알고리즘 실습(10주차)

201000287 일어일문학과 유다훈

1. 과제 설명 및 해결 방법

1. All Pairs Shortest Path Algorithm 구현

All Pairs Shortest Path는 모든 노드로부터 다른 모든 노드로의 최단 거리를 의미한다. 이번 과제에서는 APSP의 세 가지 알고리즘을 구현한다.

① Slow and Fast All Pairs Shortest Paths Algorithms

- 입력 받은 Adjacent matrix를 이용하여 APSP를 구현한다.
- APSP를 구현할 때 Matrix Multiplication을 이용한다.
- Slow 방식은 V^4의 시간이 걸린다.
- Fast 방식은 똑같은 알고리즘 구현이나 시간은 V^3logn의 시간이 걸린다.

② Floyd and Warshall Algorithms

- 입력 받은 Adjacent matrix를 이용하여 APSP를 구현한다.
- 음수가중치의 사이클이 없다고 가정하고 APSP를 구하는 알고리즘이다.
- 알고리즘의 시간복잡도는 V^3만큼의 시간이 걸린다.
- 부모 노드를 따로 저장하여 해당 노드로 가는데 어떠한 경로를 거치는지 파악한다.

2. 주요 부분 코드 설명(알고리즘 부분)

1. Extend Shortest Paths

```
int ** extend_shortest_paths(int ** l, int ** matrix, int n, int maxValue) {

    int ** newl; //새롭게 데이터 값을 저장할 2차원 배열
     newl = (int **)malloc(sizeof(int*) * n);
    for(int i=0; i<n; i++) {
            newl[i] = (int *)malloc(sizeof(int*) * n);
    }
    /* 2차원배열 동적배열 끝 */

    /* Extend Shortest Paths Algorithms */
    for(int i=0; i<n ; i++) {
        for(int j=0; j<n; j++) {
            newl[i][j] = maxValue;
            for(int k=0; k<n; k++) {
                if( newl[i][j] > l[i][k] + matrix[k][j]) {
                    newl[i][j] = l[i][k] + matrix[k][j];

                }
            }
        }
    }
    /* Extend Shortest Paths Algorithms */

    //갱신 상태 출력
    for(int i=0; i<n ; i++) {
        for(int j=0; j<n; j++) {
            printf("%d ", newl[i][j]);
        }
        printf("\n");
    }

    return newl; //Return data value
}
```

- Matrix Multiplication 방식의 핵심 알고리즘 부분
- 배열 l과 matrix는 초기에는 같은 값을 가지고 있다.
- 비교값을 저장할 새로운 배열 선언하고 비교를 시작한다.
- newl[i][j]이 l[i][k] + matrix[k][j]의 값보다 크다면, 해당 위치의 값을 l[i][k] + matrix[k][j]로 바꾼다. 행렬곱연산을 한다.
- 해당 연산이 끝나면 비교값이 저장되어 있는 배열을 리턴한다.

2. Slow All Pairs Shortest Path

```
void slow_all_pairs_shortest_paths(int **matrix, int n, int maxValue) {
    int ** l; //데이터 값을 저장할 배열 L

    l = (int **)malloc(sizeof(int*) * n);
    for (int i = 0; i < n; i++) {
        l[i] = (int *)malloc(sizeof(int*) * n);
     }
    /* 2차원 배열 동적배열 종료 */

    /* 데이터값 옮기기 */
    for(int i=0; i<n; i++) {
        for(int j=0; j<n; j++) {
            l[i][j] = matrix[i][j];
        }
    }

    /* --- */

    for(int m = 1; m<n-1 ; m++) {
        printf("%d 번째 가중치 갱신\n", m);
       l = extend_shortest_paths(l, matrix, n, maxValue);
    }
}
```

- 입력 받은 2차원 배열의 값을 기준으로 APSP를 만드는 Slow All Pairs Shortest Path를 구현한다.
- Extend Shortest path 알고리즘을 n-2번 실행해준다.

3. Fast All Pairs Shortest Path

```
void fast_all_pairs_shortest_paths(int **matrix, int n, int maxValue) {
    int m = 0;

    /* 데이터를 담을 2차원 배열의 동적할당 */
    int ** l;
    l = (int **)malloc(sizeof(int*) * n);
    for (int i = 0; i < n; i++) {
        l[i] = (int *)malloc(sizeof(int*) * n);
    }
    /* 데이터를 담을 2차원 배열의 동적할당 끝 */

    /* 데이터 옮겨담기 */
    for(int i=0; i<n; i++) {
        for(int j=0; j<n; j++) {
            l[i][j] = matrix[i][j];
        }
    }
    /* 데이터 옮겨담기 끝*/

    int count=1;

    /* m*2 씩 반복횟수를 상승시키며 Extend Shortest Path 실행 */
    while( m < n-1) {
         printf("%d 번째 가중치 갱신\n", count++);
        l = extend_shortest_paths(l, l, n, maxValue);
        if(m == 0) {
            m = m+1;
        }
        m = 2*m;
    }
}
```

- 입력받은 2차원 배열로 APSP를 만드는 함수.
- Slow APSP와 같지만, 반복문에서 m*2씩 값이 상승하며 반복한다.
- 이 방법은 APSP값이 들어있는 배열을 두 개씩 나누어 재귀적으로 계산하는 방법으로 계산속도는 V^3logn의 속도가 나온다.

4. Floyd and Warshall Algorithms

```
12  void floyd_Warshall (int **matrix, int n, int maxValue) {
13      int ** d= matrix; //매트릭스값 복사
14      int parent[n][n]; // 노드의 부모노드를 저장할 2차원 배열
15
16      /* 부모노드의 초기화 */
17      for(int i=0; i<n; i++) {
18          for(int j=0; j<n; j++) {
19              if(i==j or matrix[i][j] == maxValue){
20                  parent[i][j] = NULL;
21              } else if(i != j and matrix[i][j] < maxValue) {
22                  parent[i][j] = i;
23              }
24          }
25      }
26      /* 부모노드의 초기화 끝 */
27      //초기값 출력
28      printf("초기 데이터값\n");
29      for(int i=0; i<n ; i++) {
30          for(int j=0; j<n; j++) {
31              printf("%d ", d[i][j]);
32
33          }
34          printf("\n");
35      }
36      printf("부모값\n");
37      for(int i=0; i<n ; i++) {
38          for(int j=0; j<n; j++) {
39              printf("%d ", parent[i][j]);
40          }
41          printf("\n");
42      }
43      printf("---------------\n\n");
44      /* 초기값 출력 끝*/
45
46      /* Floyd and Warshall Algorithms */
47      for(int k=0; k<n; k++) { //모든 노드에 대해서 모든 노드로 향하는 최단거리 검색
48          for(int i=0; i<n; i++) {
49              for(int j=0; j<n; j++) {
50                  if(d[i][k] + d[k][j] < d[i][j]  ) { //현재 가중치가 현재 노드로 들어오는 부모노드와 나가는 노드의 합보다 높다면
51                      d[i][j] = d[i][k] + d[k][j]; //해당 합값으로 가중치를 바꿈
52                      parent[i][j] = parent[k][j]; //부모노드의 갱신
53                  }
54              }
55          }
56          /* 가중치와 부모노드가 갱신되는 과정을 알고리즘에 포함시켜서 출력 */
57          printf("%d 번째 가중치 갱신\n", k+1);
58          for(int i=0; i<n ; i++) {
59              for(int j=0; j<n; j++) {
60                  printf("%d ", d[i][j]);
61              }
62              printf("\n");
63          }
64          printf("%d 번째 노드의 부모 갱신\n", k+1);
65          for(int i=0; i<n ; i++) {
66              for(int j=0; j<n; j++) {
67                  printf("%d ", parent[i][j]);
68              }
69              printf("\n");
70          }
71          printf("\n");
72          /* 가중치와 부모노드가 갱신되는 과정을 알고리즘에 포함시켜서 출력 */
73      }
74      /* Floyd and Warshall Algorithms */
75  }
```

- 입력받은 2차원 배열을 이용하여 APSP를 구현하는 함수
- 모든 노드에 대해서 임의의 한 노드로 들어오는 노드들의 가중치와 임의의 한 노드에서 나가는 노드의 가중치의 합을 비교하여 갱신하는 작업.
- 노드의 가중치보다 들어오고 나가는 노드들의 가중치의 합이 더 작다면 가중치를 갱신하고 해당 부모노드를 찾아 갱신한다.

3. 결과

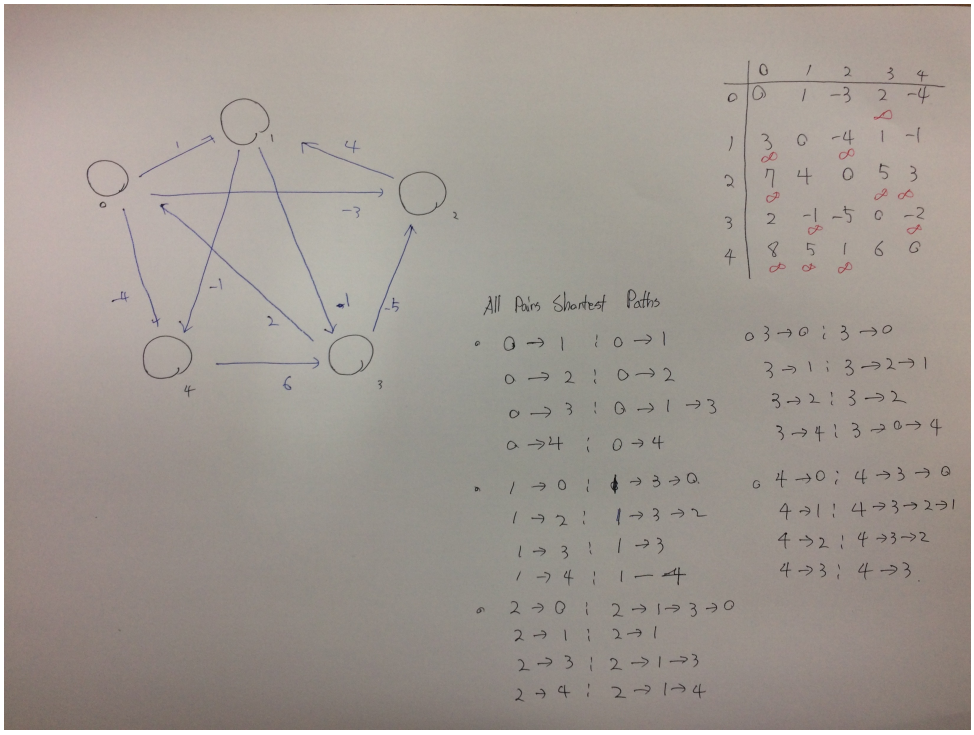
1. Matrix Multiplication

```
초기 데이터값
0 3 8 8388607 -4
8388607 0 8388607 1 7
8388607 4 0 8388607 8388607
2 8388607 -5 0 8388607
8388607 8388607 8388607 6 0
초기 데이터값 끝 --------

Slow All Pairs Shortest Paths ---------
1 번째 가중치 갱신
0 3 8 2 -4
3 0 -4 1 7
8388607 4 0 5 11
2 -1 -5 0 -2
8 8388607 1 6 0
2 번째 가중치 갱신
0 3 -3 2 -4
3 0 -4 1 -1
7 4 0 5 11
2 -1 -5 0 -2
8 5 1 6 0
3 번째 가중치 갱신
0 1 -3 2 -4
3 0 -4 1 -1
7 4 0 5 3
2 -1 -5 0 -2
8 5 1 6 0

Fast all Pairs Shortest Paths ---------
1 번째 가중치 갱신
0 3 8 2 -4
3 0 -4 1 7
8388607 4 0 5 11
2 -1 -5 0 -2
8 8388607 1 6 0
2 번째 가중치 갱신
0 1 -3 2 -4
3 0 -4 1 -1
7 4 0 5 3
2 -1 -5 0 -2
8 5 1 6 0
Hello, World!
Program ended with exit code: 0
```

- Matrix Multiplication 의 두 알고리즘, Slow와 Fast 방법의 결과를 출력한다.



- 두 방법은 실행속도만 다를 뿐 결과는 동일하다.

- 부모노드의 지정방법을 알 수 없어 계산되어 출력된 가중치를 토대로 모든 노드에 대해서의 루트를 그려보았다.
- 이때 사진 속 빨간색 infinity는 초기의 루트가 없는 가중치값이다.

2. Floyd and Warshall Algorithms

```
초기 데이터값
0 3 8 8388607 -4
8388607 0 8388607 1 7
8388607 4 0 8388607 8388607
2 8388607 -5 0 8388607
8388607 8388607 8388607 6 0
부모값
0 0 0 0 0
0 0 0 1 1
0 2 0 0 0
3 0 3 0 0
0 0 0 4 0
-------------

1 번째 가중치 갱신
0 3 8 8388607 -4
8388607 0 8388607 1 7
8388607 4 0 8388607 8388603
2 5 -5 0 -2
8388607 8388607 8388607 6 0
1 번째 노드의 부모 갱신
0 0 0 0 0
0 0 0 1 1
0 2 0 0 0
3 0 3 0 0
0 0 0 4 0

2 번째 가중치 갱신
0 3 8 4 -4
8388607 0 8388607 1 7
8388607 4 0 5 11
2 5 -5 0 -2
8388607 8388607 8388607 6 0
2 번째 노드의 부모 갱신
0 0 0 1 0
0 0 0 1 1
0 2 0 1 1
3 0 3 0 0
0 0 0 4 0

3 번째 가중치 갱신
0 3 8 4 -4
8388607 0 8388607 1 7
8388607 4 0 5 11
2 -1 -5 0 -2
8388607 8388607 8388607 6 0
3 번째 노드의 부모 갱신
0 0 0 1 0
0 0 0 1 1
0 2 0 1 1
3 2 3 0 0
0 0 0 4 0

4 번째 가중치 갱신
0 3 -1 4 -4
3 0 -4 1 -1
7 4 0 5 3
2 -1 -5 0 -2
8 5 1 6 0
4 번째 노드의 부모 갱신
0 0 3 1 0
3 0 3 1 0
3 2 0 1 0
3 2 3 0 0
3 2 3 4 0

5 번째 가중치 갱신
0 1 -3 2 -4
3 0 -4 1 -1
7 4 0 5 3
2 -1 -5 0 -2
8 5 1 6 0
5 번째 노드의 부모 갱신
0 2 3 4 0
3 0 3 1 0
3 2 0 1 0
3 2 3 0 0
3 2 3 4 0
```

- 매번 반복 마다 가중치와 부모 노드가 어떻게 갱신되는지를 출력한다.
- Floyd and Warshall Algorithm 역시 Matrix Multiplication과 동일한 결과가 출력되었다는 것을 확인할 수 있다. 즉 모든 노드로부터 다른 모든 노드로 향하는 최단거리는 두 알고리즘, Matrix Multiplication 과 Floyd and Warshall 알고리즘 이 동일한 결과와 동일한 경로임을 확인할 수 있다.