2021-1 SNU CSE Algorithms

알고리즘 Subgraph Matching Challenge Report

자유전공학부 2017-19213 이정훈

자유전공학부 2019-11618 김진형

과제 환경 및 컴파일 / 실행 방법

해당 과제의 환경(컴파일러, OS 환경), 컴파일 방법, 실행방법은 다음과 같습니다. 기본적으로 제공된 Baseline Code의 `backtracking.cc` 파일과 `backtracking.h` 파일을 수정하는 방식으로 프로젝트를 진행했습니다.

OS: Windows Subsystem Linux(WSL), Ubuntu 20.04.2 LTS

컴파일러: C/C++: g++(GCC) 9.3.0

컴파일 방법: README.md의 방식과 동일

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

실행 방법: README.md의 방식과 동일

```
cd build
./main/program <data graph file> <query graph file> <candidate set file>
```

Candidate set 만드는 방법: README.md의 방식과 동일

```
./executable/filter_vertices <data graph file> <query graph file>
```

함수 설명

```
void Backtrack::printEmbedding(const std::vector<Vertex> &embedding)
/**
 * @brief Prints embedding to stdout.
 * @param embedding complete embedding.
 * @return None
 */
```

embedding이라는 std::vector<Vertex>를 받아서 stdout에 출력하는 함수입니다. 성능 개선을 위해 출력은 std::cout이 아닌 printf()를 사용했습니다.

```
bool Backtrack::isEmbedding(const std::vector<Vertex> &embedding, const Graph &dat
a, const Graph &query)
/**
 * @brief Return true if given embedding satisfies embedding conditions, otherwise
 return false.
 * @param embedding embedding to be tested.
 * @param data data graph.
 * @param query query graph.
 * @param cs candidate set.
 * @return bool
 */
```

embedding이라는 std::vector<Vertex>, data graph, query graph, candidate set을 받아서 DAF 논문에 나온 embedding의 세 조건을 만족하는지 판별하는 함수입니다. 세 조건 중 첫 두 조건은 backtracking()을 구현하는 과정에서 이미 확인해 이 함수 안에는 세번째 조건만을 확인하는 로직이 포함되어 있습니다.

```
void Backtrack::buildDAG(const Graph &G, const Graph &q)
/**
 * @brief build DAG(q_D) from query
 * @details 메서드의 맨 아래에 만들어진 DAG(adjacency_list 형태)를 출력하는 코드가 주
석처리 돼있습니다.
 * @param G data graph
 * @param q query graph
 * @return None
 */
```

data graph와 query graph를 받아 DAF 논문에 기재된 것처럼 BFS 탐색 후 discover time 순서로 query graph를 direct해 DAG(Directed Acyclic Graph)를 만드는 함수입니다. 이 과정에서 BFS를 구현하기 위해 <queue> 라이브러리를 사용했습니다.

```
void Backtrack::transposeDAG(std::vector<Vertex>* &adj, std::vector<Vertex>* &tran
spose)

/**

 * @brief function to get Transpose of a graph taking adjacency
 * @param adj given graph to reverse
 * @param transpose transposed graph
 * @return None
 * */
```

buildDAG(…)에서 만든 DAG를 받아 모든 edge를 거꾸로 돌린 transposed DAG를 만드는 함수입니다. 추후 backtracking 과정 중 extendable() 구현을 위해 필요한 함수입니다.

```
int Backtrack::C_ini(const Graph &G, const Graph &q, Vertex u)
/**
 * @brief get the size of initial CS of query node u.
 * @details v is in C_ini(u), if label(v)=label(u) && degree(v)>=degree(u)
 *          v: node of data graph, u: node of query graph
 * @param G data graph
 * @param q query graph
 * @param u node of query graph
 * @return int
 */
```

data graph, query graph, Vertex u를 받아 DAF 논문에 기재된 C_ini() 값을 계산하는 함수입니다. C_ini() 값은 buildDAG()에서 DAG의 root vertex를 정해주기 위해 필요한 값입니다.

```
void Backtrack::backtracking(const Graph &data, const Graph &query, const Candidat
eSet &cs)
/**
 * @brief Backtrack. See <Algorithm 2> in the paper.
 * @details M, visited[]는 class 변수 (parameter X)
 * @return None
 */
```

data graph, query graph, candidate set를 받아 본격적인 backtracking 과정을 수행하는 함수입니다. DAF 논문의 Algorithm2를 구현한 함수입니다. 이 중 partial embedding을 저장하는 M과 방문된 query vertex를 기록하기 위한 visited[]는 class member variable이라 따로 함수의 인자로 주고 받지 않고 static하게 관리하였습니다.

```
Vertex Backtrack::extendable(const Graph &data, const CandidateSet &cs)
/**
 * @brief return the extendable vertex of current M, with mininum C_m
 * @param data data graph
 * @param cs candidate set
 * @details M, visited[]는 class 변수 (parameter X)
 * @return Vertex
 */
```

data graph, candidate set을 받아 현재의 partial embedding M에서 extendable한 vertex 중 C_m, 즉 candidate size가 최소인 vertex를 반환하는 함수입니다.Candidate size가 같은 경우 query vertex 숫자가 낮은 순으로 반환하도록 구현되어 있습니다.

```
std::vector<Vertex> Backtrack::C_m(Vertex u, const Graph &data, const CandidateSet
 &cs)
/**
```

```
 * @brief returns C_m(u), set of extendable candidates of u regarding partial embe
dding M
 * @details see <ch15. graph_pattern_matching> p19.
 * @return std::vector<Vertex>
 */
```

DAF 논문에 구현된 candidate size C_m을 구현한 함수입니다. 아래의 N_u를 활용해 실질적으로 해당 query vertex에 대해 유효한 candidate size를 반환합니다.

```
std::vector<Vertex> Backtrack::N_u(Vertex u, Vertex v_p, const Graph &data, const
CandidateSet &cs)
/**
 * @brief N sup(u_p) sub(u)를 구현. (see <ch15. graph_pattern_matching> p19.)
 * @details return set of vertices v, those adjacent to v_p in G such that v in C(
u)
 * @param u vertex in query
 * @param v_p = M(u_p)
 * @return std::vector<Vertex>
 */
```

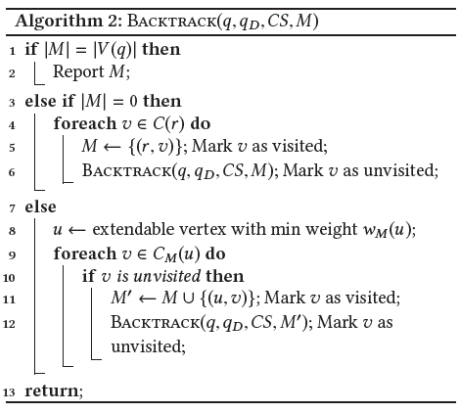
C_m()을 구현하기 위해서는 v_p와 이웃하는 data graph의 vertex들을 반환하는 N_sup(u_p)_sub(u) 함수가 필요한데, 이를 구현한 함수입니다.

Adaptive Matching Order - Choose Query Vertex with Minimum Candidate Size($|C_m(u)|$)

backtracking.cc의 backtracking()에서 백트래킹하는 순서를 구현하는데에는 DAF 논문에 소개된 Candidate-size order를 사용했습니다. 이를 위해서는 $|C_m(u)|$를 계산하기 위해 $N_u^{up}(v)$를 구현해야 하기에 이 또한 N_u() 함수로 구현했습니다. 이 과정에서 u의 parent를 구해야 하기에 buildDAG에서 나온 DAG를 transpose한 DAG가 필요했고, transposeDAG() 함수도 구현했습니다.

Backtracking 구현 방법

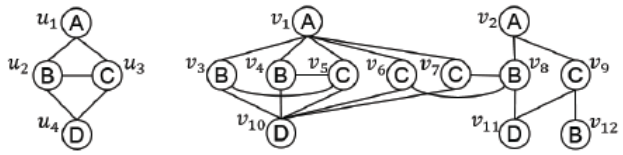
backtracking.cc의 backtracking()에서는 DAF 논문에서 소개된 Algorithm 2: BACKTRACK() 함수를 유사하게 구현하는 방식으로 백트래킹을 진행하였습니다. 대신 원래 함수의 인자로 전달되던 M을 `std::vector<std::pair<Vertex, Vertex>>` class member variable로 정의해 그때그때 update하고, v의 방문 여부를 확인하는 `std::map<Vertex, bool>` visited을 정의해 그때그때 update 하는 방식으로 구현했습니다. 또한 |M| !=0인 경우, for loop을 다 돌았을 때 다른 가지로 backtracking 하는 논리가 해당 pseudocode



에는 빠져 있어 실제 코드에서는 구현했습니다.

간단한 예시 input으로 프로그램의 동작 확인(example running)

해당 로직을 구현하는 과정에서 프로그램이 제대로 작동하는지 알아보기 위해 Graph-Pattern-Matching ppt에 소개된 간단한 예시를 활용했습니다. 이를 위해 직접 exdata.igraph, exquery.igraph, excandidate.cs 파일을 만들어 실행해봤습니다.



(a) Query graph q (b) Data graph G

```
./main/program ../data/exdata.igraph ../query/exquery.igraph ../candidate_set/exca
ndidate.cs
```

```
t 4
```

```
a 0 2 4 9
```

```
a 0 3 4 9
```

또한 비교적 embedding이 적게 나오는 lcc_hrpd_n1의 데이터를 실행한 결과는 다음과 같습니다.

```
./main/program ../data/lcc_hprd.igraph ../query/lcc_hprd_n1.igraph ../candidate_set/lcc_hprd_n1.
cs
```

```
t 50
```

```
a 937 2330 1293 161 303 1126 541 1106 5008 3088 1469 221 5672 5671 5670 5669 211 4723 1684
1376 110 687 111 730 131 1168 1379 2806 2805 684 685 4455 4468 123 5138 0 5137 1517 4700 404
160 831 280 1932 1929 179 8785 69 43 1995
```

```
…(중략)
```

```
a 280 4107 4106 1468 303 1126 541 1106 5008 3088 1469 221 5672 5671 5670 5669 211 4723 1684
1376 110 687 266 2599 131 1168 1379 2806 2805 684 685 4455 4468 123 5138 0 5137 1517 4700
404 160 831 5508 1932 1929 179 8785 915 43 1995
```

참고문헌

[1] Myoungji Han, Hyunjoon Kim, Geonmo Gu, Kunsoo Park, and Wook-Shin Han. 2019. Efficient Subgraph Matching: Harmonizing Dynamic Programming, Adaptive Matching Order, and Failing Set Together. In Proceedings of the 2019 International Conference on Management of Data (SIGMOD '19). Association for Computing Machinery, New York, NY, USA, 1429–1446. DOI:https://doi.org/10.1145/3299869.3319880