Algorithm Analysis and Design

Combinatorics

Dimitri Volper

1 Naïve Permutation Generation

Definition 1: A **permutation** is a possible arrangement of a collection of things where *order is important*.

Algorithm 1:
For each permutation of length $n-1$ and generates n permutations of length n by inserting n at positions $0, \dots, n-1$:

If abc is $n-1$ in length then there are four possible permutations for the final permutation: $\_abc, a\_bc, ab\_c, abc\_$

n	result
1	$\{(1)\}$
2	$\{(2,1),(1,2)\}$
3	$\{(3,2,1),(2,3,1),(2,1,3),(3,1,2),(1,3,2),(1,2,3)\}$

Algorithm 2:
For each permutation of length $n-1$, generate n permutations of length n by inserting n at positions $n-1, \dots, 0$:

If abc is $n-1$ in length then there are four possible permutations for the final permutation: $abc\_, ab\_c, a\_bc, \_abc$

n	result
1	$\{(1)\}$
2	$\{(1,2),(2,1)\}$
3	$\{(1,2,3),(1,3,2),(3,1,2),(2,1,3),(2,3,1),(3,2,1)\}$

Definition 2: **Minimal change** is when the next permutation can be generated by swapping the position of two elements. This property is desired when generating permutations because it has a small memory complexity.

2 Johnson-Trotter Algorithm

Each element has associated with it a direction denoted by either a left (←) or right (←) arrow.

Definition 3: An element is **mobile** if:

$$\begin{cases} \{ \vec{a} \quad , \quad \overset{?}{b} \} \quad \textit{and} \;\; a > b \\ \\ \{ \overset{?}{a} \quad , \quad \overset{\leftarrow}{b} \} \quad \textit{and} \;\; a < b \end{cases}$$

That is – element is **mobile** if its arrow is pointing at a smaller element.

Example 1:

$$\{ \overset{\leftarrow}{3} \;,\; \overset{\rightarrow}{2} \;,\; \overset{\rightarrow}{1} \;,\; \overset{\leftarrow}{4} \}$$

With elements 2 and 4 being mobile.

Generating next permutation:

1. Initialize all elements with ←.
2. While the last permutation generated has a mobile element:
   (a) Find the largest mobile element k.
   (b) Swap k with the element k is pointing to.
   (c) Reverse the direction of all elements that are larger than k.

Example 2:
Given the sequence $\{1, 2, 3\}$, generate the next permutation.

1. Initialize

$$\{ \overset{\leftarrow}{1} \;,\; \overset{\leftarrow}{2} \;,\; \overset{\leftarrow}{3} \}$$

2. (a) Largest mobile element: 3
   (b) Swap 3 with 2.
   (c) Elements to reverse direction: none

$$\{ \overset{\leftarrow}{1} \;,\; \overset{\leftarrow}{3} \;,\; \overset{\leftarrow}{2} \}$$

3. (a) Largest mobile element: 3
   (b) Swap 3 with 1.
   (c) Elements to reverse direction: none

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

4. (a) Largest mobile element: 2
   (b) Swap 2 with 1.
   (c) Elements to reverse direction: 3

$$\{ \overset{\rightarrow}{3} , \overset{\leftarrow}{2} , \overset{\leftarrow}{1} \}$$

5. (a) Largest mobile element: 3
   (b) Swap 3 with 2.
   (c) Elements to reverse direction: none

$$\{ \overset{\leftarrow}{2} , \overset{\rightarrow}{3} , \overset{\leftarrow}{1} \}$$

6. (a) Largest mobile element: 3
   (b) Swap 3 with 1.
   (c) Elements to reverse direction: none

$$\{ \overset{\leftarrow}{2} , \overset{\leftarrow}{1} , \overset{\rightarrow}{3} \}$$

7. (a) Largest mobile element: none
   (b) Elements to reverse direction: none

   Loop terminates.

3 Lexicographical order of permutations (LEX1)

Definition 4: A sequence of elements is in **lexicographical order** if it is sorted from largest to smallest.

To ensure minimal change, we want to preserve as many digits on the left side of a sequence as possible.

Example 3: $\{1, 6, 2, 5, 4, 3\}$

1. If we preserve the first *three* digits, then it will result in *hundreds* of numbers larger than the original sequence.
2. If we preserve the first *four* digits, then it will result in *tens* of numbers larger than the original sequence.

Given a permutation $p_j = \{a_1, a_2, \ldots, a_n\}$ find the next permutation p_{j+1}

Steps:

1. Scan it right to left to find the first pair of elements such that $a_i < a_{i+1}$.
2. Insert all elements from $\{a_0, \ldots, a_{i-1}\}$ into p_{j+1}.
3. Find the smallest a_k within the "tail" $(a_{i+2}, \ldots, a_n)$ such that it $a_k > a_i$.
4. Insert a_k at the end of p_{j+1}.
5. Insert at the end of p_{j+1} all remaining elements in increasing order.

Example 4:
Given the permutation $p_j = \{1, 6, 2, 5, 4, 3\}$ find the next permutation p_{j+1}

1. (a) Pair: $2 < 5$
   (b) $i = 2$
   (c) Tail: $\{4, 3\}$
   (d) Smallest element larger than a_i in tail: 3
2. Insert 1 and 6 into p_{j+1}: $\{1, 6\}$
3. Insert 3 at the end of p_{j+1}: $\{1, 6, 3\}$
4. Insert remaining elements ($\{2, 5, 4\}$) in order at the end of p_{j+1}: $\{1, 6, 3, 2, 4, 5\}$

4 Subsets

Definition 5: Distinct elements similar to permutations which the order is not important.

Definition 6: **Squashed order** is when subsets involving a number j are only listed after all subsets involving $0, \ldots, j-1$ are listed.

Example 5: Adding a new element to a sequence and the resulting subsets of that sequence:

$$1: \quad \{\}, \{1\}$$

$$2: \quad \{\}, \{1\}, \{2\}, \{1,2\}$$

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

The correctness of this algorithm can be proven easily to produce all possible 2^{n+1} subsets that are all unique.

Example 6: We can represent all possible subsets in squashed order using a bit pattern:

Iteration	Bit Pattern	Subset
0	0000	$\{\}$
1	0001	$\{1\}$
2	0010	$\{2\}$
3	0011	$\{2,1\}$
4	0100	$\{3\}$
5	0101	$\{3,1\}$
6	0110	$\{3,2\}$
7	0111	$\{3,2,1\}$
8	1000	$\{4\}$
9	1001	$\{4,1\}$
10	1010	$\{4,2\}$
11	1011	$\{4,2,1\}$
12	1100	$\{4,3\}$
13	1101	$\{4,3,1\}$
14	1110	$\{4,3,2\}$
15	1111	$\{4,3,2,1\}$

Each sequence is lexicographical because it is sorted from the largest to smallest element.

Definition 7: Hills is the pattern that gets generated when each subset is sorted from the smallest to largest element.

Iteration	Bit Pattern	Subset
0	0000	$\{\}$
1	0001	$\{1\}$
2	0010	$\{1,2\}$
3	0011	$\{1,2,3\}$
4	0100	$\{1,2,3,4\}$
5	0101	$\{1,2,4\}$
6	0110	$\{1,3\}$
7	0111	$\{1,3,4\}$
8	1000	$\{1,4\}$
9	1001	$\{2\}$
10	1010	$\{2,3\}$
11	1011	$\{2,3,4\}$
12	1100	$\{2,4\}$
13	1101	$\{3\}$
14	1110	$\{3,4\}$
15	1111	$\{4\}$

Implementation:
Instead of returning an array of all subsets/permutations, a ”on request” implementation looks like this:

```
int main ()
{
    int size = 4;
    Subsets subsets_generator(size);
    std::vector<unsigned int> subset(size);

    do
    {
        std::cout << "{";
        for (unsigned int i = 0; i < subset.size(); ++i)
            std::cout << subset[i] << " ";
        std::cout << "}" << std::endl;
    }
    while ( subsets_generator.next(subsets) );
}
```

Notice that STL permutations are implemented in the same way (but with a function):

```
#include <algorithm>
#include <iostream>
#include <iterator> //ostream_iterator

int main()
{
    int A[] = {1,2,3,4};
    const int N = sizeof(A) / sizeof(int);

    do
    {
        std::copy(A, A+N, std::ostream_iterator<int>(std::cout, " "));
        std::cout << std::endl;
    }
    while (std::next_permutation(A,A+N));
}
```

Output (notice that it is in lexicographical order):

```
1 2 3 4
1 2 4 3
1 3 2 4
1 3 4 2
1 4 2 3
1 4 3 2
2 1 3 4
2 1 4 3
2 3 1 4
2 3 4 1
2 4 1 3
2 4 3 1
3 1 2 4
3 1 4 2
3 2 1 4
3 2 4 1
3 4 1 2
3 4 2 1
4 1 2 3
4 1 3 2
4 2 1 3
4 2 3 1
4 3 1 2
4 3 2 1
```

We can also use *next_permutation* to implement the worst known deterministic sorting algorithm. Most sorting algorithms are $O(Nlog(N))$, and even bubble sort is only $O(N^2)$. This algorithm is actually $O(N!)$:

```
#include <algorithm>
#include <iostream>
#include <iterator> //ostream_iterator

template <class BidirectionalIterator>
void snail_sort(BidirectionalIterator first, BidirectionalIterator last)
{
    while (std::next_permutation(first, last)) {}
}

int main()
{
    int A[] = {8, 3, 6, 1, 2, 5, 7, 4};
    const int N = sizeof(A) / sizeof(int);

    snail_sort(A, A+N);
    std::copy(A, A+N, std::ostream_iterator<int>(std::cout, " "));
    std::cout << std::endl;
}
```

5 Gray Code

Definition 8: The ordering of binary numbers $0, \ldots, 2^{n-1}$ so that each successive permutation differs by only one bit.

Example 7:

Index	Bit Pattern
0	0000
1	0001
2	0011
3	0010
4	0110
5	0111
6	0101
7	0100
8	1100
9	1101
10	1111
11	1110
12	1010
13	1011
14	1001
15	1000

Uses of Gray code:
Algorithms may benefit from the fact that the current iteration is only slightly different from the previous. For example, each successive element may reuse data from the previous iteration and altering it.

Generating the Gray code table for a binary sequence (recursive):

1. Reference the table using $n - 1$ bits.
2. Reflect the bits (i.e. listing them in reverse order and concatenating the reversed list onto the original list).
3. Prefix the original bits with a binary 0.
4. Prefix the reflected bits with a binary 1.

The base case is when $n = 1$ bit and is the most basic Gray code.

Index	Bit Pattern
0	0
1	1

Example 8: $n = 2$:

1. Reference the table using $n = 1$ bits:

Index	Bit Pattern
0	0
1	1

2. Reflect the bits:

Index	Bit Pattern
0	0
1	1
2	1
3	0

3. Prefix the original bits with a binary 0:

Index	Bit Pattern
0	00
1	01
2	1
3	0

4. Prefix the reflected bits with a binary 1:

Index	Bit Pattern
0	00
1	01
2	11
3	10

Example 9: $n = 3$:

1. Reference the table using $n = 2$ bits:

Index	Bit Pattern
0	00
1	01
2	11
3	10

2. Reflect the bits:

Index	Bit Pattern
0	00
1	01
2	11
3	10
4	10
5	11
6	01
7	00

3. Prefix the original bits with a binary 0:

Index	Bit Pattern
0	000
1	001
2	011
3	010
4	10
5	11
6	01
7	00

4. Prefix the reflected bits with a binary 1:

Index	Bit Pattern
0	000
1	001
2	011
3	010
4	110
5	111
6	101
7	100

Example 10: $n = 4$:

Index	Bit Pattern
0	0000
1	0001
2	0011
3	0010
4	0110
5	0111
6	0101
7	0100
8	1100
9	1101
10	1111
11	1110
12	1010
13	1011
14	1001
15	1000

6 Partitions

Definition 9: A **integer partition** is all ways of representing an integer as a sum where *the order does not matter.*

Example 11:
$\{6\}$
$\{5+1\}$
$\{4+2\}$
$\{4+1+1\}$
$\{3+3\}$
$\{3+2+1\}$
$\{3+1+1+1\}$
$\{2+2+2\}$
$\{2+2+1+1\}$
$\{2+1+1+1+1\}$
$\{1+1+1+1+1+1\}$

Generating the next partition (similar to lexicographical order):

1. Find smallest number k such that $k > 1$.
2. Replace k such that $k = (k-1)$.
3. Add 1 to new partition.
4. Replace all 1's with their sum.

Example 12: $5+4+1+1$

1. Find smallest number k such that $k > 1$.
   $k = 4$
2. Replace k such that $k = (k-1)$.
   $5+3+1+1$
3. Add 1 to new partition.
   $5+3+1+1+1$
4. Replace all 1's with their sum.
   $5+3+3$

Definition 10: A **set partition** is all ways of diving n elements into subsets where *the order does not matter.*

Example 13: A sequence of 3 elements has 5 partitions:

Index	Partition
0	$\{1,2,3\}$
1	$\{1\},\{2,3\}$
2	$\{2\},\{1,3\}$
3	$\{3\},\{1,2\}$
4	$\{1\},\{2\},\{3\}$

Recursive implementation (obvious): Each partition of $n-1$ elements generates several partitions of n elements by adding the new element to each subset and as a stand-alone subset.

Example 13:
Given $n=4$ and all possible elements being $\{1,2,3,4\}$, then $\{\{2\},\{1,3\}\}$ generates 3 partitions of 4 elements:

$\{2,4\},\{1,3\}$	4 added to first subset
$\{2\},\{1,3,4\}$	4 added to second subset
$\{2\},\{1,3\},\{4\}$	4 added to a separate subset

There are 15 distinct partitions of 4 elements, 52 for 5 elements, and 203 for 6 elements.

Iterative implementation using integer partitions: Notice that each partition corresponds to an integer by counting the number of elements in each subset.

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

Each integer partition generates several set partitions, namely it's a product of some binomial coefficients. For example, to form subsets of sizes $s_1,\ldots,s_k$ of n elements $(s_1+...+s_k = n * Bin(s_1,n) * Bin(s_2,n-s_1) * \ldots * B(s_k,s_k))$

7 Combinations

Recursive Algorithm: given all combinations of $k-1$ out of $n-1$ elements AND k out of $n-1$ elements, we can build k out of n:

1. Adding the n^{th} element to all the $k-1$ out of $n-1$ element combinations.
2. Copy k out of $n-1$

Iterative Algorithm: http://www-cs-faculty.stanford.edu/ knuth/fasc3a.ps.gz