**Advanced Coding Practice HW Problem #3**

**20191599 송경호**

**A) Matrix Addition**

**A-1) 문제 인식 및 아이디어 구상**

다음과 같은 pseudo code의 시간복잡도를 최적화하는 문제이다.

텍스트, 스크린샷, 폰트, 대수학이(가) 표시된 사진

자동 생성된 설명

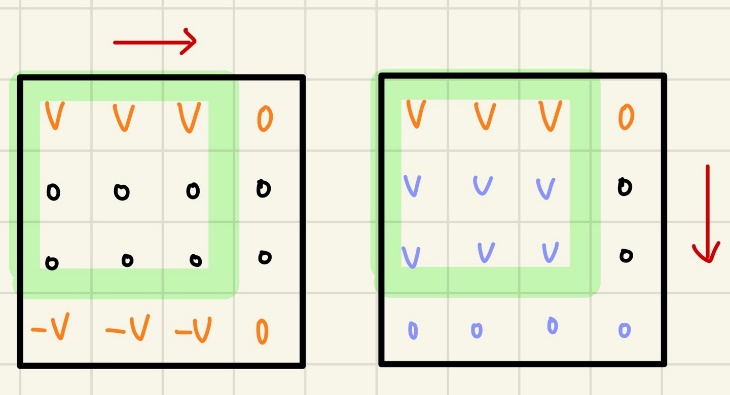
임의이 값을 가진 NxN matrix A에 R1, R2, C1, C2, V를 입력 받아, A[R1][C1] / A[R1][C2] / A[R2][C1] / A[R2][C2]네 칸을 꼭짓점으로 하는 A의 submatrix에 입력값 V를 더하는 연산을 총 N번 반복하라는 의미로 해석된다. 또한 해당 문제를 해결하기 위해 위의 코드는 각 칸에 V를 하나씩 더하여 O(N3)의 시간복잡도를 가지는 알고리즘은 채택하고 있다.

위의 알고리즘에서 매 입력마다 submatrix를 모두 순회하는 절차를 감소시키기 위해 다음 성질을 이용한다. 다음은 4x4 matrix에서 [0][0]을 꼭짓점으로 하는 3x3 submatrix에 V를 더하는 예제이다.

사각형, 직사각형, 텍스트이(가) 표시된 사진

자동 생성된 설명

먼저 위와 같이 [0][0]과 submatrix의 대각선 아래칸인 [4][4]에 V값을 대입한다. 또한 [0][4]와 [4][0]에도 V값을 대입한다.



이후 가장 좌측행을 기준으로 좌측행의 값을 우측행의 값에 더한다. 예를 들어 [0][1]에 [0][0]을 더하고 [0][2]에 [0][1]을 더하는 식으로 진행하게 되면 왼쪽 그림의 상태가 완성된다. 이후로 이번엔 가장 위쪽열을 기준으로 위쪽열의 값을 아래쪽열의 값에 더한다. 결과적으로 3x3 submatrix에 V를 더한 상태가 나오게 된다.

위 방식은 matrix에 변경할 범위와 값을 손쉽게 저장한 뒤에 마지막 순회를 통해 값을 적용할 수 있다는 점에서 택하였다.

**A-2) 알고리즘**

편의를 위해 행렬의 index는 1부터 N까지로 정의한다.

*i) NxN크기의 0으로 초기화된 행렬 B를 생성한다.*

*ii) B[R1][C1] += V, B[R2+1][C2+1] += V와 B[R1][C2+1] -= V, B[R2+1][C1] -= V 연산을 N번의 입력을 받아 수행한다. 이를 통해 행렬의 변경될 submatrix의 값을 저장할 수 있다.*

*iii) 가장 좌측행을 시작으로 우측행의 값에 좌측행을 더한다. 다음과 같은 코드로 나타난다.*

|  |
| --- |
| for i := 1 to N do  for j := 2 to N do  B[i][j] += B[i][j-1]  end  end |

*마찬가지로 가장 위쪽열을 시작으로 아래쪽열의 값에 위쪽열을 더한다. 다음과 같은 코드로 나타난다.*

|  |
| --- |
| for j := 1 to N do  for i := 2 to N do  B[i][j] += B[i-1][j]  end  end |

*iv) matrix B를 matrix A에 더한다*.

**A-3) 시간, 공간복잡도, 확인사항**

위의 알고리즘을 통해 매 입력마다 submatrix전체를 순회하며 값들을 변경할 필요없이 전체 matrix를 한번 순회하며 모두 계산이 가능하다. 이는 변경이 일어날 값과 범위를 저장해놓고 한번에 적용하기 때문이다.

한가지 주의해야 할 점은 R2 혹은 C2가 N일 때 R2+1과 C2+1에 접근해야 하는데, 이는 우리가 확인할 matrix의 범위에서 벗어남으로 따로 연산을 처리할 필요가 없다.

해당 알고리즘은 NxN matrix B를 추가로 갖고 있어야하므로 O(N2)의 공간복잡도를 갖는다. 그러나 전체 matrix를 한번만 순회하면 되기 때문에 시간복잡도 역시 O(N2)이 되어 기존 알고리즘 보다 훨씬 향상된 속도를 갖는다.

**B) Half-Circle Property**

**B-1) 문제 인식 및 아이디어 구상**

원점을 중심으로 원의 원주 위의 점들이 주어질 때, 원점을 지나는 직선을 그어 해당 점들의 집합이 직선을 기준으로 한쪽 편에만 존재할 수 있는지를 묻는 문제이다. 즉 다시 말해, 주어진 점 집합을 모두 포함하는 반원이 존재하는지를 찾으면 되는 문제이다.

반지름을 r이 주어졌을 때, 반원을 정의하기 위해선 양쪽 끝점과 어떤 방향으로 반원이 생길지를 정해야 한다고 생각했다. 따라서 먼저 반원의 방향을 정의하기 위해선 기준선이 필요하다. 기준선은 최초로 입력된 점과 원점을 잇는 직선을 사용한다.

라인, 도표, 원, 그래프이(가) 표시된 사진

자동 생성된 설명

**B-2) 알고리즘**

*i) 기준선을 기준으로 시계와 반시계 방향의 두 집합을 생성한다.*

*ii) 직선의 방정식과 부등식을 이용하여 다음 입력되는 점들을 시계방향 집합과 반시계방향 집합에 모두 넣는다.*

*ii-1) 이때, 최초점과 입력점 사이의 거리를 계산하여 두 집합에 각 각 거리가 가장 먼 좌표를 저장한다.*

도표, 라인, 원, 그래프이(가) 표시된 사진

자동 생성된 설명

*iii) 각 집합에서 가장 먼 두 점을 잇는 직선과 최초점간의 거리를 계산한다.*

*계산된 거리가 반지름보다 작을 경우 모든 점은 반원위에 있을 수 있지만 더 클 경우 모든 점은 반원위에 있을 수 없다.*

라인, 도표, 원, 그래프이(가) 표시된 사진

자동 생성된 설명

**B-3) 시간복잡도, 공간복잡도 및 고려사항**

최초점의 좌표를 (a,b)라고 할때, 각 점마다 계산해야 할 연산은 두 집합 중 어디에 속하는지를 계산하기 위한 연산과

|  |
| --- |
| *a(y-b) > b(x-a)* |

(a,b)와의 거리를 구하는 연산과 기존 최대값과의 비교이다. 이 연산들은 모두 O(1)에 수행되므로 모든 점에 대해서는 O(N)의 수행시간이 걸린다.

마지막으로 거리가 가장 먼 두 점을 잇는 직선과 (a, b)간의 거리를 구하는 연산이 끝이다. 또한 공간복잡도는 최초 점의 좌표, 각 집합의 거리 최대점을 저장해야 하므로 O(1)이 된다.

또한 최초에 아이디어 구상 시, 각 집합의 최대점이 갱신될 때마다 두 점을 잇는 직선과 (a, b)간의 거리를 구하는 연산을 수행할까 생각했지만, 만약 결과론적으로 점들이 특정 반원안에 속한다면 해당 연산을 n번 수행해야 하기 때문에 오버헤드를 줄이기 위해 마지막 과정으로 뺐다.

추가적으로, 오일러 각 회전을 통해 모든 점의 좌표들을 변환하고 오로지 두 사분면에만 점이 존재하는지 확인하는 방식도 고려했으나, 해당 방식은 전체 점의 집합을 두 번 순회해야 한다는 점과 소수 연산의 오차를 고려했을 때 위 방식보다 비효율적이라고 판단했다.