるかかる計算時間が $O(\log n)$ である（問 6.23 を参照）ことから，$O(\log n)$ である．最後に，ステップ 3 にかかる計算時間は $O(n)$ である．以上より，アルゴリズム全体の計算時間は，ある定数 c, c', c'' に対して，高々，

$$cn \log n + n \cdot (c' \log n) + c''n \;=\; O(n \log n).$$

■

[*6] ここでの T_i と主張 6.5 での T_i は定義が異なる．

6.5　整列アルゴリズムの実装

実装 6.1. 以下の図 6.7 で示されたプログラムをもとに，本章で学んだ四つの整列アルゴリズムを実装しなさい．（コメントアウトされた部分に，各整列アルゴリズムを実装する．）更に，何度か実行させて，プログラムの動作確認をしなさい．

1. バブルソート：図 6.1 を実装する．
2. クイックソート：図 6.2 を実装する．
3. マージソート：図 6.3 を実装する．
4. ヒープソート：図 6.6（及び 図 6.5）を実装する．ヒープの二分木は以下のように配列を用いて模倣できる．`a[i]` の二つの子は `a[2*i+1]`, `a[2*i+2]` となる．逆に，`a[i]` の親は `a[(i-1)/2]` となる．この配列番号は，C++ 言語での（0 から始まる）番号である．（擬似コードの 1 から始まる番号**でない**．）

注 6.8. 擬似乱数生成（入力される整数のランダム生成）には，メルセンヌツイスターを用いる．簡易的に用いる `rand()` 関数は乱数精度（周期の長さなど）がよくない．

実装 6.2. 入力される整数の個数 `NUMBER` を，1000, 2000, 3000 と増やしていった場合，四つの整列アルゴリズムの実行時間の差の変移を確認しなさい．このとき，整列前後の並びを出力させる部分を削除（またはコメントアウト）しておく必要がある．（そうしなければ，膨大な個数の整数が画面出力されてしまう．プログラムの動作は先の実装で確認済みである．ここでは，実行時間が分かればよい．）

注 6.9. 適切に実装した場合，クイックソートが最も高速になることが知られている．最悪時の計算時間は，マージソートやヒープソートに比べて大きいが，そのような入力列はかなり稀であり，（全例題上の）平均的な計算時間が $O(n \log n)$ であることも「理論的」に示されている．また，実装が（マージソートやヒープソートに比べて）単純であることも，高速になる（大きな）要因である．

整列アルゴリズムの実装

```
#include <iostream>
#include <random> // メルセンヌツイスターを用いるためのライブラリ
#include <chrono> // 時間計測のためのライブラリ
using namespace std;
using namespace chrono;

#define NUMBER  10   // 入力される整数の個数
#define MAX_NUM 1000 // 入力される整数の最大値

int main()
{
    int a[NUMBER];
    mt19937 mt{random_device{}()};
    uniform_int_distribution<int> dist(1,MAX_NUM);

    for (int i=0; i<NUMBER; i++) { // 入力される整数のランダム生成
        a[i] = dist(mt);
    }
    for (int i=0; i<NUMBER; i++) { // 整列前の並びの出力
        cout << a[i] << " ";
    }
    cout << endl;

    system_clock::time_point start, end;

    start = system_clock::now(); // 整列開始

    /*
    整列アルゴリズム
    */

    end = system_clock::now(); // 整列終了

    auto diff = end - start;
    cout << duration_cast<microseconds>(diff).count();
    cout << " microsec." << endl;

    for (int i=0; i<NUMBER; i++) { // 整列後の並びの出力
        cout << a[i] << " ";
    }
    cout << endl;

    return 0;
}
```

図 6.7: 整列アルゴリズムの実装

第 7 章

走査

到達可能性問題（reachability）

- 入力：グラフ $G = (V, E)$, $s, t \in V$
- 質問：s から t への経路が存在するか？

7.1 幅優先探索とその応用

図 7.1 に，幅優先探索（BFS: Breadth-First-Search）のアルゴリズムを示す．

入力：グラフ $G = (V, E)$, $s, t \in V$

1. s を既訪問，$V \setminus \{s\}$ を未訪問として，Q を s だけからなるキュー（queue）とする．
2. $Q \neq \emptyset$ である限り以下を繰り返す．
   (a) Q からデキューしてその要素を u とする．
   (b) N_u のうち未訪問の要素 $U \subseteq N_u$ を既訪問にする．
   (c) U の頂点を Q にエンキューする．
3. t が既訪問なら `YES` を，そうでないなら `NO` を出力する．

図 7.1: 幅優先探索

注 7.1. アルゴリズムのステップ 2 でデキューしたとき，$u = t$ となった時点で `YES` を出力して終了してもよい．（「プログラム的」には速くなる[*1]．）

[*1] 「アルゴリズム的」という最悪時の計算時間を考えた場合は変わらない．

問 7.1. １０頂点上の適当なグラフ（と始点・終点）を具体的にあげ，それに対して幅優先探索のアルゴリズムを適用させなさい．このとき，その入力によるアルゴリズムの動作を説明すること．

定理 7.1. 幅優先探索のアルゴリズム A は到達可能性問題を解く．また，A の計算時間は $O(|V|+|E|)$ である．（G が連結グラフであれば $O(|E|)$.）

証明. まず，アルゴリズムの正当性を示す．そのためには以下のことを示せばよい．

$$\begin{array}{lcl} s \text{ から } t \text{ への経路が存在する} & : & A(G=(V,E),s,t) = \texttt{YES} \\ s \text{ から } t \text{ への経路が存在しない} & : & A(G=(V,E),s,t) = \texttt{NO} \end{array}$$

$X \subseteq V$ を s から到達可能な頂点の集合，$Y \subseteq V$ をアルゴリズム終了後に既訪問となっている頂点の集合とする．

主張 7.1. $X = Y$.

証明. $(X \subseteq Y)$ $x \in X$ を任意とする．（$x \in Y$ を示せばよい.）これは，$s = v_0 \to v_1 \to v_2 \to \cdots \to v_k = x$ という経路が存在することを意味する．このとき，$v_k \in Y$ であることを示せばよい．これを $i \in [k] \cup \{0\}$ についての帰納法により示す．$i = 0$ のとき，$s \in Y$ は明らかである．次に，$v_i \in Y$ であるとする．この帰納仮定は，ある時点で（つまり，初めて v_i が既訪問になったとき）エンキューされたことを意味する．アルゴリズムより，$Q \neq \emptyset$ である限りデキューされ続けることから，いずれは（デキューしたとき）$u = v_i$ となる．$v_{i+1} \in N_{v_i}$ より，v_{i+1} が未訪問である場合には既訪問になる．（つまり，$v_{i+1} \in Y$.）

$(X \supseteq Y)$ アルゴリズムのステップ 2 の繰り返しが k 回実行されたとする．任意の $i \in [k] \cup \{0\}$ について，$Y_i \subseteq V$ を，ステップ 2 の第 i 回目の繰り返し終了後，既訪問である頂点の集合とする．（よって，$Y_0 = \{s\}, Y_0 \subseteq Y_1 \subseteq \cdots \subseteq Y_k = Y$.）以下，任意の $i \in [k] \cup \{0\}$ について $Y_i \subseteq X$ を示す．これを，ステップ 2 の繰り返し回数 $i \in [k] \cup \{0\}$ についての帰納法により示す．まず，$i = 0$ のとき，$Y_0 = \{s\}$ であることから，$Y_0 \subseteq X$ である．（$s \in X$ であるので.）次に，$i = j$ のとき $Y_j \subseteq X$ であるとする．ステップ 2 の $j+1$ 回目の繰り返しにおいて，デキューされた要素を u として，エンキューされる頂点の集合を U とする．このとき，$u \in Y_j, Y_{j+1} = Y_j \cup U$．よって，$U \subseteq N_u$ より，$U \subseteq X$.

問 7.2. $U \subseteq X$ となる理由を説明しなさい.

この主張と $Y_j \subseteq X, Y_{j+1} = Y_j \cup U$ より $Y_{j+1} \subseteq X$. 以上より，$Y_k \subseteq X$ となる. ■

この主張より，X の頂点が既訪問となり $V \setminus X$ の頂点が未訪問となる．これより，アルゴリズムの正当性が示される.

次に，計算時間を見積もる．ステップ 1 とステップ 3 にかかる計算時間はそれぞれ $O(|V|), O(1)$ である．以降，ステップ 2 にかかかる計算時間を見積もる．ステップ 2 での主な操作は以下である.

- キューの操作（デキュー・エンキュー）.
- N_u の特定（u の隣接頂点の走査）.

それぞれについて，$Q = \emptyset$ になるまでにかかった「合計の」計算時間をも積もる.

主張 7.2. 任意の頂点についてエンキューされるのは高々 1 回である．よって，デキューされるのも高々 1 回である.

問 7.3. 上の主張の理由を説明しなさい.

まず，キューの操作について，一つの頂点につきにデキュー・エンキューにかかる時間は $O(1)$ であることから，上の主張より，キューの操作にかかる計算時間は合計で $O(|V|)$ となる．次に，N_u の特定について，任意の頂点 u について u がデキューされるのは高々 1 回であり，一つの頂点 u につき N_u の特定にかかる時間は $O(|N_u|)$ である．（グラフは隣接リストで保持されているので.）よって，隣接頂点の走査にかかる計算時間の合計は，ある定数 c が存在して高々 $c \cdot \sum_{u \in V} |N_u|$. 命題 5.1 より，これは $c \cdot 2|E| = O(|E|)$ である．よって，ステップ 2 にかかる計算時間の合計は， $O(|V|) + O(|E|) = O(|V| + |E|)$ となる.

以上より，幅優先探索の計算時間は $O(|V|) + O(|E|) = O(|V| + |E|)$ となる．（G が連結グラフであれば，系 5.4 より $O(|E|)$ となる.） ■

問 7.4. グラフ $G = (V, E)$ が連結である場合，幅優先探索において，ステップ 2 の繰り返し回数はいくつか.

ＢＦＳの応用

最短経路探索問題（shortest path search）

- 入力：グラフ $G = (V, E)$，始点 $s \in V$，終点 $t \in V$
- 解：s から t への最短経路

この問題は，最短経路を出力する「関数問題」である．（到達可能性問題は「決定問題」である．）図 7.2 に，最短経路探索のアルゴリズムを示す．（各辺の距離が単一である場合のアルゴリズムである．）

入力：グラフ $G = (V, E)$, $s, t \in V$

1. s を既訪問，$V \setminus \{s\}$ を未訪問として，Q を s だけからなるキュー（queue）とする．
2. 任意の頂点 $u \in V$ について関数 $p(u)$ を未定義とする．（p が最短経路を指し示す．）
3. $Q \neq \emptyset$ である限り以下を繰り返す．
   （a）Q からデキューしてその要素を u とする．
   （b）$u = t$ なら関数 p を出力して終了する．
   （c）N_u のうち未訪問の要素 $U \subseteq N_u$ を既訪問にする．
   （d）それぞれの $v \in U$ について $p(v) = u$ と定義する．
   （e）U の頂点を Q にエンキューする．
4. `NO` を出力する．

図 7.2: 最短経路探索

定理 7.2. 最短経路探索のアルゴリズム A は最短経路探索問題を解く．A の計算時間は $O(|V| + |E|)$ である．（G が連結グラフであれば $O(|E|)$．）

7.2　深さ優先探索とその応用

図 7.3 に，深さ優先探索（DFS: Depth-First-Search）のアルゴリズムを示す．

入力：グラフ $G = (V, E)$, $s, t \in V$

1. s を既訪問，$V \setminus \{s\}$ を未訪問とする．
2. $\texttt{dfs}(s)$ を実行する．
3. t が既訪問なら YES を，そうでないなら NO を出力する．

$\texttt{dfs}(u)$

1. それぞれの $v \in N_u$ について以下を実行する．
   - v が未訪問なら v を既訪問にして $\texttt{dfs}(v)$ を実行する．

図 7.3: 深さ優先探索

注 7.2. $\texttt{dfs}(u)$ において，$u = t$ であった時点で YES を出力して終了してもよい．（「プログラム的」には速くなる[*2]．）

問 7.5. １０頂点上の適当なグラフ（と始点・終点）を具体的にあげ，それに対して深さ優先探索のアルゴリズムを適用させなさい．このとき，その入力によるアルゴリズムの動作を説明すること．

定理 7.3. 深さ優先探索のアルゴリズム A は到達可能性問題を解く．また，A の計算時間は $O(|V| + |E|)$ である．（G が連結グラフであれば $O(|E|)$．）

証明. まず，アルゴリズムの正当性を示す．そのためには以下のことを示せばよい．

$$\begin{array}{lcl} s \text{ から } t \text{ への経路が存在する} & : & A(G = (V, E), s, t) = \texttt{YES} \\ s \text{ から } t \text{ への経路が存在しない} & : & A(G = (V, E), s, t) = \texttt{NO} \end{array}$$

$X \subseteq V$ を s から到達可能な頂点の集合，$Y \subseteq V$ をアルゴリズム終了後に既訪問となっている頂点の集合とする．

主張 7.3. $X = Y$

証明. $(X \subseteq Y)$ $x \in X$ を任意とする．（$x \in Y$ を示せばよい．）これは，$s = v_0 \to v_1 \to$

[*2] 「アルゴリズム的」という最悪時間を考えた場合は変わらない．

$v_2 \to \cdots \to v_k = x$ という経路が存在することを意味する．このとき，$v_k \in Y$ であることを示せばよい．これを $i \in [k] \cup \{0\}$ についての帰納法により示す．$i = 0$ のとき，$s \in Y$ は明らかである．次に，$v_i \in Y$ であるとする．この帰納仮定は，ある時点で（つまり，初めて v_i が既訪問になったとき）$\texttt{dfs}(v_i)$ が呼び出されたことを意味する．任意の頂点 u について $\texttt{dfs}(u)$ は有限時間内に終了することから，$\texttt{dfs}(v_i)$ のステップ 1 の繰り返しにおいて，いずれは（$v_{i+1} \in N_{v_i}$ より）$v = v_{i+1}$ となり，v_{i+1} が未訪問である場合には既訪問になる．（つまり，$v_{i+1} \in Y$.）

問 7.6. 任意の頂点 u について $\texttt{dfs}(u)$ が有限時間内に終了する理由を説明しなさい．

$(X \supseteq Y)$ $\texttt{dfs}(u)$ が（アルゴリズム全体を通して）k 回呼び出されたとする．任意の $i \in [k]$ について，$Y_i \subseteq V$ を，第 i 回目の $\texttt{dfs}(u)$ の呼び出し直前に既訪問である頂点の集合とする．（よって，$Y_1 = \{s\}$, $Y_1 \subseteq Y_2 \subseteq \cdots \subseteq Y_k = Y$．$k$ 回目の呼び出しで新たに既訪問となる頂点はないことから $Y_k = Y$.）以下，任意の $i \in [k]$ について $Y_i \subseteq X$ を示す．これを，$\texttt{dfs}(u)$ の呼び出し回数 $i \in [k]$ についての帰納法により示す．まず，$i = 1$ のとき，$Y_1 = \{s\}$ であることから，$Y_1 \subseteq X$ である．（$s \in X$ であるので.）次に，$i = j$ のとき $Y_j \subseteq X$ であるとする．第 j 回目の呼び出しが $\texttt{dfs}(u)$，第 $j+1$ 回目の呼び出しが $\texttt{dfs}(v)$ であったする．このとき，$u \in Y_j$, $Y_{j+1} = Y_j \cup \{v\}$．よって，$v \in N_u$ より，$v \in X$.

問 7.7. $v \in X$ となる理由を説明しなさい．

この主張と $Y_j \subseteq X$, $Y_{j+1} = Y_j \cup \{v\}$ より $Y_{j+1} \subseteq X$．以上より，$Y_k \subseteq X$ となる．■

この主張より，X の頂点が既訪問となり $V \setminus X$ の頂点が未訪問となる．これより，アルゴリズムの正当性が示される．

次に，計算時間を見積もる．ステップ 1 とステップ 3 にかかる計算時間はそれぞれ $O(|V|)$, $O(1)$ である．以降，ステップ 2 にかかかる計算時間を見積もる．

主張 7.4. 任意の頂点 $u \in V$ について $\texttt{dfs}(u)$ が呼び出されるのは高々 1 回である．

問 7.8. 上の主張の理由を説明しなさい．

$\texttt{dfs}(u)$ での主な操作は以下である．

- N_u の特定（u の隣接頂点の走査）.

上の主張より，任意の頂点 u について `dfs`(u) が呼び出されるのは高々 1 回であり，一つの頂点 u につき N_u の特定にかかる時間は $O(|N_u|)$ である．（グラフは隣接リストで保持されているので．）よって，ステップ 2 にかかる計算時間の合計は，ある定数 c が存在して高々 $c \cdot \sum_{u \in V} |N_u|$．命題 5.1 より，これは $c \cdot 2|E| = O(|E|)$ となる．

以上より，深さ優先探索の計算時間は $O(|V|) + O(|E|) = O(|V| + |E|)$ となる．（G が連結グラフであれば，系 5.4 より $O(|E|)$ となる．） ■

問 7.9. グラフ $G = (V, E)$ が連結である場合，深さ優先探索において，再帰関数 `dfs`(u) は（合計で）何回呼び出されるか．

ＤＦＳの応用

トポロジカルソート問題（topological sort）

- 入力：有向グラフ $G = (V, E)$
- 解：$[n]$ 上の順列 $\sigma : [n] \to [n]$，つまり，$v_{\sigma(1)}, v_{\sigma(2)}, \ldots, v_{\sigma(n)}$ s.t. すべての有向辺 $(v_i, v_j) \in E$ に対して，$\sigma(i) < \sigma(j)$

7.3 BFS・DFS の実装

実装 7.1. 以下の図 7.4 で示されたプログラムをもとに，BFS と DFS を実装しなさい．（コメントアウトされた部分に，BFS・DFS を実装する．）ただし，BFS はキューを用いて，DFS はスタックを用いて（再帰関数でなく），それぞれ実装しなさい．（始点は `0`，終点は `N-1` とする．）更に，何度か実行させて，プログラムの動作確認をしなさい．

1. BFS：図 7.1 を実装する．このとき，`xxx` を `queue` に，`yyy` を `bfs` にする．
2. DFS：図 7.3 を実装する．このとき，`xxx` を `stack` に，`yyy` を `dfs` にする．

なお，隣接頂点の走査は隣接リストにて．隣接配列でなく！（第 5 章の実装を参照．）

BFS・DFS の実装

```
#include <iostream>
#include <random>
#include <list>
#include <xxx>
using namespace std;

#define N   10

list<int> vList[N];
bool visit[N]={0};

bool yyy()
{
    xxx<int> data;

    /*
    BFS「または」DFS
    隣接頂点の走査は vList にて.
    */

    return 0;
}

int main()
{
    bool a[N][N]={0};
    mt19937 mt{random_device{}()};
    uniform_int_distribution<int> dist(0,1);

    for (int i=0; i<N; i++) // グラフのランダム生成
        for (int j=i+1; j<N; j++) {
            a[i][j] = dist(mt);
            a[j][i] = a[i][j];
        }

    for (int i=0; i<N; i++) // 隣接リストへの変換
        for (int j=0; j<N; j++)
            if (a[i][j]==1) vList[i].push_back(j);

    if (yyy()) cout << "YES" << endl;
    else cout << "NO" << endl;

    return 0;
}
```

図 7.4: BFS・DFS の実装

問の略解

1. アルゴリズムとは

1.
   - 素数性判定問題
     - 入力：自然数 $n \in \mathbb{N}$
     - 質問：n は素数か？
   - 略.
2.
   - 最短経路探索問題
     - 入力：グラフ $G = (V, E)$，始点 $s \in V$，終点 $t \in V$
     - 解：s から t への最短経路
   - 略.
3. $\exists i \in [n][a_i = x]$ であれば，繰り返し回数が n 以下となるので.
4. 最悪の計算時間を見積もるので，$\forall i \in [n-1][a_i \neq x]$ かつ $a_n = x$ である入力 $x, (a_1, \ldots, a_n)$ の場合の計算時間を見積もる.（$3n + 1$ となる.）
5. 情報基礎のテキスト「第 6 章：整数，ユークリッドの互除法」を参照.
6. $r_{k+1} = 0$.
7. $2 \log y + 2$.（$\because 1 \leq r_k \leq \cdots \leq r_1/2^{(k-1)/2} \leq y/2^{(k-1)/2}$ より $k \leq 2 \log y + 1$.）
8.（およそ）$\log x + \log y$.

2. データ構造

1. n 番目.
2. 1 番目と n 番目.
3. 1 番目と n 番目.
4. n 回.
5. 1 回.

3. オーダー表記

1. ともに O .
2. 1, 3, 5, 7, 9, 11
3. 1, 2, 3, 4, 5, 6

4. グラフ

1. n を頂点の個数としたとき，辺の個数の最大は ${}_nC_2 = n(n-1)/2$ であるので.
2. 任意の辺 $(a, b) \in E$ は，$\sum_{u \in V} |N_u|$ の中で，a と b のちょうど 2 回数え上げられる.
3. 略.
4. 深さを k としたとき，$n = 2^{k+1} - 1$ であることから.

5. 探索

1. - YES の場合：$x = 2$, $(-1, 0, 2, 3, 5, 5, 7)$ に対するアルゴリズムの動作は次のようである．まず，$s = 1, t = 7$ とされ，繰り返しが実行される．$l = 4$ となり，$a_4 = 3 > x$ より，$s = 1, t = 3$ となる．次に，$l = 2$ となり，$a_2 = 0 < x$ より，$s = 3, t = 3$ となる．次に，$l = 3$ となり，$a_3 = 2 = x$ より，YES が出力され終了する.
   - NO の場合：$x = 4$, $(-1, 0, 2, 3, 5, 5, 7)$ に対するアルゴリズムの動作は次のようである．まず，$s = 1, t = 7$ とされ，繰り返しが実行される．$l = 4$ となり，$a_4 = 3 < x$ より，$s = 5, t = 7$ となる．次に，$l = 6$ となり，$a_6 = 5 > x$ より，$s = 5, t = 5$ となる．次に，$l = 5$ となり，$a_5 = 5 > x$ より，$s = 5, t = 4$ となる．最後に，$s > t$ となり NO が出力され終了する.
2. $s = t = 1$ なので.
3. $n \leq k - 1$ についての仮定なので.
4. $\exists i \in [l - 1][a_i = x]$.
5. $T(\alpha)$ は，再帰呼び出し bsearch(s, t) の実行にかかる時間と，単一時間ですむ部分にかかる時間の和となる．単一時間ですむ部分には，$s > t$ の比較演算，l の値を求める演算などがある.
6. YES の場合 $c' < c$. NO の場合 $c' > c$. NO の場合，ステップ 4 の再帰関数呼び出し

直後のステップ 1 で（$s > t$ となり）終了する．よって，$c' = 2c$ としておけば十分である．

7. 帰納法による証明の概略.

【初期段階】 $n = 1$ のとき，漸化式の初期条件式（$T(1) \leq c'$）より成り立つ.

【帰納仮定】 任意の自然数 $n \leq k-1$(ただし, $k \geq 2$)に対して $T(n) \leq c \log n + c'$ であると仮定する.

【帰納段階】 $T(k) \leq c \log k + c'$ が満たされることを示す．帰納仮定と不等式 (4.1) を用いて以下のように示される.

$$\begin{aligned} T(k) &\leq T(\lfloor k/2 \rfloor) + c \quad (\because \text{(4.1)}) \\ &\leq (c \log \lfloor k/2 \rfloor + c') + c \quad (\because \text{帰納仮定}) \\ &\leq c \log(k/2) + c' + c \\ &= c(\log k - 1) + c' + c \\ &= c \log k + c'. \end{aligned}$$

6. 整列

1. $(a_1, \ldots, a_7) = (5, 7, 2, 9, 5, 0, 3)$ に対するアルゴリズムの動作は次のようである．ステップ 2 において，$i = 1$ のとき，ステップ 2-(a) の各繰り返し後の実行結果（整数列）は以下のようである.

$$\begin{aligned} j = 1: &\quad (5, 7, 2, 9, 5, 0, 3) \\ 2: &\quad (5, 2, 7, 9, 5, 0, 3) \\ 3: &\quad (5, 2, 7, 9, 5, 0, 3) \\ 4: &\quad (5, 2, 7, 5, 9, 0, 3) \\ 5: &\quad (5, 2, 7, 5, 0, 9, 3) \\ 6: &\quad (5, 2, 7, 5, 0, 3, 9) \end{aligned}$$

つまり，ステップ 2 の $i = 1$ において，整数列が $(5, 7, 2, 9, 5, 0, 3)$ から $(5, 2, 7, 5, 0, 3, 9)$ に変わる．同様にして，ステップ 2 の各 $i \in [n]$ 回目の繰り返し後の実行結果（整数列）は以下のようである.

$$\begin{aligned} i = 1: &\quad (5, 2, 7, 5, 0, 3, 9) \\ 2: &\quad (2, 5, 5, 0, 3, 7, 9) \\ 3: &\quad (2, 5, 0, 3, 5, 7, 9) \\ 4: &\quad (2, 0, 3, 5, 5, 7, 9) \\ 5: &\quad (0, 2, 3, 5, 5, 7, 9) \\ 6: &\quad (0, 2, 3, 5, 5, 7, 9) \end{aligned}$$

よって，$(0, 2, 3, 5, 5, 7, 9)$ が出力される．

以降，問 4.1, 問 5.1 と同じような問題（具体的な入力に対するアルゴリズムの動作）の解答は省略される．

2. ステップ 2-(a) の $j = \ell - 1$ のときを考える...
3. 一つ目は上の主張より，二つ目は要素数が 1 であることから．
4. 任意の $j \in [n-k]$ について $a[j] \leq a[n-k+1]$.
5. 一つ目より，$a[1] \leq a[2]$．二つ目より，$a[2], \ldots, a[n]$ が昇順である．
6. ステップ 2-(a) の一つの繰り返しにかかる計算時間が $O(1)$，ステップ 2-(a) の繰り返し回数が $n - i$ であるから．
7. 略．（問 5.1 の解答の記述にならうこと．）
8. ステップ 3 の (a), (b) それぞれで，そうでないもの（$a[i] \geq p$ かつ $a[j] < p$）を $a[i], a[j]$ として，(c) でそれらを交換するため．
9. 各ステップの計算時間が $O(y - x + 1)$ であるから．
10. 与式は，$(z - (x+1))(y - z) \geq 0$ となるため．
11. 入力が小さい順になっているとき．その場合，計算時間 $T(n)$ が満たす漸化式は $T(n) \leq T(n-1) + cn$.
12. 再帰呼び出しが常に半分ずつになっていくような入力であるとき．その場合，計算時間 $T(n)$ が満たす漸化式は $T(n) \leq 2T(n/2) + cn$.
13. 略．
14. 各ステップの計算時間が $O(n)$ であるから．
15. n が十分大きいとき，$\lfloor n/2 \rfloor \leq n/1.5$ かつ $\lceil n/2 \rceil \leq n/1.5$ であるから．
16. ヒープの定義より．
17. 帰納法を用いて，ヒープの定義より．
18. 略．
19. x_i, x_p, x_q の三つのうち最大のものが（三頂点の中の）最も上に置かれる．
20. T_q のラベルの最大が x_q であるので．
21. `mindown`(q) を実行したとしているので．
22. $p, q < i$ であるので．
23. T_i の深さが 高々 $\lfloor \log n \rfloor$ であるので．（葉においても呼び出されるので木の深さ $+1$ が必要．）
24. 略．

25. T_{k+1} がヒープであるとする．（これが帰納仮定である．）根（頂点 1）の子（頂点 2,3）を根とした T_{k+1} の二つの部分木はそれぞれヒープである．（T_{k+1} がヒープであるので．）これを主張 6.5 に適用すれば，（`mindown`(1) 実行後の）T_k がヒープになることが示される．

7. 走査

1. 略．
2. 帰納仮定より $u \in X$ であり，$U \subseteq N_u \subseteq X$ であるため．
3. エンキューされる頂点は既訪問になるため．（エンキューされるのは未訪問である頂点だけ．）
4. $|V|$ 回．
5. 略．
6. 未訪問の頂点 u についてだけ，それを既訪問にして `dfs`(u) が呼び出されるので．（未訪問の頂点は単調に減少していく．）
7. 帰納仮定より $u \in X$ であり，$v \in N_u \subseteq X$ であるため．
8. `dfs`(u) が呼び出される直前で u は既訪問になるので．（未訪問の頂点 u にだけ `dfs`(u) が呼び出される．）
9. $|V|$ 回．

参考図書

本テキストは，アルゴリズムとデータ構造のごく一部（の初歩的なこと）しか扱っていない．アルゴリズムとデータ構造について更に学びたい学生は，以下の教科書や，そこであげられている参考図書を参照するとよい．

1. Cによるアルゴリズムとデータ構造，茨木俊秀著，オーム社，２０１４．
2. Algorithms (4th Edition), Robert Sedgewick, Addison-Wesley, 2011. （邦訳：アルゴリズムＣ：第１巻〜第３巻，近代科学社，１９９６．）
3. Introduction to Algorithms (3rd Edition), Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein, The MIT Press, 2009. （邦訳：アルゴリズムイントロダクション（第３版），近代科学社，２０１３．）

索引