Lecture 15 : 리스트 응용 희소 행렬과 문서 처리

행렬은 거의 모든 공학과 과학에 사용되는 수학적 객체로 행렬을 프로그램으로 처리할 일이 매우 많다. 일반적으로 행렬을 이차원 배열 **int M[n][n]**을 활용하여 처리하면 간단히 해결된다. 그런데 문제는 n에 매우 큰 경우이다. 예를 들어 일반적인 C/C++ 컴파일러의 경우 n이 일정이상이면 컴파일이 불가능해진다. 여러분은 자신이 사용하는 환경에서 사용할 수 있는 배열의 최대 크기를 반드시 확인해야 한다.[1] 예를 들어 배열이 automatic일 경우와 main( ) 밖에 external로 선언하는 경우 하용되는 최대 크기가 다름을 인식하고 있어여 한다.

선형대수 교재에는 학생들이 이해하기 쉽도록 대략 6 by 6 정도의 행렬이 나오는데 현실에서 사용되는 행렬의 크기는 매우 크다. 예를 들어 기상 예측을 위하여 우리나라 전 국토를 1km X 1km 단위의 셀로 나눈다고 가정하면 연근해 바다를 포함해서 대략 700 by 700 정도의 행렬이 필요하다. 만일 100m 단위로 한다면 이 크기는 100배로 증가한다. 전 지구적으로 본다면 무려 100000 by 100000 크기의 행렬이 필요하다. 그러나 이 모든 cell에 측정 장치가 있는 것은 아니기 때문에[2] 대부분은 지점은 NULL값이 지정된다. 만일 이것을 C++로 구현한다고 하면 단순히 2차원 배열을 선언하여 저장할 수 없다. 따라서 뭔가 다른 자료구조가 필요하다. 기계공학에서 사용되는 FEM(Finate Element Method, 유한요소법) 역기 매우 큰 행렬 기반의 계산이 필요하기 때문에 단순한 2차원 배열로는 불가능하다. 이를 해결하기 위한 새로운 형식의 행렬, 매우 큰 크기의 행렬용 자료구조가 필요하다. 이것을 연결 리스트로 구현하는 방법을 이번 장에서 설명한다.

먼저 다중 리스트, 즉 리스트의 원소가 기본 원소인 {int, float, double, boolean}이 아닌, 또 다른 차원의 container인 경우를 살펴보자.

15.1 다음은 리스트의 원소가 또 다른 독립된 <리스트>인 <리스트 of 리스트>인 다중 리스트를 만드는 과정이다. 그리고 그 아래에는 range-based for loop으로 이중 리스트의 모든 원소를 출력하는 코드가 있다. 이 range-based for loop을 python의 기능을 모방한 것으로 익혀두면 매우 편한 기능이다.

---

1) 강사 컴퓨터의 경우 표준 32 bit C++ 컴파일러 global(external)에는 int x[25000][25000]까지 가능하고 internal(automatic)의 경우 int y[7000][7000]이 한계임을 확인할 수 있었다. 단 한가지 주의해야 할 사항은 컴파일이 된다고 해서 제대로 동작되는 것은 아니다. 어떤 경우 memory allocation만 된 상황에서 실제 access를 하면 오류가 발생하는 수가 있기 때문에 반드시 accsee testing을 해야 한다.

2) 기상 예측은 측정장소에서 기록된 정보를 중심으로 다른 지역은 interpoplation(보간)을 통하여 예측을 하게 된다. 따라서 우리나라의 예를 들어 전체 지역은 korea[500][500], 25만개의 위치점이 있지만 1차 측정값이 부여된 지점은 1000개 안팍이다. 즉 1/250개 행렬 원소만 의미있는 값을 가진다. 이렇게 표현된 행렬은 희소행렬의 전형적인 예라고 할 수 있다.

```
int main() {
    list<string> lista(5, "apple");
    list<string> listb(8, "tomato");
    printList(lista);
    printList(listb);

    list<list <string> > listlist;  // typedef 으로 해도 좋다.
    listlist.push_back(lista);
    listlist.push_back(listb);
    cout << "Done" << endl;
}
```

```
#define deepout(msg,deepl)  cout<<"\n"<<msg<<" : \n";\
        for(auto w : deepl ){cout<<"\n";for(auto q:w)\
                              cout<< " " << q ;}
using namespace std;
typedef list<int> iLIST ;

int main() {
list< iLIST > mylist  = {  { 2, 8,} , {7}, {}, {5, 3, 1} , \
                          {4}, {0, 3, 4, 5, 8,12} } ;
   for(auto wlist : mylist ) {
      cout << "\n" ;
          for(auto s : wlist ){
               cout << " " << s ;
          }// end of inner loop
    } // end of outer loop
    deepout("이중 리스트의 출력", mylist ) ;
   return 0;
}
```

15.2 희소 행렬(**Sparse matrix**)

전체 행렬의 차원이 n by n 일 때 O(n)정도의 원소가 있으면 희소 행렬로 볼 수 있다. 예를 들어 40,000 * 40,000 matrix에서 원소의 개수가 10,000 - 50,000 개 정도 있을 경우에는 희소행렬로 보고 그에 알맞은 자료구조를 꾸며야 한다. 희소행렬은 일반적인 공학에서 매우 자주 나타나는 문제이므로 이를 위한 자료구조를 잘 가지고 있어야 한다. (온도 sampling map)

주어진 행렬에서 의미있는 원소의 갯수가 K개 라고 하고 이것이 대략 O(N)개 정도라고 할 때 이 정도의 공간을 가지고 일반적인 행렬 연산을 지원하는 자료구조를 구성해보자. 이렇게 실제 사용되는 자료의 갯수만큼의 공간으로 구성된 자료구조를 특별히 "간결 자료구조(succint data structure)"라고 부른다[3]. 이것은 이론 전산학과, 자료구조 연구자들의 가장 중요한 연구주제이기도 하다. 단 간결하지만 대부분의 연산의 동작 복잡도를 이론적 하한지(theoretical lower bound)로 맞추어야 한다.[4] 이 둘의 모순되는 조건을 충족시키는 자료구조를 만드는 것은 매우 의미있지만 어려운 일이다.

3) 부산대학교 대학원에서는 조환규 교수의 "고급자료구조 특론"에서 다루어진다.

4) 예를 들어 N개의 원소가 있는 상황에서 i) 특정 데이터의 탐색을 $O(\log N)$에, ii) 삽입과 삭제 역시 $O(\log N)$에 지원하는 자료구조를 생각해보자. 일반적인 Linear array로는 불가능하다.

10 by 10 행렬 M[10][10]의 예

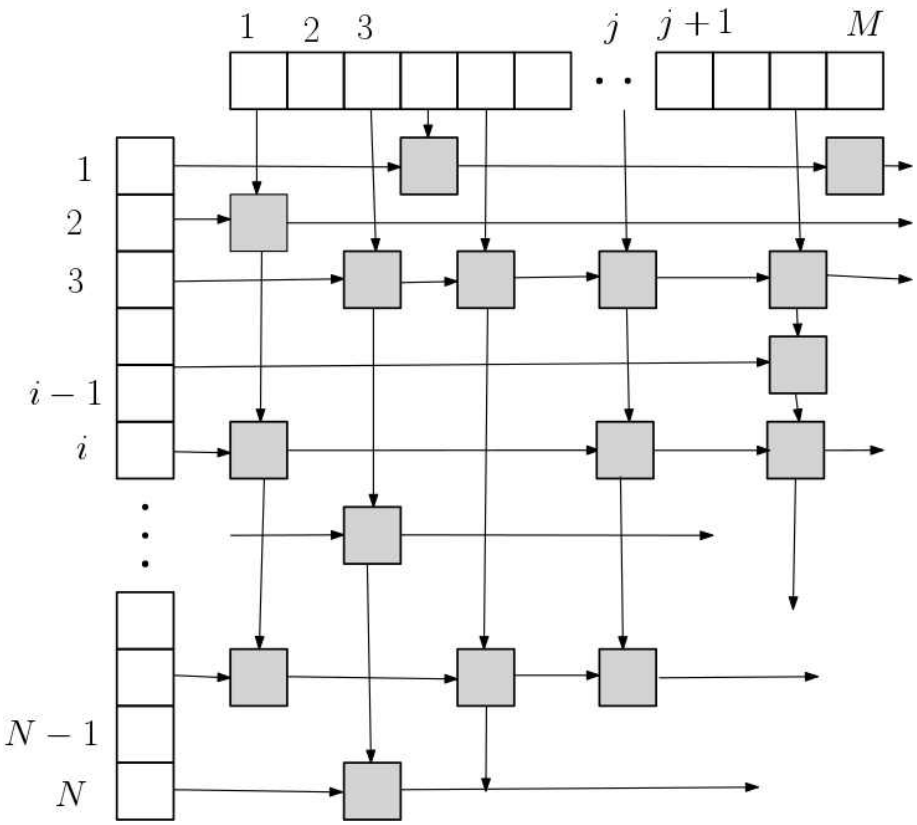
	1	2	3	4	5	6	7	8	9	10
1						1		5	6	7
2		6					-8		9	-8
3			5		45				7	
4		-4								
5					4					
6		7			65			13		
7										
8					9				6	
9	5		2							
10							-9			

25 by 25 행렬 W[25][25]의 예

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
1						-1									5										
2																		11							
3																									
4																									
5					4																			9	
6																									
7																		4							
8									6																
9																									
10																									
11																									
12																									
13					7								6					9						6	
14																									
15																								3	
16			5																						
17																									
18													9												
19																									
20			3				6																		
21																									
22																									
23															4										
24	6		-1																						4
25																									

15.5 희소 행렬(Sparse matrix) 자료구조 지원해야(support) 하는 동작(Operation)을 먼저 생각해보아야 한다. 먼저 일반적인 방법, 2차원 배열과 연결 리스트를 이용하는 두 가지의 다른 자료구조를 대상으로 비교해보자. 비교할 때는 각 자료구조의 크기를 기준으로 비교해야 한다. 1번 일반 2차원 배열의 경우에는 n*m, 희소행렬의 구조에는 실제 assign된 메모리의 양으로 계산해야 한다.

연결 리스트로 희소행렬을 구현하는 방법을 지원해야 하는 동작에 따라서 달라질 수 있지만 가장 보편적인 방법은 각 row와 column에 대하여 circular list로 연결한다는 것이다. 동시에 각 row와 column은 random access가 가능한 vector나 array로 구현한다. 이렇게 하면 특정 row나 column에는 O(1)에 갈 수 있다.



다루어야 할 행렬에서 0이 아닌 의미있는 행렬 entry의 갯수는 K개 라고 한다. 그리고 이 K는 대략 $O(\sqrt{N}\sqrt{N}) = N$ 라고 가정한다.

비교대상	x[i][j]의 경우	리스트 기반 희소행렬
전체 메모리 크기	N^2	$2N+K=cN$
M[i][j]의 값은 ?		
0 아닌 원소 출력		
M[i][*] 모두 출력		
M[i][j]=p 지정		
M + N 덧셈		
M * N 곱셈		
M[i][i] 대각원소		

문제는 이 연결 리스트 구조 희소행렬에서 새로운 원소를 삽입 삭제할 경우이다. 이 경우의 절차를 살펴보자.

질문) 왜 column만 연결해도 되는데, row와 column모두에 연결을 해야할까 ?
답) 행렬곱을 할 때 어떤 행렬은 column scan, 어떤 행렬을 row scan을 해야하기 때문이다. 만일 행렬곱 동작을 지원할 필요가 없다면 이 연결은 필요치 않다.

15.6 위와 같은 연결구조 기반의 희소 행렬에서 특정 원소를 탐색, 그리고 삽입하고 삭제하는 과정을 제시하시오.

삽입: insert M[i][j] = Value
i) i번 column에 가서 list를 scan하면서 해당 원소를 찾는다.
O(1) O(w)
2) 만일 그런 원소가 있다면 값만 udate한다. 없다면
3) column list에 해당 원소를 insert( )한다.
4) 이제 중요한 작업이 남아있다. 이 추가된 원소가 row에서도
link adjust를 해야 하는지 살펴 보아야 한다.
이 작업이 상당히 까다롭다.

15.7 어떤 동작을 많이 할 때 어떤 자료구조가 유용한지 생각해보고, 2차원 행렬에서 생각할 수 있는 다양한 응용형 동작(operation)을 아는 대로 나열해봅시다. 특히 전체 Clearing(전체 값을 모두 지우는 동작)이 매우 자주 사용된다고 할 때 어떤 자료구조가 좋은지 제시해 보시오.

일반적으로 책(book)은 일반형 리스트를 이용해서 표현하기에 매우 좋은 구조를 가지고 있다. 책의 계층적 구조를 먼저 살펴보자. 가장 낮은 단계는 문자(character)이다.

Higher Level ---------- Lower Level
문자(char)
어절 또는 단어(string)
문장(statement)
문단(paragraph)
절(section)
장(chapter)
권(book)
호(volume)

그리고 이들은 모두 하나의 total order 순서로 되어 있다. 이것이 가장 큰 특징이므로 깊이가 매우 깊은 다중 연결 리스트로 표현이 가능하다. 일단 우리는 string(어절,단어)로부터 생각해보자. 다음 코드를 참조하여 책의 구조에 어떻게 구현되어 있는지를 확인해보자.

```
#include <bits/stdc++.h>
#define allout(msg,lx)  cout<<"\n"<<msg<<" : ";\
            for(auto &w : lx ) cout<< " " << w ;
#define deepout(msg,deepl)  cout<<"\n"<<msg<<" : \n";\
            for(auto w : deepl ){cout<<"\n";for(auto q:w)\
                cout<< " " << q ;}
using namespace std;

typedef string            Word ;
typedef list <Word>        Statement   ;
typedef list <Statement>    Paragraph   ;
typedef list <Paragraph>    Section     ;
typedef list <Section>      Book       ;

int main() {

   Statement s1 {"자료구조는", "정말", "재미있는", "과목인가?"} ;
   Statement s2 {"교수님", "정말", "짱짱맨" } ;

   allout( "문장 s1 = ", s1 ) ;
   allout( "문장 s2 = ", s2 ) ;
   Paragraph DS ;

   DS.push_back( s1 ) ;
   DS.push_back( s2 ) ;
   deepout( "\n 문단의 내용을 찍어보자", DS ) ;
      return 0;
}
```

15.7 입력화일 book.inp 에는 어떤 책의 내용이 있다. 각 문장의 끝은 char '$'로 끝이 난다. 여러분은 이 화일을 읽어서 다음의 질문에 답을 하는(interactive)하게 프로그램을 작성해야 한다. 각 console에서 첫 번째 단어는 명령어 code이다. 따라서 여러분은 각 code에 해당되는 단어를 먼저 token( )으로 처리하여 그것이 무엇인지 알아야 한다.

명령어 format	내용 (해당 항목이 없으면 "N/A: 을 출력합니다.)
> word i j	i번째 문장 중 j번째 word를 출력한다.
> print i	i번째 문장을 모두 출력한다.
> delete i	i번째 문장을 리스트에서 삭제한다.
> insert i j "string"	i번째 문장의 j번째 단어 뒤에 스트링을 입력한다.
> check "string"	"string" word인 (문장,단어) 번호를 모두 출력 한다

15.8 이 문제에서 여러분이 작성해야 할 프로그램의 전체 골격을 dummy module(함수의 처음과 끝만 있고 중간은 아직 작성하지 않는 함수)을 이용하여 설계해 보시오.

```
#include "mystl.h"
#include "mytool.h"

int main( ) {
   read_text( ) ;
   interactive( ) ;
   ending( ) ;
} // end of main
```

```
void interactive( ) {
   while(1) {
       a = get_command( ) ;
       if( a == OVER) break ;
       process_cmd ( a ) ;
  } // end of while( )
} // end of interactive( )
```