본 PSet 은 저의 강의 경험과 학생들의 의견 및 Stanford CS106 과 Harvard CS50 같은 강의에서 수집된 자료를 토대로 작성되었습니다. 본 PSet 에 문제가 있거나, 질문 혹은 의견이 있다면, 언제든지 알려 주시면 감사하겠습니다. 강의 개선에 많은 도움이 되겠습니다.
idebtor@gmail.com

PSet - Graph DFS/BFS

목차

시작하기: 프로젝트 빌드

준비 운동으로 그래프라는 프로젝트를 빌드하고 그래프 메뉴와 그래프를 표시합니다. 아래와 같은 파일들이 제공됩니다. lib/nowic.lib 과 include/nowic.h 를 사용하여 프로젝트를 빌드합니다. 이번 프로젝트에서도 g++을 사용할 수 있습니다.

- graph.h - 수정 금지
- graph.cpp - 작업할 파일
- driver.cpp - 수정 금지
- graph?.txt - 그래프 파일, VS 프로젝트 폴더 또는 실행 파일(.exe)과 함께 보관합니다.
- graphx.exe - 결과 비교용 실행 파일

프로그램을 실행하면 아래와 같이 그래프 메뉴를 표시합니다:

```
Graph
n - new graph file      x - connected(v,w)
d - DFS(v=0)            e - distance(v,w)
b - BFS(v=0)            p - path(v,w)
m - print mode[adjList/graph]
Command(q to quit):
```

이제 메뉴 옵션 n 에서 그래프 파일 이름을 입력하여 그래프를 만들 수 있습니다. 유효한 그래프 파일을 지정하면 그래프의 인접 리스트를 읽고 출력합니다.

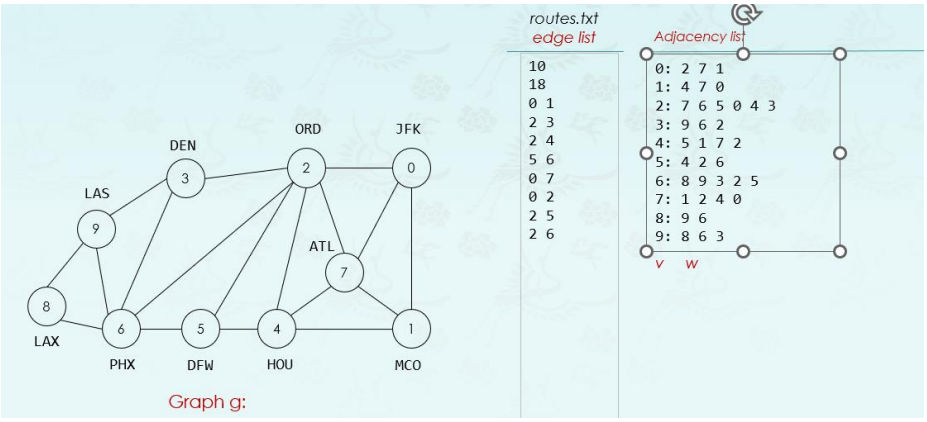
명령행에서도 그래프 파일을 지정할 수 있으며, 지정된 파일을 읽고 출력합니다.

- 그래프 파일에서 읽어 온 '점선 그래프'
- 인접 리스트
- parentDFS[], distTo[], CCID[] 등의 스크래치 버퍼 내용을 포함한 그래프 현황

```
.  [0] -------[1]--
.   |        / |   |
.   |      /   |  [2]
.   |    /     |   /
.   |  /       |  /
.  [4]--------[3]

    vertex[0..4] =    0   1   2   3   4
     color[0..4] =    0   0   0   0   0

      DFS0[0..4] =    0   4   3   2   1
      CCID[0..4] =    1   1   1   1   1
DFS parent[0..4] =   -1   2   3   4   0
      BFS0[0..4] =    0   4   1   3   2
    DistTo[0..4] =    0   1   2   2   1
BFS parent[0..4] =   -1   0   1   4   0

Graph [Graph][Tablet]   file:graph1.txt V:5 E:14 CCs:1 Deg:4
n - new graph file      x - connected(v,w)
d - DFS(v=0)            e - distance(v,w)
b - BFS(v=0)            p - path(v,w)
c - cyclic(v=0)?        m - print mode[adjList/graph]
t - bigraph(v=0)?       a - bigraph using adj-list coloring
Command(q to quit):
```

메뉴의 몇 가지 옵션을 구현하세요.

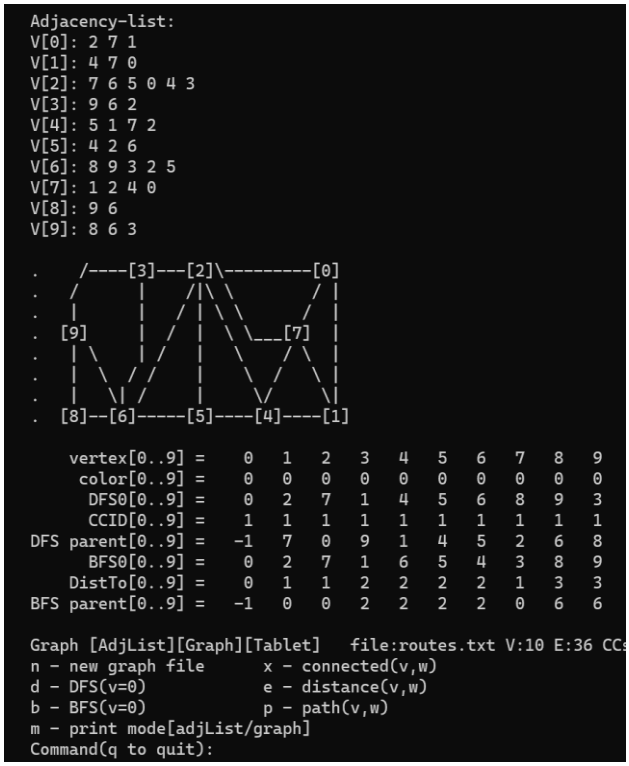
Step 1- 준비 운동



위 그래프를 참고하여 다음 과제를 수행하고, 결과물로 routes.txt 를 제출하세요.

1. PSet 에 제공된 graphx.exe 와 graph*.txt 파일들을 실행해보면서, 메뉴와 사용법에 익숙해지십시오. DFS/BFS 를 실행하고 결과물로 검토해보세요. **DFS 와 BFS 의 시퀀스**를 얻은 방법을 vertex 0 에서 출발할 뿐만 아니라 다른 vertex 에서 출발하는 연습도 해보십시오.

2. 아래 보이는 것과 같이 인접 리스트를 만들어 내는 **routes.txt** 파일을 생성하고, 또한 다음과 같은 그래프도 그릴 수 있도록 작성하십시오. .



Step 2 - 인접 리스트 표시하기 (메뉴 m)

그래프 데이터 구조를 이해하기 위해, 그래프 텍스트 파일에서 읽어온 그래프 데이터 구조의 내용을 출력해 봅시다. 그래프 텍스트 파일은 아래와 같이 세 가지 라인으로 구성되어 있습니다.

1. #과 /로 시작하는 라인은 주석입니다.
2. .(점)으로 시작하는 라인은 메뉴 목록과 함께 표시합니다.
3. 숫자로 시작하는 라인은 node 와 edge 입니다.

```
# Graph file format example:
# To represent a graph:
# The number of vertex in the graph comes at the first line.
# The number of edges comes In the following line,
# Then list a pair of vertices connected each other in each line.
# The order of a pair of vertices should not be a matter.
# Blank lines and the lines which begins with # or ; are ignored.
#
# The lines that begins with . will be read into graph data structure
# and displayed on request.
#
# For example:
.  [0] -------[1]--
.   |        / |   |
.   |      /   |  [2]
.   |    /     |   /
.   |  /       |  /
.  [4]--------[3]
```

```
#
#            vertex[0..4] =    0   1   2   3   4
#             color[0..4] =    0   0   0   0   0
#              DFS0[0..4] =    0   4   3   2   1
#              CCID[0..4] =    1   1   1   1   1
#        DFS parent[0..4] =   -1   2   3   4   0
#              BFS0[0..4] =    0   4   1   3   2
#            DistTo[0..4] =    0   1   2   2   1
#        BFS parent[0..4] =   -1   0   1   4   0
5
7
0 1
0 4
1 2
1 4
1 3
2 3
3 4
```

```
// prints the adjacency list of graph
void print_adjlist(graph g){
  if (empty(g)) return;

  cout << "your code here \n";

}
```

Step 3 - DFS(), DFS_CCs(), DFSpath()

그래프의 깊이 우선 탐색을 수행하는 DFS() 함수를 구현합니다.

이 step 은 두 단계로 구성되어 있습니다. 첫째, 필요한 데이터 구조를 초기화합니다. 둘째, DFS_CCs()를 재귀적으로 계산하는 함수 DFS()를 호출합니다. 이 DFS()는 DFSpath()와 같은 일부 함수에서 사용됩니다.

```
// runs DFS for at vertex v recursively.
// Only que, g->marked[v] and g->parentDFS[] are updated here.
void DFS(graph g, int v, queue<int>& que) {
  g->marked[v] = true; // visited
  que.push(v);         // save the path

  cout << "your code here (recursion) \n";
}
```

DFS 메뉴 옵션 d 는 필요한 데이터 구조를 초기화하는 DFS_CCs()를 먼저 호출하고 DFS()를 각 구성 요소마다 재귀적으로 호출합니다.

다음 DFS_CCs() 함수는 vertex 0 와 연결된 하나의 컴포넌트만 작동합니다. 그래프에 있는 다른 컴포넌트들에도 작동하도록 코드를 수정하세요. 초기화 부분은 수정할 필요할 필요가 없습니다.

```
// runs DFS for all components and produces DFS0[], CCID[] & parentDFS[].
void DFS_CCs(graph g) {
  if (empty(g)) return;
  for (int i = 0; i < V(g); i++) {
```

```
    g->marked[i] = false;
    g->parentDFS[i] = -1;
    g->CCID[i] = 0;
  }

  queue<int> que;
  cout << "your code here: make it work with multiple CC's\n";
  DFS(g, 0, que);
  setDFS0(g, 0, que);

  g->DFSv = {};
}
```

DFS()와 DFS_CCs()를 성공적으로 구현하면 아래와 같이 DFSpath()를 완성하세요.

```
// returns a path from v to w using the DFS result or parentDFS[].
// It has to use a stack to retrace the path back to the source.
// Once the client(caller) gets a stack returned,
void DFSpath(graph g, int v, int w, stack<int>& path) {
  if (empty(g)) return;
  for (int i = 0; i < V(g); i++) {
    g->marked[i] = false;
    g->parentDFS[i] = -1;
  }
  queue<int> q;
  DFS(g, v, q);    // DFS at v, starting vertex
  g->DFSv = q;     // DFS result at v
  path = {};

  cout << "your code here\n"; // push v to w path to the stack path
}
```

Step 4 - BFS(), BFS_CCs, distTo[] and parentBFS[]

다음 BFS_CCs() 함수는 vertex 0 와 연결된 컴포넌트에 작동합니다. 다른 컴포넌트들 (connected component)에 대해서도 작동하도록 코드를 수정하십시오. 초기화 부분은 수정할 필요할 필요가 없습니다.

```
// runs BFS for all vertices or all connected components
// It begins with the first vertex 0 at the adjacent list.
// It produces BFS0[], distTo[] & parentBFS[].
void BFS_CCs(graph g) {
  DPRINT(cout << ">BFS_CCs\n";);
  if (empty(g)) return;

  for (int i = 0; i < V(g); i++) {
     g->marked[i] = false;
     g->parentBFS[i] = -1;
     g->BFS0[i] = -1;
     g->distTo[i] = -1;
  }

  // BFS for all connected components starting from 0
  cout << "your code here to replace the next line\n";
  BFS(g, 0);

  g->BFSv = {};  // clear it not to display, queue<int>().swap(g->BFSv);
```

```
  DPRINT(cout << "\n<BFS_CCs\n";);
}
```

- while 문 안에 있는 distTo[]와 parentBFS[]의 값을 설정하도록 다음 두 라인을 완성하세요.
  이 distTo[]는 v 에서 모든 w 꼭짓점까지의 거리(값)를 가지고 있습니다.

```
// runs BFS starting at v and produces distTo[] & parentBFS[]
void BFS(graph g, int v) {
  queue<int> que;              // to process each vertex
  queue<int> sav;       // BFS result saved

  // all marked[] are set to false since it may visit all vertices
  for (int i = 0; i < V(g); i++)  g->marked[i] = false;
  g->parentBFS[v] = -1;
  g->marked[v] = true;
  g->distTo[v] = 0;
  g->BFSv = {};

  que.push(v);
  sav.push(v);

while (!que.empty()) {
    int cur = que.front(); que.pop();  // remove it since processed
    for (gnode w = g->adj[cur].next; w; w = w->next) {
      if (!g->marked[w->item]) {
        g->marked[w->item] = true;
        que.push(w->item);// queued to process next
        sav.push(w->item);// save the result

        cout << "your code here";   // set parentBFS[] & distTo[]

      }
    }
  }
  g->BFSv = sav;                // save the result at v
  setBFS0(g, v, sav);
}
```

- distTo(g, v, w) 함수는 BFS(g, v)를 호출하여 distTo[]와 parentBFS[]의 값을 설정합니다. 그런 다음 distTo[]에 저장된 두 꼭짓점 v 와 w 의 거리를 반환합니다.
- 알다시피 distTo[]는 v 와 그래프의 모든 꼭짓점 w 의 거리를 가지고 있습니다. distTo(g, v, w) 함수는 보통 드라이버에 의해 호출되며 v 와 w 의 거리를 반환합니다.

```
// returns the number of edges in a shortest path between v and w
int distTo(graph g, int v, int w) {
  if (empty(g)) return 0;
  if (!connected(g, v, w)) return 0;

  BFS(g, v);
  cout << "your code here\n";      // compute and return distance
  return 0;
}
```

BFS()를 성공적으로 구현한 후, 아래와 같이 BFSpath()를 완성하세요. 여기에 추가하는 코드는 DFSpath()에 추가한 코드와 같으므로 복사 및 붙여넣기만 하면 됩니다.

```
// returns a path from v to w using the BFS result or parentBFS[].
// It has to use a stack to retrace the path back to the source.
// Once the client(caller) gets a stack returned,
void BFSpath(graph g, int v, int w, stack<int>& path) {
  if (empty(g)) return;

  BFS(g, v);                    // g->BFSv updated already.

  path = {};                    // clear path, stack<int>().swap(path);

  cout << "your code here\n";   // push v to w path to the stack path

}
```

과제 제출

- 소스 파일 상단에 아래와 같이 아너 코드 문장을 적고 서명하세요.
  On my honor, I pledge that I have neither received nor provided improper assistance in the completion of this assignment.
  서명: _____ 분반: _____ 학번: _____
- 제출하기 전에 코드가 제대로 컴파일이 되고 실행되는지 확인하세요. 제출 직전에 급하게 코드를 수정한 후 코드가 제대로 컴파일이 될 거라고 짐작하지 않는 게 좋습니다. “거의” 작동하는 코드도 틀린 것입니다.
- 과제가 컴파일 및 실행된다면, 마감 기한 전까지 과제의 일부만 완성했더라도 제출하기 바랍니다. 컴파일 및 실행되지 않는다면 제출하지 마세요. 마감 시간 이후 24 시간 이내 제출하면, 만점에서 25% 감점하고 채점합니다. 그 이상 늦은 것은 채점하지 않으며, 0 점 처리합니다.
- 제출 후, **마감 기한 전까지** 수정 및 재제출이 가능합니다. 파일 하나만 수정하더라도 해당 파일과 관련된 파일들을 모두 재제출해야 합니다. 재제출 횟수는 제한 없습니다. 마감 기한 전에 **가장 마지막으로** 제출된 파일을 채점할 것입니다.

제출 파일 목록

- Step 1: routes.txt
- Step 2 ~ 4: graph.cpp

마감 기한 & 배점

- Due:
- Grade: