**<Data Structure>**

Assignment #3

|  |  |
| --- | --- |
| Professor | 이혁준 교수님 |
| Department | Computer Engineering |
| Student ID | 2014722030 |
| Name | 박현아 |
| Date | 2015. 12. 04 |

1. **Program Overview**
2. 배경 설정

우리가 평소 많이 사용하는 인터넷에서 IP 패킷이 서로 다른 두 호스트 사이로 흐르기 위해 즉, 하나의 호스트에서 다른 호스트로 전달되기 위한 최적의 경로를 결정할 수 있도록 해주는 Algorithm을 Link State Routing Algorithm이라고 한다. 최적의 경로를 구하기 위해 router들간의 연결되어 있는 구조와 더불어 router와 인접한 router 사이의 cost, 최대 해당 path를 지날 수 있는 한계치인 bandwidth에 대한 정보가 필요하다. 따라서 이러한 것들을 포함하고 있는 graph를 구성하여 Dijkstra Algorithm을 통해 시작점과 도착점까지의 최적 경로를 구한 후 사용자에게 할당해야 한다.

1. Dijkstra Algorithm

Dijkstra Algorithm은 서로 다른 두 정점 간의 최단 경로를 구하기 위한 Algorithm으로 시작점(s)과 도착점(v)들 사이의 최단 거리를 저장하고 있는 int형 1차원 배열 dist[v]와 이미 최단 거리가 구해진 정점은 또 다시 구할 필요가 없기 때문에 이를 나타내는 bool형 visit[v] 배열이 필요하다.

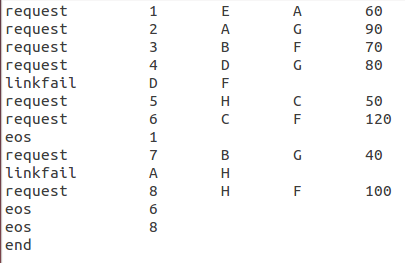
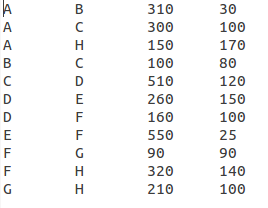
그러고 난 후 시작점(s)에서부터 연결되어 있는 경로들에 대한 cost를 dist에 할당해 준 후 이 중에서 가장 작은 값(해당 정점을 A라고 할 때)을 최단 경로로 반환하고 이에 대해 visit[A]를 true로 바꿔 준다. 또한 시작점(s)에서 A까지 가는 최단 경로가 존재하고 A에서 B까지 가는 최단 경로가 존재한다고 할 때 만약 (s-A) + (A-B)가 시작점(s)에서 B까지 가는 경로보다 짧다면 최단경로는 전자로 재 할당된다. 이를 간선 경감(edge relaxation)이라고 하고 시작점(s)에서 모든 다른 정점까지의 최단경로가 구해지면 Algorithm을 끝낸다.

1. Operation

|  |  |
| --- | --- |
| **Operation** | **Description** |
| REQUEST | Request는 service ID, 근원지(Source)/목적지(Destination) Router ID, Bandwidth를 포함하고 있는 operation으로, graph에 그려져 있는 Router의 cost에 따라 최단 경로를 할당 받고 Bandwidth를 할당한다. 만약 Bandwidth가 부족하거나 최단 경로가 존재하지 않을 경우 해당 request를 보류 Queue에 넣어놓는다. |
| LINKFAIL | Linkfail의 경우에 인접한 두 Router에 대한 정보도 함께 들어오는데 이 것은 두 Router들 사이의 Cost를 무한대로 바꿔주라는 뜻이다. 따라서 만약 해당 path를 사용하는 request가 있었다면 대역폭을 반환한 후 최단 경로를 다시 구해 재할당 해야 한다. 만약 재할당이 불가능 하다면 해당 request는 보류 목록 Queue에 들어간다. |
| EOS | EOS는 End Of Service의 줄임말로 Request의 service ID와 함께 입력되는데 해당 service ID에 해당하는 request를 찾아 만약 Bandwidth를 할당 받고 정보를 송수신하고 있다면 해당 Bandwidth를 반환한 후 request를 삭제한다. 만약 해당 request가 보류 목록 Queue에 존재할 경우 보류 목록에서 삭제시켜 준다. |

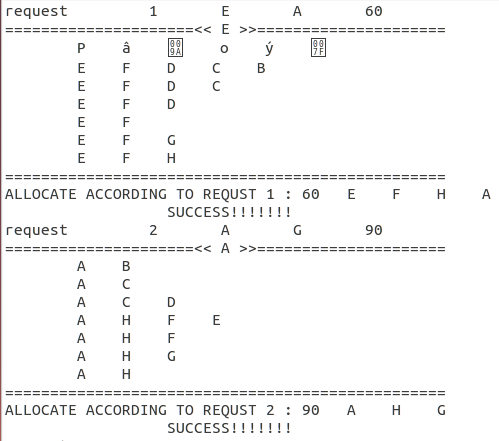
1. Sample input/output
2. Sample input

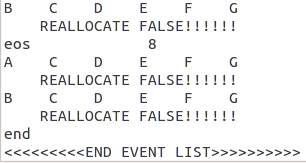
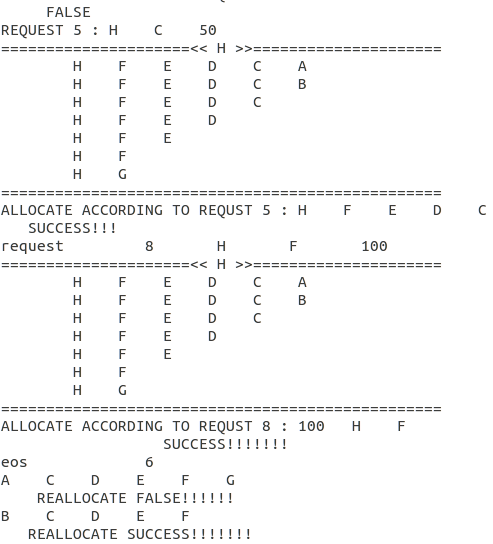
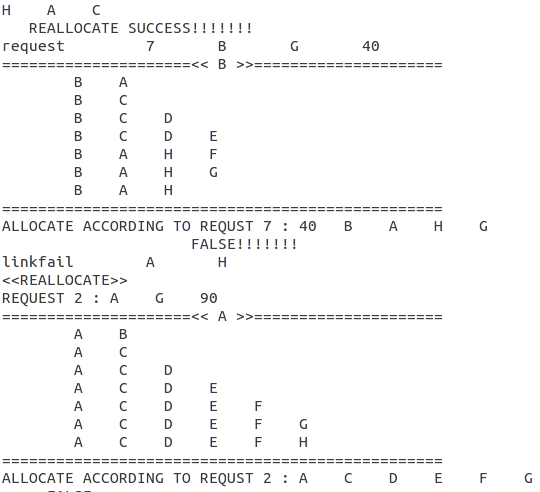
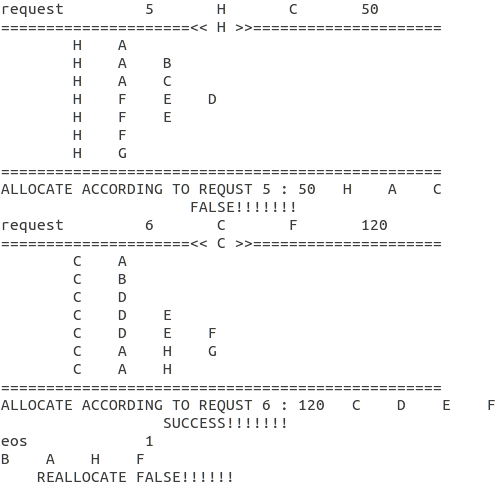
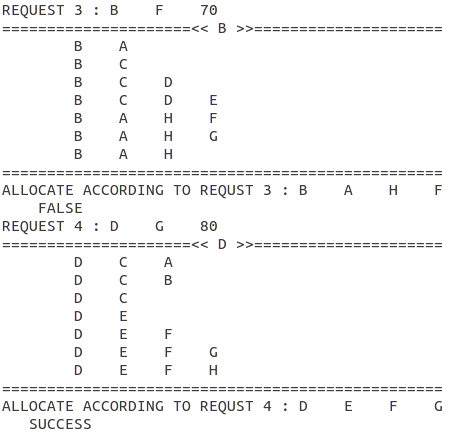
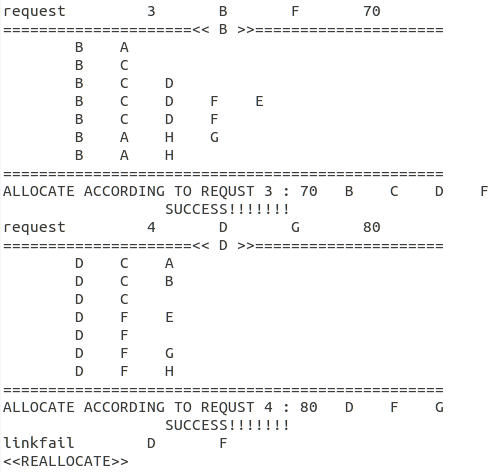
* router.in - event.in



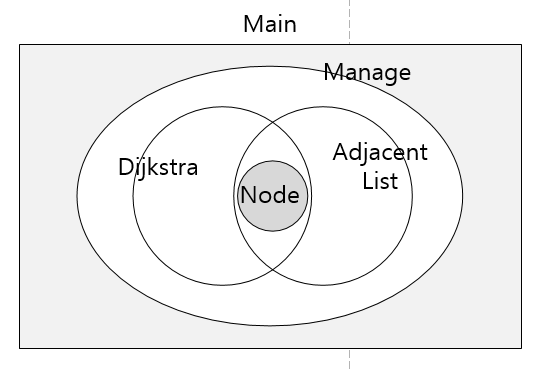
1. Sample output

* result.out





1. **Overall system class design**



* 가장 크게 Main이 존재하고 그 안에 다른 클래스들을 다루는 Manage가 가장 크게 자리잡고 Dijkstra와 Adjacent List가 또 하나의 큰 부분을 Manage 내에서 차지하고 있다. Node의 경우에 프로그램 내에서 사용되지 않는 곳이 없기 때문에 위와 같이 표현된다.

1. **Algorithm**
2. Flowchart
3. Manage Class

|  |  |  |
| --- | --- | --- |
| **Function** | **Flowchart** | **Description** |
| Realloc |  | Realloc은 LINKFAIL이 발생했을 때 인접 리스트를 수정해주고, 해당 path를 사용하는 request들에 대해 최단 경로를 재구성하고 대역폭을 할당한다. |
| DoMain |  | DoMain 함수는 main과 같은 역할을 하는 함수라는 뜻으로, router.in 파일을 통해 information을 통해 인접리스트를 구성하고 이 인접 리스트를 바탕으로 event.in을 통해 들어오는 정보에 따라 최단 경로를 구하기도 하고 인접리스트에 대한 정보를 수정하기도 한다. |

1. Dijkstra Class

|  |  |  |
| --- | --- | --- |
| **Function** | **Flowchart** | **Description** |
| ShortestPath |  | ShortestPath 함수는 인접리스트를 이용하여 특정 시작 정점에서 다른 모든 정점에 대한 최단 경로를 구하기 위한 함수이다.  마지막 네 번째에 설명 돼 있는 것을 부가적으로 덧붙여 보자면 시작 점 s 에서 w까지 가고자 하는 cost가 10일 때 s->u->w의 cost의 합산이 7인 경우에 다음과 같이 최단 경로의 변경이 발생한다. |
| Search |  | Search는 시작 정점과 도착 정점에 대한 index 값만을 매개변수로 받아 이를 통해 인접리스트 내에서 해당 Node들을 찾는다.  우선 시작 정점의 index를 통해 Node를 찾는데 만약 Node가 존재하지 않는다면 NULL을 반환한다.  만약 시작 정점의 Node를 찾았다면 다음 순서로 도착 정점의 Node를 찾고서 해당 Node를 반환한다. |
| Select |  | Select 함수는 dist와 visit 배열을 사용하여 두 배열에 같은 index값을 할당한 후 visit의 값이 false이고 또 dist의 값이 기존의 비교해오던 최소 길이보다 작을 경우 비교해주던 최소 길이를 dist의 값으로 바꿔주고 이러한 과정을 배열이 끝날 때까지 반복해 준다. |

1. AdjacentList Class

|  |  |  |
| --- | --- | --- |
| **Function** | **Flowchart** | **Description** |
| Insert |  | 인접 리스트를 만드는 함수 Insert의 매개변수로는 시작 vertex, 도착 vertex, 두 router 간의 bandwidth와 cost가 들어온다.  우선 인접 리스트 내에 시작 vertex와 같은 값을 가지는 start Node가 존재하는지 확인한다. 만약 존재한다면 도착 vertex에 대한 Node만을 동적 할당해 준 후 오른쪽 Node들 사이에 vertex 크기에 맞춰 insert해준다. 만약 같은 시작 vertex가 존재하지 않는다면 시작 vertex와 도착 vertex에 대한 Node를 모두 생성해 준 후 시작 vertex의 값에 따라 down들 사이에 insert해 준다.  Bandwidth와 cost 값은 도착 vertex의 Node에 저장된다. |

1. Performance Evaluation
2. Manage class

* DoMain

request, linkfail, eos와 같이 다양한 연산이 존재하고 여기서 가장 많은 loop을 사용한 operation은 3개를 사용한 linkfail이기 때문에 linkfail에 의해 O(n) = n^3으로 나타내어 질 수 있다.

* Realloc

Realloc 함수의 경우 Dijkstra class의 Shortest Path를 사용하고 그 외에 함수 내에서 다른 loop문을 사용하고 있지 않기 때문에 O(n) = n^2 – n이 된다.

1. AdjacentList class

* Insert

while문 안에 while문을 사용하고 있고 그 외에 코드의 길이가 길긴 하지만 상수 적으로 변하는 값들이기 때문에 O(n) = n^2이다.

1. Dijkstra class

* ShortestPath

시작 점에 대해 dist에 값을 대입하는 for문에 대해서 length만큼 looping을 하기 때문에 length를 n이라고 할 때, 해당 for문에서 O(n) = n이다. 또한 전체적인 최단 길이를 구하는 for문에서 O(n) = n(n-2) = n^2 – 2n 이고 두 값을 더하면 O(n) = n^2 – n이 된다.

* Search

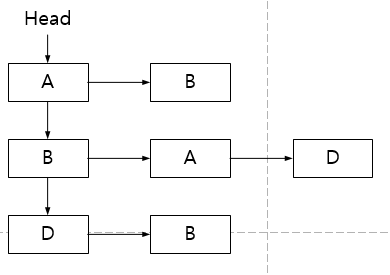
while문 안에 또 while문이 있기 때문에 O(n) = n^2라고 볼 수 있다.

* Select

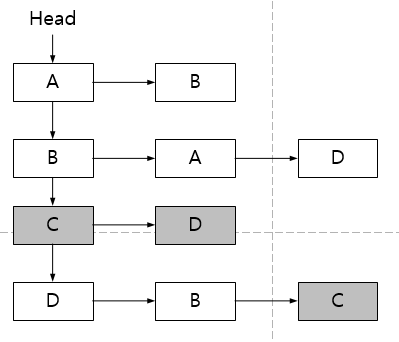
for문 하나로 구성되어 있기 때문에 O(n) = n+k(k는 임의의 상수)가 될 수 있는데 있는 O(n) = n으로 다시 나타낼 수 있다.

1. **Data Structure Diagram**
2. Insert of Adjacent List
3. 동일한 Start vertex Node가 존재하지 않을 때

* 기존의 인접 리스트

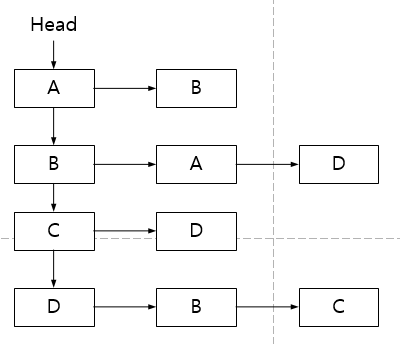


* CD 경로 insert

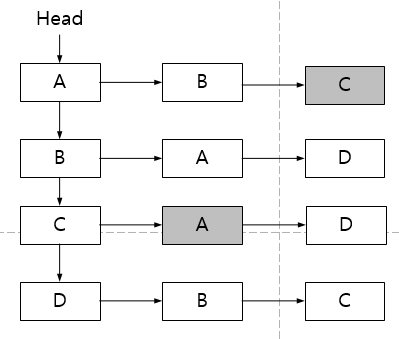


1. 동일한 Start vertex Node가 존재한 경우

* 기존의 인접 리스트



* CA 경로 insert



1. **Discussion and Conclusion**

처음 Dijkstra algorithm에 대한 설명을 들었을 때 경로상 앞으로 나아갔다가 다시 전 단계 정점으로 돌아오는 양상을 보이길래 Stack을 쓰면 되겠다는 생각을 가졌다. 그래서 Stack을 이용하여 최단 경로를 구하는 코드를 구현했었는데 생각했던 것만큼 쉽게 구현되지 않고 또 코드가 복잡해졌다. 또 지나온 정점을 다시 되돌아 가도록 code를 design하니 이미 최단 경로를 구한 정점에 다시 돌아가는 일도 발생했다. Stack을 사용하여 구현하는 것은 불가능할 것 같다는 생각에 다른 방법을 생각해보기로 했다. 그래서 실습 시간 중에 조교님이 알려 주신 예제코드를 visual studio에 옮겨 계속해서 디버깅을 돌려보며 탐구하였더니 Dijkstra algorithm에 대한 이해와 코드를 design하는 데에 많은 도움이 되었다.

이번 과제를 하면서 인간의 사고를 프로그래밍 하여 로봇을 조종하거나 어떠한 일을 수행하는 프로그램을 만드는 일이 얼마나 힘들고 경이로운 일인지 알게 되었다. Dijkstra algorithm이라는 것이 최단 경로를 구하기 위한 algorithm인데, 어떠한 경로 graph가 주어졌을 때 인간은 직관적으로 최단 경로를 빠르게 찾아낼 수 잇지만 컴퓨터는 시작점에서 모든 다른 정점들까지의 경로에 대한 값을 비교해야만 최단 경로를 알아낼 수 있다. 단순 (어려운 것일 수도 있지만) 최단 경로를 구하는 것에 있어서도 코드 구현이 복잡해지고 어려워 지는데, 예를 들어 즉각적인 현재 날씨, 도로 교통 상황 등 여러 정보를 통해 길을 안내하는 차량 navigation의 경우에 프로그램이 얼마나 복잡하게 구현되어 있을지 상상도 되지 않는다.

비록 데이터 구조 수업의 마지막 과제에서 요구하는 조건을 모두 만족시키는 프로그램을 구현하지는 못했지만 이것이 끝이라고 생각하지 않고 과제에는 나오지 않았지만 수업시간 중에 배운 다양한 Algorithm들을 실제로 구현해보며 실력향상에 힘쓰고자 한다.

마지막으로 과제를 구현하면서 알 수 없는 error를 발생시킨 경우가 있었는데, pCur이라는 Node가 존재하고 pCur->get\_pRight() 또한 존재하는데 pCur->get\_pRight()->get\_vertex()를 하게 되면 디버깅 창에 pCur의 right Node가 NULL이라고 나타나면서 vertex를 가져오는 것이 불가능 했다. 분명 pCur의 right에 주소가 존재함을 확인했음에도 불구하고 이러한 일이 발생하는 원인을 알 수 없어 다른 Node를 선언하여 이 Node에 pCur의 right Node의 주소 값을 할당해 준 후 진행하였더니 error가 발생하지 않았다.

또 다른 error는 Manage에 선언되어 있는 ofstream output을 Dijkstra class 내에서도 사용해 주기 위해 Dijkstra 내에 ofstream out을 선언해 주고 public 함수를 통해 해당 private 변수에 output의 주소를 할당해 주고자 했는데 디버깅을 돌려보니 액세스 위반이 발생했다. 그래서 위 변수들을 동적 할당을 시도해 보았지만 여전히 error를 발생시켰다. 이 문제를 어떻게 할지 고민하다가 코드 구현의 방식을 달리해 보기로 했다. 2차 과제를 구현할 당시에 text file이 닫히더라도 text file에 쓰여 있는 내용 후부터 누적되어 쓰도록 하는 buffer를 걸어주었던 것을 생각해 내어 output file에 내용을 쓰게 될 경우마다 해당 텍스트 파일을 열었다 닫았다 해주는 방식을 택했다.