

자료구조 스터디

[학생 위치정보 관리 프로그램]



인하대학교 컴퓨터정보공학과

12141567 양선미

010-8975-5515

pdsunmii@gmail.com

개요

■ 설계의 목적

- 셔틀버스 노선 계획을 위해 인접리스트로 그래프를 구현하여 학생위치정보 관리 프로그램을 설계하고 자료구조를 이해한다.

■ 요구사항

- 파일로부터 데이터 불러와서 학생과 도로정보를 그래프로 구축
- 표준입력으로 주어진 각 질의에 대한 답을 표준출력으로 출력

- 학생 추가 (질의 A S E, 출력 N)
 - A : 학생 추가 질의를 나타내는 기호
 - S : 학생의 학번 (중복 없음)
 - E : 학생의 이메일 주소
 - N : 그래프의 총 학생 수
- 학생정보 출력 (질의 P S, 출력 E C)
 - P : 학생정보 출력 질의를 나타내는 기호
 - C : 연결된 도로의 수
 - 학생이 존재하지 않는다면 "Not found" 출력
- 도로 추가 (질의 I W S V, 출력 R)
 - I : 도로 추가 질의를 나타내는 기호
 - W : 도로 번호 (중복없음)
 - V : 학생의 학번
 - R : 그래프의 총 도로 수
 - S나 V 학생이 존재하지 않는다면 "R Not found" 출력
- 학생 삭제 (질의 X S, 출력 N R)
 - X : 학생 삭제 질의를 나타내는 기호
 - 학생이 존재하지 않는다면 "N R Not found" 출력
- 도로 삭제 (질의 Z W, 출력 R)
 - Z : 도로 삭제 질의를 나타내는 기호
 - 도로가 존재하지 않는다면 "R Not found" 출력

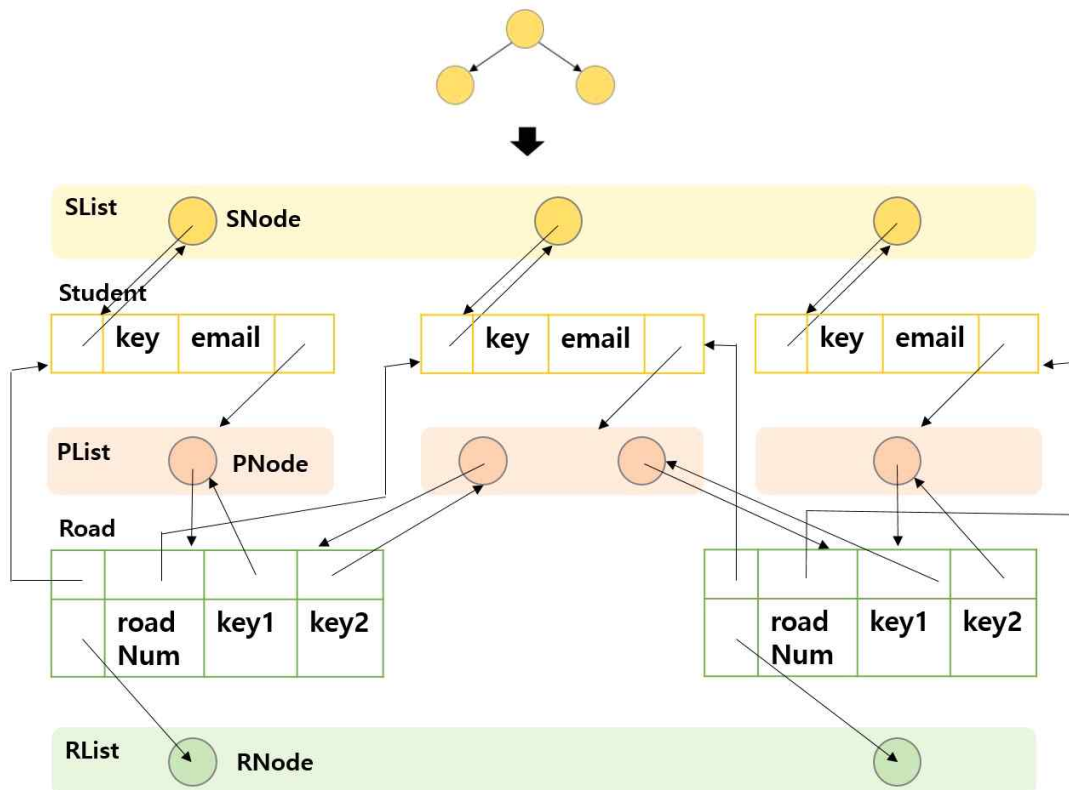
- 연결된 도로 출력 (질의 L S, 출력 E W1 W2 ... WN)
 - L : 연결된 도로 출력 질의를 나타내는 기호
 - E : 연결된 도로의 개수
 - Wi : 연결된 도로의 번호
 - 학생이 존재하지 않는다면 "Not found" 출력
- 도로정보 출력 1 (질의 F W, 출력 S V)
 - F : 도로정보 출력 1 질의를 나타내는 기호
 - 두 학생의 학번 중에서 학번이 작은 학생의 학번을 먼저 출력
 - 도로가 존재하지 않는다면 "Not found" 출력
- 도로정보 출력 2 (질의 O S W, 출력 V)
 - O : 도로정보 출력 2 질의를 나타내는 기호
 - S나 W가 존재하지 않는 경우와 S와 W를 이용할 수 없는 경우 "Not available" 출력
- 도로존재 여부 확인 (질의 K S V, 출력 W)
 - K : 도로존재 여부 확인 질의를 나타내는 기호
 - S와 V가 존재하지 않거나 연결하는 도로가 존재하지 않는 경우 "Not found" 출력
- 프로그램 종료 (질의 Q)
 - Q : 프로그램 종료 질의를 나타내는 기호

■ 개발환경

- 운영체제 : Windows 10 Home K
- 컴파일러 : Visual Studio 2013
- 언어 : C++

필요한 자료구조 및 기능

■ 자료구조



- 그래프

- 정점과 정점을 잇는 간선들과 정점들의 집합 ($G=(V, E)$)
- 간선에 방향이 존재하는 방향그래프와 존재하지 않는 무향그래프가 있으며 위는 무향그래프이다.
- 그래프를 표현하는 방법에는 크게 인접연결리스트로 표현하는 법과 인접행렬로 표현하는 방법 2가지가 있다.
- 위는 인접연결리스트로 그래프를 구현하였으며 이 표현법은 정점들의 sequence, 매개 sequence, 간선들의 sequence, 정점과 간선객체들로 구성된다.

■ 기능

- 파일 정보를 이용한 그래프 구축
 - SList, RList를 만들고 N번만큼 파일을 읽어서 학생 추가 함수 실행
 - M번만큼 파일을 읽어서 도로 추가 함수 실행
- 학생 추가
 - 새로운 Student 객체를 만들고 입력한 key값과 email값 입력
 - 객체의 구성요소 초기화
 - SList에 객체를 삽입하는 함수 실행
- 학생 정보 출력
 - SList에서 입력된 key값을 가지는 Student 객체를 가리키는 SNode를 검색
 - 요구사항에 맞게 학생 정보 출력
- 도로 추가
 - SList에서 입력된 key1, key2값을 가지는 Student 객체를 가리키는 SNode를 검색 후 s1, s2에 주소 저장
 - 새로운 Road 객체 생성 후 구성요소의 값들을 넣어주고 RList에 객체를 삽입하는 함수 실행
 - 상황에 맞게 전 객체의 PList와 후 객체의 PList에 새로운 PNode를 추가하는 함수 실행
 - 요구사항에 맞게 출력

- 학생 삭제

- SList에서 입력된 key값을 가지는 Student 객체를 가리키는 SNode를 검색 후 tmp에 주소 저장
- 상황에 맞게 tmp를 SList에서 삭제
- 상황에 맞게 Student 객체의 PList에 있는 PNode를 따라가면서 인접학생들의 PNode들과 Road가 가리키는 RNode 삭제 후 Road 삭제
- Student 객체가 가리키는 PList 삭제 후 Student 객체 삭제
- 요구사항에 맞게 출력

- 도로 삭제

- RList에서 입력된 roadNum값을 가지는 Road 객체를 가리키는 RNode를 검색 후 tmp에 주소 저장
- 상황에 맞게 정점들과 연결된 PNode를 삭제
- tmp가 가리키는 Road 삭제
- 상황에 맞게 RList에서 RNode 삭제
- 요구사항에 맞게 출력

- 연결된 도로 출력

- SList에서 입력된 key값을 가지는 Student 객체를 가리키는 SNode를 검색 후 tmp에 저장
- Student 객체가 가리키는 PList에 있는 모든 PNode들이 가리키는 Road의 RoadNum 출력
- 요구사항에 맞게 출력

- 도로 정보 출력 1

- RList에서 입력된 roadNum값을 가지는 Road 객체를 가리키는 RNode를 검색 후 tmp에 주소 저장
- tmp가 가리키는 Road의 전 객체와 후 객체의 key값을 비교
- 요구사항에 맞게 출력

- 도로 정보 출력 2

- SList에서 입력된 key값을 가지는 Student 객체를 가리키는 SNode를 검색 후 s에 저장
- Student 객체가 가리키는 PList의 PNode들을 검색하면서 각각의 PNode들이 가리키는 Road의 roadNum과 입력된 roadNum값이 같은 Road가 존재 하는지 확인
- 요구사항에 맞게 출력

- 도로존재 여부 확인

- SList에서 입력된 key1, key2값을 가지는 Student 객체를 가리키는 SNode를 검색 후 s1, s2에 주소 저장
- s1,s2 중에 차수가 작은 쪽이 가리키는 PList의 PNode들을 보면서 각각이 가리키는 Road의 전/후 객체가 큰 쪽이라면 roadNum을 출력
- 요구사항에 맞게 출력

- 종료

- query를 Q를 받는다면 exit(0)

기능별 알고리즘 명세

■ 파일 정보를 이용한 그래프 구축

- 수도코드

Algorithm Graph(filename)

 read N M with filename

 create SList, RList

 for i <- 0 to N-1

 read StudentInfo with filename

 addStu(key, email)

 for l <- 0 to M-1

 read RoadInfo with filename

 addRoad(roadNum, key1, key2, 0)

시간복잡도.

: $O(N+M)$

addStu 함수는 최악수행시간이 $O(1)$ 이고 addRoad 함수는 최악수행시간이 $O(\text{sz of slist})$ 인데 SList의 size 그래프 구축 시 N이기 때문에 for문 도는 시간과 합치면 $2N+M$ 이고 시간복잡도는 $O(2N+M)$ 이 된다.

■ 학생 추가

- 수도코드

Algorithm addStu(key, email)

```
create Student s
s <- key, email
initialize plist of s
create snode of s
insert(s) to slist
return sz of slist
```

- 시간복잡도

: $O(1)$

insert함수는 $O(1)$ 이 걸리고 나머지 과정도 상수시간에 수행될 수 있으므로 시간복잡도는 $O(1)$ 이 된다.

■ 학생정보 출력

- 수도코드

Algorithm printInfo(key)

```
tmp <- head of slist
while tmp is not NULL
    if key of student of tmp is key
        print "email sz of plist"
    tmp <- next of tmp
if tmp is NULL
    print "Not found"
```

- 시간복잡도

: $O(\text{degree of } s)$ (s 는 key를 키로 갖는 vertex)

최악의 경우 입력된 key값과 같은 Student 객체가 가리키는 PList를 모두 검색하기 때문에 PList의 size만큼, 즉 Student 객체의 차수만큼의 시간복잡도를 가진다.

■ 도로 추가

- 수도코드

Algorithm addRoad(roadNum, key1, key2, email)

tmp<-head of slist

while tmp is not NULL

if key of student of tmp is key1

s1<- student of tmp

if s2 is not NULL

break

if key of student of tmp is key2

s2<- student of tmp

if s1 is not NULL

break

if s1 is NULL or s2 is NULL

print "sz of rlist Not found"

else

create Road r

r<- roadNum, key1, key2, s1, s2

create prepnod, nextpnod, rnode of r

insert(r) to rlist

if plist of prepu of r is NULL

create newplist

```

        plist of prestu of r <- newplist
        insert(newplist, r, 0) to newplist
    else
        insert(plist, r, 0) to plist of prestu of r
    if plist of nextstu of r is NULL
        create newplist
        plist of nextstu of r <- newplist
        insert(newplist, r, 1) to newplist
    else
        insert(plist, r, 1) to plist of nextstu of r
    if q is 1
        print sz of rlist

```

- 시간복잡도

: $O(n)$ ($n=|V|$)

최악의 경우 SList의 모든 SNode들과 비교 후에 s1과 s2를 찾을 수 있기 때문에 시간복잡도는 $O(n)$ 이다.

■ 학생 삭제

- 수도코드

Algorithm deleteStu(key)

tmp <- head of slist

while tmp is not NULL

if key of student of tmp is key

break

tmp <- next of tmp

if tmp is NULL

print "sz of slist sz of rlist Not found"

else

s <- student of tmp

delete tmp and link others in slist

sz of slist--

if plist of s is NULL

print "sz of slist sz of rlist"

else

tmp <- head of plist of s

while tmp is not NULL

delete pnode of adjacent student

sz of plist of adjacent student--

delete rnode and link others in rlist

sz of rlist--

delete road of tmp

tmp <- next of tmp

print sz of slist sz of rlist

delete plist of s, s

- 시간복잡도

: $O(n + \text{degree of } s)$

최악의 경우 SList의 모든 SNode들과 비교 후에 원하는 학생 s 를 찾을 수 있기 때문에 sz of slist만큼 시간이 걸릴 수 있고 그 s 의 PList의 sz만큼 보면서 각각을 삭제해야 하기 때문에 $O(n + \text{degree of } s)$ 이 된다.

■ 도로 삭제

- 수도코드

Algorithm deleteRoad(roadNum)

tmp <- head of rlist

while tmp is not NULL

if roadNum of road of tmp is roadNum

break

tmp <- next of tmp

if tmp is NULL

print "sz of rlist Not found"

else

delete prepnod and link others in plist

sz of plist of prestu--

delete nextpnod and link others in plist

sz of plist of nextstu--

delete road of tmp

delete rnode and link others in rlist

sz of rlist--

print "sz of rlist"

- 시간복잡도

: $O(m)$ ($m=|E|$)

최악의 경우 모든 RList의 RNode의 Road 값과 비교해서 RoadNum과 맞는 Road를 찾을 수 있기 때문에 $O(m)$ 이 된다.

■ 연결된 도로 출력

- 수도코드

Algorithm printLinkedRoad(key)

tmp <- head of slist

while tmp is not NULL

if key of student of tmp is key

break

tmp <- next of tmp

if tmp is NULL

print "Not found"

else if plist of student of tmp is NULL

print "0"

else

print "sz of plist of student of tmp"

p <- head of plist of student of tmp

while p is not NULL

print " roadNum"

break

print "endl"

- 시간복잡도

: $O(n + \text{degree of } s)$

최악의 경우 SList의 모든 SNode들과 비교 후에 원하는 학생 s 를 찾을 수 있기 때문에 sz of slist만큼 시간이 걸릴 수 있고 그 s 의 PList의 sz만큼 보면서 각각이 가리키는 Road들의 roadNum을 출력할 수 있기 때문에 총 $O(n + \text{degree of } s)$ 가 걸린다.

■ 도로정보 출력 1

- 수도코드

Algorithm printRoad(roadNum)

tmp <- head of rlist

while tmp is not NULL

if roadNum of road of tmp is roadNum

break

tmp <- next of tmp

if tmp is NULL

print "Not found"

else if key of prestu > key of nextstu

print "key of nextstu key of prestu"

else

print "key of prestu key of nextstu"

- 시간복잡도

: $O(m)$ ($m=|E|$)

최악의 경우 모든 RList의 RNode의 Road 값과 비교해서 RoadNum과 맞는 Road를 찾을 수 있기 때문에 $O(m)$ 이 된다.

■ 도로정보 출력 2

- 수도코드

Algorithm printStu(key, roadNum)

 s <- head of slist

 while s is not NULL

 if key of student of s is key

 break

 s <- next of s

 if s is NULL

 print "Not available"

 else

 p <- head of plist of student of s

 while p is not NULL

 if roadNum of road of p is roadNum

 break

 p <- next of p

 if p is NULL

 print "Not available"

 else

 if key of prestu is key

 print "key of nextstu"

 else

 print "key of prestu"

- 시간복잡도

: $O(n + \text{degree of student of } s)$

최악의 경우 SList의 모든 SNode들과 비교 후에 원하는 학생을 찾을 수 있기 때문에 sz of slist만큼 시간이 걸릴 수 있고 그 학생의 PList의 sz만큼 보면서 roadNum을 가지는 Road를 찾을 수 있기 때문에 $O(n + \text{degree of student of } s)$ 가 걸린다.

■ 도로존재 여부 확인

- 수도코드

Algorithm exist(key1, key2)

tmp<-head of slist

while tmp is not NULL

if key of student of tmp is key1

s1<- tmp

if s2 is not NULL

break

if key of student of tmp is key2

s2<- tmp

if s1 is not NULL

break

tmp <- next of tmp

if s1 is NULL or s2 is NULL

print "Not found"

else

if plist of student of s1 is NULL or plist of student of s2 is NULL

print "Not found"

else if sz of plist of student s1 is larger than other

p <- head of plist of student of s2

while p is not NULL

if key1 is key of adjacent student

print "roadNum"

break

p <- next of p

if p is NULL

print "Not found"

else

p <- head of plist of student of s1

while p is not NULL

if key2 is key of adjacent student

print "roadNum"

break

p <- next of p

if p is NULL

print "Not found"

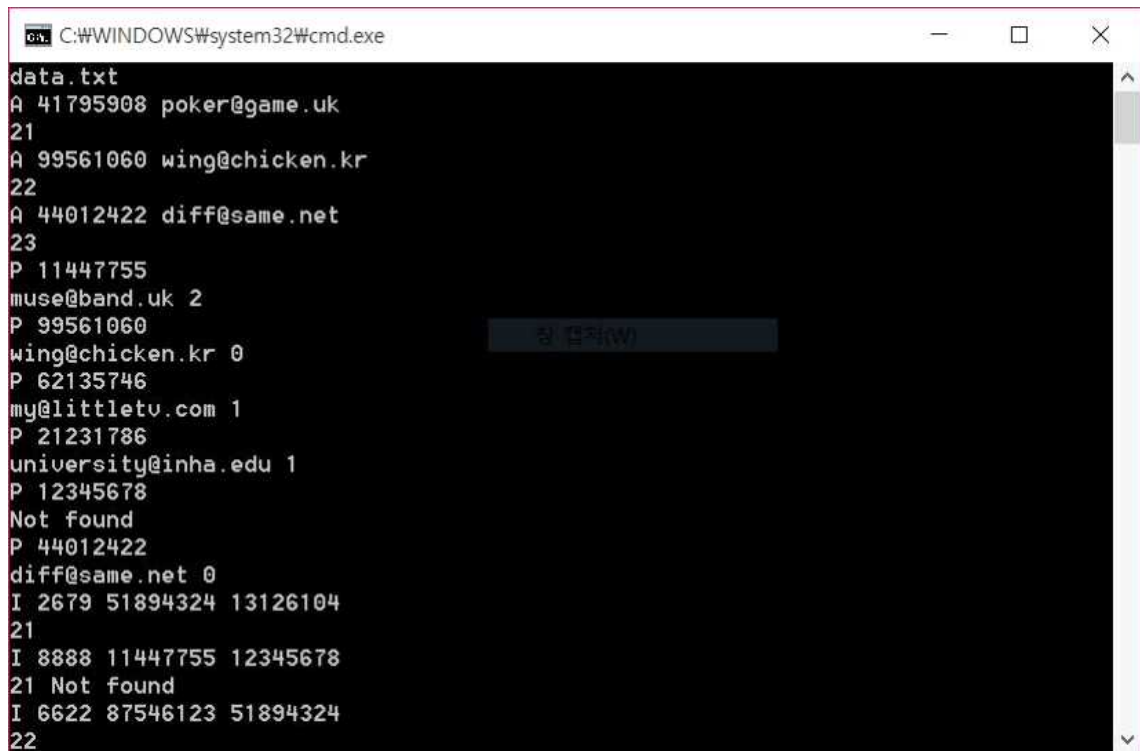
- 시간복잡도

: $O(n + \min(\text{degree of } s1, \text{degree of } s2))$

최악의 경우 SList의 모든 SNode와 비교해야 s1과 s2를 찾을 수 있고 s1과 s2 중에 plist의 sz가 작은 쪽의 plist를 보면서 roadNum과 같은 값을 가지는 road를 찾으면 시간복잡도는 $O(n + \min(\text{degree of } s1, \text{degree of } s2))$ 가 된다.

인터페이스 및 사용법

■ 스크린샷



```
C:\WINDOWS\system32\cmd.exe
data.txt
A 41795908 poker@game.uk
21
A 99561060 wing@chicken.kr
22
A 44012422 diff@same.net
23
P 11447755
muse@band.uk 2
P 99561060
wing@chicken.kr 0
P 62135746
my@littletv.com 1
P 21231786
university@inha.edu 1
P 12345678
Not found
P 44012422
diff@same.net 0
I 2679 51894324 13126104
21
I 8888 11447755 12345678
21 Not found
I 6622 87546123 51894324
22
```

- query.txt의 일부분을 입력했을 때의 결과이다.

■ 사용법

- 앞서 요구사항에 설명한대로 질의형식대로 입력을 주면 그에 대한 출력이 나올 것이다.

평가 및 개선사항

■ 본 결과의 장점 및 단점

- 장점 : 인접행렬로 구현한 것에 비해 정점과 정점 사이의 간선들의 수가 적을 경우에 공간 면에서 효율적이다. 또한 학생 추가 함수의 최악 수행시간이 인접행렬로 구현 시 $O(n^2)$ 이 걸리기 때문에 비교적 매우 빠르다. 따라서 학생 추가 기능을 많이 사용하는 어플리케이션에 사용하면 유리하다.
- 단점 : 도로존재여부 확인 함수의 경우 인접행렬로 구현 시 $O(n)$ 이면 되기 때문에 비교적 느리다. 따라서 도로 존재 여부 기능을 많이 사용하는 어플리케이션에 사용하면 좋지 않다.

■ 향후 개선방향

- 해시테이블을 이용하여 SList와 RList, PList를 구현하면 key값 또는 roadNum값 등을 탐색할 때 평균수행시간이 $O(1)$ 으로 빨라져 거의 모든 기능의 수행시간이 빨라질 수 있다.