'95 대한전기학회 하계학술대회 논문집(1995. 7. 20~22)

링크구조를 사용한 유한요소법의 계행렬 압축 기법

°정 래혁*, 이 복용**, 정 해덕†, 이 기식*
*단국대학교, **대유공업전문대학, †목포대학교

A Compression Method of The System Matrix for The Finite Element Method using Linked List

°Lae-Hyuk Jeong*, Bok-Yong Lee**, Hae-Duk Jung†, Ki-Sik Lee*
*Dankook University,**DaeYeu Technical College, †Mokpo Nat'l University

Abstract - This paper presents compression algorithm of a system matrix for electromagnetic analysis by the finite element method. Generally the solution of the finite element analysis is the more accurate the more number of nodes. The memory of a computer limit to number of nodes. Therefore it is needed the technique of compress the system matrix. This algorithm is useful to handle non-zero-terms that can be generated during the application of boundary condition.

1. 서 론

유한요소법은 해석영역을 미소영역(요소)으로 세분하고 세분된 각 미소영역의 요소행렬로부터 전 해석영역의 계행렬을 구성한 후, 이를 풀이 결과를 얻는 수치 해석적 방법의 하나이다. 이러한 유한요소법의 해는 일반적으로 절점의 수를 증가시킬수록 정확도가 증가한다고 알려져 있다[1]. 그러나 절점수는 컴퓨터의 메모리에 의하여 제한되므로, 메모리의 효율적 이용 방법이 필요하다.

유한요소법의 계행렬은 대부분 영의 값을 갖는 희소행렬(sparse matrix)이고 대각요소를 중심으로 대칭의 형태를 가지므로 본 연구에서는 링크구조(linked list)를 이용하여 계행렬 비영요소의 약 1/2을 이용하는 효율적인 알고리즘을 제시하였다. 또한 이 알고리즘은 계산 중에 발생할 수 있는 새로운 비영요소의 처리가 용이하므로 다양한 문제의 해결에 사용될 수 있다.

2. 압축기법

본 장에서는 계행렬을 압축하는 알고리즘과 계산 중에 발생하는 새로운 비영요소의 처리방법에 대하여 기술하였다. 알고리즘은 C 언어를 사용하여 C 프로그램으로 표현하였다.

2.1 링크구조

링크구조란 컴퓨터의 기억공간에서 서로 연결된 블럭(block)을 말한다. 각각의 블럭은 다른 블럭을 가리키는 요소를 적어도 하나 이상 가지고 있다. 링크구조를 구성하는 기억장소 내의 블럭은 구조체(structure)로 만들어지며, 주어진 링크구조에 있어서 모두 똑같은 형과 크기(size)를 가진다.

그림 1과 같이 링크구조의 블럭은 가장 마지막 블럭을 제외하고 그 후속 블럭을 가리키는 포인터를 포함하고 맨 처음의 것을 제외하고 그 선행 블럭에 의해 지시된다. 링크구조의 가장 마지막 블럭은 null을 가리키며 맨 처음 블럭은 처음을 표시하는 포인터에 의해 지시된다[2].

링크구조의 장점은 기억장소가 고정되어 있는 배열을 이용한 기억장소의 할당과는 달리 프로그램을 수행하는 동안 요구에 따라 어떤 형태로든지 기억장소의 할당이 가능하다는 점이다. 메모리 할당시 배열은 그 크기만큼 램(RAM)에서 연속된 번지가 필요하여 램의 빈 공간이 배열의 크기보다 큰 경우에도 번지가 연속되지 않을 경우 기억장소의 할당이 불가능하게 된다. 그러나 링크구조는 각 블럭별로 메모리를 할당하므로 램의 번지와는 관계없이 크기만 맞으면 언제나 할당이 가능하다. 링크구조는 새로운 블럭의 추가, 삽입과 다른 링크구조와의 결합으로 길어질 수 있으며, 블럭의 삭제로 줄어들 수도 있다. 말하자면 링크구조는 계획된 제어에 의해 동적으로 크기와 모양이 변할 수 있는 기억장소 이다[3].

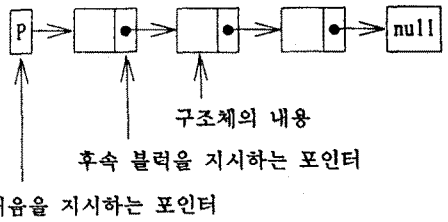


그림 1 링크구조

2.2 링크구조를 이용한 계행렬 압축

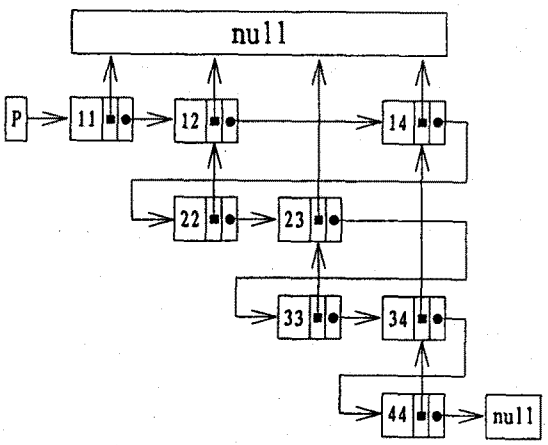
이미 설명한 바와 같이 계행렬은 대부분의 값이 영인 희소행렬이고 대각요소를 중심으로 대칭의 형태를 가지고 있다. 영의 값을 갖는 부분은 연립방정식의 해에 영향을 미치지 못하므로 컴퓨터를 이용한 연립방정식의 풀이에는 메모리의 낭비만 가져오게 된다. 이런 낭비를 없애기 위하여 실제 컴퓨터의 메모리에는 계산에 필요한 비영요소만 기억시키는 계행렬 압축기법이 필요한 것이다. 그림 2의 (a)는 압축하지 않은 상태의 행렬이고 (b)는 링크구조를 이용하여 압축한 행렬이다.

그림 2의 (b)에 나타나있는 링크구조의 각 블럭들을 본 논문에서는 아래와 같은 구조체로 구성하였다.

```
typedef struct System SYM;
struct System {
    double s;
    long int i, j;
    SYM *Next, *Before;
};
```

$$\begin{bmatrix} S11 & S12 & 0 & S14 \\ S21 & S22 & S23 & 0 \\ 0 & S32 & S33 & S34 \\ S41 & 0 & S43 & S44 \end{bmatrix}$$

(a) 압축하지 않은 계행렬



(b) 링크구조를 이용하여 압축한 계행렬

그림 2 계행렬의 압축

구조체 System은 5개의 변수로 구성된다. 각각의 역할을 알아보면 다음과 같다.

s : 계행렬의 값을 가진 변수.
i : 행렬 중에서 행의 위치를 가진 변수.
j : 행렬 중에서 열의 위치를 가진 변수.
*Next : 1행 1열로 시작하여 마지막 행의 마지막 열 까지 각각 자기 오른쪽의 비영요소를 지시하는 포인터.
그 행의 마지막 요소일 때 다음 행의 대각요소를 지시하고 마지막 열의 마지막 요소일 때 null을 지시 한다.
*Before : 각 대각요소에서 시작하여 같은 열의 위로 올라가며 비영요소를 지시하는 포인터. 더이상 없을 경우 null을 지시한다.
구조체 System의 크기는 표 1과 같다.

표 1 구조체 System의 크기

	byte	개수	합계
long int	4	2	8
double	8	1	8
pointer	4	2	8
합계			24

그림 2의 (b)에서 보듯이 압축한 계행렬은 대각요소를 중심으로 위쪽의 요소로만 구성되어 있다. 압축하지 않은 계행렬의 한 행은 압축한 행렬에서 대각요소를 중심으로 *Before를 따라 *Before가 null까지 가고 *Next를 따라 *Next가 다음 행의 대각요소까지 가면 완성된다.

압축된 계행렬의 블럭수는 절점의 수가 n개 이고 압축하지 않은 계행렬의 비영요소가 s개 일 때 (s+n)/2와 같다. 이때 계행렬의 크기는 (s+n)×12 [byte]가 된다. 기존의 비영요소 전체를 기억하는 알고리즘은 비영요소의 개수를 계산하여 기억장소를 할당하지 않고 일반적으로 절점수의 8배를 비영요소의 개수로 가정하고 기억장소를 할당한다. 이렇게 하는 이유는 비영요소의 개수를 계산하는 시간과 알고리즘이 복잡해지는 것을 피하기 위해서 이다. 이때 계행렬의 크기는 n×96 [byte]이다.

2.3 계행렬 조립

요소행렬의 값을 압축한 계행렬에 대입하는 것은 요소행렬의 값이 들어갈 위치를 계행렬에서 찾는 일이 가장 큰 비중을 차지한다. 본 논문에서 제시한 알고리즘은 구조체의 두 포인터를 이용하여 위치를 찾는다. 블럭구조의 계행렬에서 i행의 j번째 요소를 찾을 때 i번째 대각요소에서 i가 j보다 큰 경우 Before를 따라가며 찾고 반대인 경우는 Next를 따라가며 위치를 찾는다.

2.4 새로운 비영요소의 처리방법

기존의 배열을 이용한 압축방법은 배열의 크기를 바꿀 수 없는 단점 때문에 주기경계조건이나 등 포텐셜 경계조건을 도입한 문제에서 새롭게 생겨나는 비영요소의 처리가 어려웠다. 그러나 링크구조를 이용하면 위의 두 가지 경계조건을 가진 문제도 용이하게 계행렬을 압축할 수 있다. 주기경계조건의 적용 법은 다음과 같다.

$$\begin{bmatrix} S1 & S2 & S3 & S4 & S5 \\ S6 & S7 & S8 & 0 & S9 \\ S10 & S11 & S12 & S13 & S14 \\ S15 & 0 & S16 & S17 & S18 \\ S19 & S20 & S21 & S22 & S23 \end{bmatrix} \begin{bmatrix} A1 \\ \boxed{A2} \\ A3 \\ \boxed{A4} \\ A5 \end{bmatrix} = \begin{bmatrix} F1 \\ F2 \\ F3 \\ F4 \\ F5 \end{bmatrix}$$

(a) 경계조건 적용 전의 계행렬

$$\begin{bmatrix} S1 & S2+S4 & S3 & 0 & S5 \\ S6+S15 & S7+S17-1 & S8+S16 & 1 & S9+S18 \\ S10 & S11+S13 & S12 & 0 & S14 \\ 0 & 1 & 0 & -1 & 0 \\ S1 & S20+S22 & S21 & 0 & S23 \end{bmatrix} \begin{bmatrix} A1 \\ \boxed{A2} \\ A3 \\ \boxed{A4} \\ A5 \end{bmatrix} = \begin{bmatrix} F1 \\ F2+F4 \\ F3 \\ 0 \\ F5 \end{bmatrix}$$

(b) 경계조건 적용 후의 계행렬

그림 3 주기경계조건 적용법

구조물의 배열이 주기적으로 반복되고 장(field)을 형성하는 전원(source) 역시 주기적으로 반복되면 구조물의 한 주기에 대해서만 해석하므로써 전체를 대신할 수 있다. 이런 해석대상에는 주기경계조건을 적용해야 한다. 그림 3의 (a)를 주기성이 있는 대상의 계행렬이라 하고, A2와 A3가 각각 반대편의 경계에 위치한 절점의 포텐셜이라 하자. 그러면 이 두 포텐셜은 같은 값을 가지게 된다. 그림 3의 (b)는 위의 조건을 (a)의 계행렬에 도입한 것이다. 이때 제 2행의 요소 또는 제 2열의 요소가 영의 값을 하나라도 가질 때 새로운 비영요소가 발생 한다. 이런 비영요소는 링크구조로 되어있는 계행렬에 삽입해야 한다.

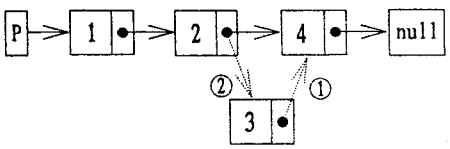


그림 4 비영요소의 삽입

그림 4는 링크구조에 새로운 블럭을 삽입하는 방법을 나타낸 것이다. 그림에서 2번 블럭은 4번 블럭을 지시하고 있다. 2번과 4번 사이에 3번을 삽입하려면 우선 2번 블럭이 지시하는 블럭을 3번이 지시하게 한 후 2번 블럭으로 3번을 지시하게 하면 된다. 같은 방법을 계 행렬에 적용하면 우선 Next를 따라가며 i와 j를 비교하여 삽입할 위치를 찾아 Next를 위와 같은 방법으로 지시하게 한 후 삽입한 행렬과 같은 열의 대각요소부터 Before를 따라가며 i를 비교하여 삽입할 위치를 찾아 Before를 바꾸면 된다. 이와 같이 링크구조를 사용하면 간단한 방법으로 비영요소의 삽입이 가능하다. 물론 반대로 경계조건을 적용할 때 생겨난 영의 값을 갖는 요소도 Before와 Next를 바꾸어 줌으로서 소거할 수 있다.

3. 시뮬레이션 및 검토

본 논문에서 제안한 압축방법과 기존의 압축방법을 비교하였다. 비교대상 알고리즘은 다음과 같은 3개의 1차원 배열로 구성하였고 2차원 정자장 문제에 적용하였다.

Irow : 계행렬 각 행의 비영요소 개수를 1행부터 누적하여 기억하는 배열. long integer형 배열로 절점수 만큼의 개수를 가진다.

Jcol : 각 비영요소의 위치 중 열에 대한 정보를 기억한다. long integer형 배열로 절점수의 8배 만큼의 개수를 가진다.

S : 비영요소의 값을 기억하는 배열. double형 변수로 절점수의 8배 만큼의 개수를 가진다.

우선 기억용량을 비교하면 표 2와 같다.

표 2 기억용량의 비교

단위 : byte

절점수	500	1000	1531	1910	2424	2993	3560
링크구조	46248	93912	143784	179616	228960	282744	337752
기존의 방법	48000	96000	146976	183360	232704	287328	341760

이미 설명한 바와 같이 비교대상인 기존의 압축방법은 비영요소의 개수를 절점수의 8배로 가정한 것이다. 그 이유는 일반적으로 절점 하나에 접할 수 있는 다른 절점의 개수를 최대 8개로 보고 최대한의 비영요소 수를 배열의 크기로 결정한 것이다.

계산시간의 비교는 표 3과 같다.

표 3 계산시간의 비교

계산종류	계행렬 조립	[S]×[A] 1회 수행
절점수	3,560 [개]	
비영요소의 수	24,336 [개]	
링크구조	0.546 [초]	0.104 [초]
기존의 방법	0.464 [초]	0.060 [초]

표 3에서는 링크구조를 이용한 알고리즘이 기존의 방법보다 계산속도가 느림을 보여주고 있다. 표 3에서 [S]×[A] 1회 수행은 반복법을 이용한 해법에서 계행렬과 솔루션 벡터의 적을 1회 수행한 시간이다.

4. 결 론

링크구조를 이용하여 계행렬을 압축하는 방법을 제시하였다. 링크구조는 동적으로 크기와 모양이 변할 수 있는 기억장소로 프로그램 수행 중에 발생하는 여러 가지 변화에 능동적으로 대처할 수 있는 장점이 있으나 기존의 압축방식보다 계산 속도가 느린 점이 단점으로 지적된다. 계산속도의 향상을 위한 최적의 알고리즘을 찾는 것이 앞으로 진행 되어야 할 연구과제이다.

참 고 문 헌

[1] 임단호, "전기계의 유한요소법", 동명사, 1987.

[2] Johannes J. Matin, "Data Types and Data Structures", Prentice Hall International, 1986.

[3] 한상영, "자료구조론", 영지문화사, 1990.