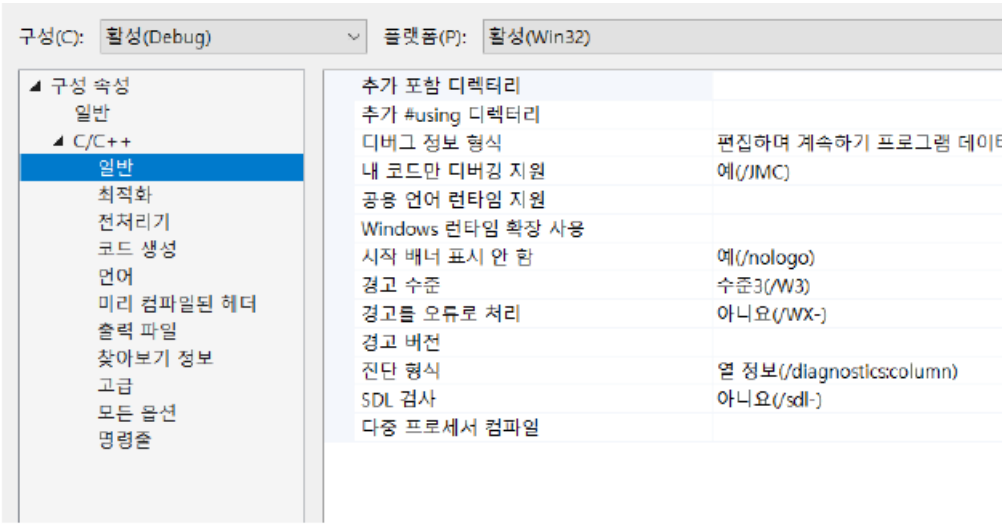
알고리즘 HW3 – DFS / Topological Sort

2016025469 서건식

컴파일환경:Windows10 x64, Visual Studio 2019

- Fprint,fscanf,fopen 등의 파일 입출력 함수를 사용하였기 때문에




SDL 검사를 '아니요'로 설정 후 컴파일 하였습니다.

3-1:

- 인풋은 input3-1.txt입니다.

  Description of Code:

  1. 전역변수

```
int color[100]; //0 : white , 1 : gray , 2 : black
vector<int> adj[100];
bool check[100];
int cross = 0;
int result[100];
int cnt = 0;
```

  - 수도코드와 거의 동일하나, check 배열은 cross edge를 체크하기 위해 생성하였습니다.
  - Cnt는 result를 count하기 위한 변수입니다.

2. DFS

```
16  void DFS(int n)
17  {
18      check[n] = true;
19      color[n] = 1;
20
21      result[cnt++] = n;
22
23      for (int i = 0; i < adj[n].size(); i++)
24      {
25          int a = adj[n][i];
26          if (color[a] == 0) { // 흰색
27              cout << "(" << n << "," << a << ") " << "tree edge" << endl;
28              DFS(a);
29          }
30          else if (color[a] == 1) // 회색
31          {
32              cout << "(" << n << "," << a << ") " << "back edge" << endl;
33          }
34          else if (color[a] == 2 &&  cross == 0) // 검은색
35          {
36              cout << "(" << n << "," << a << ") " << "forward edge" << endl;
37          }
38          else if (color[a] == 2 && cross != 0) // 반대쪽에서 넘어왔을 때
39          {
40              cout << "(" << n << "," << a << ") " << "cross edge" << endl;
41
42          }
43      }
44
45      color[n] = 2;
46
47
48  }
49
```

- 수도코드를 참고하여 작성하였습니다. 인접배열전체를 돌면서 각각의 노드에 대해 DFS를 시행합니다.
- 특히 color가 black이고, cross를 체크했을 때 1이라면 반대쪽에서 넘어온 것으로, 따로 조건문을 처리하였습니다.

3. Main

```
51  int main()
52  {
53      int n=0; // 노드의 갯수
54      int e;
55      char a;
56
57      for (int i = 0; i < 100; i++)
58      {
59          check[i] = false;
60          color[i] = 0;
61      }
62      FILE* file = fopen("input3-1.txt", "r");
63      fscanf(file, "%d₩n", &n);
64
65
66      for (int i = 1; i <= n; i++)
67      {
68          for (int j = 1; j <= n; j++)
69          {
70              fscanf(file, "%d", &e);
71              if (e == 1)
72              {
73                  adj[i].push_back(j);
74              }
75              if (feof(file))
76              {
77                  break;
78              }
79
80          }
81
82      }
```

■ 각 전역변수들을 초기화 해주고, input값을 입력 받아 인접행렬에 쌓아줍니다.

■ Adj는 vector로 구현하였습니다.

```
83      DFS(1);
84
85      for (int i = 1; i <= n; i++)
86      {
87          if (check[i] == false)
88          {
89              cross++;
90              DFS(i);
91          }
92      }
93      for (int i = 0; i < n; i++)
94      {
95          cout << result[i] << ' ';
96      }
97
98      fclose(file);
99  }
```

■ DFS를 시행합니다.

■ Check가 안된 노드는 cross를 증가시켜 시작점이 바뀌었음을 체크합니다. 그리고 그 노드부터 다시 DFS를 시행합니다.

3-2:

- 인풋은 input3-2.txt입니다.

  Description of Code:

1. 전역변수

```
int n = 0; // 노드의 갯수
bool visit[100];
vector<int> adj[100];
int result[100];
int component[100];
int cnt = 0;
```

   - 전역변수 입니다.
   - Component 배열에 대한 설명은 DFS_ALL()함수에서 설명하겠습니다.

2. DFS & DFS_ALL

```
15  void DFS(int k)
16  {
17      visit[k] = true;
18      result[cnt++] = k;
19      for (int i = 0; i < adj[k].size(); i++)
20      {
21          int a = adj[k][i];
22          if (visit[a] == false) {
23              DFS(a);
24          }
25      }
26
27  }
28
29
30  int DFS_ALL()
31  {
32      int cmp = 0;
33      int component_count =0;
34      for (int i = 0; i < 100; i++)
35      {
36          visit[i] = false;
37      }
38
39      for (int i = 1; i <= n; i++)
40      {
41          if (visit[i] == false)
42          {
43              DFS(i);
44              cmp++;
45
46              for (component_count; component_count < cnt; component_count++)
47              {
48                  component[result[component_count]] = cmp; // 각 노드의 component 번호 저장하는 코드
49                  //DFS 결과 Array에 있는 노드의 index를 나타내는 component array에 각 노드의 component값을 저장
50              }
51
52          }
53      }
54
55      return cmp;
56  }
57
```

   - DFS는 3-1과 구조가 같습니다. 다만, color에 대한 명시만 빠졌습니다.
   - DFS_ALL은 component의 개수만큼 DFS를 돌리기 위한 것입니다. (그래프 전체에 적용시키기 위한 함수)

노드의 개수만큼 반복문을 돌면서, 방문하지 않은 노드에 대해서 DFS를 시행합니다. 시행된 경우의 수에서 탐색한 노드에는 cmp가 부여되고, 같은 component가 부여되는 것입니다.

3. Main

```
60  int main()
61  {
62      int e;
63      char a;
64
65      for (int i = 0; i < 100; i++)
66      {
67          visit[i] = false;
68      }
69      FILE* file = fopen("input3-2.txt", "r");
70      fscanf(file, "%d₩n", &n);
71
72
73      for (int i = 1; i <= n; i++)
74      {
75          for (int j = 1; j <= n; j++)
76          {
77              fscanf(file, "%d", &e);
78              if (e == 1)
79              {
80                  adj[i].push_back(j);
81                  adj[j].push_back(i);
82              }
83              if (feof(file))
84              {
85                  break;
86              }
87
88          }
89
90      }
```

■ 파일에서 input을 받을 때, undirected기 때문에 adj[j].push_back(i); 를 추가하였습니다.

```
91      int cmp = DFS_ALL();
92
93      for (int i = 0; i < n; i++)
94      {
95          cout << result[i] << ' ';
96      }
97
98      cout << endl;
99
100     for (int i = 1; i <= n; i++)
101     {
102         cout << i << ':' << component[i] << endl;
103     }
104
105
106     fclose(file);
107 }
```

■ 모든 노드에 대해서 DFS를 시행한 다음 결과와 component값을 출력합니다.

3-3:

- 인풋은 input3-3.txt입니다.

  Description of Code:

  1. DFS

```
int DFS(int n)
{
    check[n] = true;
    color[n] = 1;
    int dag = 1;


    for (int i = 0; i < adj[n].size(); i++)
    {
        int a = adj[n][i];
        if (color[a] == 0) { // 흰색
            DFS(a);
        }
        else if (color[a] == 1) // 회색
        {
            dag = 0; //back edge가 발생하면 dag가 아니므로 0으로 바꿔줌.
        }

    }

    color[n] = 2;
    result[cnt++] = n;
    return dag;
}
```

  - Dag의 판별을 위해서, 조건문을 하나 더 추가합니다.
  - Back edge의 발생유무를 따져 dag의 값을 변경합니다. 맞으면 1, 아니면 0을 반환합니다.

  2. Main

     A. Main의 나머지 과제와 동일한 부분은 생략하였습니다.

```
    if (DFS(1))
    {
        cout << 1 << endl;
    }

    for (int i = 1; i <= n; i++)
    {
        if (check[i] == false)
        {
            cross++;
            DFS(i);
        }
    }
    for (int i = n-1 ; i >= 0; i--)
    {
        cout << result[i] << ' ';
    }

    fclose(file);
}
```

     - If문에서 dag를 체크한 후, 맞으면 1을 출력해줍니다.

       그 이후 결과를 출력하는데, topological sort의 결과는 DFS 결과의 반대이므로, 거꾸로 출력합니다.