5. データ構造

1～4 章までに学んだことに基づき、計算機でデータを処理する時に、処理の柔軟性を実現しつつ計算機資源を有効に利用するために作られた、代表的なデータ構造について説明する。プログラムは、データ構造とコンピュータの処理手順（6 章アルゴリズム）の二つから構成され、データ構造をきちんと定義することではじめて、アルゴリズムを明確に構築することができる。

5.1 基本的なデータ構造

ここでは、基本的なデータ構造として、配列、リスト、キュー、スタックについて説明する。

注：コンピュータでは 0 から数字を数え始めることが多い。これは、符号無しの 2 進数で可能な限り多くの数を表すためには、0 から数え始める方が都合が良いからである。

(1) 配列

数値のみを並べたデータ構造。各要素へのアクセスは、配列の添え字を用いて行う。

1 次元配列

変数名（ここでは A）に添え字を付して N 個のデータを 1 次元で表す方法。

例えば、A[0], A[1],..., A[N-2], A[N-1]。

2 次元配列

配列に添え字が 2 つあり、それぞれの要素数を N,M とすると N×M 個の要素を持つ配列。

A[0, 0], A[0, 1], ..., A[0, M-1]

.....

.....

A[N-1, 0], A[N-1, 1], ..., A[N-1, M-1]

添字の数を増やせば 3 次元、4 次元...とした多次元配列で記述できる。

なお、主記憶上では最終的に 1 次元のデータとなるため、プログラミング言語によっては、多次元配列が 1 次元の配列と等価な形でアクセスできることが多い。

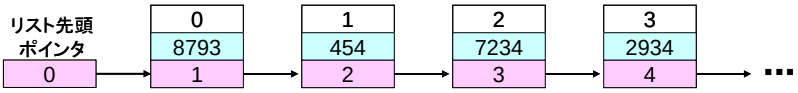
例：C 言語では、N×M の 2 次元配列 A は、以下の 1 次元配列としてもアクセスできる。

A[0, 0], A[0, 1], ..., A[0, M-1], A[1, 0], A[1,1], ..., A[N-1, 0], ..., A[N-1, M-1]

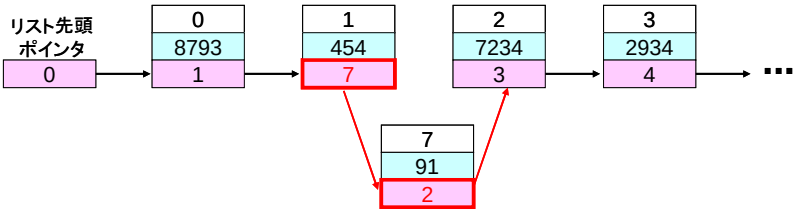
(2) リスト

リストは、データのつながりに意味をもたせたデータ構造。データの中に次につながるデータのアドレス(格納位置)を示す**ポインタ**という項目を持つことが特徴。ポインタの値を変更することでデータの追加や削除に柔軟に対応できる。

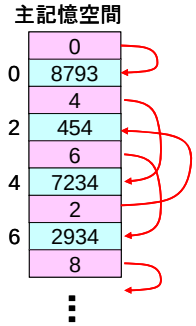
リストの論理的な構造は以下のような図を用いて表されることが多い。四角の中の数値は、上からアドレス、データ、ポインタを示している。



上記のポインタを変更することでデータを任意の順に並び変えることが可能である。また、下記のように、データを挿入／削除する部分の前のポインタを変更することにより、データの挿入／削除も可能である。以下の例では、アドレス 1 にあるデータ 454 とアドレス 2 にあるデータ 7234 の間に、アドレス 7 に追加したデータ 91 を挿入している。この場合、データ 454 に付随するポインタを挿入するデータのアドレスである 7 に書き換え、挿入するデータ 91 に付随するポインタに次のデータのアドレスである 2 を書き込むことで実現する。このようにリストにおけるデータの順序は、配列における物理的位置ではなく、論理的に決めることができる。



リストの物理的な構造は、配列にデータとポインタを組で持たせた形となる。通常、アドレスは配列の添え字や実アドレスをもちいる。1 つのエントリは組み合わせで表現するため、構造体を使える言語では、構造体を使って表すことが多い。



リスト派生としては、以下のような形がある。

単方向（線形リスト）： 次のデータのポイントのみをもつ。

双方向リスト: 前のデータのポインタと次のデータのポインタを持つ

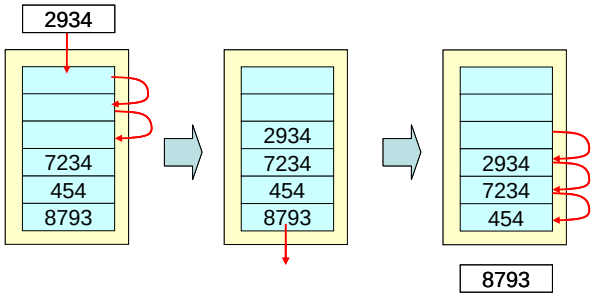
環状リスト: 最後のデータの次のポインタを先頭データとする。

(3) キューとスタック

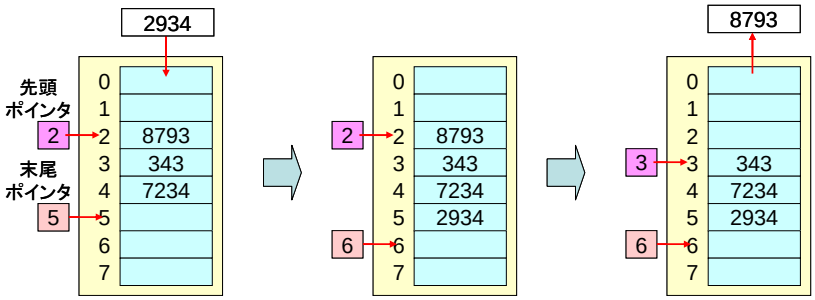
配列のようなデータの並びがあり、データの格納と取り出しの順番に着目したデータ構造として、**キュー**(待ち行列)と**スタック**がある。

キュー

配列の一方の端でデータが挿入され、他方の端でデータが取り出される。先入れ先出し(FIFO; First In First Out,最初にいれたものを最初に出す)。また、キューを待ち行列とも呼び、銀行の ATM の行列のように先に並んだ人から順番に処理を実行するときに利用する。



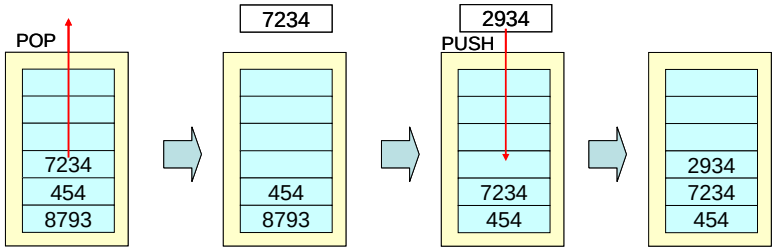
実際にキューを実現する場合、いちいちデータを移動させるのは無駄なので、配列と先頭ポインタ、末尾ポインタを使うことになる。この場合、配列の最後のエントリまでデータが埋まった場合、次のデータは配列の先頭に入れることになる。そのため、ポインタは更新するごとに、配列のエントリ数で割って剰余をポインタとする(配列数が 2 の n 乗の場合、剰余の演算は n 個の 1 のビット列と AND で済ませることができるので、よく 2 の n 乗のエントリ数が使われる)。



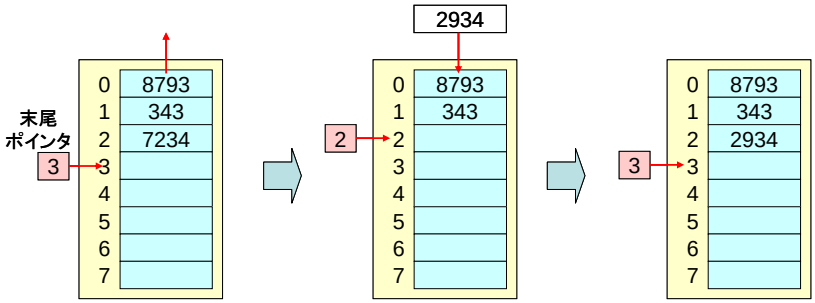
スタック

配列の一方の端だけで、データの挿入と取り出しを行うデータ構造。

最も新しいデータがデータの取り出しの対象となる。これを後入れ先出し(LIFO:Fast In First Out,最後に入れたものを最初に出す)とよぶ。スタックでは配列の先頭を**底**、末尾を**頂上**という。また、スタックに新しいデータを格納する操作を **PUSH**、取り出す操作を **POP** という。

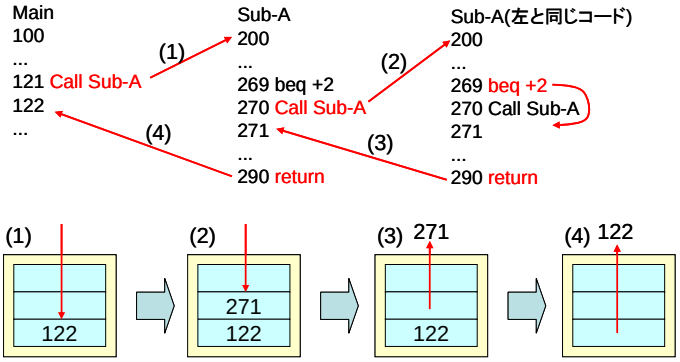


キューと同様、スタックも主記憶上の配列として実現することができる。キューとは異なり、ポインタは 1 つになるし、ポインタ更新時に剰余を取る必要もない(ただし、値が配列の範囲を超えていないか確認する必要はある)。



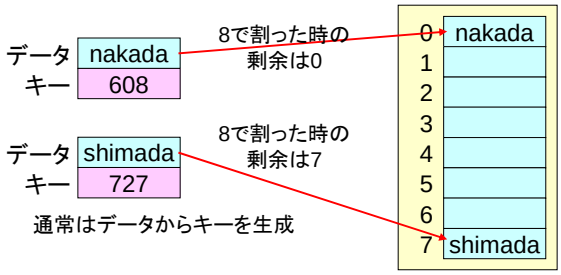
スタックの利用例： サブルーチンの戻りアドレスの保持

サブルーチン(Sub)に飛ぶごとに戻ってくるアドレスを PUSH し、プログラム実行で PUSH されたアドレスに達したらこのアドレスを POP することにより戻りアドレスを保持できる。このようなデータ構造により、同一のサブルーチンの再帰的な呼び出しを複数回行っても、スタックに保存されている(同じ)戻りアドレスの回数だけ戻れば、正しい動作を実現できる。



5.2 ハッシュ

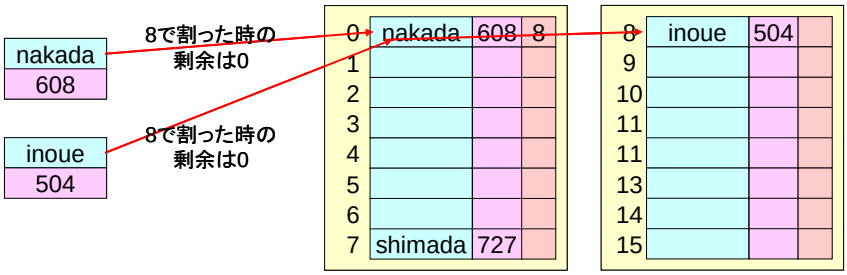
探索したいデータのキーの値から直接データの格納する添え字やアドレスを決める方法によってデータを表に保存する方法を**ハッシュ**と呼ぶ。キーの値からデータの格納場所を求めるために**ハッシュ関数**を用いる。ここでハッシュ関数とは、一定演算によって、キーの値からハッシュ値(データの格納アドレスや配列の添え字と対応)を求める関数をいう。なお、キーの生成については、一般的には、データの値から生成することが多く、データの値から直接ハッシュ値を生成することもある。このデータを格納する表は、**ハッシュ表**と呼ぶ。



ハッシュ法の問題は、異なるキーの値から同じハッシュ値になる場合があり、このことを衝突またはシノニムという。衝突の発生した場合に対応する格納法に、**チェイン法**と**オープンアドレス法**がある。

チェイン法

同じハッシュ値を持つデータをリストを用いてつなぐ方法。この方法では、データを格納する表にポインタとキーの値を格納する部分を追加する。ハッシュ値を用いてデータをハッシュ表に登録しようとした時に、エントリに既にデータがある(衝突した)場合は、ポインタの内容を空きエントリを指すように更新するとともに、空きエントリに後から登録するデータを保存する。

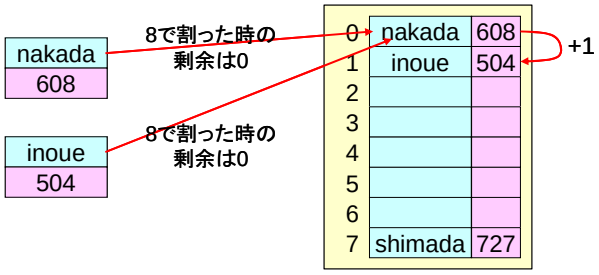


上記の登録の仕方では、後から登録するデータへのアクセスの方がポインタをたどる必要がある。そのため、最初のアクセスでポインタを読み出す構造にして、そのポインタを後から登録したデータの方から順番に指すように設定することで、後から登録されたデータの方が読み出しやすくなるようにした実装もある。

オープンアドレス法

衝突が発生したときに、再度格納するアドレスを計算し直す(再ハッシュ）方法。一般的に、再ハッシュの方法は「求めたハッシュ値+1」とし、ハッシュ表に開き領域が見つかるまで順次探索する。

オープンアドレス法は、最大でハッシュ表のサイズまでしかデータを登録できないが、ポインタ領域を確保する必要がないため、主記憶領域は節約される。



5.3 木構造

木とは、データの階層的な関係を表現するために用いられるデータ構造である。木構造を表すため、以下のような用語を定義する。

節(node)： 木構造においてデータが格納される部分

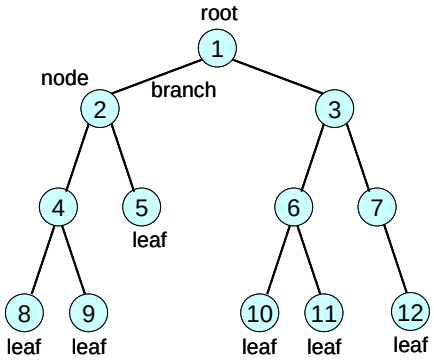
枝(branch)： 節を結ぶ線

根(root)： 最上位の節

葉(leaf)： 自分の下に節を持たない節

部分木： 木構造のある節以下を、新たな木構造として考えたもの。

深さ： 根から葉までの距離



情報技術においては、**2 分木**と呼ばれる、1 つの節に 0-2 個の子の節を持つ木構造をよく用いる。以下では、2 分木の特徴を利用した情報操作技術について説明する。。

(1) 2 分探索木

以下のルールに従ってデータを配置することにより探索効率を上げる方法。

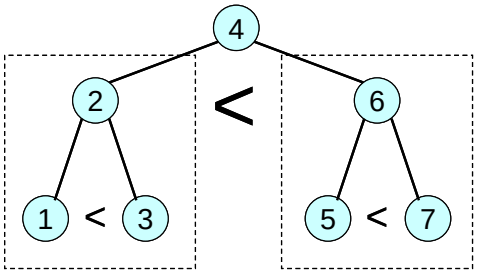
任意の根とその部分木について、

ルール 1： 左部分木に含まれる値は根の値より小さい。

ルール 2： 右部分木に含まれる要素は根の値よりも大きい。

探索したい値が節の値より大きいとき右、小さいとき左の枝をたどれば目的の値に到達する。探したい値が 5 とすると、4 < 5 なので右に行く、続いて 5 < 6 であるから左へ行く

と 5 に到達する。



(2) 2 分木の走査

2 分木の走査法には、

1.**前順**：節点、左部分木、右部分木の順に走査する。

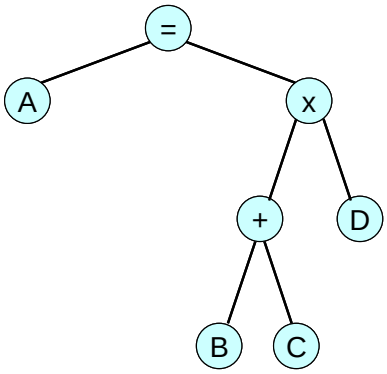
2.**間順**：左部分木、節点、右部分木の順に走査する。

3.**後順**：左部分木、右部分木、節点の順に走査する。

の 3 つがある。

(3) 数式と 2 分木構造

節に四則演算を記入し、葉に変数を記入することにより、四則演算を木で表現できる。A=(B+C)×D を 2 分木で書くと以下の図のようになる。



2 分木の走査をすると

1.**前順**：=A×＋BCD　　(ポーランド記法)

2.**間順**：A=(B+C)×D

3.**後順**：ABC+D×=　　(逆ポーランド記法)

となる。例えば、後順は、逆ポーランド記法に変換されるため、その結果をもとにコンパイラにより機械語に変換することできる。このように、2 分木のデータ構造は、数式表現法に応用されている。

第 5 章の問題集

問題 5.1　表は、配列を用いた連結セルによるリストの内部表現であり、リスト[東京、品川、名古屋、新大阪]を表している。このリストを[東京、新横浜、名古屋、新大阪]に変化させる操作はどれか。ここで A(i,j)は表の第 i 行第 j 列を表す。例えば A(3,1)=名古屋、A(3,2)=4 である。また→は代入を表す。

列

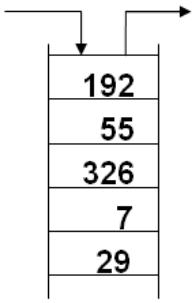
行	1	2
1	東京	2
2	品川	3
3	名古屋	4
4	新大阪	0
5	新横浜	

	第1の操作	第2の操作
ア	5→A(1,2)	A(A(1,2),2)→A(5,2)
イ	5→A(1,2)	A(A(2,2),2)→A(5,2)
ウ	A(A(1,2),2)→A(5,2)	5→A(1,2)
エ	A(A(2,2),2)→A(5,2)	5→A(1,2)

（平 18 秋午前問 13）

問題 5.2　PUSH 命令 でスタックにデータをいれ、POP 命令でスタックからデータを取り出す。動作中のプログラムにおいて、ある状態から次の順で 10 個の命令を実行したとき、スタックの中のデータは図のようになった。一番目の PUSH 命令でスタックに入れたデータはどれか。

PUSH→PUSH→POP→PUSH→PUSH→PUSH→PUSH→POP→POP→PUSH



（平 18 秋午前問 13）

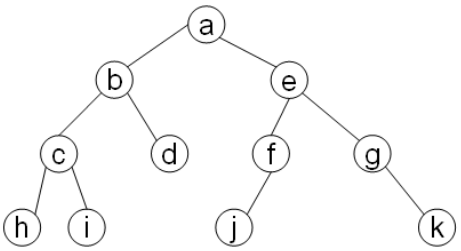
問題 5.3　２分木の走査の方法には、その順序について次の三つがある。

1.前順：節点、左側分木、右側分木の順に走査する。

2.間順：左側分木、節点、右側分木の順に走査する。

3.後順：左側分木、右側分木、節点の順に走査する。

図に示す２分木に対して全順に走査を行い、節の値を出力した結果はどれか。



ア　abchidefjgk イ　abechidfjgk ウ hcibdajfegk エ hicdbjkgea　　（平 15 秋午前問 12）

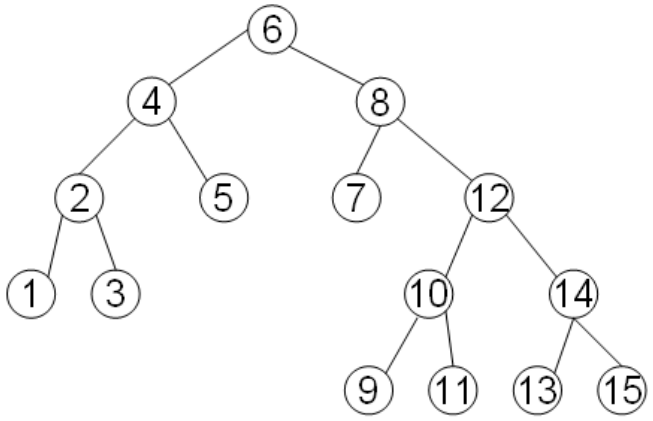
問題 5.4 アルファベット３文字で構成されているキーがある。次の式によってハッシュ値 h を決めるとき、キー「SEP」と衝突するのはどれか。ここで、a mod b は、a を b で割った余りを表す。

h = （キーの各アルファベットの順位の総和） mod 27

アルファベット	順位	アルファベット	順位	アルファベット	順位
A	1	K	11	U	21
B	2	L	12	V	22
C	3	M	13	W	23
D	4	N	14	X	24
E	5	O	15	Y	25
F	6	P	16	Z	26
G	7	Q	17		
H	8	R	18		
I	9	S	19		
J	10	T	20		

ア　APR イ　FEB　ウ JAN エ NOV　　（平 15 春午前問 14）

問題 5.5 次の２分探索木の要素 12 を削除したとき、その位置に別の要素を移動するだけで２分探索木を再編成するには、削除された要素の位置にどの要素を移動すればよいか。



ア　9 イ　10　ウ 13 エ 14　　（平 16 秋午前問 12）

問題 5.6 データ構造に関する記述のうち、適切なものはどれか。

ア　２分木は、データ間の関係を階層的に表現する木構造の一種であり、すべての節が二つの子を持つデータ構造である。

イ　スタックは、最初に格納したデータを最初に取り出す先入先出しのデータ構造である

ウ　線形リストは、データ部と次のデータの格納先を指すポインタ部から構成されるデータ構造である。

エ　配列は、ポインタの付け替えだけでデータの挿入・削除ができるデータ構造である。

（平 17 秋午前問 13）