テーマ２：指数時間アルゴリズム（特に列挙問題を例に）

本テーマの目的

- 指数時間かかる場合も工夫は重要！
- 列挙アルゴリズムへの招待：これからのアルゴリズムの潮流の一つ！

2.1. 導入

一般に（時間）計算量がサイズパラメータ（たとえば n）に対して指数的に増加するようなアルゴリズムは使いづらい[1]．しかしだからといって諦めるわけにはいかない場合もある．また，だからといってアルゴリズムの工夫が無意味というわけでもない．まずは，その例として次の問題を考えよう．

部分和問題
入力： 自然数の列 $s_1, ..., s_n$ と目標値 W
（サイズパラメータは n とする）
仕事： $\sum_{i \in U} s_i = W$ となる $U \subseteq \{1, ..., n\}$ を求めよ．

この問題に対して $O(n2^n)$ 時間のアルゴリズムは簡単に考えることができる．一方，これは NP-困難な問題の一つとして有名な問題なので，多項式時間のアルゴリズムは期待できそうにない．では，それ以上の工夫は不可能だろうか？実は $O(n(1.414...)^n)$ 時間のアルゴリズムを考えることができる．はたしてそれは意味があるのだろうか？

2.2. 列挙問題入門[2]

時間計算量が指数時間かかるけれども「それを避けられない」問題の典型例として，今回のテーマでは列挙問題を取り上げてみたい．列挙問題とは，与えられた条件を満たすものを漏れなく重複なく出力する問題のこと．コンピュータの科学技術の研究へのパラダイムの一つとして，シミュレーション，データマイニングなどの次に来る潮流だと期待される．列挙問題は，必ずしも指数時間の計算量になるとは限らないが，列挙される物の個数自体が（サイズパラメータに対して）指数個になる場合が少なくない．この場合には（全体で）指数時間かかるのは避けられない．だからと言って効率よいアルゴリズムの工夫のしがいがない，というわけではまったくない．

[1] この「使いづらい」には二通りの意味がある．n が少しでも大きくなると，スパコンなどを駆使してもとてもまともな計算時間にはならない，という「使いづらさ」．また（そのためでもあるが）どのような場面で利用するかの調整（妥当な n に収めるような工夫）が難しい，という意味での「使いづらさ」である．

[2] 本日の講義の以下の説明では，計算成果観ＧＣＯＥなどが共催で行った「列挙学校 2008」における岡本吉央氏（北陸先端大）の講義ノートを参考にさせて頂きました．なお，今年の秋にも行う予定です（９月２８日〜３０日，国際湘南村）．是非，ご参加ください．

例として次の問題を考えてみよう．

<u>総和一定部分集合列挙問題</u>
入力： n, W
仕事： $\{1, ..., n\}$ の部分集合で和が W となるものを列挙せよ．

列挙アルゴリズムに必要な要件を挙げてみる．以下で，サイズパラメータを n，出力されるべき物（条件を満たす物）の総数を m とする．

正当性１：条件を満たす物をすべてを出力すること．
正当性２：同じ物は一度しか出力しないこと．
効率１：領域計算量は n に対して小さく（たとえば多項式以下）であること．
効率２：時間計算量は
<u>出力多項式時間</u>：n と m の多項式以下であること．
<u>慣らし多項式時間遅延</u>：n に対して多項式以下，m に対して線形以下であること．
<u>多項式時間遅延</u>：各出力の間隔時間が n の多項式以下であること．
※もちろん多項式でも次数が低いほどよい（たとえば定数，線形など）．

2.3. 列挙アルゴリズムの設計法の基本１：再帰による設計

アルゴリズムの設計法（もっと言えば，設計指針，設計パラダイム）の重要なものの一つが再帰による設計である．総和一定部分集合列挙問題に対して再帰的な考え方で設計したアルゴリズムを紹介しよう．

```
procedure sum_enum(X, m, W) {
  param.s  X: set of integers; m: max. element; W: target sum;
  ※ Initially X = ∅, m = n, and W ≤ m(m + 1)/2.
    if W = 0 then output X;
    else if W < m then
      sum_enum(X, m − 1, W);
    else if W > m(m − 1)/2 then
      sum_enum(X ∪ {m}, m − 1, W − m);
    else (i.e., m ≤ W ≤ m(m − 1)/2) {
      sum_enum(X ∪ {m}, m − 1, W − m);   sum_enum(X, m − 1, W); }
}
```

このアルゴリズムの出力遅延時間は $O(n)$ である．

お楽しみ問題（テーマ２：列挙アルゴリズム）

2.1. 上記のアルゴリズムの正当性を証明せよ（ヒント：入力引数 X, m, W が所望の条件を常に満たしていることを示すことが重要）．

2.4. 列挙アルゴリズムの設計法の基本２：Gray 符号による設計

もう一つ問題を考えよう．今度はさらに簡単な（少なくともそう見える）次の問題である．

偶サイズ部分集合列挙問題
入力： n
仕事： $U = \{1, ..., n\}$ の要素数が偶数個の部分集合を列挙せよ．

この問題が簡単に思えるのは，列挙される物の個数が 2^{n-1} とわかっている点である．多くの場合，列挙される物の個数をうまい式の形で表わすことはできないが，それに比べるとはるかに扱いやすい問題である．実際，つぎのような再帰のアルゴリズムを考えることができる．

```
procedure even_enum(X, m) {
  param.s  X: set of integers (initially ∅);   m: max. element (initially n);
    if m = 1 then
      if |X| is odd then output X ∪ {1} else output X;
    else {
      even_enum(X ∪ {m}, m − 1);   even_enum(X, m − 1); }
}
```

このアルゴリズムも出力遅延時間は $O(n)$ である．また，$\Omega(n)$ でもある．では，出力遅延時間は $O(1)$，つまり各部分集合を定数時間で次々と出力することはできるだろうか？基本は，もっと簡単な集合，すなわち $U' = \{1, ..., n-1\}$ の全部分集合の列挙である．というのも，U' の各部分集合 X' に対し，$|X'|$ が奇数ならば $X' \cup \{n\}$ を，偶数ならば X' を出力すれば目標の U の偶サイズ部分集合の列挙ができるからである．

そこで以下では，$U = \{1, ..., n\}$ の全部分集合の列挙を出力遅延時間は $O(1)$ で行うアルゴリズムを考える．その鍵となるのは，Gray 符号である．Gray 符号は二進列をハミング距離 1 で並べる方法である．たとえば，3 ビットの Gray 符号は

$$000,\ 001,\ 011,\ 010,\ 110,\ 111,\ 101,\ 100$$

となる．ここでも再帰の考え方が重要．k ビットの Gray 符号（に従う二進列の列）を $G(k)$，その二進列の列を逆に並べたものを $G(k)^{\mathrm{R}}$ とすると，

$$G(k+1) \ = \ 0G(k),\ 1G(k)^{\mathrm{R}}$$

と定義することができる．

ただし，後で都合のよいように，$G(k)$ の i 個目の二進列から $i+1$ 目の二進列を求める方法も考えておこう．ここは天下り的だがその規則を示しておく．

補題 2.1. b_i を $G(k)$ の i 番目の二進列とする．
(1) $b_1 = 0^k$，$b_{2^k} = 10^{k-1}$.
(2) i が奇数のときは，$b_{i+1} = b_i$ の最下位ビットをフリップしたもの．
(3) i が偶数のときは，$b_{i+1} = b_i$ の $\ell + 1$ ビット目をフリップしたもの．ただし，ℓ は b_i で最も右で 1 が立っているビット位置.

この Gray 符号の $G(n)$ を用い，各二進列に対し，その二進列で 1 の立っているビット位置（最下位ビットを 1，最上位ビットを n とする）に対応する要素からなる集合を U の部分集合と考えるのである．たとえば，$G(3)$ に対する部分集合は

$$\{\,\},\ \{1\},\ \{2,1\},\ \{2\},\ \{3,2\},\ \{3,2,1\},\ \{3,1\},\ \{3\}$$

である．また，X_i で $G(n)$ の i 番目の二進列に対応する部分集合を表わすことにする．すると上記の補題は次のようになる．

補題 2.2. $A\Delta\{a\}$ で，$A \cup \{a\}$（$a \notin A$ のとき），$A - \{a\}$（$a \in A$ のとき）を表わす．
(1) $X_1 = \emptyset$，$X_{2^n} = \{n\}$.
(2) i が奇数のときは，$X_{i+1} = X_i\Delta\{1\}$.
(3) i が偶数のときは，$X_{i+1} = X_i\Delta\{j+1\}$．ただし，$j = \min(X_i)$.

以上の考察から，U の部分集合の列挙アルゴリズムとして次のようなものを考えることができる．

```
Algorithm subset_enum(n)
    input n: integer;
      i = 1;
      X = ∅;  output X;
      while i ≤ 2^n {
        if i is odd then {
          X = X Δ {1};  output X;  i = i + 1; }
        else {
          j = min(X);
          X = X Δ {j + 1};  output X;  i = i + 1; }
      }
end.
```

お楽しみ問題続き（テーマ２：列挙アルゴリズム）

2.2. 補題 2.1 を証明せよ．

出力を別にすると，このアルゴリズムの while ループの一回の繰り返しは定数時間で実現できる．唯一工夫が必要な点は集合 X に対し min(X) を求める計算である．これも，X をビット列で表わしておけば[3]，そのビット列上でのビット演算を用いることで定数時間で計算可能である[4]．

最後に出力だが，これは X をすべて出力する（そうすると $O(n)$ 時間かかってしまう）のではなく，その差分（つまり加えられるか取り除かれる要素 1 つ）だけを表示するようにすれば定数時間で実現可能である．

[3]列挙問題では列挙のされる物の総数は 1 ワードか 2 ワードに収まると仮定することが多い．（あくまで私の解釈だが）そうでないと，列挙数そのものがまとも数でなくなるからだろう．ここでもそのように仮定する．

[4]ここでは省略する．前述の岡本氏からの情報によれば「ハッカーの楽しみ」という本にあるそうなので参照して欲しい．