14 가중치 그래프 ADT

목적

이번 실습에서 여러분은

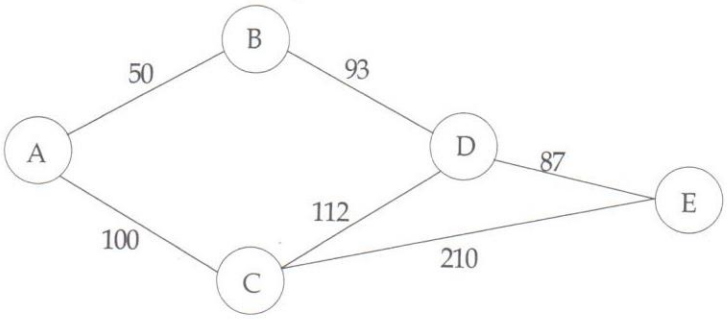
- 가중치 그래프 ADT를 정점 리스트와 인접 행렬를 사용해서 구현한다.
- 그래프에서 정점들의 간의 최소 비용 경로를 찾는 루틴을 개발한다.
- 정점을 추가하고 그래프가 적절한 색채인지를 체크하는 함수를 구현한다.
- 부적절한 색채가 5가지색 미만으로 만들 수 있는 그래프 생성하여 4색의 원리를 조사하라

개요

많은 관계는 선형(liner) 또는 계층적 데이터 구조를 사용해서 쉽게 표현될 수 없다. 고속 네트워크로 연결된 도시들간의 관계는 이런 관계들 중에 하나이다. 고속 네트워크에서 경로들에 대해 장소들의 관계가 선형 또는 계층적이라고 설명하는 것은 가능하다. 우리는 대다수의 고속 네트워크가 선형도 계층적이지도 않다는 것을 아는데 충분한 횟수만큼 원을 방문했다. 우리가 필요한 것은 네트워크에서 각 도시와 다른 도시들을 연결해 주는 데이터 구조이다. 이 자료구조의 타입은 그래프로 언급된다.

트리와 그래프는 노드들의 집합으로 이루어져 있고(정점들로 불리는) 간선의 집합으로 이루어져 있다. 트리와 달리 그래프내의 간선은 단순하게 부모와 자식을 연결

하는 것이 아니고 어떠한 정점들의 쌍과 연결할 수 있다.



그래프에서 각 정점들은 유일한 특별한 도시를 표시하는 유일한 라벨을 갖는다. 각 간선은 각 경로 순회의 비용(거리, 시간, 돈의 단위로 측정)을 표시하는 가중치를 갖는다. 그래프내의 간선은 무방향이다. 즉 정점 A와 B를 연결하는 간선이 있다면 이 간선은 A에 B로 또는 B에서 A로 이동시키는데 사용된다. 가중, 무방향 그래프는(weighted, undirected graph) 고속 네트워크에서 경로들을 사용하여 도시 사이의 순회 비용을 표현한다. 이 실습에서 여러분은 가중, 무방향 그래프들의 구현과 응용에 초점을 둔다.

가중치 ADT

원소

각 정점은 그래프에서 라벨(타입 char*)을 갖는다. 정점은 추가 데이터를 포함할 수 있다.

구조

그래프내에서 정점 사이의 관계인 각 간선은 정점들의 한 개의 쌍을 연결하는 무방향으로 연결된 간선들의 집합을 사용하여 표현된다.

기능

WtGraph (int maxnumber = defMaxGraphSize)

요구사항:

없음.

결과:

객체 생성자. 빈 그래프를 생성한다. MaxNumer 정점들을 갖는 그래프에 대해 충분한 메모리를 할당한다.

~WtGraph ( )

요구사항:

없음.

결과:

객체 파괴자. 그래프를 적재하는데 사용된 메모리 할당을 해제한다.

void insertVertex (Vertex newVertex)

요구사항:

그래프는 가득 차 있지 않다.

결과:

그래프에 정점을 삽입한다. 정점이 이미 그래프내에 존재하면 그 정점을 갱신한다.

void insertEdge (char *v1, char *v2, int wt)

요구사항:

그래프는 정점 v1과 v2를 포함한다.

결과:

정점 v1과 v2를 연결하는 무방향 간선을 그래프에 삽입한다. 간선의 가중치는 wt이다. 만일 이들 정점들을 연결하는 간선이 이미 있으면 그 간선의 가중치를 갱신한다.

int retrieveVertex (char *v, Vertex &vData) const

요구사항:

없음.

결과:

정점 v에 대해 그래프를 검색한다. 이 정점이 발견되면 정점의 데이터를 vData에 복사하고 1을 반환한다. 발견하지 못할시는 vData를 정의하지 않고 0을 반환한다.

int edgeWeight (char *v1, char *v2, int &wt) const

요구사항:

그래프는 정점 v1과 v2를 포함한다.

결과:

v1과 v2를 연결하는 간선에 대한 그래프를 검색한다. 만일 이 간선이 존재한다면 1과 간선의 가중치 wt를 반환한다. 그렇지 않은 경우는 정의되지 않은 wt와 0을 반환한다.

void removeVertex (char *v)

요구사항:

그래프는 간선 v를 포함한다.

결과:

그래프에서 정점을 제거한다.

void removeEdge (char *v1, char *v2)

요구사항:

그래프는 정점 v1과 v2를 포함한다.

결과:

그래프에서 정점 v1과 v2를 연결하는 간선을 제거한다.

void clear ( )

요구사항:

없음.

결과:

모든 정점과 간선을 그래프에서 제거한다.

int empyt ( ) const

요구사항:

없음.

결과:

그래프가 비어 있으면(정점이 없으면)1을 반환, 그렇지 않으면 0을 반환.

int full ( ) const

요구사항:

없음

결과:

그래프가 가득 차 있으면 1을 반환 그렇지 않으면 0을 반환

void showStructure ( ) const

요구사항:

없음.

결과:

배열 형태에 정점들을 갖는 그래프와 근접 행렬 형태에 있는 간선을(가중치와 함께) 출력한다. 만일 그래프가 비어 있으면 "Empty graph"를 출력한다. 이 기능은 시험와 디버깅을 목적으로 사용된다.

실습 14 : 표지

날짜_____ 구분_____

성명_____

강사가 여러분에게 할당한 다음에 연습문제들에 할당된 열에 체크표시를 하고 이 겉장을 앞으로의 실습에서 여러분이 제출할 과제물 앞에 붙이시요

	할 일	완 료
실습 전 연습	V	
연 결 연습	V	
실습 중 연습 1		
실습 중 연습 2		
실습 중 연습 3		
실습 후 연습 1		
실습 후 연습 2		
	총 점	

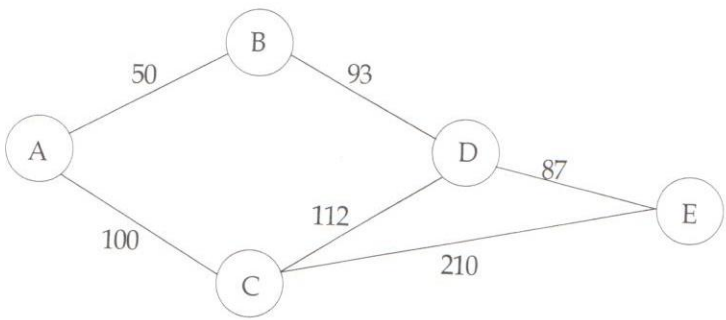
실습 14 : 실습 전 연습

날짜_____ 구분_____

성명_____

여러분은 여러 방법으로 그래프를 표현할 수 있다. 이번 실습에서 여러분은 정점들의 집합을 저장하는데 배열을 간선의 집합을 저장하는데 인접 행렬을 사용한다. 인접 행렬의 원소(j, k)는 인덱스 j 정점에서 인덱스 k 정점으로 가는 간선에 대한 정보를 갖는다. 가중 그래프에서 각 행렬 원소는 간선에 대응하는 가중치를 갖는다. 특별하게 선택된 가중치는 그래프에서 없는 간선을 가리키는데 사용된다.

다음 그래프는



정점 리스트와 아래에 있는 인접 행렬을 얻는다. 데쉬(dash)는 그래프에 없는 간선을 표시한다.

정점 리스트	
색인	표시
0	A
1	B
2	C
3	D
4	E

인접 행렬					
시작 \ 종점	0	1	2	3	4
0	-	50	100	-	-
1	50	-	-	93	-
2	100	-	-	112	210
3	-	93	112	-	87
4	-	-	210	87	-

정점 A는 배열 인덱스가 0이고, 정점 C는 배열 인덱스가 2이다. 인접 행렬에서 정점 A에서 C로의 간선의 가중치는 인접 행렬의 원소(0, 2)에 저장된다.

1단계 : 정점들 저장의 배열과 간선들 저장에 인접 행렬을 사용하는 가중치 그래프 ADT에서 기능을 구현하라. 그래프에서 정점의 수는 고정되어 있지않다. 따라서 여러분은 그래프 내의 실제 정점(size)의 수 뿐만 아니라 그래프가 가질 수 있는 정점의 최대 갯수(maxSize)를 저장할 필요가 있다. 파일 *wtgrph.h*에 있는 다음의 선언을 기본으로 구현하라. ShowStructure 기능의 구현은 파일 *show14.cpp*에 있다.

```
class Vertex
{
  public:

    char label [vertexLabelLength];   // Vertex label
};

class WtGraph
{
  public:

    // Constructor
    WtGraph (int maxNumber = defMaxGraphSize);

    // Destructor
    ~WtGraph ( );

    // Graph manipulation operations
    void insertVertex (Vertex newVertex); // Insert vertex
    void insertEdge (char *v1, char *v2, int wt); // Insert
                                                  // edge
    int retrieveVertex (char *v, Vertex &vData) const;
                                          // Get vertex
    int edgeWeight (char *v1, char *v2, int &wt) const;
                                          // Get edge wt.
    void removeVertex (char *v);          // Remove vertex
    void removeEdge (char *v1, char *v2); // Remove edge
    void clear ( );                       // Clear graph

    // Graph status operations
    int empty ( ) const;                  // Graph is empty
```

```
    int full ( ) const;                    // Graph is full

    // Output the graph structure -- used in testing/debugging
    void showStructure ( ) const;

    // In-lab operation
    int allEven ( ) const;

  private:

    // Facilitator functions
    int index (char *v) const; // Converts vertex label to
                               // an adjacency matrix index
    int& edge ( int row, int col ); // Set/get edge weight
                                    // using adjacency
                                    // matrix indices

    // Data members
    int maxSize,  // Maximum number of vertices in the graph
        size;     // Actual number of vertices in the graph
    Vertex *vertexList;  // Vertex list
    int *adjMatrix;      // Adjacency matrix
};
```

여러분의 public 멤버 함수 구현은 인접 행렬내의 원소들을 접근하기 위해 **edge( )** 함수를 사용해야 한다. 예를 들어 할당 문

```
edge(2,3) = 100;
```

은 edge( ) 함수를 가중치 100을 두 번째 열과 인접 행렬의 세 번째 열 원소에 할당하는데 사용하고 if 문장

```
If (edge(j,k) == infiniteEdgeWt)
    cout << "Edge is missing from graph" << end1;
```

가 인덱스가 j인 정점과 k인 정점을 연결하는 간선이 있는지 시험하기 위해 이 함수를 사용한다.

2단계 : 가중 그래프 ADT의 구현을 저장하라 파일 *wtgrap.cpp*에 저장하라. 여러분의 코드를 문서로 확인하라.

실습 14 : 연결 연습

날짜_____ 구분_____

성명_____

여러분의 강사와 함께 여러분이 이 연습을 실습 기간 동안이나 실습 기간 중에 완료할 수 있는지 체크하라.

*test14.cpp*의 시험 프로그램은 다음의 명령들을 사용하여 여러분이 구현한 가중치 그래프 ADT에 대한 상호적인 시험을 허용한다.

명령어	동작
+v	정점 V를 삽입
=v w wt	정점 v와 w를 연결하는 절선을 삽입. 이 절선의 가중치는 wt
?v	v를 가져온다(retrieve)
#v w	정점 v와 w를 연결하는 절선을 가져오고 그 가중치를 반환
-v	정점 v를 제거한다.
!v w	정점 v와 w를 연결하는 절선을 삭제
E	그래프가 비었는지 조사
F	그래프가 가득 찼는지 조사
C	그래프를 clear
Q	테스트 프로그램을 종료

v와 w가 개별적인 문자들이 아닌 정점 레이블(type char*)을 표시하는 것을 주의하라. 여러분은 이 명령어를 공백을 포함해서 위에 나타난 정확한 형태를 사용하여 주의깊게 입력해야 한다.

1단계 : 가중치 그래프 ADT의 여러분의 구현에 대한 시험 계획을 준비하라. 여러분의 시험 계획은 그래프내에서 여러 방법으로 연결된 정점들을 다루어야 한다. 존재하지 않는 정점과 간선을 가져오는 시험 항목을 포함하여라. 시험 계획 형태는 다음과 같다.

2단계 : 여러분의 시험 계획을 실행하라. 구현에 잘못이 발견되면 수정하고 다시 실행하라.

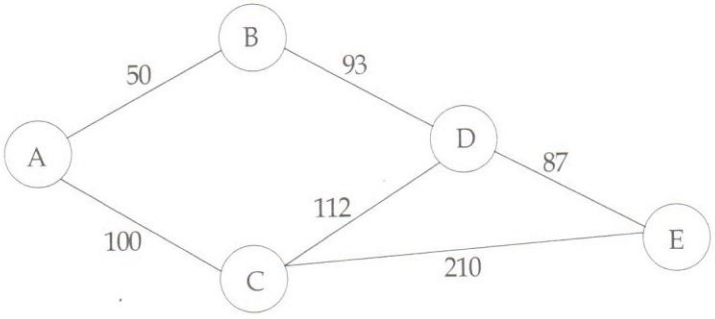
가중치 그래프 ADT에서 기능들에 대한 시험 계획			
시험 항목	명령어	예상 결과	검사

실습 14 : 실습 중 연습 1

날짜_____ 구분_____

성명_____

많은 가중치 그래프의 많은 응용에서 여러분은 한 쌍의 정점을 연결하는 간선이 있는지를 결정하는 것이 아니라 정점들을 연결한 경로가 있는지 결정할 필요가 있다. 확장된 인접 행렬 개념을 확장해서 여러분은 경로 행렬을 생성할 수 있다. 인덱스 j인 정점에서 인덱스 k인 정점으로의 최소(최단)경로의 비용을 갖는 원소(j, k) 경로 행렬을 생성할 수 있다. 다음의 그래프는



아래의 경로 행렬을 얻는다.

정점 리스트	
색인	표시
0	A
1	B
2	C
3	D
4	F

경로 행렬					
시작 \ 종점	0	1	2	3	4
0	0	50	100	143	230
1	50	0	150	93	180
2	100	150	0	112	199
3	143	93	112	0	87
4	230	180	199	87	0

이 그래프는 정점 A에서 정점 E의 몇 개의 경로를 갖는다. 이 정점을 연결하는 최소 비용 경로는 원소(0, 4)에 저장된다. 0은 정점 A의 인덱스이고 4는 정점 E의 인덱스이다. 대응하는 경로는 ABDE이다.

이 경로 행렬을 생성함에 있어 우리는 한 정점에서 그 자신으로 가는 비용이 '0'인 경로가 존재한다고 가정한다. 즉 form(j, j)의 원소들이다. 이 가정은 한 정점에서 그 자신의 정점으로 가는 운행은 사건이 아니라는 시각을 바탕으로 하고 따라서 비용은 없다. 그래프에 정보를 어떻게 적용하려는지에 따라 여러분은 여러개의 가정을 사용할 수 있다.

그래프에 대해 주어진 인접 행렬에서 우리는 모든 간선이 경로인 것으로 경로 행렬을 구성할 수 있다. 한 개의 간선을 갖는 경로들은 다음의 이유를 적용함으로써 두 개의 간선을 갖는 경로로 형성되도록 합성된다..

```
if 정점 j에서 정점 m으로 가는 경로가 있고
   정점 m에서 정점 k로 가는 경로가 있으면
then 정점 j에서 정점 k로 가는 경로가 존재한다.
```

우리는 더 많은 간선을 구성하는 경로 형태를 만들기 위해 이 같은 이유를 이렇게 새로 발생된 경로에 적용할 수 있다. 이 과정에서 핵심은 완전하고 효과적인 방법으로 경로들을 열거하고 결합시킨다는 것이다. 이 작업에 대한 한 방법은 다음의 알고리즘에서 설명한다. Warshall의 알고리즘이라고 알려진 변수 j, k와 m은 정점은 정점 label이 아닌 인덱스들을 참조한다.

경로 행렬이 간선 행렬과 같도록 하기위해 경로 행렬을 초기화한다(모든 간선은 경로이다)

한 정점에서 자기 자신으로 비용이 '0'인 하나의 경로를 생성한다.

```
for(m=0; m < size; m++)
   for(j =0; j < size; j++)
      for(k=0; k<size; k++)
         if 정점 j에서 정점 m으로 가는 경로가 있고
         정점 m에서 정점 k로 가는 경로가 있으면
         then 정점 j에서 정점 k로 가는 경로를 경로 행렬에 추가한다.
```

이 알고리즘은 비용이 아닌 정점 간의 경로의 존재를 만든다. 다행히 위의 사용된 이유를 확장하여 여러분은 쉽게 정점간의 최소 비용을 결정할 수 있다.

```
if 정점 j에서 정점 m으로 가는 경로가 있고
정점 m에서 정점 k로 가는 경로가 있고
j에서 m k로 가는 경비는 경로 행렬에서 원소(j,k)보다 작다.
then  원소(j,m)과 (m,k)의 합으로 원소(j,k)를 대치한다.
```

이러한 이유를 이전의 알고리즘에 결합해서 **Floyd's 알고리즘**으로 알려진 다음의 알고리즘을 얻는다.

경로 행렬을 초기화하면 간선 행렬과 같다(모든 간선은 경로이다).

각 정점 자신으로 가는 정점에서 비용이 0인 경로를 생성하라.

```
for(m=0; m<size; m++)
   for(j=0; j<size; j++)
      for(k=0; k<size; k++)
         if 정점j에서 정점 m으로 가는 경로가 존재하고
        정점 m에서 정점 k로 가는 경로가 존재하고
         원소(j,m)과 (m,k)의 합은 경로 배열의 원소(j,k)보다 작다.
        Then 원소(j,m)과 (m,k)의 합으로 원소(j,k)를 대치한다.
```

다음의 가중 그래프 ADT 기능은 그래프의 경로 행렬을 계산한다.

Void computePath ( )

요구사항:

없음.

결과:

그래프의 경로 행렬을 계산한다.

1단계 : 데이터 멤버

```
Int *pathMatrix:        // Path matrix
```

와 함수 프로토타입

```
void computePaths ( );   // Computes path matrix
```


을 파일 *wtgraph.h*에 있는 WtGraph 클래스 선언에 추가한다.

2단계 : 위에서 서술한 computePaths 기능을 구현하고 파일 *wtgraph.cpp*에 추가한다.

3단계 : 정점 리스트와 인접행렬을 추가하여 그래프의 경로 행렬을 출력하는 showStructure( ) 함수로 파일 *wtgraph.cpp*에 있는 showStructure( ) 함수를 대치하라.

4단계 : 시험 프로그램 *test14.cpp*에서 "//PM"으로 시작되는 행에서 주석 경계자와 문자 'PM'을 제거하여 'PM' 시험을 활성화하라.

5단계 : 다양한 가중치를 가지고 여러 방법으로 연결되는 정점을 갖는 그래프를 포함하는 computePath 기능에 대한 시험 계획을 준비하라. 한 쌍의 정점 사이의 간

선이 동일한 정점 사이의 다중 경로보다 높은 비용을 갖는다는 것을 시험 항목에 포함하도록 한다. 위에서 보여준 그래프에서 간선 CE와 경로 CDE는 이러한 성질을 갖는다. 시험 계획은 다음과 같다.

6단계 : 여러분의 시험 계획을 실행하라. ComputePaths 구현에 잘못이 발견되면 수정하고 다시 실행하라.

computePaths 기능에 대한 시험 계획			
시험 항목	명령어	예상 결과	검사