Graph-Pattern-Matching-Challenge Report

2019-12731 강지혁
2016-19620 정대용

Github Repository : https://github.com/GanziDaeyong/algorithm_challenge

1. 알고리즘 특징

본 알고리즘은 Matching order로 candidate-size ordering을 사용한다. Path-size order에 비해 간명하면서도 충분히 좋은 성능이 보장되기 때문에 이를 택하였다. 탐색 시에는 mapping이라는 map을 활용하여 동적 프로그래밍 방식으로 진행하였으며, 백트래킹의 경우, stack을 핵심 구조로 하여, 이전 분기까지 연속적으로 거슬러 올라가는 DFS 방식으로 진행하였다. 이와 관련한 자세한 내용은 아래 [2. 알고리즘 구조 설명]과 코드 주석을 참고하길 바란다.

Query graph의 경우, DAG과 같은 별도의 구조 없이 알고리즘을 구현하였다. Root를 설정한 후, candidate-size order로 순회하는 것 자체가 그래프에 방향성을 부여하는 것이기 때문에 DAG와 동일한 효과를 갖는다. 구체적으로는, 인접한 vertex 중 이미 mapping에 존재하는 vertex를 부모로, 그렇지 않은 vertex를 자식으로 판단하는 방식으로 구현하였다. 이러한 방식은 직접 DAG에 접근하는 것에 비해 시간복잡도는 크지만, build-DAG라는 일련의 과정을 생략할 수 있고, DAG가 저장될 메모리를 절약할 수 있으며, 간단한 원리로 작동하기에 확장성이 뛰어나다는 이점이 있다.

2. 알고리즘 구조

다음은 본 알고리즘의 주요 멤버들이다.

mapping
map 구조이며, 맵핑에 성공한 query vertex와 data vertex를 pair로 묶어 보관한다.

EV
set 구조이며, 부모가 mapping에 존재하여 extendable 한 query vertex를 보관한다.

stack
stack 구조이며, 아직 맵핑에 성공하지 못한 query vertex, data vertex의 pair를 보관한다.

credit
bool 자료형이며, 진행과 백트래킹 여부를 나타내는 지표로 사용된다.

본 알고리즘은 아래의 네 메서드로 이루어진다.

PrintAllMatches()

그래프 탐색과 백트래킹이 이루어지는 중심 메서드이다. 해당 메서드는 네 부분으로 구성된다. Initialization Part에서는 mapping을 포함한 정적 멤버들과 변수들을 초기화한다. Main Part는 다시 동적 프로그래밍을 통해 맵핑을 진행해나가는 Proceeding Part와 맵핑 실패 시 백트래킹하는 BackTracking Part로 구분된다. Proceeding Part에서는 완성된 맵핑을 출력하고, 완성되지 않은 경우 EV를 확장하며 맵핑을 계속 진행한다. 이 과정에서 이미 충족된 맵핑 쌍은 mapping에 보관되며, 이후 탐색할 맵핑은 이에 기반하므로 하므로 동적 프로그래밍의 요소를 충족한다. 가능한 Extendable vertex를 찾지 못하면, credit을 false로 설정하고 다음 시행 시 Backtracking Part로 진행된다. Extendable vertex를 찾으면, 그 pair를 stack에 삽입하고, 이는 후에 맵핑되어 mapping에 쌓이게 된다. BackTracking Part에서는 stack 구조를 활용하여 마지막으로 맵핑된 query vertex(curr)까지 pop()을 통해 지속적으로 백트래킹한다. 이때 stack과 mapping, EV 전부 백트래킹이 종료되는 시점으로 복구(roll back)된다. 동시에 credit을 true로 설정하여 다음 시행 시 다시 동적 프로그래밍의 방식으로 맵핑을 진행해 나간다. 정리하면, mapping을 활용한 동적 프로그래밍 방식으로 탐색을 진행해나가고, 실패할 경우 stack을 이용한 DFS 방식의 백트래킹을 진행하는 로직을 갖추고 있다.

tryMap(Vertex u , Vertex v)

Query vertex u와 data vertex v가 맵핑 가능한지 확인하는 보조 메서드이다. 맵핑 가능한 경우 true를 리턴하며, 불가능한 경우, 가령 u나 v가 이미 mapping 내에 존재하거나, u의 부모 vertex와 맵핑된 data vertex와 v가 연결되어있지 않으면 false를 리턴한다.

GetExtendable()

Extendable한 query vertex와 extendable candidates를 찾는 보조 메서드이다. Candidate size가 가장 작은 query vertex와 그에 대응하는 extendable candidates를 pair로 묶어 리턴한다.

GetCm(Vertex u)

Query vertex u의 CS 중 맵핑 가능한 vertex들을 찾는 메서드이다. 찾은 vertex들을 배열에 담아 리턴한다.

3. 탐색 결과

읽어주세요

각 matching 결과마다 맨 뒤에 공백 (" ")이 하나 출력됩니다. 이 점 채점 시 양해 부탁드립니다.

대부분의 경우에서 subgraph를 올바르게 탐색하였다. 문제 조건에 부합하게, 각 그래프당 최대 10만개까지의 결과를 출력한다. 알고리즘 구조 상, 크기가 작거나 비교적 간단한 형태의

그래프의 경우 수 ms 수준으로 매우 준수한 성능을 보였으며, 크기가 크거나 복잡한 형태의 그래프의 경우에도 최대 8000ms, 즉 8초를 넘지 않는 성능을 보였다. 상기 github 링크 /result 경로에서 출력 결과 또한 확인해볼 수 있으며(.out 확장자), 아래는 예시로서 그 중 하나를 캡처한 것이다.

```
t 50
a 30 4107 4106 1468 303 1126 541 1106 5008 3088 1469 221 5672 5671 5670 5669 211 4723 1684 1376 110 687 111 2599 131 1168 1379 2806 2805 684 685 4455 4468 123 5138 0 5137 1517 4700 404 160
a 30 4107 4106 1468 303 1126 541 1106 5008 3088 1469 221 5672 5671 5670 5669 211 4723 1684 1376 110 687 111 2599 131 1168 1379 2806 2805 684 685 4455 4468 123 5138 0 5137 1517 4700 404 160
a 30 4107 4106 1468 303 1126 541 1106 5008 3088 1469 221 5672 5671 5670 5669 211 4723 1684 1376 110 687 111 2599 131 1168 1379 2806 2805 684 685 4455 4468 123 5138 0 5137 1517 4700 404 160
a 30 4107 4106 1468 303 1126 541 1106 5008 3088 1469 221 5672 5671 5670 5669 211 4723 1684 1376 110 687 111 2599 131 1168 1379 2806 2805 684 685 4455 4468 123 5138 0 5137 1517 4700 404 160
a 30 4107 4106 1468 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 8
a 30 4107 4106 1468 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 8
a 30 4107 4106 1468 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 8
a 30 4107 4106 1468 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 8
a 30 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
a 30 4107 4106 1468 303 1126 541 1106 5008 3088 1469 221 5672 5671 5670 5669 211 4723 1684 1376 110 687 266 937 131 1168 1379 2806 2805 684 685 4455 4468 123 5138 0 5137 1517 4700 404 160 8
a 30 4107 4106 1468 303 1126 541 1106 5008 3088 1469 221 5672 5671 5670 5669 211 4723 1684 1376 110 687 266 86 131 1168 1379 2806 2805 684 685 4455 4468 123 5138 0 5137 1517 4700 404 160 83
a 30 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
```

[query: lcc_hprd_n1 / data: lcc_hprd / cs: lcc_hprd_n1]의 출력 결과 중 일부

4. 프로그램 실행 환경

언어: C++11

컴파일러: GCC 9.2.0

IDE: JetBrains CLion 2021.1.2

실행 방법: https://github.com/SNUCSE-CTA/Graph-Pattern-Matching-Challenge의 Compile and Execute 방법과 동일하다.