VNU Journal of Science, Mathematics - Physics 28 (2012) 1-10

Some new combinatorial algorithms with appropriate representations of solutions

Hoang Chi Thanh[1,*], Nguyen Quang Thanh[1]

[1]*Department of Computer Science, VNU University of Science, 334-Nguyen Trai Street, Thanh Xuan, Hanoi*

Received 01 September 2011, received in revised form 30 March 2012

Abstract. Combinatorial problems are those problems, whose requirements are an association of some conditions. The construction of efficient algorithms to find solutions of the combinatorial problems is still an interesting matter. In this paper, we choose appropriate representations for desirable solutions of the permutation problem and the partition problem. Then we sort the representations of a problem's solutions in the alphabetical order. Owing to it we construct two new algorithms for quickly finding all solutions of these problems.

1 Introduction

Permutations and partitions of a finite set are applied in many areas of sciences and technologies, e.g. scheduling problem, control problem and path finding problem... So modifying an exciting algorithm or constructing a new algorithm to generate permutations or partitions are attracted many researches [1-6,8]. There are some algorithms to generate a set's permutations, e.g. a reverse alphabetical order algorithm, an algorithm based on adjacent transpositions…[2-6] and some algorithms to generate a set's partitions, e.g. an algorithm based on integer pointers [5,6], an algorithm based on matrix [1]... Recently, J. Ginsburg constructed a method determining a permutation from its set of reductions [2], M. Monks reconstructed permutations from cycle minors [4] and T. Kuo proposed a new method for generating permutations based on factorial digits [3]. But the above algorithms are rather long and complicated.

When designing an algorithm to a problem, one of the first steps is a solution representation. An appropriate representation can make the algorithm simpler and faster. Based on the notion of inversion of a permutation [5], we propose a notion of inversion vector and show that the inversion vector becomes a good representation of permutations. We construct a novel algorithm generating a set's permutations by inversion vectors. Using indices of the blocks in a partition we can represent the

[*] Corresponding author. Tel.: +84 91 2011 765
Email: thanhhc@vnu.vn

partition by a sequence of indices and construct a new efficient algorithm to generate a set's all partitions. The algorithms are short, simple and easy to implement.

The remainder of this paper is organized as follows. In section 2 we present permutations generation by inversion vectors. The section 3 is devoted to partition problem. The last section contains conclusion.

2. Permutations generation by inversion vectors

Let X be a finite set. *A permutation* of the set X is a checklist of its all elements. It is easy to see that each permutation of the set X is a bijection from X to itself.

2.1 Permutation problem

Given an n-element set X. Find all permutations of the set X.

It is easy to show that the number of permutations is $n!$. The number of elements n is as big as great the time to find all permutations.

Identify $X \equiv \{1, 2, \ldots, n\}$. Let denote S_n the set of all permutations of the n-element set X. A permutation $f \in S_n$ can be represented by a positive integer sequence of the length n as follows:

$$f = < f(1)\ f(2) \ldots f(n) >, \text{ where } f(i) \in X \text{ and } f(i) \neq f(j),\ 1 \leq i \neq j \leq n.$$

For simplicity, the above sequence can be written as:

$$f = < a_1\, a_2 \ldots a_n >, \text{ where } a_i = f(i)\ ,\ i = 1, 2, \ldots, n.$$

In the sequence there is a pair of integers, the preceding is greater than following one. Such a pair is an inversion of the permutation.

Definition 2.1: A pair (a_i, a_j) with $i < j$ is called *an inversion* of the permutation $< a_1\, a_2 \ldots a_n >$ iff $a_i > a_j$.

The notion of inversion is used to determine the sign of a permutation [5,6]. We use it to generate all permutations.

2.2 Inversion vector of a permutation

An inversion vector of a permutation is defined as follows:

Definition 2.2: The n-dimension vector $(d_1, d_2, ..., d_n)$ is called *an inversion vector* of the permutation $< a_1\, a_2 \, ... \, a_n >$ iff $d_j = |\{ i \mid i < j , a_i > a_j \}| , j = 1, 2, ..., n$.

By definition, the coordinator d_j is indeed the number of components of the permutation, greater than a_j and to its left. The coordinator d_j is called the number of inversions created by the component a_j. Note that following components do not effect the number of inversions of preceding ones.

Example 2.3: The sequences of permutations and inversion vectors of an 3-element set.

No	Permutations	Inversion vectors
1	1 2 3	0 0 0
2	1 3 2	0 0 1
3	2 3 1	0 0 2
4	2 1 3	0 1 0
5	3 1 2	0 1 1
6	3 2 1	0 1 2

By Definition 2.2, the first coordinator d_1 of an inversion vector of any permutation always equals 0 (one choice), the second coordinator d_2 equals 0 or 1 (two choices)... So:

$$0 \le d_j \le j - 1 \text{ , where } j = 1, 2, ..., n. \qquad (2.1)$$

Let denote $\mathbb{N}$ the set of all positive integers. Set:

$$V_n = \{ (d_1, d_2, ..., d_n) \mid d_j \in \mathbb{N} , 0 \le d_j \le j - 1 \text{ , } j = 1, 2, ..., n \}.$$

This is the set of all n-dimension integer vectors satisfying (2.1). Of course:

$$| V_n | = n!$$

The relationship between the set of permutations S_n and the set of vectors V_n is pointed out by the following theorem.

Theorem 2.1: Two sets S_n and V_n are isomorphic.

Proof: Construct a mapping $H : S_n \to V_n$ as follows:

$$\forall f = < a_1\, a_2 \, ... \, a_n > \in S_n \text{ , } H(f) = (d_1, d_2, ..., d_n),$$

where $(d_1, d_2, ..., d_n)$ is the inversion vector of the permutation $< a_1\, a_2 \, ... \, a_n >$.

We show that the mapping H is a bijection. Because two sets S_n and V_n are finite and have the same cardinality, so we show only that the mapping H is injective.

Assume that two following permutations have the same inversion vector:

$$\exists < a_1\, a_2 \, ... \, a_n >, < b_1\, b_2 \, ... \, b_n > \in S_n : (d_1^a, d_2^a, ..., d_n^a) = (d_1^b, d_2^b, ..., d_n^b).$$

Due to $d_n^a = d_n^b$ so both a_n and b_n are less than d_n^a elements in X. So, $a_n = b_n$ $(= n - d_n^a)$.

Remove a_n and b_n in the above permutations, we get two permutations $< a_1 a_2 \dots a_{n-1} >$, $< b_1 b_2 \dots b_{n-1} >$ of the n-1 element set $X \setminus \{a_n\}$ that have the same inversion vector (d_1^a, d_2^a, ..., d_{n-1}^a). Repeat the above reasoning we have: $a_{n-1} = b_{n-1}$.

Analogously, we show that: $< a_1 a_2 \dots a_n > = < b_1 b_2 \dots b_n >$. The two permutations are identical. So the mapping H is an injection. This proves the theorem.

Thus, the number of inversion vectors of permutations of a set is equal to the number of the permutations. In other words, each vector belonging to V_n is indeed an inversion vector of some permutation in S_n.

One used to represent a permutation of an n-element set by a matrix of 2 rows and n columns or an integer sequence with the length of n or a directed graph of n nodes [5,6]. Theorem 2.1 points out that inversion vector is a good representation of permutations.

2.3 Using inversion vectors to generate permutations

Based on Theorem 2.1 we construct a new two step algorithm to generate all permutations of an n-element set as follows:

1) Consequently generating inversion vectors in the set V_n.
2) Determining a permutation corresponding to the just found inversion vector.

2.3.1 Generating inversion vectors

To perform the step 1 we consider each inversion vector (d_1, d_2, ..., d_n) in V_n as a word $d_1\ d_2 \dots d_n$ on the alphabet $\mathcal{N}$. We sort these words in ascending by the alphabetical order (see Example 2.3). So,

- The first inversion vector (the least) is 0 0 0 ... 0 0, corresponding to the identical permutation < 1 2 3 ... n-1 n >.

- The last inversion vector (the most) is 0 1 2 ... n-2 n-1, corresponding to the permutation < n n-1 n-2 ... 2 1 >, that is the reverse of the identical permutation.

Assume that $d = (d_1, d_2, \dots, d_n)$ is an inversion vector. We have to find an inversion vector $d' = (d'_1, d'_2, \dots, d'_n)$ next to the above inversion vector in the sorted sequence.

By the alphabetical order, the vector d' is inherited a left part as long as possible of the vector d from the coordinator indexed 1 to the coordinator indexed k-1, where:

$$k = \max\{ j \mid d_j < j - 1 \}.$$

Then the coordinators indexed from 1 to k-1 are unchanged:

$$d'_i = d_i \ , \ i = 1, 2, \dots, k\text{-}1;$$

The coordinators indexed from k to the last are determined as follows:

$$d'_k = d_k + 1,$$
$$\text{and } d'_i = 0 , i = k+1, k+2, ..., n.$$

The step 1 terminates when all inversion vectors in V_n have been generated. It means, when the last inversion vector 0 1 2 ... n-2 n-1 was generated. At that time, the variable: $k = 0$.

This is a termination condition of this algorithm.

2.3.2 Determining permutation from inversion vector

Let (d_1, d_2, ... , d_n) be an inversion vector. We have to find a permutation $< a_1\, a_2 \, ... \, a_n >$, whose inversion vector is the above vector. All components a_i (i = 1, 2, ..., n) of the permutation belong to the set $X = \{1, 2, \ldots, n\}$. To determine the components, we use a list LX represented by an integer array, that contains remaining elements of the set X after each component selection. Elements in the list XS are sorted in ascending. First,

$$LX[i] = i , \ i = 1, 2, ..., n.$$

As a_n is less than d_n elements in the list LX so $a_n = LX[n - d_n]$.

Remove a_n from LX and gather the list, we get a new n-1 element list. Then, $a_{n-1} = LX[n\text{-}1\text{-}d_{n-1}]$.

Repeat the above process until the list LX becomes empty, we find all components of the permutation $< a_1\, a_2 \, ... \, a_n >$.

2.3.3 New algorithm generating permutation from inversion vector

Use an integer array $D[1..n]$ to contain inversion vectors, an integer array $F[1..n]$ for permutations and an integer variable l for the current length of the list LX.

As the above analysis we construct a detail algorithm to generate permutations from inversion vectors as follows.

Algorithm 2.1 (Generating permutation from inversion vector):

Input: A positive integer n.

Output: A sequence of all permutations of the set $\{1, 2, \ldots, n\}$, their inversion vectors are sorted by ascending in the alphabetical order.

```
Begin
  D[1..n] ← 0 ;
  repeat
    FIND_PER();
    k ← n ;
    while D[k] = k−1 do { D[k] ← 0 ; k ← k−1 } ;
    if k ≥ 2 then D[k] ← D[k]+1 ;
  until k = 0 ;
```

```
9 End.

10 Procedure FIND_PER() ;
11  begin
12    for i ← 1 to n do LX[i]← i ;
13    l ← n ;
14    for i ←n downto 1 do
15       { j ←l-D[i] ; F[i] ←LX[j] ; l  ←l-1 ;
16         for i ←j to l do LX[i] ←LX[i+1] ; }
17    print F[1..n] ;
18 end ;
```

Complexity of the algorithm:

To generate a permutation, the algorithm executes two following steps:

- Instructions 5-7 determine a changing position k and generates an inversion vector with the complexity of $O(n)$.

- The procedure *FIND_PER*() in instructions 10-18) determines and prints the corresponding permutation by the procedure call 4) with the complexity of $O(n.\ln n)$.

So the complexity of a permutation generation is $O(n.\ln n)$. The total complexity of the algorithm 2.1 is $O(n.\ n!.\ln n)$.

The complexities of the algorithm 2.1, the reverse alphabetical order algorithm and the algorithm based on adjacent transpositions presented in [5,6] are the same. But the algorithm 2.1 is more simpler.

3. Generating set partitions by indices

3.1 Set partition problem

Let X be a finite set.

Definition 3.1: *A partition* of the set X is a family $\{A_1, A_2, \ldots, A_k\}$ of subsets of X, satisfying the following properties:

1) $A_i \neq \varnothing$, $1 \leq i \leq k$;
2) $A_i \cap A_j = \varnothing$, $1 \leq i < j \leq k$;
3) $A_1 \cup A_2 \cup \ldots \cup A_k = X$.

Problem: Given a set X. Find all partitions of the set X.

The number of all partitions of an n-element set is denoted by Bell number B_n , calculated by the following recursive formula [5,6]:

$$B_n = \sum_{i=0}^{n-1} \binom{n-1}{i} B_i \text{ , where } B_0 = 1.$$

The number of all partitions of an n-element set grows up as quickly as the factorial function does. For example,

n	B_n
1	1
3	5
5	52
8	4.140
10	115.975
15	1.382.958.545
20	51.724.158.235.372

Given an n-element set X. Let identify the set $X \equiv \{1, 2, 3, ..., n\}$.

To ensure the uniqueness of representation, we sort subsets in a partition on their least element and enumerate these subsets starting with 1.

Let $\pi = \{A_1, A_2, \ldots, A_k\}$ be a partition of the set X. Each subset A_i is called *a block* of the partition π. In the partition, the block A_i (i = 1, 2, 3, ...) has the index i. Each element $j \in X$, belonging to some block A_i has also the index i. It means, every element of X can be represented by the index of a block where this element resides.

Of course, the index of element j is not greater than j. Each partition can be represented by a sequence of n indices. The sequence can be considered as a word with the length of n on the alphabet X. We sort these words in the ascending order. Then,

- The smallest word is 1 1 1 ... 1. It corresponds to the partition $\{\{1,2,3, ... , n\}\}$. This partition consists of one block only.

- The greatest word is 1 2 3 ... n. It corresponds to the partition $\{\{1\}, \{2\}, \{3\}, ... , \{n\}\}$. This partition consists of n blocks, each block has only one element. This is an unique partition that has a block with the index n.

Theorem 3.1: For $n \geq 0$, $B_n \leq n!$.

It means, the number of an n element set's all partitions is not greater than the number of all permutations on the same set.

Proof: Follow from the index sequence representation of partitions.

We use an integer array $IA[1..n]$ to represent a partition, where $IA[i]$ stores the index of the block that includes element i. Element 1 always belongs to the first block, element 2 may belong to the first or the second block. If element 2 belongs to the first block then element 3 may belong to the first or the second block only. And if the element 2 belongs to the second block then element 3 may belong to the first, the second or the third block.

Hence, the element i may only belong to the following blocks:

$$1, 2, 3, \dots, \max(IA[1], IA[2], \dots, IA[i\text{-}1]) + 1.$$

It means, for every partition:

$$1 \le IA[i] \le \max(IA[1], IA[2], \dots, IA[i\text{-}1]) + 1 \le i \text{ , where } i = 2, 3, \dots, n.$$

This is an invariant for all partitions of a set. We use it to find partitions.

Example 3.2: The partitions of a three-element set and the sequence of their index representations.

No	Partitions	$IA[1..3]$
1	{{1, 2, 3}}	1 1 1
2	{{1, 2}, {3}}	1 1 2
3	{{1, 3}, {2}}	1 2 1
4	{{1}, {2, 3}}	1 2 2
5	{{1}, {2}, {3}}	1 2 3

3.2 A new algorithm for partitions generation

It is easy to determine a partition from its index array representation. So, instead of generating all partitions of the set X we find all index arrays $IA[1..n]$, each of them can represent a partition of X. These index arrays will be sorted in the ascending order.

The first index array is 1 1 1 ... 1 1 and the last index array is 1 2 3 ... n-1 n. So the termination condition of the algorithm is:

$$IA[n] = n$$

Let $IA[1..n]$ be an index array representing some partition of X and let $IA'[1..n]$ denote the index array next to IA in the ascending order.

To find the index array IA' we use an integer array $Imax[1..n]$, where $Imax[i]$ stores $\max(IA[1], IA[2], \dots, IA[i\text{-}1])$. The array $Imax$ gives us possibilities to increase indexes of the array IA. Of course,

$$Imax[1] = 0 \text{ and } Imax[i] = \max(Imax[i\text{-}1], IA[i\text{-}1]) \text{ , } i = 2, 3, \dots, n.$$

Then,

$$IA'[i] = IA[i] \text{ , } i = 1, 2, \dots, p\text{-}1, \text{ where } p = \max\{ j \mid IA[j] \le Imax[j] \};$$

$$IA'[p] = IA[p] + 1 \text{ and } IA'[j] = 1 \text{ , } j = p\text{+}1, p\text{+}2, \dots, n.$$

Basing on the above properties of the index arrays, we construct the following algorithm for generating all partitions of a set.

Algorithm 3.2 (Generation of a set's all partitions)

Input: A positive integer *n*.

Output: A sequence of an *n*-element set's all partitions, whose index representations are sorted by ascending.

```
Begin
 for i ← 1 to n-1 do IA[i] ← 1 ;
 IA[n] ← Imax[1] ← 0 ;
 repeat
   for i ← 2 to n do
    if Imax[i-1] < IA[i-1] then Imax[i] ← IA[i-1]
                           else Imax[i] ← Imax[i-1] ;
   p  ← n ;
   while IA[p] = Imax[p]+1 do
          { IA[p] ← 1 ; p ← p-1 } ;
   IA[p] ← IA[p]+1 ;
   Print the corresponding partition ;
 until IA[n] = n ;
End.
```

The algorithm's complexity:

The algorithm finds an index array and prints the corresponding partition with the complexity of O(n). Therefore, the total complexity of the algorithm is O($B_n.n$). .

The algorithm 3.2 is much simpler and faster than the pointer-based algorithm 1.19 presented in [5].

The algorithm is short, simple and easy to implement.

4. Conclusion

In this paper, we propose two new efficient algorithms to generate all permutations and all partitions of a finite set. Permutations are represented by inversion vectors whilst partitions by sequences of indices. The alphabetical order is used to sort representations of the problem's solutions in both algorithms. The obtained results point out that choosing appropriate representations for desirable solutions takes a great part in algorithm design. It makes an algorithm simpler, shorter and faster.

Acknowledgment

The author would like to acknowledge the Asia Research Center and the Korea Foundation for Advanced Studies as the sponsors for the research project.

References

[1] K. Cameron, E.M. Eschen, C.T. Hoang and R. Sritharan, The list partition problem for graphs, *Proceedings of the fifteenth annual ACM-SIAM symposium on Discrete algorithms*, New Orleans 2004, pp. 391-399.

[2] J. Ginsburg, Determining a permutation from its set of reductions, *Ars Combinatoria,* No. 82, 2007, pp. 55-57.

[3] T. Kuo, A new method for generating permutations in lexicographic order, *Journal of Science and Engineering Technology*, Vol. 5, No. 4, 2009, pp. 21-20.

[4] M. Monks, Reconstructing permutations from cycle minors, *The Electronic Journal of Combinatorics,* No. 16, 2009, #R19.

[5] W. Lipski, Kombinatoryka dla programistów, WNT, Warszawa, 1982.

[6] H.C. Thanh, *Combinatorics*, VNUH Press, 1999 (in Vietnamese).

[7] H.C. Thanh, *Bounded sequence problem and some its applications*, Proceedings of Japan-Vietnam Workshop on Software Engineering, Hanoi - 2010, pp. 74-83.

[8] H.C. Thanh, N.T.T. Loan, N.D. Ham, *From Permutations to Iterative Permutations*, International Journal of Computer Science Engineering and Technology, Vol. 2, Issue 7, 2012, pp. 1310-1315.