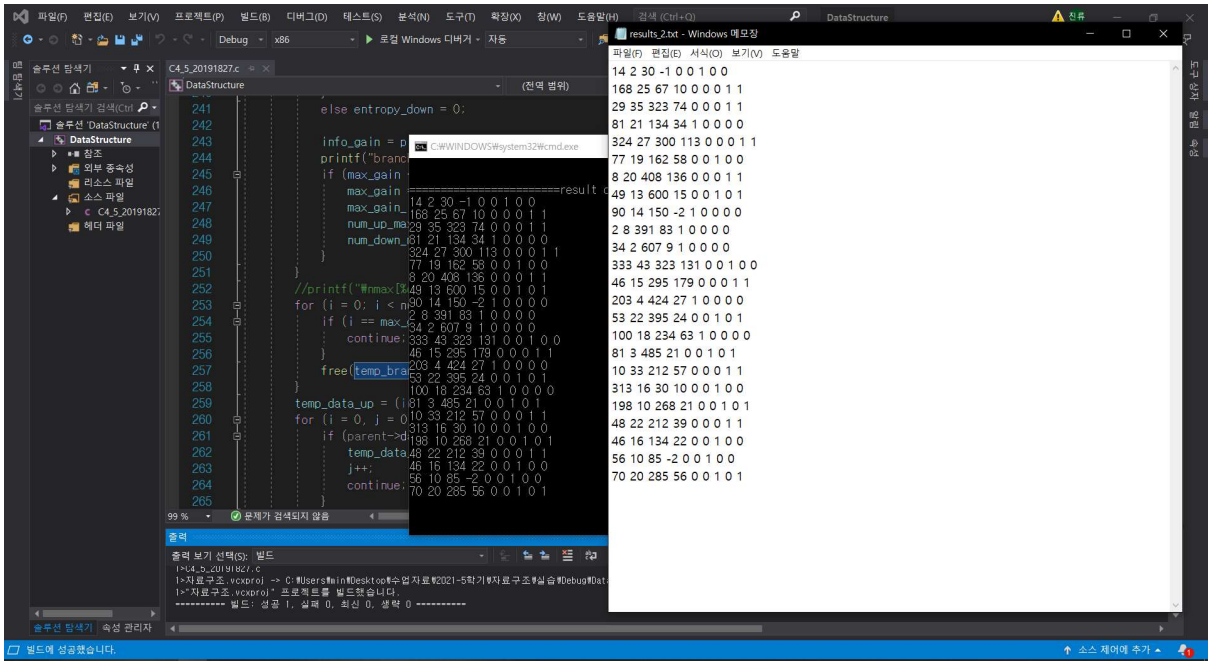

우선 알고리즘을 구현하기 위한 트리의 전반적인 구조를 설계해야 했다.

재귀적 구현을 위해 여러 번의 시행착오를 거쳐, 트리의 구조를 확정했다.

우선 트리의 구성요소애는 왼쪽자식, 오른쪽자식 포인터가 존재하며, 초과, 이하로 분기 완료된 데이터셋, 분기점, 데이터 클래스, 트리 레벨, 사용된 attribute, 데이터 개수로 구성되어 있다.

트리는 헤드노드부터 시작되는데, 헤드노드에는 전체 데이터를 담았다.

가변적인 데이터들을 담기 위해 2차원 배열과 포인터를 적절히 활용하였다.

처음 학습 데이터를 읽어서 헤드노드를 세팅한다. 분기 완료된 데이터셋이 두 개가 붙어있으므로 왼쪽인지 오른쪽인지에 해당하는 방향을 인수로 받아와야 한다.

현 시점에서 가능한 분기점들을 찾아내는 작업을 한다. 이 때 이전에 사용된 attribute는 고려하지 않는다. 각 attribute 마다 정렬한 뒤에 가능 분기점들을 뽑아낸다.

가능분기점들에 대해 게인을 구한 뒤 최적 분기점을 찾아낸다.

엔트로피 계산 중 발생하는 log2(0) 을 해결하기 위해 간단한 함수를 작성하였다.

분기가 완료되면 재귀적으로 계속 반복한다.

종료조건은 한 쪽 데이터만 남거나, 가능 분기점이 없는 경우이다.

학습이 완료된 후에는 메모리를 정리한다. 예외 방지를 위해 매크로를 활용했다.

마찬가지로 재귀적으로 구현하였다.

테스트 데이터를 읽어온 뒤에 저장된 분기점에 맞춰 분류를 시작한다.

학습데이터를 이용해 만든 트리의 말단에 다다르면 클래스를 부여한다.

결과를 출력 파일에 저장한 뒤 메모리를 모두 정리한다.

디버깅을 위해 각 트리를 프린트한다.

거의 5일 이상을 오롯이 본 과제에만 매달렸다. 수많은 시행착오를 거쳤지만 아직도 제대로 만든 것인지 확실하지가 않다. 정확한 정보 습득의 부족으로 세부적인 구현의 미묘한 차이들이 군데군데 발생하였으며, 이를 검토하기 위해 불필요하게 많은 시간이 소모되었다. 무작정 코딩을 시작하기 앞서, 프로그래밍을 위한 정확한 지식을 습득하는 것과 보다 자세하고 명확한 명세서를 확보하는 것이 무엇보다 중요하다는 것을 알게 되었다.