알고리즘

- Assignment #2 Implementation of Floyd's Algorithm-

과목명	알고리즘
교수명	정민영 교수님
학　과	소프트웨어 학과
학　번	20170266
이　름	김승욱
제출일	2021-11-04

1. 개요

Floyd's Algorithm을 이용해 각 정점에서 다른 모든 정점까지의 최단 거리와 경로를 구하는 프로그램을 구현하고, 알고리즘의 시간 및 공간 복잡도를 이해해 동적 프로그래밍 전략을 사용해 분할 정복 전략에서 시간 복잡도를 단축시킬 수 있는 방법을 이해한다.

2. 구현 언어 및 방법

C Language를 이용해 Floyd's Algorithm을 구현한다. 동적 프로그래밍 전략을 사용하기 위해 Floyd 알고리즘의 출력값을 저장할 W matrix(인접 행렬), D matrix(최단거리 행렬), P matrix(경로 행렬)을 전역변수로 생성한다. 또한 최단거리의 경로를 출력하기 위해 tmp matrix와 (int)tmpIdx를 전역변수로 생성했다.

floyd 함수는 W matrix를 통해 D matrix(최단거리 행렬)과 P matirx(경로 행렬)을 계산한다. 분할 정복 전략과 동적프로그래밍 전략을 이용해 구현한다.

D matirx를 구하기 위해 D^{k-1}과 D^{k}의 재귀 관계식을 정립하면 두가지 경우로 나눌 수 있는데, 첫 번째 경우는 $\{v_1, v_2, ..., v_k\}$ 에 속한 마디만 중간마디로 거쳐서 최소한 하나는 v_k를 거치지 않는 경우이다. 이 경우 $D^{k}[i][j] = D^{(k-1)}[i][j]$ 의 재귀 관계식을 갖는다.

두번째 경우는 $\{v_1, v_2, ..., v_k\}$에 속한 마디만 중간마디로 거쳐서 모두 v_k를 거치는 경우이다. 이 경우 $D^{k}[i][j] = D^{(k-1)}[i][k] + D^{(k-1)}[k][j]$의 재귀 관계식을 갖는다. 따라서 첫번째 경우나 두번째 경우중 최소값이 $D^{k}[i][j]$의 값임을 알 수 있다. 두 경우를 모두 고려한 재귀 관계식은 $D^{k}[i][j] = \min(D^{(k-1)}[i][j], D^{(k-1)}[i][k] + D^{(k-1)}[k][j])$이다.

P matrix의 경우 D matirx를 구하는 과정에서 필요한 k를 할당하는 것으로 구할 수 있다. 이 과정을 통해 floyd 함수를 고려하면 3중 for문을 통해 D matrix와 P matrix를 구할 수 있다.

path함수와 pathPrint 함수는 D matrix에 저장된 최단 거리로 가는 경로를 구하고 출력하는 함수이다. path함수는 재귀적으로 구현하며 tmp matrix에 최단거리로 가는 경로의 정점을 저장한다. pathPrint 함수에서는 path 함수에서 구한 tmp matrix에 저장된 정점의 갯수와, 각 정점을 출력하는 함수이다.

main 함수에서는 인접행렬 W를 입력받아 명령인자 'array'일 경우 num, D matrix, P matrix를 출력하도록 설정했고, 명령인자 'path'일 경우 최단거리의 정점의 갯수와 각 경로를 출력하도록 설정했다.

3. 실험 결과 및 분석

floyd Algorithm을 동적 프로그래밍 전략을 사용하지 않는다면 모든 정점에 대한 경우의 수를 고려해야 하기 때문에 시간복잡도 O(n!)를 갖는다.

동적 프로그래밍 전략을 사용해 구현한 floyd Algorithm의 시간 복잡도를 계산해보자. 입력크기를 n(그래프에서 마디의 개수)로 정하고 단위 연산을 for문 안의 명령문(instruction)으로 정할 경우 3중 for문이기 때문에 이 알고리즘은 $O(n^3)$의 시간 복잡도를 갖음을 알 수 있다. 추가적으로 path 함수를 포함한 pathPrint 함수는 O(n)의 시간복잡도를 갖는다.

4. 결론

분할 정복 전략은 재귀 호출을 사용한다. 재귀 호출을 사용하는 알고리즘 중 일부는 이미 계산한 이전 항을 다시 계산하는 경우가 있기 때문에 이 경우에 시간 복잡도를 향상시키기 위해 동적 프로그래밍 전략을 사용해 중복된 계산과정을 배열과 같은 저장 공간에 미리 저장해 둠으로써 시간 복잡도를 향상 시킬 수 있다. 이번 Floyd's Algorithm도 동적 프로그래밍 전략을 사용하지 않았더라면 모든 경우의수에 계산을 하기 때문에 O(n!)의 시간 복잡도를 갖지만, 중복된 계산을 최소화 시킴으로써 $O(n^3)$의 시간 복잡도까지 시간 복잡도를 최적화할 수 있었다.