행열의 계산 프로그램

32214913\_홍석일

1. 문제 기술

행렬의 덧셈, 뺄셈, 곱셈, element-wise 나눗셈의 구현이 필요하다.

추가적으로, 난수에 의해 행렬 내부의 값을 할당하는 과정이 필요하다.

1. 분석 및 설계
2. 행열(matrix)의 설계  
   행열의 계산에 있어서 필요한 변수의 기본적인 형태는 변수이다. 행열은 기본적인 형태로 보아 2차원 배열과 유사하나, 이번 환경에서는 행과 열의 곱(전체 원소의 개수)만큼의 길이를 갖는 1차원 배열을 이용하여 포인터 연산을 통해 2차원 배열로 사용하고자 한다.  
   배열의 동적할당시 calloc함수를 사용하여 할당 초기값을 0으로 할당하고, 원소는 random함수를 이용해 1부터 10까지의 값을 할당하도록 구현.
3. 행열의 덧셈, 뺄셈, 나눗셈 설계  
   위의 세 형식의 연산은 행과 열의 크기가 동일한 두 행열에 대한 1:1매칭 사칙연산이기 때문에 for문을 이용해 전체 원소의 개수에 대해 매칭되는 원소의 연산을 한다.  
   이후, answer이라는 새로 선언된 배열에 저장해 연산의 결과를 남긴다.
4. 행열의 곱 설계  
   앞서 설계한 세 방식과는 다르게 연산에 사용되는 두 행열의 크기조건이 조금 다르다.  
   한 행열의 열값과 다른 행열의 행값이 같은 경우에 연산이 가능하다.  
   행렬의 곱셈 과정의 경우 3중 for문을 이용해 연산한다.
5. 희소행렬, 전치행렬 구성 설계
   1. 희소행렬: 행렬 하나만을 사용해 int [0이 아닌 값읠 개수 + 1][3]의 배열을 선언한다.  
      0번째 원소에는 행과 열값, 0이 아닌 값의 개수를 입력해둔 후, 이후의 원소에 0이 아닌 값의 정보를 저장하도록 만든다.
   2. 전치행렬: 이것 역시 행렬 하나만을 사용해 answer행렬에 값을 할당한다.  
      다만, answer행렬의 행과 열은 mat1행열의 행과 열값을 바꾼 것으로 한다.  
      이후 포인터 연산을 통해 행과 열값을 뒤바꾼다.
6. 실행시간 측정  
   각 연산 전에 time\_t형식의 start변수를 이용해 연산 시작 직전 시간을 기록한다.  
   이후 연산이 종료된 후 end 변수로 시간을 기록한 후 end-start연산을 통해 소요된 시간을 출력한다.
7. 실행

텍스트, 스크린샷, 디자인이(가) 표시된 사진

자동 생성된 설명텍스트, 스크린샷, 디자인이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 디자인이(가) 표시된 사진

자동 생성된 설명텍스트, 스크린샷, 디자인이(가) 표시된 사진

자동 생성된 설명

행렬의 사칙연산 확인

텍스트, 스크린샷, 폰트, 디자인이(가) 표시된 사진

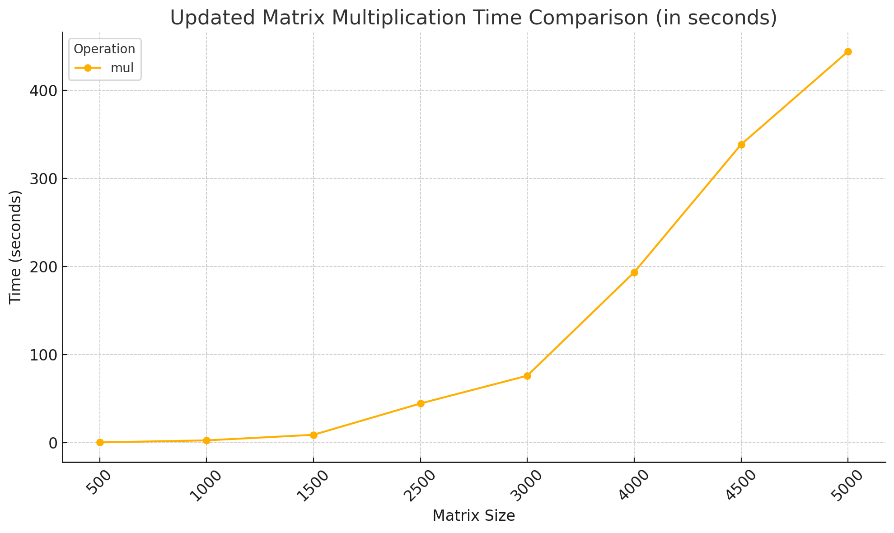
자동 생성된 설명텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

전치행렬 및 희소행렬 코드 정상작동 확인

1. 연산시간 비교

텍스트, 도표, 라인, 그래프이(가) 표시된 사진

자동 생성된 설명  
덧셈, 뺄셈, 나눗셈의 소요시간을 비교한 것이다. Matrix Size는 행과 열의 크기이다.  
사칙연산의 경우 대체로 비슷 연산시간을 가지며, 연산할 수의 수가 많아짐에 따라 점점 기울기가 높아지는 경향을 보인다. 실제로 한 변의 크기가 커짐에 따라 연산량 역시 기하급수적으로 늘어나니, 어느정도 크기와 소요시간에 대한 비례관계가 성립한다고 볼 수 있다.  
연산에 사용된 시간의 경우 40000\*40000행열의 연산도 3초를 넘기지 않을 정도로 빠르게 끝난다. 모든 원소에 대해서 1:1로 한 번의 연산만을 하기 때문에 시간복잡도는 원소의 수 N에서 크게 벗어나지 않는다.  
더 이상 표본의 개수를 늘리면 메모리 부족으로 인해 제대로 프로그램이 작동하지 않기 때문에 더 큰 크기의 행렬로 실험하는 것이 불가능했다.  
  


행열곱의 소요시간을 비교한 것이다. 겉보기엔 덧셈, 뺄셈 등의 계산과 비슷해 보일지 모른다. 하지만, 행열의 원소 수는 훨씬 적은 반면에 소요시간은 400초를 훌쩍 넘어간다는 점에서 훨씬 급격하게 실행시간이 상승함을 알 수 있다. 시간복잡도를 계산해보자면, 3중 for문을 사용하고 있다는 점에서 N^3가량의 시간복잡도를 갖는다는 것을 알 수 있다.

1. 토의 및 개선점
   1. 행렬의 크기를 일정 이상 증가시킬 수 없어 행과 열이 50000이상의 값을 갖는, 더 큰 크기의 행렬의 계산이 불가능했다.  
      사유는 메모리의 부족으로 50000정도의 값으로 실험했을 때에는 메모리가 부족해 할당이 제대로 되지 않는 모습을 보았다.  
      보다 제대로 테스트하기 위해선, 64GB이상의 메모리를 가진 시스템이 필요해보인다.
   2. 행렬곱 연산의 경우 너무 많은 계산시간이 소요됐다.  
      시간복잡도가 훨씬 높은 알고리즘이기 때문에 연산량을 조금만 늘려도 큰 차이로 실행시간이 늘어났기 때문에 덧셈 과 뺄셈 등의 연산과 유의미한 비교를 하기 어려웠다.
   3. 모듈라 프로그래밍을 하기에 애매한 코드 길이의 프로그램이다.  
      각 행렬의 계산 함수와 행렬의 할당, 출력 등 여러 부가기능을 위해 module파일을 만들었지만, 각 기능을 함수로 구현할시 사용되는 코드의 길이가 애매하여 실질적인 모듈라 프로그래밍이라고 보기엔 어려웠다. 텍스트, 스크린샷, 라인, 폰트이(가) 표시된 사진

      자동 생성된 설명텍스트, 스크린샷, 라인, 폰트이(가) 표시된 사진

      자동 생성된 설명

