East Asian Math. J. **21** (2005), No. 2, pp. 249–260

CAS IMPLEMENTATION OF RECURSIVE STRUCTURE IN A SPANNING TREE

KEEHONG SONG

ABSTRACT. Experimentation using computer plays an important part in education and research in graph theory. The purpose of this paper is to develop the CAS techniques for the hands-on approach in graph theory specifically on the topic of constructing the spanning tree. This paper discusses the advantages of CAS as the software system for doing graph theory and introduces the software solutions integrating multimedia user interface developed by the author, which extend the functionality of the existing CAS-based graph theory software package.

1. Introduction

Since Cayley [1] proved that the number of spanning tree of n vertices T_n is n^{n-2}, there have been several alternative derivations of the formula. As is often the case, the counting algorithm of the spanning tree for a given number of vertices turns out to be a constructive algorithm for generating the spanning tree. This paper examines the various counting algorithms which are conducive to the implementation of the recursive structure in a spanning tree. For the purpose, the critical part of the programming code and the relevant technology involving Mathematica [11], the standard Computer Algebra System (CAS), examples are presented in this paper.

Received November 15, 2005.
2000 Mathematics Subject Classification: 97U70, 94C15.
Key words and phrases: graph theory, spanning tree, computer algebra system.

There have been significant amount of efforts in developing the computer software packages dedicated for graph theory. However, doing graph theory using CAS can be a different experience in many aspects as it provides a large set of symbolic capabilities.

This paper introduces two significant ways of harnessing the CAS kernel in experimenting with the ideas in graph theory. One is to use the desktop application with multimedia user interface and the other to build the web application with the improved user interactivity by connecting the kernel with the COM object, the Microsoft technology to link the different applications running on the Windows platform.

2. Data Structure of a Tree

To determine the right representation of a given tree structure is essential element in the computer implementation of the graph theoretic problem solving. Overall there are two models of representing the graph structure. One is the list structure which consists mainly of the left and right parentheses, ending up looking like a LISP programming. The other one is the object model which is commonly used in the graph theory packages, a notable system being Combinatorica [7], a Mathematica standard add-on package, where the data structure consists of head, vertices, and edge to constitute a graph object, `Graph[edges,vertices]`.

List structure

Typical representation of a data structure of a tree includes a set of list resembling the codes of the programming language, LISP, in terms of both appearance and functionality. For example, the set of parenthesis such as `{{b},{a},{{{e},{d},{f}},{c},{g}}}` represents the graph shown in Figure 1.

First and foremost, the advantage of representing graph using the list structure is the efficiency and simplicity in processing a certain type of graph algorithm. As an example, a tree represented in that structure would provide the advantage in the speed of execution in the implementation of Huffman code as it uses the recursion driven

by way of pattern matching. For another example, the generation of the entire set of the binary tree would be easily accomplished using the relevant recurrence relation

$$b_{n+1} = b_0 b_n + b_1 b_{n-1} + b_2 b_{n-2} + \cdots + b_{n-1} b_1 + b_n b_0.$$

The CAS translation of the recurrence relation represents exactly the flow of logic implied therein as evident in the following code segment.[1]

```
Table[Distribute[{catalanTree[Take[m,i]],catalanTree[
Take[m,{i+1,Length[m]}]]},List],{i,0,Length[m]}],{2}]]
```

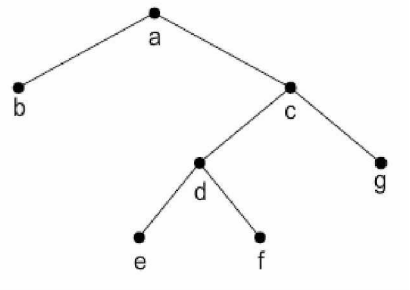

Figure 1.

Object model

This is the data structure commonly used by the graph theory software packages such as Combinatorica. Compared with the list structured, this type of data structure would be suitable for the larger system as it is more flexible in handling the more complex set of information. For the case of Combinatorica, it starts with the head, Graph , followed by the edges and the vertices information to form the graph object, `Graph[edges,vertices]`.

3. Algorithms for Constructing a Spanning Tree

There are several different methods in counting the number of the spanning tree with n vertices. Most common and simple way would

[1]The file containing the programming codes for this paper is downloadable from: http://home.pusan.ac.kr/~math/research/spanning_tree.zip

be the one using the idea of Pruefer code. When experimenting with the recursive structure of the spanning tree, the use of CAS proves to be effective, as each detail of the expression of the recurrence equation has its own natural interpretation.

Algorithm 1: Using Pruefer code

As the standard way of constructing spanning tree, the idea is based on the fact that there is one-to-one correspondence between the spanning tree with n vertices and the string with alphabet of size $n-2$. Using this algorithm, it would be straightforward to construct and display all the spanning tree with 5 vertices(Figure 2).

```
graf=Map[CodeToLabeledTree, Strings[{1,2,3,4,5},3]];
pics=Map[ShowGraph, graf];
Show[GraphicsArray[Partition[pics,25]]]
```

Figure 2.

Algorithm 2: Using the recurrence equation

$T_n = \sum_{m>0} \frac{1}{m!} \sum_{k_1+\cdots+k_m=n-1} \binom{n-1}{k_1,\cdots,k_m} k_1 \cdots k_m T_{k_1} \cdots T_{k_m}$

A spanning tree with n vertices can be made simply by connecting all the spanning 4 trees, with the sum of each tree being n-1. Thus we need to find the integer solution $k_1, k_2, \cdots, k_m$ to the equation $k_1 + k_2 + \cdots + k_m = n - 1$. For that, we have several different options in getting the solution. First, we can see the equation as the integer partition and use the recursion, $p_{n,k} = p_{n-k,k} + p_{n,k-1}$, where $p_{n,k}$ is the number of partition of the positive integer, n, into parts, each of which does not exceed k. For the case where the number of vertices is 4, we need the positive integer solution for the equation, $k_1 + k_2 + \cdots + k_m = 3$, It is quite simple to obtain the solution set for the equation using Mathematica.

```
Map[Permutations,Partitions[3,3]]//Flatten[#,1]&
{{3},{2,1},{1,2},{1,1,1}}
```

Another way of getting at the solution of the problem is to see it as a partition of the set with each subset contains at least one element. That is exactly the idea in the Stirling number of the second kind $\{{n \atop k}\}$, satisfying the recurrence equation, $\{{n \atop k}\} = \{{n-1 \atop k-1}\} + k\{{n-1 \atop k}\}$. The essential part of the coding for the recurrence relation would be

```
Map[Prepend[#,{First[l]}]&,stirling2nd[Rest[l],k-1]]
```

Alternatively, instead of looking at the cases of the positive integer solution, we can look at the problem as a whole, then the question becomes to obtain the Bell number,B_n, satisfying the equation, $B_n = \sum_{i=0}^{n-1} \binom{n-1}{i} B_i$.

The essential part of the coding would be implemented in the following manner:

```
Table[Map[Map[Prepend[#,Flatten@{First[l],s}]&, Bell[
Complement[r,s=#]]]&,KSubsets[r,i]],{i,0,n-1}]]
```

Using the code, we can obtain the identical result to the previous one.

```
Union[Map[Map[Length,#]&, Bell[{1,2,3}]]]
{{3}, {1, 2}, {2, 1}, {1,1,1}}
```

Alternatively, we can also provide another positive integer solution to the equation by way of using the idea of subsets. Let's consider the case of choosing $k-1$ elements $x_1, x_2, \cdots, x_{k-1}$, where $0 < x_1 < x_2 < \cdots < x_{k-1} < n+k$ from a set of positive integer from 1 to $n+k-1$. Also define $d_k, d_{k-1}, \cdots, d_1$ to be the difference between the adjacent integer, $n+k-x_{k-1}, x_{k-1}-x_{k-2}, \cdots, x_2-x_1, x_1-0$. Then we have $d_1+d_2+\cdots+d_k = n+k$, $d_i \geq 1$, $1 \leq i \leq k$. Just for the solution to the equation, the *Combinatorica* package command, Compositions, gives the desired result, which can be coded as shown in the following.

```
Map[(#[[2]]-#[[1]]-1)&,Partition[Join
[{0},#,{n+k}],2,1]]&
s=Table[Compositions[3,i],{i,1,3}]//Flatten[#,1]&
```

```
Select[s,!MemberQ[#,0]&]
{{3}, {1, 2}, {2, 1}, {1, 1, 1}}
```

Algorithm 3: Using the recurrence equation

$T_n = \sum_{k=1}^{n-1} k\binom{n-2}{k-1} T_k T_{n-k}$

A spanning tree with n vertices has $n-1$ edges and the process of deleting an edge in sequence yields two separate spanning trees. Thus we have,

$$(n-1)T_n = \sum_{k=1}^{n-1} k\binom{n-1}{k-1} k(n-k) T_k T_{n-k}.$$

Here the interpretation of the expression, $\sum_{k=1}^{n-1}\binom{n-1}{k-1}$, would be the bipartite partition of the given spanning tree. Now we just give a combinatorial argument in proving the equality, $\sum_{k=1}^{n-1}\binom{n-1}{k-1} = \binom{n}{2}$. We can verify the equality experimentally by coding each expression separately.

```
Map[{Complement[{1,2,3,4,5},#],If[MemberQ[#,1],Join[{1},
#], #]}&,Flatten[Table[KSubsets[2,3,4,5,k],{k,1,4}],1]]
stirling2nd[{1,2,3,4,5},2]
{{{1, 2, 3, 4}, {5}}, {{2, 3, 4}, {1, 5}}, {{1, 3, 4}, {2, 5}},
{{3, 4}, {1, 2, 5}}, {{1, 2, 4}, {3, 5}}, {{2, 4}, {1, 3, 5}},
{{1, 4}, {2, 3, 5}}, {{4}, {1, 2, 3, 5}}, {{1, 2, 3}, {4, 5}},
{{2, 3}, {1, 4, 5}}, {{1, 3}, {2, 4, 5}}, {{3}, {1, 2, 4, 5}},
{{1, 2}, {3, 4, 5}}, {{2}, {1, 3, 4, 5}}, {{1}, {2, 3, 4, 5}}}
```

Now as the recursion suggests, each partition breaks into bipartite components until the division reaches the trivial case of a spanning tree. We can create the growing spanning tree in the process of backtracking. Each process is simply the action of graph union and edge additions.

Algorithm 4: Using the degree sequence of the tree

Given the spanning tree with n vertices and the degree of each being $d_1, d_2, \cdots, d_n$ $(d_1 \geq d_2 \geq \cdots \geq d_n = 1)$, the recursive equation for the number of spanning tree becomes

$T(n; d_1, \cdots, d_n) = \sum_{i=1}^{n-1} T(n-1; d_1, \cdots, d_i - 1, \cdots, d_{n-1})$

as the spanning tree with n vertices can be constructed by simply connecting v_n with the trees with $n-1$ vertices [6].

Using the recursion, Cayleys result can be deduced using induction. Its easy to verify that

$$T(n; d_1, \cdots, d_n) = \binom{n-2}{d_1-1, \cdots, d_n-1}.$$

For the case n = 3, it is trivial. Now

$(x_1 + x_2 + \cdots + x_k)^n = (x_1 + x_2 + \cdots + x_k)^{n-1}(x_1 + x_2 + \cdots + x_k)$.
The equation leads to the equation

$$(x_1 + x_2 + \cdots + x_k)^n = \sum \binom{n}{r_1, \cdots, r_k} x_1^{r_1} x_2^{r_2} \cdots x_k^{r_k}.$$

This yields the relationship

$$\binom{n}{r_1, \cdots, r_k} = \sum_{i=1}^{k} \binom{n-1}{r_1, \cdots, r_i - 1, \cdots, r_k}.$$

Hence

$$\begin{aligned} T(n+1; d_1, \cdots, d_{n+1}) &= \sum_{i=1}^{n} T(n; d_1, \cdots, d_i - 1, \cdots, d_{n+1}) \\ &= \sum_{i=1}^{n} \binom{n-2}{d_1-1, \cdots, d_i-2, \cdots, d_{n+1}-1} \\ &= \binom{n-1}{d_1-1, \cdots, d_i-1, \cdots, d_{n+1}-1}. \end{aligned}$$

Thus we have

$$\sum_{(d_1-1)+\cdots+(d_n-1)=n-2} T(n; d_1, \cdots, d_n) = n^{n-2}.$$

Now the counting formula for the number of spanning tree involving multinomial suggests the constructive algorithm to obtain the degree sequence from which the spanning trees are made. The solution to the following equation is going to be the degree sequence $d_1, d_2, \cdots, d_n$.

$$(d_1 - 1) + (d_2 - 1) + \cdots + (d_n - 1) = n - 2.$$

Obviously, the arbitrary sequence of the vertex degrees will not make a graph. So we need an algorithm to test whether a given sequence is eligible for the spanning tree construction, which calls for a testing device called *graphic*.

Erdos and Gallai [2] proved that a degree of sequence $\{d_1, \cdots, d_n\}$ is *graphic* if and only if the sequence satisfies the property

$$\sum_{i=1}^{r} d_i \leq r(r-1) + \sum_{i=r+1}^{n} \min(r, d_i).$$

This condition could have been used as a tester, however, it would be more computationally efficient to use the method devised by Hakimi [5], who proved another characterization of graphic sequences.

That is simply the fact that a degree sequence with $n \geq 3$ and $d_1 \geq 1$ is *graphic* if and only if the following sequence is *graphic*.

$$\{d_2 - 1, d_3 - 1, \cdots, d_{d_1+1} - 1, d_{d_1+2}, \cdots, d_p\}.$$

The Mathematica implementation for the tester in essence would resemble the code

```
GraphicQ[Join[Take[sorted,{2,m+1}]-1,Drop[sorted,m+1]]]
```

The *Combinatorica* package command, RealizeDegreeSequence creates the graph structure through the filter, that is, the idea of *graphic*, to test whether a given degree sequence makes a spanning tree.

For the case where the number of vertices is 5, we need to find the integer solution for the equation $(d_1 - 1) + (d_2 - 1) + \cdots + (d_n - 1) = 5 - 2$ as we have done earlier in this paper.

```
s=Compositions[3,5]
```

For demonstration purpose, take the following case as an example.

$\{d_1 - 1, d_2 - 1, \cdots, d_n - 1\} = \{0, 0, 1, 1, 1\}$

```
ShowGraph[RealizeDegreeSequence[{0,0,1,1,1}+1]]
```

In the final stage of construction, the degree sequence method is bound to yield the duplicate tree whose vertices are to be rearranged to differential the different tree of the same shape [4].

Algorithm 5: Using matrix-tree

Given a loop-free graph G with edge matrix $A = (a_{ij})$, $a_{ij} = 1$ if v_i is the head of e_j, $a_{ij} = -1$ if v_i is the tail of e_j, and $a_{ij} = 0$ otherwise. Then the number of spanning tree of G is $\det(A_0 A_0^T)$, where A_0 is the matrix with a row being deleted [3].

```
edg={{1,2},{1,4},{2,4},{3,5},{3,4},{4,5}};
directedPair=If[Random[]>0.5,Reverse[#],#]& /@edg
dg=FromOrderedPairs[directedPair];
```

The diagonal elements of the matrix $A_0 A_0^T$, $(2, 2, 2, 4)$ correspond to the degree sequence $(1, 2, 3, 4)$. So we know the number of spanning tree of the graph G is equal to $\det(A_0 A_0^T)$=9.

We need to find the all the $(n-1) \times (n-1)$ submatrix of the matrix A_0, and then the nonzero elements in the set of the determinants of

the matrix found in the previous computation corresponds to the spanning tree.

```
id=KSubsets[Range[6],4]
Map[Transpose[Transpose[a0][[#]]]&,id]
Map[Det,%]
{0,0,0,-1,-1,-1,-1,-1,-1,0,-1,-1,-1,0,0}
```

Now with the edge matrices determined, it is straightforward to construct the corresponding spanning trees.

4. Technology on Graph Theory

The computers play an important role in teaching and research in graph theory. Compared with the common programming language such as C/C++ or Java, CAS has its unique strength in delivering the service in response to the need from the education and research community. The *Combinatorica* package is the kind for the purpose. Although it is considered to be one of the most powerful systems today, the package leaves something to be desired in several aspects. One apparent weakness would be its lack of interactivity in the graphical output which comes in postscript format. The *Graph Editor*, developed by the author, is one example that harnesses the technology to enhance the 9 existing graph theory software packages.

The multimedia-enhanced software tools come in two different modes: desktop application and web application. Each one is fairly sophisticated in functionality as the graph editor software packages, but has its own strength and weakness. Briefly, the desktop application is a kind of wrapper for *Combinatorica* package as it receives the result from the package and converts its output into the Flash file format, currently the de factor web animation standard. That way, the software enhancer enables the rubber-band effect mainly for experimenting with graph isomorphism.

Desktop application

It has the definite advantages of the standard desktop applications sophisticated userinterface. Not only does it a routine I/O

operations, it saves the graphical output in the memory for later use along with a powerful graph editing capabilities [10] (Figure 3).

Equally important, the desktop application for graph theory converts the huge set of graph theory functionality into a set of menus to choose from, making it accessible to the naive users.

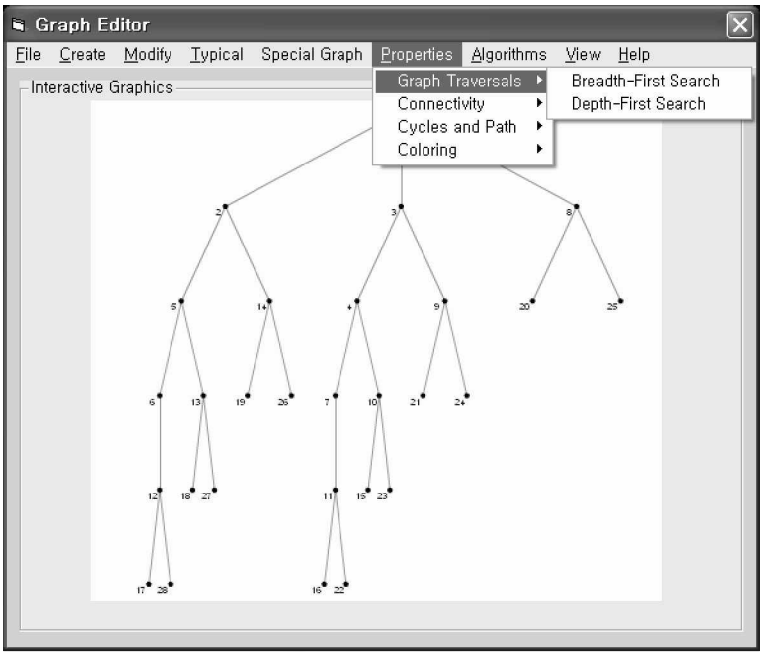

Figure 3.

Web application

Although coming in lightweight in file size, it covers the same range of the graphics operation commands in the *Combinatorica* package. The fact that it gives the advanced user the more degree of freedom can be view as weakness, since it is not as accessible to the naive users as the desktop counterpart. Again, it produces the same Flash file format output corresponding to the graph output produced by *Combinatorica* package so that the experienced user can experiment with the graph theory operation with flexibility. As a side note, when it comes to delivering the service of graph editing over the web, both type are technically feasible with the recent techno-

logical advancement such as .NET technology, Flash Remoting, and others [3][8][9] (Figure 4).

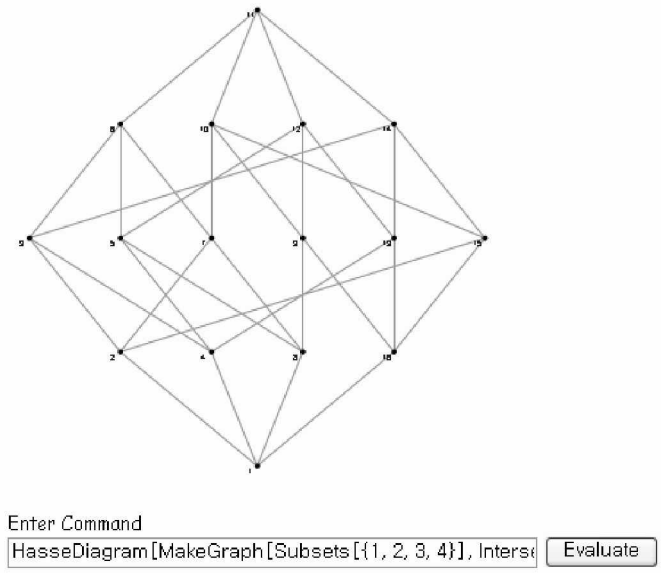

Figure 4.

5. Summary

Since Cayley [1] first proved that the number of spanning trees for the given n vertices is n^{n-2}, a number of alternative proofs have been presented. Initially these proofs have been for counting purposes, but in this paper these counting strategies were extended to the spanning tree construction algorithms. In doing so, due to 11 computational complexity, the use of computer is a must and, especially, the use of CAS(Computer Algebra System), by virtue of its symbolic capabilities and versatility, serves the purpose effectively. Even then the power of CAS becomes more accessible to the users in general with the kernel connectivity with multimedia technology such as Flash.

REFERENCES

[1] A. Cayley, *A theorem on trees*, Quart, J. Math. **23** (1889), 376–378.
[2] P. Erdos and T. Gallai, *Graphs with prescribed degrees of vertices (Hungarian)*, Mat. Lapok **11** (1960), 264–274.
[3] S. Even, *Graph Algorithm*, Computer Science Press, 1979.
[4] R. Gould, *Graph Theory*, The Benjamin-Cummings, 1988.
[5] S. Hakimi, *On the realizability of a set of integers as degrees of the vertices of a graph*, SIAM, J. Appl. Math. **10** (1962), 496–506.
[6] L. Lovász, *Combinatorical Problems and Exercises, 2nd ed.*, North-Holland, 1993.
[7] S. Skiena, *Implementing Discrete Mathematics: Combinatorics and Graph Theory with Mathematica*, Addison-Wesley, 1990.
[8] K. H. Song, *Flash-Enabled User Interface for CAS*, Internet Accessible Mathematical Computation, a Workshop at ISSAC 2003, Drexel University, Philadelphia, PA, USA.
[9] K. H. Song, *Developing Computational Web Animation - using Flash and .NET Technology*, Internet Accessible Mathematical Computation, a Workshop at ISSAC 2004, University of Cantabria, Santander, Spain.
[10] K. H. Song, *Packaging Mathematica Packages*, Wolfram Technology Conference 2005, Champaign, Illinois, USA.
[11] S. Wolfram, *Mathematica: A System for Doing Mathematics by Computer*, Addison-Wesley, 2004.

Department of Mathematics Education
Pusan National University
Pusan 609-735, Korea
E-mail: khsong@pusan.ac.kr