文字列の辞書順列挙

●問題

入力として非負整数 m と n が与えられたときに，英文字 a〜z のうちの最初の m 文字を使って，(文字の重複を許して) 最大長 n の文字列を，(文字列の重複を許さず) 辞書順にすべて列挙したい.

(1) 文字列を辞書順に列挙するプログラムを作成せよ.
(2) プログラムの動作を解説せよ.
(3) 利用したデータ構造やアルゴリズムについて論じよ.
(4) 実行時間について実験・考察し，$m = 5, n = 15$ の場合の実行時間を推測せよ.
(5) 可能な別解について論じよ.

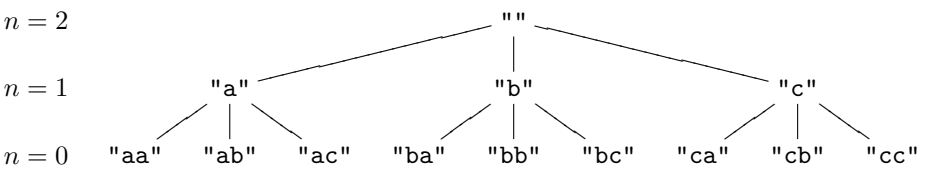
●解答

(1) プログラム (記述言語は Python) を以下に示す.

```
1: def strings(cs, n, s):
2:     "文字列を再帰的に辞書順で列挙"
3:     if n == 0: return [s]
4:     else:      return [s] + [x for c in cs for x in strings(cs, n-1, s+c)]
```

(2) 上記のプログラムを解説する．関数 strings の引き数は，使える文字の列 cs，最大の文字列長 n，直前に生成した文字列 s，の三つである．ただし，最初の呼び出しでは，s として空文字列 "" を渡す．関数 strings が返すのは，長さ n 以下の文字列を辞書順に並べてそれぞれを s に連結したリストである．$n = 0$ のときは，付加すべき文字列は空文字列だけなので，引き数の s だけからなるリストを返す (3 行目)．$n \geq 1$ のとき，長さ n 以下の文字列は，長さ $n-1$ 以下の文字列 s そのものか，あるいは，可能な文字の候補 cs から 1 文字 c を順に選んで，それぞれ s の後に付加したもの s+c のどちらかであり，それらを順に並べたリストを返す (4 行目).

たとえば $m = 3, n = 2$ の場合，次の呼び出し木に沿って関数 strings で文字列リストが生成される．第 1 引き数 cs は常に ['a','b','c'] で変わらない．第 2 引き数 n を次に示す木の各段の左に示し，第 3 引き数 s を木の節点に示す.



strings(cs, 2, "") の結果は，"", strings(cs, 1, "a") の各文字列，strings(cs, 1, "b") の各文字列，strings(cs, 1, "c") の各文字列からなる文字列リストである．これは，上に示した木を先行順に走査した結果に相当する．$n = 1$ で s が "a", "b", "c" の各場合の strings の結果は，各 s を根とする三つの部分木の先行順の走査の結果であり，それぞれ再帰呼び出しにより計算される.

(3) 利用したデータ構造やアルゴリズムについて述べる．文字列リストを求める上記のアルゴリズムでは，Python の組み込みデータ型である文字列やリストを使った．Python の文字列やリストを使うことで，走査や連結の処理を簡潔に書けるためである．文字列リストを列挙するアルゴリズムは,

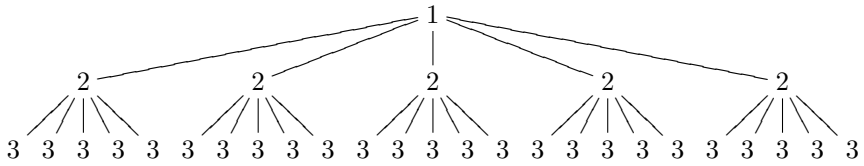
(1) の説明で述べたように，再帰関数を使って多分木の先行順の走査に相当する処理を実現している．これは，長さが 1 短い場合に対する解を使って必要な解を構成する，分割統治法のアルゴリズムである．

(4) 文字の種類数を $m = 5$ に固定したときの，実行時間について実験・考察する．まず，文字列の最大長 n を変えたときの処理時間を測定した．この結果を表 1 に示す．

表 1: 文字列の列挙の実行時間

n	実行時間 (秒)	n	実行時間 (秒)
0	0.000011	6	0.025195
1	0.000019	7	0.132174
2	0.000058	8	0.672387
3	0.000188	9	4.078343
4	0.000895	10	22.095753
5	0.004912	11	124.600594

$n \geq 4$ の場合に，n が 1 増えると実行時間がおよそ 5 倍になることが分かる．$n = 15$ の場合の実行時間を推定するため，処理時間をより詳細に見積もる．文字列リストの生成と表示には，(2) で示した木構造を走査して各文字の連結や表示をする時間がかかる．(2) の例では $m = 3$ の場合なので 3 分木を考えたが，今回は $m = 5$ なので 5 分木を考える．空文字列に対しても，連結の処理や改行の表示の時間がかかるので，完全 5 分木で深さ i ($i \geq 0$) の各節点で $i + 1$ (文字列の長さ+1) 単位の手間がかかると想定する．たとえば，$m = 5, n = 2$ の場合の木は次の通りである．



合計の処理時間を，この木の節の値の総和に比例すると見積もる．節の合計値 $\mathrm{num}(n)$ は，

$$\mathrm{num}(n) = \sum_{i=0}^{n} (i+1) \cdot m^i$$

で表せる．n が 0 から 15 の範囲の $\mathrm{num}(n)$ の値を表 2 に示す．

表 2: $\mathrm{num}(n)$ の値

n	$\mathrm{num}(n)$	n	$\mathrm{num}(n)$
0	1	8	4272461
1	11	9	23803711
2	86	10	131225586
3	586	11	717163086
4	3711	12	3890991211
5	22461	13	20980834961
6	131836	14	112533569336
7	756836	15	600814819336

文字列すべてを列挙するための推定実行時間 $t(n)$ は $c \cdot \mathrm{num}(n)$ で表せる．$n = 11$ のときの測定値

を使って，係数 c の値を求める．

$$c = \frac{t(n)}{\mathrm{num}(n)} = \frac{124.600594}{717163086} \approx 1.74 \times 10^{-7}$$

これを基に，実行時間の実測値と推定値を片対数グラフで表示したものを図 1 に示す．

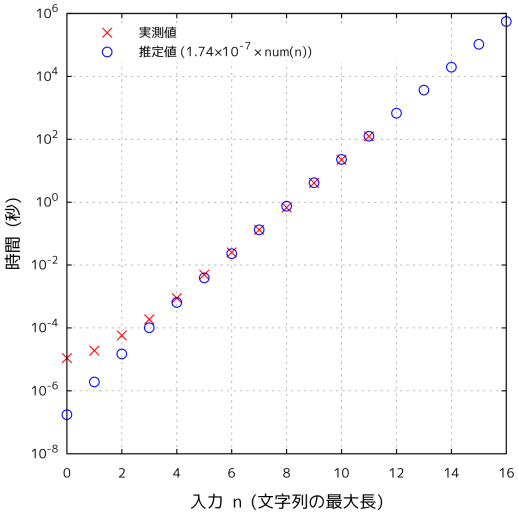


図 1: 実行時間の実測値と推定値

$n \geq 6$ の場合に適切な推定値が得られていることを確認できる．以上の結果から，$n = 15$ の場合の実行時間を推定する．

$$t(n) = c \cdot \mathrm{num}(n) \approx (1.74 \times 10^{-7}) \times 600814819336 \approx 1.05 \times 10^{5}(\text{秒})$$

つまり，$n = 15$ の場合，同じ実行環境での実行に 29 時間程度必要だと推定できる．

(5) 別解について考察する．(1) では，文字列のリストを作ってからリストを走査して表示する方法を示した．これ以外に，再帰手続き中に表示する方法もある．この場合のプログラムを以下に示す．

```
1: def strings(cs, n, s):
2:     "文字列を辞書順で再帰的に表示"
3:     print(s)
4:     if n >= 1:
5:         for c in cs:
6:             strings(cs, n-1, s+c)
```

文字列の構築と表示の手続きを一つにまとめられるが，文字列を表示する以外の目的には使えなくなる．また，文字列すべてを再帰的に構成する以外にも，文字列を一つ受け取って (あるいは記憶して) 次の文字列を構成することを繰り返しても，文字列を列挙できる．記憶量が最小限に抑えられ，列挙の中断や再開が自由にできるが，再帰と比べてアルゴリズムが複雑になる．

●参考文献

文字列の列挙などの組み合わせアルゴリズムについては，次の文献が参考になる．

- “Combinatorial Algorithms”, A. Nijenhuis and H. S. Wilf, second edition, Academic Press, 1978. https://www.math.upenn.edu/~wilf/website/CombAlgDownld.html

---

山田 俊行

https://www.cs.info.mie-u.ac.jp/~toshi/