1　アルゴリズム演習 5

1.1　アルゴリズムの計算量による分類

　アルゴリズムは、ほぼ出つくしたようで、計算量によって以下の様に整理、分類されています。

L : 計算量がサイズ n の一次式で表されるアルゴリズム
問題例:n 人からなる住所録から、ある特定の人の住所を探し出す問題

Q : 計算量がサイズ n の二次式で表されるアルゴリズム
問題例:n 個の数値データを小さい順に並べ変える問題

T : 計算量がサイズ n の三次式で表されるアルゴリズム
問題例:n 元連立方程式を解く問題

P : 計算量がサイズ n の多項式で表されるアルゴリズム
問題例:以上の例すべての問題

EXP : 計算量がサイズ n の指数とする関数で表されるアルゴリズム
問題例:n 個の入力を持ち、指定された入出力関係を満たす、最も簡単な計算機回路の設計問題

S : 計算量は問わず、ともかく「有限時間内に解ける」アルゴリズム

この領域の外に「一般的にはどうしても解けない問題」、「その問題を解くアルゴリズムが存在しない」領域があります。

1.2　ソート

　ソートとは、与えられたデータを小さい順または大きい順に並べ換えることです。
ソート技法には、いろいろありますが、ここでは基本選択法と呼ばれているアルゴリズムを用います。

n 個のデータの中から最小項を探し、それを第 1 項と交換します。これで第 1 項は確定したので、次に第 2 項以後の $n-1$ 個のデータの中から最小項を探し、それを第 2 項と交換します。これを $n-1$ 回くり返せば n 個のデータが小さい順に並べ換えられます。

1.3 kihon.c

```
/*
   kihon.c
  基本選択法によるソート
  C 言語 137 頁
*/

#include <stdio.h>

#define N 8

int main()
{
  int a[] = {100, 55, 65, 33, 211, 66, 31, 90};
  int j;
  int k;
  int min;
  int s;
  int dumy;

  for(k = 0; k < N-1; k++){
    min = a[k];
    s   = k;
    for(j = k + 1; j < N; j++){
      if(a[j] < min){
        min = a[j];
        s   = j;
      }
    }
    dumy = a[k];
    a[k] = a[s];
    a[s] = dumy;
  }
```

```
  for(k = 0; k < N; k++){
    printf("%5d", a[k]);
  }

  return 0;
}
```

2 AI による大規模データ処理入門

2.1 系統的探索　情報コース「問題解決の数理」から

情報コース「問題解決の数理」の 146 頁に系統的探索の説明があり、縦型探索 (深さ優先探索)、横型探索 (幅優先探索) のアルゴリズムを下記の様に説明されています。

縦型探索 (深さ優先探索) のアルゴリズム

1. 探索を開始する点を OpenList に入れる。ClosedList は空にする。
2. OpenList が空なら探索は失敗 (解がない) して終了。
3. OpenList の先頭の点 n を取り除き、リスト ClosedList に入れる。
4. n が目標点であるなら探索は成功して終了。
5. n から 1 ステップで移れる点のうち、OpenList にも ClosedList にも含まれていないものがあれば、それらすべてを OpenList の先頭に入れて 2 へ、なければ、そのまま 2 へ。

深さ優先探索は、OpenList に記憶する点の数が幅優先探索に比較するとはるかに少なく、計算機のメモリをあまり消費しないため、大きな状態空間の探索に適している。ただし、目標点が探索の開始点から近くにあっても、探索量が多くなる可能性がある。無限に深く点が続く場合には目標点に到達できない。

横型探索 (幅優先探索) のアルゴリズム

1. 探索を開始する点を OpenList に入れる。ClosedList は空にする。
2. OpenList が空なら探索は失敗 (解がない) して終了。

3. OpenList の先頭の点 n を取り除き、リスト ClosedList に入れる。
4. n が目標点であるなら探索は成功して終了。
5. n から 1 ステップで移れる点のうち、OpenList にも ClosedList にも含まれていないものがあれば、それらすべてを OpenList の末尾に入れて 2 へ、なければ、そのまま 2 へ。

幅優先探索は、開始点の近くから探索するので、開始点の近くにある目標点を効率的に探索するのに適している。また、点から枝分かれが有限であれば、深さが無限の状態空間でも目標点に到達できる。ただし、点からの枝分かれが多い場合、OpenList に記憶する点の数が多くなり、計算機のメモリを多く消費する。

2.2 横型探索と縦型探索の実行例

「AI による大規模データ処理入門」の例では、縦型探索が少ない探索手順で目標点に到達しています。

横型探索の実行例

オープンリスト　1[0],
クローズドリスト

オープンリスト　2[1],3[1],5[1],
クローズドリスト　1[0],

オープンリスト　3[1],5[1],4[2],
クローズドリスト　1[0],2[1],

オープンリスト　5[1],4[2],6[3],7[3],
クローズドリスト　1[0],2[1],3[1],

オープンリスト　4[2],6[3],7[3],8[5],
クローズドリスト　1[0],2[1],3[1],5[1],

オープンリスト　6[3],7[3],8[5],
クローズドリスト　1[0],2[1],3[1],5[1],4[2],

オープンリスト　7[3],8[5],
クローズドリスト　1[0],2[1],3[1],5[1],4[2],6[3],

オープンリスト　8[5],
クロースドリスト　1[0],2[1],3[1],5[1],4[2],6[3],7[3],

オープンリスト　999[8],
クロースドリスト　1[0],2[1],3[1],5[1],4[2],6[3],7[3],8[5],

ゴールを見つけました
999[8]<-8[5]<-5[1]<-1[0]

縦型探索の実行例

オープンリスト　1[0],
クロースドリスト

オープンリスト　5[1],3[1],2[1],
クロースドリスト　1[0],

オープンリスト　8[5],4[5],3[1],2[1],
クロースドリスト　1[0],5[1],

オープンリスト　999[8],7[8],4[5],3[1],2[1],
クロースドリスト　1[0],5[1],8[5],

ゴールを見つけました
999[8]<-8[5]<-5[1]<-1[0]