과목: 알고리즘

교수: 정민영 교수님

Algorithm [Assignment #2]

- Implementation of Floyd’s Algorithms -

홍지훈

이름: 홍지훈

학과: 소프트웨어학부

분반: 나

학번 : 20201777

**과제: Floyd’s 알고리즘을 구현하고, 각 정점에서 시작해서 정점으로 가는 경로 및 거리를 출력**

**주제 및 목표: 가중치 포함 그래프의 각 정점에서 다른 모든 정점까지의 최단거리와 경로를 구하는 Floyd’s 알고리즘을 직접 구현하고 평가함으로써 시간 및 공간 복잡도를 이해함**

Floyd’s 알고리즘은 그래프에서 최단 거리와 경로를 구할 수 있는 알고리즘이다.

먼저 인접 행렬로 구현된 그래프를 가지고, 경로의 값들을 비교해가며 가장 적은 가중치를 가지는 경로를 찾는다. 이때 단위연산은 중간경로를 거치는 과정과 현재 계산된 값 두 가지를 비교하는 과정이 되고, 이때 3중 반복문을 사용하기 때문에, 시간복잡도는 O(n^3)이 된다.

**구현**

**기본적으로 구현은 C++을 사용하였고, 예시코드와 강의자료의 코드를 참고하였다.**

Int N: 정점의 개수

Int W[][]: 정점의 가중치를 담는 2차원 배열

Int D[][]: 최소 거리를 담는 2차원 배열

Int P[][]: 정점까지 가는 중간경로(중 가장 큰 것)를 담는 2차원 배열

**Floyd’s 알고리즘 구현**

모든 P 배열의 요소들을 0(중간경로 없음)으로 만들어주고, D 배열의 요소들을 W의 요소들로 만들어주었다.

Floyd’s 알고리즘은 간단한 3중 반복문으로 구현하였다. 모든 반복문은 1부터 N까지 돌며, 각각 I, j, k를 가지고 있다. k의 값을 중간정점으로 두는 i to j 거리와 현재 계산된 i to j 거리를 비교하여 더 짧은 쪽을 D 배열에 넣어준다.

**이때, 정점이 없는 배열과 두 정점이 같은 경우 INF와 0으로 나누는 게 아닌 모두 0으로 처리 해야 하기 때문에** D[i][k]와 D[k][j]가 0이 아닐 경우에만 비교하고, i와 j가 같지 않을 때만 처리해주도록 만들었다.

이때 최소거리 D[i][j]가 D[i][k] + D[k][j]보다 크면, 중간경로 P[i][j]에 k를 넣어주고, D[i][j]를 최소거리로 바꾸어준다.

1. for(int k = 1; k <= N; k++) {
2. for(int i = 1; i <= N; i++) {
3. for(int j = 1; j <= N; j++) {
4. if((D[i][k] != 0 && D[k][j] != 0)
5. && ((D[i][k] + D[k][j] < D[i][j])
6. || (D[i][j] == 0 && i != j))) {
7. P[i][j] = k;
8. D[i][j] = D[i][k] + D[k][j];
9. }
10. }
11. }
12. }

**Path 경로와 경로의 개수를 출력**

Path경로의 경우는 재귀함수를 사용하였다. 구하고싶은 경로의 정점들(q, r)을 인수로 받아서, 중간경로 P[q][r]이 존재하면(0이 아니면), (q, P[q][r])을 인수로 받아서 계속 진행을 한 후, P[q][r]을 출력, 그후 (P[q][r], r)을 인수로 다시 함수를 호출하여 진행을 한다.

양식상으로는 경로를 출력해주기 전에 경로의 개수를 출력해주어야 하는데, 마땅한 방법이 생각나지않아, 경로를 구하는 방식과 거의 똑같은 방식으로 개수만 카운트하여 코드를 짜 주었다.

1. void GetPath(int q, int r) {
2. if(P[q][r] != 0) {
3. GetPath(q, P[q][r]);
4. cout << " " << P[q][r];
5. GetPath(P[q][r], r);
6. }
7. }
8. int GetPathCount(int q, int r) {
9. if(P[q][r] == 0)
10. return 0;
11. return 1 + GetPathCount(q, P[q][r]) + GetPathCount(P[q][r], r);
13. }

**실험 결과 및 분석**

Array 결과

1. 10
2. 0 6 7 4 4 8 6 4 7 7
3. 4 0 3 6 3 3 6 6 1 7
4. 5 8 0 7 5 6 3 4 4 4
5. 5 4 3 0 6 6 3 5 4 4
6. 6 4 3 3 0 4 4 3 5 5
7. 8 4 7 10 7 0 10 10 5 7
8. 2 5 4 5 3 3 0 2 1 1
9. 5 5 4 3 1 5 5 0 3 6
10. 8 4 7 7 4 2 8 7 0 6
11. 8 5 5 5 2 3 6 5 1 0
12. 0 0 0 0 0 5 0 0 2 0
13. 0 0 0 5 0 9 3 0 0 7
14. 7 0 0 0 0 9 0 0 7 7
15. 7 0 0 0 8 9 0 7 7 7
16. 7 0 0 0 0 0 0 0 0 7
17. 0 0 2 5 2 0 0 2 2 0
18. 0 9 0 8 8 9 0 0 0 0
19. 0 5 5 0 0 5 5 0 0 7
20. 2 0 2 5 0 0 5 0 0 0
21. 7 9 5 5 0 9 0 5 0 0

첫 줄에 정점의 개수 N이 출력되고, 그다음 D, P matrix가 순서대로 출력된다.

Path 결과 (길어서 생략)

정점 0에서 0까지 가는 경로부터,

순서대로 정점 N에서 N까지 가는 경로까지 한 줄씩 출력된다.

각 i줄마다 첫 번째 인수는 중간경로의 개수, 그 뒤로 중간경로를 순서대로 출력해준다.

수행 시간을 측정해 보았지만, N의 개수가 10이라 수행 시간은 매우 적게 나왔다. 그러나 3중 for문을 사용하기 때문에, N이 커지면 커질수록, 수행 시간도 많이 늘어날 것으로 예상된다.

이때 시간복잡도는 O(n^3)이 된다.

**결론**

결론적으로 Floyd’s 알고리즘은 그래프의 모든 정점으로의 최단 거리를 구하는 알고리즘이다. 3중 반복문을 사용하기 때문에 정점의 개수 N이 커지면 커질수록 알고리즘의 수행 시간도 높은 폭으로 올라간다. 또한 Path를 찾는 과정에서도 N^2만큼 재귀 함수가 돌아가기 때문에 사용에 주의를 해야 할 것 같다.