CSE3013 (컴퓨터공학 설계 및 실험 I)
WIN-3 결과 보고서

서강대학교 컴퓨터공학과 박수현 (20181634)

서강대학교 컴퓨터공학과

1 목적

실험과 과제에서 제시한 문제를 이해하고, 이를 해결하기 위해 사용한 알고리즘 및 자료 구조를 기술한다.

2 문제

2.1 실험

`struct Point`

- `float x, y`: 2차원 직교좌표계 상의 (x,y) 좌표이다.

`struct Line`

- `int xl, yl`: 선분의 시점의 2차원 직교좌표계 상의 (xl,yl) 좌표이다.
- `int xr, yr`: 선분의 종점의 2차원 직교좌표계 상의 (xr,yr) 좌표이다.

`struct node`: `class mylist`를 구성하는 노드이다.

- `Point *point`: 노드의 값이다.
- `node *next`: 다음 노드에 대한 참조이다.

`class mylist`: 링크드 리스트를 구현한 자료구조이다. 자료가 n개일 때, 이 자료구조의 공간 복잡도는 $\mathcal{O}(n)$이다.

- `int num` (private): 링크드 리스트가 포함하는 노드의 개수이다.

- `node *head` (private): 링크드 리스트의 첫 노드이다.
- `node *prev_node` (private): 현재 탐색하는 노드의 앞에 위치한 노드이다.
- `node *curr_node` (private): 현재 탐색하는 노드이다.
- `node* del()` (protected): 노드를 삭제하는 메서드이다.
- `mylist()` (public): 링크드 리스트 생성자이다. 빈 링크드 리스트를 생성한다. 시간 복잡도는 $\mathcal{O}(1)$이다.
- `virtual ~mylist()` (public): 링크드 리스트 소멸자이다. 모든 원소를 순회하며 삭제한다. 시간 복잡도는 $\mathcal{O}(n)$이다.
- `bool isEmpty()` (public): 링크드 리스트가 비어 있는지 아닌지 확인한다. 결과는 `(num == 0)`과 같으며, 시간 복잡도는 $\mathcal{O}(1)$이다.
- `void add(Point tpoint)` (public): 링크드 리스트의 맨 앞에 새 노드를 추가한다. 시간 복잡도는 $\mathcal{O}(1)$이다.
- `node* move_first(void)` (public): `curr_node`를 맨 앞의 노드로 이동한다. 시간 복잡도는 $\mathcal{O}(1)$이다.
- `node* move_next(void)` (public): `curr_node`를 다음 노드로 이동한다. 시간 복잡도는 $\mathcal{O}(1)$이다.

알고리즘

`void init_data()` : 값들을 초기화한다. 시간 복잡도는 $\mathcal{O}(1)$이다.

1 메모리에서 $mLine$, $mPoint$, m_flow_point 해제
2 $mLine\_num \leftarrow 0$, $mPoint\_num \leftarrow 0$, $init\_state \leftarrow$ false, $draw\_state \leftarrow$ false, $sele\_state \leftarrow$ false, $curr\_point \leftarrow 0$

`void data_read(LPCTSTR fname)` : 파일 이름이 `fname`인 파일을 읽는다.

1 $file \leftarrow$ 파일 `fname`을 읽기 모드로 열기
2 $mLine_num \leftarrow file$에서 읽기
3 $mLine$의 메모리를 $mLine_num \times \text{sizeof}(\text{Line})$만큼 할당
4 $0 \leq i < mLine_num$인 모든 정수 i에 대해 반복:
  a $(mLine_i)_{xl}, (mLine_i)_{yl}, (mLine_i)_{xr}, (mLine_i)_{yr} \leftarrow file$에서 읽기
  b $(mLine_i)_{xl} > (mLine_i)_{xr}$인 경우:
    i $(mLine_i)_{xl}$과 $(mLine_i)_{xr}$ 교환
    ii $(mLine_i)_{yl}$과 $(mLine_i)_{yr}$ 교환
5 $mPoint_num \leftarrow file$에서 읽기
6 $mPoint$의 메모리를 $mPoint_num \times \text{sizeof}(\text{Point})$만큼 할당
7 $0 \leq i < mPoint_num$인 모든 정수 i에 대해 반복:
  a $mPoint_x, mPoint_y \leftarrow file$에서 읽기
8 $file$ 닫기
9 $init\_state \leftarrow$ true

`void waterfall_Solver()` : 선택된 시작점에 대해 주어진 문제를 해결한다. m개의 물구멍에 대해 최악의 경우 n개의 선분에 대해 다른 모든 선분들과의 교점을 찾는 작업을 진행하므로, 시간 복잡도는 $\mathcal{O}\left(mn^2\right)$이다.

1 m_flow_point의 메모리를 sizeof(mylist)만큼 할당
2 $init_state$가 거짓일 경우: 메서드 종료
3 $P \leftarrow mPoint_{curr\_point}$
4 m_flow_point에 P 추가
5 $P_y > 0$일 경우 반복:
- **a** 선분 l, 점 $M = (0,0)$ 선언
- **b** $0 \leq i < mLine_num$인 정수 i에 대해 반복:
  - **i** $k \leftarrow mLine_i$
  - **ii** $kxmin \leftarrow \min(k_{xl}, k_{xr})$, $kxmax \leftarrow \max(k_{xl}, k_{xr})$
  - **iii** $kxmin < P_x < kxmax$가 아닐 경우: 다음 반복 단계로 진행
  - **iv** $ratio \leftarrow \dfrac{P_x - k_{xl}}{k_{xr} - k_{xl}}$
  - **v** $Q_y \leftarrow ratio\left(k_{yr} - k_{yl}\right) + k_{yl}$
  - **vi** $Q_y < M_y$일 경우: 다음 반복 단계로 진행
  - **vii** $Q_y > P_y$일 경우: 다음 반복 단계로 진행
  - **viii** $Q \leftarrow (P_x, Q_y)$
  - **ix** $M \leftarrow Q$, $l \leftarrow k$
- **c** $M = (0,0)$인 경우:
  - **i** $base \leftarrow (P_x, 0)$
  - **ii** m_flow_point에 $base$ 추가
  - **iii** 반복 **5** 종료
- **d** $M \neq (0,0)$인 경우:
  - **i** m_flow_point에 M 추가
- **e** $l_{yl} < l_{yr}$인 경우:
  - **i** $p \leftarrow \left(l_{xl}, l_{yl}\right)$
  - **ii** $P \leftarrow p$
- **f** $l_{yl} \geq l_{yr}$인 경우:
  - **i** $p \leftarrow \left(l_{xr}, l_{yr}\right)$
  - **ii** $P \leftarrow p$
- **g** m_flow_point에 P 추가

`void drawBackground(CDC* pDC)` : DC에 배경 오브젝트를 그린다. **2** – **7**은 최상단과 최하단의 경계선을, **8** – **11**은 물구멍을, **12** – **15**는 선분들을 그린다. m개의 물구멍과 n개의 선분을 $\mathcal{O}(1)$의 작업으로 모두 그리므로, 시간 복잡도는 $\mathcal{O}(m+n)$이다.

1 정수 i, CPen $MyPen$ 선언

2 $init_state$가 거짓일 경우: 메서드 종료
3 $MyPen \leftarrow$ 형식 `PS_SOLID`, 크기 10, 색상 `RGB(0, 0, 154)`인 펜 생성
4 pDC의 그리기 도구로 $MyPen$ 선택
5 그리기 도구를 $(gXmin, gYmin)$으로 이동, $(gXmax, gYmin)$까지 선 긋기
6 그리기 도구를 $(gXmin, gYmax)$으로 이동, $(gXmax, gYmax)$까지 선 긋기
7 $MyPen$ 삭제
8 $MyPen \leftarrow$ 형식 `PS_SOLID`, 크기 10, 색상 `RGB(0, 0, 0)`인 펜 생성
9 pDC의 그리기 도구로 $MyPen$ 선택
10 $0 \le i < mPoint_num$인 정수 i에 대해 반복:
  a $pt \leftarrow (gXmin, gYmax) + \left(20\,(mPoint_i)_x, -20\,(mPoint_i)_y\right)$
  b 그리기 도구를 pt로 이동, pt까지 선 긋기
11 $MyPen$ 삭제
12 $MyPen \leftarrow$ 형식 `PS_SOLID`, 크기 3, 색상 `RGB(180, 0, 0)`인 펜 생성
13 pDC의 그리기 도구로 $MyPen$ 선택
14 $0 \le i < mLine_num$인 정수 i에 대해 반복:
  a $l \leftarrow mLine_i$
  b $l_s \leftarrow (gXmin, gYmax) + (20 l_{xl}, -20 l_{yl})$
  c $l_e \leftarrow (gXmin, gYmax) + (20 l_{xr}, -20 l_{yr})$
  d 그리기 도구를 l_s로 이동, l_e까지 선 긋기
15 $MyPen$ 삭제

`void drawStartPoint(CDC* pDC)` : DC에 시작점을 그린다. 시간 복잡도는 $\mathcal{O}(1)$이다.

1 CPen $MyPen$ 선언
2 $init\_state$, $sele\_state$ 중 하나 이상이 거짓일 경우: 메서드 종료
3 $MyPen \leftarrow$ 형식 `PS_SOLID`, 크기 10, 색상 `RGB(255, 0, 0)`인 펜 생성
4 pDC의 그리기 도구로 $MyPen$ 선택
5 $pt \leftarrow (gXmin, gYmax) + \left(20\,(mPoint_{curr\_point})_x, -20\,(mPoint_{curr\_point})_y\right)$
6 그리기 도구를 pt로 이동, pt까지 선 긋기
7 $MyPen$ 삭제

`void drawWaterflow(CDC* pDC)` : DC에 물의 진행 방향을 그린다. 링크드 리스트 m_flow_point의 모든 원소를 순회하므로 시간 복잡도는 $\mathcal{O}(|m_flow_point|)$이다.

1 CPen $MyPen$ 선언
2 $init\_state$, $sele\_state$, $draw\_state$ 중 하나 이상이 거짓일 경우: 메서드 종료
3 $MyPen \leftarrow$ 형식 `PS_SOLID`, 크기 3, 색상 `RGB(0, 255, 255)`인 펜 생성
4 pDC의 그리기 도구로 $MyPen$ 선택
5 node 포인터 $temp$ 선언

6 $waterfall\_Solver()$
7 m_flow_point의 크기가 0일 경우: 메서드 종료
8 $temp \leftarrow m_flow_point$의 첫 노드
9 $curr \leftarrow (gXmin, gYmax) + \left(20\left(temp_{point}\right)_x, -20\left(temp_{point}\right)_y\right)$
10 $temp \leftarrow m_flow_point$의 다음 노드가 NULL 포인터가 아닐 경우 반복:
  a $curr \leftarrow (gXmin, gYmax) + \left(20\left(temp_{point}\right)_x, -20\left(temp_{point}\right)_y\right)$
  b $curr$까지 선 긋기
11 $MyPen$ 삭제

2.2 과제

교재에 소개된 문제 해결 방법에 사용된 자료구조로는 $G = (V, E)$의 사이클을 체크하는 데 $\mathcal{O}\left(V^2\right)$의 시간이 걸린다.

하지만 그래프 G의 트리 여부 체크는 $\deg^-$를 이용한다면

$$(\text{G is connected}) \wedge \left(\text{card}\left\{i \middle| \deg^- V_i = 0\right\} = 1\right) \wedge \left(\text{card}\left\{i \middle| \deg^- V_i > 1\right\} = 0\right)$$

으로도 가능하다. 물론

$$(\text{G is connected}) \wedge (E = V - 1)$$

로도 가능하지만 트라이가 트리인지 그 connectivity를 검증하기 위해서는 $\deg^- V_i = 0$인 V_i부터 탐색을 시작해야 하므로 이를 만족하는 V_i를 구해야 할 필요성이 요구된다. 이 방법을 이용해 트리 여부를 검증한다면 G를 완전히 탐색하는 데 BFS, DFS 등의 방법을 이용하여 $\mathcal{O}(V + E)$가, 모든 $V_i \in G$에 대해 $\deg^- V_i$를 계산하는 데 $\mathcal{O}(E)$가 걸리므로 이 때의 시간 복잡도는 $\mathcal{O}(V + E)$이다.

입력은 각 간선을 이루는 두 개의 정점으로만 주어지므로 $V \leq 2E$임이 자명하다. 따라서 $\mathcal{O}(V + E)$의 시간 복잡도를 갖는 알고리즘은 $\mathcal{O}\left(V^2\right)$의 시간 복잡도를 갖는 알고리즘보다 개선되었다고 할 수 있을 것이다.

자료구조는 C++ STL[Standard Template Library]의 `std::vector`, `std::tuple`, `std::queue`, `std::unorderd_map`을 사용하였다. `std::unorderd_map`은 키를 해싱해 저장하는 Hash Map을 구현한 것으로 해시 충돌이 없을 경우 읽기/쓰기에는 $\mathcal{O}(1)$이 걸린다.

그래프 정보를 담는 `graphs`의 자료구조는 `std::vector<std::unordered_map<int, std::vector<int>>>`이다.

`void Init()` : 값들을 초기화한다.

1 $currCase \leftarrow 0$, $cases \leftarrow 0$, $graphs.clear()$

`void Read_File(const char* filename)` : 파일을 읽어서 그래프를 메모리에 인접 리스트 형태로 저장한다.

1 $fin \leftarrow$ 파일명이 $filename$인 파일의 std::ifstream 열기
2 $cases \leftarrow fin$에서 읽기
3 $graphs$의 크기를 $cases$로 조정
4 $0 \le tc < cases$인 정수 tc에 대해 반복:
  a $n \leftarrow fin$에서 읽기
  b $graph = graphs_{tc}$
  c $0 \le i < n$인 정수 i에 대해 반복:
    i $u, v \leftarrow fin$에서 읽기
    ii 리스트 $graph[u]$에 v 추가
5 fin 닫기

`char* Check_Tree(int tc)` : tc번째 테스트 케이스의 그래프가 트리인지 아닌지 판단한다. 시간 복잡도는 위에서 보였듯이 $G = (V, E)$에 대해 $\mathcal{O}(V + E)$이다.

1 $graph = graphs_{tc-1}$
2 $flag \leftarrow$ true, std::unordered_map<int, int> $indegree$, std::unordered_map<int, bool> $visit$ 선언
3 $graph$의 모든 $\{u, u \to v$의 간선이 존재하는 $v\}$에 대해 반복:
  a $indegree_u \leftarrow 0$, $visit_u \leftarrow$ false
  b $u \to v$의 간선이 존재하는 v에 대해 반복:
    i $indegree_v \leftarrow 0$, $visit_v \leftarrow$ false
4 $graph$의 모든 $\{u, u \to v$의 간선이 존재하는 $v\}$에 대해 반복:
  a $u \to v$의 간선이 존재하는 v에 대해 반복:
    i $indegree_v \leftarrow indegree_v + 1$
5 $indegree\_0\_count \leftarrow 0$, $root \leftarrow -1$ 선언
6 $indegree$의 모든 $\{u, \deg^- u\}$에 대해 반복:
  a $\deg^- u = 0$인 경우:
    i $root \leftarrow u$
    ii $indegree\_0\_count \leftarrow indegree\_0\_count + 1$
  b $\deg^- u > 1$인 경우:
    i $flag \leftarrow$ false
7 $indegree_0_count \neq 1$인 경우: $flag \leftarrow$ false
8 $flag$가 참인 경우:
  a std::queue<int> q 선언
  b q에 $root$ 추가, $visit_{root} \leftarrow$ true
  c q가 비어 있지 않을 경우 반복:
    i $u \leftarrow q$의 첫 번째 원소
    ii q의 첫 번째 원소 제거

iii 리스트 $graph_u$의 모든 원소 v에 대해 반복:

(1) $visit_v$가 참인 경우: 다음 반복 단계로 진행

(2) $visit_v \leftarrow$ true, q에 v 추가

d $visit$의 모든 $\{u, u$의 방문 여부$\}$에 대해 반복:

i u의 방문 여부가 거짓일 경우: $flag \leftarrow$ false, 반복 종료

9 $flag$가 참인 경우: 문자열 `Case` tc `is a tree.` 반환

10 $flag$가 거짓인 경우: 문자열 `Case` tc `is not a tree.` 반환

3 학습

실습과 과제를 진행하면서 다음과 같은 내용을 습득할 수 있었다.

- MFC의 구성 요소를 간단하게 이해할 수 있게 되었다.
- Visual Studio에서 MFC 프로젝트를 생성하고, VS의 여러 도구들을 이용해 유저 인터페이스를 구성하고 코드를 작성하는 등 Windows 어플리케이션을 개발해 볼 수 있었다.
- DC에 CPen을 이용해 원하는 두께와 색상의 점, 선분 등을 그리는 방법을 알 수 있었다.