I. Matching Order

초반에 저희 팀에서 구현해보고자 했던 Matching Order 알고리즘은 다음과 같았습니다.

Algorithm 5: GenerateMatchingOrder

Input: a query graph q, a data graph G and the candidate set C
Output: a matching order π, a pivot dictionary $\mathcal{P}$ and the backward neighbors BN_q^π

1 **begin**
2 $q^w \leftarrow GenerateWeightedGraph(q, G, C)$;
3 $V_C \leftarrow \{u \in V(q) | u.core \geqslant 2\}, V_{NC} \leftarrow V(q) - V_C$;
4 $\mathcal{P} \leftarrow \{\}, w^*[u] \leftarrow |V(G)|$ for all $u \in V(q)$;
5 Set $BN_q^\pi(u)$ to empty for all $u \in V(q)$, set UN to empty;
6 $u^* \leftarrow arg\min_{u \in V_C} \frac{|u.C|}{u.core}, \pi \leftarrow (u^*)$;
7 **foreach** $u \in N(u^*) - \pi$ **do**
8 $BN_q^\pi(u) \leftarrow BN_q^\pi(u) \cup \{u^*\}$;
9 **if** $w(u^*, u) \leqslant w^*[u]$ **then**
10 $w^*[u] \leftarrow w(u^*, u), \mathcal{P}[u] \leftarrow u^*$;
11 Add u to UN if $u \notin UN$;
12 **while** $|\pi| < |V_C|$ **do**
13 $u^* \leftarrow arg\min_{u \in UN \cap V_C} \frac{w^*[u]}{|BN_q^\pi(u)|^2}$;
14 Add u^* to π, remove u^* from UN;
15 Same as Lines 7-11;
16 **while** $|\pi| < |V(q)|$ **do**
17 $u^* \leftarrow arg\min_{u \in UN \cap V_{NC}} \frac{w^*[u]}{d^2(u)}$;
18 Add u^* to π, remove u^* from UN;
19 Same as Lines 7-11;
20 **return** $(\pi, \mathcal{P}, BN_q^\pi)$;

Subgraph Matching with Effective Matching Order and Indexing(Shixuan Sun and QiongLuo) 라는 논문에 실린 Generate Matching Order라는 함수를 구현해보고자 했습니다.

간략하게 설명하자면, 해당 알고리즘은 각 vertex의 core value와, 각 vertex의 candidate size를 사용해 Matching order 시작점을 고른 후,

1. core_structure 에 있는(u.core>=2)인 점들을 우선시
2. backward_neighbor(vertex 보다 Matching order가 작은 것 중 vertex와 Edge를 지닌 것) 의 개수가 큰 것일수록
3. estimated search breadth가 작을수록(pseudo code에서 w*[u]에 해당)

Matching order를 더 앞으로 배정하는 전략을 사용합니다.

하지만 graph theory에 대한 이해도가 부족한 탓인지, 첫번째, 두번째 while loop에서 UN ∩ Vc(Vnc)가 공집합이 되는 현상을 해결하지 못해, 부득이하게 해당 전략을 이번 Challenge에서 사용하지는 못하였습니다.

대신, 이 알고리즘을 탐구해보며 얻은 인사이트를 바탕으로 몇몇 매칭 오더를 시도하였습니다.

시도 1. DAG ordering
Query 그래프로부터 DAG를 생성하여, DAG를 BFS로 traverse 하면서 생긴 ordering을 그대로 Matching order에 활용하였습니다.

시도 2. Min Candidate Size
위의 알고리즘에서 weighted graph를 만들 때나, Matching Order의 첫번째 원소를 구할 때 Candidate Size가 중요하게 다뤄진다는 점에서, Candidate Size를 기준으로 Matching order를 만들었습니다.

시도 3. Min Candidate Size/ core value
An O(m) Algorithm for Cores Decomposition of Networks(Batagelj) 논문을 참고하여 그래프에서 각 점의 core value를 구하는 메소드를 만들었습니다.
그 후, Candidate Size 가 작을수록, core_value가 클수록 더 우선시되는 Matching order를 가질 수 있도록 Candidate Size/core value를 기준으로 Matching order를 만들었습니다.

시도 4. Min Weight/degree^2
Query 그래프에 대해 weighted graph를 다음 규칙에 따라 생성하였습니다.
Edge (u, u')에 대하여 w(u, u')는 m/candidate_size(u), w(u', u)는 m/candidate_size(u')입니다.
이때 m은 data graph에서 u.C와 u'.C의 Vertex들 사이에 존재하는 Edge 의 개수입니다.
V에 대해 extendable vertice가 v1, v2, v3 일 때, w(v1/v2/v3, v)/degree(v1/v2/v3)^2을 평가지표로 사용해 가장 작은 값을 지닌 vertex를 고릅니다.

시도 1, 2, 3, 4 모두 tree-based approach로, root는 (candidate size / degree)가 가장 작은 vertex로 설정하여, extendable vertices 중 하나를 고를 때 해당 전략들을 사용했습니다.

II. Backtracking
Challenge의 skeleton code를 바탕으로 PrintAllMatches만을 호출하면 모든 출력이 완료됩니다. PrintAllMatches에서는 DAG를 구성하면서 matching order를 결정한 뒤, 재귀적으로 작동하는 함수 FindMatches를 호출합니다. FindMatches는 matching이 완료된 vertex의 수를 level로 하여 backtracking을 수행합니다.
특정 level에서 FindMatches는, order에 맞는 vertex u의 candidate set의 각 원소 v를 u의 matching으로 가정한 후 다음과 같은 조건 2가지를 검사합니다.

1. Vertex v가 이전에 matching된 vertex 중에 존재하지는 않는가?
이전에 matching된 vertex 중에 v가 존재하는 경우, embedding의 injective 성질을 만족하지 못하게 됩니다. data에서 candidate element vertex는 matching이 유효할 경우

data_matched_vertex_[vertex]라는 vector에 true를 check하고 다음 recursion을 시행한 후 false를 check합니다. 만약 candidate element v가 data_matched_vertex_[v] == true를 만족할 경우, v는 이전에 matching이 수행된 vertex이므로 (u, v)는 유효하지 않은 matching이며, u의 candidate set에서 다른 vertex v를 탐색합니다.

2. Vertex v가 이전에 matching된 vertex와 연결 관계를 만족하는가?
Query DAG에서 vertex u를 가리키던 vertex는 이미 matching이 완료된 vertex입니다. (임의의 u가 extendable하도록 matching order를 정해야 하며, 실제로도 그렇게 정하였습니다.) data graph에서 이들의 matching 대상이 v와 연결되어 있어야 「$(M(u), M(u')) \in E(G)$ for every $u, u' \in E(q)$」 조건을 만족하게 됩니다. v와 연결되지 않은 것이 하나라도 있다면 즉시 u의 candidate set에서 다른 vertex v를 탐색합니다.

위의 두 조건을 모두 만족할 경우 (u, v) 매칭이 유효하다고 판단하고 다음 level을 진행합니다. 이와 같은 방법으로 level이 query의 vertex 수와 같아지면 모든 vertex의 matching이 끝난 것이며, 이를 출력하면 하나의 embedding을 찾은 것입니다.

III. 프로그램 실행방법 및 Environment

프로그램은 C++ 언어를 사용하여 짜여져 있고,
프로그램 실행방법은 해당 Challenge의 github README 에 나와있는 것과 같이

Main program:

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

으로 빌드하여 실행하면 됩니다.