

再帰について

再帰関数を簡単に言うと、「自分で自分を呼び出す関数」のことです。

メリットとしては、解こうとする問題が再帰的な性質を備えていると素直にプログラムを説くことが出来ます。処理を終わらせるために、処理条件の記述が必要。今回の課題4だと $n=1$ の時に、結果を `print` 出力して終了する。となっている。

再帰関数の使用例として、

1. データ処理（複数のデータをソートしたり、繰り返し処理を行う場合）
2. 複雑な問題の解決（今回のハノイの塔のような問題）

などが挙げられる。

今回の課題では、簡単な挿話の計算を行った後に再帰関数で簡単に解くことが出来るハノイの塔を解く。

1. スタックとキューについての違い

スタックは、データを上から積んでいき、上から取り出すというデータ配列のこと。

スタックを例えると、積み木のようなものである。積み木は、上から積むことが出来るが、崩さないようにしたから取り出すのは至難の業である。そのため例えるなら、積み木のようなデータ配列である。

キューは、データを上から積んでいき、下から取り出すというデータ配列のこと。

キューを例えると、チケットの販売所である。チケットの販売所は、後ろから人が増えていき、前から人が減っていく。そのため例えるなら、チケットの販売所である。

1.1. 今回取り上げた探索手法について

線形探索

線形探索とは、最も単純なアルゴリズムである。流れとしては、以下の通りである。

1. 配列などに格納されたデータの中から、まず戦闘の要素を探しているデータと比較する。
2. 一致しなければ2番目の要素と比較する。
3. 1,2を繰り返し、途中でデータを発見したら、その位置を返す

最も単純なアルゴリズムと称される通り、行っていることは、1,2,3の通り

シンプルである。

利点としては、先頭から比較するため、事前にデータ列の整列などを行う必要が無いことが挙げられる。

しかし、データが見つからなかった場合、比較回数は要素数に正比例して増大するという欠点がある。

二分探索

プログラムの流れは以下の通りである。

1. データを昇順または、降順に並べ替える
2. そして、データが全体の前半分にあるか後ろ半分にあるかを判定する
3. 半分になったデータ群の中央と探しているデータを比較する
4. 2,3 を繰り返し、中央のデータが探しているデータに一致するか、データが見つからなかったときに探索を終了する

プログラムは、この流れで実行される。

利点としては、線形探索に比べて計算量を減らすことが出来るというものがある。

しかし、データを並べ替える手間が発生する欠点がある。

エラーが発生したときに、プログラムを調べる方法として、二分探索の考え方を使うとうまくいくことがある。例えば、ある関数の上半分を削除して結果を確認、次に下半分を削除して結果を確認するというやり方がある。

1.2. その他の探索手法について

木構造での探索

① 幅優先探索

探索を開始する独钴炉から近いものをリストアップし、さらにそれぞれを細かく調べていく方法を幅優先探索という。

本を読むときに目次を観て全体を把握し、さらにそれぞれ章について概要を読み、さらに内容を読むという用に徐々に深く読んでいくようなイメージ

深さ優先探索に比べてメモリ使用量を抑えることが出来る。

② 深さ優先探索

目的の方に進めるだけ進んで、進めなくなったら戻る方法を深さ優先探索と言う。

再帰処理が使われることが多く、オセロや将棋などの対戦型のゲーム探索を行う場合には必須の探索方法である。

最短でたどり着けるものを一つだけ見つける場合には、身つかった時点で、処理を終了できるため、幅優先探索に比べて高速に動作する。

第 6 回

バブルソート

リストのとなったデータを比較して、大小の順序が違っているときは並べ替えていく方法です。データが移動していく様子を、水中で泡が浮かんでいく様子に例えて、バブルソートという名前が付けられている。

リストの先頭と次のデータから初めて、左の方が大着れば右と交換することを、1 つずつずらしながら繰り返す。リストの最後尾まで到達すると、1 回目の比較は終了だ。このとき、リストの最後尾にはデータの最大値が入り

2 回目が一番右端を除いて同様の比較を行うと、最後から 2 番目が決まる。これをくり貸すと、全てが並べ替え

られ、ソートは終了だ。(図 3.1 参照)

改良するとしたら

バブルソートで変更が発生しなかった場合に処理を打ち切り、高速化することを考える。処理中に好感が多かったかどうかを記録し、降雨感が起こらなかった場合はそれ以降の処理を行わないモノとする。



図 3.1 バブルソート

・クイックソート

クイックソートはリストから出来とうにデータを 1 つ選んで、これを基準として小さい要素と大木要素の分割し、それぞれのリストでまた同じような処理を繰り返してソートする方法だ。

一般的には「分割当時法」とも飛ばれる方法に分類されるソート方法で、小さい単位に分割して処理することを再帰的に繰り返す。これ以上分けられないようなサイズ まで分割できれば、それをまとめた結果を求める。

一般的には「分割統治法」とも呼ばれる方法に分類されるソート方法で、小さい単位に分割して処理することを再帰的に繰り返す。これ以上分けられないような細部まで分割できれば、それをまとめた結果を求める方法だ。

(図 3.2 参照)

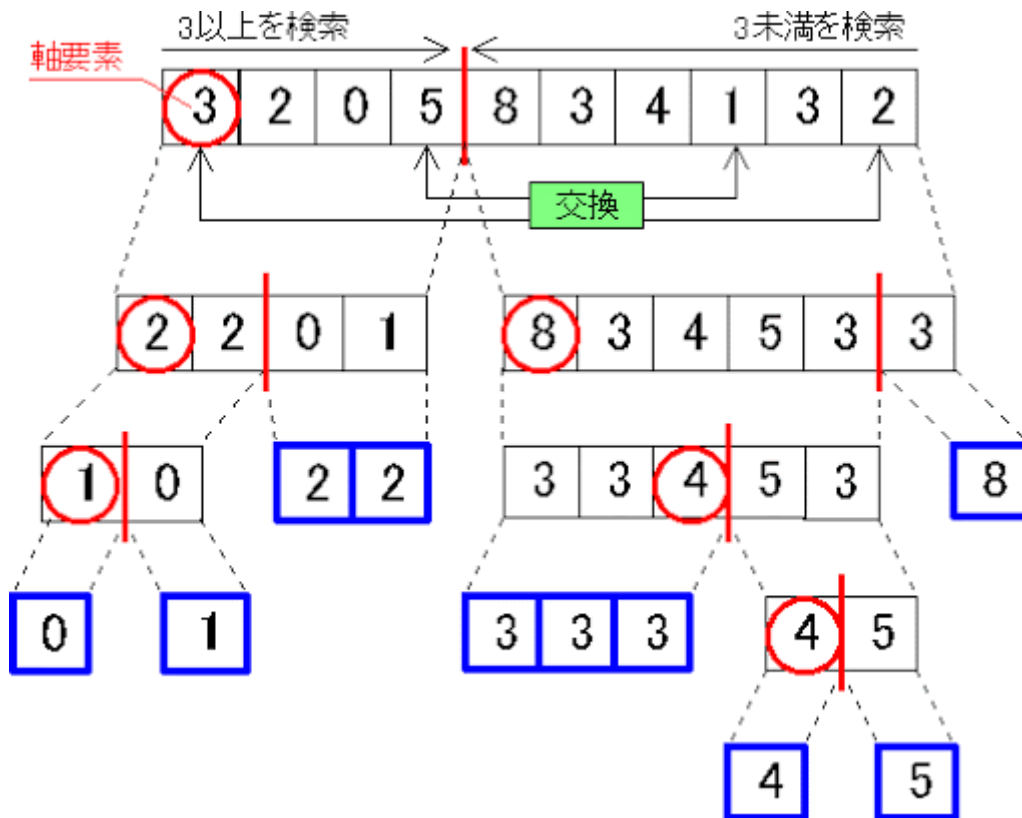


図 3.2 クイックソート

・マージソート

マージソートは、ソートしたいデータが入ったリストを2つに分割することを繰り返し、全てがバラバラ担った状態から、これらのリストを統合(マージ)する方法だ。この統合する際に、そのリスト内で値が小さい順に並ぶように実装することで、全体が1つのリストになったときには全ての値がソート済みになっている。(図 3.3 参照)

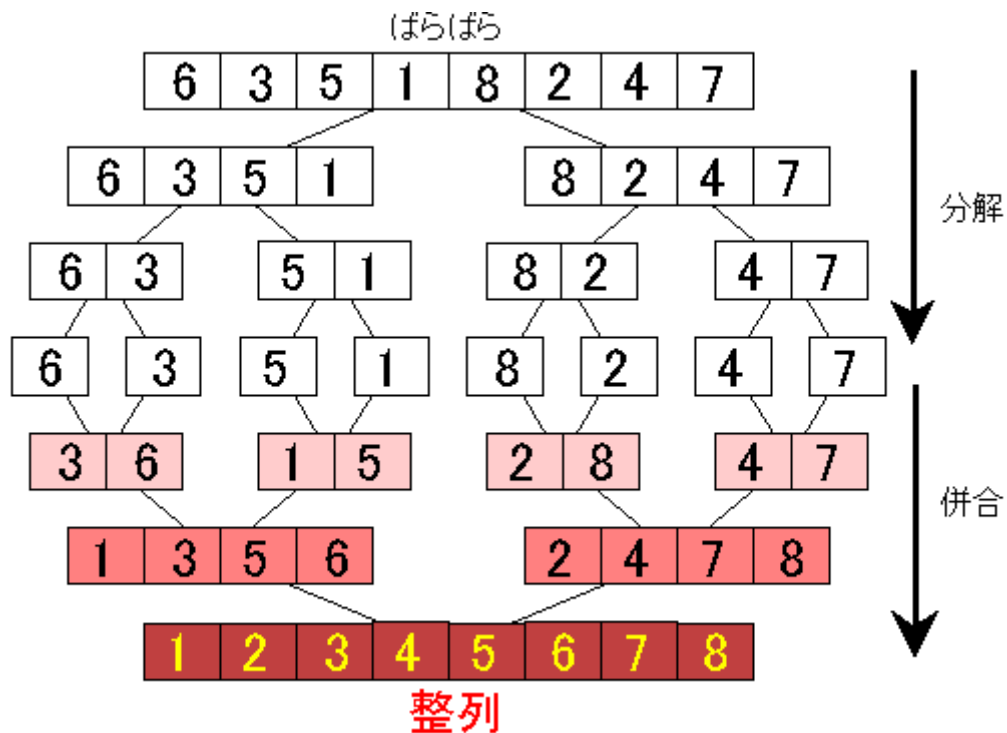


図 3.3 マージソート

1. 「連想配列」についてまとめよ。

連想配列とは、添え字に好きな名前を付けることが出来る配列のこと

一般的に考えられている配列は、最初が 0、次に 1、2 と続いて値が入っているのに対して、連想配列だと、値を好きな数字や文字に変えることが出来るというもの。例えば、配列の 1 つめに、「和歌山」、次の配列には、「愛媛」という形に値を入れていくというもの。

1. コンピュータで数値計算を行う際の以下の事項について調べよ。単なる語句の説明にとどまらず、具体例を挙げるとなおい。

1.1. 情報落ち

コンピュータで絶対の大きさが極端に異なる数字を足したり引いたりしたときに、小さい値の情報が無視されてしまう現象。

1.2. オーバーフロー

コンピュータ上で、数値の計算結果がその格納領域に収まる範囲を超えること。

1.3. アンダーフロー

浮動小数点演算処理について、計算結果の指数部が小さくなりすぎ、使用している記述方式では数値が表現できなくなることである。

1.4. 変換誤差

ほとんどの 10 進数の「小数」は、2 進数に正しく変換できない。そのため、変換するときに誤差が出来てしまう。

1.5. 桁落ち

桁落ちとは、非常に近い大きさの小数同士で減算を行ったときに、有効数字が減る現象の事。
コンピュータでは浮動小数点数の数値計算において生じる

1.1. モンテカルロ法

数値計算手法の一つで、乱数を用いた試行を繰り返す事により近似解を求める手法。

ある事象をモデル化した数式や関数があるとき、その定義域に含まれる値をランダムにたくさん生成して実際に計算をおこない、得られた結果を統計的に処理することで推定値を得ることが出来る。
数式を解析的に解くのが困難あるいは不可能な場合でも数値的に近似解を求めることが出来る。

1.2. ユークリッドの互除法

ユークリッドの互除法とは、二つの自然数の最大公約数を求めるアルゴリズムの一つ。二数の最大公約数は両者とも割り切る事が出来る自然数（公約数）のうち最大の者だが、これは 大きい方を小さい方で割ったあまりと小さい方との最大公約数に等しいという性質があり、これを利用して効率的に算出する。

1.3. エラトステネスのふるい

エラトステネスのふるいの計算量は、 $O(n \log \log n)$ だ。この証明と一般的なエラトステネスのふるいの流れを記述する。

1. 2 から n までの整数を並べる。
2. 生き残っている数の中で一番小さく、まだ p として使われていないものを新たに p とおき、 p 以外の p の倍数を全て消す。
3. 2 の操作を繰り返していき、 p が \sqrt{n} を超えたら終了
4. 3 の状態でふるいにかけて残っているものが素数

2 でふるいおとすのに $\frac{n}{2}$ 回、3 でふるい落とすのに $\frac{n}{3}$ 回、5 でふるい落とすのに $\frac{n}{5}$ 回、... の演算を行うので計算量は、

$$\sum_{p < \sqrt{n}} \frac{n}{p} = \frac{n}{2} + \frac{n}{3} + \frac{n}{5} + \dots$$

(ただし、和は \sqrt{n} 以下の素数全体に関して取る)

ここで n が十分大きいとき \sqrt{n} までの素数の逆数和はおおよそ $\log \log \sqrt{n} = \log \log n + \log \frac{1}{2}$ くらいなので、計算量は $O(n \log \log n)$ となる。

ハッシュとは、 $O(1)$ という計算量で探索を行える優れたアルゴリズム。なぜ、ここまで計算量が少なく探索を行うことが可能なのか？

これが原理だ。データを登録するときに、そのデータ自身の値を使って何らかの計算を行って、格納位置 (通常、データ小僧としては配列を使うので、その添え字のこと) を決定します。

登録するデータが整数と仮定する。そこで、登録するデータの値を n とした時、 n を 100 で割ったあまりを格納位置にするというルールにする。そのときの格納位置は、 $537 \% 100$ という計算により、37 に決定される。

探索するときも、同じ計算をすれば、即座に格納位置を取り出せるというアルゴリズム。

1. 木構造について、まとめよ。

枝分かれして広がり、一つの親に対して複数の子を持つ構造のこと

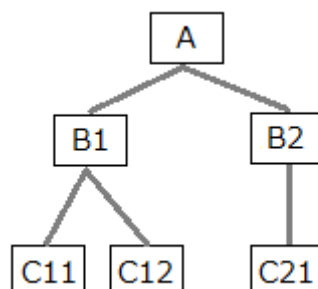


図 1.1 木構造の模式図

Aを起点として、下に行くほど広がっている。この図1のような構造が木構造だ。図1に押して資格で表現した要素を「ノード」といい、要素同士繋ぐ線の部分を「エッジ」と言う。

根っこの方（上）が「親」であり、広がっていく先が、「子」となる。

図 1.1 内の A と B1 の関係を言葉で表すと、A は B1 にとっての親であり、B1 は A にとっての子である。この言葉を図にすると図 1.2 のようになる。

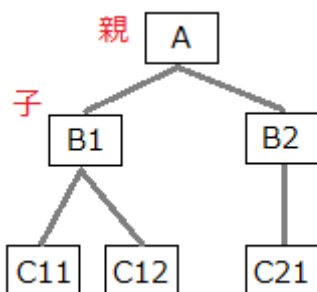


図 1.2 親子の関係

2. BM(Boyer-Moore)法について調査し、その内容をレポートにまとめよ。

BM 法は、かつて「最速」と言われた文字列検索アルゴリズム
どのようなものを説明していきます。

図 1.1 の上のテキストから Pattern を順番に見ていく。

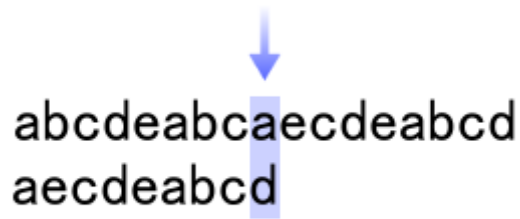
Text: abcdeabcaecdeabcd
Pattern: aecdeabcd

図 1.1 文字列について

文字列の検索なので、Text を前から順番に見ていく

説明上、この「見ている位置」を「ポイント」と呼ぶことにする(通常のプログラムで使うポイントとは違う。)

この BM 法が他のアルゴリズムと違うのはポイントの最初の位置(図 1.2 参照)だ。パターンの末尾にポイントを置き、末尾と一致しているところをまず探す。他の一般的な探索アルゴリズムは最初の文字を見る。



abcdeabcaecdeabcd
aecdeabcd

図 1.2 BM 法のポインタの位置

今回は”a”と”d”なので一致しない。末尾が一致しない場合、その位置にパターンの中の最も右寄りにある同じ文字合わせる(同じ文字がなかったらパターンの長さ分だけ一気に移動させる。)そうすると、次に一致する可能性がある場所まで行きに移動できる。このように移動させると、最大で文字の長さ、(図 1.3 なら 9 文字分)移動させることができる。そのため効率的だ。



abcdeabcaecdeabcd
aecdeabcd

図 1.3 BM 法のポインタ位置の移動

BM 法は、文字をスキップして、少ない計算量で実装することができるアルゴリズムで