HW02

수리과학부 2019-16022 박채연

1. Measure the time on various inputs (different values of n, sparse and dense, etc.) and discuss the results.

1) Vertex의 개수가 3000개일 때, 간선의 개수(=density)를 증가시키며 수행시간을 측정하였다.

아래의 명령어를 통해 Vertex가 3000개일 때, 간선의 개수가 1000개, 5000개, 10000개, 50000개, 100000개, 200000개, 300000개인 input 파일을 각각 생성하였다.

```
# ./generator 3000 1000 1.txt
# ./generator 3000 5000 2.txt
# ./generator 3000 10000 3.txt
# ./generator 3000 50000 4.txt
# ./generator 3000 100000 5.txt
# ./generator 3000 200000 6.txt
# ./generator 3000 300000 7.txt
```

위의 7개의 input 파일을 이용하여 각각의 자료구조(adjacency matrix, adjacency list, adjacency array)를 사용했을 때 수행시간을 측정하면 아래와 같다.

- adjacency matrix를 사용했을 때

```
# ./main 1 1.txt m1.txt
Total running time : 0.012777s

# ./main 1 2.txt m2.txt
Total running time : 0.012790s
# ./main 1 3.txt m3.txt
Total running time : 0.014073s
# ./main 1 4.txt m4.txt
Total running time : 0.014506s

# ./main 1 5.txt m5.txt
Total running time : 0.019800s
# ./main 1 6.txt m6.txt
Total running time : 0.019835s
# ./main 1 7.txt m7.txt
Total running time : 0.021524s
```

- adjacency list를 사용했을 때

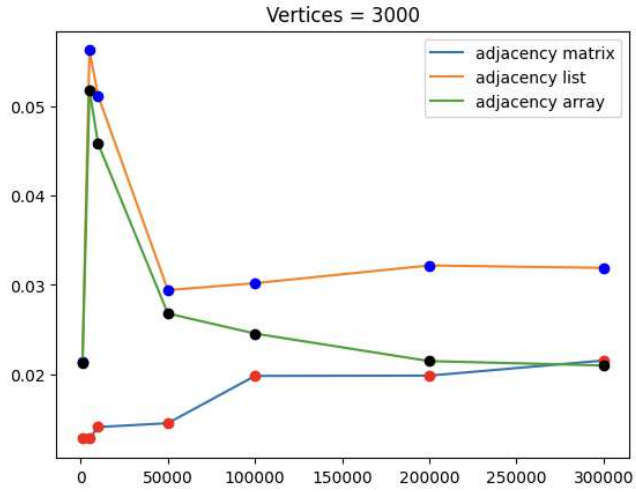
```
# ./main 2 1.txt l1.txt
Total running time : 0.021377s
# ./main 2 2.txt l2.txt
Total running time : 0.056265s
# ./main 2 3.txt l3.txt
Total running time : 0.051167s

# ./main 2 4.txt l4.txt
Total running time : 0.029416s
# ./main 2 5.txt l5.txt
Total running time : 0.030191s
# ./main 2 6.txt l6.txt
Total running time : 0.032175s
# ./main 2 7.txt l7.txt
Total running time : 0.031899s
```

- adjacency array를 사용했을 때

```
# ./main 3 1.txt a1.txt
Total running time : 0.021198s
# ./main 3 2.txt a2.txt
Total running time : 0.051854s
# ./main 3 3.txt a3.txt
Total running time : 0.045794s
# ./main 3 4.txt a4.txt
Total running time : 0.026799s
# ./main 3 5.txt a5.txt
Total running time : 0.024544s
# ./main 3 6.txt a6.txt
Total running time : 0.021463s
# ./main 3 7.txt a7.txt
Total running time : 0.020972s
```

이를 그래프로 표현하면 아래와 같다.



Vertex의 개수가 3000개일 때, 간선의 개수에 따른 수행시간을 비교해본 결과 전반적으로 adjacency list를 이용했을 때의 수행 시간이 가장 느렸고 adjacency matrix를 이용했을 때의 수행 시간이 가장 빨랐다.

한편, 간선의 개수가 증가할수록 수행 시간이 느려질 것이라고 예상했었지만 실제 실험 결과는 이와 달랐다. adjacency list와 adjacency array를 이용했을 때, 간선의 개수가 1000개, 5000개, 10000개로 늘어남에 따라 수행시간이 느려졌다. 그러나 간선의 개수가 이보다 더 커지면 오히려 간선의 개수가 10000개일 때보다 훨씬 수행시간이 빨랐다. 이는 strongly connected components의 개수 때문에 발생한 현상일 것이라고 예상가능하다. l3.txt와 l4.txt를 출력한 결과는 각각 다음과 같다. l3.txt와 l4.txt는 각각 간선의 수가 10000개, 50000개일 때 adjacency list 방법으로 계산한 output결과인데, strongly connected components의 개수가 각각 254개, 1개이다.

```
# cat l3.txt
254
11 81 112 114 118 146 196 206 219 224 228 237 240 257 269 270 281 282 309 316 357 368 375 387 430 431 442 4
50 464 467 484 493 497 523 573 578 584 585 597 623 633 642 669 694 700 701 730 738 746 774 775 777 785 791
794 806 816 816 833 845 849 850 851 855 884 897 900 901 912 916 917 928 947 960 996 997 1011 1017 1021 1033
 1046 1072 1086 1125 1133 1141 1144 1152 1158 1174 1180 1184 1192 1197 1211 1212 1214 1249 1265 1267 1279 1
280 1292 1310 1312 1317 1320 1328 1331 1338 1341 1361 1369 1442 1444 1449 1451 1458 1469 1480 1484 1485 149
1 1504 1505 1508 1556 1565 1583 1594 1614 1629 1635 1639 1646 1651 1667 1674 1676 1692 1698 1702 1723 1724
1728 1729 1750 1752 1761 1767 1777 1784 1798 1809 1825 1830 1832 1836 1855 1863 1870 1879 1902 1923 1928 19
33 1936 1956 1959 1962 1980 1993 2016 2027 2034 2053 2068 2079 2090 2098 2100 2106 2111 2121 2127 2134 2136
 2155 2159 2160 2166 2180 2185 2224 2232 2249 2286 2299 2303 2314 2324 2325 2328 2358 2383 2404 2408 2418 2
455 2483 2511 2516 2530 2533 2574 2600 2604 2616 2628 2641 2651 2659 2660 2665 2669 2672 2685 2691 2694 270
4 2728 2750 2763 2773 2777 2791 2801 2810 2837 2843 2875 2883 2890 2896 2905 2909 2916 2950 2961 2971 2972
2975 2978 2997

# cat l4.txt
1
3000
```

l5.txt, l6.txt, l7.txt 를 모두 출력해보면 l4.txt 결과와 같았다. 간선의 개수가 특정 개수를 초과하면 strongly connected component의 개수가 1개가 될 확률이 크고, 이로 인해 연산 수행 시간이 감소한 것이라고 예상할 수 있다.

한편, adjacency matrix를 이용했을 때의 연산 수행시간은 strongly connected component의 개수에 크게 영향을 받지 않았다.

2) Vertex의 개수가 5000개일 때, 간선의 개수(=density)를 증가시키며 수행시간을 측정하였다.

아래의 명령어를 통해 Vertex가 5000개일 때, 간선의 개수가 1000개, 5000개, 10000개, 50000개, 100000개, 200000개, 300000개인 input 파일을 각각 생성하였다.

```
# ./generator 5000 1000 1.txt
# ./generator 5000 5000 2.txt
# ./generator 5000 10000 3.txt
# ./generator 5000 50000 4.txt
# ./generator 5000 100000 5.txt
# ./generator 5000 200000 6.txt
# ./generator 5000 300000 7.txt
```

위의 7개의 input 파일을 이용하여 각각의 자료구조(adjacency matrix, adjacency list, adjacency array)를 사용했을 때 수행시간을 측정하면 아래와 같다.

- adjacency matrix를 사용했을 때

```
# ./main 1 1.txt m1.txt
Total running time : 0.026215s
# ./main 1 2.txt m2.txt
Total running time : 0.030240s
# ./main 1 3.txt m3.txt
Total running time : 0.031780s
# ./main 1 4.txt m4.txt
Total running time : 0.036508s
# ./main 1 5.txt m5.txt
Total running time : 0.032163s
# ./main 1 6.txt m6.txt
Total running time : 0.030283s

# ./main 1 7.txt m7.txt
Total running time : 0.033611s
```

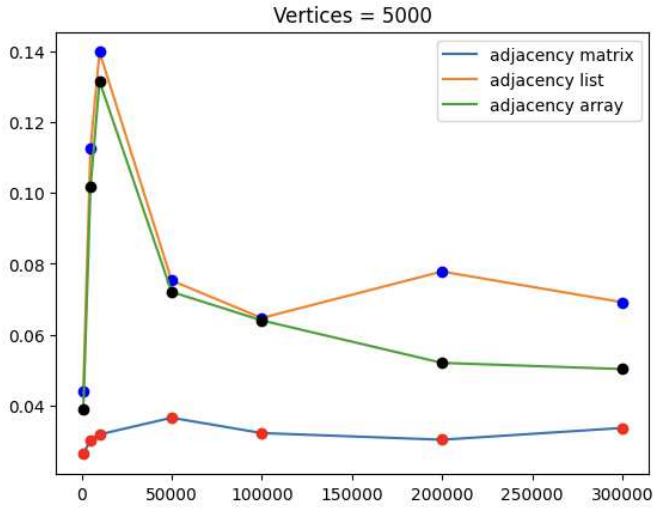
- adjacency list를 사용했을 때

```
# ./main 2 1.txt l1.txt
Total running time : 0.043939s
# ./main 2 2.txt l2.txt
Total running time : 0.112614s
# ./main 2 3.txt l3.txt
Total running time : 0.139843s
# ./main 2 4.txt l4.txt
Total running time : 0.075274s
# ./main 2 5.txt l5.txt
Total running time : 0.064682s
# ./main 2 6.txt l6.txt
Total running time : 0.077817s
# ./main 2 7.txt l7.txt
Total running time : 0.069149s
```

- adjacency array를 사용했을 때

```
# ./main 3 1.txt a1.txt
Total running time : 0.038800s
# ./main 3 2.txt a2.txt
Total running time : 0.101780s
# ./main 3 3.txt a3.txt
Total running time : 0.131540s
# ./main 3 4.txt a4.txt
Total running time : 0.072059s
# ./main 3 5.txt a5.txt
Total running time : 0.064001s
# ./main 3 6.txt a6.txt
Total running time : 0.052004s
# ./main 3 7.txt a7.txt
Total running time : 0.050251s
```

이를 그래프로 표현하면 아래와 같다.



Vertex의 개수가 5000개일 때, 간선의 개수에 따른 수행시간을 비교해본 결과 전반적으로 adjacency list를 이용했을 때의 수행 시간이 가장 느렸고 adjacency matrix를 이용했을 때의 수행 시간이 가장 빨랐다. 또한, Vertex의 개수가 3000개였을 때보다 수행 시간이 느려진 것을 확인 가능하다.

그래프의 개형이 Vertex의 개수가 3000개일 때와 유사함을 알 수 있다. adjacency list와 adjacency array를 이용했을 때, 간선의 개수가 1000개, 5000개, 10000개로 늘어남에 따라 수행시

간이 느려졌다. 그러나 간선의 개수가 이보다 더 커지면 오히려 간선의 개수가 10000개일 때보다 훨씬 수행시간이 빨랐다. 이는 strongly connected components의 개수 때문에 발생한 현상일 것이라고 예상가능할 수 있다. l3.txt와 l4.txt를 출력한 결과는 아래와 같다. 이때, l3.txt 출력결과가 길어 일부만 캡쳐하였다. l3.txt와 l4.txt는 각각 간선의 수가 10000개, 50000개일 때 adjacency list 방법으로 계산한 output결과인데, strongly connected components의 개수가 각각 1902개, 1개이다.

```
# cat l3.txt
1902
3 5 7 8 10 11 12 16 17 21 28 30 31 33 34 40 44 47 51 53 55 58 60 61 62 64 66 68 71 76 79 80 82 85 88 89 90
92 94 100 104 105 107 108 109 110 113 117 120 127 131 134 136 149 151 153 156 157 159 160 161 162 176 177 1
78 180 183 184 187 189 190 196 197 198 199 201 206 211 225 229 232 234 237 238 240 242 246 247 249 251 254

# cat l4.txt
1
5000
```

l5.txt, l6.txt, l7.txt 를 모두 출력해보면 l4.txt 결과와 같았다. 간선의 개수가 특정 개수를 초과하면 strongly connected component의 개수가 1개가 될 확률이 크고, 이로 인해 연산 수행 시간이 줄어든 것이라고 예상할 수 있다.

한편, adjacency matrix를 이용했을 때의 연산 수행시간은 strongly connected component의 개수에 크게 영향을 받지 않았다.

3) 간선의 개수가 1000개일 때, Vertex의 개수를 증가시키며 수행시간을 측정하였다.

아래의 명령어를 통해 간선의 개수가 1000개일 때, Vertex의 개수가 1000개, 2000개, 3000개, 4000개, 5000개인 input 파일을 각각 생성하였다.

```
# ./generator 1000 1000 1.txt
# ./generator 2000 1000 2.txt
# ./generator 3000 1000 3.txt
# ./generator 4000 1000 4.txt
# ./generator 5000 1000 5.txt
```

위의 5개의 input 파일을 이용하여 각각의 자료구조(adjacency matrix, adjacency list, adjacency array)를 사용했을 때 수행시간을 측정하면 아래와 같다.

- adjacency matrix를 사용했을 때

```
# ./main 1 1.txt m1.txt
Total running time : 0.001560s
# ./main 1 2.txt m2.txt
Total running time : 0.004148s
# ./main 1 3.txt m3.txt
Total running time : 0.014264s
# ./main 1 4.txt m4.txt
Total running time : 0.017856s
# ./main 1 5.txt m5.txt
Total running time : 0.029288s
```

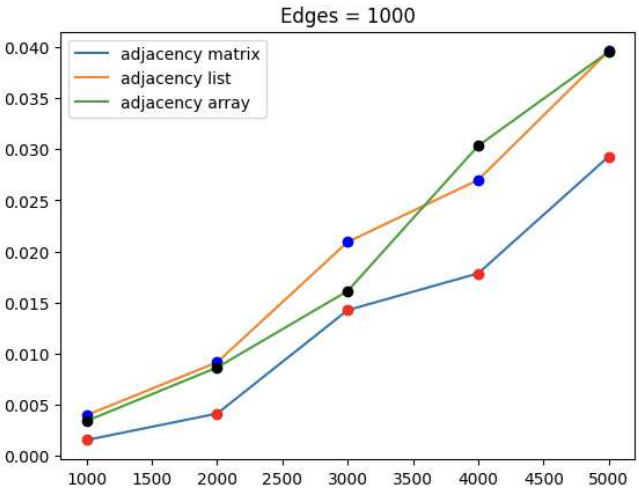
- adjacency list를 사용했을 때

```
# ./main 2 1.txt l1.txt
Total running time : 0.003992s
# ./main 2 2.txt l2.txt
Total running time : 0.009166s
# ./main 2 3.txt l3.txt
Total running time : 0.020930s
# ./main 2 4.txt l4.txt
Total running time : 0.026999s
# ./main 2 5.txt l5.txt
Total running time : 0.039596s
```

- adjacency array를 사용했을 때

```
# ./main 3 1.txt a1.txt
Total running time : 0.003425s
# ./main 3 2.txt a2.txt
Total running time : 0.008657s
# ./main 3 3.txt a3.txt
Total running time : 0.016136s
# ./main 3 4.txt a4.txt
Total running time : 0.030333s
# ./main 3 5.txt a5.txt
Total running time : 0.039532s
```

이를 그래프로 표현하면 아래와 같다.



간선의 개수가 1000개로 고정되어있을 때, Vertex의 개수가 증가할수록 수행시간이 느려지는 것을 확인 가능하다. 특히, adjacency matrix를 사용했을 때 가장 빨랐다.

4) 간선의 개수가 10000개일 때, Vertex의 개수를 증가시키며 수행시간을 측정하였다.

아래의 명령어를 통해 간선의 개수가 10000개일 때, Vertex의 개수가 1000개, 2000개, 3000개, 4000개, 5000개인 input 파일을 각각 생성하였다.

```
# ./generator 1000 10000 1.txt
# ./generator 2000 10000 2.txt
# ./generator 3000 10000 3.txt
# ./generator 4000 10000 4.txt
# ./generator 5000 10000 5.txt
```

위의 5개의 input 파일을 이용하여 각각의 자료구조(adjacency matrix, adjacency list, adjacency array)를 사용했을 때 수행시간을 측정하면 아래와 같다.

- adjacency matrix를 사용했을 때

```
# ./main 1 1.txt m1.txt
Total running time : 0.001755s
# ./main 1 2.txt m2.txt
Total running time : 0.006613s
# ./main 1 3.txt m3.txt
Total running time : 0.012317s
# ./main 1 4.txt m4.txt
Total running time : 0.018283s
# ./main 1 5.txt m5.txt
Total running time : 0.027605s
```

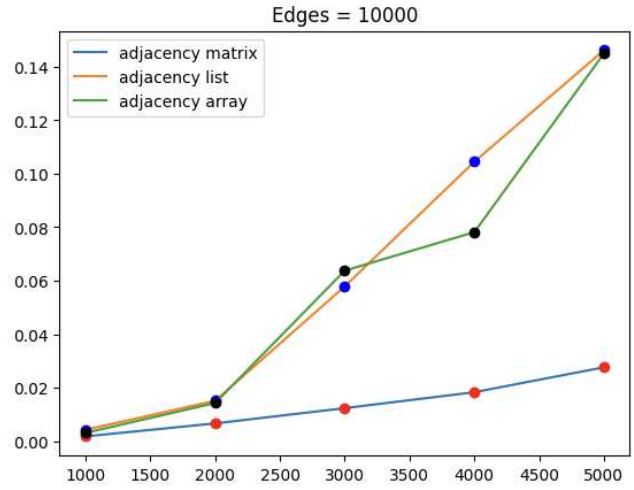
- adjacency list를 사용했을 때

```
# ./main 2 1.txt l1.txt
Total running time : 0.004234s
# ./main 2 2.txt l2.txt
Total running time : 0.015074s
# ./main 2 3.txt l3.txt
Total running time : 0.057080s
# ./main 2 4.txt l4.txt
Total running time : 0.104517s
# ./main 2 5.txt l5.txt
Total running time : 0.146216s
```

- adjacency array를 사용했을 때

```
# ./main 3 1.txt a1.txt
Total running time : 0.003061s
# ./main 3 2.txt a2.txt
Total running time : 0.014255s
# ./main 3 3.txt a3.txt
Total running time : 0.063767s
# ./main 3 4.txt a4.txt
Total running time : 0.078159s
# ./main 3 5.txt a5.txt
Total running time : 0.145011s
```

이를 그래프로 표현하면 아래와 같다.



그래프의 개형이 간선의 개수가 1000개일 때와 유사하다. 간선의 개수가 10000개로 고정되어 있을 때, Vertex의 개수가 증가할수록 수행시간이 느려지는 것을 확인 가능하다. 또한, 간선의 개수가 1000개였을 때보다 수행 시간이 느려진 것을 확인 가능하다.

2. Explain how your program works

'main.c' 파일에 main 프로그램을 구현하고 'gcc -std=gnu99 -O2 main.c -o main' 명령어를 통해 컴파일 하였다. 그 결과 main이라는 실행파일이 생성되는데, 이는 실행 모드, input 파일 경로, output 파일 경로를 순서대로 인자로 입력받는다. 실행모드에는 총 3가지가 있는데, 1은 adjacency matrix를, 2는 adjacency list를, 3은 adjacency array를 이용하여 strongly connected components를 구한다. 실행모드 별로 사용하는 자료구조만 다르고, 전체적인 로직은 비슷하다.

```
switch(mode){
    // Mode 1: an adjacency matrix
    case 1:
        makeMatrix(fpin, n);
        calMatrix(fpout, n);
        break;
    // Mode 2: an adjacency list
    case 2:
        makeList(fpin, n);
        calList(fpout, n);
        break;
    // Mode 3: an adjacency array
    case 3:
        makeArray(fpin, n);
        calArray(fpout, n);
        break;
}
```

위 이미지는 main문 내의 핵심 로직이다. 실행모드가 1일 경우 makeMatrix()와 calMatrix() 함수를, 2일 경우 makeList()와 calMatrix() 함수를, 3일 경우 makeArray()와 calArray()를 순차적으로 호출한다. 'make~' 함수는 input 파일을 읽어들여 ~에 해당하는 자료구조를 initialize하는 함수이고, 'cal~' 함수는 ~에 해당하는 자료구조를 이용하여 strongly connected components를 구하는 함수이다. 해당 함수에서 수행하는 계산은 아래와 같다.

① graph G에서 DFS를 이용하여 각각의 vertex에 대해 finish time 구하기

② finish time이 느린 vertex 순으로 graph G_R에서 DFS를 수행하기

③ ②에서 구한 tree를 strongly connected component로 기록하기

마지막으로 'cal~' 함수 내에서 find_answer() 함수를 공통적으로 호출하였는데, 이는 앞에서 구한 strongly connected component 내 element 들에 대해 xor 연산을 수행하고 배열에 저장 후 이를 정렬하여 output 파일에 기록하는 역할을 수행한다. 함수별 보다 자세한 설명은 소스코드 내 주석을 통해 확인 가능하다.

마지막으로, 제출 목록에 포함되지 않아 해당 파일을 제출하진 않았지만 'generator.c' 파일은 input 파일을 생성하기 위해 작성된 소스코드이다. 'gcc -std=gnu99 -O2 generator.c -o generator' 명령어를 통해 컴파일 하면 'generator'이라는 이름의 실행 파일이 생성되는데 이는 노드 수, 간선 수, 생성하고자 하는 input 파일의 경로를 순서대로 인자로 입력받는다. 예를 들어, './generator N M ./1.in'이라는 명령어는 현재 디렉토리에서 '1.in' 이라는 파일을 생성하여 첫번째 줄에 노드 수인 N을 출력하고 두번째 줄에 간선 수인 M를 출력한다. 다음 M개 줄에 간선을 출력한다.

```
for (int i = 0; i < M; i++){
    int temp_1 = 0, temp_2 = 0;
    while (temp_1 == temp_2 || visited[temp_1][temp_2]){
        temp_1 = (rand() % N) + 1;
        temp_2 = (rand() % N) + 1;
    }
    fprintf(fp, "%d %d\n", temp_1, temp_2);
    visited[temp_1][temp_2] = 1;
}
```

이때, 위와 같이 while 문을 작성하여 self-loop 및 중복되는 간선이 없도록 하였다.

3. Write down the environment you run your program and how to run it

Window에서 VS code를 이용해서 프로그램을 작성하였고, 도커 환경 내에서 n 값과 자료 구조(an adjacency matrix, an adjacency list, an adjacency array)에 따른 수행 속도를 관찰하기 위해 테스트를 수행하였다. Docker Desktop에서 'docker pull yehyun/cta:assignment' 명령어를 통해 도커 이미지를 불러오고 'docker run -it yehyun/cta:assignment bash' 명령어를 통해 실행하였다.

과제 공지사항에서 명시된 바와 같이 'gcc -std=gnu99 -O2 main.c -o main' 명령어로 main.c 파일을 컴파일하고, './main x ./1.in ./1.out'명령어로 프로그램을 실행하였다. 이때 'x' 자리에는 1, 2, 3 중 하나가 올 수 있는데, 1은 adjacency matrix를, 2는 adjacency list를, 3은 adjacency array를 이용하는 방법이다.