順列生成の開発Ⅰ

2013.05.09

問題１～４

省略。

番外１

途中省略。

【順列から階乗進数への変換】

順列：(a_n , a_{n-1} , … a_1)　　ただし、$a_k = 0 \sim k$ とする

階乗進数：(b_n , b_{n-1} , … b_2) とする。

1)　順列 a_n , a_{n-1} , … a_1 は、ｎ次の超立方体の格子点と考える事ができる。従って、

最上位の桁は一致するので、$b_n = a_n$ とする。

（残りの $a_{n-1} \sim a_1$ のバリエーションは、(n-1)次超立方体の格子点の数　ｎ！個に留意）

2)　$a_{n-1} \sim a_1$ の中で、$a_n < a_k$ となるものは、$a_k = a_k - 1$ とする。

更新後の $a_{n-1} \sim a_1$ は、(n-1)次超立方体の格子点と考えることができる。

3) 従って、最上位の桁は一致するので、 $b_{n-1} = a_{n-1}$ となる。

4)　以下、2)、3)と同様の処理を繰り返し、b_2 まで確定すれば終了とする。

例　３１２０の階乗進数の求め方

1)により、　３１２０　　３ｘｘ

2)により、　３１２０　（変更なし）

3)により、　３１２０　　３１ｘ

2)により、　３１１０　（２ → １に変更）

3)により、　３１１０　　３１１

4) 求める階乗進数は　３１１である。

【階乗進数から順列への変換】

テキストの方法以外に、上記の逆変換の手順でも可能となる。(本質的には同じ)

順列：(a_n , a_{n-1} , … a_1)　　ただし、$a_k = 0 \sim k$ とする。

階乗進数：(b_n , b_{n-1} , … b_2) とする。

0) 最下位の $a_1 = 0$ とする。

1) $a_2 = b_2$ とする。

2) $a_{2-1} \sim a_1$ の中で、$a_2 \leq a_k$ となるものは、$a_k = a_k + 1$ とする。

3) $a_3 = b_3$ とする。

4) 以下、2)、3)と同様の処理を繰り返し、a_n まで確定すれば終了とする。

例　３１１の順列への変換

0)　より、　３１１０

1)　より、　下位１が決まる。　３１１０

2)　より、　該当なし。　３１１０

3)　より　次の桁が決まる。　３１１０

2)　より、　２ → １に変更　３１２０

3)　より、　最後の桁が３と決まる。

2)　より、　該当なし。　３１２０

4)　より、　順列は　３１２０となる。

【番外２】順列型完全ハッシュ関数とその活用事例

順列型完全ハッシュ関数および組み合わせ型完全ハッシュ関数は、パズルの解法では良く使われる関数である。以下では、順列型完全ハッシュ関数とその活用事例を紹介する。

注：**完全ハッシュ関数とは、ハッシュ値の衝突が発生しないようなハッシュ関数**を言う。

個々のハッシュ関数のソースは、別紙４参照。

１）順列型完全ハッシュ関数　ソース上は toNum 関数として定義

$0 \sim (n-1)$の n 個の数字を使ったすべての順列に $0 \sim (n+1)!-1$の番号を割り振る関数を考える。個々の順列と番号（ハッシュ値）との対応は、以下の手順で行う。

a. **順列から階乗進数に変換**する。この階乗進数を$(a_{n-1}, a_{n-2}, \cdots\ a_1)$と表記する。

b. 階乗進数から１０進数への変換を行う。

$$m = \sum_{k=1}^{n} a_k \cdot k!$$
$$= a_1 + 2 \cdot (a_2 + 3 \cdot (a_3 + \cdots + n \cdot (a_n)) \cdots)$$

２）順列型完全ハッシュ関数の逆関数　ソース上は toPos 関数として定義

ハッシュ関数の値域が小さい場合は変換テーブルを利用することも現実的にはありうるが、ここでは定義に基づいて逆関数を作る。

a. １０進数から階乗進数への変換を行う。この階乗進数を$(a_{n-1}, a_{n-2}, \cdots\ a_1)$と表記する。

b. **階乗進数から順列に変換**する。

【補足】

テキストで紹介されているアルゴリズムで上記のハッシュ関数 toNum2,とその逆関数 toPos2 を実装し、性能比較を実施した結果、**テキストのアルゴリズムが優れていることが判明**した。

N=11 での実測結果(単位秒)

toPos	**toPos2**	toNum	**toNum2**

13.56	**9.58**	8.27	**6.67**

3）８パズルの最長手数局面の探索　　プログラムとその実行結果は別紙５，６参照

8	2	5
7	*	1
6	4	3

＊は空きセル

——>

*	1	2
3	4	5
6	7	8

ゴール局面

８パズルは、15 パズルの縮小版で、パズル解法の説明では良く引用されるパズル問題である。

通常はある局面からゴール局面までの最短手数が問題とされるが、次の問題として、一番難しい局面(手数が最長となる局面と定義)を探す問題がある。今回は、上記のハッシュ関数の活用例として、この問題を解いてみる。

最長局面を探索する最もポピュラーな方法は幅優先探索である。ゴールの局面から１手前の局面を生成し、その局面から更に１手戻した局面へと順に続けていき、新たな局面が見つからなくなったとき、終了となる。この探索では、それぞれの局面を記憶しておき、新たな局面かどうかを判定しなければならない。この処理のため、局面のデータ(各ピースの順番)を完全ハッシュ関数(toNum 関数)を使って数値に変換し、同一局面かどうかのチェックをする。ここで、完全ハッシュ関数という特長が十分活かされている。

【局面の管理方法】

局面の表現：空きセルを０として、ボード上の数字の並びとする順列で表現する。

局面の総数：９！＝ 362880 個　　——>　十分メモリに常駐できる範囲である。(重要ファクタ)

局面を管理するテーブル：HT [362880] とする。HT にはゴールまでの最小手数を格納する。

初期値は　０(ゴール局面)　／　－１(その他の局面：まだ最小手数が未定という意味で)

（例）

・ゴール局面：012345678　のハッシュ値は０　——>　HT[0] = 0

・上記の問題の局面：825701643 のハッシュ値は、2555309

——>　HT[2555309]＝ｈ(ゴールまでの手数)

【探索アルゴリズムの概要】

0) HT を上記の初期状態に設定する。

1) hand=0 とする。(hand は現在の最小手数を表し、ループを制御する変数)

2) HT をスキャンして、hand に該当する局面のハッシュ値を抽出する。ハッシュ値よりその局面を復元する。(toPos 関数を使用してハッシュ値を順列に戻す)

3) 得られた局面から１手戻した局面を生成する。

4) 新しく生成された局面の順列から toNum 関数により、ハッシュ値ｈを求める。

5) HT[ｈ] =－1 ならば、その局面はまだ最小手数が未定の局面であるから、HT[ｈ]=hand+1 とする。ＨＴ[h] ≠－１ならば、その局面の最小手数はすでに確定しているので何もしない。

6) hand に該当する局面がなくなるまで、上記の 2)~5)を繰り返す。

7) 上記の処理で hand+１の局面が生成された場合は、hand++して、2)～6)の処理を繰り返す。

8) 新たな局面が存在しない場合、処理を終了する。最長手数は、最後に処理した hand である。また、その局面は、ＨＴ[h] = hand となるｈから求めることができる。

【探索結果】

最長手数は３１手

その局面は　以下の２つ

```
---- 1 ----      ---- 2 ----
8  *  6         8  7  6
5  4  7         *  4  1
2  3  1         2  5  3
```

以上