

알고리즘 실습 보고서

-Greedy-

전공 :컴퓨터공학과

분반 : 05반

학번 :201701988

이름 :김수빈

1. 실행 환경

본 실습은 Windows10 64bit, jdk 1.8.0_221, Eclipse EE가 설치된 환경에서 실행되었다

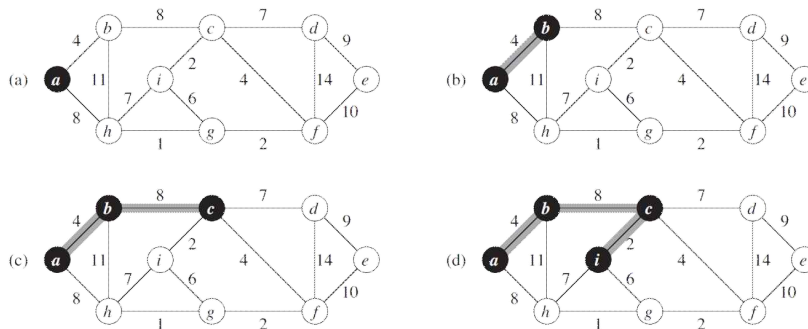
2. 과제 설명

- 출제된 과제에 대한 설명

homework 1

Prim's Algorithm을 통해 가중치의 총합이 최소가 되는 최소 신장 트리를 구한다.

데이터는 다음 그래프를 보며 직접 입력하여 사용한다.



min priority queue를 사용하여 가장 가중치가 낮은 edge를 선택한다.

edge의 탐색 순서를 보이며, 각각의 가중치와 그 총합을 계산한다.

homework 2

data12.txt파일을 읽고 Greedy algorithm을 사용하여 Huffman code를 얻기 위한 Tree를 구축하고, 구축한 Tree에서 Huffman code 값을 얻고 이에 따라 인코딩한 파일과 테이블을 hw12_05_201701988_encoded.txt, hw12_05_201701988_table.txt로 출력한다.

또한 반대로 data12_encoded.txt와 data12_table.txt를 읽고 이로부터 디코딩하여 hw12_05_201701988_decoded.txt로 출력한다.

3. 문제 해결 방법

- 문제를 해결하기 위해 자신이 사용한 방법, 아이디어에 대한 설명

먼저, 그래프를 구현하기 위해 ArrayList<Node>와 함께 HashMap<String, Integer> nodeIndex를 사용하는데, 인덱스로 노드 이름을 사용할 수 없기 때문에 HashMap으로 인덱스를 관리한다.

프로그램을 실행시키기 위한 main문이 있는 testClass.java로 두 개의 코드를 실행시키며, Prim's Algorithm을 실행시키기 위한 데이터 초기화, Huffman coding을 위한 파일 읽기를 수행한다.

#Homework 1

Prim's Algorithm을 통해 MST를 만들기 위해 필요한 클래스는 Node.java, primAlgorithm.java, priorityQueue.java가 있다.

Node 클래스는 그래프 노드와 간선을 나타내며, 연결된 Node connect, connect까지의 가중치 key, 노드의 이름인 String name을 갖는다.

priorityQueue 클래스는 과제에서 요구한 min priority queue이다. extract_min을 통해 최소값

을 삭제하고 반환하며, build_min_heap으로 최소 우선순위로 재정렬한다.

primAlgorithm 클래스에서는 executePrimAlgorithm을 통해 실행한다. 먼저, 시작점 노트의 key 값을 0으로 시작하고 나머지는 Integer.MAX_VALUE로 초기화한다. min priority queue인 Queue에서 extract_min으로 key값이 최소인 노드를 삭제하고 반환한다. 삭제한 노드 u와 Queue 내부의 노드들의 간선 가중치값을 검사하며 기존의 가중치 값보다 작으면 갱신한다. Queue가 빌 때까지 반복한다. 총합은 반복문 전의 int value값을 선언하고 간선을 선택할때마다 가중치값을 더해주고 반복문이 끝나면 출력한다.

Homework 2

Greedy algorithm을 통해 Huffman decoding, encoding을 하기 위해 필요한 클래스는 TreeNode.java, Huffman.java, priorityQueue_forHuffman.java가 있다.

TreeNode 클래스는 트리를 구현하기 위해 만든 것으로 TreeNode left, TreeNode right를 가지며, 중복해 나타난 횟수 key, 노드의 이름인 String name을 갖는다.

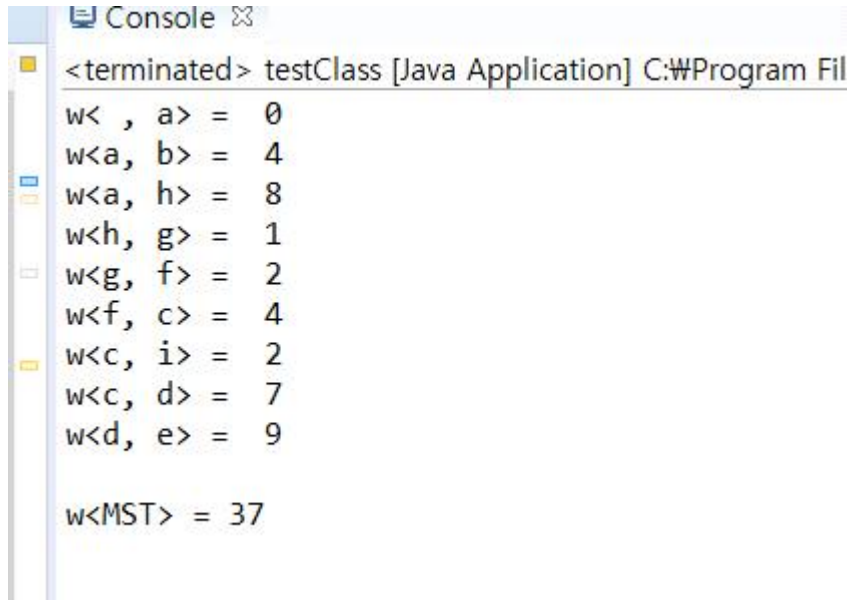
priorityQueue_forHuffman 클래스는 앞서 priorityQueue와 같이 과제에서 요구한 min priority queue이며, TreeNode를 사용하여 구현하기 위해 추가했다. 마찬가지로 extract_min을 통해 중복횟수가 가장 작은 TreeNode를 삭제하고 반환하며, build_min_heap으로 최소 우선순위로 재정렬한다.

Huffman 클래스에서는 생성자에서 문자의 중복횟수를 세어 TreeNode로 분리하고, makeTree에서 priorityQueue_forHuffman을 통해 트리를 구성한다. 가장 적은 횟수로 나타는 TreeNode를 x, y 두 개 뽑아 TreeNode z를 구성하고 Queue에 다시 삽입한다. 반복문이 끝나면 Queue.extract_min을 통해 루트노드를 반환한다. makeTable 메소드에서는 startNode로부터 각 노드까지의 경로를 저장하는 테이블을 구성한다. 반복문을 통해 모든 노드에 대해 startNode와의 경로를 HashMap에 저장한다. 경로는 get 메소드에서 반환받아 사용하며, get은 재귀를 통해 왼쪽 TreeNode에서 찾으면 경로에 0을 덧붙이고 오른쪽 TreeNode에서 찾으면 경로에 1을 더해 경로를 반환한다. encodingFile 메소드는 만든 HashTable 테이블을 통해 인코딩하여 파일을 출력하는 메소드다. 위와 같은 과정을 encoding메소드에서 다루며, makeTree->makeTable->encodingFile 순으로 메소드를 실행한다.

마찬가지로 디코딩 과정도 decoding 메소드가 관리하며 File을 읽어 디코딩할 문장을 저장하고, 테이블을 읽어 HashMap을 구성한다. decodingFile 메소드를 통해 디코딩할 문장을 문자열 하나씩 디코딩 테이블을 보며 바꾼다.

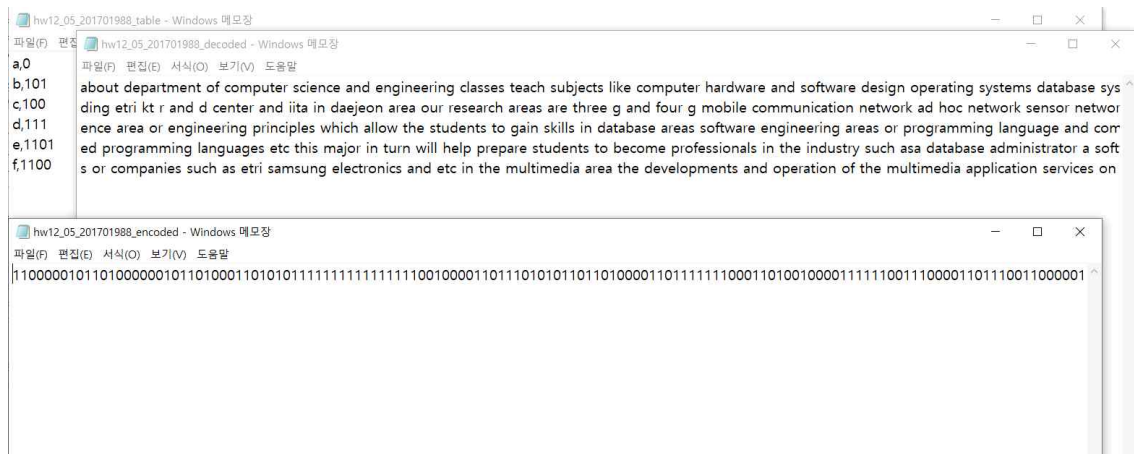
결과 화면

#homework1 :



```
<terminated> testClass [Java Application] C:\Program Fil  
w< , a> = 0  
w<a, b> = 4  
w<a, h> = 8  
w<h, g> = 1  
w<g, f> = 2  
w<f, c> = 4  
w<c, i> = 2  
w<c, d> = 7  
w<d, e> = 9  
  
w<MST> = 37
```

#homework2 :



5. 느낀점 및 고찰

자료구조 시간 때 배운 내용을 활용하여, 큰 어려움은 없었습니다.