**자료구조.2021**

**과제 1**

20172674

신동녘

**다항식 – 방식1**

텍스트이(가) 표시된 사진

자동 생성된 설명

*(실행결과 캡처화면)*

먼저 다항식의 저장 방식 중 첫번째 방식은 고차 순으로 모든 항(계수가 0인 항 포함)의 계수를 배열에 넣어 관리하는 방식입니다. 수식 두 개를 입력 받아, 이 두 식에 대한 덧셈과 곱셈의 식을 계산합니다. 이렇게 정리된 네 가지 식 중 한 가지 식에 사용자 입력값을 대입하여 결과값을 구하는 문제입니다.

텍스트이(가) 표시된 사진

자동 생성된 설명덧셈의 경우 apos와 bpos에 해당하는 인덱스 관리 변수를 생성하여 두 식의 차수가 더 큰 쪽이 있다면 자동으로 결과 배열에 추가시키고, 두 식의 차수가 같다면 더한 값을 결과 배열(사진 상 z에 해당함)에 추가시키는 방식으로 진행하였습니다. 이 때 배열로 다항식을 관리하는 방식에 따라 계수가 0이더라도 차수가 존재한다면 칸을 차지하고 있고 이에 따라 결과 배열은 빈틈없이 자리가 두 다항식의 최고 차수에 맞게 채워질 수 있습니다.

이 때 기존에 주어진 poly\_add 함수의 while문의 반복 조건은 동차 다항식 두 개가 주어질 경우 올바른 계산이 이루어지지 않았기 때문에 두 차수가 모두 상수항을 가리킬 때로 설정하였다는 점에서 어느 다항식의 경우에든 올바르게 작동할 수 있도록 하였습니다.

텍스트이(가) 표시된 사진

자동 생성된 설명곱셈의 경우 방식1이 계수가 없는 차수에도 0을 채워 넣는 다는 점에서 곱셈 결과 최고차 항으로부터 상수항까지 담을 수 있는 길이의 0 배열을 생성하였고, 이후 모든 항을 단순 교차 연산하여 i+j 즉 곱했을 때의 차수에 맞는 칸에 곱셈 결과를 더해 주어 결과를 도출하였습니다.

**다항식 – 방식2**

텍스트이(가) 표시된 사진

자동 생성된 설명

*(실행결과 캡처화면)*

다음 다항식의 저장 방식 중 두번째 방식은 고차 순으로 항(계수가 0인 항은 포함 포함 하지 않고)의 계수와 차수를 동시에 배열에 넣어 관리하는 방식입니다. 수식 두 개를 입력 받아, 이 두 식에 대한 덧셈과 곱셈의 식을 계산합니다. 이렇게 정리된 네 가지 식 중 한 가지 식에 사용자 입력값을 대입하여 결과값을 구하는 문제입니다.

텍스트이(가) 표시된 사진

자동 생성된 설명방식2의 경우 방식1처럼 차수 자체를 기록하는 변수를 따로 만들 필요가 없었습니다. 각 계수는 각자의 차수를 담고 있으므로 차수에 직접 접근하여 최고 차수부터 순차적으로 탐색하는 방식으로 조건문에 관한 판단을 진행할 수 있었습니다. 두 다항식의 차수를 비교하며 공통된 항을 가지고 있을 때에만 더해주고 그렇지 않은 경우 더 고차의 항을 가지고 있는 쪽만 결과 배열에 옮겨 담았습니다.

텍스트이(가) 표시된 사진

자동 생성된 설명곱셈의 경우 방식1처럼 간단하게 이루어지지 않았습니다. 가장 먼저 앞선 방식과 마찬가지로 이중 배열을 통해 각 행에 접근합니다. 이 때 임시로 곱셈결과가 되는 계수와 차수로 이루어진 배열을 만들어 두었습니다. 또 z\_list라는 z라는 결과 배열에 담긴 차수를 모아두는 배열을 만들어두어 임시 배열과 비교할 수 있도록 하였고 해당 차수가 이미 존재한다면 해당 차수 위치의 계수를 증가시키고, 없다면 임시로 만들어 둔 배열 자체를 임시 배열에 추가 및 차수를 저장하는 z\_list배열에도 차수를 저장해 주었습니다.

텍스트이(가) 표시된 사진

자동 생성된 설명

마지막으로 두 방식 모두에 해당 수식에 값을 대입하여 결과값을 얻을 때, 다항식 네 개를 리스트로 묶어 간단하게 관리할 수 있도록 하였습니다.

**행렬 – 방식1(좌), 방식2(우)**

텍스트, 원격, 제어이(가) 표시된 사진

자동 생성된 설명텍스트이(가) 표시된 사진

자동 생성된 설명

*(실행결과 캡처화면)*

행렬의 규격과 두 행렬에 대한 데이터를 입력 받고, 이에 대해 2차원 배열로 저장하거나, 희소 행렬방식으로 저장하고, 각각의 방식에 대한 행렬의 덧셈과 곱셈을 진행하여 데이터 관리의 효율성을 확인하는 문제입니다.

텍스트이(가) 표시된 사진

자동 생성된 설명먼저 2차원 배열로 저장된 행렬의 덧셈과 곱셈은 경우의 수에 관한 고려 없이 각각의 위치에 알맞는 연산을 진행하였습니다. 특히 덧셈의 경우 C라는 결과 행렬을 입력된 규격에 맞게 0으로 초기화하여 각각의 위치에 해당하는 두 행렬의 원소를 더해주었습니다. 곱셈은 일반 행렬 연산과 같이 모든 원소에 대해 c[i][j]원소에 대해서 각 위치에 누적되어야하는 행렬의 곱들의 위치를 반복문에 따라 알맞게 설정해주었습니다. 예컨데 a와 b의 행렬 곱에서 c[i][j]위치에 누적되는 곱은 (a[i][0]\*b[0][j])+ (a[i][1]\*b[1][j])+(a[i][2]\*b[2][j])+…+(a[i][n]\*b[n][j])에 해당합니다.

희소행렬의 덧셈은 행렬을 저장한 배열상의 좌표값과 실제 행렬에서의 좌표 값의 차이가 있다는 점에서 텍스트이(가) 표시된 사진

자동 생성된 설명2차원 배열에 저장하는 행렬과 그 계산 방식에서 궤를 달리합니다. 따라서 행렬 a와 b의 위치를 정해두고 각각을 순차 탐색하면서 row와 col의 대소 비교를 통해 완전히 좌표가 같은 경우에만 두 수를 더 한 값과 좌표를 하나의 배열로 만들어 결과 배열인 z에 추가합니다. 그 외에 값에서 차이가 날 경우 좌표값이 더 작은 행렬을 가진 값먼저 마찬가지로 좌표값과 해당 값으로 이루어진 임의의 배열을 만들어 결과 배열인 z에 추가하였습니다. 이 때 while문의 반복 조건은 어느 한 쪽 행렬이라도 마지막에 이르면 종료되는 것으로 설정하였으므로 마지막부분에 상대적으로 다른 행렬에 비해 길이가 길어 처리되지 못하고 남아있는 부분에 대해 처리하는 부분을 추가하였습니다.

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명마지막으로 희소행렬의 곱셈입니다. 희소행렬은 덧셈과 마찬가지로 모든 좌표에 대한 모든 좌표의 곱을 수행하는 과정에서 행렬의 곱 연산이 실제 이루어지는 부분에 관해서만 처리를 해야하고 여러 곱 연산이 더해져 최종 결과 배열의 한 좌표를 이루게 된다는 점에서 까다로웠습니다. 먼저 앞서 언급한 것처럼 조건 문에서 연산 하는 행렬의 Y좌표와 피연산 행렬의 X좌표가 같은지를 판단합니다. 이 때 isAppended 변수는 곱해져서 만들어진 좌표에 값이 존재하지 않을 경우 새로 행렬의 좌표와 값을 곱해주기 위함입니다. 이때 연산 행렬의 Y좌표와 피연산 행렬의 X좌표가 같고 두 항 모두가 0이 아니라면 곱해진 값은 (연산 행렬의 X좌표, 피연산행렬의 Y좌표) 위치에 누적된다는 아이디어를 활용하여 코드를 완성하였습니다.

텍스트이(가) 표시된 사진

자동 생성된 설명

*데이터를 2차원 배열 행렬로 변환하는 함수(좌), 희소행렬로 변환하는 함수(우)*

마지막으로 데이터를 입력 받아 2차원 배열 형태의 행렬을 만드는 함수와 그 데이터를 희소 행렬로 변화하는 함수 입니다. 먼저 2차원 배열 형태로 생성하는 normal\_matrix 함수는 데이터 리스트롸 규격을 함께 매개변수로 받아, 규격만큼의 반복문을 통해 한 행의 횟수만큼의 리스트를 만들고 해당 행을 결과 배열에 추가하는 방식으로 진행하여 N X N 행렬을 완성할 수 있었습니다. 다음 희소행렬을 만드는 sparse\_matrix 함수는 입력받은 행렬을 규격에 따라 이중 for문으로 순회하면서 데이터가 있을 경우에만 [행, 열, 값] 으로 이루어진 리스트를 생성하여 결과 배열에 추가하였습니다.

**감사합니다.**