修 士 論 文

区間グラフにおける区間表現からMPQ-treeを効率よく構成するアルゴリズムに関する研究

北陸先端科学技術大学院大学
情報科学研究科情報処理学専攻

斎藤 寿樹

2007年3月

修 士 論 文

区間グラフにおける区間表現からMPQ-treeを効率よく構成するアルゴリズムに関する研究

指導教官　上原隆平 助教授

審査委員主査　上原隆平 助教授
審査委員　金子峰雄 教授
審査委員　宮地充子 助教授

北陸先端科学技術大学院大学
情報科学研究科情報処理学専攻

510040 斎藤 寿樹

提出年月: 2007 年 2 月

概 要

区間グラフは区間表現を持つグラフで，様々な応用があることが知られている．*MPQ*-tree は区間グラフを表現するデータ構造の 1 つである．*MPQ*-tree は区間グラフの同型性判定に使用でき，多くの区間表現を内包するコンパクトなデータ構造である．既存の *MPQ*-tree を構成するアルゴリズムは区間グラフのグラフ構造を入力とし，多くの場合分けを必要とする．そのため，アルゴリズムは複雑であり，本質的に $O(n+m)$ 時間 · 領域かかる．

本論文では，区間表現を入力とし，直接 *MPQ*-tree を構成する高速なアルゴリズムを示す．このアルゴリズムは場合分けが少なく単純であり，かつ $O(n)$ 時間 · で効率よく実行できる．

目 次

第1章　はじめに

区間グラフは1950年代の後半に数学者のHajösと，分子生物学者のBenzerが独立に研究を始めたグラフクラスである[1]．あるグラフ$G = (V, E)$ $(|V| = n, |E| = m)$が次の性質を満たすとき，Gは区間グラフであるという．

> Vの各頂点は数直線上のある区間に対応し，2つの頂点間に辺が存在するとき，
> またそのときに限り，対応する2つの区間は重なりを持つ．

この区間の集合を区間グラフの区間表現という．区間グラフは一つの区間を時間，温度などと考えることにより，様々な応用がある．特に，バイオインフォマティクスにおいて，DNAの断片を一つの区間と考えることができる．このとき膨大な情報はグラフではなく区間表現で与えられる．このような区間グラフの実際の応用では，区間グラフは区間表現が入力として与えられることが想定される．

MPQ-treeは区間グラフの認識のために考案されたデータ構造で，*PQ*-treeを拡張して得られる[2]．*MPQ*-treeは与えられた2つの区間グラフの同型性の判定に用いることができる．また複数の区間表現を作ることができ，$O(n)$領域で表現できるコンパクトな構造である．このように*MPQ*-treeは区間グラフの応用に有用なデータ構造である．

よって，区間表現を入力として与え，*MPQ*-treeを構成する効率の良いアルゴリズムは，実用上，重要である．

既存のアルゴリズムを用いて，区間表現を入力として*MPQ*-treeを構成する手順を示す[2]．この手順は2通り存在する．1つは次の手順である．まず，入力として与えられた区間表現をグラフ表現に変換する．そして，グラフ表現から対応する*PQ*-treeを構成する．最後に，*PQ*-treeから*MPQ*-treeを構成する．この方法は図1.1の(a)の手順である．この方法はグラフ表現から*PQ*-treeを構成するとき，多くの場合分けを必要とするため，実装が容易ではない．また，グラフ表現が$O(n + m)$領域を用いるので，本質的に$O(n + m)$時間かかってしまう．

2つ目は次の手順である．まず，入力として与えられた区間表現をグラフ表現に変換する．そして，グラフ表現から対応する*MPQ*-treeを直接構成する．この方法は図1.1の(b)の手順である．この方法はグラフ表現から*PQ*-treeを構成するときよりも，場合分けの数を減らしている．しかし，まだ場合分けの数が10以上あり，実装は複雑なものになってしまう．また，*PQ*-treeを経由するときと同様に$O(n + m)$時間かかってしまう．

このように，既存の手法では入力の区間表現をグラフ表現に変換しなければならなかった．

本論文では，区間表現を入力として与え，*MPQ*-treeを直接構成する．このアルゴリズムはグラフ表現を用いず，$O(n)$時間$O(n)$領域で動作する．また，スタック操作を中心と

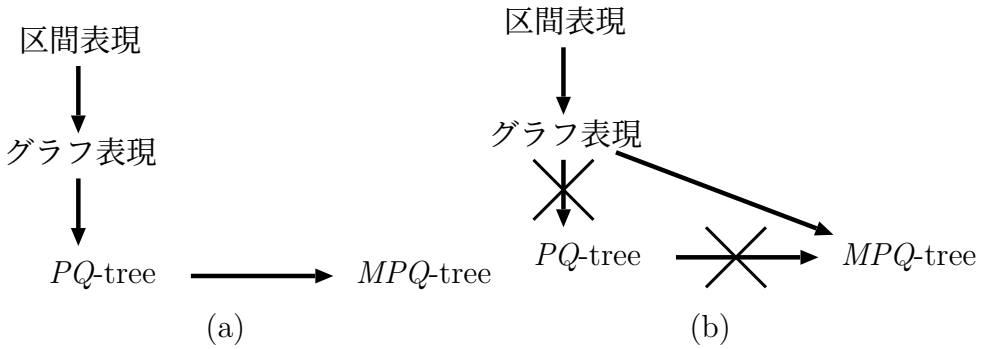


図 1.1: 既存のアルゴリズムでの構成

した単純なアルゴリズムで，実装も容易である (図 1.2).

本論文で提案する手法は図 1.2 のように区間表現から直接 *MPQ*-tree を構成する．しかし，入力の区間表現は冗長であるため，扱いが難しい．よって，処理を簡単にするため，入力の区間表現を冗長性のないコンパクトな区間表現に変換する．そして，コンパクトな区間表現から対応する *MPQ*-tree を構成する．

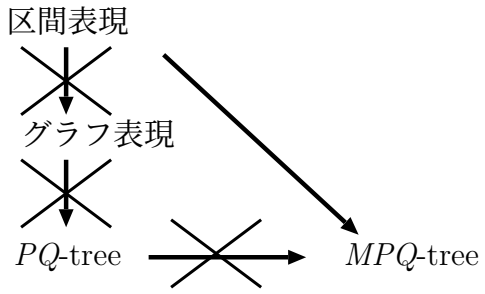


図 1.2: 本論文での構成

本論文では，第 2 章で用語の定義，第 3 章で冗長な区間表現からコンパクトな区間表現を構成するアルゴリズムについて，第 4 章でコンパクトな区間表現から *MPQ*-tree を構成するアルゴリズムについて，第 5 章でまとめと今後の課題について述べる．

第2章　準備

2.1　グラフ

2.1.1　グラフの表記

グラフ $G = (V, E)$ は**頂点**の集合 V と**辺**の集合 E からなる [4]．E の要素は V の要素の 2 つの組である．辺のそれぞれの要素を**端点**と呼ぶ．頂点の数を $n = |V|$ とし，辺の数を $m = |E|$ とする．辺 $e \in E$ の端点が頂点 $v_i, v_j \in V$ であるとき，$e = (v_i, v_j)$ または $e = (v_j, v_i)$ と書く．このとき，v_i と v_j は**隣接**しているという．例えば，図 2.1 のグラフ G は 7 個の頂点 $v_1, \ldots, v_7$ から構成される．辺集合 E は $E = \{e_1, \ldots, e_{10}\}$ であり，$e_1 = (v_1, v_2)$, $e_2 = (v_1, v_3)$ である．

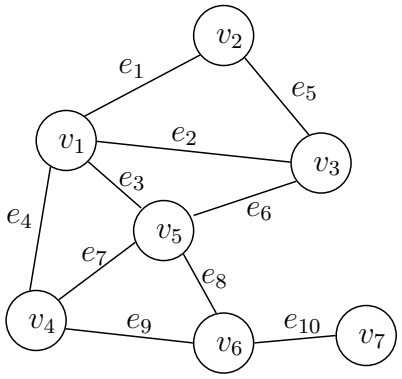


図 2.1: グラフ

2 つのグラフ $G_1 = (V_1, E_1)$ と $G_2 = (V_2, E_2)$ が与えられたとする．V_1 から V_2 への 1 対 1 の写像 f が存在し，任意の $x, y \in V_1$ に対し

$$(x, y) \in E_1 \Leftrightarrow (f(x), f(y)) \in E_2$$

となるとき，G_1 と G_2 は**同型**であるといい，$G_1 \cong G_2$ と書く．図 2.2 において，(a) と (b) は同型なグラフである．しかし，(a) と (c) 及び (b) と (c) は同型なグラフではない．

グラフ $G = (V, E)$ が与えられたとき，$V' \subseteq V$ と $E' \subseteq E$ を満たすグラフ $H = (V', E')$ を G の**部分グラフ**という．頂点の部分集合 $V' \subseteq V$ を与えたとき，$G_{V'} = (V', E_{V'})$ は V' によって G から**誘導される部分グラフ**であるという．ただし，$E_{V'} = \{(v_i, v_j) \mid (v_i, v_j) \in E$ かつ $v_i, v_j \in V'\}$ である．

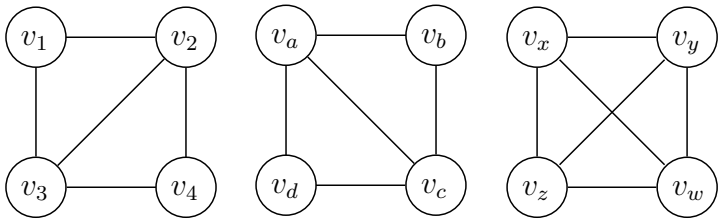


図 2.2: 同型なグラフと同型でないグラフ

グラフ G の相異なるすべての頂点が隣接しているとき，G は**完全グラフ**であるという．n 頂点の完全グラフを K_n と書く．図 2.3 のグラフは左から K_3,K_1,K_6 の例である．

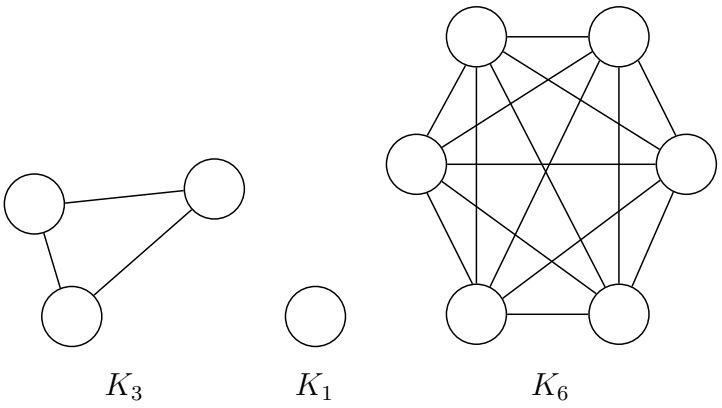


図 2.3: 完全グラフ

グラフ $G = (V, E)$ から頂点集合 $V' \subseteq V$ で誘導される部分グラフが完全グラフであるとき，V' は**クリーク**であるという [1]．また，$|V'| = r(G_{V'} \cong K_r)$ のとき，**r-クリーク**であるという．例えば，G 中の 1 つの頂点は 1-クリークである．G に V' を真に含むクリークがないとき，クリーク V' は**極大**であるという．図 2.5 は図 2.4 のすべての極大クリークである．

グラフ $G = (V, E)$ の頂点 $x \in V$ に隣接している頂点の集合 $V'(= \{y \in V \mid (x, y) \in E\})$ がクリークであるとき，x は**単体的**であるという．図 2.4 において，v_1 に隣接している頂点集合 $\{ v_2, v_3, v_4\}$ はクリークである．よって，v_1 は単体的頂点である．同様に，v_2, v_6 も単体的頂点である．

グラフ $G = (V, E)$ において，$i = 1, 2, \ldots, l$ に対して $(v_{i-1}, v_i) \in E$ であるとき，頂点の列 $[v_0, v_1, v_2, \ldots, v_l]$ は v_0 から v_l への長さ l の**パス**であるという．$i = 1, 2, \ldots, l$ に対して $(v_{i-1}, v_i) \in E$ で，かつ $(v_l, v_0) \in E$ であるとき，頂点の列 $[v_0, v_1, v_2, \ldots, v_l, v_0]$ を長さ $l+1$ の**閉路**と呼ぶ．また，$i, j (i \neq j)$ に対して $v_i \neq v_j$ であるとき，閉路 $[v_0, v_1, v_2, \ldots, v_l, v_0]$ は**単純**であるという．$i, j \in \{0, 1, 2, \ldots, l\} (i \neq j$ かつ $|i - j| \neq 1, l)$ に対して $(v_i, v_j) \in E$ であるとき，(v_i, v_j) は単純閉路 $[v_0, v_1, v_2, \ldots, v_l, v_0]$ の**弦**であるという．

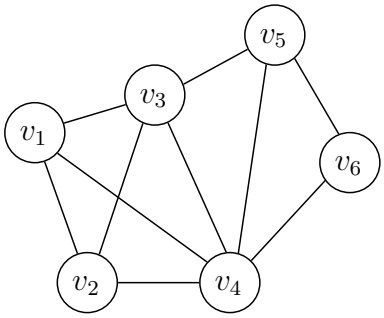


図 2.4: グラフ G

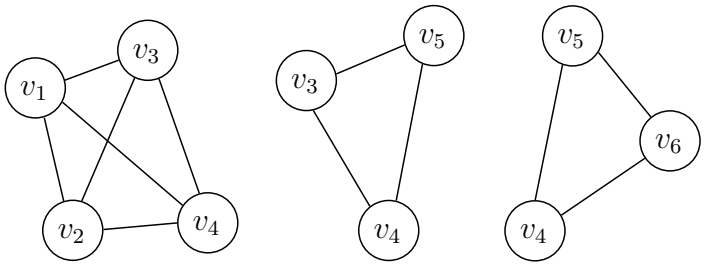


図 2.5: 図 2.4 の極大クリーク

図 2.4 において，頂点の列 $[v_1, v_2, v_3, v_4, v_2, v_3, v_5, v_4, v_6]$ は長さ 8 のパスである．また，$[v_1, v_2, v_4, v_3, v_1]$ は長さ 4 の単純閉路であり，(v_1, v_4) はこの単純閉路の弦である．さらに，単純閉路 $[v_1, v_2, v_3, v_1]$ は弦を持たない閉路である．

グラフ $G = (V, E)$ 上の任意の相異なる 2 頂点間にパスが存在するとき，そのグラフは**連結**であるという．

グラフ $T = (V, E)$ が連結で，かつ単純閉路を持たないとき，T は**木**であるという．根と呼ばれる頂点が 1 つ存在する木を**根付き木**という．根付き木の各頂点のことを**ノード**と呼ぶ．根付き木 $T = (V, E)$ 上の相異なる 2 つのノード x, y に対して，x と根を結ぶパス上に y が存在するとき，y は x の**先祖**であるといい，x は y の**子孫**であるという．またこのとき，$(x, y) \in E$ ならば，y は x の**親**であるといい，x は y の**子**であるという．ある 2 つのノードが共通の親を持つとき，この 2 つのノードは**兄弟**であるという．また，子を持たないノードを**葉**といい，葉ではないノードを**内部ノード**という．図 2.6 は根付き木の例である．v_1 は根付き木 T の根である．v_4 は v_5, v_6, v_7, v_8 の先祖である．v_2, v_3, v_4 は共通の親 v_1 を持つので，兄弟である．また，v_2, v_3, v_6, v_7, v_8 は葉である．

各ノードの子の間に並ぶ順序が定められている根付き木を**順序木**という．順序木 $T = (V, E)$ のノード x が k 個の子を持っているとする．このとき，x のそれぞれの子は第 1 子，第 2 子，$\ldots$，第 k 子となる．2 つのノード y, z が x の子であり，それぞれ第 i 子，第 j 子 $(i, j = 1, 2, \ldots, k$ かつ $i < j)$ のとき，y は z の**兄**であるといい，z は y の**弟**であるという．y が x の第 1 子であるとき，y を x の**長男**という．また，z が x の第 k 子であるとき，z を x の**末子**という．図 2.7 の (a) と (b) は根付き木として同じである．しかし，v_1 の子の順

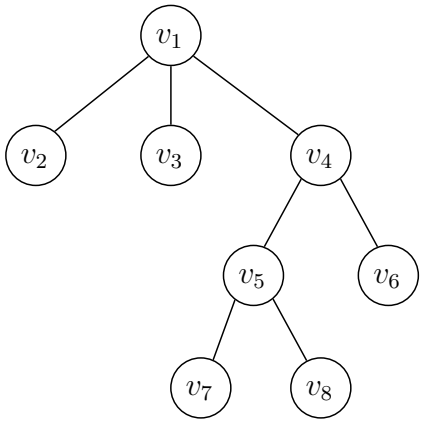


図 2.6: 根付き木

序と v_5 の子の順序が異なるので，(a) と (b) は順序木として異なる木である．

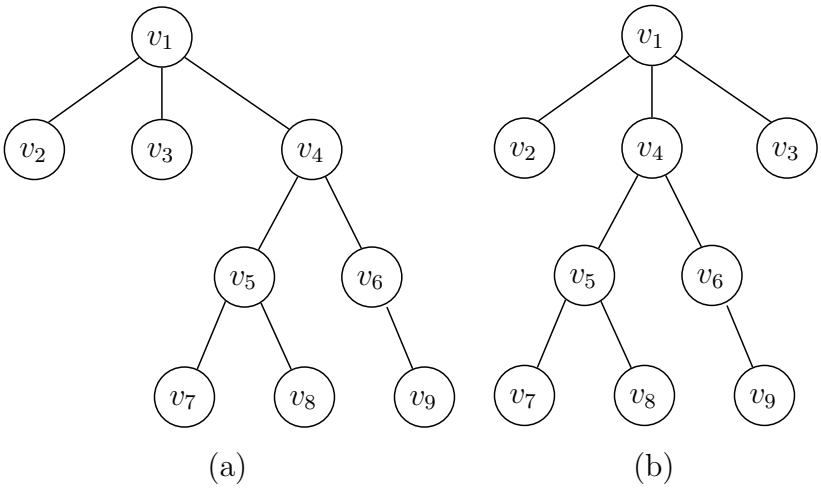


図 2.7: 順序木 T

2.1.2 区間グラフ

グラフ $G = (V, E)$ に対し，数直線上の区間の集合 $\{I_v | v \in V\}$ が存在し，以下を満たすとき，G は**区間グラフ**であるという [1]．すなわち

$v_1, v_2 \in V, (v_1, v_2) \in E \Leftrightarrow I_{v_1}$ と I_{v_2} は共通部分を持つ．

このような区間の集合 $\{I_v\}_{v \in V}$ を G の**区間表現**という．本論文において，それぞれの区間は閉区間とする．つまり，区間 I_v の右端点と区間 I_u の左端点が数直線上の同じ値に存在するとき，対応する 2 つの頂点 u, v は隣接する．区間グラフと対応する区間表現を図 2.8 に示す．

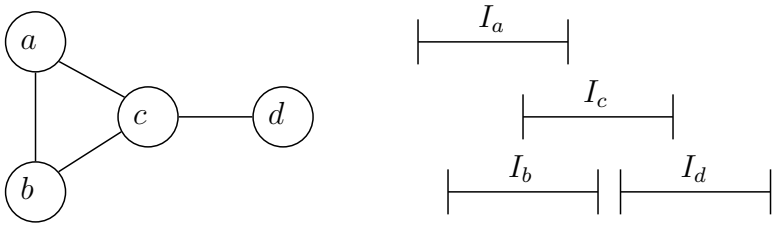


図 2.8: 区間グラフと対応する区間表現

区間グラフは弦グラフと呼ばれるグラフクラスに属する．弦グラフとは長さ 4 以上の全ての単純閉路に必ず弦を持つというグラフクラスである．

定理 2.1. [1] 区間グラフは弦グラフである．

弦グラフは次のような性質を持つ．

定理 2.2. [1] n 頂点の弦グラフは高々n 個の極大クリークを持つ．また，極大クリークの数が n 個であるとき，その弦グラフは辺を持たない．

集合 I の部分集合の族 $\{I_i\}_{i\in V}$ が以下を満たすとき，$\{I_i\}_{i\in V}$ は *Helly Property* を満たすという．

$\forall V' \subset V, \forall i,j \in V'$ において $I_i \cap I_j \neq \phi \Rightarrow \bigcap_{i\in V'} I_i \neq \phi$

定理 2.3. [1] 数直線上の区間の族は *Helly Property* を満たす．また，グラフ G が区間グラフであるとき，またそのときに限り，G のすべての頂点 x に対して，x を含んでいる極大クリークが連続で現れるように G の極大クリークを線形に並べることができる．

2.2 区間表現

本節では本論文で扱う区間表現について述べる．1 章で述べたように，現実の問題では実際のデータを元にした区間表現（時間軸や DNA の文字列など）が入力として与えられることが多い．

図 2.9: DNA データ

本論文では次のような区間表現を入力として扱う．

1. 各区間の端点は数直線上の整数点として，数値の小さい順に与える

2. 複数の端点が同じ整数点を取らない

この区間表現には同一の区間グラフの区間表現が複数存在するという冗長性が存在するため，この区間表現を**冗長な区間表現**と呼ぶ．図 2.10 は冗長な区間表現の例である．

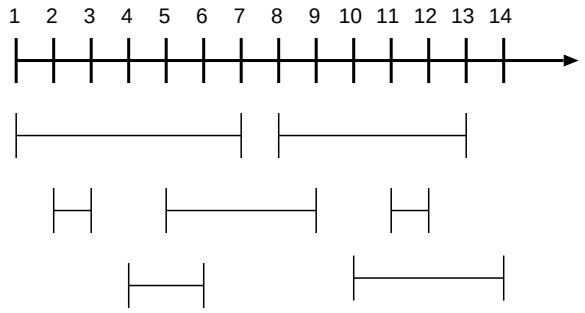


図 2.10: 冗長な区間表現

区間 x に対し，次の値を定義する．

1. $l(x)$：x の左の端点が存在する数直線上の値

2. $r(x)$：x の右の端点が存在する数直線上の値

3. $length(x)$：区間 x の長さ $(r(x) - l(x))$

また，ある区間の端点 e に対し，次の値を定義する．

1. $num(e)$：端点 e の区間の番号

2. $kind(e)$：端点の種類 L, R

先ほど述べたように，入力として与える区間表現には冗長性が存在するため，扱いが難しくなってしまう．そこで，本論文で述べるアルゴリズムでは，まず与えられた区間表現を冗長性のない区間表現である**コンパクトな区間表現**に変換する．ここでコンパクトな区間表現とは以下で定義される区間表現である．それぞれの区間の端点が 1 から始まる連続した整数点上に存在すると仮定する．さらに整数点 i を含む区間の集合を $N[i]$ とする．$N[i] = \phi$ となる大きな i を含まない $N[i] \neq \phi$ である最大の i のことを**最大の整数点**という．このとき，最大の整数点未満の正整数 i に対し，

$$N[i] \setminus N[i+1] \neq \phi \text{ かつ } N[i+1] \setminus N[i] \neq \phi$$

が成り立つ．図 2.11 において，(a) は冗長な区間表現であり，(b) は (a) に対応するコンパクトな区間表現である．また，このコンパクトな区間表現の最大の整数点は 4 である．

コンパクトな区間表現には次の性質がある．

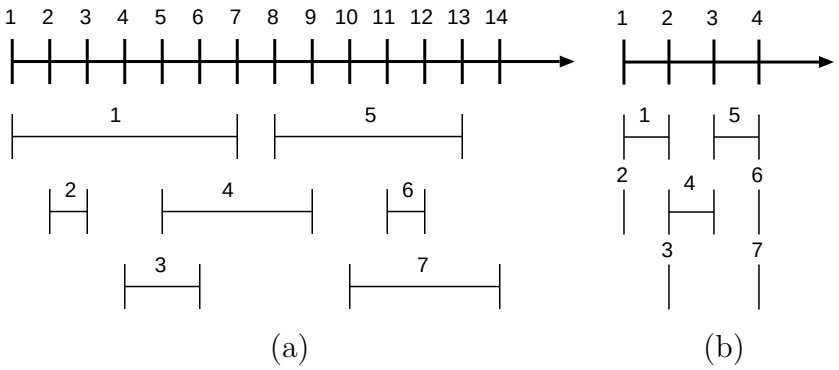


図 2.11: 冗長な区間表現と対応するコンパクトな区間表現

補題 2.4. *CI* を与えられた区間グラフ G のコンパクトな区間表現とする．このとき，*CI* の各整数点 i を含む区間の集合 $N[i]$ が $N[i] \neq \phi$ のとき，$N[i]$ に対応する G の頂点集合 V_i は G の極大クリークになる．

証明. 整数点 i を含む区間の集合を $N[i] = \{I_1, I_2, \ldots, I_j\}$ とする．$I_1, \ldots, I_j$ はそれぞれ $G = (V, E)$ の頂点 $v_1, \ldots, v_j$ に対応しており，$V_i = \{v_1, \ldots, v_j\}$ とする．このとき，V_i の頂点に対応したそれぞれの区間は i を含むので，V_i はクリークである．ここで V_i が G の極大クリークではないと仮定すると，V_i を真に含むクリーク $V_i' = \{v_1, \ldots, v_j, v_{j+1}\}$ が存在する．ここで，定理 2.3 より，区間表現は *Helly Property* を満たす．したがって，V_i' のすべての頂点に対応する区間はある整数点 i' を含む．ここで i' は，この条件を満たし，i に最も近い整数点と仮定する．V_i' は V_i を真に含むので，$i \neq i'$，$V_i \subsetneq V_i' \subseteq N[i']$ となる．

ここで，i と i' の間には，V_i を含むクリークは存在しないことを示す．$i < i'' < i'$ または $i' < i'' < i$ を満たす $i''(|i'' - i| = 1)$ が存在したと仮定する．*CI* は区間グラフの区間表現なので，$N[i] \cap N[i']$ の要素である区間は，すべて点 i'' を含まなければならない．ここで，$N[i] \cap N[i'] = N[i]$ なので，$N[i] \subseteq N[i'']$ となる．$N[i] \subset N[i'']$ は i' に関する仮定に矛盾する．また，$N[i] = N[i'']$ は *CI* がコンパクトであることに矛盾する．よって，$i < i'' < i'$ を満たす i'' は存在せず，$|i - i'| = 1$ となり，2 つの整数点 i, i' は隣接する．

このとき，$N[i]$ と $N[i']$ は隣り合う整数点の区間の集合となり，また $N[i] \setminus N[i'] = \phi$ となるため，コンパクトな区間表現の定義に反する．よって，各整数点の区間の集合は極大クリークである． □

定理 2.5. *CI* を与えられた n 頂点の区間グラフ G のコンパクトな区間表現とする．このとき，*CI* の最大の整数点は高々 n である．

証明. 補題 2.4 から，整数点 i を含む区間の集合 $N[i]$ に対応する G の頂点集合 $V[i]$ は極大クリークになり，同様に $N[i+1]$ に対応する頂点集合 $V[i+1]$ も極大クリークになる．ここで，コンパクトな区間表現の定義 $N[i] \setminus N[i+1] \neq \phi$ かつ $N[i+1] \setminus N[i] \neq \phi$ より，$V[i]$ と $V[i+1]$ は相異なる極大クリークとなる．また，定理 2.1 より区間グラフの極大クリークの数は高々 n である．よって，端点の最大の整数点は高々 n になる． □

区間の番号に対して，一定の規則を仮定できると，構成アルゴリズムを単純化できる．そこで，区間の番号付けに次の規則を当てはめる．

1. $l(x) < l(y)$ ならば $x < y$.
2. $l(x) = l(y)$ かつ $length(x) > length(y)$ ならば $x < y$.

このような順序にしたコンパクトな区間表現を**左端点優先の長さ順序**のコンパクトな区間表現ということにする．

図 2.12 において，(a) は冗長な区間表現から得られたコンパクトな区間表現とする．(a) のコンパクトな区間表現の番号を左端点優先の長さ順序にすると (b) になる．

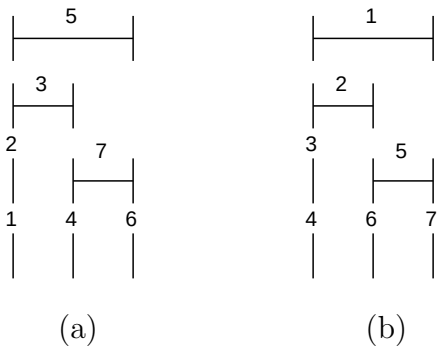


図 2.12: 左端点優先の長さ順序のコンパクトな区間表現

相異なる 2 つの区間 x, y の端点が

$l(x) < l(y) \leq r(x) < r(y)$ または $l(y) < l(x) \leq r(y) < r(x)$

のとき，x と y は**部分交差**しているという．図 2.13(a)，(b) の区間 x と y は部分交差している．

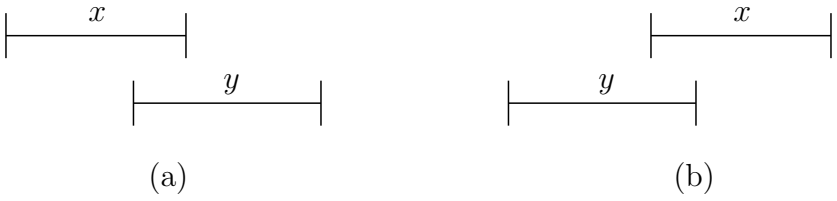


図 2.13: 部分交差

2.3 PQ-tree と MPQ-tree

***PQ*-tree** は 1976 年に Booth と Leuker によって区間グラフを認識するために導入されたデータ構造である [1]．PQ-tree は P ノードと Q ノードの 2 種類の内部ノードを持つ順

序木である．Qノードの子の数は3以上である．また，葉はラベルと呼ばれる整数の集合を持つ．

PQ-tree T に対して，次の2つの操作を有限回当てはめることで，他の PQ-tree T' を得ることができるとき，T と T' は同値であるといい，$T = T'$ と書く．

1. Pノードの子の順序を任意に入れ換える
2. Qノードの子の左右の順序を逆にする

***MPQ*-tree**(Modified PQ-tree) は PQ-tree を拡張したデータ構造である [2]．これは1989年に Korte と Möhring によって区間グラフの認識を行うために提案された．MPQ-tree は区間グラフに対する標準形である．つまり，2つの区間グラフが同型であるとき，またそのときに限り，それぞれに対応する MPQ-tree は同型である．MPQ-tree を用いることで，入力の線形時間で区間グラフの同型性判定を行うことができ，また対応する区間グラフのすべての区間表現を作り出すことができる．

MPQ-tree は Pノードと Qノードの2種類の内部ノードを持つ順序木である．Pノードはラベルを持ち，また葉もラベルを持つ．Qノードの子の数を k とすると，Qノードは k 個のセクションと呼ばれるラベルを持った集合に分割される．

MPQ-tree T と区間グラフ G について以下の関係が成り立つとき，T は G に対応しているという．

> G に対応するコンパクトな区間表現 $CI = \{I_i\}_{i \in V}$ が存在し，T は次のように CIから得られる．
>
> 1. T の葉は CIの各整数点に対応する．
> 2. T の葉は対応する整数点を含んだ長さ0の区間の集合をラベルとして持つ．
> 3. 任意の $i \in V'(\subset V)$ に対して I_iと I_j が部分交差するような $j \in V'$ が存在する，または $I_i = \bigcup_{j \in V'} I_j$ のとき，$\{I_i\}_{i \in V'}$ は1つの Qノードに対応する．
> 4. $\{I_i\}i \in V_i$ を整数点 i の区間の集合とする．V_i' を $V_i' \subset V'$ かつ $V_i' \subset V_i$ とする．任意の V_j' に対して，$\{I_i\}_{i \in V_i'} \neq \{I_j\}_{j \in V_j'}$ となる V_i' が存在するとき $\{I_i\}_{i \in V_i'}$ は1つのセクションに対応する．
> 5. Qノードに対応した区間を除いた区間の集合を $\{I_i\}_{i \in V_{\bar{q}}}$ とする．$\forall i, j \in V'(\subset V_{\bar{q}})$ に対して $I_i = I_j$ のとき，Pノードは $\{I_i\}_{i \in V_i}$ のラベルをもつ．
> 6. 2つのノードまたはセクション N_1, N_2 にそれぞれ対応する区間の集合 $\{I_i\}_{i \in V_i}, \{I_j\}_{j \in V_j}(V_i, V_j \subset V)$ が次を満たすとき，N_1 は N_2 の親になる．
>
> $\bigcup_{j \in V_j} I_j \subset \bigcup_{i \in V_i} I_i$ かつ $\bigcup_{j \in V_j} I_j \subset \bigcup_{k \in V_k} I_k \subset \bigcup_{i \in V_i} I_i$ となる区間の集合 $\{I_k\}_{k \in V_k}(V_k \subset V)$ が存在しない．

定理 2.6. MPQ-tree T に対応する区間グラフ G は一意に定まる．

証明. T は G に対応するコンパクトな区間表現 CI から得られたとする．T に上の逆の操作を行うと，明らかに CI のみを得る．区間表現が一意に決まるので，T に対応する区間グラフは一意に定まる． □

定理 2.7. 区間グラフ G に対応した MPQ-tree を T とし，T と異なる MPQ-tree を T' とする．T に次の 2 つの操作を繰り返し施して T' が得られるとき，またそのときに限り，T' は G に対応する．

1. P ノードの子の順序を任意に入れ換える
2. Q ノードのセクションの順序を逆順にする

図 2.14(a) は区間グラフである．(b) は (a) に対応する PQ-tree であり，(c) は (a) に対応する MPQ-tree である．

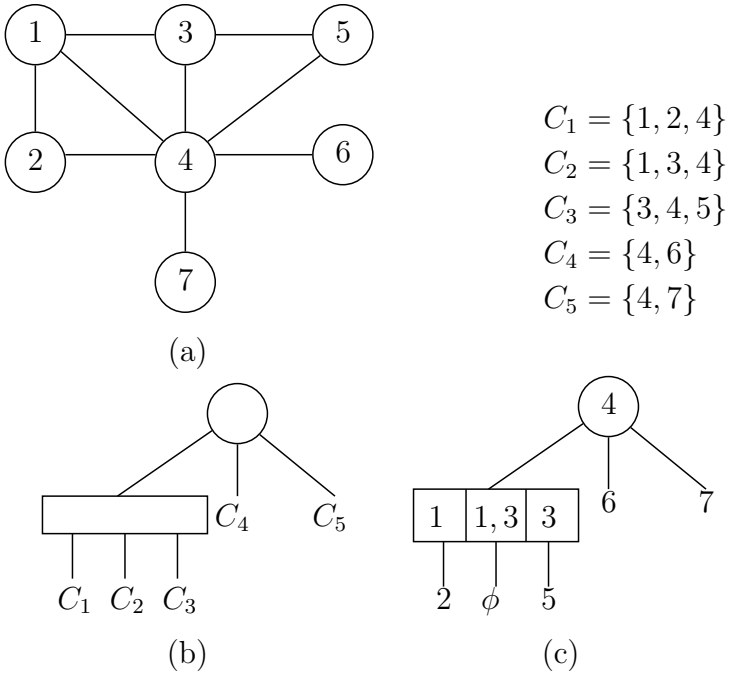


図 2.14: グラフとグラフに対応する PQ-tree と MPQ-tree

MPQ-tree には次の性質がある．

定理 2.8. [2][3] T^* を区間グラフ $G = (V, E)$ に対応する MPQ-tree とする．

1. G のそれぞれの極大クリークは T^* の根から葉までのパス上に存在する頂点ラベルの集合の 1 つと一致する．
2. T^* において，G の頂点 v は 1 つの葉，または 1 つの P ノード，または 1 つの Q ノードの 2 つ以上連続したセクションに 1 回だけ現れる．

3. T^* の根は全ての極大クリークに属している頂点を含む．また，葉は単体的頂点を含む．

MPQ-tree の各セクションやそのセクションの部分木には次のような規則が存在する．

補題 2.9. [2][3] $\hat{Q}$ を *MPQ*-tree の Q ノードとする．$\hat{Q}$ のセクションを $S_1, \ldots, S_k$ の順であるとし，また U_i を $S_i (1 \leq i \leq k)$ より下の部分木に存在する頂点の集合とする．このとき，以下の性質が成り立つ．

1. $S_{i-1} \cap S_i \neq \phi$ $(2 \leq i \leq k)$,
2. $S_1 \subseteq S_2$ かつ $S_k \subseteq S_{k-1}$,
3. $U_1 \neq \phi$ かつ $U_k \neq \phi$,
4. $(S_i \cap S_{i+1}) \setminus S_1 \neq \phi$ かつ $(S_{i-1} \cap S_i) \setminus S_k \neq \phi$ $(2 \leq i \leq k-1)$,
5. $S_{i-1} \neq S_i$ $(2 \leq i \leq k)$,
6. $(S_{i-1} \cup U_{i-1}) \setminus S_i \neq \phi$ かつ $(S_i \cup U_i) \setminus S_{i-1} \neq \phi$ $(2 \leq i \leq k)$.

第3章　コンパクトな区間表現の構成

3.1　コンパクトな区間表現の構成

まず，冗長な区間表現のデータ構造を示す．入力として与えられる冗長な区間表現 RI は実際のデータの各端点を順番に並べたものである．RI には各端点の属する区間の番号とその端点が区間の左端点であるか，右端点であるかの情報が，左にあるものから順に双方向連結リストで格納されている．双方向連結リストの最初の端点を $head(RI)$ とする．端点 x は次の端点へのポインタ $next(x)$ と 1 つ前の端点へのポインタ $prev(x)$ を持つ．また，リストの最後の端点を x としたとき，$next(x) = NIL$ とする．図 3.1(a) は冗長な区間表現であり，そのデータ構造は (b) になる．

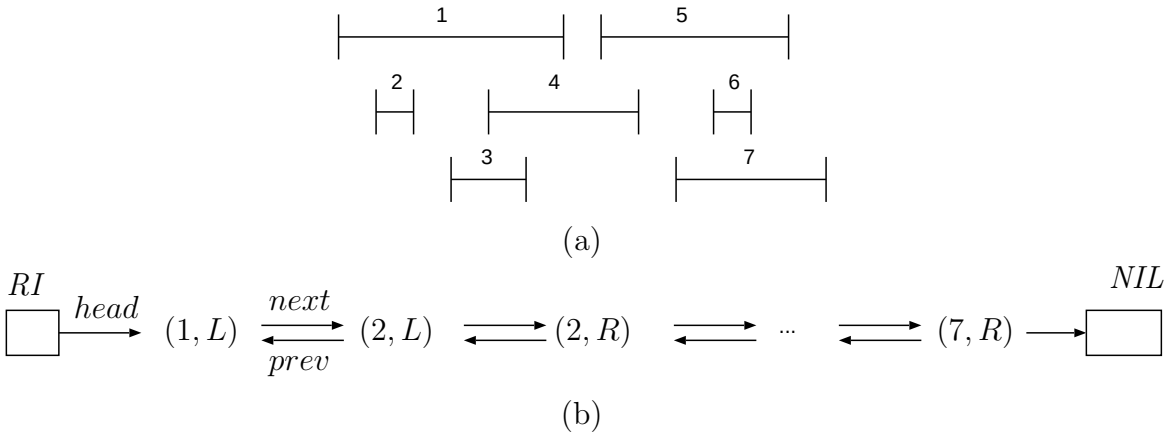
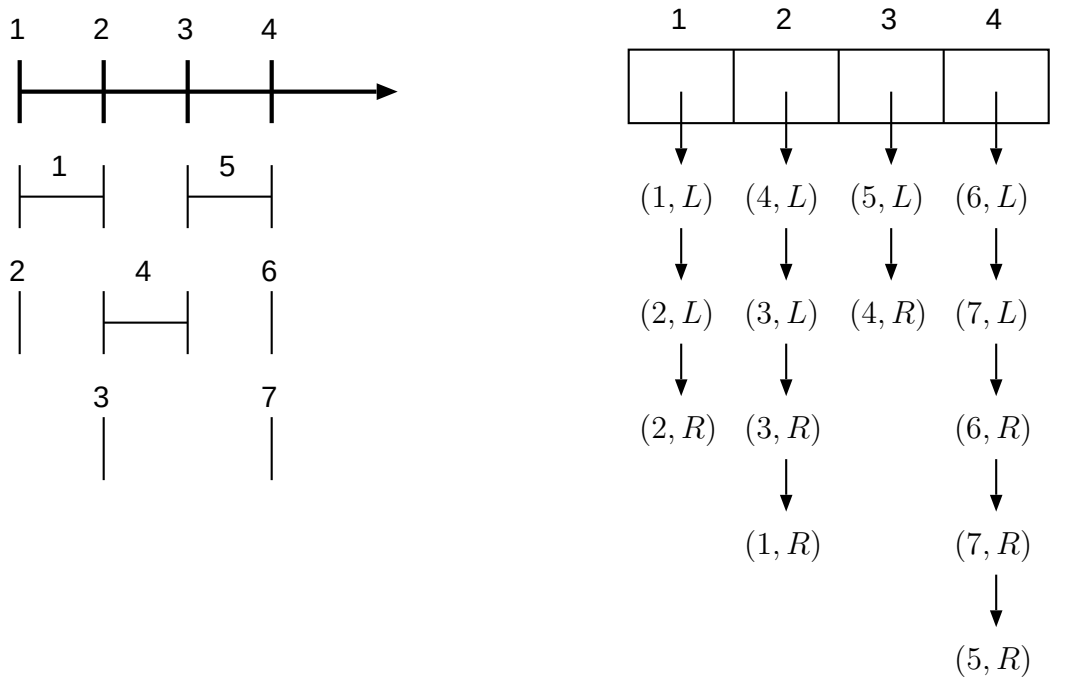


図 3.1: 冗長な区間表現のデータ構造

次に，コンパクトな区間表現のデータ構造を示す．コンパクトな区間表現の最大の整数点を $maxCI$ とする．このとき，コンパクトな区間表現は $maxCI$ 個の要素を持つ配列 $CI[1..maxCI]$ で表される．添字は数直線の整数点であり，配列の各要素はその整数点上に存在する端点の連結リストである．

各端点 x は次の端点へのポインタ $next(x)$ を持つ．また，$head(CI[i])$ は CI の整数点 i での最初の端点へのポインタを表す．整数点の小さい値からすべての端点を読み込む処理をスイープという．コンパクトな区間表現をこのデータ構造にするとスイープがしやすくなる．図 3.2 の (a) はコンパクトな区間表現であり，(b) はそのコンパクトな区間表現に対応するデータ構造である．



(a) (b)

図 3.2: コンパクトな区間表現のデータ構造

アルゴリズム 1 は冗長な区間表現 RI をコンパクトな区間表現 CI に変換することができる．このアルゴリズムの概略を次に示す．

まず，RI の端点を左から順番に読み込んでいく．そして，読み込んだ端点を CI の整数点 i にリストの後ろから挿入する．ただし，左端点を読み込み，かつ 1 つ前に読み込んだ端点が右端点だったとき，整数点を 1 つ右にシフトしてから CI のリストに挿入する．

RI の連続する 2 つの端点が次の順に現れるとき，CI では 2 つの端点は同一整数点上に存在する．

1. 左端点 → 右端点
2. 左端点 → 左端点
3. 右端点 → 右端点

それぞれを図示すると，図 3.3 の (a)，(b)，(c) のときである．

RI の連続する 2 つの端点が

右端点 → 左端点

のとき，2 つの区間は重なりがないので，CI では 2 つの端点は同じ整数点上に存在しない．図 3.4 のとき，2 つの端点は同じ整数点上に存在しない．

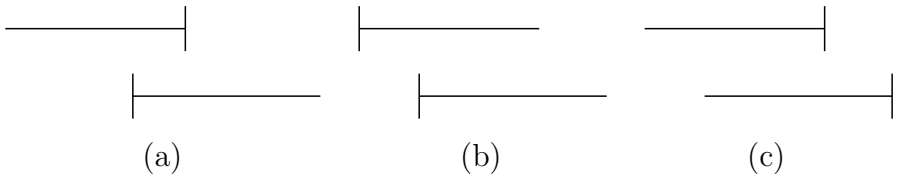


図 3.3: 同一整数点上に存在する端点

図 3.4: 同一整数点上に存在しない端点

補題 3.1. アルゴリズム 1 で得られる，任意の $CI[i]$ に左端点と右端点が存在する．

証明. 任意の $CI[i]$ に左端点が存在することを証明する．

まず，$i = 1$ とすると，$CI[i]$ は区間表現の最も左に存在する端点を表すので，必ず左端点が存在する．

$i = k$ $(k > 1)$ のとき，$CI[k]$ に左端点が存在すると仮定する．$CI[k+1]$ に端点が挿入されるならば，必ずアルゴリズムの 6 行目が実行される．アルゴリズムの 6 行目が実行されるのは，読み込んだ端点が左端点で、かつ 1 つ前の端点が右端点であるときにのみである．そのため，$CI[k+1]$ にも左端点が存在する．

よって，任意の $CI[i]$ に左端点が存在する．同様にして，任意の $CI[i]$ に右端点も存在する． □

定理 3.2. 入力に冗長な区間表現を与えたとき，アルゴリズム 1 はコンパクトな区間表現を出力する．

証明. 補題 3.1 から，任意の $i \in \{1, \ldots, maxCI - 1\}$ に対して，$CI[i]$ に右端点が必ず存在し，$CI[i+1]$ に左端点が必ず存在する．ここで，$maxCI$ は CI の最大の整数点である．アルゴリズム 1 は，添字の小さな $CI[1]$ から順に端点を格納している．そのため，$CI[i]$ に右端点が存在する区間は $CI[i+1]$ に存在しない．また，同様に $CI[i+1]$ に左端点が存在する区間は，$CI[i]$ に存在しない．よって，$CI[i]$ と $CI[i+1]$ には必ず異なる区間が存在するので，$N[i] \setminus N[i+1] \neq \phi$ かつ $N[i+1] \setminus N[i] \neq \phi$ が成り立つ．よって，アルゴリズム 1 の出力する区間表現はコンパクトな区間表現である． □

定理 3.3. n 個の区間の冗長な区間表現が入力されたとき，アルゴリズム 1 は $O(n)$ 時間と $O(n)$ 領域で実行できる．

証明. n 個の区間が入力されているので，読み込まれる端点の数は $2n$ である．よって，アルゴリズムの 5 行目の $while$ 文は $O(n)$ 回繰り返される．アルゴリズムの 3 行目，9 行目に使用されているリストの最後尾への挿入操作はリストの最後のポインタを保持しておけば，$O(1)$ 時間で実行できる．その他の行は代入操作なので $O(1)$ 時間で実行できる．よって，全体の実行時間は $O(n)$ である．

Algorithm 1: コンパクトな区間表現の構成

Input: 冗長な区間表現: RI
Output: コンパクトな区間表現: CI

```
1  i ← 1;
2  e ← head(RI);
3  リスト CI[1] の最後尾へ (num(e), kind(e)) を挿入;
4  e ← next(e);
5  while e ≠ NIL do
6      if (kind(e) = L かつ kind(prev(e)) = R) then
7          i = i + 1;
8      end
9      リスト CI[i] の最後尾へ (num(e), kind(e)) を挿入;
10     e ← next(e);
11 end
12 return CI;
```

n 個の区間が入力されているとき，各区間の端点は右端点と左端点が存在する．そのため，端点の総数は $2n$ 個となる．よって，冗長な区間表現のデータ構造に必要な領域は $O(n)$ である．同様に，コンパクトな区間表現のデータ構造に必要な領域も $O(n)$ である． □

3.2 左端点優先の長さ順序のコンパクトな区間表現の構成

左端点優先の長さ順序のコンパクトな区間表現のデータ構造はコンパクトな区間表現と同じである．さらに，コンパクトな区間表現のデータ構造の端点の格納順に次の規則を与える．

> 1 つの整数点上で，番号の小さい順に左端点を格納する．次に番号の大きい順に右端点を格納する．

図 3.5(a) は冗長な区間表現から得たコンパクトな区間表現のデータ構造であり，(b) は (a) に左端点優先の長さ順序の規則を与えた図である．アルゴリズム 2 を用いることで，コンパクトな区間表現を左端点優先の長さ順序のコンパクトな区間表現に変換することができる．

ここで，アルゴリズム 2 の動作について示す．入力のコンパクトな区間表現を CI とする．出力は左端点優先の長さ順序のコンパクトな区間表現 $LLorderCI$ である．このアルゴリズムは CI に対してスイープを行う．どの区間も左端点から読み込まれる．スイープが $CI[i-1]$ の端点をすべて読み込んだとする．また，ここまでに読み込んだ左端点の数を h とする．$CI[i]$ を見たとき，$(x_1, L) \rightarrow (x_2, L) \rightarrow \cdots \rightarrow (x_k, L)$ の順で左端点が格納されてい

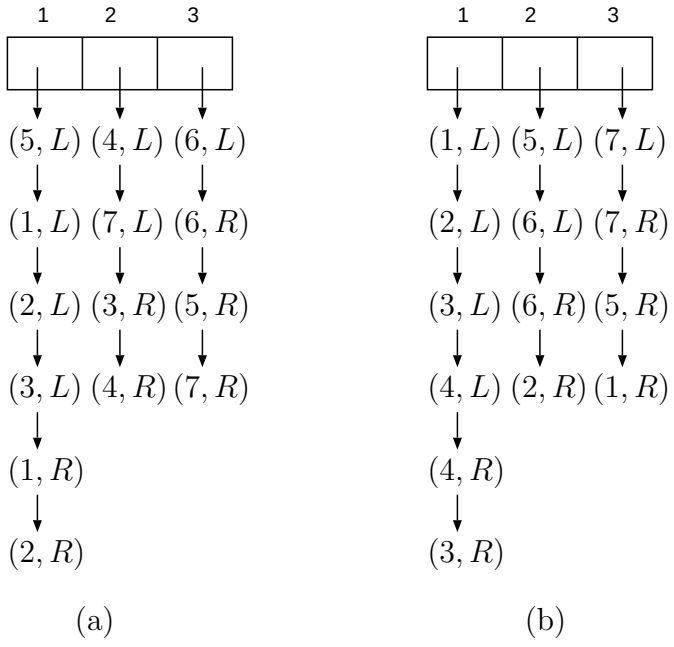


図 3.5: 左端点優先の長さ順序のコンパクトな区間表現のデータ構造

たとする．(x_1, L) が $h+1$ 個目に読み込まれた左端点であるので，左端点優先の長さ順序のコンパクトな区間表現 $LLorderCI[i]$ の左端点は $(h+1, L) \to (h+2, L) \to \cdots \to (h+k, L)$ の順番に格納される．図 3.6 はその様子である．図の (a) は CI であり，(b) は $LLorderCI$ である．

ある区間の左端点 (x_a, L) が $CI[i]$ に存在し，右端点 (x_a, R) が $CI[j]$ に存在したとする $(i \leq j)$．このとき，x_a は $h+1, \ldots, h+k$ のうちのいずれかである．左から端点を読み込んでいくので，区間の長さが短い順に番号を付けることが簡単である．左端点優先の長さ順序から，左端点が同じ整数点上に存在するとき，区間は長さの短い方から大きい番号を付ける．よって，$CI[i]$ に左端点が存在する区間は右端点を読み込まれた順に $h+k, h+(k-1), \ldots, h+1$ の番号を付ける．そして，x_a は $h+b$ に番号付けされるとき，$h+b$ の右端点が j に存在していたことを配列 $ExistR$ に格納しておく．スイープが終わってから，$ExistR$ を後ろから参照し，$LLorderCI$ の区間の番号の大きい順から右端点を $LLorderCI$ に格納していく．その様子を図 3.7 に示す．

定理 3.4. アルゴリズム 2 を用いると，コンパクトな区間表現は左端点優先の長さ順序になる．

証明. アルゴリズム 2 の 6,7 行目から，左端点を読み込んだとき，読み込んだ数の左端点を 1 つリストの最後尾に格納する．そのため，$LLorderCI[i]$ に左端点を番号の小さい値から連続に格納する．

CI の左端点 e_l を読み込んだとき，アルゴリズムの 8 行目で $ExistL[num(e_l)]$ に左端点が存在した CI の整数点を与える．

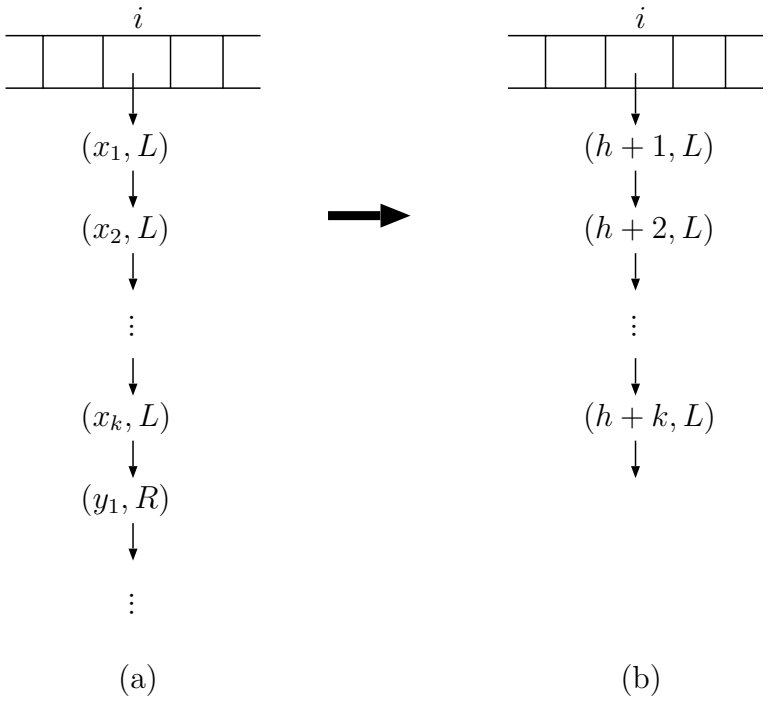


図 3.6: 左端点の格納

9行目で，$LLorderCI[i]$ に存在する左端点の番号である $Numleft$ を $B[i]$ に与え，$LLorderCI[i]$ への番号の候補として与える．

11行目は右端点 e_r を読み込んだときに実行される．$num(e_r)$ の左端点は整数点 $ExistL[num(e_r)]$ に存在した．CI の e_r の区間の番号は $LLorderCI$ で $B[ExistL[num(e_r)]]$ になる．そこで，$ExistR[B[ExistL[num(e_r)]]]$ に右端点が存在した整数点 i を格納しておく．e_r に $B[ExistL[nume_r]]$ を与えたので，12 行目で，$B[ExistL[nume_r]]$ をデクリメントし次の候補とする．

17,18 行目で，$ExistR$ を添字の大きい順に $LLorderCI$ の最後尾へ挿入していく． □

定理 3.5. アルゴリズム 2 は n 個の区間が与えられたとき，$O(n)$ 時間と $O(n)$ 領域で実行できる．

証明. アルゴリズム 2 の 2，3，4，14 行でコンパクトな区間表現のすべての端点を読み込んでいる．つまり，スイープを 1 回行っており，端点を読み込む回数は $O(n)$ 回である．また，17 行目の for 文は明らかに $O(n)$ 回繰り返す．その他の行の実行時間は $O(1)$ であるので，アルゴリズム全体の実行時間は $O(n)$ である．

アルゴリズム 2 で使用される配列のサイズはすべて高々 n である．また，コンパクトな区間表現の領域は $O(n)$ である．よって，このアルゴリズムで必要とする領域は $O(n)$ である． □

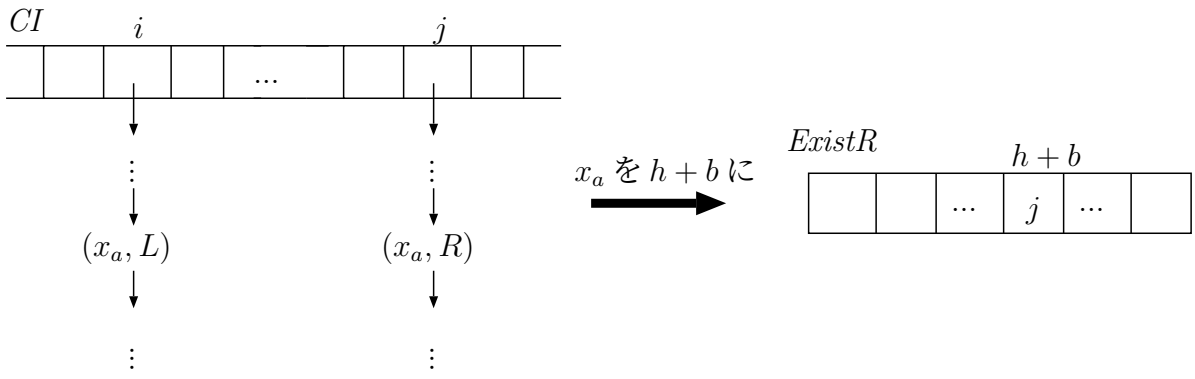


図 3.7: 区間の番号付け

Algorithm 2: 左端点優先の長さ順序のコンパクトな区間表現の構成

Input: コンパクトな区間表現 CI

Output: 左端点優先の長さ順序のコンパクトな区間表現 $LLorderCI$

```
1  NumLeft ← 0;
2  for i = 1 to maxCI do
3      e ← head(CI[i]);
4      while e ≠ NIL do
5          if kind(e) = L then
6              NumLeft ← NumLeft + 1;
7              リスト LLorderCI[i] の最後尾へ (NumLeft, L) を挿入;
8              ExistL[num(e)] ← i;
9              B[i] ← NumLeft;
10         else                                   /* kind(e) = R のとき */
11             ExistR[B[ExistL[num(e)]]] ← i;
12             B[ExistL[num(e)]] ← B[ExistL[num(e)]] − 1;
13         end
14         e ← next(e);
15     end
16 end
17 for i = n downto 1 do
18     リスト LLorderCI[ExistR[i]] の最後尾へ (i, R) を挿入;
19 end
```

第4章　MPQ-treeの構成

本章では MPQ-tree を構成する手順について述べる．本章において，葉は子を持たない P ノードとして扱う．3 章のアルゴリズムを用いて，入力の区間表現は左端点優先の長さ順序のコンパクトな区間表現 $LLorderCI$ に変換されているとする．本章で述べるアルゴリズムは，$LLorderCI$ を入力として，$LLorderCI$ を 2 回スイープし MPQ-tree を構成する．

図 4.1 に概略を示す．図の矢印 1 は 1 回目のスイープを表し，矢印 2 は 2 回目のスイープを表す．$LLorderCI$ を入力として，1 回目のスイープはそれぞれの区間が属するノードとその種類 (P ノードか Q ノードか) を表す配列 $LaminarA$ を作成する．2 回目のスイープは，作成した $LaminarA$ と入力の $LLorderCI$ を用いて，MPQ-tree を構成する．

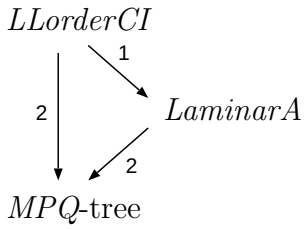


図 4.1: MPQ-tree 構成の流れ

1 回目のスイープをすることによって，区間表現の各区間が MPQ-tree を構成したときに，P ノードか Q ノードのどちらになるのか，また同じノードにラベル付けされる区間はどれかがわかる．図 4.2 の (a) を入力の区間表現とし，1 回目のスイープで (b) のノードを作成する．2 回目のスイープではそれらの情報を元に，それぞれの P ノードや Q ノードの親子関係を築き，MPQ-tree の構造を構成する．図 4.2 の (c) は (a),(b) の情報を元にして，親子関係を築いた MPQ-tree である．

4.1 節で 1 回目のスイープアルゴリズムについて記述し，4.2 節で 2 回目のスイープアルゴリズムについて記述する．

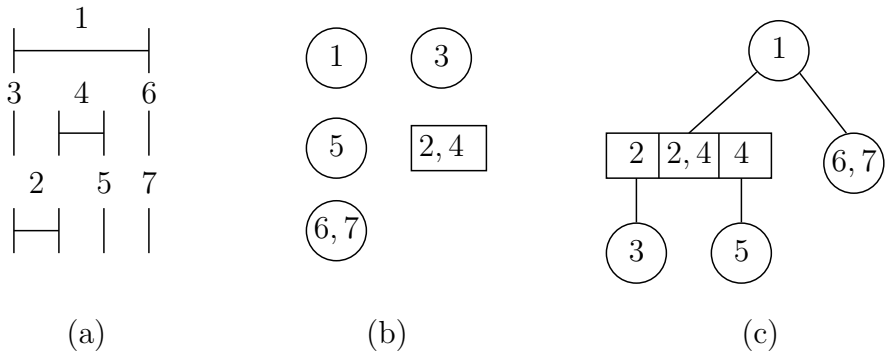


図 4.2: 2 回スイープの流れ

4.1 1回目のスイープアルゴリズム

1 回目のスイープアルゴリズムをアルゴリズム 3 に示す．このアルゴリズムの入力は左端点優先の長さ順序のコンパクトな区間表現 *LLorderCI* であり，出力はそれぞれの区間が属するノードとその種類を表す配列 $LaminarA[1..n]$ である．n は入力の区間の数である．*LaminarA* の添字は区間の番号を表し，要素 $LaminarA[x]$ は次の 2 つのデータを持つ．

1. ノードの種類 $kind \in \{P, Q\}$
2. ノードの番号 num

LLorderCI の端点を左からスイープすると次のことが言える．

補題 4.1. 左端点優先の長さ順序のコンパクトな区間表現 *LLorderCI* に対し，左から端点をスイープすると，左の端点 $1, 2, 3, \ldots, n$ の順に読み込まれる．

1 回目のスイープアルゴリズムであるアルゴリズム 3 は同じ Q ノードに分類される区間を見つけるために，部分交差している区間をスタック S と 2 種類のフラグ $Flag[1..n]$,*flag* を用いて探す．

まず，スタック S に対する操作について説明する．アルゴリズム 3 は端点を順に読み込んでいき，以下の操作を行う．

1. 区間 x の左の端点を読み込んだとき，S に区間の番号を格納する ($PUSH(S, x)$)．
2. 右の端点を読み込んだとき，S の要素を 1 つ取り出す ($POP(S)$)．

アルゴリズム 3 の入力は左端点優先の長さ順序のコンパクトな区間表現である．よって，補題 4.1 から左の端点は $1, 2, \ldots, n$ の順で読み込まれる．また，右の端点は長さの短い順に読み込まれる．e の区間 $num[e]$ とスタックから取り出された値 x が一致していないとき，区間 $num(e)$ と x は部分交差していることがわかり，2 つの区間は Q ノード Q_q に分類される．

次に，フラグ $Flag[1..n]$ について説明する．フラグ $Flag[1..n]$ は配列 $\Pi[1..n]$ を用いて管理される．$\Pi[1..n]$ は読み込まれた右の端点 e とスタックから取り出した値 x が異なるとき，$\Pi[num(e)]$ に x が代入される．$\Pi[1..n]$ に対して，次を定義しておく．

$\Pi' \subset \Pi$ が $\forall i \in \Pi'$ に対して $\Pi[i] \in \Pi'$ のとき，Π' は閉じているという．

$Flag[size(S)]$ には，配列 Π' が閉じていないとき，作成した Q ノードの番号 q が格納される．$Flag[size(S)]$ が 0 でないとき，読み込まれた端点の区間 $num[e]$ は Q ノード Q_q に分類される．こうすることにより，部分交差している区間をすべて見つけることができ，部分交差をしている区間を同一の Q ノード Q_q に分類することができる．

最後に，フラグ *flag* について説明する．右の端点を読み込んで配列が閉じたとする．このとき，Q ノード Q_q に分類され，かつ部分交差する区間の端点はすべて読み込まれたことになる．そこで *flag* に Q ノードの番号 q を格納する．次に読み込まれた右の端点を e とすると，区間 $num(e)+1$ が Q_q に分類されるとき，区間 $num(e)$ は Q_q に分類される．

定理 4.2. アルゴリズム 3 は Q ノードに分類される区間を洩れなく見つけ，作成した Q ノードに番号を 1 つだけ与える．

証明. 以上の議論から，スタック S の操作とフラグ $Flag[1..n]$ を用いることにより，部分交差しているすべての区間を捜し出すことができる．ここで，Q_q に分類される区間の集合を $I_{ii \in V'}$ とする．次に読み込まれた端点の区間 I を $I = \bigcup_{i \in V'} I_i$ とする．フラグ *flag* を用いることで，区間 I が Q_q に分類されることがわかる．

それぞれのフラグに Q ノードの番号を格納することにより，同一の Q ノードに分類される区間がわかる． □

Q ノードに分類されなかった区間は P ノードになる．P ノードに分類される区間の右端点 e が読み込まれたとき，P ノード作成処理であるアルゴリズム 4 を実行する．そして，区間 $num(e)+1$ が P_p に分類されるとき，区間 $num(e)$ は P_p に分類される．そうでないとき，新しく P ノードを作成する．

定理 4.3. アルゴリズム 3 は $O(n)$ 時間かかり，$O(n)$ 領域必要とする．

証明. アルゴリズム 3 は 1 回スイープを行っているため，読み込む端点の数は $O(n)$ である．スタックへの操作 *PUSH* と *POP* は $O(1)$ 時間処理が可能である．また，その他の行は簡単な比較演算や代入文を行っているので，$O(1)$ で実行される．よって，アルゴリズム全体の計算時間は $O(n)$ である．

スタック S は左の端点の読み込まれた数だけ容量があれば十分なので，高々 $O(n)$ 領域である．配列 $LaminarA$ は区間の数だけ要素を用意すればよい．よって，アルゴリズム全体の領域は $O(n)$ である． □

アルゴリズム 3 における部分交差している区間の見つけ方を例を用いて説明する．図 4.3(a) の左端点優先の長さ順序のコンパクトな区間表現 *LLorderCI* が入力されたとする．このとき，端点は次の順序で読み込まれる．

Algorithm 3: 1 回目のスイープアルゴリズム

Input: 左端点優先の長さ順序のコンパクトな区間表現：$LLorderCI$

Output: 配列 $LaminarA$

```
1  スタック S の初期化;
2  配列 LaminarA, Flag[1..n], N[1..n] の初期化;
3  flag, p, q の初期化;
4  for i = 1 to maxCI do
5      e ← head(LLorderCI[i]);
6      while e ≠ NIL do
7          if (kind(e)) = L then
8              PUSH(S, num(e));
9          else
10             x ← POP(S);
11             if x = num(e) then
12                 if Flag[size(S) + 1] = 0 then
13                     if Laminar[num(e) + 1] = (Q, flag) then
14                         LaminarA[num(e)] ← (Q, flag);
15                     else
16                         P ノード作成処理 (アルゴリズム 4);
17                         flag ← 0;
18                     end
19                 else
20                     LaminarA[num(e)] ← (Q, Flag[size(S) + 1]);
21                     Flag[size(S)] = Flag[size(S) + 1];
22                 end
23             else                                          /* x ≠ num(e) */
24                 Π[num(e)] ← x;
25                 if Π[x] = NIL then                /* 配列が閉じていないとき */
26                     Q ノード作成処理 (アルゴリズム 5);
27                     Flag[size(S)] ← q;
28                 else                                  /* 配列が閉じたとき */
29                     Flag[size(S)] ← 0;
30                     frag ← num(LaminarA[x]);
31                 end
32             end
33             Flag[size(S) + 1] ← 0;
34         end
35     end
36 end
37 return LaminarA;
```

Algorithm 4: Pノード処理

if $LaminarA[num(e)+1] \neq (P,p)$ **then**
  $p \leftarrow p+1$;
end
$LaminarA[num(e)] \leftarrow (P,p)$;

Algorithm 5: Qノード処理

if $LaminarA[num(e)] = NIL$ **then**
  $q \leftarrow q+1$;
  $LaminarA[num(e)] \leftarrow (Q,q)$;
  $LaminarA[x] \leftarrow (Q,q)$;
else
  $LaminarA[x] \leftarrow LaminarA[num(e)]$;
end

$$(x,L) \rightarrow (y,L) \rightarrow (z,L) \rightarrow (x,R) \rightarrow (y,R) \rightarrow (z,R)$$

このとき，スタックに行われる処理は次の手順で行われる．

$$PUSH(S,x) \rightarrow PUSH(S,y) \rightarrow PUSH(S,z)$$
$$\rightarrow POP(S) \rightarrow POP(S) \rightarrow POP(S)$$

3つ目の操作 $PUSH(S,z)$ を行ったときの様子を図 4.3(1) に示す．

スタックへの4つ目の操作 $POP(S)$ の様子を図 4.3(2) に示す．この POP 操作により，スタックから取り出した番号は z となる．また，読み込んだ端点の区間の番号は x である．そのため，スタックから取り出した値と読み込んだ端点の区間の番号が異なる．このとき，x と z は部分交差していることがわかり，同じ Q ノードに分類される．ここで，読み込んだ端点の区間 x と z の値が異なるので，配列 $\Pi[x]$ に z を代入する．このとき，$\Pi[z]$ には何も値が入っていないので，配列は閉じていない．配列が閉じていないとき，フラグを立てておく．

次の POP 操作の様子を図 4.3(3) に示す．次のスタックへの操作 $POP(S)$ を行うと，スタックから取り出した番号は y となる．また，そのとき読み込んだ端点の区間の番号は y であり，スタックから取り出した値と読み込んだ端点の区間の番号は一致する．このとき，フラグが立っているため，y は x や z と同じ Q ノードに分類される．

最後の $POP(S)$ の様子を図 4.3(4) に示す．このとき，スタックから取り出した番号は x となる．また，読み込んだ端点の区間の番号は z である．読み込んだ端点の区間 z とスタックから取り出した番号 x が異なるので，$\Pi[z]$ に x を代入する．図 4.3(4') から $\Pi[x]$ はすでに値が入っているので，順列は閉じている．このように配列が閉じるまで，読み込ま

れた端点の区間は同じ Q ノードに分類される．そして，配列が閉じたらフラグを降ろし，次の端点を読み込む．

以上のように行い，Q ノードに分類される区間を見つける．

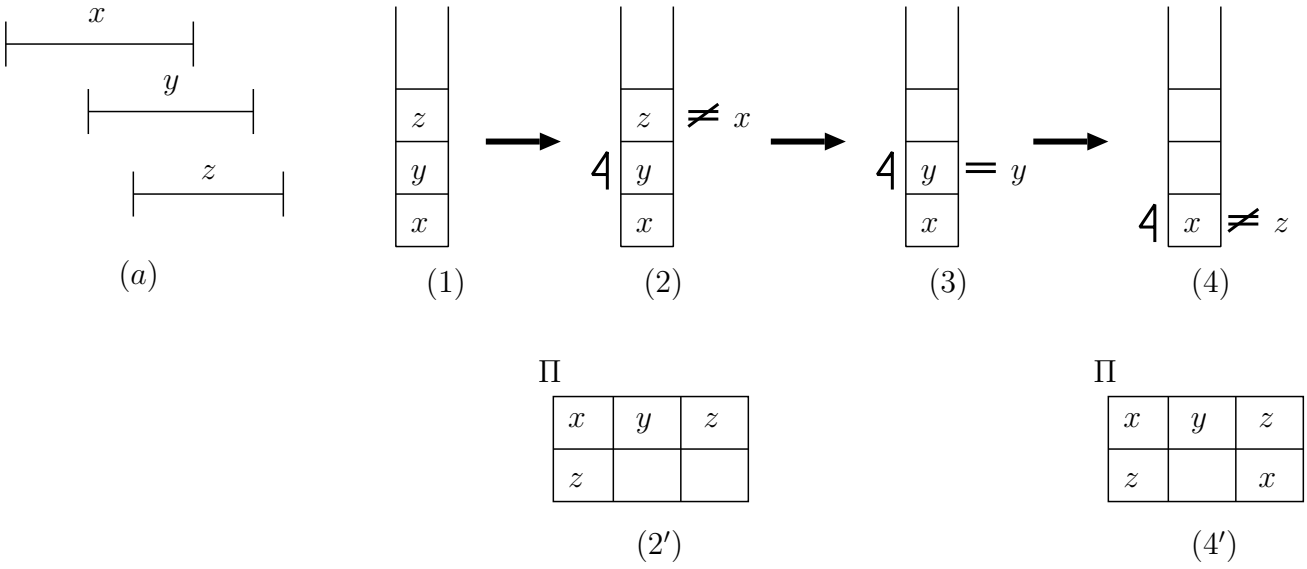
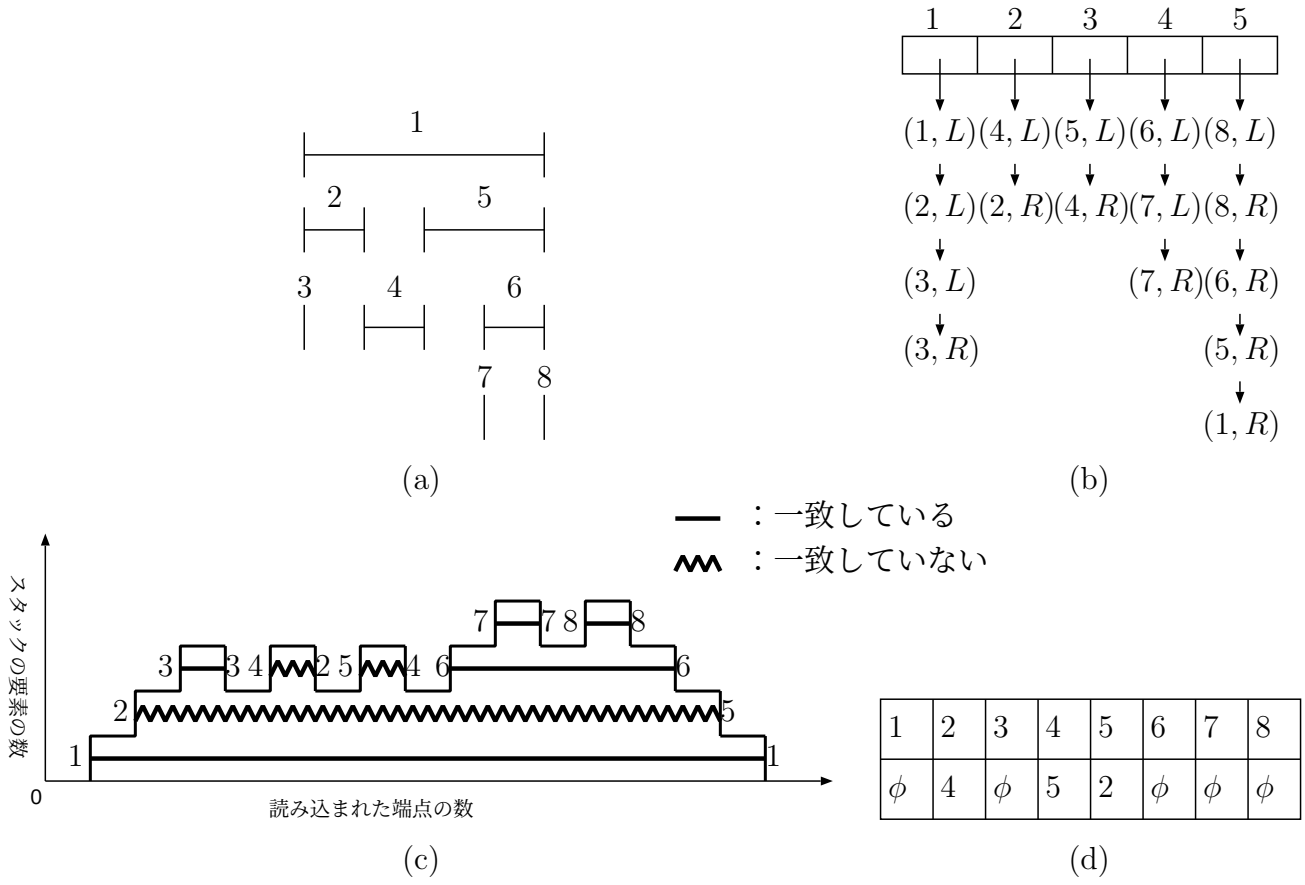


図 4.3: スタックと配列の動作

図 4.4 の (a) は左端点優先の長さ順序のコンパクトな区間表現を表し，(b) はそのデータ構造を表す．(c) は (b) をスイープしたときのスタックの様子である．左端点が読み込まれたとき，スタックにその区間の番号が入力され，要素が 1 つ増える．スタックに格納される区間の番号を左に書く．右端点が読み込まれたとき，スタックから 1 つデータを取り出すので, 要素が 1 つ減る．読み込まれた端点の番号を右に書く．また，スタックから取り出される番号は左に記された番号になる．そして，右の数と左の数を比較する．比較した値が等しいところは直線でつなぎ，等しくないところは波線でつなぐ．

比較を行った結果，図 4.4 の区間の番号 1,3,6,7,8 は一致している．それに対し，図の 2,4,5 は一致しない．

図の (c) から 2 と 4 が部分交差していることがわかり，2,4 は Q ノード $\hat{Q}$ に分類される．また，4 と 5 が部分交差していることもわかるので，4,5 は $\hat{Q}$ に分類される．このことから，2，4，5 は $\hat{Q}$ に分類される．さらに，$l(1) = l(2)$ かつ $r(1) = r(5)$ から，1 も $\hat{Q}$ に分類される．その他の部分交差が発見されなかった区間は P ノードとなる．



1	2	3	4	5	6	7	8
ϕ	4	ϕ	5	2	ϕ	ϕ	ϕ

図 4.4: スタックの様子

4.2　2回目のスイープアルゴリズム

2回目のスイープアルゴリズムをアルゴリズム6に示す．アルゴリズム6は左端点優先の長さ順序のコンパクトな区間表現 *LLorderCI* と1回目のスイープアルゴリズムで求めた配列 *LaminarA* から *MPQ*-tree を構成する．

MPQ-tree のすべてのノード N は次のデータを持つ．

1. ノードの種類 *kind*
2. ノードの番号 *num*
3. 親へのポインタ *parent*

P ノード $\hat{P}$ は上に加えて次のデータを持つ．

1. 区間の集合 *set*
2. 子へのポインタのリスト *Child*
   特に，リストの先頭は $\hat{P}$ の末子で $head(Child(\hat{P}))$ と表す．

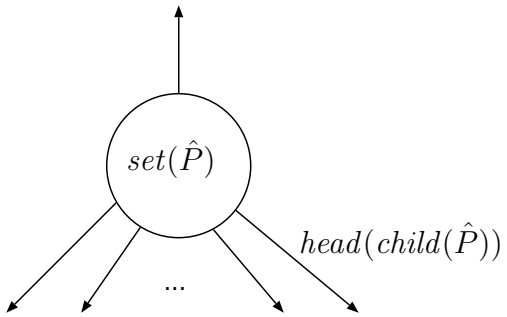


図 4.5: P ノード $\hat{P}$ のデータ構造

P ノードを図示すると図 4.5 になる．

Q ノード $\hat{Q}$ のデータ構造について説明する．$\hat{Q}$ はセクション $S_1, S_2, \ldots, S_k$ で構成されており，それぞれのセクションが子を持つ．Q ノードの末子とつながっているセクションを末子セクションという．図で表すと，末子セクションは一番右に存在するセクションである．$\hat{Q}$ の末子セクションを S_k とする．$\hat{Q}$ は S_k へのポインタ $section(\hat{Q})$ を持つ．

セクション S_i は次のデータを持つ．

1. Q ノードへのポインタ *node*
2. 子へのポインタ *child*
3. 両隣りのセクションへのポインタ
   S_i の右隣りのセクション： *right*
   S_i の左隣りのセクション： *left*
4. 端点の集合 *set*

Q ノード $\hat{Q}$ のデータ構造は図 4.6 になる．

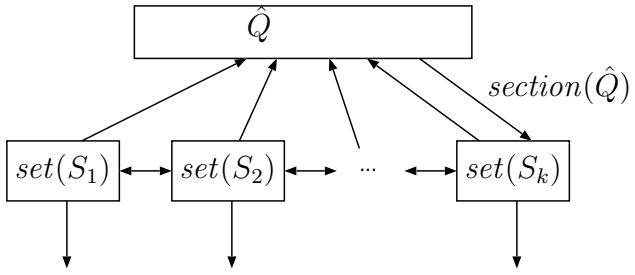


図 4.6: Q ノードのデータ構造

アルゴリズム 6 はスイープを 1 回行う．スイープを行うと，任意の区間は右の端点より先に左の端点が読み込まれる．スイープの途中で，左の端点を読み込み，右の端点を読み込んでいない区間を不完全な区間という．それに対し，左と右の両方の端点が読み込まれ

た区間のことを完全な区間という．不完全な区間が存在するノードのことを不完全なノードといい，完全な区間しか存在しないノードを完全なノードという．

アルゴリズム 6 はスタック S を用いて親子関係を築く．S にはノードの種類と番号の組を格納する．x がラベル付けされるノードを N_x とする．N_x は $LaminarA[x]$ からわかる．各ノードに対しフラグを与え，N_x がスタックに格納されているとき，フラグが立っているとし，N_x がスタックに格納されていないとき，フラグが立っていないとする．

ここで，左の端点 (x, L) を読み込んだとする．このとき，N_x のフラグが立っているかを調べる．フラグが立っているとき，x は S の先頭のノードにラベル付けされる．フラグが立っていないとき，N_x は S の先頭のノードの子となる．そして，N_x を S に格納し，フラグを立たせる．フラグが立っていないときの様子を図 4.7 に示す．

次に，右の端点 (x, R) が読み込まれたとする．ここで，ノード N_x が完全となるとき，$POP(S)$ を行い，スタックの先頭の要素である N_x を取り出す．ノード N_x がまだ完全ではないとき，何もしない．

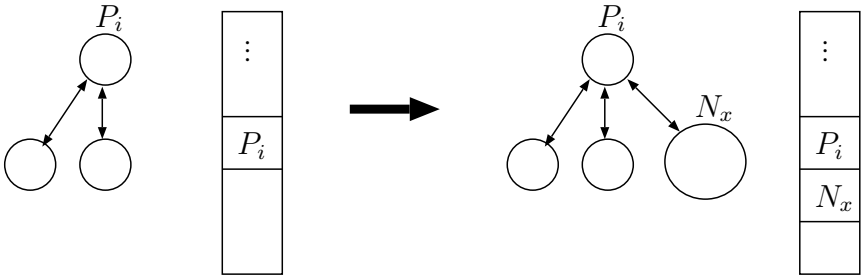


図 4.7: フラグが立っていないときの例

定理 4.4. アルゴリズム 6 はアルゴリズム 3 で得られた各ノードの親子関係を正しく築くことができる．

証明. 各ノードは対応するコンパクトな区間表現において，包含関係になっている．よって，以上の議論より親子関係を築くことができる． □

定理 4.5. アルゴリズム 6 は $O(n)$ 時間と $O(n)$ 領域かかる．

証明. このアルゴリズムはスイープを 1 回行っている．8 行目，21 行目，23 行目ではメモリ $O(1)$ のメモリ領域の確保と，定数回のポインタの書き換えを行うので，これらの処理は $O(1)$ 時間かかる．他の行は代入やスタック操作となっているので，$O(1)$ 時間かかる．よって，全体の計算時間は $O(n)$ 時間である．

入力の $LLorderCI$ と $LaminarA$ の領域は $O(n)$ であり，出力される MPQ-tree も $O(n)$ 領域である．また，スタックの必要な領域は高々$O(n)$ である．よって，このアルゴリズムに必要な計算領域は $O(n)$ である． □

Algorithm 6: 2回目のスイープアルゴリズム

Input: 左端点優先の長さ順序のコンパクトな区間表現 *LLorderCI*, 1回目のスイープで求めた配列 *LaminarA*

Output: 対応する *MPQ*-tree

```
1  for i = 1 to maxCI do
2      e ← head(LLorderCI[i]);
3      while e ≠ NIL do
4          N_x ← LaminarA[num(e)];
5          Write(N_x, e);
6          if kind(e) = L then
7              if N_xのフラグ ≠ 0 then
8                  スタックの先頭の子に N_x を作成;
9                  PUSH(S, N_x);
10             end
11             N_x のフラグを 1 増加;
12         else                                  /* 右の端点を読み込んだとき */
13             N_x のフラグを 1 減少;
14             if N_xのフラグ = 0 then
15                 POP(S);
16             end
17         end
18     end
19     if kind(N_x) = Q then                     /* セクション処理 */
20         if set(Section(N_x)) が空 then
21             セクション Section(N_x) と left(Section(N_x)) を結合;
22         end
23         N_x に新しくセクションを作成;
24     end
25 end
```

第5章　おわりに

区間表現を入力として与えたとき，その区間表現に対応する MPQ-tree を構成する場合分けの少ない, かつ高速なアルゴリズムを提案した.

今後の課題は本質的に異なるコンパクトな区間表現の列挙が考えられる．また本論文で提案したアルゴリズムを実装して，実際に高速であることを実験的に示す.

参考文献

[1] M. C. Golumbic. *Algorithmic Graph Theory and Perfect Graphs*. ANNALS OF DISCRETE MATHEMATICS 57. ELSEVIER, 2004.

[2] N. Korte and R. H. Möhring. An incremental linear-time algorithm for recognizing interval graphs. *SIAM Journal on Computing*, 18(1):68–81, 1989.

[3] R. Uehara. Canonical Data Structure for Interval Probe Graphs. *Lecture Notes in Computer Science Vol. 3341*, pp. 859–870, 2004.

[4] R. Diestel. *Graph Theory*. Graduate Texts in Mathematics 173. Springer, 1997.

[5] A. Gibbons. *Algorithmic Graph Theory*. Cambridge University Press, 1985.