アルゴリズムデザイン

山本真基

２０２２年9月

機械的な手順で解決可能な問題を解く際には，それぞれの問題にあった個別のアルゴリズムが必要である．その一方で，アルゴリズムの「設計手法」に着眼すると，それらに共通する統一的な手法がいくつか存在する．本講義では，その代表である分割統治法・貪欲法・動的計画法を学習する．アルゴリズムとデータ構造の授業で学習した多くのアルゴリズムがこれらのいずれかにあたること，及び，現実社会で遭遇する典型的な問題がこれらで解決されることを，理論的な解析を通じて確認する．

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 $2^{\log x} = x$, $\mathrm{e}^{\ln x} = x$.）
- $X \stackrel{\text{def}}{=} Y$ は，X の定義は Y であることを意味する．

目次

第 1 章

アルゴリズムの設計手法

機械的な手順で解決可能な問題を解く際には，それぞれの問題にあった個別のアルゴリズムが必要である．その一方で，アルゴリズムの「設計手法」に着眼すると，それらに共通する**統一的な手法**がいくつか存在する．本章では，ソーティングアルゴリズムを例に，二つの異なるソーティングアルゴリズムが，分割統治法と呼ばれる統一的な手法とみなせることを説明する．

1.1 ソーティングアルゴリズムを例に

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．）

1.1.1 クイックソート

図 1.1 に，クイックソートのアルゴリズムを示す．

問 1.1. 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．

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 1.1: クイックソート

定理 1.1. クイックソートのアルゴリズム 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．

注 1.1. 計算時間といえば，「最悪時」の計算時間を指す．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．

1.1.2 マージソート

図 1.2 に，マージソートのアルゴリズムを示す．

問 1.2. 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

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 1.2: マージソート

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 1.2. マージソートのアルゴリズム 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．

注 1.2. 計算時間の解析は付録を参照．

1.1.3　統一的な手法

クイックソートのアルゴリズムの主要部分は qsort(a, x, y) であり，マージソートのアルゴリズムの主要部分は msort(a, x, y) である．それぞれの関数は異なるが，これらはともに，図 1.3 に示される形式の関数 func(a, x, y) とみなせる．この関数そのものは

func(a, x, y)

1. ある条件が満たされればリターン．
2. processA.
3. ある整数 $b \in [x, y]$ について，func$(a, x, b-1)$, func(a, b, y).
4. processB.

図 1.3: 統一的な手法

（qsort, msort と同様）再帰関数となっている．ステップ 3 が再帰呼び出しとなっており，整列区間 $[x, y]$ について以下の二つの条件が満たされる．

1. 整列区間 $[x, y]$ が二つの部分に分割される．
2. 各整列区間 $[x, b-1]$, $[b, y]$ が（再帰呼び出し後）それぞれで整列される．

アルゴリズムのステップ 1 では，再帰呼び出しの打ち止め条件が記述される．ステップ 2 とステップ 4 が再帰呼び出し前後で行う処理である．qsort(a, x, y) は，func(a, x, y) の processB がない形式となる．

以上のように，全体の問題を二つ（以上）の部分問題に分割して，それぞれを解くことで得られた解を統合する手法が**分割統治法**である．次章では，整列問題以外に適用可能な問題とアルゴリズムを説明する[*1]．

1.2　多項式時間アルゴリズム

アルゴリズムが対象としている問題は，多項式時間[*2]で解くことができる問題と，（多項式時間では解けそうもない）「NP 困難」と呼ばれる問題に（多くが）二分される．本書で

[*1] ユークリッドの互除法や二分探索も分割統治法の一種（特殊な場合）とみなせる．これらは，分割された部分問題のうち一つだけ解けばよいことが保証される．そのような設計手法は，特に，*decrease and conquer* と呼ばれる．

[*2] 多項式時間，及び，オーダー表記の定義は，アルゴリズムとデータ構造のテキストを参照.

は，多項式時間で解くことができる問題**にも**適用可能な以下の設計手法を取り上げる[*3].

1. 分割統治法（divide and conquer）
2. 貪欲法（greedy）
3. 動的計画法（dynamic programming）

これら以外にも，以下のような設計手法もあるが，それらは主に，NP 困難な問題に対する「ヒューリスティックな」（発見的な）設計手法として適用されることが多い[*4].

- バックトラッキング（backtracking）
- 分枝限定法（branch and bound）
- 局所探索法（local search）

[*3] 貪欲法は NP 困難な問題に対する近似アルゴリズムとして，動的計画法は NP 困難な問題に対する擬多項式時間アルゴリズムとして適用されることが多い.

[*4] 理論的な解析が難しくなる場合が多く，それゆえ，計算機実験による評価が主である.

第 2 章

分割統治法

前章で説明したよう，全体の問題を二つ（以上）の部分問題に分割して，それぞれを解くことで得られた解を統合する手法が**分割統治法**である．

2.1 整数積

整数積（integer multiplication）

- 入力：整数 $x, y \in \mathbb{Z}$
- 解：xy

この問題は，二つの整数の積を出力する「関数問題」である．以下では，（一般性を失うことなく）入力 x, y は自然数であるとして，二進表記（ビット表現）で表されるものとする．まず，図 2.1 に，初等的な計算手法，筆算の一つである**長乗法**[*1]を示す．

注 2.1. アルゴリズムのステップ 2 の $x \cdot 2^{i-1}$ は，（ビット表現された）x を $i-1$ ビットずらす（桁上げする）シフト演算である．

問 2.1. $n = 5, m = 3$ として，（二進表記された）自然数 x, y を具体的にあげ，それに対して長乗法を適用させなさい．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．

[*1] 小学生の時に習った掛け算の筆算．

入力：自然数 $x, y \in \mathbb{N}$

1. $x = (x_n, x_{n-1}, \ldots, x_2, x_1) \in \{0,1\}^n$, $y = (y_m, y_{m-1}, \ldots, y_2, y_1) \in \{0,1\}^m$ とする．
2. $a_0 = 0$ として，それぞれの $i \in [m]$ について以下を繰り返す．
   - $y_i = 1$ なら $a_i = a_{i-1} + x \cdot 2^{i-1}$ とする．
   - そうでないなら $a_i = a_{i-1}$ とする．
3. a_m を出力する．

図 2.1: 長乗法

定理 2.1. 長乗法 A は整数積問題を解く．A の計算時間は $O(nm)$ である．

注 2.2. 単一コスト RAM モデルでは，二つの値の四則演算を単一コスト（1 ステップ）で行える計算モデルであった．ここでは，四則演算そのものが問題であるため，（値全体の演算でなく）「ビット演算」が単一コストで行えるものとする．（十進数で表記された場合は，一桁ごとの四則演算が単一コストで行えるものになる．）

証明. まず，アルゴリズムの正当性を示す．これは，アルゴリズムのステップ 2 が長乗法であることから示される．

次に，計算時間を見積もる．ステップ 2 にかかる計算時間が $O(nm)$ であることを示せばよい．ステップ 2 の繰り返し回数が m であることから，任意の $i \in [m]$ について，$a_i = a_{i-1} + x \cdot 2^{i-1}$ にかかる計算時間が $O(n)$ であることを示せばよい．

問 2.2. この事実（$a_i = a_{i-1} + x \cdot 2^{i-1}$ にかかる計算時間が $O(n)$ であること）を示しなさい．

よって，アルゴリズムの計算時間が $O(nm)$ であることがいえる． ■

入力される自然数の長さ（ビット長）が等しければ，つまり，$|x| = |y| = n$ であれば，長乗法の計算時間は $O(n^2)$ となる．これを改良したのが，図 2.2 に示す**カラツバのアルゴリズム**である[*2]．

[*2] コルモゴロフ（確率論の創始者）は，長乗法が最適な（最速の）計算手法であると予想していた！

入力：自然数 $x, y \in \mathbb{N}$（$|x| = |y| = n$）

- $\texttt{karatsuba}(x, y)$ を出力する．

$\texttt{karatsuba}(x, y)$

1. $|x| = |y| = 1$ なら xy を返す．
2. $x = (x_m, \ldots, x_2, x_1) \in \{0,1\}^m$, $y = (y_m, \ldots, y_2, y_1) \in \{0,1\}^m$ とする．
3. $k = \lfloor m/2 \rfloor$ として，

$$\begin{array}{rclrcl} x^{\mathrm{upper}} & = & (x_m, \ldots, x_{k+1}), & x^{\mathrm{lower}} & = & (x_k, \ldots, x_1) \\ y^{\mathrm{upper}} & = & (y_m, \ldots, y_{k+1}), & y^{\mathrm{lower}} & = & (y_k, \ldots, y_1) \end{array}$$

4. 以下を実行する．（$x^{\mathrm{upper}} \geq x^{\mathrm{lower}}$, $y^{\mathrm{upper}} \geq y^{\mathrm{lower}}$ とする．）
   - $z^{\mathrm{upper}} = \texttt{karatsuba}(x^{\mathrm{upper}}, y^{\mathrm{upper}})$.
   - $z^{\mathrm{lower}} = \texttt{karatsuba}(x^{\mathrm{lower}}, y^{\mathrm{lower}})$.
   - $z^{\mathrm{middle}} = z^{\mathrm{upper}} + z^{\mathrm{lower}} - \texttt{karatsuba}(x^{\mathrm{upper}} - x^{\mathrm{lower}}, y^{\mathrm{upper}} - y^{\mathrm{lower}})$.
5. $z = z^{\mathrm{upper}} \cdot 2^{2k} + z^{\mathrm{middle}} \cdot 2^k + z^{\mathrm{lower}}$ を返す．

図 2.2: カラツバのアルゴリズム

注 2.3. ステップ 4 にて，z^{middle} を以下のようにすると，長乗法と同じ計算時間になる．

$$z^{\mathrm{middle}} = \texttt{karatsuba}(x^{\mathrm{upper}}, y^{\mathrm{lower}}) + \texttt{karatsuba}(y^{\mathrm{upper}}, x^{\mathrm{lower}}).$$

再帰呼び出し（掛け算）を 1 回減らすことで，アルゴリズムの高速化が図られる．

問 2.3. $n = 5$ として，（二進表記された）自然数 x, y を具体的にあげ，それに対してカラツバ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 2.2. カラツバのアルゴリズム A は整数積問題を解く．A の計算時間は $O(n^{\log 3})$（$\log_2 3 \approx 1.585$）である．

証明. まず，アルゴリズムの正当性を示す．これは，アルゴリズムが示すよう，以下の恒

等式より明らかである．

$$z \;=\; z^{\mathrm{upper}} \cdot 2^{2k} + z^{\mathrm{middle}} \cdot 2^{k} + z^{\mathrm{lower}}.$$

ただし，($x^{\mathrm{upper}} \geq x^{\mathrm{lower}}$, $y^{\mathrm{upper}} \geq y^{\mathrm{lower}}$ であるとき)

$$\begin{array}{lrcl} 1. & z^{\mathrm{upper}} & = & x^{\mathrm{upper}} y^{\mathrm{upper}} \\ 2. & z^{\mathrm{lower}} & = & x^{\mathrm{lower}} y^{\mathrm{lower}} \\ 3. & z^{\mathrm{middle}} & = & z^{\mathrm{upper}} + z^{\mathrm{lower}} - \left(x^{\mathrm{upper}} - x^{\mathrm{lower}}\right)\left(y^{\mathrm{upper}} - y^{\mathrm{lower}}\right) \end{array}$$

問 2.4. $x^{\mathrm{upper}} \leq x^{\mathrm{lower}}$, $y^{\mathrm{upper}} \geq y^{\mathrm{lower}}$ であるとどうなるか．

次に，計算時間を見積もる．$|x| = |y| = n$ である入力 x, y に対して，`karatsuba`(x, y) にかかる計算時間を $T(n)$ とする．このとき，ある定数 c, c' が存在して，以下の漸化式が成り立つ．

$$\begin{array}{lcl} T(n) & \leq & 3T(\lceil n/2 \rceil) + cn \\ T(1) & \leq & c' \end{array} \tag{2.1}$$

問 2.5. この漸化式が成り立つ理由を説明しなさい．

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 ℓ に対して $n = 2^{\ell} \geq 1$ であるとする．($\ell = \log n.$) このとき，

$$T(n) \leq 3T(n/2) + cn \tag{2.2}$$
$$T(n/2) \leq 3T(n/2^2) + c(n/2) \tag{2.3}$$
$$T(n/2^2) \leq 3T(n/2^3) + c(n/2^2) \tag{2.4}$$
$$\vdots$$

上の式において，不等式 (2.3) を不等式 (2.2) へ代入すると，

$$\begin{aligned} T(n) &\leq 3T(n/2) + cn \\ &\leq 3(3T(n/2^2) + c(n/2)) + cn \\ &= 3^2 T(n/2^2) + (1 + 3/2) \cdot cn. \end{aligned}$$

更に，この不等式に不等式 (2.4) を代入すると，

$$\begin{aligned} T(n) &\leq 3^2 T(n/2^2) + (1 + 3/2) \cdot cn \\ &\leq 3^2(3T(n/2^3) + c(n/2^2)) + (1 + 3/2) \cdot cn \\ &= 3^3 T(n/2^3) + (1 + (3/2)^1 + (3/2)^2) \cdot cn. \end{aligned}$$

このように，順次，不等式を代入していけば，

$$\begin{aligned} T(n) &\le 3^1 T(n/2) + cn \\ &\le 3^2 T(n/2^2) + ((3/2)^0 + (3/2)^1) \cdot cn \\ &\le 3^3 T(n/2^3) + ((3/2)^0 + (3/2)^1 + (3/2)^2) \cdot cn \\ &\vdots \\ &\le 3^i T(n/2^i) + ((3/2)^0 + (3/2)^1 + \cdots + (3/2)^{i-1}) \cdot cn. \end{aligned}$$

$n = 2^\ell$ より，上の不等式で $i = \ell$ とおけば，

$$T(n) \ \le \ 3^\ell T(1) + ((3/2)^0 + (3/2)^1 + \cdots + (3/2)^{\ell-1}) \cdot cn.$$

更に，$\ell = \log n$（$\because n = 2^\ell$），$T(1) \le c'$（$\because$ 不等式 (2.1)）より（$c' \le c$ としておけば），

$$\begin{aligned} T(n) &\le 3^\ell T(1) + ((3/2)^0 + (3/2)^1 + \cdots + (3/2)^{\ell-1}) \cdot cn \\ &\le c' \cdot 3^\ell + 2((3/2)^\ell - 1) \cdot cn \quad (\because\ T(1) \le c') \\ &\le c' \cdot 3^\ell + 2c \cdot 3^\ell \quad (\because\ n = 2^\ell) \\ &\le 3c \cdot 3^{\log n} \quad (\because\ c' \le c) \\ &= 3cn^{\log 3}. \end{aligned}$$

一般の自然数 n についても上の式が成り立つことが示される[*3]．よって，任意の自然数 n について，$T(n) \le 3cn^{\log 3} = O(n^{\log 3})$． ■

注 2.4. カラツバのアルゴリズムより高速なアルゴリズムに，高速フーリエ変換（FFT: Fast Fourier Transform）を利用した，ショーンハーゲ・シュトラッセン（Schönhage-Strassen）のアルゴリズムがある．計算時間は $O(n \log n \cdot \log(\log n))$ である[*4]．

問 2.6. 注 2.3 で示した z^{middle} にすると，漸化式 2.1 はどのようになるか．

[*3] 分割統治法の計算時間を求めるのに用いられる “The Master Theorem” によって．

[*4] ２０１９年，$O(n \log n)$ 時間のアルゴリズムが発見された．

2.2　行列積

行列積（integer multiplication）

- 入力：行列 $A, B \in \mathbb{Z}^{n \times n}$
- 解：AB

この問題は，二つの行列の積を出力する「関数問題」である．ここでは，（擬似コードの単純化，更に，計算時間の解析の簡略化のため）正方行列を扱う．（一般の行列についても同様のことがいえる．）n 次元の正方行列 A を $A = (a_{ij})_{1 \leq i,j \leq n}$ と表記する．これは，A の i 行 j 列の要素が a_{ij} であることを示す．まず，図 2.3 に，行列積の定義に基づいた計算手法を示す．

入力： 行列 $A, B \in \mathbb{Z}^{n \times n}$

1. $A = (a_{ij})_{1 \leq i,j \leq n}$, $B = (b_{ij})_{1 \leq i,j \leq n}$ とする．
2. 任意の $i, j \in [n]$ について，$c_{ij} = \sum_{k \in [n]} a_{ik} b_{kj}$ とする．
3. $C = (c_{ij})_{1 \leq i,j \leq n}$ を出力する．

図 2.3: 定義に基づいた計算手法

問 2.7. $n = 3$ として，行列 A, B を具体的にあげ，それに対して定義に基づいた計算手法を適用させなさい．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 2.3. 図 2.3 のアルゴリズム A は行列積問題を解く．A の計算時間は $O(n^3)$ である．

注 2.5. ここでは，単一コスト RAM モデルを適用する．つまり，行列の要素ごとの四則演算は単一コスト（1 ステップ）で行えるものとする．

証明. まず，アルゴリズムの正当性を示す．これは，アルゴリズムのステップ 2 が行列積の定義であることから示される．

次に，計算時間を見積もる．ステップ 2 にかかる計算時間が $O(n^3)$ であることを示せばよい．任意の $i, j \in [n]$ について，$c_{ij} = \sum_{k\in[n]} a_{ik}b_{kj}$ にかかる計算時間が $O(n)$ であることを示せばよい．

問 2.8. この事実（$c_{ij} = \sum_{k\in[n]} a_{ik}b_{kj}$ にかかる計算時間が $O(n)$ であること）を示しなさい．

よって，アルゴリズムの計算時間が $O(n^3)$ であることがいえる． ■

これを改良したのが，図 2.4 に示す**シュトラッセンのアルゴリズム**である．ここでは，（擬似コードの単純化，更に，計算時間の解析の簡略化のため）ある非負整数 ℓ に対して $n = 2^\ell$ とする．（そうでない場合についても同様のことがいえる．）

問 2.9. $n = 2^\ell$ でない場合，図 2.4 に示した擬似コードについて，どのような不都合が生じるか．

注 2.6. アルゴリズムのステップ 3 で分割された行列は，すべて正方行列になる．

問 2.10. $n = 4$ として，行列 A, B を具体的にあげ，それに対してシュトラッセン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 2.4. シュトラッセンのアルゴリズム A は行列積問題を解く．A の計算時間は $O(n^{\log 7})$（$\log_2 7 \approx 2.81$）である．

証明. まず，アルゴリズムの正当性を示す．行列積の定義より，

$$\begin{aligned} AB &= \left(\begin{array}{c:c} A_{11} & A_{12} \\ \hdashline A_{21} & A_{22} \end{array}\right)\left(\begin{array}{c:c} B_{11} & B_{12} \\ \hdashline B_{21} & B_{22} \end{array}\right) \\ &= \left(\begin{array}{c:c} A_{11}B_{11} + A_{12}B_{21} & A_{11}B_{12} + A_{12}B_{22} \\ \hdashline A_{21}B_{11} + A_{22}B_{21} & A_{21}B_{12} + A_{22}B_{22} \end{array}\right) \end{aligned}$$

入力：行列 $A, B \in \mathbb{Z}^{n \times n}$　// A, B は正方行列

- `strassen`(A, B) を出力する．

`strassen`(A, B)　// $A = (a_{ij})_{1 \le i,j \le m}$, $B = (b_{ij})_{1 \le i,j \le m}$

1. A, B の次元が 1 なら 1×1 行列 $(a_{11}b_{11})$ を返す．
2. $C = (c_{ij})_{1 \le i,j \le m}$ とする．（C が戻り値となる．）
3. $k = m/2$ として，A を以下のように四等分割する．

$$\begin{array}{rclcrcl} A_{11} & = & (a_{ij})_{1 \le i,j \le k}, & & A_{12} & = & (a_{ij})_{1 \le i \le k, k+1 \le j \le m} \\ A_{21} & = & (a_{ij})_{k+1 \le i \le n, 1 \le j \le k}, & & A_{22} & = & (a_{ij})_{k+1 \le i,j \le m} \end{array}$$

   B, C についても同様．
4. 以下を実行する．
   - $M_1 =$ `strassen`$(A_{11} + A_{22}, B_{11} + B_{22})$.
   - $M_2 =$ `strassen`$(A_{11} + A_{12}, B_{22})$.
   - $M_3 =$ `strassen`$(A_{21} + A_{22}, B_{11})$.
   - $M_4 =$ `strassen`$(A_{11}, B_{12} - B_{22})$.
   - $M_5 =$ `strassen`$(A_{22}, B_{21} - B_{11})$.
   - $M_6 =$ `strassen`$(A_{12} - A_{22}, B_{21} + B_{22})$.
   - $M_7 =$ `strassen`$(A_{21} - A_{11}, B_{11} + B_{12})$.
5. C を返す．
   - $C_{11} = M_1 - M_2 + M_5 + M_6$
   - $C_{12} = M_2 + M_4$
   - $C_{21} = M_3 + M_5$
   - $C_{22} = M_1 - M_3 + M_4 + M_7$

図 2.4: シュトラッセンのアルゴリズム

よって，行列 C の定義は，

$$\begin{array}{rcl} C_{11} & = & A_{11}B_{11} + A_{12}B_{21} \\ C_{12} & = & A_{11}B_{12} + A_{12}B_{22} \\ C_{21} & = & A_{21}B_{11} + A_{22}B_{21} \\ C_{22} & = & A_{21}B_{12} + A_{22}B_{22} \end{array}$$

一方，アルゴリズムのステップ 4 で定義された $M_1, \ldots, M_7$ を用いて，ステップ 5 で示

されたようにも，$C_{11}, C_{12}, C_{21}, C_{22}$ が表される．

問 2.11. この事実（$M_1, \ldots, M_7$ を用いて $C_{11}, C_{12}, C_{21}, C_{22}$ が表されること）を示しなさい．

次に，計算時間を見積もる．n 次元正方行列 A, B に対して，`strassen`(A, B) にかかる計算時間を $T(n)$ とする．（仮定より $n = 2^\ell$ とする．）このとき，ある定数 c, c' が存在して，以下の漸化式が成り立つ．

$$\begin{array}{rcl} T(n) & \leq & 7T(n/2) + cn^2 \\ T(1) & \leq & c' \end{array} \tag{2.5}$$

問 2.12. この漸化式が成り立つ理由を説明しなさい．

この漸化式は，漸化式 (2.1)（カラツバのアルゴリズムの計算時間（$n = 2^\ell$））と同様に解くことができる．

問 2.13. 漸化式 (2.1) の解法にならって，漸化式 (2.5) の解法を示しなさい．

よって，（$n = 2^\ell$ の）任意の自然数 n について，$T(n) = O(n^{\log 7})$. ■

注 2.7. シュトラッセンのアルゴリズムより高速なアルゴリズムに，コッパースミス・ウィノグラッド（Coppersmith-Winograd）のアルゴリズムがある．計算時間は $O(n^{2.38})$ である．

2.3 幾何的問題

最近点対問題（closest pair points）

- 入力：点集合 $p_1, p_2, \ldots, p_n \in \mathbb{R}^2$
- 解：最近点対 (p_a, p_b)

この問題は，二次元平面上の n 個の点が与えられ（$p_i = (x_i, y_i)$ とする），最も距離[*5]が小さい点対を出力する「関数問題」である．ここでは，点対 (p_i, p_j) の距離を $d(p_i, p_j)$ と表記する．以下では，（一般性を失うことなく）点の座標は整数であるとする．

[*5] $p_i = (x_i, y_i)$, $p_j = (x_j, y_j)$ の距離は $\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$ である．

問 2.14. 一次元直線上であれば（与えられた点が直線上にあれば）どのようなアルゴリズムが考えられるか.

まず，図 2.5 に，最近点対問題を解く単純なアルゴリズムを示す.

入力：点集合 $p_1, p_2, \ldots, p_n \in \mathbb{Z}^2$ // $P = \{p_1, \ldots, p_n\}$ とする.

1. P のすべての点対 (p_i, p_j) から距離が最小なものを出力する.

図 2.5: 単純なアルゴリズム

注 2.8. 点対の距離は定数時間で計算できる.（単一コスト RAM モデルのため.）

命題 2.5. 図 2.5 のアルゴリズム A は最近点対問題を解く．A の計算時間は $O(n^2)$ である.

次に，図 2.6 に，最近点対問題を解く分割統治アルゴリズムを示す.

問 2.15. $n = 8$ として，点集合 P を具体的にあげ，それに対して図 2.6 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 2.6. 図 2.6 のアルゴリズム A は最近点対問題を解く．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．このためには，`closest`(P_x, P_y) が P の最近点対を出力することを示せばよい．そのためには，`closest`(Q_x, Q_y) が点集合 Q の最近点対を出力することを示せばよい[*6]．これを，$|Q|$ についての帰納法により示す．$|Q| \leq 3$ のときは明らかである．$|Q| \leq m - 1$（$m \geq 4$）のとき，`closest`(Q_x, Q_y) が最近点対を出力するとする.（これが帰納仮定である.）

[*6] Q は集合，Q_x, Q_y は（x, y 座標について Q を昇順に整列した）「点列」である.

入力：点集合 $p_1, p_2, \ldots, p_n \in \mathbb{Z}^2$ // $P = \{p_1, \ldots, p_n\}$ とする．

1. P を x 座標，y 座標について昇順に整列させる．（それぞれ，P_x, P_y とする．）
2. `closest`(P_x, P_y) を出力する．

`closest`(Q_x, Q_y) // $Q = \{q_1, \ldots, q_m\}$，$Q_x = (q_1, \ldots, q_m)$ とする．

1. $m \leq 3$ なら Q の最近点対 (q_i, q_j) を返す．
2. $k = \lfloor m/2 \rfloor$ として，$L = \{q_1, \ldots, q_k\}$, $R = \{q_{k+1}, \ldots, q_m\}$ とする．
3. L, R を y 座標について昇順に整列させる．（それぞれ，L_y, R_y とする．また，$L_x = (q_1, \ldots, q_k)$, $R_x = (q_{k+1}, \ldots, q_m)$ とする．）
4. $(l_1, l_2) =$ `closest`(L_x, L_y), $(r_1, r_2) =$ `closest`(R_x, R_y) とする．
5. $d = \min\{d(l_1, l_2), d(r_1, r_2)\}$ として，$S \subseteq Q$ を以下のように定義する．

$$S \stackrel{\text{def}}{=} \{q = (x, y) \in Q : |x_k - x| \leq d\}.$$

   ただし，$q_k = (x_k, y_k)$ とする．
6. S を y 座標について整列させる．（$S_y = (s_1, \ldots, s_t)$ とする．）
7. 任意の $i \in [t]$ について（S_y について昇順に），
   - s_i と $s_{i+1}, \ldots, s_{i+11}$ との距離を求め，それら（s_i との１１点対）の最近点対とそれまでの最近点対 (a, b) との比較をして，最近点対 (a, b) を更新する．ただし，$a \in L$, $b \in R$ とする．
8. $\{(a, b), (l_1, l_2), (r_1, r_2)\}$ の最近点対を返す．

図 2.6: 分割統治アルゴリズム

$|Q| = m$ のときを考える．$Q_x = (q_1, \ldots, q_m)$ が x 座標について昇順に整列されているものとする．（アルゴリズムもそうなっている．）アルゴリズムのステップ 4 の実行後，帰納仮定より，(l_1, l_2), (r_1, r_2) が，それぞれ L, R の最近点対となる．（その距離を，順に，$d(l_1, l_2)$, $d(r_1, r_2)$，更に，$d = \min\{d(l_1, l_2), d(r_1, r_2)\}$ とする．）

事実 2.1. Q の最近点対は，L, R の最近点対か， L, R の間の最近点対かである．

よって，L, R の間の最近点対を求めればよい．このためには，L, R の間の最近点対の距離が d 未満であった場合，ステップ 7 の実行後，(a, b) が L, R の間の最近点列となることを示せばよい．（ステップ 8 で三つの点対の中から最近点対を返す．）

$d = \min\{d(l_1, l_2), d(r_1, r_2)\}$ とする．このとき，点の座標の整数性より $d \geq 1$ である．$q_k = (x_k, y_k)$ とする．x-y 二次元座標平面において，$x = x_k$ の直線を X，$x = x_k - d$, $x = x_k + d$ の直線を，それぞれ X_l, X_r とする．このとき，点集合 S は，Q の点で X_l

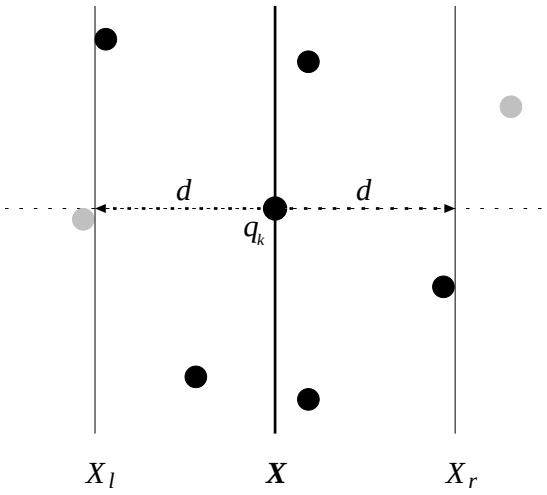


図 2.7: 点集合 S

と X_r の間にあるものである．（よって，$q_k \in S$.）

主張 2.1. L, R の間の最近点対の距離が d 以下であれば，S の点対である．

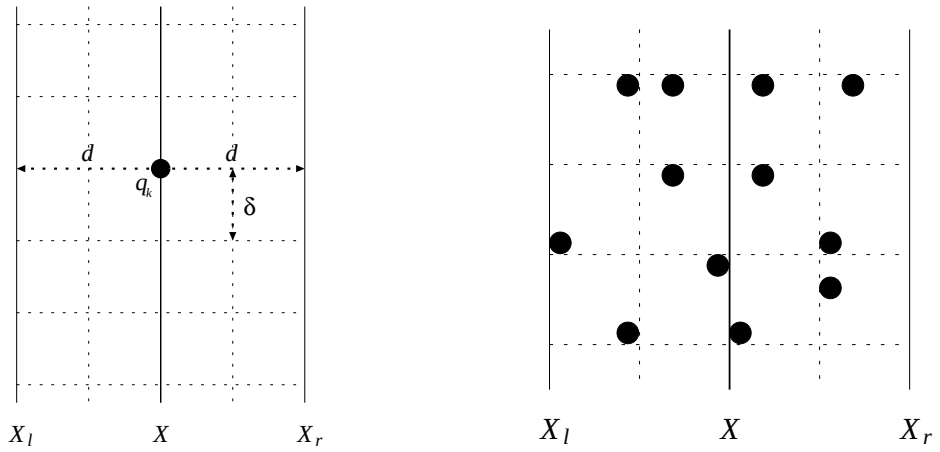
問 2.16. この主張が成り立つ理由を説明しなさい．

この主張より，L, R の間の d 未満の最近点対を求めるためには，L, R の間の全点対の距離を求める必要はなく，S の点対（の距離）を調べればよいことが分かる．（アルゴリズムもそうなっている．）

主張 2.2. $S_y = (s_1, \ldots, s_t)$ とする．（y 座標について S の昇順の点列．）L, R の間の最近点対を (s_i, s_j)（$i < j$）とする．$d(s_i, s_j) < d$ とする．一般性を失うことなく $s_i \in L$, $s_j \in R$ とする．（(s_i, s_j) が L, R の間の点対であるため．）このとき，$j \leq i + 11$ である．

証明. $\delta = d/2$ とする．以下の図 2.8-(a) で示されたよう，点 q_k を格子点とした，直線 X_l と X_r の間の δ-グリッドを考える．このとき，どのグリッドについても S の点は（そのグリッド内に）高々一つまでしか存在できない．

問 2.17. この事実が成り立つ理由を説明しなさい．



(a) δ-グリッド　　(b) 各グリッド内の点

図 2.8: δ-グリッドと各グリッド内の点

よって，$i < j$ より[*7]，図 2.8-(b) で示されたよう，$d(s_i, s_j) < d$ を満たす s_j は，s_i があるグリッドを除いた高々 11 個のグリッドにある．よって，$j \leq i + 11$ となる． ■

この主張より，L, R の間の最近点対の距離が d 未満であった場合，ステップ 7 の実行後，(a, b) が L, R の間の最近点対となることが示される．よって，`closest`(Q_x, Q_y) は Q の最近点対を出力する．

次に，計算時間を見積もる．まず，アルゴリズム（本体）のステップ 1 にかかる計算時間は $O(n \log n)$ である．（マージソートを用いるなどして．）よって，ステップ 2 にかかる計算時間が $O(n \log n)$ であることを示せばよい．そのためには，$|Q| = m$ である点集合 Q に対して，`closest`(Q_x, Q_y) にかかる計算時間が $O(m \log m)$ であることを示せばよい．`closest`(Q_x, Q_y) にかかる計算時間を $T(m)$ とする．

問 2.18. `closest`(Q_x, Q_y) の各ステップにかかる計算時間を示しなさい．特に，ステップ 3 及び 6 にかかる計算時間が $O(m)$ である理由を説明しなさい．

この問より，ある定数 c, c'（$c' \leq c$）が存在して，以下の漸化式が成り立つ．

$$\begin{aligned} T(m) &\leq 2T\left(\lceil \tfrac{m}{2} \rceil\right) + cm \\ T(1) &\leq c' \end{aligned}$$

[*7] y 座標では，s_i が s_j 以下である．

問 2.19. この漸化式を（帰納法により）解くことにより，`closest`(Q_x, Q_y) にかかる計算時間が $O(m \log m)$ となることを示しなさい．

よって，A の計算時間は $O(n \log n)$ である．■

第 3 章

貪欲法

何かしらの指標に基づいた優先度によってアルゴリズムのふるまいを決める手法が**貪欲法**である．ここでは，そうすることによって「最適解」が求められる（最適化）問題とそれを（多項式時間で）解くアルゴリズムを取り上げる[*1]．

3.1 最短経路探索問題（その１）

ここでは，特に断らない限り，グラフといった場合，連結無向グラフを指すものとする．また，頂点が重複しない単純経路を扱い，単に，経路という[*2]．

定義 3.1

$G = (V, E)$ をグラフとする．G の頂点の集合を $V(G)$，G の辺の集合を $E(G)$，と表記する．（つまり，$V(G) = V$, $E(G) = E$ である．）

任意の辺 $e \in E$ について，グラフ $G' = (V, E \setminus \{e\})$ を $G \setminus \{e\}$ と表記する．また，任意の $f = (u, v) \in V \times V$ について，グラフ $G' = (V, E \cup \{f\})$ を $G \cup \{f\}$ と表記する．

定義 3.2

$G = (V, E)$ を連結グラフ，$\ell : E \to \mathbb{R}_0^+$ を辺の長さとする．$s, t \in V$ を異なる頂点とする．P を s から t への G 上の任意の経路とする．P の**長さ**を $|P|$ と表し

[*1] 一般に，貪欲法によって最適解を（多項式時間で）求められる問題は少なく，最適解に近い解，「近似解」，を求めるアルゴリズムによく用いられる．（ナップサック問題を解く近似アルゴリズムなどに．）また，理論的な解析が難しいヒューリスティクス（発見的手法）にもよく用いられる．

[*2] グラフや経路については，アルゴリズムとデータ構造のグラフの章を参照．

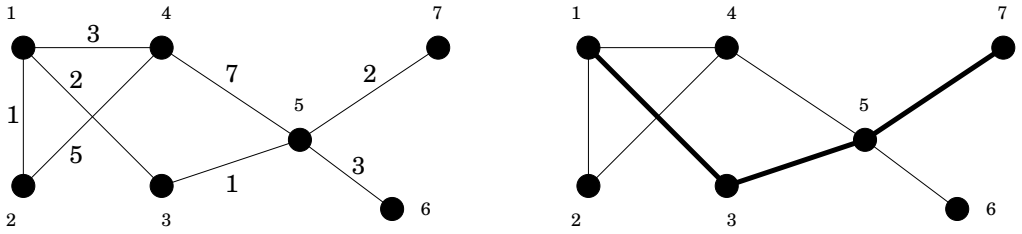
て，以下のように定義する．

$$|P| \stackrel{\text{def}}{=} \sum_{e \in E(P)} \ell(e).$$

s から t への G 上の経路の長さが最小である経路を，s から t への**最短経路**という．また，その長さを s から t への**最短距離**，または，単に，**距離**といい，$\delta_s(t)$ と表記する．（s が自明な場合は，単に，$\delta(t)$ と表す．）

注 3.1. $s = t$ のとき，s から t への距離を 0 とする．（$\delta_s(s) = 0$.）

例 3.1 (最短経路)**.** 以下の右図は，左図で示された長さ付きグラフ（辺の長さは辺上に示された値）上の，頂点 1 から頂点 7 への最短経路である．頂点 1 から頂点 7 への距離は 5 となる．（$\delta_1(7) = 5$.）



(a) 長さ付きグラフ　　(b) 最短経路

図 3.1: 最短経路

最短経路探索問題（shortest path search）

- 入力：グラフ $G = (V, E)$，長さ $\ell : E \to \mathbb{R}_0^+$，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[*3]．ここでは，議論の単純化のため，入力を無向グラフに限定する．（有向グラフに対しても同様のことが成り立つ．）更に，最短経路は単純としてよい[*4]．

[*3] 辺の長さは非負とする．長さが（負の数を含む）実数であるものを，次章では「最小コスト経路探索問題」と呼ぶ．（詳細は次章を参照．）このように問題を区別する理由は問 3.5 を参照．

[*4] グラフの辺の長さが非負であるため．

命題 3.1. $G=(V,E)$ をグラフ，$\ell : E \to \mathbb{R}_0^+$ を辺の長さとする．$s,t \in V$ を異なる頂点とする．$P=(a_1,\ldots,a_k)$ を s から t への G 上の最短経路とする．（$a_1=s$, $a_k=t$.）このとき，任意の $i \in [k]$ について，

$$\delta_s(a_i) \;=\; \sum_{j\in[i-1]} \ell(a_j,a_{j+1}).$$

問 3.1. この事実が成り立つことを証明しなさい.

図 3.2 に，最短経路探索問題を解く**ダイクストラのアルゴリズム**を示す.

入力：グラフ $G=(V,E)$，長さ $\ell : E \to \mathbb{R}_0^+$，始点 $s \in V$，終点 $t \in V$

1. 関数 $d : V \to \mathbb{R}_0^+$ について，$d(s)=0$，任意の頂点 $v \in V \setminus \{s\}$ について $d(v)=\infty$，関数 $p : V \to V$ について，任意の頂点 $v \in V$ について $p(v)=v$，とする.
2. Q を V の全頂点からなる優先度付きキューとする．（Q の優先度は関数 d の値の小さい順とする.）
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）$d(u)=\infty$ なら繰り返しを終了する.
   （c）各 $v \in N_u$ について，以下が満たされれば，$d(v)=\alpha$, $p(v)=u$ とする.

$$d(v) \;>\; \alpha \;\stackrel{\text{def}}{=}\; d(u)+\ell(u,v).$$

4. $d(t)<\infty$ であれば，関数 p によって辿れる t から s への経路の逆を，そうでないなら到達不可能を出力する.

図 3.2: ダイクストラのアルゴリズム

注 3.2. 優先度付きキューとは，各要素が優先度をもつキューである．（ここでは，関数 d の値がそれとなる.）デキューすると，優先度の最も高い要素（ここでは d の値の最小の頂点）が取り出される．（よって，単純なキューの FIFO とは異なる.）エンキューされた後，デキューされるまでに優先度が更新されることもある．（ステップ 3-(c) にて関数 d

が更新される.）優先度付きキューの「更新手順」は，それを実現するデータ構造に依存する.（詳細は，ダイクストラ法の実装の節を参照.）

事実 3.1. 任意の頂点についてデキューされるのは高々 1 回である.（ステップ 2 で全頂点がエンキューされ，ステップ 3 でエンキューされることはないため.）

注 3.3. ステップ 3 の実行途中では，p が「暫定的」な最短経路を，d は p が示す経路の長さ（暫定距離）を保持する.（事実 3.3 を参照．ステップ 3 の終了後には，それぞれが最短経路と最短距離を示すことが，定理 3.2 で証明される.）

事実 3.2. ステップ 3 では，p は s からの経路を，d は p が示す経路の長さを保持する.（アルゴリズムのステップ 3-(c) より.）

問 3.2. １０頂点上の適当なグラフ（と長さ，始点・終点）を具体的にあげ，それに対してダイクストラ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.

問 3.3. ダイクストラのアルゴリズムのどのあたりが貪欲的か説明しなさい.

注 3.4. ここでは，ダイクストラのアルゴリズムの設計手法を貪欲法とみなした．次章の動的計画法という設計手法と見なすこともできる．詳細は次章を参照.

定理 3.2. ダイクストラのアルゴリズム A は最短経路探索問題を解く．A の計算時間は $O(|V| + |E| \log |V|)$ である.（連結グラフであれば $O(|E| \log |V|)$.）

注 3.5. 定理の計算時間は，優先度付きキューに二分ヒープを用いた場合である．フィボナッチヒープと呼ばれるデータ構造を用いると，A の計算時間は $O(|E| + |V| \log |V|)$ に改良される.（用いるデータ構造によって計算時間が異なる！）

証明. まず，アルゴリズムの正当性を示す．そのためには，更に一般的な次の命題を示す.

主張 3.1. 任意の頂点 $u \in V$ について，s から u への最短距離を $\delta(u)$ とする．このとき，ステップ 3 全体の終了後，任意の頂点 $u \in V$ について，$d(u) = \delta(u)$.

証明. ステップ 3 の任意の繰り返しにおいて，デキューされた頂点の集合を S とする[*5]．主張を示すために，以下が成り立つことを示す．

$$(*) \cdots \text{任意の頂点 } u \in S \text{ について } d(u) = \delta(u)$$

これを $|S|$（ステップ 3 の繰り返し回数）についての帰納法により示す．

まず，$|S| = 1$ のとき，$u = s$ であることから，$d(s) = \delta(s) = 0$ であることは明らかである．ステップ 3 の i 回目の繰り返し終了後の S を S_i と表記する．（よって，$|S_i| = i$.）$|S| = r - 1$（$S = S_{r-1}$）のとき，上の条件 $(*)$ が満たされるものとする．（これが帰納仮定である．）$|S| = r$（$S = S_r$）のときを考える．デキューされた頂点を u とする．（$S_r = S_{r-1} \cup \{u\}$.）一般性を失うことなく $\delta(u) < \infty$ とする．$d(u) = \delta(u)$ を示すために，$d(u) \geq \delta(u)$ かつ $d(u) \leq \delta(u)$ を示す．前者は明らか．

問 3.4. 前者が明らかである理由を説明しなさい．

以降，後者を示す．ステップ 3-(c) より，以下の事実が成り立つ．

事実 3.3. 任意の $S \subseteq V$，任意の $v \in V$ について，

$$d(v) \;=\; \min_{u \in S}\{d(u) + \ell(u, v) : (u, v) \in E\}.$$

s から u への最短経路を $P = (a_1, a_2, \ldots, a_k)$（ただし，$a_1 = s$, $a_k = u$）とする．（よって，$\sum_{i \in [k-1]} \ell(a_i, a_{i+1}) = \delta(u)$.）まず，$\{a_1, \ldots, a_k\} \setminus S \neq \emptyset$ である場合を考える．$\{a_1, \ldots, a_k\} \setminus S$ の（P の順で）最初の頂点を a_j とする．（よって，$\{a_1, \ldots, a_{j-1}\} \subseteq S$ かつ $a_j \notin S$.）このとき，

$$\begin{aligned}
\delta(u) &= \sum_{i \in [k-1]} \ell(a_i, a_{i+1}) \\
&\geq \delta(a_{j-1}) + \ell(a_{j-1}, a_j) \quad (\because \text{ 命題 3.1 と } j < k) \\
&= d(a_{j-1}) + \ell(a_{j-1}, a_j) \quad (\because \ a_{j-1} \in S_{r-1} \text{ より帰納仮定}) \\
&\geq d(a_j) \quad (\because \ a_{j-1} \in S_{r-1} \text{ と事実 3.3}) \\
&\geq d(u) \quad (\because \ a_j \notin S \text{ と } u \text{ の選択}).
\end{aligned}$$

これより，$\delta(u) \geq d(u)$．$\{a_1, \ldots, a_k\} \setminus S = \emptyset$（$\{a_1, \ldots, a_k\} \subseteq S$）である場合も同様に示される．よって，後者が示され，前者と合わせて $d(u) = \delta(u)$ となる． ■

次に，計算時間を見積もる．ステップ 1, 2, 4 にかかる計算時間はいずれも $O(|V|)$ である．以降，ステップ 3 にかかかる計算時間を見積もる．ステップ 3 の主な操作は以下である．

[*5] 事実 3.1 より，$|S|$ とステップ 3 の繰り返し回数は等しい．

- キューの操作（デキューと関数 d, p の更新にともなうキューの更新）.
- N_u の特定（u の隣接頂点の走査）.

それぞれについて，ステップ 3 の全体が終了するまでにかかった「合計の」計算時間を見積もる．まず，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（事実 3.1 より），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 $c \cdot \sum_{u \in V} |N_u|$．これは $c \cdot 2|E| = O(|E|)$ である[*6]．次に，キューの操作について，一つの頂点につきにデキューにかかる時間は $O(1)$ であることから，デキューにかかる計算時間の合計は $O(|V|)$ となる．

主張 3.2 (命題 3.4)**.** 任意の頂点について，関数 d, p の更新にかかる計算時間は $O(1)$，それにともなうキューの更新にかかる計算時間は $O(\log |V|)$ である．

この主張より，（最悪）走査した隣接頂点それぞれにつきキューの更新が必要であるとして，キューの更新にかかる計算時間の合計は高々，

$$c \cdot \sum_{u \in V} |N_u| \cdot \log |V| \ = \ c(2|E|) \log |V| \ = \ O(|E| \log |V|).$$

よって，ステップ 3 にかかる計算時間の合計は，$O(|E|) + O(|V|) + O(|E| \log |V|) = O(|V| + |E| \log |V|)$ となる．

以上より，ダイクストラのアルゴリズムの計算時間は，$O(|V|) + O(|V| + |E| \log |V|) = O(|V| + |E| \log |V|)$ となる．（連結グラフであれば $O(|E| \log |V|)$.） ■

問 3.5. この定理のアルゴリズムの正当性より，関数 ℓ の値域が非負であることは，ダイクストラのアルゴリズムが適用可能であるための十分条件である．逆に，それが（適用可能であるための）必要条件でもある．その理由を説明しなさい.

事実 3.4. ダイクストラのアルゴリズムで得られる p が表すグラフは木（s を根とした有向木）である.

定義 3.3

$G = (V, E)$ をグラフとする．ダイクストラのアルゴリズムの p が示す木を，**最短経路木**という.

以降では，最短経路木は無向グラフとみなす.

[*6] アルゴリズムとデータ構造（または組合せ論）のグラフの章を参照.

事実 3.5. 最短経路木は，s から各頂点への最短経路を示す．

ダイクストラ法の実装：優先度付きキュー

ダイクストラ法を実装する．そのためには，「ヒープ」を用いた「**優先度付きキュー**」の実装が必要となる[*7]．ここでは，各データは，ＩＤ（整数）とコスト（整数）の「ペア」であり，コストを優先度付きキューの「キー」とする．ヒープの構築の詳細は巻末の付録を参照のこと．

注 3.6. 優先度付きキューにおいて，ID が頂点 $v \in V$ に，コストが暫定距離 $d(v)$ に対応している．

実装 3.1. ヒープ構築のプログラム（巻末の付録）を，（効率のよい）キー更新が可能な優先度付きキューに拡張する．以下の要領に従い，図 3.3 で示されたコードをヒープ構築のメイン関数（巻末の付録の図 B.1）に「追加」して，キー更新を備えた優先度付きキューを実装しなさい．

1. まず，任意のＩＤに対して，そのＩＤをもつノードが（$O(1)$ で！）参照できるよう，大きさ３０の `int` 型の配列 `node_idx` を（大域変数として）用意する．この配列は以下を意味する．

   `node_idx[i]==j` $\iff$ ＩＤが `i` のものが `node_data[j]` に格納される．
2. この `node_idx` に適切な値が入るよう，`gen_node_data` 及び `make_heap` を修正する．つまり，`gen_node_data` 後であれば，任意の `i` について `node_idx[i]=i` であるように．また，`make_heap` 後であれば，例えば，`node_data[10].id=0` であれば，`node_idx[0]=10` となるように．
3. 以下の関数を実装する．（追加するコードから呼び出される．）このとき，`node_idx` の値を「適切に」更新する必要がある．
   - `push(id, cost)`：`id`, `cost` をもつ（新たな）ノードをヒープへ追加する．
   - `pop()`：ヒープから最小のコストをもつノードを削除する．
   - `change(id, cost)`：`id` をもつノードのコストを `cost` に変更する．このとき，`node_idx` を利用すれば，その `id` をもつノードが（$O(1)$ で！）特定できる．（`node_data` の 全ノードを走査するようなことは**しない**．）

[*7] C++ で用意されているライブラリ `priority_queue` では，キー更新が用意されていない．（２０２１年現在．）同じ ID（異なるコスト）をもつノードを許容することで，キー更新したように模倣することもできる．（定理 3.2 で示された計算時間で！）

上の関数の push や pop により，ヒープに保持されるノードの個数が変化する．よって，その値を記憶しておく変数が必要となる．（大域変数にしておくとよい．）この変数により，ゼロ番目からどこまでが実際のデータなのかが分かる．（print_heap ではそこまでを出力する．）

命題 3.3. このプログラムはキー更新が可能な優先度付きキューである．特に，（ヒープの更新後）ヒープの根に優先度の最も高い要素がある．

注 3.7. キー更新は change 関数でなされる．

問 3.6. この命題が成り立つ理由を説明しなさい．

命題 3.4. 任意のＩＤに対して，そのＩＤのコストを $O(1)$ で更新できる．また，ノードの個数を n としたとき，それに伴うヒープの更新を $O(\log n)$ で行える．

問 3.7. この命題が成り立つ理由を説明しなさい．

実装 3.2 (チャレンジ問題)**.** 以下の事項に注意してプログラムを作成しなさい．

- 予期せぬ ID（負の数や３０以上など）が入力されるとエラーが起きる．（データを保持する配列の大きさが３０であるため．）それを回避するプログラムに修正しなさい．
- 既にあるＩＤを追加したり，ヒープにない ID のコストを更新する場合は不都合が生じる．それを回避するプログラムに修正しなさい．（node_idx を利用する．node_idx[i]$= -1$ で，ヒープにないことを意味するようにすればよい．）
- サイズ１以下のヒープの pop は特別扱いすることが必要となる．（なぜ？）

メイン関数へ追加するコード

```
int op, id, cost;
while (cout <<
        "Input an operation number: 1:push, 2:pop, 3:change -> "
        && cin >> op) {
        switch (op) {
        case 1: // push
                cout << "Input an ID number -> ";
                cin >> id;
                cout << "Input its cost -> ";
                cin >> cost;
                push(id, cost);
                break;

        case 2: // pop
                pop();
                break;

        case 3: // change
                cout << "Input an ID number -> ";
                cin >> id;
                cout << "Input its cost -> ";
                cin >> cost;
                change(id, cost);
                break;

        default:
                cout << "Input 1, 2, or 3." << endl;
        }

        print_heap();
}
```

図 3.3: メイン関数へ追加するコード

実装 3.3. 以上の優先度付きキューを**適切に利用して**，ダイクストラのアルゴリズムを，以下の要領に従い，図 3.4 で示されたメイン関数を用いて実装しなさい．

1. まず，３０個の頂点上のグラフを保持する二次元配列 `graph[N][N]`，及び，その隣接リスト `vList[N]`，更に，最短経路木を保持する一次元配列 `path[N]` を（大域変数として）用意する．
2. グラフ $G = (V, E)$ をランダムに生成する．（巻末の付録を参照．）そのグラフを出力する．（これらを関数 `gen_graph()`, `print_graph()` で行う．）
3. 優先度付きキューを初期化する．（これを関数 `init_heap()` で行う．）
4. 始点を $s = 0$，終点を $t = 29$ として，s から t への（G 上の）最短経路をダイクストラのアルゴリズムで探索する．（これを関数 `dijkstra()` で行う．）
5. 到達可能であれば `path` が示す経路を「始点から順に」出力する．（配列 `path` は，終点から辿っていかざるをえず，それを**順に**表示すると逆順になってしまう！スタックか再帰を用いると解決できる．）到達不可能であればその旨を出力する．（グラフはランダム生成なためそういう場合もある．）

ダイクストラ法のメイン関数

```
int main()
{
        gen_graph();
        print_graph();

        init_heap();
        dijkstra();

        print_path();

        return 0;
}
```

図 3.4: ダイクストラ法のメイン関数

3.2　最小全域木問題

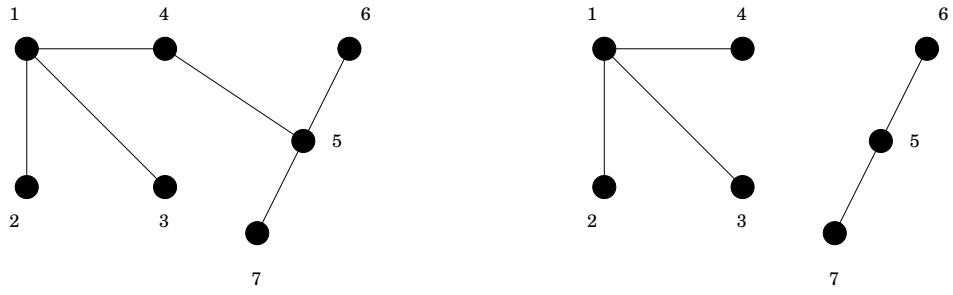
ここでも前節と同様，特に断らない限り，グラフといった場合，連結無向グラフを指すものとする[*8]．

定義 3.4

$G = (V, E)$ をグラフとする．（必ずしも連結しているとは限らない．）G に閉路が存在しないとき，G を**森**という．G が連結グラフであれば**木**という．

注 3.8. 木は森である．（逆は必ずしも成り立たない．）

例 3.2 (木と森)**.** $V = \{1, \ldots, 7\}$ とする．以下の左図は V 上の木，右図は V 上の森である．



(a) 木　　(b) 森

図 3.5: 木と森

事実 3.6. $G = (V, E)$ 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．

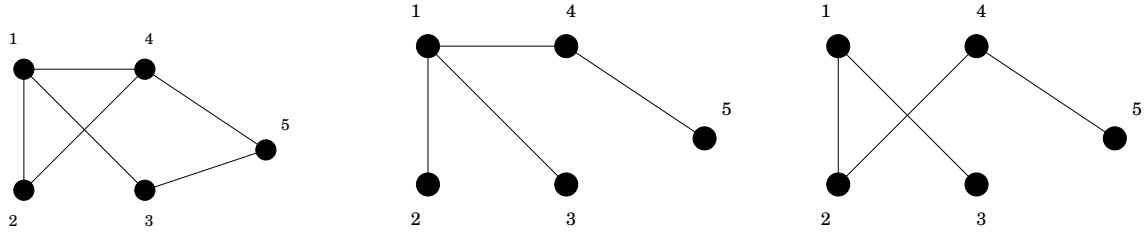
命題 3.5. $G = (V, E)$ が木であれば $|E| = |V| - 1$.

[*8] 木や根付き木については，アルゴリズムとデータ構造のグラフの章を参照．

定義 3.5

$G = (V, E)$ をグラフとする．$G' = (V, E')$ を G の部分グラフとする．（$E' \subseteq E$.）G' が木であるとき，G' を G の**全域木**という．

例 3.3 (全域木)**.** 以下の図の (b), (c) は，(a) で示されたグラフの全域木の例である．



(a) グラフ　(b) 全域木その 1　(c) 全域木その 2

図 3.6: 全域木

注 3.9. ダイクストラのアルゴリズムによって構築される最短経路木は全域木である．

事実 3.7. $G = (V, E)$ をグラフ, T を G の全域木とする．任意の辺 $e = (u, v) \in E \backslash E(T)$ について，次のことが成り立つ．u から v への（唯一の）経路を P とする．（事実 3.6 より.）このとき，任意の辺 $f \in E(P)$ について，$T \setminus \{f\} \cup \{e\}$ は全域木となる．

問 3.8. この事実が成り立つ理由を説明しなさい．

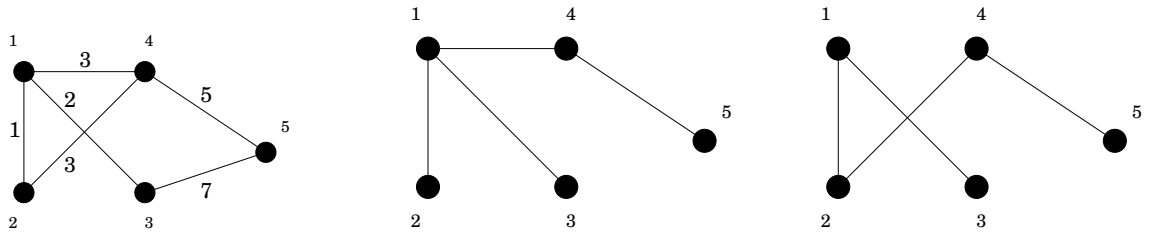
定義 3.6

$G = (V, E)$ をグラフ，$c : E \to \mathbb{R}$ を辺のコストとする．T を G の任意の全域木とする．T の**コスト**を $|T|$ と表して，以下のように定義する．

$$|T| \stackrel{\text{def}}{=} \sum_{e \in E(T)} c(e).$$

T のコストが最小である全域木を，G の**最小全域木**という．

例 3.4 (最小全域木)**.** 以下の図の (b), (c) は，(a) で示されたコスト付きグラフ（辺のコストは辺上に示された値）の最小全域木である．



(a) コスト付きグラフ　(b) 最小全域木その 1　(c) 最小全域木その 2

図 3.7: 最小全域木

最小全域木問題（minimum spanning tree）

- 入力：連結グラフ $G = (V, E)$，コスト $c : E \to \mathbb{R}$
- 解：最小全域木

この問題は，最小全域木を出力する「関数問題」である[*9].

3.2.1 プリムのアルゴリズム

図 3.8 に，最小全域木問題を解く**プリムのアルゴリズム**を示す.

注 3.10. ステップ 4 の任意の繰り返しにおいて，$T = (U, F)$ は（U 上の）木である．ステップ 4 では，次のことがなされる．$e = (u, v) \in (\overline{U} \times U) \cap E$ の中で $c(e)$ が最小となる辺 $e \in E$ を選び，$T = (U \cup \{u\}, F \cup \{e\})$ とする．（$p(u) = v \in U$ となる.）

事実 3.8. ステップ 4 の終了後，T は G の全域木となる[*10]．（G は連結グラフなので.）

問 3.9. １０頂点上の適当な連結グラフ（とコスト） を具体的にあげ，それに対してプリム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 3.6. プリムのアルゴリズム A は最小全域木問題を解く．A の計算時間は $O(|E| \log |V|)$ である.

[*9] コストは非負である必要はない．（最短経路探索問題と違って.）
[*10] 厳密には，ステップ 4 の繰り返し回数についての帰納法により示される.

入力：グラフ $G=(V,E)$，コスト $c: E \to \mathbb{R}$　// $V=[n]$ とする．

1. 関数 $d: V \to \mathbb{R}$ について，$d(1)=-\infty$，任意の頂点 $v \in V \setminus \{1\}$ について $d(v)=\infty$，関数 $p: V \to V$ について，任意の頂点 $v \in V$ について $p(v)=v$，とする．
2. Q を V の全頂点からなる優先度付きキューとする．（Q の優先度は関数 d の値の小さい順とする．）
3. $U=F=\emptyset$ とする．（$T=(U,F)$ が出力となる．）
4. $Q \neq \emptyset$ 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）$U=U \cup \{u\}$，$F=F \cup \{(u,p(u))\}$，とする．
   （c）各 $v \in N_u \setminus U$ について，以下が満たされれば，$d(v)=\alpha$, $p(v)=u$ とする．

$$d(v) \ > \ \alpha \ \stackrel{\text{def}}{=} \ c(u,v).$$

5. $T=(U,F)$ を出力する．

図 3.8: プリムのアルゴリズム

注 3.11. 定理の計算時間は，優先度付きキューに二分ヒープを用いた場合である．フィボナッチヒープと呼ばれるデータ構造を用いると，A の計算時間は $O(|E|+|V|\log|V|)$ に改良される．（用いるデータ構造によって計算時間が異なる！）

証明. まず，アルゴリズムの正当性を示す．$|V|=n$ とする．G の最小全域木（の一つ）を T_0 とする．（一般に，T_0 は唯一とは限らない．）以下，T はアルゴリズムの T を指す．アルゴリズムのステップ 4 では，頂点 u と辺 $(u,p(u))$ が T に追加される．$E(T)$ に追加された順に，$e_1,\ldots,e_{n-1}$ とする[*11]．つまり，ステップ 4 の終了後の T について $E(T)=\{e_1,\ldots,e_{n-1}\}$．このとき，以下のように $k \in [n-1]$ を定義する．

$$k \ \stackrel{\text{def}}{=} \ \arg\min_{i \in [n-1]} \{i : e_i \notin E(T_0)\}.$$

つまり，$\{e_1,\ldots,e_{k-1}\} \subseteq E(T_0)$ かつ $e_k \notin E(T_0)$．（$e_k \in E(T)$．）

主張 3.3. ある $f \in E(T_0) \setminus \{e_1,\ldots,e_{k-1}\}$ が存在して，$|T_0 \setminus \{f\} \cup \{e_k\}| = |T_0|$.

証明. ステップ 4 の i 回目の繰り返し直前の U を U_i とする．（よって，$U_1=\{1\}$，$U_n=V$．）$e_k=(u,v)$ とする．ただし，$v \in U_{k-1}$, $U_k \setminus U_{k-1}=\{u\}$．$T_0$ は全域木

[*11] 事実 3.8 より，ステップ 4 の終了後の T について $|E(T)|=n-1$.

であることから，u から v への経路 P が（唯一に）存在する．（$e_k \notin E(T_0)$ より，$e_k \notin E(P)$.）$u \notin U_{k-1}$ かつ $v \in U_{k-1}$ より，$u' \notin U_{k-1}$ かつ $v' \in U_{k-1}$ となる辺 $f = (u', v') \in E(P) \setminus \{e_1, \ldots, e_{k-1}\}$ が（少なくとも一つ）存在する．

問 3.10. そのような辺 $f = (u', v')$ が存在する理由を説明しなさい．

事実 3.7 より，$T_0 \setminus \{f\} \cup \{e_k\}$ は全域木となる．また，アルゴリズムのステップ 4 での u の選び方から，$c(f) \geq c(e_k)$ が成り立つ．

問 3.11. この不等式が成り立つ理由を説明しなさい．（アルゴリズムの u の選び方が根拠になる理由は？）

以上より，$|T_0 \setminus \{f\} \cup \{e_k\}| \leq |T_0|$ となる．T_0 は最小全域木であるので，$T_0 \setminus \{f\} \cup \{e_k\}$ も最小全域木となる．（不等式は等号で成立する．） ■

この主張より，$\{e_1, \ldots, e_{k-1}, e_k\} \subseteq E(T_0)$ となる最小全域木 T_0 が存在する．同様のことを適用すれば，$\{e_1, \ldots, e_{n-1}\} = E(T_0)$ となる最小全域木 T_0 が存在することになる．ステップ 4 の終了後の T は $E(T) = \{e_1, \ldots, e_{n-1}\}$ であることから，アルゴリズムは最小全域木を出力する．

次に，計算時間を見積もる．ステップ 4 以外にかかる計算時間はいずれも $O(|V|)$ である．以降，ステップ 4 にかかかる計算時間を見積もる．これは，ダイクストラのアルゴリズムの計算時間の解析と全く同じである．

問 3.12. 何がどのように同じなのかを説明しなさい．

よって，ステップ 4 にかかる計算時間は $O(|E| \log |V|)$ となり，プリムのアルゴリズムの計算時間は $O(|E| \log |V|)$ となる． ■

問 3.13. $s = 1$ としたダイクストラのアルゴリズムを（t への最短経路でなく）最短経路木を構築するアルゴリズムと見なした場合，（最小全域木を構築する）プリムのアルゴリズムと似ている．（特に，繰り返し部分.）類似点と相違点を説明しなさい．

3.2.2　クラスカルのアルゴリズム

図 3.9 に，最小全域木問題を解く**クラスカルのアルゴリズム**を示す.

入力：グラフ $G=(V,E)$，コスト $c: E \to \mathbb{R}$

1. コスト c の昇順に E を並べ替える．（並べ替え後の辺の集合を $E=\{e_1, e_2, \ldots, e_m\}$ とする.）
2. $T=(V,\emptyset)$ とする．（T が出力となる.）
3. 任意の $i \in [m]$ について，$|E(T)| < |V|-1$ である限り以下を繰り返す.
   - $T \cup \{e_i\}$ に閉路がないなら $T = T \cup \{e_i\}$ とする.
4. T を出力する.

図 3.9: クラスカルのアルゴリズム

注 3.12. ステップ 3 の任意の繰り返しにおいて, T の各連結成分はそれぞれ木である[*12]. （ステップ 2 では，それぞれはただ一つの頂点だけからなる木である.）

事実 3.9. ステップ 3 の終了後，T は G の全域木となる．（$|E(T)| = |V|-1$ より[*13]命題 3.5 から.）

注 3.13. ステップ 3 において，$T \cup \{e_i\}$ に閉路があるかどうかの「判定方法及び手順」は，それを実現するデータ構造に依存する．詳細は，クラスカル法の実装の節を参照.

問 3.14. １０頂点上の適当な連結グラフ（とコスト） を具体的にあげ，それに対してクラスカル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 3.7. クラスカルのアルゴリズム A は最小全域木問題を解く．A の計算時間は $O(|E| \log |V|)$ である.

[*12] T は「森」である.
[*13] もしそうでなければ，G が連結グラフであることに反する.

証明. まず，アルゴリズムの正当性を示す．$|V| = n$ とする．G の最小全域木（の一つ）を T_0 とする．以下，T はアルゴリズムの T を指す．アルゴリズムのステップ 3 では，e_i が i の小さい順に走査される．$E(T)$ に追加された順に，$e_{i_1}, \ldots, e_{i_{n-1}}$ とする[*14]．($i_1 < i_2 < \cdots < i_{n-1}$, $e_{i_1} = e_1$.) つまり，ステップ 3 の終了後の T について $E(T) = \{e_{i_1}, \ldots, e_{i_{n-1}}\}$．このとき，以下のように $k \in [n-1]$ を定義する．

$$k \stackrel{\text{def}}{=} \arg\min_{j \in [n-1]} \{j : e_{i_j} \notin E(T_0)\}.$$

つまり，$\{e_{i_1}, \ldots, e_{i_{k-1}}\} \subseteq E(T_0)$ かつ $e_{i_k} \notin E(T_0)$．($e_{i_k} \in E(T)$.)

主張 3.4. ある $f \in E(T_0) \setminus \{e_{i_1}, \ldots, e_{i_{k-1}}\}$ が存在して，$|T_0 \setminus \{f\} \cup \{e_{i_k}\}| = |T_0|$.

証明. $e_{i_k} = (u, v)$ とする．以下，$E(T) = \{e_{i_1}, \ldots, e_{i_k}\}$ とする．(つまり，ステップ 3 で $T = T \cup \{e_{i_k}\}$ とした直後の T.) T_0 は全域木であることから，u から v への経路 P が (唯一に) 存在する．($e_{i_k} \notin E(T_0)$ より，$e_{i_k} \notin E(P)$.) このとき，ある辺 $f \in E(P) \setminus E(T)$ が（少なくとも一つ）存在する．

問 3.15. そのような辺 f が存在する理由を説明しなさい．

事実 3.7 より，$T_0 \setminus \{f\} \cup \{e_{i_k}\}$ は全域木となる．また，アルゴリズムのステップ 3 での e_i の選び方から，$c(f) \geq c(e_{i_k})$ であるとしてよい[*15]．これは，次のように背理法により示される．$F = E(P) \setminus E(T)$ とする．(上の問より $F \neq \emptyset$.) すべての $f \in F$ に対して，$c(f) < c(e_{i_k})$ であると仮定する[*16]．このとき，ステップ 3 において，すべての $f \in F$ は e_{i_k} より前に（辺 e_i として）走査された．

問 3.16. この事実が成り立つ理由を説明しなさい．

ステップ 3 において $f \in F$ が走査されたときのことを考える．このとき，$f \notin E(T)$ より，$T \cup \{f\}$ に（唯一の）閉路 C_f がある．(この閉路は f を含む.) よって，それぞれの $f \in F$ についての経路 $C_f \setminus \{f\}$ と $E(P) \cap E(T)$ は，$e_{i_k} = (u, v)$ の両端である u から v への「T 上」の経路 Q となる．

[*14] 事実 3.9 より，ステップ 3 の終了後の T について $|E(T)| = n - 1$.

[*15] つまり，$c(f) \geq c(e_{i_k})$ であるような（$f \notin E(T)$ である）辺 $f \in E(P)$ が存在する．

[*16] これが背理法の仮定である．

問 3.17. この事実が成り立つ理由を説明しなさい.

これより，T 上の経路 Q と辺 $e_{i_k} \in E(T)$ は T が閉路を含むことになり，T が木であることに矛盾する．よって，ある $f \in F$ が存在して，$c(f) \geq c(e_{i_k})$ である．

以上より, $|T_0 \backslash \{f\} \cup \{c_{i_k}\}| \leq |T_0|$ となる．T_0 は最小全域木であるので, $T_0 \backslash \{f\} \cup \{c_{i_k}\}$ も最小全域木となる．（不等式は等号で成立する.）　■

この主張より，$\{e_{i_1}, \ldots, e_{i_k}\} \subseteq E(T_0)$ となる最小全域木 T_0 が存在する．同様のことを適用すれば，$\{e_{i_1}, \ldots, e_{i_{n-1}}\} = E(T_0)$ となる最小全域木 T_0 が存在することになる．ステップ 3 の終了後の T は $E(T) = \{e_{i_1}, \ldots, e_{i_{n-1}}\}$ であることから，アルゴリズムは最小全域木を出力する.

次に，計算時間を見積もる．ステップ 1 にかかる計算時間は（マージソートなどを用いれば）$O(|E| \log |V|)$ である[*17]．ステップ 2, 4 にかかる計算時間はいずれも $O(|V|)$ である．以降，ステップ 3 にかかる計算時間を見積もる．これは命題 3.10 より示される．よって，ステップ 3 にかかる計算時間は $O(|E| \log |V|)$ となり，クラスカルのアルゴリズムの計算時間は $O(|E| \log |V|)$ となる.　■

クラスカル法の実装：Union-Find 木

クラスカル法を実装する．そのためには,「素集合データ構造」を保持する「Union-Find 木」の実装が必要となる.

定義 3.7

A, B を集合とする．$A \cap B = \emptyset$ であるとき，A, B は**素集合**であるという．集合 V の**素集合データ構造**とは，V の分割 $V_1, \ldots, V_k$（V_i, V_j は素集合）を保持するデータ構造である.

例 3.5 (素集合データ構造)**.** $V = [10]$ として，V の三分割 $[V_1, V_2, V_3]$ を，$V_1 = \{4, 6, 8\}$, $V_2 = \{2, 3, 5, 7\}$, $V_3 = \{1, 9, 10\}$, とする．各 V_i をそれぞれリストで保持するデータ構造は，最も単純な素集合データ構造である．他に，各 V_i を根付き木で保持するデータ構造も素集合データ構造である.

[*17] $|E| \leq |V|^2$ より $\log |E| = O(\log |V|)$．よって，$|E| \log |E| = O(|E| \log |V|)$.

問 3.18. クラスカルのアルゴリズムのステップ 3 において，閉路があるかどうかの判定と素集合データ構造がどのような関係にあるかを説明しなさい.

図 3.9 で示されたクラスカルのアルゴリズムのステップ 3 は，素集合データ構造を根付き木で保持すると，以下の図 3.10 のようになる.（T はクラスカルのアルゴリズムの T である.）このアルゴリズムを **Union-Find アルゴリズム**と呼ぶ．また，関数 p が示す（V 上の）有向グラフを **Union-Find 木**と呼ぶ.

注 3.14. Union-Find 木は，一般には（頂点 V 上では）「森」である．ステップ 3 全体の終了後（素集合が一つの集合になったとき）木となる.

3. (a) 関数 $p : V \to V$，関数 $s : V \to \mathbb{N}$ について，すべての頂点 $v \in V$ について $p(v) = v$, $s(v) = 1$ とする.
   (b) 任意の $i \in [m]$ について，$|E(T)| < |V| - 1$ である限り以下を繰り返す.
      i. $e_i = (u_i, v_i)$ として，$a = \texttt{Find}(u_i)$, $b = \texttt{Find}(v_i)$ とする.
      ii. $a \neq b$ なら $T = T \cup \{e_i\}$ として，$\texttt{Union}(a, b)$ とする.

$\texttt{Union}(u, v)$

- 以下のうちのどちらか一方を実行する.
  - $s(u) \geq s(v)$: $p(v) = u$, $s(u) = s(u) + s(v)$.
  - $s(u) < s(v)$: $p(u) = v$, $s(v) = s(v) + s(u)$.

$\texttt{Find}(v)$

1. $p(v) = v$ なら v を返す.
2. $\texttt{Find}(p(v))$ を返す.

図 3.10: Union-Find アルゴリズム

注 3.15. ステップ 3-(b) の任意の繰り返しにおいて，Union-Find 木と T は（一般には）**異なる**.（辺の向きの有・無だけでなく.）

問 3.19. 問 3.14 の解答に対して，省略した閉路があるかどうかの判定手順に，Union-Find アルゴリズムを適用させなさい．このとき，問 3.14 で考案した入力によるアルゴリズムの動作を説明すること．また，注 3.15 にあるよう，Union-Find 木と最小全域木が異なる（または偶然に同一になった）ことを確認しなさい．

補題 3.8 (Union-Find 木)**.** P を Union-Find 木（p が示す森）とする．アルゴリズムのステップ 3-(b) の任意の繰り返しにおいて，以下が成り立つ．

1. P の任意の連結成分 P' について，$p(w) = w$ を満たす頂点 $w \in V(P')$ は唯一であり，P' は w を根とした根付き木である．
2. P の連結成分の頂点集合（の集合）は T のそれに等しい．
3. P の任意の連結成分 P' の深さは高々 $\log |V(P')|$.

注 3.16. 注 3.15 にあるよう，P と T（の辺）は（一般には）異なる．

証明. ステップ 3-(b) の繰り返し回数 $i \in [m]$ についての帰納法により示す．帰納段階を考える．（第 i 回目の繰り返し $e_i = (u_i, v_i)$ を考える．）$a \neq b$ とする．（そうでなければ明らか．）u_i, v_i の属する P の連結成分を P_u, P_v とする．また，u_i, v_i の属する T の連結成分を T_u, T_v とする．帰納仮定より，$P_u \neq P_v$ かつ $T_u \neq T_v$.

問 3.20. 帰納仮定のどの事柄から $P_u \neq P_v$ かつ $T_u \neq T_v$ であるかを説明しなさい．

一般性を失うことなく $s(a) \geq s(b)$ とする．このとき，$p(b) = a$ とすることによって，P_u, P_v は連結される．（$p(w) = w$ となる $w \in V(P_u) \cup V(P_v)$ は $w = a$ のみ．）これより一つ目が示される．また，$T = T \cup \{e_i\}$ より，T_u, T_v は連結される．これより（帰納仮定の二つ目から）二つ目が示される．

三つ目は，P_u, P_v の深さが高々 $\log(s(a) + s(b))$ となることを示せばよい．帰納仮定より P_u, P_v の深さは高々 $\log |V(P_u)|$, $\log |V(P_v)|$ である．P_u の最深の葉が P_v のそれの場合を考えればよい．これは，高々 $\log s(b) + 1$ である．よって，

$$\log s(b) + 1 \;=\; \log s(b) + \log 2 \;=\; \log 2s(b) \;\leq\; \log(s(a) + s(b)).$$

■

命題 3.9. Union-Find アルゴリズムは，クラスカルのアルゴリズムのステップ 3 の閉路があるかどうかを判定する．

証明. 補題の一つ目より，$a = \texttt{Find}(u_i)$, $b = \texttt{Find}(v_i)$ は，（$\texttt{Find}(v)$ のアルゴリズムより）u_i, v_i の属する根付き木の根をそれぞれ示す．補題の二つ目より，$a = b$ が満たされるかどうかで，$T \cup \{e_i\}$ に閉路があるかどうかを判定できる．（$a \neq b$ と $T \cup \{e_i\}$ に閉路がないことは同値である．） ■

命題 3.10. Union-Find アルゴリズムの計算時間は $O(|E| \log |V|)$ である．

証明. ステップ 3-(a) にかかる計算時間は $O(|V|)$ である．以降，ステップ 3-(b) にかかる計算時間を見積もる．ステップ 3-(b) での主な操作は以下の二つである．

- $\texttt{Find}(v)$
- $\texttt{Union}(v)$

前者にかかる計算時間は，補題の三つ目より $O(\log |V|)$．また，後者にかかる計算時間は $O(1)$．よって，ステップ 3-(b) の任意の繰り返しにかかる計算時間は $O(\log |V|)$ となり，ステップ 3-(b) 全体にかかる計算時間は $O(|E| \log |V|)$ となる． ■

実装 3.4. 以上の Union-Find 木（及び Union-Find アルゴリズム）を**適切に利用して**，クラスカルのアルゴリズムを，以下の要領に従い，図 3.11 で示されたメイン関数を用いて実装しなさい．

1. まず，３０個の頂点上のグラフを保持する二次元配列 `graph[N][N]`，及び，その辺の集合を保持するベクター `edge`，更に，最小全域木を保持する二次元配列 `tree[N][N]` を（大域変数として）用意する．（隣接リストを用意する必要はない．）
2. グラフ $G = (V, E)$ をランダムに生成する．（巻末の付録を参照．）そのグラフを出力する．（これらを関数 `gen_graph()`, `print_graph()` で行う．）
3. 辺をコストについて昇順に整列させる．（これはライブラリ関数 `sort` を利用する．）
4. Union-Find 木を初期化する．（これを関数 `make_set()` で行う．）

5. 最小全域木をクラスカルのアルゴリズムで探索する．（これを関数 kruskal() で行う．）
6. 連結であれば tree が示す全域木を出力する．非連結であればその旨を出力する．（グラフはランダム生成なためそういう場合もある．）

メイン関数

```
int main()
{
        gen_graph();
        print_graph();

        sort(edge.begin(), edge.end());
        make_set();
        kruskal();

        print_tree();

        return 0;
}
```

図 3.11: クラスカル法のメイン関数

3.3　ハフマン符号

最小平均符号語長語頭符号

- 入力：$A = \{a_1, \ldots, a_n\}$, $p_1, \ldots, p_n \in \mathbb{Q}^+$ s.t. $\sum_{i \in [n]} p_i = 1$
- 解：A の最小平均符号語長語頭符号

この問題は，最小平均符号語長語頭符号を出力する「関数問題」である．この語頭符号は，**ハフマン符号**として知られている．図 3.12 に，ハフマン符号を構成するアルゴリズムを示す．

入力：$A = \{a_1, \ldots, a_n\}$, $p_1, \ldots, p_n \in \mathbb{Q}^+$ s.t. $\sum_{i\in[n]} p_i = 1$

1. $V = \{v_1, \ldots, v_n\}$（各 v_i が a_i に対応）, $U = \{u_1, \ldots, u_{n-1}\}$ を頂点集合，$E = \emptyset$ を辺集合とする．（$T = (V \cup U, E)$ が出力となる．）
2. 関数 $p : V \cup U \to \mathbb{Q}^+$ について，任意の $i \in [n]$ に対して $p(v_i) = p_i$，任意の $u \in U$ に対して $p(u) = \infty$ とする．
3. Q を $V \cup U$ からなる優先度付きキューとする．（Q の優先度は関数 p の値の小さい順とする．）
4. $Q \neq \emptyset$ である限り以下を繰り返す．（繰り返すごとに $j \in [n-1]$ を j++ する．）
   (a) Q から二度デキューしてその要素を a, b とする．
   (b) $E = E \cup \{(a, u_j), (b, u_j)\}$ とする．
   (c) $p(u_j) = p(a) + p(b)$ とする．
5. $T = (V \cup U, E)$ を出力する．

図 3.12: ハフマン符号

注 3.17. T は u_{n-1} を根とした根付き二分木となる．各頂点 u_j の有向辺 $(u_j, a), (u_j, b)$ に 0/1 のラベルを付ければ，T はラベル付き二分木となる．各 a_i の符号は，根 u_{n-1} から a_i への有向経路のラベルの列となる．よって，T が A のハフマン符号を表す．

問 3.21. $n = 5$ として $p_1, \ldots, p_5$ に適当な確率の値を具体的にあげ，それに対してハフマン符号を構成する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 3.11. ハフマン符号を構成するアルゴリズム A は最小平均符号語長語頭符号である．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．これは，情報理論のテキストを参照[*18]．

次に，計算時間を見積もる．ステップ 1,2,3,5 にかかる計算時間は $O(n)$ である．ス

[*18] 標準的なテキストであれば載っている．

テップ 4 にかかる計算時間は $O(n \log n)$ である[*19]. ■

[*19] ダイクストラのアルゴリズムやプリムのアルゴリズムで用いた優先度付きキューの計算時間に同じ.

第 4 章

動的計画法

ダイクストラのアルゴリズムでは，アルゴリズムのステップ 3 にて，優先度の高い要素（始点からの暫定距離が最も小さい頂点）が選ばれる．グラフ $G = (V, E)$，長さ $\ell : E \to \mathbb{R}_0^+$，始点 $s \in V$，終点 $t \in V$ に対して，ダイクストラのアルゴリズムを実行させた結果，ステップ 3-(a) にてデキューされた頂点の列が，順に，$v_1, \ldots, v_n$ であったとする[*1]．（$v_1 = s$.）任意の $i \in [n]$ について，$S_i = \{v_1, \ldots, v_i\}$ とする[*2]．関数 $D : [n] \times V \to \mathbb{R}_0^+$ を以下のように定義する[*3]．任意の $i \in [n]$，任意の $v \in V$ について，

$$D(i, v) \stackrel{\text{def}}{=} v \text{ の } S_i \text{ からの暫定距離.}$$

このとき，デキューされた順が $v_1, \ldots, v_n$ **であるもとで**，以下の漸化式が成り立つ[*4]．

$$D(i, v) = \min \left\{ D(i-1, v),\ \min_{u \in N_v} \{D(i-1, u) + \ell(u, v)\} \right\}. \tag{4.1}$$

この漸化式は，以下の表 4.1 のように表される．つまり，この漸化式に従って，（任意の $i \in [n]$ について）表の第 i 行目（の任意の $v \in V$）が第 $i-1$ 行目から導かれることを表している．実際，それがアルゴリズムのステップ 3 でなされることである．特に，表の対角（赤で塗られたマス）の値が，ステップ 3 の第 i 回目の繰り返しでデキューされた頂点 v_i の $d(v_i)$ の値，つまり，最短距離 $\delta(v_i)$ となる．よって，任意の $j \in [n]$ について，表の第 j 列について以下が満たされる．

$$\forall k \in [n] : k \geq j [D(k, j) = D(j, j)].$$

[*1] 実際，優先度付きキューによって，この順序が（効率よく）決められる．

[*2] これは，定理 3.2 の証明の S_i に同じ．

[*3] ダイクストラのアルゴリズムの関数 d に同じ．ステップ 3 の第 i 回目の繰り返しの $d(v)$ の値である．

[*4] 定理 4.1 の証明の漸化式 (4.2) を参照．ただし，漸化式 (4.2) では，頂点の順序は仮定されていない．

	v_1	v_2	$\cdots$	v_{i-1}	v_i	v_{i+1}	$\cdots$	v_n
1	↓ $D(1,v_1)$	$D(1,v_2)$	$\cdots$	$D(1,v_{i-1})$	$D(1,v_i)$	$D(1,v_{i+1})$	$\cdots$	$D(1,v_n)$
2	$D(2,v_1)$	↓ $D(2,v_2)$	$\cdots$	$D(2,v_{i-1})$	$D(2,v_i)$	$D(2,v_{i+1})$	$\cdots$	$D(2,v_n)$
$\vdots$	$\vdots$	$\vdots$	↓ $\ddots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
i-1	$D(i\text{-}1,v_1)$	$D(i\text{-}1,v_2)$	$\cdots$	↓ $D(i\text{-}1,v_{i-1})$	$D(i\text{-}1,v_i)$	$D(i\text{-}1,v_{i+1})$	$\cdots$	$D(i\text{-}1,v_n)$
i	$D(i,v_1)$	$D(i,v_2)$	$\cdots$	$D(i,v_{i-1})$	↓ $D(i,i)$	$D(i,v_{i+1})$	$\cdots$	$D(i,v_n)$
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$	↓ $\ddots$	$\ddots$	$\vdots$
n	$D(n,v_1)$	$D(n,v_2)$	$\cdots$	$D(n,v_{i-1})$	$D(n,v_i)$	$D(n,v_{i+1})$	↓ $\cdots$	$D(n,v_n)$

表 4.1: 漸化式 (4.1) を表した表

つまり，表の対角（赤で塗られたマス）の値は，その下に続く同列（薄赤で塗られたマス）の値に引き継がれる[*5]．そのもとで，ステップ 3 の第 i 回目の繰り返しでは，表の（第 i 行の）対角以降（灰色で塗られたマス）の値が求められる[*6]．その結果，（デキューされた順が $v_1, \ldots, v_n$ **であるもとで**）以下が満たされる．

$$i+1 \;=\; \arg\min_{j \in [n] \setminus [i]} \{D(i, v_j)\}.$$

これは，優先度付きキューの先頭の ID が v_{i+1} であることを意味する[*7]．このことから，ダイクストラのアルゴリズムは，漸化式に従って表のマスの値を求めるアルゴリズムと見なすことができる．

問 4.1. 表 4.1 の大きさは $|V|^2$ であるが，ダイクストラのアルゴリズムの計算時間は $O(|E| \log |V|)$ である．$|E| = O(|V|)$ であるようなグラフ $G = (V, E)$ が入力された場合，ダイクストラのアルゴリズムの計算時間は $O(|V| \log |V|)$ となり，表の大きさ $|V|^2$ より小さな計算時間でアルゴリズムが終了する．この理由を説明しなさい．

以上のように，それまでの最適解をもとに順次，最適解を求める手法[*8] が**動的計画法**である．[*9]．本章では，この手法が適用される代表的な問題とアルゴリズムを説明する．

[*5] ステップ 3 でデキューされた頂点 u の $d(u)$ の値が更新されないことに対応する．
[*6] ステップ 3 にて，u の隣接頂点 $v \in N_u$ の $d(v)$ を更新することに対応する．
[*7] よって，表の対角の値はその直上の値にも等しい．その下に続くマスだけでなく．
[*8] ダイクストラのアルゴリズムでは，S_i の最適解をもとに $V \setminus S_i$ の暫定的な最適解が求められる．
[*9] よって，ダイクストラのアルゴリズムを動的計画法と見なすこともできる．$D(i, v)$ の値が（最短な経路を探索する問題なので）最小の頂点を選ぶことに着眼した手法と見なせば貪欲法となる．

4.1 最短経路探索問題（その２）

ここでは，特に断らない限り，グラフといった場合，連結「有向」グラフを指すものとする[*10] [*11]．更に，単に，経路といった場合，単純でない経路**も**指すものとする．問 3.5 で示されたように，ダイクストラのアルゴリズムが正しく動作するための必要十分条件は，辺の長さが非負であることである[*12]．負の長さの辺があるようなグラフの最短経路探索を，ここでは，最小コスト経路探索問題と呼ぶ[*13]．

最小コスト経路探索問題（minimum cost path search）

- 入力：有向グラフ $G = (V, E)$，コスト $c : E \to \mathbb{R}$，始点 $s \in V$，終点 $t \in V$
- 解：s から t への最小コスト単純経路

この問題は，最小コスト単純経路を出力する「関数問題」である．図 4.1 に，最小コスト経路探索問題を（条件付きで）解く**ベルマン・フォードのアルゴリズム**を示す．

注 4.1. ここでは，有向グラフ $G = (V, E)$ について，$v \in V$ の隣接頂点の集合 N_v を以下のように定義する．

$$N_v \stackrel{\text{def}}{=} \{u \in V : (u, v) \in E\}.$$

注 4.2. ステップ 2 の実行途中では，p が「暫定的」な最小コスト経路を，d は p が示す経路のコストを保持する．入力グラフに負のコストの閉路がないなら，p が示す経路は単純である．

問 4.2. １０頂点上の適当な有向グラフ（とコスト，始点・終点）を具体的にあげ（負のコストの閉路がないように），それに対してベルマン・フォード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．

[*10] 有向グラフでは，(u, v) と (v, u) は異なる辺で共存することもある．また，辺 (v, v) はないものとする．

[*11] 有向グラフが連結であるとは，任意の頂点 $s, t \in V$ について，s から t への経路が存在することである．

[*12] ダイクストラのアルゴリズムでは無向グラフを扱ったが，有向グラフでも同様である．

[*13] そのため，長さでなくコスト，更に，最短距離でなく最小コストという用語を用いる．それに応じて，問題の名前を区別して呼ぶ．

入力：有向グラフ $G=(V,E)$，コスト $c: E \to \mathbb{R}$，始点 $s \in V$，終点 $t \in V$

1. 関数 $d: V \to \mathbb{R}$ について，$d(s)=0$，任意の頂点 $v \in V \setminus \{s\}$ について $d(v)=\infty$，関数 $p: V \to V$ について，任意の頂点 $v \in V$ について $p(v)=v$，とする．
2. 任意の $i \in [|V|-1]$ について以下を繰り返す．
   - 任意の $v \in V \setminus \{s\}$ について以下を繰り返す．
     (a) $u = \arg\min_{u \in N_v}\{d(u)+c(u,v)\}$．
     (b) $d(v) > d(u)+c(u,v)$ なら $d(v)=d(u)+c(u,v)$, $p(v)=u$ とする．
3. $d(t)<\infty$ であれば，関数 p によって辿れる t から s への経路の逆を，そうでないなら到達不可能を出力する．

図 4.1: ベルマン・フォードのアルゴリズム

定理 4.1. 入力グラフに負のコストの閉路がないなら，ベルマン・フォードのアルゴリズム A は最小コスト経路探索問題を解く．A の計算時間は $O(|V||E|)$ である．

証明. まず，アルゴリズムの正当性を示す．任意の $i \in [|V|-1] \cup \{0\}$，任意の $v \in V$ に対して，$L(i,v)$ を，s から v への高々 i 個の辺を用いた最小コスト経路のコストとする．ただし，高々 i 個の辺を用いた経路がない場合は，$L(i,v)=\infty$ と定義する．（よって，$L(0,s)=0$，任意の $v \in V \setminus \{s\}$ について $L=(0,v)=\infty$.）

主張 4.1. 最小コスト経路は単純経路となる．

問 4.3. その理由を説明しなさい．

主張 4.2. 任意の $i \in [|V|-1]$，任意の $v \in V$ に対して，以下の漸化式が成り立つ．

$$L(i,v) \;=\; \min\left\{ L(i-1,v), \min_{u \in N_v}\{L(i-1,u)+c(u,v)\} \right\}. \tag{4.2}$$

証明. 任意の $i<k$ について漸化式が成り立つとする．$i=k$ のときを考える．$L(k,v)<\infty$ と仮定する．（そうでなければ，任意の $i \le k$ について $L(i,v)=\infty$，任意の $i \le k-1$，任意の $u \in N_v$ について $L(i,u)=\infty$ である．この事実から，漸化式が成り立つのは明らか.）高々 k 個の辺を用いた s から v への最小コスト経路を，$P=(a_0,a_1,\ldots,a_\ell)$（ただし，$a_0=s$, $a_\ell=v$, $\ell \le k$）とする．このとき，P は以下のうちのどちらか一方である．

- $\ell < k$.（高々 $k-1$ 個の辺を用いた経路.）
- $\ell = k$.（ちょうど k 個の辺を用いた経路.）

前者のとき，単に，$|P| = L(k-1, v)$ である．一方，後者のとき，$a_{k-1} \in N_v$ であることから，$|P| = \min_{u \in N_v}\{L(k-1, u) + c(u, v)\}$ である．よって，$|P|$ はこれら二つのうちの小さい方となる．これは漸化式が成り立つことを意味する． ■

アルゴリズムは，この漸化式に従い，それぞれの $i \in [|V| - 1]$ について，更に，それぞれの $v \in V$ について，$L(i, v)$ の値を求めている．このとき，d が L となり，p がそのコストの経路を保持する．負のコストの閉路がなければ，p が示すグラフは s を根とした根付き木となる．これは，任意の頂点 $v \in V$ について（ステップ 2 の終了後）$d(v) = \delta(v)$ であることを意味する[*14]．

次に，アルゴリズムの計算時間を見積もる．ステップ 1, 3 にかかる計算時間はいずれも $O(|V|)$ である．以降，ステップ 2 にかかる計算時間を見積もる．このために，ステップ 2 の内側の繰り返し全体にかかる計算時間を見積もる．これは，ダイクストラのアルゴリズムの計算時間の解析（の一部）と全く同じである[*15]．

問 4.4. 何がどのように同じなのかを説明しなさい．

よって，ステップ 2 にかかる計算時間は $O(|V||E|)$ となり，ベルマン・フォードのアルゴリズムの計算時間は $O(|V||E|)$ となる． ■

問 4.5. ベルマン・フォードのアルゴリズムの実行過程を，表 4.1 のように表すと似た表になる．表の大きさは $|V|^2$ であるが，ベルマン・フォードのアルゴリズムの計算時間は $O(|V||E|)$ である．（$|V|$ vs. $|E|$.）この理由を説明しなさい．

問 4.6. ベルマン・フォードのアルゴリズムの計算時間が $O(|E| \cdot |V|)$ であり，ダイクストラのアルゴリズムのそれは $O(|E| \cdot \log |V|)$ である．（$|V|$ vs. $\log |V|$.）双方ともに同じ表を求めるアルゴリズムであるのに，計算時間が異なる理由を説明しなさい．

[*14] ステップ 2 の終了後，任意の $v \in V$ について $d(v) = L(|V| - 1, v)$ となる．
[*15] グラフは隣接リストで保持されているものとする．

問 4.7. 負のコストの閉路があった場合，ベルマン・フォードのアルゴリズムは正しく動作しない．つまり，最小コストである「単純」経路を探索できない．反例となる有向グラフ（始点・終点）をあげ，その理由を示しなさい．

問 4.8 (チャレンジ問題)**.** ベルマン・フォードのアルゴリズムを利用すると，負のコストの閉路を（あれば）探索することができる．そのアルゴリズムの擬似コードを示しなさい．

注 4.3. 一般に，最小コスト経路探索問題は NP 困難な問題であり，効率よく解くアルゴリズムが存在しないと予想されている[*16]．

4.2　ナップサック問題

ナップサック問題（knapsack）

- 入力：集合 $U = [n]$，$(w_1, p_1), \ldots, (w_n, p_n) \in \mathbb{N} \times \mathbb{N}$, $W \in \mathbb{N}$
- 解：$S \subseteq [n]$ s.t. $\sum_{i \in S} w_i \leq W$
- 最大化：$\sum_{i \in S} p_i$

以下，一般性を失うことなく，任意の $i \in [n]$ について $w_i \leq W$ とする[*17]．この問題は，重さ w_i の合計が W 以下で価値 p_i の合計が最大になる集合 S を出力する「関数問題」である．図 4.2 に，ナップサック問題を解くアルゴリズムを示す．

注 4.4. 早見表 $T(i, w)$ は，$[i]$ の部分集合の中で，重さの合計が w 以下で価値の合計が最大になる集合を指す．

問 4.9. 5 個の適当なもの（重さ w_i・価値 p_i と W）を具体的にあげ，それに対して動的計画法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．

[*16] 最長経路探索問題（グラフの辺の長さは非負）は NP 困難であり，その問題から帰着される．（還元はすべての辺の距離を負のコストにするだけである．）

[*17] そうでなければ，i が S に含まれることはない．

入力：集合 $U = [n]$，$(w_1, p_1), \ldots, (w_n, p_n) \in \mathbb{N} \times \mathbb{N}$, $W \in \mathbb{N}$

1. 任意の $i \in [n]$，任意の $w \in [W]$ について，早見表 $T(i, w)$ を未定義とする．
2. 任意の $w \geq w_1$ について $T(1, w) = \{1\}$，任意の $w < w_1$ について $T(1, w) = \emptyset$ とする．
3. 任意の $i \in [n] \setminus \{1\}$ について（昇順に）以下を繰り返す．
   - 任意の $w \in [W]$ について，以下を繰り返す．
     (a) $w < w_i$ なら $T(i, w) = T(i-1, w)$ とする．
     (b) そうでないなら，$S_1 = T(i-1, w)$, $S_2 = T(i-1, w - w_i) \cup \{i\}$ とする．
     $P(S_1) > P(S_2)$ なら $T(i, w) = S_1$，そうでないなら $T(i, w) = S_2$ とする．
     ただし，任意の $S \subseteq [n]$ に対して，$P(S) = \sum_{i \in S} p_i$ とする．
4. $T(n, W)$ を出力する．

図 4.2: 動的計画法

定理 4.2. 図 4.2 のアルゴリズム A はナップサック問題を解く．A の計算時間は $O(nW)$ である．

注 4.5. 計算時間 $O(nW)$ のようなアルゴリズムは擬多項式時間アルゴリズムと呼ばれる．（計算時間 nW は必ずしも（入力長の！）多項式とはならない．）

証明. まず，アルゴリズムの正当性を示す．次のような関数 $S : [n] \times [W] \to 2^{[n]}$ を考える．任意の $i \in [n]$，任意の $w \in [W]$ について，

$$S(i, w) \stackrel{\text{def}}{=} \arg\max_{S \subseteq [i]} \left\{ \sum_{j \in S} p_j \ : \ \sum_{j \in S} w_j \leq w \right\}.$$

つまり，$S(i, w)$ は，$[i]$ の部分集合の中で，重さの合計が w 以下で価値の合計が最大になる集合である．

主張 4.3. 任意の $i \in [n] \setminus \{1\}$，任意の $w \in [W]$ について，以下の漸化式が成り立つ．

$S_1 = S(i-1, w)$, $S_2 = S(i-1, w-w_i) \cup \{i\}$ としたとき,

$$S(i,w) \;=\; \begin{cases} \arg \max\limits_{S \in \{S_1, S_2\}} \left\{ \sum\limits_{j \in S} p_j \right\} & : \quad w \geq w_i \\ S_1 & : \quad w < w_i \end{cases} \tag{4.3}$$

ただし，$\sum_{j \in \emptyset} p_j = 0$ とする．漸化式の初期値について，任意の $i \in [n]$ について $S(i,0) = \emptyset$，また，任意の $w \in [W]$ について，$w < w_1$ なら $S(1,w) = \emptyset$，$w \geq w_1$ なら $S(1,w) = \{1\}$ とする．

証明. 任意の $i < k$ について漸化式が成り立つとする．$i = k$ のときを考える．$w_k \leq w$ と仮定する．（そうでなければ，$S(k,w) = S(k-1,w)$ は明らか.）このとき，$S(k,w)$ は以下のうちのどちらか一方である.

- $S(k-1,w)$.（k を含まない.）
- $S(k-1, w-w_k) \cup \{k\}$.（k を含む.）

前者のとき，単に，$S(k,w) = S(k-1,w)$ である．一方，後者のとき，$k \in S(k,w)$ であることから，$S = \min\{S(k-1,w), S(k-1,w-w_k) + p_k\}$ である．これは漸化式が成り立つことを意味する． ■

アルゴリズムの早見表 T は関数 S に同じである．つまり，アルゴリズムは，この漸化式に従い，それぞれの $i \in [n] \setminus \{1\}$，更に，それぞれの $w \in [W]$ について，$S(i,w)$ の値を求めている．最適解は，$S(i,w)$ の定義より $S(n,W)$ である.

次に，計算時間を見積もる．ステップ 1 には計算時間がかからないものとする.. ステップ 2 にかかる計算時間は $O(W)$ である．ステップ 3 にかかる計算時間は $O(nW)$ である．ステップ 4 にかかる計算時間は $O(n)$ である．（$T(n,W) \subseteq [n]$.）よって，アルゴリズムの計算時間は $O(nW)$ である． ■

注 4.6. ナップサック問題は，入力について $(w_i, p_i) \in \mathbb{N} \times \mathbb{N}$ であるが，p_i に整数性がなくてもこの定理は成り立つ.（w_i には整数性が必要である.）

問 4.10. 図 4.2 で示されたアルゴリズムの実行過程を，表 4.1 のように表しなさい．このとき，表のマスの値を求める際，漸化式 (4.3) がどのように用いられるかを説明しなさい.

系 4.3. W が n の多項式であるなら，ナップサック問題は（n の）多項式時間で解くことができる．

命題 4.4. $P = \sum_{i\in[n]} p_i$ とする．このとき，ナップサック問題は $O(nP)$ で解くことができる．

問 4.11 (チャレンジ問題)**.** この命題を証明しなさい．

問 4.12. 図 4.2 で示された動的計画法を，再帰を用いると以下のようになる．再帰関数 `dp_knap`(i, w) の擬似コードを示しなさい．

動的計画法：再帰を用いた記述

入力：：集合 $U = [n]$，$(w_1, p_1), \ldots, (w_n, p_n) \in \mathbb{N} \times \mathbb{N}$, $W \in \mathbb{N}$

1. `dp_knap`(n, W) を出力する．

`dp_knap`(i, w)

⋮

4.3　巡回セールスマン問題

ここでは，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．更に，経路といった場合，単純経路を指すものとする．

定義 4.1

$G = (V, E)$ をグラフとする．任意の頂点 $u, v \in V$ について $(u, v) \in E$ であるとき，G を**完全グラフ**という．$|V| = n$ のとき，V 上の完全グラフを K_n と表す．

巡回セールスマン問題（traveling salesman）

- 入力：完全グラフ $K_n = (V, E)$，長さ $\ell : E \to \mathbb{R}^+$
- 解：最短巡回路

この問題は，最短巡回路を出力する「関数問題」である．まず，図 4.3 に，巡回セールスマン問題を解く単純なアルゴリズムを示す．

入力：完全グラフ $K_n = (V, E)$，長さ $\ell : E \to \mathbb{R}^+$

1. V 上のすべての巡回路から最短なものを出力する．

図 4.3: 単純なアルゴリズム

命題 4.5. 図 4.3 のアルゴリズム A は巡回セールスマン問題を解く．A の計算時間は $O(n!)$ である．

事実 4.1. $n! \approx 2^{n \log n}$.

次に，図 4.4 に，巡回セールスマン問題を解く **BHK（Bellman-Held-Karp）アルゴリズム**を示す．

定義 4.2

$G = (V, E)$ をグラフとする．$P = (p_1, \ldots, p_k)$, $Q = (q_1, \ldots, q_\ell)$ を G の経路とする．ただし，$(p_k, q_1) \in E$, $V(P) \cap V(Q) = \emptyset$．このとき，$P \circ Q$ を以下のように定義する．

$$P \circ Q \stackrel{\text{def}}{=} (p_1, \ldots, p_k, q_1, \ldots, q_\ell).$$

注 4.7. 早見表 $T(U, t)$（$1, t \in U$, $t \neq 1$）は，U の頂点を**すべて**通る頂点 1 から頂点 t への最短「経路」を指す．

注 4.8. ステップ 3-(a) において，$U \setminus \{1, t\} \neq \emptyset$ である．

問 4.13. 5 頂点上の完全グラフに対して，各辺に適当な長さを具体的にあげ，それに対して BHK 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

入力：完全グラフ $K_n = (V, E)$，長さ $\ell : E \to \mathbb{R}^+$ // $V = [n]$ とする．

1. 任意の $U \subseteq V$，任意の $t \in U$ について，早見表 $T(U, t)$ を未定義とする．
2. 任意の $t \in V \setminus \{1\}$ について，$T(\{1, t\}, t) = (1, t)$ とする．
3. 任意の $i \in [n-1] \setminus \{1\}$ について（昇順に），任意の $U \subseteq V$（ただし，$1 \in U$ かつ $|U \setminus \{1\}| = i$）について，以下を繰り返す．
   - 任意の $t \in U \setminus \{1\}$ について，以下を繰り返す．
     (a) 任意の $u \in U \setminus \{1, t\}$ について，$T_u = T(U \setminus \{t\}, u) \circ (t)$ とする．
     (b) $T(U, t)$ を $|T_u|$ が最小の経路 T_u とする．
4. $|T(V, t)| + \ell(t, 1)$ の値が最小となる閉路 $T(V, t) \circ (1)$ を出力する．

図 4.4: BHK アルゴリズム

動作を説明すること．

定理 4.6. BHK アルゴリズム A は巡回セールスマン問題を解く．A の計算時間は $O(n^2 \cdot 2^n)$ である．

証明. まず，アルゴリズムの正当性を示す．次のような関数 $C(U, t)$ を考える．任意の $U \subseteq V$（$1 \in U$），任意の $t \in U \setminus \{1\}$ について，

$$C(U, t) \stackrel{\text{def}}{=} U \text{ の頂点をすべて通る頂点 } 1 \text{ から頂点 } t \text{ への最短経路．}$$

よって，$C(\{1, t\}, t) = (1, t)$．

主張 4.4. 任意の $U \subseteq V$（ただし，$1 \in U$ かつ $|U \setminus \{1\}| \geq 2$），任意の $t \in U \setminus \{1\}$ について，以下の漸化式が成り立つ．

$$C(U, t) \;=\; \arg \min_{P \in \{C(U \setminus \{t\}, u) : u \in U \setminus \{1, t\}\}} \{|P| + \ell(u, t)\} \;\circ\; (t).$$

ただし，$|P|$ は経路 P の長さである．

証明. 任意の $i < k$，任意の $U \subseteq V$（ただし，$|U \setminus \{1\}| = i$）について漸化式が成り立つとする．$i = k$ のとき，更に，任意の $U \subseteq V$（ただし，$|U \setminus \{1\}| = k$）を考える．

$C(U,t) = (a_1, \ldots, a_k)$（ただし，$a_1 = 1$, $a_k = t$）とする．更に，$P_0 = (a_1, \ldots, a_{k-1})$, $a_{k-1} = u_0$ とする．よって，$|C(U,t)| = |P_0| + \ell(u_0, t)$．このとき，

$$|C(U \setminus \{t\}, u_0)| \ = \ |P_0|. \tag{4.4}$$

問 4.14. この事実（式 (4.4)）が成り立つ理由を説明しなさい．

これは漸化式が成り立つことを意味する．

問 4.15. この事実が成り立つ（式 (4.4) から漸化式が導かれる）理由を説明しなさい．

■

アルゴリズムの早見表 T は関数 C に同じである．つまり，アルゴリズムは，この漸化式に従い，それぞれの $i \in [n-1] \setminus \{1\}$，それぞれの $U \subseteq V$（ただし $|U \setminus \{1\}| = i$），更に，それぞれの $t \in U \setminus \{1\}$ について，$C(U,t)$ の値（最短経路）を求めている．最適解は，$C(U,t)$ の定義より，$|C(V,t)| + \ell(t,1)$ が最小となる閉路 $C(V,t) \circ (t,1)$ である．

次に，計算時間を見積もる．ステップ 1 には計算時間がかからないものとする．ステップ 2, 4 にかかる計算時間はいずれも $O(|V|)$ である．ステップ 3 にかかる計算時間は以下の式となる．

$$\sum_{i \in [n-1] \setminus \{1\}} i(i-1) \binom{n-1}{i} \ \leq \ n^2 \cdot 2^n.$$

問 4.16. ステップ 3 にかかる計算時間が上式の左辺になること，また，この不等式が成り立つ理由を説明しなさい．

よって，ステップ 3 にかかる計算時間は $O(n^2 \cdot 2^n)$ となり，アルゴリズムの計算時間は $O(n^2 \cdot 2^n)$ である．■

問 4.17. 図 4.4 で示された BHK アルゴリズムを，再帰を用いると以下のようになる．再帰関数 `dp_tsp`(U,t) の擬似コードを示しなさい．

BHK アルゴリズム：再帰を用いた記述

入力：完全グラフ $K_n = (V, E)$，長さ $\ell : E \to \mathbb{R}^+$ // $V = [n]$ とする．

1. 任意の $t \in V \setminus \{1\}$ について，$P_t = \texttt{dp\_tsp}(V, t)$ とする．
2. $|P_t| + \ell(t, 1)$ の値が最小となる閉路 $P_t \circ (1)$ を出力する．

$\texttt{dp\_tsp}(U, t)$

⋮

注 4.9. BHK アルゴリズムより（真に）高速なアルゴリズムは発見されていない．（例えば，$O(n^{100} \cdot 1.999^n)$ で解く[*18]ことができるかどうかは未解決である．

[*18] $n^2 \cdot 2^n \gg n^{100} \cdot 1.999^n$.

第 5 章

その他

これまでにあげた設計手法には当てはまらないもののうち，重要な（かつ応用されることの多い）アルゴリズムを紹介する．

5.1 ネットワークフロー問題

ここでは，特に断らない限り，グラフといった場合，有向グラフを指すものとする．

定義 5.1

$G = (V, E)$ を有向グラフ，$c : E \to \mathbb{R}^+$ を容量，$s, t \in V$ をソース・シンク，とする．関数 $f : E \to \mathbb{R}_0^+$ が s から t への**フロー**であるとは，以下の条件を満たすことである．

- 容量制約：$\forall e \in E \left[f(e) \leq c(e) \right]$
- 流量保存：$\forall v \in V \setminus \{s, t\} \left[\sum_{e=(u,v) \in E} f(e) = \sum_{e=(v,w) \in E} f(e) \right]$

このとき，f の**流量**を $|f|$ と表記して以下のように定義する．

$$|f| \stackrel{\text{def}}{=} \sum_{e=(s,v) \in E} f(e).$$

事実 5.1. 任意のフロー f について，

$$|f| = \sum_{e=(s,v) \in E} f(e) = \sum_{e=(v,t) \in E} f(e).$$

事実 5.2. $G = (V, E)$ を有向グラフ，$c : E \to \mathbb{R}^+$ を容量，$s, t \in V$ をソース・シンク，とする．s から t への（G 上の）経路がなければ，s から t へのフローは存在しない．

最大フロー問題（maximum flow）

- 入力：有向グラフ $G=(V,E)$，容量 $c:E\to\mathbb{R}^+$，$s,t\in V$
- 解：s から t への最大フロー $f:E\to\mathbb{R}_0^+$

この問題は，最大フローを出力する「関数問題」である．図 5.1 に，最大フロー問題を解くフォード・ファルカーソンのアルゴリズムを示す．

定義 5.2

$G=(V,E)$ を有向グラフ，$c:E\to\mathbb{R}^+$ を容量，$s,t\in V$ をソース・シンク，$f:E\to\mathbb{R}_0^+$ を s から t へのフローとする．このとき，f の**残余ネットワーク**を $G_f=(V,E_f)$ と表記して以下のように定義する．

$$E_f \stackrel{\text{def}}{=} E\cup\{(v,u):\exists(u,v)\in E[f(u,v)>0]\}.$$

このとき，G_f の残余容量 $c_f:E_f\to\mathbb{R}_0^+$ を以下のように定義する．

$$c_f(e) \stackrel{\text{def}}{=} \begin{cases} c(e)-f(e) & : \quad e\in E \\ f(u,v) & : \quad e=(v,u)\in E_f\setminus E \end{cases}$$

入力：有向グラフ $G=(V,E)$, $c:E\to\mathbb{R}^+$, $s,t\in V$

1. 任意の辺 $e\in E$ について $f(e)=0$ とする．（f が最大フローとなる．）
2. G_f に s から t への経路がある限り以下を繰り返す．
   （a）s から t への経路を P とする．
   （b）$e\in E(P)$ の $c_f(e)$ のうち最小の値を $c_{\min}$ とする．
   （c）任意の $e\in E(P)$ について，

$$\begin{array}{rcl} f(e) & = & f(e)+c_{\min} \quad : \quad e\in E \\ f(v,u) & = & f(u,v)-c_{\min} \quad : \quad e=(v,u)\in E_f\setminus E \end{array}$$

3. f を出力する．

図 5.1: フォード・ファルカーソンのアルゴリズム

問 5.1. 7 頂点上の適当なグラフ（と容量，ソース・シンク）を具体的にあげ，それに対してフォート・ファルカーソン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 5.1. フォード・ファルカーソンのアルゴリズム A は最大フロー問題を解く．A の計算時間は $O(|E| \cdot f_{\max})$ である．（グラフ G の最大フローの流量を $f_{\max}$ とする．）

注 5.1. フォード・ファルカーソンのアルゴリズムの計算時間は，入力グラフの最大フローの流量に依存する．それに依存しないアルゴリズムにエドモンズ・カープのアルゴリズムがあり，計算時間は $O(|V| \cdot |E|^2)$ である[*1]．

定義 5.3

$G = (V, E)$ を有向グラフ，$c : E \to \mathbb{R}^+$ をコスト，$s, t \in V$ を任意の頂点，とする．V の二分割 $[A, B]$ を G の**カット**と呼ぶ．特に，$s \in A$, $t \in B$ であるとき，G の s-t **カット**と呼ぶ．カット $[A, B]$ について，以下を**カット集合**と呼ぶ．

$$C(A, B) \stackrel{\text{def}}{=} \{(u, v) \in E : u \in A, v \in B\}.$$

このとき，$C(A, B)$ の**大きさ**を $|C(A, B)|$ と表記して以下のように定義する．

$$|C(A, B)| \stackrel{\text{def}}{=} \sum_{e \in C(A,B)} c(e).$$

定理 5.2 (最大フロー最小カット定理)**.** $G = (V, E)$ を有向グラフ，$c : E \to \mathbb{R}^+$ を容量及びコスト，$s, t \in V$ をソース・シンク，とする．このとき，G の（c を容量とした）最大フローの流量は，G の（c をコストとした）最小 s-t カットの大きさに等しい．

命題 5.3. $G = (V, E)$ を有向グラフ，$c : E \to \mathbb{R}^+$ を容量及びコスト，$s, t \in V$ をソース・シンク，とする．f を G の最大フロー，$[S, T]$ を G の最小 s-t カットとする．このとき，G_f（f の残余ネットワーク）上の s から到達可能な頂点の集合は S である．

[*1] その後，Dinic により $O(|V|^2 \cdot |E|)$ に，更に，データ構造を工夫することにより $O(|V||E| \log |V|)$ に改良された．

系 5.4. 最小 s-t カットを求める問題は多項式時間で計算可能である.

注 5.2. 無向グラフに対しても最小 s-t カットは同様に定義され，多項式時間で求められる.（任意の無向辺 (u, v) に対して，同じコストの有向辺 $(u, v), (v, u)$ をもつ有向グラフを考えればよい.）

問 5.2. この系が成り立つ理由を説明しなさい.

注 5.3. 最大カットを求める問題は NP 困難と呼ばれ，多項式時間では計算不可能であると思われている.

5.2　文字列探索

定義 5.4

Σ をアルファベットとする．$s = s_1 \cdots s_n$, $p = p_1 \cdots p_m$ を Σ 上の文字列とする.（つまり，$s \in \Sigma^n$, $p \in \Sigma^m$.）p が s に**出現する**とは,

$$\exists i \in [n - m + 1], \forall j \in [m][s_{i+j-1} = p_j].$$

このとき，上の条件が満たされる i を**出現番号**と呼ぶ.

以降，一般性を失うことなく $m \leq n$ とする.

文字列探索問題（string searching）

- 入力：文字列 s, p
- 解：p が s に出現するなら出現番号，そうでないなら 0

この問題は，出現番号を出力する「関数問題」である．まず，図 5.2 に，文字列探索問題を解く単純なアルゴリズムを示す.

定理 5.5. 図 5.2 のアルゴリズム A は文字列探索問題を解く．A の計算時間は $O(nm)$ である.

入力：文字列 s, p: $s = s_1 \cdots s_n$, $p = p_1 \cdots p_m$

1. それぞれの $i \in [n - m + 1]$ について以下を実行する．
   - $s_i \cdots s_{i+m-1} = p$ なら i を出力して終了．
2. 0 を出力する．

図 5.2: 単純なアルゴリズム

5.2.1 KMP アルゴリズム

図 5.3 に，**Knuth-Morris-Pratt（KMP）アルゴリズム**を示す．

入力：文字列 s, p: $s = s_1 \cdots s_n$, $p = p_1 \cdots p_m$

1. `lookup_table` を実行して，シフト早見表 $T : [m] \to [m-1] \cup \{0\}$ を作成する．
2. $j = 0$ とする．
3. それぞれの $i \in [n]$ について（昇順に）以下を繰り返す．
   (a) $j > 0$ かつ $p_{j+1} \neq s_i$ である限り $j = T(j)$ とする．
   (b) $p_{j+1} = s_i$ なら $j++$ とする．
   (c) $j = m$ なら $i - m + 1$ を出力して終了する．
4. 0 を出力する．

`lookup_table`

1. $T(1) = 0$ とする．
2. $j = 0$ とする．
3. それぞれの $i \in [m] \setminus \{1\}$ について（昇順に）以下を繰り返す．
   (a) $j > 0$ かつ $p_{j+1} \neq p_i$ である限り $j = T(j)$ とする．
   (b) $p_{j+1} = p_i$ なら $j++$ とする．
   (c) $T(i) = j$ とする．

図 5.3: KMP アルゴリズム

注 5.4. 早見表 $T : [m] \to [m-1] \cup \{0\}$ は以下を意味する．任意の $j \in [m]$ について，

$$T(j) \ = \ \max\{k \in [j-1] \cup \{0\} : p_1 \cdots p_k \text{ が } p_1 \cdots p_j \text{ の接尾語 }\}$$

問 5.3. $\Sigma = \{a, b, c\}$, $n = 11$, $m = 5$ として文字列 s, p を具体的にあげ，それに対して KMP 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 5.6. KMP アルゴリズム A は文字列探索問題を解く．A の計算時間は $O(n+m)$ である．（$m \leq n$ を適用すれば $O(n)$.）

5.2.2　BM アルゴリズム

図 5.4 に，**Boyer-Moore（BM）アルゴリズム**を示す．

入力：文字列 s, p: $s = s_1 \cdots s_n$, $p = p_1 \cdots p_m$

1. 以下のように定義されるシフト早見表 $T : \Sigma \to [m] \cup \{0\}$ を作成する．任意の文字 $c \in \Sigma$ について，

$$T(c) \ = \ \begin{cases} \max\{k \in [m] : p_k = c\} & ; \quad c \text{ が } p_1 \cdots p_m \text{ に出現} \\ 0 & : \quad \text{o.w.} \end{cases}$$

2. $i = j = m$ とする．
3. $i \leq n$ である限り以下を繰り返す．
   （a）$j = 0$ であれば $i+1$ を出力して終了する．
   （b）$s_i = p_j$ であれば，$i--, j--$ とする．
   （c）そうでないなら．$\ell = T(s_i)$ として，i を以下とする．
   - $j - \ell \geq 1$ のとき，$i = i + (m - \ell)$
   - $j - \ell < 0$ のとき，$i = i + (m - j) + 1$

   その上で，$j = m$ とする．
4. 0 を出力する．

図 5.4: BM アルゴリズム

注 5.5. 早見表 T は，c が p に出現する場合，最大（最右）の出現番号を表す．

注 5.6. ステップ 3-(c) において，$s_i \neq p_j$ より，$j - \ell \neq 0$ である．$j - \ell \geq 1$ の場合，$i = i + (m - j) + (j - \ell)$ である．これは以下のような個数の右シフトを意味する．

- $j - \ell \geq 1$ のとき，$j - \ell$
- $j - \ell < 0$ のとき，1

特に，$T(s_i) = 0$ であれば，$i = i + m$ と更新され，$s_{i+1} \cdots s_{i+m}$ との文字列比較になる．

問 5.4. $\Sigma = \{a, b, c\}$, $n = 11$, $m = 5$ として文字列 s, p を具体的にあげ，それに対して BM 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 5.7. BM アルゴリズム A は文字列探索問題を解く．A の計算時間は $O(nm)$ である．

注 5.7. BM アルゴリズムの最悪時の計算時間は，図 5.2 で示された単純なアルゴリズムのそれに等しい．ただ，平均的な計算時間は $O(n/m)$ であることが（理論的に）示されている[*2]．

注 5.8. ガリル規則と呼ばれる文字列比較の効率化を BM アルゴリズムに適用すれば，（最悪時の）計算時間は $O(n)$ となる．

[*2] 現実的には，出現しない文字により m 個のシフトが発生することによる高速化から，grep などの文字列探索に用いられている．

付録 A

マージソートの計算時間の解析

マージソートの計算時間を求めるためには，$y-x+1=n$ として[*1]，$\texttt{msort}(a,x,y)$ にかかる計算時間が $O(n\log n)$ であることを示せばよい．$\texttt{msort}(a,x,y)$ にかかる計算時間を $T(n)$ とする[*2]．$\texttt{msort}(a,x,y)$ のステップ 3 以外にかかる計算時間の合計は $O(n)$ であることから，以下の漸化式が成り立つ：ある定数 c,c' が存在して，

$$\begin{array}{rcl} T(n) & \leq & 2T(\lceil n/2\rceil)+cn, \\ T(1) & \leq & c'. \end{array}$$

これを用いて，ある定数 d が存在して，$T(n)\leq dn\log n+dn$ が成り立つことを n についての帰納法により示す．$n=1$ のときは明らか．（$c'\leq d$ とすればよい．）任意の $n\leq k-1$ について，$T(n)\leq dn\log n+dn$ が成り立つとする．（これが帰納仮定である．）以下，$T(k)\leq dk\log k+dk$ が成り立つことを示す．漸化式より，

$$\begin{aligned} T(k) &\leq 2T(\lceil k/2\rceil)+ck \\ &\leq 2\left(d\lceil k/2\rceil\log\lceil k/2\rceil+d\lceil k/2\rceil\right)+ck \quad (\because \text{帰納仮定}) \\ &\leq 2\left(d(k/2+1)\log(k/1.5)+d(k/2+1)\right)+ck \quad (\because k \text{ は十分に大きい}) \\ &= (dk+2d)\log(k/1.5)+dk+2d+ck \\ &= dk\log k-dk\log 1.5+2d\log(k/1.5)+dk+2d+ck \\ &= (dk\log k+dk)-((d\log 1.5-c)k-2d\log(k/1.5)-2d) \\ &\leq dk\log k+dk. \end{aligned}$$

最後の不等式は，次の事実から得られる．d をある程度大きくすれば $\alpha\stackrel{\text{def}}{=}d\log 1.5-c>0$ となり，更に，k が十分に大きければ $\alpha k\geq 2d\log(k/1.5)+2d$ となる．よって，

$$(d\log 1.5-c)k-2d\log(k/1.5)-2d\ \geq\ 0.$$

[*1] 整列する整数の個数が n であるということである．

[*2] アルゴリズムとデータ構造のテキストにある解析方法とは（若干）異なる．

付録 B

ヒープの構築

ヒープは二分木のデータ構造である．整列問題（昇順）を解くヒープソートで用いるヒープでは，各ノードには一つの整数が保持される[*1]．そして，任意の親・子のペアに対して，親ノードの保持する整数が子ノードの保持する整数より**大きい**，という性質を持っていた．（昇順にソートするため．）

ここでは，これを，各ノードにＩＤ（整数）とコスト（整数）の「ペア」が保持され，親ノードの保持する「コスト」が子ノードの保持する「コスト」より**小さい**，という性質を持つヒープを構築する．以下の要領に従い，図 B.1 で示されたメイン関数を用いて実装しなさい．

1. id（int 型）と cost（int 型）をメンバとした構造体 node を定義する．
2. node を型とする大きさ３０の配列 node_data を（大域変数として）用意する．
3. node_data の要素の初期値として，各要素のＩＤは配列のインデックス番号と同じ値に（ID は０から２９），コストは０から１００までのランダムな[*2]値とする．（これを関数 gen_node_data で行う．）
4. 上で説明したように，（また，ヒープソートでヒープを構築したように）**コスト**に関したヒープを構築する[*3]．（これを関数 make_heap で行う．）
5. （適切にヒープが構築されているかを確認するため）配列の各要素の値（ＩＤとコスト）を出力する[*4]．（これを関数 print_heap で行う．）

[*1] 実際には，（一次元）配列に整数が保持され，その配列を用いてヒープを構築する．アルゴリズムとデータ構造のテキスト，特に，ヒープソートの実装（二分木を配列で模倣する説明）を参照．

[*2] 擬似乱数生成については，メルセンヌツイスターを用いる．次章のグラフのランダム生成，及び，アルゴリズムとデータ構造のテキスト，特に，整列アルゴリズムの実装（整数のランダム生成）を参照．

[*3] アルゴリズムとデータ構造のテキスト，特に，ヒープソートのアルゴリズム及びその実装を参照．

[*4] 一次元配列をインデックス順に出力すればよい．木の深さごとに改行して表示すると見やすくなる．

ヒープ構築のメイン関数

```
int main()
{
        gen_node_data();
        print_heap(); //  ヒープ構築「前」のデータ

        make_heap();
        print_heap(); // ヒープ構築「後」のデータ

        return 0;
}
```

図 B.1: ヒープ構築のメイン関数

事実 B.1. 最小のコストをもつＩＤを（$O(1)$ で）特定することができる.（根のノード，つまり，node_data[0].id がそれである.）

注 B.1. 優先度付きキューは，最も優先度の高いものが（定数時間で）特定できればよい.（「全データ」が整列されている必要はない.）

付録 C

重み付きグラフのランダム生成

長さやコストの付いた重み付きグラフを生成する[*1]．更に，隣接頂点を効率よく走査するために，グラフを隣接リストで保持する．（隣接行列でなく．）以下の要領に従い，図 C.1 で示された関数を用いて実装しなさい．

1. vertex（int 型）と weight（int 型）をメンバとした構造体 edge を定義する．
2. edge を型とする大きさ３０の配列 vList を（大域変数として）用意する．
3. 辺のランダム生成[*2]，辺の重みのランダム付与をする．その上で，（隣接行列を）隣接リストへ変換する．（これが，gen_graph() で行われることである．）

実際には，メイン関数から gen_graph() を呼び出してプログラムの動作確認をする[*3]．

[*1] ダイクストラ法やクラスカル法の実装の際，入力グラフのランダム生成に用いる．詳細はひな形を参照．
[*2] 定数 DENSITY は，辺が生成される割合を示す．頂点次数の平均は N/(DENSITY+1) となる．
[*3] メイン関数にて，print_graph() を呼び出す．その関数にて graph 及び vList の値を出力する．

グラフのランダム生成

```
#define N            30
#define DENSITY       9
#define MAX_WEIGHT   10

int graph[N][N] = {0};

struct neighbor {
        int vertex, weight;
};
list<neighbor> vList[N];

void gen_graph()
{
        mt19937 mt{random_device{}()};
        uniform_int_distribution<int> dist(0,DENSITY);
        uniform_int_distribution<int> wdist(1,MAX_WEIGHT);
        int cost;

        for (int i=0; i<N; i++) // 辺のランダム生成
                for (int j=i+1; j<N; j++)
                        graph[i][j] = dist(mt);

        for (int i=0; i<N; i++) // 隣接リスト（無向グラフ）への変換
                for (int j=i+1; j<N; j++)
                        if (graph[i][j]==1) {
                                cost = wdist(mt);
                                vList[i].push_back({j,cost});
                                vList[j].push_back({i,cost});
                        }
}
```

図 C.1: グラフのランダム生成

問の略解

以下，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．解答の記述要領は，アルゴリズムとデータ構造のテキストで課されたそれに同じ．

1. アルゴリズムとは

1. 略.
2. 略.

2. 分割統治法

1. 略.
2. $a_{i-1}, x \cdot 2^{i-1} < 2^{n+i}$ であり，$x \cdot 2^{i-1}$ は高々 n ビットであるため.
3. 略.
4. z^{middle} を以下とする.

$$z^{\mathrm{middle}} = z^{\mathrm{upper}} + z^{\mathrm{lower}} + \texttt{karatsuba}(x^{\mathrm{lower}} - x^{\mathrm{upper}}, y^{\mathrm{upper}} - y^{\mathrm{lower}}).$$

5. $n = 1$ のときは明らか．そうでないとき，ステップ 4 以外にかかる計算時間は $O(n)$ である.
6. $$\begin{array}{rcl} T(n) & \leq & 4T(\lceil n/2 \rceil) + cn \\ T(1) & \leq & c' \end{array}$$
7. 略.
8. 任意の $i, j \in [n]$，任意の $k \in [n]$ について，$a_{ik}b_{kj}$ は 1 ステップで計算できるため.
9. 次元が奇数になったとき，$A_{12}, B_{12}, A_{21}, B_{21}$ が正方行列でなくなる.
10. 略.

11. 以下の等式が成り立つことを示す.

$$\begin{array}{lcl} M_1 - M_2 + M_5 + M_6 & = & A_{11}B_{11} + A_{12}B_{21} \\ M_2 + M_4 & = & A_{11}B_{12} + A_{12}B_{22} \\ M_3 + M_5 & = & A_{21}B_{11} + A_{22}B_{21} \\ M_1 - M_3 + M_4 + M_7 & = & A_{21}B_{12} + A_{22}B_{22} \end{array}$$

12. $n = 1$ のときは明らか．ステップ 4（の再帰呼び出し）以外にかかる計算時間は $O(n^2)$．ステップ 4 では，7 回の再帰関数の呼び出しがなされる．
13. 一般式は以下のようになる．

$$T(n) \ \leq \ 7^i T(n/2^i) + \left((7/4)^0 + (7/4)^1 + (7/4)^1 + \cdots + (7/4)^{i-1}\right) cn^2.$$

これを計算すると $T(n) \leq 3cn^{\log 7}$ となる．カラツバのアルゴリズムの計算時間（$n = 2^\ell$）の解析を参照.

14. 点列を整列させ，隣接点の距離を求めて，その距離の最小の点対を出力する．
15. 略.
16. 対偶を示す．S の点対でなければ，L, R 間の最近点対の距離が d より大きい．（なぜ？）
17. 二つ（以上）存在したら，L, R 内の最近点対の距離が d（以上）であることに矛盾する．（なぜ？）
18. ステップ 3 及び 6 にかかる計算時間が $O(m)$ であるのは，Q を y 座標の昇順に整列させた Q_y を利用しているため．（呼び出しごとに整列アルゴリズムを実行させない.）ステップ 7 にかかる計算時間は $O(m)$.
19. ある定数 d に対して，$T(m) \leq dm \log m + dm$ が成り立つことを帰納法により示す．（マージソートの計算時間の解析に同じ．付録を参照.）

3. 貪欲法

1. 背理法により示される．つまり，ある $i \in [k]$ について，命題の等式が成立しないと仮定する．このとき，P が最短経路にならない．（なぜ？）
2. 略.
3. ステップ 3 にて，優先度の高い要素（始点からの暫定距離が最も小さい頂点）を選ぶこと.
4. 事実 3.2 より，p は s から u へのある経路を示す.
5. その対偶を示す．つまり，負の長さの辺をもつあるグラフを考案して，ダイクストラのアルゴリズムが適用できないこをを示す．（3 頂点グラフで反例が示せる！）
6. ヒープの性質より.

7. `node_idx[i]` によって，ID が i であるノードを $O(1)$ で特定できる．また，ヒープの深さは $O(\log n)$ であるため，ヒープの更新にかかる計算時間は $O(\log n)$.
8. $T \cup \{e\}$ は唯一の閉路 C をもち，$C = P \cup \{e\}$ であるため.
9. 略.
10. そうでなければ $u \notin U_{k-1}$ かつ $v \in U_{k-1}$ に矛盾する．（なぜ？）
11. $u, u' \notin U_{k-1}$ かつ $v, v' \in U_{k-1}$ より（アルゴリズムの）u の選び方から.
12. 略.（定理 3.2 の優先度付きキューの計算時間の解析を参照.）
13. - 類似点：双方ともに全域木を構築する．（p がそれを示す.）頂点番号 1 だけからなる部分木から始め，全域木の頂点に含める順番が d の小さい順である.
    - 相違点：d, p を更新させる（d に関する）条件.（どのように違う？）
14. 略.
15. そうでなければ T が木であることに矛盾する．（なぜ？）
16. $c(f) < c(e_{i_k})$ であり，ステップ 3 での e_i の選び方より.
17. u から v へ P（の辺 $f \in E(P)$）を辿る際，$f \notin E(T)$ であれば $C_f \setminus \{f\}$ を辿る．$C_f \setminus \{f\}$ は T 上の経路であることから，全体は T 上の経路となる.
18. ステップ 3 の任意の繰り返しにおいて，T の各連結成分は木であり，それぞれの頂点集合 $V_1, V_2, V_3, \ldots$ が素集合である．よって，ステップ 3 の i 回目の繰り返しの $e_i = (u, v)$ に対して，u, v が同じ集合に属することが $T \cup \{e_i\}$（の連結成分）に閉路があることと同値になる.（事実 3.6 より.）
19. 略.
20. 一つ目より，P_u, P_v は a, b を根とした根付きである．よって，$a \neq b$ より（根の唯一性から）$P_u \neq P_v$．二つ目（P, T の連結成分の頂点集合は同じ）より，$T_u \neq T_v$.
21. 略.

4. 動的計画法

1. 表の大きさが計算時間を示すものではない．実際，値を求めることがなされないマスが多くある.
2. 略.
3. 負のコストの閉路がないため.
4. 略.（定理 3.2 の計算時間の解析（隣接頂点の走査）を参照.）
5. 表の各行を求めるのに計算時間が $O(|E|)$ かかる.
6. 問 4.1，問 4.5 の解答を参照.
7. 高々 5 頂点のグラフの反例をあげ，その理由を説明すること.

8. ベルマン・フォードのアルゴリズムのステップ 2 とステップ 3 の間に，以下のステップを追加する．
   - 任意の $e = (u, v) \in E$ について以下を繰り返す．
     $d(u) + c(u, v) < d(v)$ なら，負の長さの閉路がある，と出力して終了．

   このアルゴリズムの正当性は次のように背理法により示される．任意の $e = (u, v) \in E$ について，$d(u) + c(u, v) \geq d(v)$ であったとする．負のコストの閉路を $(a_1, \ldots, a_k)$（$a_1 = a_k$）とする．背理法の仮定より，任意の $i \in [k-1]$ について，

   $$d(a_i) + c(a_i, a_{i+1}) \ \geq \ d(a_{i+1})$$

   この不等式を $i \in [k-1]$ について辺々足し合わせると，

   $$\sum_{i\in[k-1]} d(a_i) + \sum_{i\in[k-1]} c(a_i, a_{i+1}) \geq \sum_{i\in[k-1]} d(a_{i+1})$$
   $$\iff \sum_{i\in[k-1]} c(a_i, a_{i+1}) \geq 0 \quad \left(\because \sum_{i\in[k-1]} d(a_i) = \sum_{i\in[k-1]} d(a_{i+1})\right)$$

   これは，閉路 $(a_1, \ldots, a_k)$ のコストが負であることに矛盾する．
9. 略．
10. 略．
11. 次のような関数 $S : [n] \times [P] \to 2^{[n]}$ を考える．任意の $i \in [n]$，任意の $p \in [P]$ について，

    $$S(i, p) \ \stackrel{\text{def}}{=} \ \arg\min_{S\subseteq[i]} \left\{ \sum_{j\in S} w_j \ : \ \sum_{j\in S} p_j \geq p \right\}.$$

    このとき，任意の $i \in [n] \setminus \{1\}$，任意の $p \in [P]$ について，以下の漸化式が成り立つ．$S_1 = S(i-1, p)$, $S_2 = S(i-1, p-p_i) \cup \{i\}$ としたとき，

    $$S(i, p) \ = \ \arg \min_{S\in\{S_1,S_2\}} \left\{ \sum_{j\in S} w_j \right\}.$$

    ただし, 任意の $p \leq p_1$ について $S(1, p) = \{1\}$, 任意の $p > p_1$ について $S(1, p) = \emptyset$ とする．最適解は $S(n, p_0)$ である．ただし，

    $$p_0 \ \stackrel{\text{def}}{=} \ \arg\max_{p\in[P]} \left\{ p : \sum_{j\in S(n,p)} w_j \leq W \right\}.$$

    アルゴリズムは，この漸化式に従う（図 4.2 で示されたような）動的計画法である．
12. `dp_knap`(i, w)

(a) $i = 1$ のとき，$w \geq w_1$ であれば $\{1\}$ を，そうでないなら $\emptyset$ を返す．

(b) $S_1 = \texttt{dp\_knap}(i-1, w)$, $S_2 = \texttt{dp\_knap}(i-1, w-w_i) \cup \{i\}$ とする．

(c) $P(S_1) > P(S_2)$ なら S_1 を，そうでないなら S_2 を返す．ただし，任意の $S \subseteq [n]$ に対して，$P(S) = \sum_{i \in S} p_i$ とする．

13. 略．

14. もしそうでないなら，$C(U,t) = (a_1, \ldots, a_{k-1}, a_k)$ （$a_{k-1} = u_0$, $a_k = t$）であることに反する．

15. 漸化式では，$u \in U \setminus \{1,t\}$ のうち，つまり，u に対応した $P = C(U \setminus \{t\}, u)$ のうち，$|P| + \ell(u,t)$ が最小になる P を用いた $P \circ (t)$ を $C(U,t)$ としている．式 (4.4) は，$P = P_0$, $u = u_0$ を意味する．（なぜ？）

16. まず，計算時間が左辺になることについて．ステップ 3 の一つ目の任意の $i \in [n] \setminus \{1\}$ についてが，$\sum_{i \in [n-1] \setminus \{1\}}$ に対応する．このとき，式中の i は $|U \setminus \{1\}|$ を意味する．任意の $U \subseteq V$ が $\binom{n-1}{i}$ に対応する．任意の $t \in U \setminus \{1\}$ が i に，任意の $u \in U \setminus \{1,t\}$ が $i-1$ に対応する．不等式が成り立つ理由は以下より．

$$\begin{aligned}
\sum_{i \in [n-1] \setminus \{1\}} i(i-1) \binom{n-1}{i} &\leq n^2 \sum_{i \in [n-1] \setminus \{1\}} \binom{n-1}{i} \\
&\leq n^2 \sum_{i \in [n-1]} \binom{n-1}{i} \\
&= n^2 \cdot 2^{n-1} \\
&\leq n^2 \cdot 2^n.
\end{aligned}$$

17. $\texttt{dp\_tsp}(U,t)$

(a) $U = \{1,t\}$ なら $(1,t)$ を返す．

(b) 任意の $u \in U \setminus \{1,t\}$ について，$P_u = \texttt{dp\_tsp}(U \setminus \{t\}, u)$ とする．

(c) $|P_u|$ が最小の経路 P_u を返す．

5. その他

1. 略．
2. 定理 5.1，注 5.1，命題 5.3 より．
3. 略．
4. 略．

参考図書

本テキストは，アルゴリズムデザインのごく一部（の初歩的なこと）しか扱っていない．アルゴリズムデザイン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. Introduction to Algorithms (4th Edition), Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein, The MIT Press, 2011. （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．）
2. Algorithm Design, Jon Kleinberg, Éva Tardos, Addison-Wesley, 2005. （邦訳：アルゴリズムデザイン，浅野孝夫，浅野泰仁，小野孝男，平田富夫，共立出版，２００８．）
3. 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．）