Efficient Algorithms to Rank and Unrank Permutations in Lexicographic Order

Blai Bonet   Universidad Simón Bolívar   bonet@ldc.usb.ve

Introduction

- Ranking is the process of assigning a given permutation π over n numbers, a unique index (rank) $r(\pi) \in \{0, \ldots, n!-1\}$
- Unranking is the inverse: given a rank $r \in \{0, \ldots, n!-1\}$, find a permutation π such that $r(\pi) = r$
- Ranking is used to compute the index for PDB lookups, and is the **most critical** operation during node generation
- Ranking/unranking algorithms are said to be in lexicographic order iff lexicographically consecutive permutations are ranked into consecutive integers. Thus, e.g., the identity permutation is ranked into 0 and the reverse permutations is ranked into $n!-1$
- It has been argued that a lexicographic ranking can be more efficient for search as it tend to generate references (PDB indices) with some degree of locality

Related Work

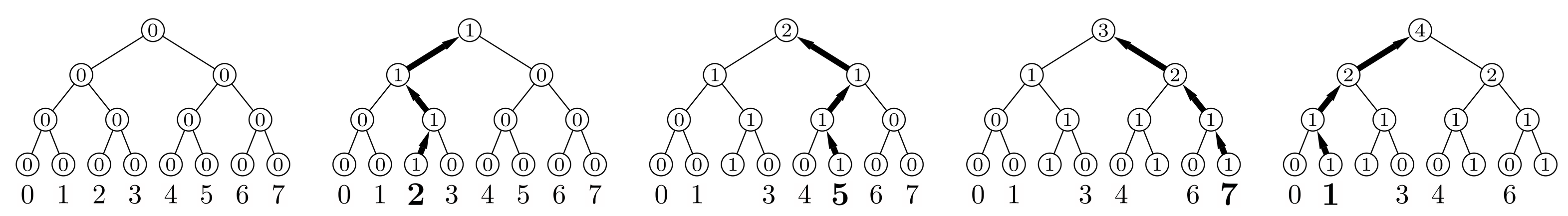
- Naive algorithms run in $O(n^2)$ time
- Knuth (1973) presents $O(n \log n)$ lexicographic algorithms based on modular arithmetic
- Using a data structure of Dietz (1989), the number of operations can be reduced to $O(n \log n/\log\log n)$ yet the implementation is complex and hidden constants are high
- Myrvold and Ruskey (2001) present non-lexicographic algorithms that run in linear time (these are commonly used in heuristic search)
- Korf and Schultze (2005) present linear time lexicographic algorithms that require a precomputed table of exponential size
- Korf and Schultze's algorithms are **non-uniform**

Contribution

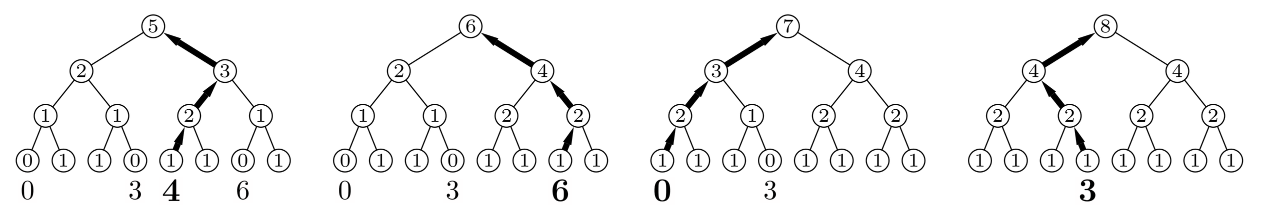
- Novel lexicographic algorithms that run in $O(n \log n)$ time
- The algorithms are simple and efficient and, in our experiments, ran faster than the linear time algorithms of Myrvold and Ruskey for permutations up to size 128, and fell a bit short for sizes up to 1,024
- Also, give novel non-uniform lexicographic algorithms that require much less space than Korf and Schultze's

Uniform Algorithms

Ranking: $\pi = \langle 25714603 \rangle \to d = \langle 24512200 \rangle \to r = \#$

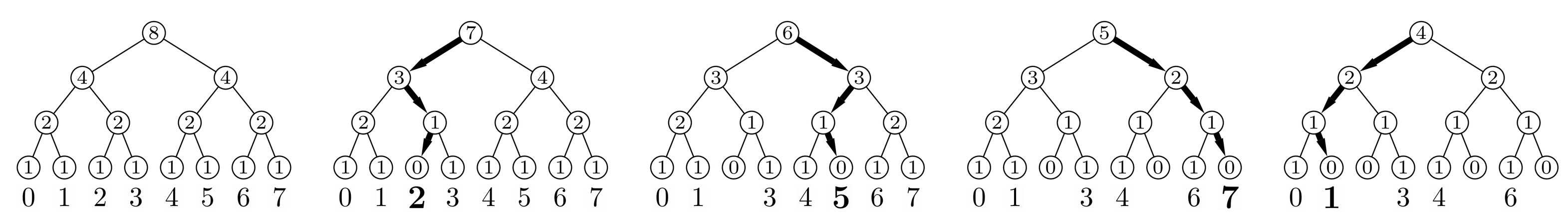


(a) initial   (b) $\pi_0 = 2, d_0 = 2$ (c) $\pi_1 = 5, d_1 = 4$ (d) $\pi_2 = 7, d_2 = 5$ (e) $\pi_3 = 1, d_3 = 1$

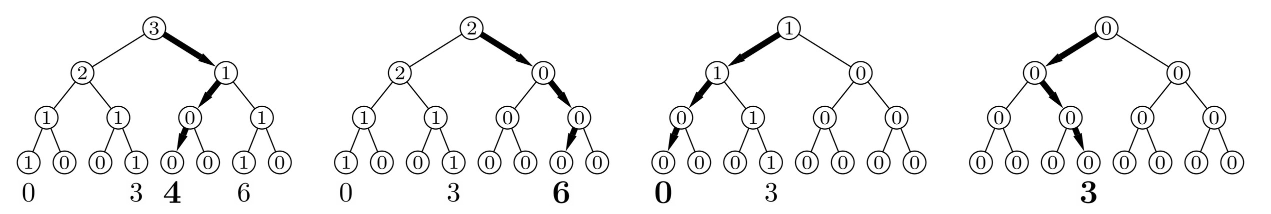


(f) $\pi_4 = 4, d_4 = 2$ (g) $\pi_5 = 6, d_5 = 2$ (h) $\pi_6 = 0, d_6 = 0$ (i) $\pi_7 = 3, d_7 = 0$

Unranking: $r = \# \to d = \langle 24512200 \rangle \to \pi = \langle 25714603 \rangle$



(a) initial   (b) $d_0 = 2, \pi_0 = 2$ (c) $d_1 = 4, \pi_1 = 5$ (d) $d_2 = 5, \pi_2 = 7$ (e) $d_3 = 1, \pi_3 = 1$



(f) $d_4 = 2, \pi_4 = 4$ (g) $d_5 = 2, \pi_5 = 6$ (h) $d_6 = 0, \pi_6 = 0$ (i) $d_7 = 0, \pi_7 = 3$

```
Input: permutation π and array T
Output: rank of π
begin
    k := ⌈log n⌉
    rank := 0
    for i = 1 to 2^(1+k) − 1 do T[i] := 0
    for i = 1 to n do
        ctr := π[i]
        node := 2^k + π[i]
        for j = 1 to k do
            if node is odd then
                ctr := ctr − T[(node ≫ 1) ≪ 1]
            T[node] := T[node] + 1
            node := node ≫ 1
        T[node] := T[node] + 1
        rank := rank · (n + 1 − i) + ctr
    return rank
end
```

```
Input: factorial digits d, array π, and array T
Output: permutation π
begin
    k := ⌈log n⌉
    for i = 0 to k do
        for j = 1 to 2^i do
            T[2^i + j − 1] := 1 ≪ (k − i)
    for i = 1 to n do
        digit := d[i]
        node := 1
        for j = 1 to k do
            T[node] := T[node] − 1
            node := node ≪ 1
            if digit ≥ T[node] then
                digit := digit − T[node]
                node := node + 1
        T[node] := 0
        π[i] ← node − 2^k
end
```

Non-Uniform Algorithms

Korf and Schultze's Ranking Algorithm:

$$
\begin{aligned}
d_i &= \pi_i - \#\{\text{elements to the left of position } i \text{ that are less than } \pi_i\} \\
&= \pi_i - T_r[N \ll (n - \pi_i)]
\end{aligned}
$$

where $T_r[x]$ equals the number of 1-bits in the binary representation of x. In the example,

i	π_i	N before	$N \ll (n-\pi_i)$	T_r	d_i	N after
0	2	0000 0000	0000 0000	0	2	0000 0100
1	5	0000 0100	0010 0000	1	4	0010 0100
2	7	0010 0100	0100 1000	2	5	1010 0100
3	1	1010 0100	0000 0000	0	1	1010 0110
4	4	1010 0110	0110 0000	2	2	1011 0110
5	6	1011 0110	1101 1000	4	2	1111 0110
6	0	1111 0110	0000 0000	0	0	1111 0111
7	3	1111 0111	1110 0000	3	0	1111 1111

New Algorithm:

- Replace table T_r of size $O(2^n \log n)$ bits by a single table T_r^* of size $O(2^m \log m)$ bits, and perform $\lceil n/m \rceil$ lookups instead of one. For example, for $m = 4$, we have that

$$
T_r[x] \;=\; T_r^*[x \& 1111] \;+\; T_r^*[(x \gg 4) \& 1111]\,.
$$

Therefore,

$$
\begin{aligned}
d_i &= \pi_i - T_r[N \ll (n - \pi_i)] \\
&= \pi_i - T_r^*[(N \ll (n - \pi_i)) \& 1111] - T_r^*[((N \ll (n - \pi_i)) \gg 4) \& 1111] \\
&= \pi_i - T_r^*[N_1] - T_r^*[N_2]\,.
\end{aligned}
$$

In the example,

i	π_i	N before	$N \ll (n-\pi_i)$	$T_r^*[N_2]$	$T_r^*[N_1]$	d_i	N after
0	2	0000 0000	0000 0000	0	0	2	0000 0100
1	5	0000 0100	0010 0000	1	0	4	0010 0100
2	7	0010 0100	0100 1000	1	1	5	1010 0100
3	1	1010 0100	0000 0000	0	0	1	1010 0110
4	4	1010 0110	0110 0000	2	0	2	1011 0110
5	6	1011 0110	1101 1000	3	1	2	1111 0110
6	0	1111 0110	0000 0000	0	0	0	1111 0111
7	3	1111 0111	1110 0000	3	0	0	1111 1111

Interesting Tradeoffs:

Θ	$m = 1$	$m = \log n$	$m = n^\epsilon$	$m = n/\log n$	$m = n/k$	$m = n$
time	n^2	$n^2/\log n$	$n^{2-\epsilon}$	$n \log n$	n	n
space	1	$n \log\log n$	$2^{n^\epsilon} \log n$	$2^{n/\log n} \log n$	$2^{n/k} \log n$	$2^n \log n$

Experiments