Algorithm for Generating All Spanning Trees of Complete Undirected Graphs

El Bekkaye Mermri[1], Hideki Katagiri[2], Masatoshi Sakawa[3] and Kosuke Kato[4]
Graduate School of Engineering, Hiroshima University
1-4-1, Kagamiyama, Higashi Hiroshima, Japan.
[1]mermri,[3]sakawa,[4]kato@msl.sys.hiroshima-u.ac.jp
[2]katagiri-h@hiroshima-u.ac.jp

ABSTRACT

Let G be a complete undirected graph with n vertices, e edges and m spanning trees. In this paper we give an algorithm for finding explicitly all spanning trees of G. Our technique is based on the representation of spanning trees by Prüfer numbers. To represent all Prüfer numbers with $n-2$ digits we define what we call base-n. The algorithm requires a time of $O(nm)$ and space of $O(n)$. For finding all spanning trees explicitly of undirected graphs, the best known algorithm requires a time of $O(e+n+nm)$ and space of $O(e+n)$.

Keywords: Spanning tree; undirected graph; Prüfer number; enumeration problem

1. INTRODUCTION

Let G be an undirected connected graph with n vertices, e edges and m spanning trees. A spanning tree is a connected sub-graph of G which contains all the vertices, but no cycle. The problem of finding all spanning trees of directed and undirected graphs arises in many applications of network and telecommunication designs. This problem has been the interest of many authors, and several algorithms have been proposed [1, 2, 3, 4, 5]. We distinct two kinds of algorithms to enumerate all spanning trees. The first one outputs all spanning trees of the graph explicitly. In the second one, the spanning tree need not be explicitly output and only a computational tree which gives relative changes between spanning trees is required.

Here we consider the problem of finding explicitly all spanning trees of complete undirected graphs. In 1978, Gabow and Myers [2] employed a technique called backtrack based on depth-first search method to solve the problem. Their algorithm requires a time complexity of $O(e+n+nm)$ and space of $O(e+n)$. In 1997, Matsui [4] proposed an algorithm which traverse a rooted spanning tree on its polytope and generates all the spanning trees. This algorithm finds a new spanning tree by exchanging two edges. It requires the same time and space complexities as one of Gabow and Myers. For outputting explicitly all spanning trees, these algorithms are the best known in term of time and space complexities.

In this paper we introduce a new technique for outputting explicitly all spanning trees of complete undirected graphs, which improve the above time and space complexities. Our approach is based on the representation of spanning trees by Prüfer numbers. One of the classical theorems in enumeration is Cayley's theorem [6], which says that in a complete undirected graph with n vertices there are n^{n-2} distinct labeled trees. Prüfer provided a constructive proof of Cayley's theorem by establishing a one to one correspondence between such spanning trees and the set of all permutations of $n-2$ digits [7]. Prüfer numbers are an $n-2$ digit sequences, where the digits are n different numbers.

First, we construct a new algorithm to decode a Prüfer number into a spanning tree in a time of $O(n)$. The known decoding algorithm requires a time of $O(n \log n)$ with the aid of a heap (see [8, 9], for instance). In order to list all possible Prüfer numbers and each one must be represented in the list exactly once, we define what we call base-n. This base contains n digits (numbers): $0, 1, \cdots, n-1$. Similarly to the decimal base, we define an addition and order operators. Then we may see a Prüfer number not as a code but as a number of the base-n . Hence, to find all spanning trees, we simply increment in base-n 0 by 1 for $n^{n-2}-1$ times. Then each number corresponds to a Prüfer number which is decoded to output a unique spanning tree.

We show that our algorithm requires a time complexity of $O(nm)$ and space of $O(n)$. For outputting

explicitly all spanning trees in complete undirected graphs our algorithm is optimal.

In section 2 we describe Prüfer numbers and its relationship with spanning trees. In section 3 we introduce the definition of base-n. Section 4 is devoted to the description and the analysis of the algorithm for finding all spanning trees. Finally, we present some numerical results in section 6.

2. PRÜFER NUMBERS

Let G be a complete undirected graph with n vertices, and let d be a positive integer. In what follows we need the following definitions.

- A vertex v in G is called of degree d if it is a common vertex of d edges.
- A vertex of degree 1 is called a *leaf vertex*.
- A Prüfer number, P, is an $n-2$ digit sequence: $P = [p_0, p_1, \cdots, p_{n-3}]$, where the digits p_i, $0 \leq i \leq n-3$, are numbers in $\{0, 1, \cdots, n-1\}$.
- Let P be Prüfer number. The set of numbers in $\{0, 1, \cdots, n-1\}$ which are not digits part of P, is called child of P and denoted by R.

The relationship between Prüfer numbers and spanning trees are given by the following algorithms.

Algorithm 1 shows how to construct (or decode) a Prüfer number from a given spanning tree T. For the algorithm to be non trivial, the tree T should have at least two edges, $n \geq 3$.

Algorithm 1:

(1) Construct P by appending digits to the right; thus, P is constructed from left to right. Let i be the smallest labeled vertex of degree 1 in T, and let p be the vertex antecedent of i. Then we set p to the end of P.

(2) Remove the edge (i, p) from the tree T, and update T to $T \backslash \{(i, p)\}$.

(3) While the tree T has two or more edges return to step (1).

As an example on how this algorithm works, we consider the tree shown in figure 1. The smallest labeled vertex with degree 1, is the vertex numbered 3. We therefore select 1 as the first digit of P, $P = [1]$. We then remove the edge $(1, 3)$ from T, and vertex 1 becomes the smallest labeled vertex of degree 1. So the next digit of P is 6, $P = [1, 6]$. We continue the process until there are two vertices left, $\{2, 6\}$. Then we stop with $P = [1, 6, 0, 0, 2]$ is the Prüfer number corresponding to the tree in figure 1.

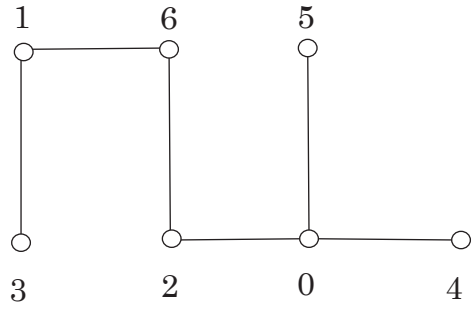

Figure 1: Graph of T

Conversely, it is also possible to construct (or decode) a unique spanning tree corresponding to a Prüfer number P, by using the following algorithm.

Algorithm 2:

(1) Let P be a given Prüfer number with $n-2$ digits, and let R be its child. The tree T is initialized to an empty set, $T = \emptyset$.

(2) Let i be the left-most digit in P, and let k be the smallest element of R. Add the edge (i, k) to the tree T. Then remove the left-most digit from P, and also remove k from R. If i does not occur anymore in what remains in P, put it into the set R.

(3) Repeat step (2) until no digits remain in P.

(4) Add the last edge, with the two remaining nodes in the set R, to the tree T.

To illustrate this algorithm, let us consider Prüfer number of the previous example, $P = [1, 6, 0, 0, 2]$. Then numbers in the set $\{0, 1, 2, 3, 4, 5, 6\}$ which are not digits part of P consist of the following set $R = \{3, 4, 5\}$. Let T be an empty set, $T = \emptyset$, then we add the edge $(1, 3)$ to T and remove 3 from R and 1 from P. The digit 1 is no longer in P, then 1 is added to the set R. We get $P = [6, 0, 0, 2]$ and $R = \{1, 4, 5\}$. Vertex 1 now is the smallest labeled element in R. Thus we add $(6, 1)$ to the tree T. Next, we remove 6 from P and 1 from R. The digit 6 is no longer in P, then it is added to the set R. We get $P = [0, 0, 2]$ and

$R = \{6, 4, 5\}$. Vertex 4 now is the smallest labeled element in R, then we add $(0, 4)$ to the tree T. Next, we remove 0 from P and 4 from R, the digit 0 is still in P. Hence we obtain $P = [0, 2]$ and $R = \{5, 6\}$. We repeat the process until no digits remain in P.

This decoding algorithm can be carried out in time of $O(n \log n)$ with the aid of a heap. In order to decrease the time complexity to be of $O(n)$, we define the following decoding algorithm.

Algorithm 3:

(1) Let P be a given Prüfer number with $n-2$ digits, and let R be its child. Elements in R are put in decreasing order from left to right. The tree T is initialized to an empty set, $T = \emptyset$.

(2) Let i be the left-most digit in P, and let k be the left-most element in R. Add the edge (i, k) to the tree T, and then remove the left-most digit from P. If i does not occur anymore in what remains in P, then replace k by i in R; otherwise delete k from R.

(3) Repeat step (2) until no digits remain in P.

(4) Add the last edge, with the two remaining nodes in the set R, to the tree T.

Remark 1:

(i) Let T be a tree and P_T its corresponding Prüfer number encoded by Algorithm 1. If P_T is decode by Algorithm 2 it will produce the same tree T, and conversely.

(ii) Let T be a tree and P_T its corresponding Prüfer number encoded by Algorithm 1. If P_T is decoded by Algorithm 3 it may not give the same tree T. However, in the algorithm of finding all spanning trees we only need a decoding algorithm.

Theorem 1 Let G be a complete undirected graph with n vertices and let P be an $n-2$ digits Prüfer number. Then Algorithm 3 constructs one and only one spanning tree corresponding to P. Moreover, for two different Prüfer numbers the algorithm produces two different spanning trees. Hence Algorithm 3 constructs a one to one correspondence between $n-2$ digits Prüfer numbers and spanning trees of the graph G. This algorithm requires a time complexity of $O(n)$.

Proof. It is easy to see that Algorithm 3 we constructs a unique tree with $n-1$ different edges, which spans the n vertices. Consequently T is a spanning tree.

Now, we show that for two different Prüfer numbers the algorithm constructs two different spanning trees. Let us consider two different Prüfer numbers $P_1 = [p_0^1, p_1^1, \cdots, p_{n-3}^1]$ and $P_2 = [p_0^2, p_1^2, \cdots, p_{n-3}^2]$, and let R_1 and R_2 be their corresponding children, respectively. Algorithm 3 decodes P_1 and P_2 to produce two spanning trees T_1 and T_2, respectively. We distinct the two following cases:

First case. Assume that R_1 and R_2 are different. Then there exists an element, vertex label, r in $(R_1 \cup R_2) \backslash (R_1 \cap R_2)$. Assume that $r \in R_1 \backslash R_2$, hence the vertex r is of degree 1 in T_1 and is at least of degree 2 in T_2. Consequently, trees T_1 and T_2 are different. The same argument can be applied if $r \in R_2 \backslash R_1$.

Second case. Assume that R_1 and R_2 are same. If p_0^1 and p_0^2 are different digits, then (p_0^1, r) and (p_0^2, r) consist of two different edges of T_1 and T_2, respectively, where r denotes the left-most digit in the original child $R_1 = R_2$. Since r is a label vertex of degree 1, then trees T_1 and T_2 are different. We now assume that p_0^1 and p_0^2 are equal. Let i be the smallest index such that $p_i^1 \neq p_i^2$. Then at the $(i-1)$th iteration of Algorithm 3, trees T_1 and T_2 have the same edges. In the ith iteration of Algorithm 3, we add to the tree T_1 the edge (p_i^1, r), and to the tree T_2 the edge (p_i^2, r), where is the left-most digit in R_1 and R_2. The element r is either equal to $p_{i-1}^1 = p_{i-1}^2$, if p_{i-1}^1 is no longer in the remain digits of P_1, or r is a vertex of degree 1. In both cases the edge (p_i^1, r) is an element of T_1 and not an element of T_2, also the edge (p_i^2, r) is an element of T_2 and not an element of T_1, which means that T_1 and T_2 are different trees. Hence, for a Prüfer number P, Algorithm 3 constructs one and only one spanning tree.

Now, we show that Algorithm 3 can be performed in time of $O(n)$. Indeed, let Q be an n vector such that $Q(i)$ stores the degree of the vertex labeled i. We note that if a vertex is of degree 1 it belongs to R, otherwise it belongs to P. The pseudo code for step (1) of Algorithm 3 is outlined as follows: At first, each component of the vector Q is initialized to 1, then

for $i = 1$ to $n-2$ do
begin
    $p := P(i)$;
    $Q(p) := Q(p) + 1$;
end

for $i = 1$ to n do
begin
  if $(Q(i) = 1)$ then
  begin
    $R(r) := i$; (r is initialized to 0)
    $r := r + 1$;
  end
end

Now it is clear that step (1) of Algorithm 3 is of $O(n)$. Steps (2) and (3) are also of $O(n)$. Hence Algorithm 3 is of $O(n)$. The proof is complete. ■

3. DEFINITION OF BASE-n

Let n be the number of vertices in a complete undirected graph G. In section 1 we have seen that there is a one to one correspondence between the set of all spanning trees in a complete undirected graph and all Prüfer numbers with $n-2$ digits. In this section we will give an algorithm for representing all Prüfer numbers with $n-2$ digits, and each Prüfer number is represented in the list exactly once. To do that, we define what we call base-n.

We introduce the definition of a base-n as an extension of the notion of the known numerical bases, base 10 (decimal), base 2 (binary), for instance. In base 10 we use the numerals $0, 1, \cdots, 9$ to represent all numbers. Each column is a power of 10; the first (right-most) column is used for ones $1s$, and the next for $10s$, and so on. With n columns we can represent numbers from 0 to $10^n - 1$.

For understanding, we may see a number P in base-n not as a number, but as a code for a number. Using the rules found in base 10, we can describe the base-n as follows:

(i) The numerals (digits) used in base-n are $0, 1, 2, \cdots, n-1$.

(ii) The columns are power of n: n^0, n^1, $n^2, \cdots$, and so on.

(iii) With r columns we can represent numbers from 0 to $n^r - 1$.

Then each number P in base-n can be written as: $P = p_{r-1}p_{r-2}\cdots p_0$, where r is the number of columns and p_i, $0 \le i \le r-1$, are decimal numbers of the set $\{0, 1, \cdots, n-1\}$. In order to distinct columns of a number in base-n, the columns are separated by a space or a comma. For example, the following are numbers in base-25:

$$P_1 = [20, 0, 1, 3, 5, 6] \quad \text{or} \quad P_1 = 20\ 0\ 1\ 3\ 5\ 6.$$

$$P_2 = [24, 7, 13] \quad \text{or} \quad P_2 = 24\ 7\ 13.$$

Let $P = p_{r-1}p_{r-2}\cdots p_0$ be a number in base-n, then P can be converted into a decimal number as follows:

$$(P)_{10} = \sum_{i=0}^{r-1} p_i n^i.$$

The addition operation in the base-n uses the same rule as in the decimal base. Let
$P = p_{r-1}p_{r-2}\cdots p_0$ and $Q = q_{r-1}q_{r-2}\cdots q_0$, be two numbers in base-n. The algorithm for calculating the base-n representation of the sum, $S = P + Q$, is given as follow:

Algorithm 4:
At the first step, we add (in base 10) p_0 and q_0, $s = p_0 + q_0$. Then s can be represented in base-n by,

$$(s)_n = c_0 n^1 + d_0 n^0.$$

Take $s_0 = d_0$, and curry c_0. In the second step, we calculate the sum $p_1 + q_1$ and add c_0,

$$s = p_1 + q_1 + c_0,$$

which can be represented in base-n by

$$(s)_n = c_1 n^1 + d_1 n^0.$$

Take $s_1 = d_1$ and curry c_1. We continue in this way, calculating for each i in the range $0 \le i \le r-1$, the digits

$$s = p_i + q_i + c_{i-1}, \quad (s)_n = c_i n^1 + d_i n^0.$$

Then we take $s_i = d_i$ and c_i will be carried to the next step. Finally we have $s_r = c_r$, and the representation of the sum, $S = P + Q$, in base-n is given by

$$S = s_r s_{r-1} \cdots s_o.$$

As an example, let $P_1 = [24, 22]$, $P_2 = [15, 10, 20]$ and $P_3 = [8]$ be numbers in base-25. Then we have $P_1 + P_3 = [1, 0, 5]$ and $P_2 + P_3 = [15, 11, 3]$.

Definition 1 Let $P = p_m p_{m-1} \cdots p_0$ and $Q = q_m q_{m-1} \cdots q_0$ be two numbers in base-n. Then we say:

- P and Q are equal "$P = Q$" if $p_i = q_i$, for every i, $0 \le i \le m$.

- P is greater than Q "$P > Q$" if there exists an index $j \leq m$ such that $p_i > q_i$ for any i, $j \leq i \leq m$.
- P is less than Q "$P < Q$" if $Q > P$.

Now, we can represent Prüfer numbers with $n-2$ digits as numbers in base-n with $n-2$ digits. Then to list all possible Prüfer numbers it suffices to increment in base$-n$ 0 by 1 for $n^{n-2}-1$ times. Then we get a list of all Prüfer numbers with $n-2$ digits where each Prüfer number is represented in the list exactly once. This list is expressed as follows:

$$P_0 < P_1 < \cdots < P_{m-2} < P_{m-1}, \quad P_{i+1} = P_i + 1, \tag{1}$$

where the sing + denotes the addition operator in base-n, $P_0 = [0, 0, \cdots, 0]$, $P_1 = [0, 0, \cdots, 1]$, $\cdots$, $P_{m-1} = [n-1, n-1, \cdots, n-1]$ and $m = n^{n-2}$.

4. MAIN ALGORITHM

In section 2 we have seen that there is a one to one correspondence between the set of all spanning trees in a complete undirected graph with n vertices and the set of all Prüfer numbers with $n-2$ digits. To represent all spanning trees it suffices to cross all Prüfer numbers with $n-2$ digits, as described in the previous section, and then decode each one to its unique corresponding spanning tree. A Prüfer number can be decoded by Algorithm 2 or Algorithm 3. In order to minimize the time complexity we will use our algorithm, Algorithm 3 (see section 2), to decode a Prüfer number into its unique corresponding spanning tree. Let G be completed undirected graph with n vertices and m spanning trees. The main algorithm for outputting explicitly all spanning trees of G is described in the following steps.

Algorithm Main:

(1) Initialize P to the value 0, $P = [0, 0, \cdots, 0]$.

(2) Decode P by Algorithm 3, then get its corresponding spanning tree.

(3) Set $P = P + 1$, where the plus sing "+" stands for the addition operator of the base$-n$.

(4) Decode P by Algorithm 3, then get its corresponding spanning tree.

(5) Iterate steps (3) and (4) $m-1$ times.

Remark 2: This algorithm is easy to parallelize. Indeed, since Prüfer numbers can be ordered in base-n, see relation (1), then we can partition the list of all Prüfer numbers as follows: $\mathcal{P}_1 = \{P_1, P_2, \cdots, P_{i_1}\}$, $\mathcal{P}_2 = \{P_{i_1+1}, P_{i_1+2}, \cdots, P_{i_2}\}, \cdots, \mathcal{P}_k = \{P_{i_{k-1}+1}, P_{i_{k-1}+2}, \cdots, P_{i_k}\}$, where k is the number of partitions and $i_k = m-1$. Each partition $\mathcal{P}_i$ corresponds to a task to be sent to a processor or to a computer, which will be processed independently from the others.

Theorem 2 Let G be a complete undirected graph with n vertices. Then Algorithm Main outputs all spanning trees explicitly in a time of $O(mn)$ and space of $O(n)$, where $m = n^{n-2}$ is the number of all possible spanning trees in the graph G.

Proof. It is clear that the time complexity of step (3) in Algorithm Main is of $O(n)$ (see Algorithm 4). Theorem 1 shows that the time complexity of step (4) is of $O(n)$. Since steps (3) and (4) are iterated $m-1$ times, then Algorithm Main requires a time of $O(mn)$. It is easy to see that steps (1) through (4) require a space memory of $O(n)$. Hence the proof is complete. ■

5. COMPUTATIONAL EXPERIMENT

Experiments on Algorithm Main, were conducted on undirected complete graphs with number of vertices, n, from 8 through 11. The run time of the program for outputting explicitly all possible spanning trees of the graph, in seconds, is denoted by $T(n)$. The number of all spanning trees in the graph is denoted by m, $m = n^{n-2}$. This algorithm was coded in C++ programming language and implemented on a computer with a CPU Celeron 1.7GHz. The following table illustrates the computational results:

n	8	9	10	11
$T(n)$	0.06	1.23	28.22	672.41
$\frac{T}{nm}10^8$	2.86	2.85	2.82	2.59

Table 1. Computation time.

We remark that the ratio $T(n)/mn$ decreases while n increases. Hence the time $T(n)$ is bounded by nm as it was shown in Theorem 2, i.e., there exists a positive constant k such that $T(n) \leq kmn$. In this example we may take $k = 2.86 \times 10^{-8}$ for graphs with number of vertices greater or equal to 8.

6. CONCLUSION

This paper presents a new algorithm "Algorithm Main" for outputting explicitly all spanning trees in complete undirected graphs. The algorithm is based on the representation of spanning trees by Prüfer numbers. To list all possible Prüfer number with $n-2$ digits we define what we call base-n. Then each Prüfer number corresponds to a unique spanning tree. The algorithm requires a time of $O(nm)$ and space of $O(n)$. For finding all spanning trees explicitly of complete undirected graphs, the best known algorithm requires a time of $O(e+n+nm)$ and space of $O(e+n)$, where the parameters n, e, m denote, respectively, number of nodes, number of edges and number of spanning trees in the graph. The analysis of the algorithm and numerical examples are given.

References

[1] H.N. Gabow, Two algorithms for generating weighted spanning trees in order, SIAM Journal on Computing 6 (1977) 139-150.

[2] H.N. Gabow, E.W. Myers, Finding all spanning trees of directed and undirected graphs, SIAM Journal on Computing 7 (3)(1978) 280-287.

[3] S. Kapoor, H. Ramesh, Algorithms for enumerating all spanning trees of undirected and weighted graphs, SIAM Journal on Computing 24 (2) (1995) 247-265.

[4] T. Matsui, A flexible algorithm for generating all the spanning trees in undirected graphs, Algorithmica 18 (1997) 530-544.

[5] G.J. Minty A simple algorithm for listing all spanging trees of a graph, IEEE Trans. Circuit Theory, CT-12 (1965) 120-125.

[6] A. Cayley, A theorem on trees, Quartery Journal of Mathematics 23 (1889) 376-378.

[7] H. Prüfer Neuer beweis eines satzes uber permutationen, Arch. Math. Phys. 27 (1918) 742-744.

[8] C.C. Palmer, A. Kershenbaum, Representing trees in gemetic algorithm, IMB T.J. Watson Research Center.

[9] Rothlauf F, Goldberg D-E. Pruefernumbers and genetic algorithms: A lesson how the low locality of an encoding can harm the performance of GAs. Proceedings of PPSN VI, LNCS 1917, 395-404, Springer Verlag, Berlin 2000.

[10] R.E. Tarjan, R.C. Read, Bounds on backtrack algorithms for listing cycles, paths and spanning trees, Networks, 5 (1975) 237-252.