알고리즘 HW 2 보고서

- Graph Pattern Matching Challenge -

ENGINEERING
College of Engineering Seoul National University
서 울 대 학 교 공 과 대 학

과 목 명	알고리즘
담 당 교 수	박근수
학 과	자유전공학부 / 컴퓨터공학부
학 번	2017-12146 / 2018-14745
이 름	김태정 / 이준영

1. 작동 환경

김태정

- 운영체제: Ubuntu 20.04.2 LTS (Windows 10 Home에서 WSL Version 1 사용)
- 사용언어: Microsoft Visual C++ 2019

이준영

- 운영체제: Windows 10 Pro / Ubuntu 20.04.2 LTS (WSL2)
- 사용언어: JetBrains CLion 2021.1

2. 프로그램 작동 방법

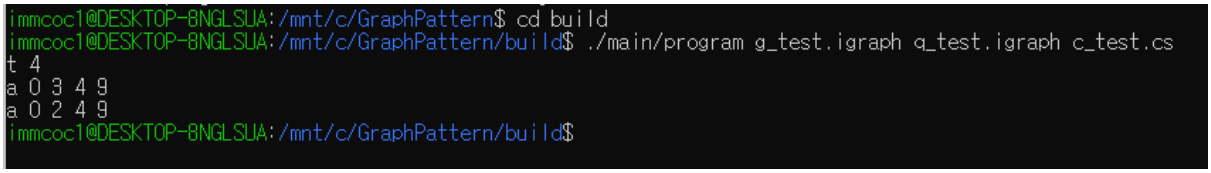
해당 프로그램의 작동 방법은 https://github.com/SNUCSE-CTA/Graph-Pattern-Matching-Challenge에 등록된 기본 프로그램과 동일하다. 우분투 콘솔창에서 다음의 명령어들을 실행한다.

```
mkdir build
cd build
cmake ..
make
```

이후 생성된 build 폴더에 입력 값으로 사용할 data graph 파일, query graph 파일, candidate set 파일을 저장하고, build 폴더에서 다음 명령어를 수행한다. 파일명 입력 시 파일의 확장자까지 전부 포함하여 입력한다.

./main/program <data graph 파일명> <query graph 파일명> <candidate set 파일명>

이후 해당 프로그램은 콘솔창에 가능한 embedding들을 출력하며, 출력한 embedding이 총 100,000개가 될 경우 출력을 종료한다.



3. 알고리즘 설명

1) 개요

해당 프로그램은 DAF 알고리즘을 이용하여 embedding을 찾는 프로그램으로, 'BuildDAG'와 'Backtracking' 두 단계를 거친다. Candidate set를 만드는 과정은 과제 조건상 생략한다.

'BuildDAG'는 주어진 query graph를 바탕으로 acyclic directed graph 'DAG'와 DAG의 모든 edge를 반대방향으로 바꾼 'DAG_invert'를 만드는 과정이다. 해당 프로그램에서는 DAG를 각 원소가 vector인 vector로 구성하였다. DAG vector의 i번째 원소 DAG[i]에 해당하는 vector는, ID가 i인 query vertex에서 향하는 다른 query vertex들의 ID를 원소로 가진다. DAG_invert[i]는 반대로 ID가 i인 query vertex로 향하는 다른 query vertex들의 ID를 원소로 가진다. Root에 해당하는 vertex(들어오는 edge가 없는 vertex)는 (해당 query에 맞는 candidate set의 크기)/(해당 query vertex와 이어진 다른 vertex의 개수)가 가장 작은 query vertex로 선정한다.

'Backtracking'은 'BuildDAG'단계에서 만든 DAG와 DAG_invert를 기준으로, query graph에 맞춰 embedding을 재귀적으로 확장하며 embedding을 완성시키는 단계이다. Embedding은 배열로 구성하였으며, i번째 원소는 ID가 i인 query에 해당하는 candidate vertex의 ID이다. 재귀적으로 함수를 호출하며 연산하는 도중 embedding_size가 query vertex의 수와 같아지면 embedding이 완성되었다는 뜻이므로 해당 embedding을 출력한다. 이 프로그램에서는 Backtracking 단계를 find_Embedding 함수가 수행한다.

2) Backtracking 세부 설명

find_Embedding 함수의 주요 인자로는 query_v, data_v, embedding[], can_visit[], embedding_size가 있다. query_v와 data_v는 현재 다룰 query와 이와 매칭될 data vertex를 의미한다. embedding[]는 현재까지 완성된 partial embedding을 담고 있는 배열이다. can_visit[]는 다음 방문할 query를 고르기 위해 이미 방문한 query와 방문 가능한 인접 query를 기록해두는 배열이다. embedding_size는 embedding에 실제로 채워진 원소의 개수이다.

어떤 partial embedding을 확장하기 위해, query u와 이와 매칭될 data vertex v를 추가하기 위해서는 다음 조건이 성립해야 한다: DAG 상에서 query u의 부모 원소를 u_e라 할 때, embedding 안에 모든 u_e query들이 채워져 있고, 그들과 매칭된 data vertex들 모두 v와 이웃한다. 이를 구현하기 위해 find_Embedding의 마지막에서 DAG_invert를 통해 DAG 상의 부모 query를 찾고, embedding에서 부모 query와 매치된 data vertex 중 현재 다룰 data vertex와 이웃하지 않은 vertex가 있으면 return을 통해 해당 함수를 벗어난다.

한편, embedding이 injective 조건을 만족하도록 유지하기 위해선 새로 들어올 data vertex가 기존 embedding에 들어있던 data vertex와 같아서는 안 된다. 새로 들어올 data vertex와 같은 data vertex가 embedding 안에서 발견될 경우 위의 경우와 마찬가지로 해당 함수를 벗어난다. 이 조건까지 만족하였을 경우 embedding[query_v]에 data_v를 저장하며 해당 쌍을 embedding에 본격적으로 포함시킨다. 또한 can_visit[query_v]의 값을 수정하여 해당 query가 이미 방문 되었음을 기록한다.

embedding이 완성되었을 경우 해당 embedding을 출력한 후 return하고, 프로그램 시작 후 찾아낸 embedding의 개수가 100,000개이면 return true를 통해 다른 재귀가 일어나지 않게끔 한다. return이 일어나지 않았을 경우 다음 탐색 대상을 찾는 단계에 들어선다.

3) 다음 탐색 대상 선택 방법 (Matching order)

이 프로그램의 matching order는 candidate-size order를 따른다. 즉, 현 partial embedding에서 확장 가능한 query vertex 중에서 그 query에 해당하는 candidate vertex의 수가 가장 적은 query vertex를 다음 방문 대상으로 선택한다.

DAG를 통해 현재까지 만들어진 partial embedding의 query vertex들과 이어질 수 있는 다른 query vertex들을 찾아내고, 이들이 방문 가능한 후보 query임을 can_visit[] 배열에 표시한다. 표시된 query들 중 자신에 해당하는 candidate set의 크기가 가장 작은 query vertex가 다음으로 방문할 대상(next_query)으로 지정된다. next_query와 이와 매칭되는 candidate set의 모든 data vertex에 대하여 재귀적으로 find_Embedding를 호출함으로써 다음 vertex 선택이 완료된다.