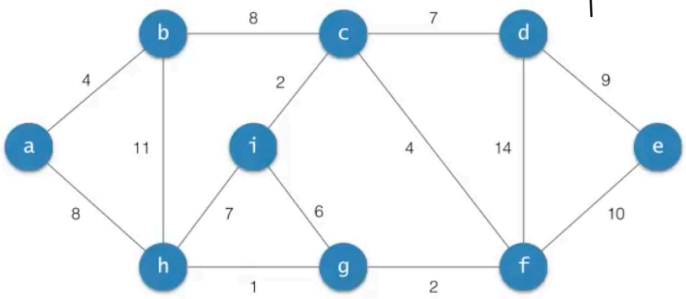
개요

신장 트리(Spanning Tree) 혹은 신장 부분 그래프(Spanning Subgraph)는 모든 꼭짓점을 포함하는 부분 그래프를 말한다. 즉 다시 말해서 트리의 특수한 형태로 그래프의 모든 정점을 포함하면서 그 모든 정점들이 연결되어 있어야하며, 트리의 속성을 만족하기 위해 순환은 존재하지 않는 형태를 말한다. 또한 이를 만족하기 위해 N개의 정점을 가지고 있으면 정확히 N-1개의 이음선으로 연결되어 있어야 한다. 이번 실험은 이러한 신장트리를 최소의 비용으로 만족하는 트리를 찾는 것으로써, 이를 최소 비용 신장트리(Minimum Spanning Tree)라고 한다. 최소 비용 신장트리는 다양한 분야에서 응용되는 가장 기본적인 그래프 문제이다. 보통 네트워크(통신, 도로, 가스배관 등) 디자인과 같이 추상적으로 언급되는데, 연결되어 있으면서 비용이 최소화 된다는 것은 모든 종류의 네트워크에서 요구되어지는 것이기 때문이다. 또한 이미지 프로세싱과 같은 분야에서도 이와 같은 알고리즘을 많이 사용된다.

구체적인 예를 들자면, 아래의 [그림 1]과 같이 N개의 도시가 있고 도시와 도시를 연결하는 도로의 비용 W가 있다.



[그림 1] 예제 그래프

위의 그래프에서 정점들이 도시들이고 이음선의 가중치가 비용이라고 가정한다. 이음선이 없는 경우에는 어떠한 이유로 두 도시를 직접 연결하는 도로를 만드는 것이 불가능하거나 그 비용이 무한대라고 가정한다. 예산이 충분하다면 모든 도시간의 도로를 만들 수 있지만, 이는 불가능하므로 최소의 비용을 가지고 모든 도시들이 연결되도록 하는 방법을 구할 때 최소 비용 신장 트리문제라고 한다 이 때 그래프의 조건을 요약하자면 다음과 같다.

1. **무방향 가중치 그래프**
2. **각 정점 (u, v)에 대해 가중치 W(u, v)가 있으며**
3. **모든 정점들이 서로 연결된다.**
4. **가중치의 합이 최소가 된다.**

이러한 최소 비용 신장트리를 구하기 위한 가장 유명한 알고리즘이 바로 Prim과 Kruskal 알고리즘이다. 이번 실험에서는 이 두 알고리즘으로 주어진 문제에 대한 최소 비용 신장트리를 구할 것이다.

1. Prim 알고리즘으로 MST구하기

프림 알고리즘은 지금까지 연결된 정점에서 연결된 이음선들 중 하나씩 선택하면서 최소 비용 신장트리를 만들어 가는 방식이다. 그 과정을 설명하면 다음과 같은 과정을 반복한다.

1. **임의의 정점 하나에서 시작한다.**
2. **선택한 정점과 인접한 정점들 중의 가중치가 최소인 이음선을 선택한다.**
3. **선택한 이음선에 이어진 정점을 정점의 집합에 포함시킨다.**
4. **모든 정점이 다 선택될 때까지 2번 항목을 반복한다.**

이러한 프림 알고리즘의 과정을 의사코드로 나타내면 다음과 같다.

```
void prim(int n, const number W[][], set_of_edges& F) {
    index i, vnear;
    number min;
   edge e;
```

```
    index nearest[2…n];
    nuber distance[2…n];

    F = 공집합;
    for(i = 2; i <= n; i++) {
        nearest[i] = 1;
        distance[i] = W[1][i];
    }

    repeat(n −1 times) {
        min = 무한대;
        for(i=2; i <= n; i++) {
            if(0 <= distance[i] < min) {
                min = distance[i];
                vnear = I;
            }
        e = evnear 가 인덱스인 마디를 y 에 추가한다.
        add e to F;
        distance[vnear] = −1;
        for(i = 2; i <= n; i++) {
            if(W[i][vnear] < distance[i]) {
                distance[i] = W[i][vnear];
                nearest[i] = vnear;
            }
    }
}
```

[코드 1] Prim 알고리즘 의사코드

의사코드를 C언어로 구현하면 다음과 같다.

```
/*
 F = 그래프에 대한 최소비용 신장트리 안에 있는 이음선(Edge)의 집합
 Y = 최소비용 신장트리의 마디(Vertex) 집합
 nearest[i] = vi에 가장 가까운 Y에 속한 마디의 인덱스
 distance[i] = vi와 nearest[i]가 인덱스인 두 마디를 연결하는 이음선의 가중치
*/
void Prim(int start) {
   int nearest[MAX] = {0};
   int distance[MAX] = {0};
   int i, j;
   int min, vnear = 0; // distance[i]값이 최소가 되는 마디 인덱스

   for(i = 0; i < MAX; i++) {
      if(i == start) continue;    // 모든 마디에 대하여 Y에 속한 가장 가까운 마디
      nearest[i] = start;         // (nearest[i])는 v1으로 초기화하고,
      distance[i] = W[start][i];  // Y로부터 거리(distance[i])는 vi와 v1을
   }                              // 연결하는 이음선의 가중치로 초기화한다.

   for(j = 1; j < MAX; j++) {  // MAX – 1번 반복
      // MAX – 1개의 마디를 Y에 추가한다.
      min = INF;
      for(i = 0; i < MAX; i++) {
         if(i == start) continue;
         if(0 <= distance[i] && distance[i] < min) { // 각 마디에 대하여 distance[i]를 검사하여
            min = distance[i];                  // Y에 가장 가까이 있는 마디(vnear)을 찾는다.
            vnear = i;
         }
      }

      distance[vnear] = FALSE;
```

```
        for(i = 0; i < MAX; i++) {
            if(i == start) continue;            // Y에 속하지 않은
            if(W[i][vnear] < distance[i]) {     // 각 마디에 대하여
                distance[i] = W[i][vnear];      // Y로부터의 거리(distance[i])를
                nearest[i] = vnear;            // 갱신한다.
            }
        }
    }

    printf("\tdistance: \n");
    print1DMatrix(distance);
    printf("\tnearest: \n");
    print1DMatrix(nearest);
}
```

[코드 2] Prim()

주어진 문제를 이 알고리즘을 통해 v1을 선택하여 최소 비용 신장트리를 구하면 다음과 같은 결과가 나온다.

```
distance:
[ 0]    [-1]    [-1]    [-1]    [-1]    [-1]    [-1]    [-1]    [-1]    [-1]

nearest:
[ 0]    [ 0]    [ 3]    [ 0]    [ 3]    [ 9]    [ 2]    [ 3]    [ 7]    [ 8]
```

[그림 2] 정점 v1을 선택했을 때 최소 비용 신장트리

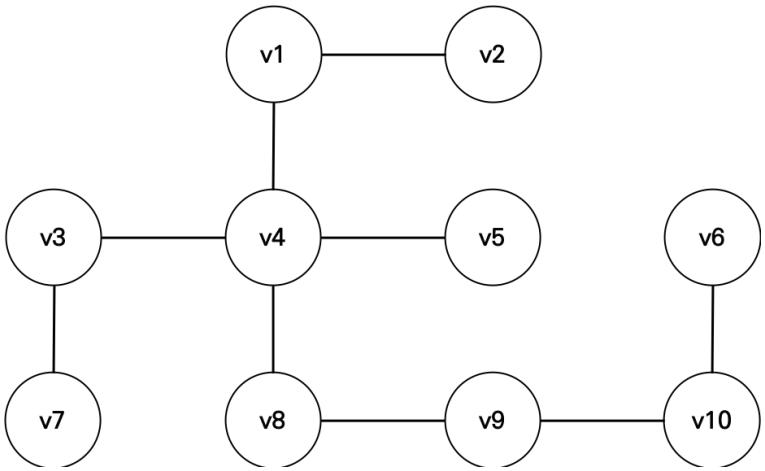
마찬가지로 v10을 선택하여 최소 비용 신장트리를 구해도 마찬가지의 결과가 도출된다.

```
distance:
[-1]    [-1]    [-1]    [-1]    [-1]    [-1]    [-1]    [-1]    [-1]    [ 0]

nearest:
[ 3]    [ 0]    [ 3]    [ 7]    [ 3]    [ 9]    [ 2]    [ 8]    [ 9]    [ 0]
```

[그림 3] 정점 v10을 선택했을 때 최소 비용 신장트리

이렇게 구한 최소 비용 신장 트리의 모습은 다음과 같다.



[그림 4] 주어진 그래프의 최소 비용 신장 트리

2. Kruskal 알고리즘으로 MST구하기

크루스칼 알고리즘은 프림 알고리즘과 달리 가중치가 최소인 이음선을 하나씩 선택하면서 신장 트리를 만드는 알고리즘이다. 그 과정을 설명하면 다음과 같은 과정을 반복한다.

1. **모든 이음선을 가중치에 따라 오름차순으로 정렬**
2. **가중치가 가장 낮은 이음선부터 차례대로 선택하면서 트리를 구성**
3. **만약 이음선을 선택했을 때 순환이 존재하면 다음 이음선으로 넘어간다.**
4. **N-1개의 이음선이 선택될 때 까지 2번 항목을 반복한다.**

이러한 크루스칼 알고리즘의 과정을 의사코드로 나타내면 다음과 같다.

```
void kruskal(int n, int m, set_of_edges E, set_of_edges& F) {
   index i, j;
   set_pointer p, q;
   edge e;

   E에 속한 m개의 이음선을 가중치가 작은 것부터 차례로 정렬한다;
   F = 공집합;
   initial(n)
   while(F에서 이음선의 수는 n-1보다 작다) {
      e = 아직 고려하지 않은 이음선 중 가중치가 최소인 이음선;
      i, j = e로 연결된 마디의 인덱스;
      p = find(i);
      q = find(j);
      if(!equal(p, q)) {
         merge(p, q);
         e를 F에 추가;
      }
   }
}
```

[코드 3] Kruskal 알고리즘 의사코드

의사코드를 C언어로 구현하면 다음과 같다.

```
void Kruskal() {
   int i, j;
   int tmp, tmp2;
   int edge_count = 0;
   int sum = 0;
   int cycle = 0;
   int w[MAX][MAX];
   int V[MAX];
   Edge *edge = calloc(MAX*MAX, sizeof(Edge));
   memcpy(w, W, sizeof(W));

   for(i = 0; i < MAX; i++)
      for(j = i; j < MAX; j++)
         if(W[i][j] != INF && W[i][j] != 0) {
            edge[edge_count].i = i;
            edge[edge_count].j = j;
            edge[edge_count].weight = W[i][j];
            edge_count++;
         }

   Divide(edge, 0, edge_count - 1, edge_count); // 가중치 오름차순 합병 정렬
   for(i = 0; i < edge_count; i++)
      printf("\t(%d, %d) %d\n", edge[i].i, edge[i].j, edge[i].weight);

   memset(V, TRUE, sizeof(V));

   for(tmp = 0; tmp < edge_count; tmp++) {
      cycle = 0;

      for(i = 0; i < MAX; i++)
```

```
            if(w[edge[tmp].i][i] == -1 && w[i][edge[tmp].j] == -1) {
               w[edge[tmp].i][edge[tmp].j] = -1;
               w[edge[tmp].j][edge[tmp].i] = -1;
            }

         if(V[edge[tmp].i] == -1 && V[edge[tmp].j] == -1)
            for(i = 0; i < MAX; i++)
               if(w[edge[tmp].i][i] == -1 && w[i][edge[tmp].j] == -1) {
                  cycle = 1;
                  break;
               }

         if(cycle == 0) {
            printf("\t(%d, %d) ", edge[tmp].i, edge[tmp].j);
            V[edge[tmp].i] = -1;
            V[edge[tmp].j] = -1;
            w[edge[tmp].i][edge[tmp].j] = -1;
            w[edge[tmp].j][edge[tmp].i] = -1;
            sum += edge[tmp].weight;
         }

         for(i = 0; i < MAX; i++)
            for(j = 0; j < MAX; j++)
               for(tmp2 = 0; tmp2 < MAX; tmp2++)
                  if(w[i][tmp2] == -1 && w[tmp2][j] == -1) {
                     w[i][j] = -1;
                     w[j][i] = -1;
                  }
      }
      printf("\n\tSum = %d\n", sum);
}

void Divide(Edge *edge, int left, int right, int n) {
      int middle = (left + right) / 2;
      if (left < right) {
         Divide(edge, left, middle, n);
         Divide(edge, middle + 1, right, n);
         Combine(edge, left, middle, right, n);
      }
}

void Combine(Edge *edge, int left, int middle, int right, int n) {
      Edge* temp = copyArray(edge, n);
      int i = left;
      int j = middle + 1;
      int l;
      int position = left;

      /* 분할 정렬된 list의 합병 */
      while (i <= middle && j <= right) {
         if (edge[i].weight <= edge[j].weight)
            temp[position++] = edge[i++];
         else temp[position++] = edge[j++];
      }
      //남아 있는 값들을 일괄 복사
      if (i > middle)
         for (l = j; l <= right; l++)
            temp[position++] = edge[l];
      else
         for (int l = i; l <= middle; l++)
            temp[position++] = edge[l];
      for (int i = 0; i < n; i++)
         edge[i] = temp[i];
}
```

[코드 4] Kruskal()

주어진 문제를 이 알고리즘을 통해 을 선택하여 최소 비용 신장트리를 구하면 다음과 같은 결과가 나온다.

```
(3, 7)  (7, 8)  (2, 6)  (5, 9)  (3, 4)  (8, 9)  (0, 3)  (2, 3)  (0, 1)
Sum = 107
```

[그림 5] Kruskal 알고리즘 결과

3. ChainedMatrixMultiplication.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define MAX 10        //노드의 개수
#define INF 987654321  //간선의 가중치가 없는 경우 무한대 표시
#define TRUE 1
#define FALSE -1

typedef struct Edge {
   int weight;
   int i, j;
} Edge;

void Prim(int start);
void Kruskal(void);
void print1DMatrix(int matrix[MAX]);
void print2DMatrix(int matrix[MAX][MAX]);
void Divide(Edge *edge, int left, int right, int n);
void Combine(Edge *edge, int left, int middle, int right, int n);
Edge* copyArray(Edge *array, int n);

// W = 이음선의 가중치(존재O=가중치, 존재X=INF, i==j = 0)
const int W[MAX][MAX] = {
   {0, 32, INF, 17, INF, INF, INF, INF, INF, INF},
   {32, 0, INF, INF, 45, INF, INF, INF, INF, INF},
   {INF, INF, 0, 18, INF, INF, 5, INF, INF, INF},
   {17, INF, 18, 0, 10, INF, INF, 3, INF, INF},
   {INF, 45, INF, 10, 0, 28, INF, INF, 25, INF},
   {INF, INF, INF, INF, 28, 0, INF, INF, INF, 6},
   {INF, INF, 5, INF, INF, INF, 0, 59, INF, INF},
   {INF, INF, INF, 3, INF, INF, 59, 0, 4, INF},
   {INF, INF, INF, INF, 25, INF, INF, 4, 0, 12},
   {INF, INF, INF, INF, INF, 6, INF, INF, 12, 0}
};

int main(void) {
   Prim(9);
   Kruskal();
   return 0;
}

/*
 F = 그래프에 대한 최소비용 신장트리 안에 있는 이음선(Edge)의 집합
 Y = 최소비용 신장트리의 마디(Vertex) 집합
 nearest[i] = vi에 가장 가까운 Y에 속한 마디의 인덱스
 distance[i] = vi와 nearest[i]가 인덱스인 두 마디를 연결하는 이음선의 가중치
 */
void Prim(int start) {
   int nearest[MAX] = {0};
   int distance[MAX] = {0};
   int i, j;
   int min, vnear = 0; // distance[i]값이 최소가 되는 마디 인덱스

   for(i = 0; i < MAX; i++) {
      if(i == start) continue;    // 모든 마디에 대하여 Y에 속한 가장 가까운 마디
      nearest[i] = start;        // (nearest[i])는 v1으로 초기화하고,
      distance[i] = W[start][i];  // Y로부터 거리(distance[i])는 vi와 v1을
   }                             // 연결하는 이음선의 가중치로 초기화한다.

   for(j = 1; j < MAX; j++) {  // MAX - 1번 반복
      // MAX - 1개의 마디를 Y에 추가한다.
      min = INF;
      for(i = 0; i < MAX; i++) {
         if(i == start) continue;
```

```
            if(0 <= distance[i] && distance[i] < min) { // 각 마디에 대하여 distance[i]를 검사하여
               min = distance[i];                  // Y에 가장 가까이 있는 마디(vnear)을 찾는다.
               vnear = i;
            }
         }

         distance[vnear] = FALSE;
         for(i = 0; i < MAX; i++) {
            if(i == start) continue;           // Y에 속하지 않은
            if(W[i][vnear] < distance[i]) {    // 각 마디에 대하여
               distance[i] = W[i][vnear];      // Y로부터의 거리(distance[i])를
               nearest[i] = vnear;            // 갱신한다.
            }
         }
      }

      printf("\tdistance: \n");
      print1DMatrix(distance);
      printf("\tnearest: \n");
      print1DMatrix(nearest);
}

void Kruskal() {
      int i, j;
      int tmp, tmp2;
      int edge_count = 0;
      int sum = 0;
      int cycle = 0;
      int w[MAX][MAX];
      int V[MAX];
      Edge *edge = calloc(MAX*MAX, sizeof(Edge));
      memcpy(w, W, sizeof(W));

      for(i = 0; i < MAX; i++) {
         for(j = i; j < MAX; j++) {
            if(W[i][j] != INF && W[i][j] != 0) {
               edge[edge_count].i = i;
               edge[edge_count].j = j;
               edge[edge_count].weight = W[i][j];
               edge_count++;
            }
         }
      }

      // 가중치 오름차순 합병 정렬
      Divide(edge, 0, edge_count - 1, edge_count);
      for(i = 0; i < edge_count; i++) {
         printf("\t(%d, %d) %d\n", edge[i].i, edge[i].j, edge[i].weight);
      }

      memset(V, TRUE, sizeof(V));

      for(tmp = 0; tmp < edge_count; tmp++) {
         cycle = 0;

         for(i = 0; i < MAX; i++) {
            if(w[edge[tmp].i][i] == -1 && w[i][edge[tmp].j] == -1) {
               w[edge[tmp].i][edge[tmp].j] = -1;
               w[edge[tmp].j][edge[tmp].i] = -1;
            }
         }

         if(V[edge[tmp].i] == -1 && V[edge[tmp].j] == -1)
            for(i = 0; i < MAX; i++) {
               if(w[edge[tmp].i][i] == -1 && w[i][edge[tmp].j] == -1) {
                  cycle = 1;
                  break;
               }
            }

         if(cycle == 0) {
```

```
            printf("\t(%d, %d) ", edge[tmp].i, edge[tmp].j);
            V[edge[tmp].i] = -1;
            V[edge[tmp].j] = -1;
            w[edge[tmp].i][edge[tmp].j] = -1;
            w[edge[tmp].j][edge[tmp].i] = -1;
            sum += edge[tmp].weight;
        }

        for(i = 0; i < MAX; i++) {
            for(j = 0; j < MAX; j++) {
                for(tmp2 = 0; tmp2 < MAX; tmp2++) {
                    if(w[i][tmp2] == -1 && w[tmp2][j] == -1) {
                        w[i][j] = -1;
                        w[j][i] = -1;
                    }
                }
            }
        }
    }
    printf("\n\tSum = %d\n", sum);
}


void Divide(Edge *edge, int left, int right, int n) {
    int middle = (left + right) / 2;
    if (left < right) {
        Divide(edge, left, middle, n);
        Divide(edge, middle + 1, right, n);
        Combine(edge, left, middle, right, n);
    }
}

void Combine(Edge *edge, int left, int middle, int right, int n) {
    Edge* temp = copyArray(edge, n);
    int i = left;
    int j = middle + 1;
    int l;
    int position = left;

    /* 분할 정렬된 list의 합병 */
    while (i <= middle && j <= right) {
        if (edge[i].weight <= edge[j].weight)
            temp[position++] = edge[i++];
        else temp[position++] = edge[j++];
    }
    //남아 있는 값들을 일괄 복사
    if (i > middle)
        for (l = j; l <= right; l++)
            temp[position++] = edge[l];
    else
        for (int l = i; l <= middle; l++)
            temp[position++] = edge[l];
    for (int i = 0; i < n; i++)
        edge[i] = temp[i];
}

Edge* copyArray(Edge *edge, int n) {
    Edge* temp = calloc(n, sizeof(Edge));
    for (int i = 0; i < n; i++) {
        temp[i] = edge[i];
    }
    return temp;
}

void print1DMatrix(int matrix[MAX]) {
    int i;
    for(i = 0; i < MAX; i++) {
        printf("\t[%2d]", matrix[i]);
    }
    printf("\n\n");
}

void print2DMatrix(int matrix[MAX][MAX]) {
```

```
    int i, j;
    for(i = 0; i < MAX; i++) {
        for(j = 0; j < MAX; j++) {
            printf("\t[%2d]", matrix[i][j]);
        }
        printf("\n");
    }
    printf("\n");
}
```