자료구조 (특별)

Chap 01. 배열 (sparse matrix) - 실습

2019년 하계 특별 학기

컴퓨터과학과
민경하

Contents

Sparse matrix (희소 행렬)

1.1 자료 구조

1.2 일반 행렬의 연산

1.3 희소 행렬의 리스트 연산

1.4 희소 행렬의 연산

프로그램 구성

- smat.cpp
  - main 함수
- mat.cpp
  - 일반 행렬에 대한 연산
- sparse.h
  - 희소 행렬에 대한 자료 구조
- sparse.cpp
  - 희소 행렬에 대한 연산

프로그램 구성

- 행렬 읽기 및 시간 측정 → time.h

```
#include <time.h>

int main()
{
    clock_t tik, tok;

    tik = clock();
    // do something
    tok = clock();
    printf("Loading: %f\n", (float)(tok - tik) / CLOCKS_PER_SEC);
}
```

프로그램 결과

- 일반 행렬 연산을 구현한다
- 리스트 연산을 이용해서 희소 행렬 연산을 구현한다
- 일반 행렬 연산의 시간과 희소 행렬 연산의 시간을 비교한다

1.1 자료 구조

- 행렬의 자료구조
  - 2차원 배열

```
int n;
int **matA;
int **matB;
int **matC;
```

1.1 자료 구조

- 희소 행렬의 자료구조
  - 1차원 배열
    - 배열의 각 원소는 행, 렬, 값을 저장함
  - 보조 기억 장소로 startPos와 rowTerms를 포함

```
class element {
public:
        int row;
        int col;
        int val;
};
```

```
class smatrix {
        int n;
        int cnt;
        element *list;

        int *startPos, *rowTerms;
}
```

1.2 일반 행렬의 연산

- 행렬의 연산 (mat.cpp)
  - 행렬 읽기 (load)
  - 전치 (transpose)
  - 합 (add)
  - 곱 (multiply)

1.3 희소 행렬의 리스트 연산

- 리스트 연산 (sparse.h & sparse.cpp)
  - 추가 (append)
    - 리스트의 마지막에 원소를 삽입
  - 저장 (store)
    - 리스트의 i번째 위치의 원소를 제거하고 새로운 원소로 대체

1.4 희소 행렬의 연산

- 희소 행렬의 연산
  - 일반 행렬을 희소 행렬 표현으로 변환
  - startPos와 rowTerms를 계산
  - fastTranspose
  - fastAdd
  - fastMult

1.4 희소 행렬의 연산

- 일반 행렬을 희소 행렬 표현으로 변환

```
void smatrix::convert_to_smat(int n, int **mat)
{
    //  1. count non-zero elements

    //  2. alloc smat

    //  3. build smat

}
```

1.4 희소 행렬의 연산

- startPos와 rowTerms를 계산

```
void smatrix::ready()
{
    // 1. rowTerms 계산

    // 2. startPos 계산 ← rowTerms를 이용
}
```

1.4 희소 행렬의 연산

- fastAdd → 16번 merge ( )의 변형

```
void smatrix::fastAdd(smatrix smatA, smatrix smatB)
{
    // 1. smatA의 행 == smatB의 행
        // 1.1 smatA의 열 == smatB의 열

        // 1.2 smatA의 열 < smatB의 열

        // 1.3 smatA의 열 > smatB의 열

    // 2. smatA의 행 < smatB의 행

    // 3. smatA의 행 > smatB의 행
}
```

1.4 희소 행렬의 연산

- fastMult

```
void smatrix::fastMult(smatrix smatA, smatrix smatB)
{
  // smatA의 모든 행과 smatB의 모든 열에 대해서 연산 →

  // smatA의 모든 행과 smatB의 transpose의 모든 행에 대해서 연산

  // smatC[i, j, k] ← smatA의 i번째 행과 smatBT의 j번째 형의 곱
     // smatA의 i번째 행 ← startPosA[i] ~ startPosA[i] + rowTermsA[i]
     // smatBT의 j번째 행 ← startPosB[i] ~ startPosB[i] + rowTermsB[i]

}
```