알고리즘
Graph pattern matching challenge 보고서
홍영서, 윤성원

백트래킹 과정 설명

```
void Backtrack::SubgraphSearch(const Graph &data, const Graph &query,
                               const CandidateSet &cs, std::vector<std::pair<Vertex, Vertex>> match){
  if(match.size()==query.GetNumVertices()){
    PrintAllMatches(query, match);          1

  } else {
    Vertex nextQueryVertex = NextQueryVertex(query, cs);  2

    std::vector<Vertex> candidates = GetCandidates(cs, nextQueryVertex);

    for(Vertex v : candidates){
      std::pair<Vertex, Vertex> pair = {nextQueryVertex, v};
      if(IsExtendable(pair, match, data, query, cs)==true){  3
        match.push_back(pair);


                                          4
        UpdateState(query, nextQueryVertex);
        SubgraphSearch(data, query, cs, match);
        match.pop_back();


        RestoreState(query, nextQueryVertex);
      }
    }
  }
}
```

0. main.cc에서 호출하는 함수는 SubgraphMain() 함수인데, Preprocess() 함수를 불러서 백트래킹 과정에 필요한 변수를 만드는 역할을 한다. 총 60초(1분) 이후에 프로그램이 종료될 수 있도록 시간을 쟀다. 매칭된 모든 경우들은 match라는 이름의 vector에 저장해두어 가능한 모든 경우들을 방문하도록 했다.

1. 매칭이 완료된 (u,v)를 원소로 하는 vector(match)를 받아 함수가 시작될 때 vector의 원소 개수와 query의 vertex 개수를 비교하여 같으면 전체 매칭이 끝났다는 의미이므로 출력한다. 같지 않으면 매칭을 진행하는데, NextQueryVertex() 함수에서 다음 방문할 query vertex를 고른다. 그 vertex의 candidate에 대해 조건을 확인하고 가능한 경우라면 match에 포함시키고, 재귀적으로 자신을 호출한다. 재귀함수가 끝나면 return된 후 match.pop_back()을 호출함으로써 방금 다녀온 vertex를 backtrack할 수 있도록 한다.

2. Matching order - NextQueryVertex 함수를 이용해 query graph에서 다음에 매칭될 vertex를 찾는다.

이 때 함수 작동 방식은 다음과 같다.

1) 아직 매칭 되지 않은 vertex중에 parent들은 모두 매칭 된 vertex들을 찾는다. 이번 과제의 input은 undirected graph이므로 여기서 말하는 parent는 어느 한 vertex를 기준으로 삼았을 때, 자기 자신보다 앞서 자신을 가리키고 있는 vertex가 몇 개인지를 나타내는 변수이다. 곧 root는 여러 개일 수 있으며, 가능한 후보 next query들을 vector에 담아

보관하도록 한다.

2) 가능한 next query vertex 중 candidate 수가 가장 적은 vertex를 택한다.

위와 같이 next query vertex가 정해지면, 그 query vertex의 candidate을 input으로 받은 cs에서 확인하여 각 candidate에 대해 IsExtendable 함수를 호출하도록 한다.

3. 매칭될 vertex의 각 candidate들에 대해 IsExtendable 함수를 이용해 매칭 가능한지 확인한다.

```
bool Backtrack::IsExtendable(std::pair<Vertex, Vertex> pair, std::vector<std::pair<Vertex, Vertex>> match,
                             const Graph &data, const Graph &query, const CandidateSet &cs){
  for(auto it=match.begin(); it!=match.end(); it++){
    std::pair<Vertex, Vertex> p = *it;                                    1
    if(pair.second == p.second) return false;  // 중복 여부 확인
  }
  std::vector<Vertex> parentID = query.GetParentID(pair.first);
  for(size_t i=0; i<parentID.size(); i++){
    for(size_t j=0; j<match.size(); j++){                                 2
      if(match[j].first==parentID[i]) {
        if(!data.IsNeighbor(match[j].second, pair.second)) return false;
        break;
      }
    }
  }
  return true;
}
```

이 때 함수 작동 방식은 다음과 같다.

1) 이미 매칭되었으면 false를 리턴한다. (match vector에서 확인한다.)

2) 매칭된 parent들(data graph vertex)과 해당 candidate이 neighbor로 연결되어 있는지 확인한다. 만일 neighbor가 아니라면 해당 candidate은 잘못되었으므로 false를 리턴한다. 이전에 matching된 parent들은 global variable로 선언한 map에 각 ID를 저장하고 있어 그것을 이용한다.

3) 위 조건들을 모두 만족한 경우에는 true를 리턴하도록 한다.

4. UpdateState()에서는 인자로 들어온 vertex가 matching되었다는 표시를 하고, RestoreState()에서는 matching되지 않았다고 표시하도록 한다. 그럼으로써 backtracking을 진행할 때 여러 candidate들에 대하여 다시 backtracking할 수 있도록 한다.

5. 매칭이 가능한 candidate들은 match vector에 넣고 recursive하게 백트래킹을 진행한다.

실행 환경과 실행 방법 설명

1. 실행 환경 - window, c++

2. 실행 방법 - cmake로 컴파일 후

program.exe [data graph] [query graph] [candidate] 로 실행한다.