Project2

통신네트워크

Contents

Introduction

Introduction

■ 목적

- Dijkstra algorithm과 Bellman-ford algorithm을 이용하여 노드들 사이의 최단경로를 구해보자
  - 코드의 ? 부분을 채워 Dijkstra algorithm과 Bellman-ford algorithm을 구현

■ 제출

- 제출 기한
  - 11/23 (수) 23시 59분까지
- 제출 파일 : 압축폴더 (Project2_학번_이름.zip)로 제출
  - PDF (Project2_학번_이름.pdf)
    - Data1의 경로에서 노드 A를 소스 노드로 지정했을 때 나온 최단경로들
      (Dijkstra 알고리즘과 Bellman-ford 알고리즘 둘 다 보여줄 것)
    - Data 1의 경로를 통해 Dijkstra 알고리즘과 Bellman-ford 알고리즘 동작 설명
      - » Forwarding table을 그려서 보여줄 것
    - Bellman-ford 알고리즘에서 노드 C에서의 weight가 바뀐 후 최단 경로를 업데이트하는 원리 설명
  - Dijkstra 관련 파일들
    - dijkstra.c, dijkstra.h, graph.c, graph.h, main.c, data1, data2, data3, data4
  - Bellman-ford 소스 파일
    - bellman.c, bellman.h, graph.c, graph.h, main.c, data1, data2, data3, data4

Dijkstra Algorithm

- **Algorithm**
  - 소스 노드와 다른 노드들 사이의 최단 경로를 구하는 centralized 알고리즘
  - 소스 노드와 다른 노드들 사이의 최단 경로의 거리를 cost라는 변수로 표현한다.
  - **Assumption**
    - 데이터는 (node 개수, edge 개수, edge 1, weight 1, … , edge k, weight k)로 구성
    - 두 노드 사이의 weight는 방향에 따라 다를 수 있다.
      - Ex. 노드 A에서 노드 B로 가는 거리는 4, 노드 B에서 노드 A로 가는 거리는 3으로 다를 수 있다.
  - 실행
    - 최종적으로 소스 노드를 선택하고 그 소스 노드로부터 각 노드까지의 최단경로 구할 것

Dijkstra Algorithm

- **Initialize**
  - Forwarding table을 초기화하기 위한 함수
    - 소스 노드를 제외한 모든 노드들의 cost는 INF
    - 소스 노드를 제외한 모든 노드들의 predecessor 노드는 아무것도 없는 형태로
    - 소스 노드의 cost는 0
    - 소스 노드의 predecessor 노드는 소스 노드 자신으로

```
void Initialize(pGraph pG, int s)
{
        int i;

        for (i = 0; i < gGet_N(pG); i++)
        {
                gSet_in_S(?)    //node i is not in the set S
                gSet_cost(?)    //cost of node i = INF
                gSet_prenode(?) //predecessor of i = nothing
        }
        gSet_cost(?)    // cost of source = 0
        gSet_prenode(?) // predecessor of source = source
}
```

Dijkstra Algorithm

- **All_in_s**
  - 모든 노드의 최단 경로가 정해졌는지를 확인하는 함수
  - 모든 노드의 최단 경로가 정해졌다면 (forwarding table에 있다면) true
  - 한 노드라도 최단 경로가 정해져 있지 않다면 (forwarding table에 없다면) false

```
bool all_in_S(pGraph pG)
{
        int i;
        for (i = 0; i < gGet_N(pG); i++)
                if (?)  //when node i is not in S
                        return false;
        return true;
}
```

Dijkstra Algorithm

- **Extract_Min**
  - Forwarding table에 있는 노드 중 소스 노드와의 거리가 가장 짧은 노드 선택하는 함수
  - Forwarding table에서 가장 작은 cost를 가진 노드 결정 (i=1~N)

```
int Extract_Min(pGraph pG)
{
        int i, min;

        // find the first node not in S
        i = 0;
        while (?);      //check minimum value in S
        min = ?

        for (; i < gGet_N(pG); i++)
        {
                // pass for node in S
                if (?)
                        continue;

                // renew min when node with less cost is found
                if (?)
                        min = i;
        }
        return min;
}
```

Dijkstra Algorithm

- **Relaxation**
  - 노드 u로부터 노드 v까지의 최단 경로를 업데이트 하기 위한 함수
  - 노드 u의 cost, 노드 v의 cost, 노드 u와 노드 v 사이의 weight를 graph 함수를 통해 불러온다.
  - 노드 v의 cost가 노드 u의 cost와 노드 u와 노드 v 사이의 weight를 더한 값보다 크다면 노드 v의 cost는 노드 u의 cost와 노드 u와 노드 v사이의 weight를 더한 값으로 업데이트되고 노드 u의 predecessor 노드를 업데이트한다.

```
void Relaxation(pGraph pG, int u, int i)
{
        int cost_u, cost_v, weight_uv;
        double updated_weight;

        cost_u = ?;     // cost of u
        cost_v = ?;     // cost of v (v = sink of i-th edge of u)
        weight_uv = ?;  // weight of directed edge from u to v

        if (INF == cost_u)
                return;

        if (?)  //update cost from u to i
        {
                gSet_cost(pG, ?, ?);     //update cost as new cost
                gSet_prenode(pG, ?, ?); //Update predecessor of node u
        }
}
```

Bellman-ford Algorithm

- **Algorithm**
  - 노드들 간의 최단 경로 (cost)를 구하는 decentralized 알고리즘
  - 소스 노드와 다른 노드들 사이의 최단 경로의 거리를 cost라는 변수로 표현한다.
  - **Assumption**
    - 데이터는 (node 개수, edge 개수, edge 1, weight 1, … , edge k, weight k)로 구성
      - Edge : 노드와 연결되어 있는 link
      - Weight : 노드 간의 직접적인 거리
    - 두 노드 사이의 weight는 방향에 따라 다를 수 있다.
      - Ex. 노드 A에서 노드 B로 가는 거리는 4, 노드 B에서 노드 A로 가는 거리는 3으로 다를 수 있다.
    - 통신을 하던 중에 노드 사이의 link가 끊겨 weight가 100으로 바뀌는 상황 연출
      - 노드 C에서 다른 노드로 가는 link가 끊어졌다고 가정
      - Weight가 바뀌기 전의 forwarding table 참고하여 cost 변경
  - 실행
    - 최종적으로 소스 노드를 선택하고 그 소스 노드로부터 각 노드까지의 최단경로 구할 것
      - Weight가 바뀌기 전과 바뀐 후 둘 다 구할 것

Dijkstra Algorithm

- **Bellman_Ford_update**
  - 노드 C에서 다른 노드들까지의 weight가 바뀌었을 때 기존의 forwarding table을 참고하여 Bellman-Ford 알고리즘을 다시 실행하여 forwarding table을 업데이트 하는 함수
  - 노드 C에서 weight가 바뀜을 명시하고 이 때 노드 C에서 다른 노드까지의 cost를 초기화한다.
    - gChange_weight()를 이용하여 weight 변경
    - Cost 초기화할때 INF로 초기화할 것
  - 바뀐 cost와 weight를 참고하여 다시 Bellman-ford 알고리즘 실행하여 최단 경로 구하기

```
bool Bellman_Ford_update(pGraph pG, int s, int c)
{
        int l;
        int i, j, k;

        //weight from certain node is changed
        ?
        for (l = 0; l < ?; l++)
        {
                ?
        }
        ?

        for (i = 0; i < gGet_N(pG) -1; i++)
        {
                for (j = 0; j < gGet_N(pG); j++)
                        for (k = 0; k < gGet_deg(pG, j); k++)
                                Relaxation(pG, j, k);
                gRenew_cost(pG);
        }

        for (j=0; j < gGet_N(pG); j++)
                for (k=0; k < gGet_deg(pG, j); k++)
                        if (Exist_neg_Cycle(pG, j, k))
                                return false;

        return true;
}
```

Dijkstra Algorithm

■ Initialize

– Dijkstra 알고리즘과 마찬가지로 전체 알고리즘을 실행하기 전
모든 노드의 weight와 cost들을 초기화한다. (Forwarding table 초기화)

- 소스 노드를 제외한 모든 노드들의 cost는 INF
- 소스 노드를 제외한 모든 노드들의 predecessor 노드는 아무것도 없는 형태로
- 소스 노드의 cost는 0
- 소스 노드의 predecessor 노드는 소스 노드 자신으로

```
void Initialize(pGraph pG, int s)
{
        int i;

        for (i = 0; i < gGet_N(pG); i++)
        {
                gSet_newcost(?);        //cost of node i = INF
                gSet_prenode(?);        //predecessor of node i = nothing
        }
        gSet_newcost(?);        //cost of source = 0
        gSet_prenode(?);        //predecessor of source = source
        gRenew_cost(?); //update all costs of graph
}
```

Dijkstra Algorithm

- **Relaxation**
  - 노드 u로부터 노드 v까지의 최단 경로를 업데이트 하기 위한 함수
  - 노드 u의 cost, 노드 v의 cost, 노드 u와 노드 v 사이의 weight를 graph 함수를 통해 불러온다.
  - 노드 v의 cost가 노드 u의 cost와 노드 u와 노드 v 사이의 weight를 더한 값보다 크다면 노드 v의 cost는 노드 u의 cost와 노드 u와 노드 v사이의 weight를 더한 값으로 업데이트되고 노드 u의 predecessor 노드를 업데이트한다.

```
void Relaxation(pGraph pG, int u, int i)
{
        int cost_u, cost_v, weight_uv;

        cost_u = ?      // cost of u
        cost_v = ?;     // cost of v (v = sink of i-th edge of u)
        weight_uv = ?;  // weight of directed edge from u to v

        if (INF == cost_u)
                return;

        if (?)
        //update the cost from u to i
        {
                gSet_newcost(pG, ?, ?);
                gSet_prenode(pG, ?, ?);
        }
}
```

Dijkstra Algorithm

- **Exist_neg_Cycle**
  - 노드 u로부터 노드 v까지의 최단 경로가 정상적으로 업데이트되었는지 확인하는 함수
  - Cost를 비교하는 과정은 Relaxation 함수와 매우 유사
    - 노드 u로부터 노드 v까지의 최단 경로를 업데이트 하기 위한 함수
    - 노드 u의 cost, 노드 v의 cost, 노드 u와 노드 v 사이의 weight를 graph 함수를 통해 불러온다.
    - 노드 v의 cost가 노드 u의 cost와 노드 u와 노드 v 사이의 weight를 더한 값보다 크다면 true를 반환, 아니라면 false를 반환

```
bool Exist_neg_Cycle(pGraph pG, int u, int i)
{
        int cost_u, cost_v, weight_uv;

        cost_u = ?;     //cost of u
        cost_v = ?;     //cost of v (v = sink of i-th edge of u)
        weight_uv = ?;  //weight of directed edge from u to v

        if (INF == cost_u)
                return false;

        if (?)  //cost has correct values (hint. compare costs)
                return true;

        return false;
}
```

Simulation

■ 컴파일 및 실행

- 컴파일
  - Dijkstra : gcc -o main main.c graph.c dijkstra.c
  - Bellman : gcc -o main main.c graph.c bellman.c
- 실행
  - Example : ./main data1

```
jongmin@jongmin-VirtualBox:/media/sf_share/project_teach/project2/Answer/Dijkstr
a$ gcc -o main main.c graph.c dijkstra.c
jongmin@jongmin-VirtualBox:/media/sf_share/project_teach/project2/Answer/Dijkstr
a$ ./main data1
```

Dijkstra

```
jongmin@jongmin-VirtualBox:/media/sf_share/project_teach/project2/Answer/Bellman
$ gcc -o main main.c graph.c bellman.c
jongmin@jongmin-VirtualBox:/media/sf_share/project_teach/project2/Answer/Bellman
$ ./main data1
```

Bellman

Simulation

■ 실행 결과 예시

```
A file with graph data has been open.

   A graph has been created with 5 nodes.
   An edge has been created (A -> B, weight: 10)
   An edge has been created (A -> C, weight: 3)
   An edge has been created (B -> C, weight: 1)
   An edge has been created (B -> D, weight: 2)
   An edge has been created (C -> B, weight: 4)
   An edge has been created (C -> D, weight: 8)
   An edge has been created (C -> E, weight: 2)
   An edge has been created (D -> E, weight: 7)
   An edge has been created (E -> D, weight: 9)

   => 5 nodes & 9 edges have been created.

        input source (A, B or ...): A

        A to B,    cost:  7    path: B <- C <- A
        A to C,    cost:  3    path: C <- A
        A to D,    cost:  9    path: D <- B <- C <- A
        A to E,    cost:  5    path: E <- C <- A
jongmin@jongmin-VirtualBox:/media/sf_share/project_teach/project2/Answer/Dijkstra$
```

Dijkstra

```
A file with graph data has been open.

   A graph has been created with 5 nodes.
   An edge has been created (A -> B, weight: 10)
   An edge has been created (A -> C, weight: 3)
   An edge has been created (B -> C, weight: 1)
   An edge has been created (B -> D, weight: 2)
   An edge has been created (C -> B, weight: 4)
   An edge has been created (C -> D, weight: 8)
   An edge has been created (C -> E, weight: 2)
   An edge has been created (D -> E, weight: 7)
   An edge has been created (E -> D, weight: 9)

   => 5 nodes & 9 edges have been created.

        input source (A, B or ...): A

        A to B,    cost:  7    path: B <- C <- A
        A to C,    cost:  3    path: C <- A
        A to D,    cost:  9    path: D <- B <- C <- A
        A to E,    cost:  5    path: E <- C <- A

        A to B,    cost: 10    path: B <- A
        A to C,    cost:  3    path: C <- A
        A to D,    cost: 12    path: D <- B <- A
```

Bellman