알고리즘분석 실습 자료

7주차

Photo by Piotr Guzik on Unsplash

실습 소요 시간 100분

4장 탐욕적인 접근방법 (Greedy Algorithm)

실습프로그램

- ✓ Prim 알고리즘
- ✓ Kruskal 알고리즘

Greedy Algorithm

- 탐욕적인 알고리즘(greedy algorithm)은 결정을 해야 할 때마다 그 순간에 가장 좋다고 생각되는 것을 해답으로 선택함으로써 최종적인 해답에 도달한다.

- 그 순간의 선택은 그 당시(local)에는 최적이다. 그러나 최적이라고 생각했던 해답들을 모아서 최종적인(global)해답을 만들었다고 해서, 그 해답이 궁극적으로 최적이라는 보장이 없다.

- 따라서 탐욕적인 알고리즘은 항상 최적의 해답을 주는지를 반드시 검증해야 한다.

탐욕적인 알고리즘 설계 절차

1. **선정과정(selection procedure)**

   현재 상태에서 가장 좋으리라고 생각되는(greedy) 해답을 찾아서 해답모음(solution set)에 포함시킨다.

2. **적정성점검(feasibility check)**

   새로 얻은 해답모음이 적절한지를 결정한다.

3. **해답점검(solution check)**

   새로 얻은 해답모음이 최적의 해인지를 결정한다.

최소비용신장트리 (minimum spanning tree)

- 신장트리가 되는 G의 부분그래프 중에서 가중치가 최소가 되는 부분그래프를 **최소비용신장트리(minimum spanning tree)**라고 한다. 여기서 최소의 가중치를 가진 부분그래프는 반드시 트리가 되어야 한다. 왜냐하면, 만약 트리가 아니라면, 분명히 순환경로(cycle)가 있을 것이고, 그렇게 되면 순환경로 상의 한 이음선을 제거하면 더 작은 비용의 신장트리가 되기 때문이다.

- 관찰: 모든 신장트리가 최소비용신장트리는 아니다.

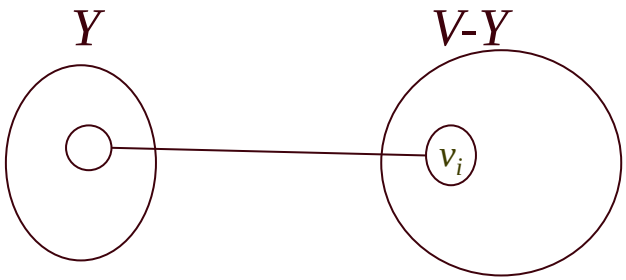
탐욕적인 알고리즘

- 문제: 비방향성 그래프 $G = (V,E)$가 주어졌을 때, $F \subseteq E$를 만족하면서, (V,F)가 G의 최소비용신장트리(MST)가 되는 F를 찾는 문제.
- 알고리즘:

1. $F := \emptyset$;
2. 최종해답을 얻지 못하는 동안 다음 절차를 계속 반복
   - (a) **선정 절차**:
     적절한 최적해 선정절차에 따라서 하나의 이음선을 선정
   - (b) **적정성 점검**:
     선정한 이음선을 F에 추가시켜도 사이클이 생기지 않으면,
     F에 추가.
   - (c) **해답 점검**: $T = (V,F)$가 신장트리이면, 사례해결. T는 최소비용신장트리.

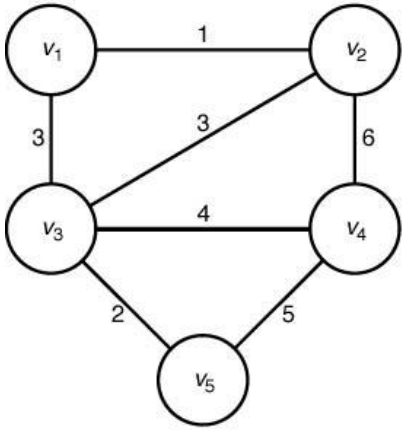
Prim의 알고리즘(1930)

1. F :=Ø;
2. Y := {v_1};
3. while(사례 미해결){
   (a) **선정 절차/적정성 점검**: V - Y에 속한 정점 중에서, Y에 가장 가까운 정점 하나를 선정.
   (b) 선정한 정점을 Y에 추가.
   (c) Y로 이어지는 이음선을 F에 추가.
   if (Y==V)
   (d) **해답 점검**: $Y = V$가 되면, $T = (V, F)$가 최소비용신장트리
   }

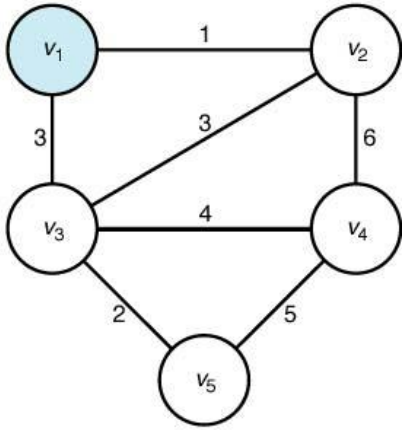


Prim's 알고리즘

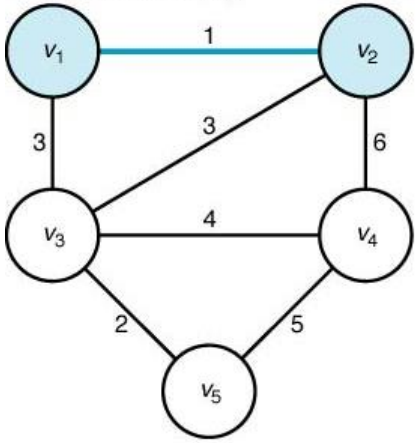
Determine a minimum spanning tree.



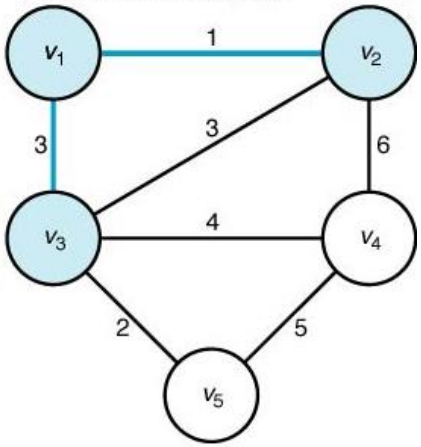
1. Vertex v_1 is selected first.



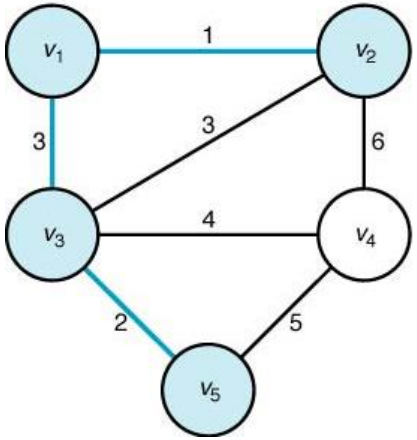
2. Vertex v_2 is selected because it is nearest to $\{v_1\}$.



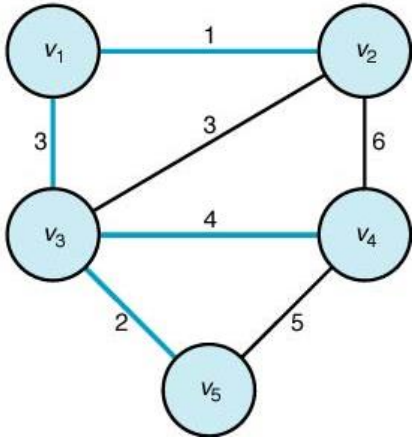
3. Vertex v_3 is selected because it is nearest to $\{v_1, v_2\}$.



4. Vertex v_2 is selected because it is nearest to $\{v_1, v_2, v_3\}$.



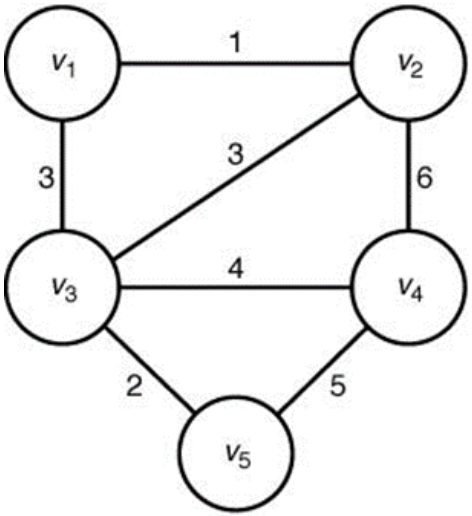
5. Vertex v_4 is selected.



[실습프로그램] Prim 알고리즘

```
import utility

inf = 1000
w=[[0,  1,  3,inf, inf],
   [1,  0,  3,6,   inf],
   [3,  3,  0,4,   2],
   [inf,6,  4,0,   5],
   [inf,inf,2,5,   0]]
```




```
F=set()
utility.printMatrix(w)
n=len(w)
nearest=n*[0]
distance=n*[0]
for i in range(1,n):
    nearest[i]=0
    distance[i]=w[0][i]



     prim's 알고리즘 구현



print()
print(F)
```

Prim 알고리즘 output

```
>>>
   0    1    3 1000 1000
   1    0    3    6 1000
   3    3    0    4    2
1000    6    4    0    5
1000 1000    2    5    0

{(4, 2), (2, 0), (1, 0), (3, 2)}
>>>
```

Kruskal의 알고리즘(1956)

1. $F := \phi$;
2. 서로소(素, disjoint)가 되는 V의 부분집합 들을 만드는데, 각 부분집합 마다 하나의 정점만 가짐
3. E안에 있는 이음선을 가중치의 비내림차순으로 정렬
4. while(답을 구하지 못했음){
   (a) **선정 절차**: 최소가중치를 갖고 있는 다음 이음선을 선정
   (b) **적정성 점검**: 만약 선정된 이음선이 두 개의 서로소인 정점을 잇는다면, 먼저 그 부분집합을 하나의 집합으로 합하고, 그 다음에 그 이음선을 F에 추가한다.
   (c) **해답 점검**: 만약 모든 부분집합이 하나의 집합으로 합하여 지면,
   그 때 $T = (V,F)$가 최소비용신장트리.
   }

```
void kruskal(int n, int m,     // 입력: 정점의 수 n, 에지의 수 m
             set_of_edges E,   // 입력: 가중치를 포함한 이음선의 집합
             set_of_edges& F) {// 출력: MST를 이루는 이음선의 집합
  index i, j;
  set_pointer p, q;
  edge e;
  E에 속한 m개의 이음선을 가중치의 비내림차순으로 정렬;
  F = ϕ;
  initial(n);
  while (F에 속한 이음선의 개수가 n-1보다 작다) {
    e = 아직 점검하지 않은 최소의 가중치를 가진 이음선;
    (i, j) = e를 이루는 양쪽 정점의 인덱스;
    p = find(i);
    q = find(j);
    if (!equal(p,q)) {
      merge(p,q);
      e를 F에 추가;
    }
  }
}
```

[실습프로그램] Kruskal Algorithm

```
parent = dict()
rank = dict()

def make_singleton_set(v):
    parent[v] = v
    rank[v] = 1

def find(v):
    if parent[v] != v:
        parent[v] = find(parent[v])
    return parent[v]

def union(r1, r2):
    if r1 != r2:
        if rank[r1] > rank[r2]:
            parent[r2] = r1
            rank[r1] += rank[r2]
        else:
            parent[r1] = r2
            if rank[r1] == rank[r2]: rank[r2] += rank[r1]
```

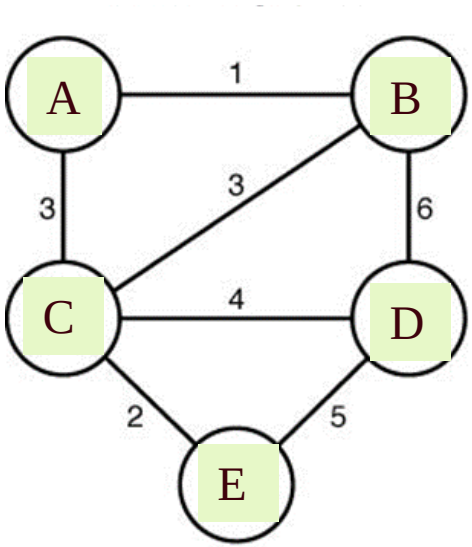
```
def kruskal(graph):




    Kruskal algorithm 구현




graph = {
        'vertices': ['A', 'B', 'C', 'D', 'E'],
        'edges': set([
            (1, 'A', 'B'),
            (3, 'A', 'C'),
            (3, 'B', 'C'),
            (6, 'B', 'D'),
            (4, 'C', 'D'),
            (2, 'C', 'E'),
            (5, 'D', 'E'),
            ])
        }
mst=kruskal(graph)
print(mst)
```



Kruskal Algorithm output

```
>>>
{(2, 'C', 'E'), (4, 'C', 'D'), (1, 'A', 'B'), (3, 'A', 'C')}
>>>
```