제 4 장 욕심쟁이 방법(탐욕적인 방법) (Greedy Method)

- 일반적인 방법
- 최소-비용 신장 트리
- 단일 출발지 최단-경로

일반적인 방법

- 가장 간단한 설계 방법이나 다양한 문제들에 적용될 수 있다.
- 많은 문제들에서 n 개의 입력이 주어지며 특정 제약조건을 만족하는 해를 구한다.
- 가능 해(feasible solution): 주어진 제약조건을 만족하는 해
  최적 해(optimal solution) : 가능 해 중 주어진 목적함수를 최소 또는 최대로 하는 해
- 두 가지 유형: 부분 집합형, 순서형
- 부분 집합형의 욕심쟁이 방법은 한번에 한 개의 입력을 고려하며, 그 입력이 제약조건을 만족시키면 해의 집합에 포함시키고, 그렇지 않으면 버린다(예: 최소 비용 신장 트리 문제의 Kruskal 알고리즘).
  이때 한 개의 입력은 남은 입력들 중 가장 큰 이익을 주는 것으로 선택한다.

- 부분 집합형의 제어 추상화:

```
  SolType Greedy(Type a[], int n)
// a[1:n] contains the n inputs
{
    SolType solution = EMPTY;      // Initialize the solutions.
    for (int i=1; i <= n; i++) {
         Type  x = Selection(a) ;     // 여기에 Greedy 전략이 사용됨
         if Feasible(solution, x)
             solution = Union(solution, x) ;
     }
     return solution;
}
```

- 순서형의 욕심쟁이 방법은 특정한 순서로 입력을 고려한다. 각 결정은 이미 내려진 결정을 이용하여 계산될 수 있는 최적 평가 기준을 사용하여 만들어 진다. 이때 사용되는 최적 평가 기준이 Greedy 전략에 해당한다 (예: 단일 출발지 최단 경로 문제의 Dijkstra 알고리즘).
- 순서형의 제어 추상화:

```
  SolType Greedy(Type a[], int n)
// a[1:n] contains the n inputs
{
    SolType solution = EMPTY;      // Initialize the solutions.
    for (int i=1; i <= n; i++) {
         Type  x = Selection(a) ;    // 여기에 Greedy 전략이 사용됨
         solution = Union(solution, x) ;
     }
     return solution;
}
```

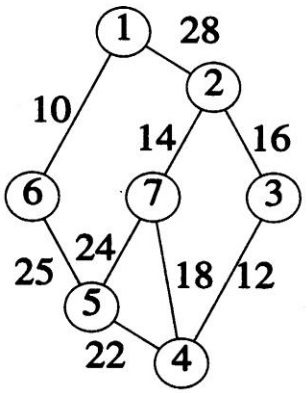
4.1 최소-비용 신장 트리

- 최소비용 신장 트리**(minimum cost spanning tree):** 가중치 무방향 그래프에서 신장 트리의 비용은 신장 트리에 포함된 간선들의 비용이며, 최소 비용 신장 트리는 최소의 비용을 갖는 신장 트리이다.
- 최소 비용 신장 트리는 다음의 제한적인 조건을 갖는다.

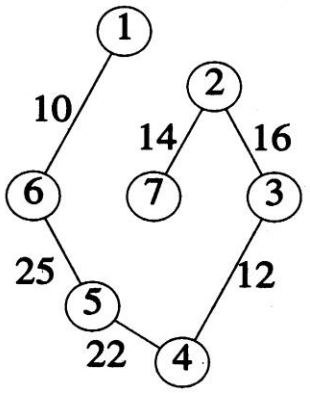
(i) 그래프 내의 간선들만을 사용해야 한다.

(ii) n-1 개의 간선만을 포함해야 한다.

(iii) 사이클을 포함하지 않는다.



(a)



(b)

욕심쟁이 방법

- 최소-비용 신장 트리를 구하기 위한 욕심쟁이 방법은 간선들을 하나씩 처리함으로써 트리를 구성한다. 고려되는 다음의 간선은 특정 최적화 기준(욕심쟁이 전략)에 따라 선택된다.
- Kruskal의 알고리즘:

  그래프의 간선들을 비용의 오름차순으로 차례차례 고려하며, 사이클을 형성하지 않을 때 포함시킨다.
- Prim의 알고리즘:

  지금까지 포함된 간선들의 비용의 합을 최소로 증가시키는 간선을 선택한다.
- 간선들의 부분 집합으로 해를 만들므로 부분 집합형이다.

Kruskal의 알고리즘

- 한 번에 하나씩 비용이 가장 작은 간선을 선택하여 T에 이미 포함된 간선들과 사이클을 형성하지 않는 간선들만을 차례로 T에 추가한다. T에 n-1개의 간선들이 존재하면 멈춘다.
- 알고리즘

```
T = Ø
while ((T가 n-1개 미만의 간선을 포함) && (E가 공백이 아님)) {
    E에서 최소 비용 간선 (v,w) 선택;
    E에서 (v,w)를 삭제;
    if ((v,w)가 T에서 사이클을 형성하지 않음) T에 (v,w)를 추가;
    else (v,w)를 거부;
}
if (T가 n-1개 미만의 간선을 포함) cout << "신장 트리 없음" << endl;
```

- 시간복잡도: O(e log e)

Kruskal 알고리즘의 예

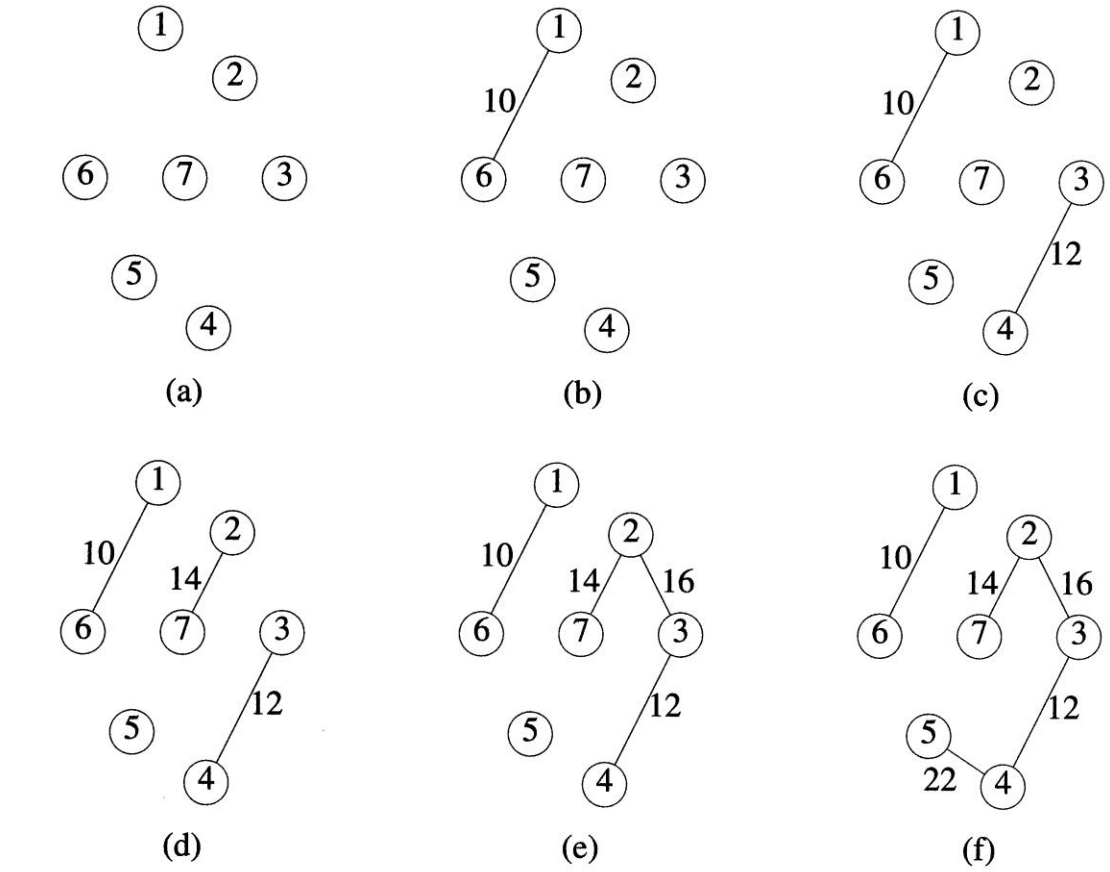


Figure 4.8 Stages in Kruskal's algorithm

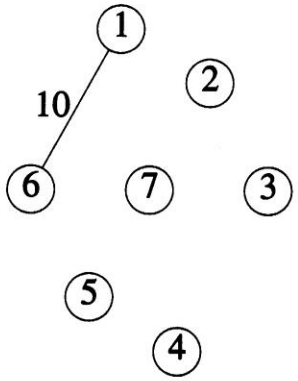
Prim의 알고리즘

- 트리 T에 인접한 간선들 중 T와 사이클을 형성하지 않는 최소 비용 간선 (u, v)를 구해 T에 추가한다. T에 n-1 개의 간선이 포함될 때까지 이러한 추가 단계를 반복한다. (초기에 T의 정점 집합은 하나의 정점만을 포함한다.)
- 알고리즘

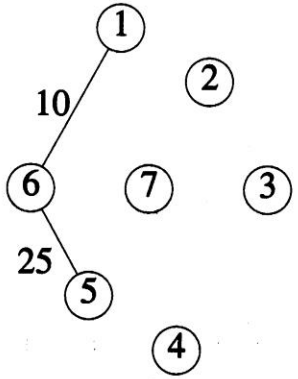
```
// G가 최소한 하나의 정점을 가진다고 가정.
TV = {1}; // 정점 1로 시작. 간선은 비어있음.
for(T=Ø ; T의 간선수가 n-1보다 적음; (u,v)를 T에 추가)
{
  u ∈ TV이고 v ¬∈ TV인 최소 비용 간선을 (u,v)라 함;
  if (그런 간선이 없음) break;
  v를 TV에 추가;
}
if (T의 간선수가 n-1보다 적음) cout << "신장트리 없음" << endl;
```

- 시간복잡도: $O(n^2)$

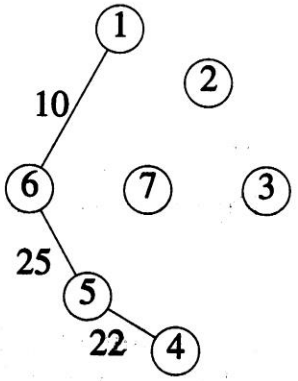
Prim 알고리즘의 예



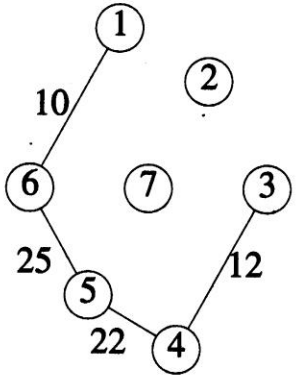
(a)



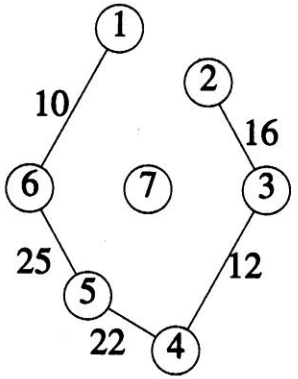
(b)



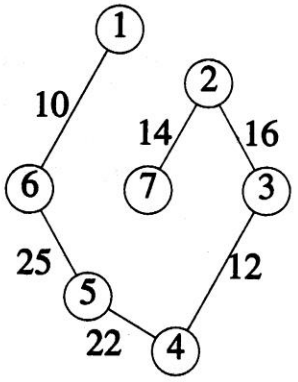
(c)



(d)



(e)



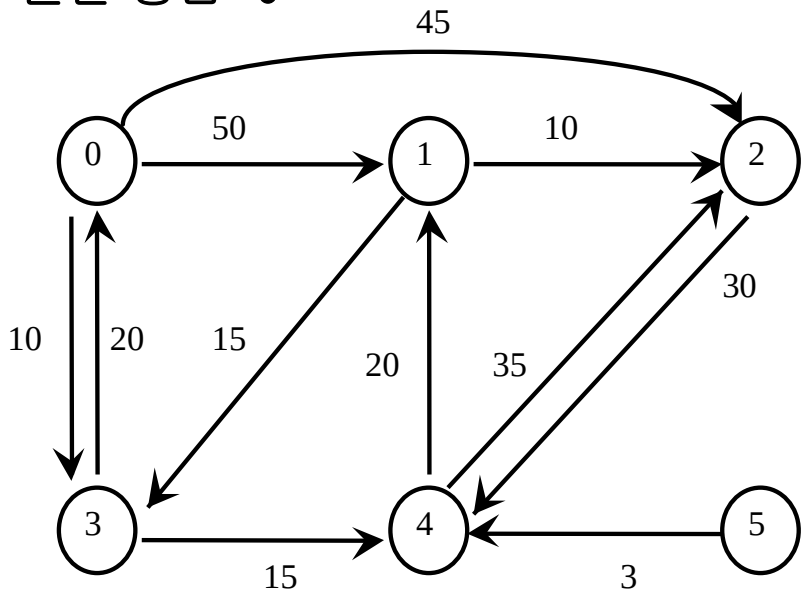
(f)

4.2 단일 출발지 최단-경로

- 가중치 그래프, 출발지(source), 목적지(destination), 방향그래프
- 최단 경로 문제:
  - 단일 출발지/모든 종점(간선의 길이가 양수일 경우)
  - 단일 출발지/모든 종점(일반적일 경우)
  - 모든 쌍의 최단 경로
- 여기서는 단일 출발지/모든 종점(양수인 경우) 만을 생각
  - “A 에서 B 로 가는 최단 경로와 최단 비용이 무엇인가?” 를 포함
  - Dijkstra 알고리즘

최단 경로의 예

- 출발 정점: 0



(a) 그래프

Path	*Length*
1) 0, 3	10
2) 0, 3, 4	25
3) 0, 3, 4, 1	45
4) 0, 2	45

(b) 0 에서부터 최단 경로

욕심쟁이 방법

- 출발점 v 를 포함하여 이미 최단 경로가 발견된 정점들의 집합을 S라 하자. 정점 w가 S에 속하지 않을 때, dist[w]를 v에서 시작하여 S에 있는 정점들만을 거쳐 w까지의 최단 경로의 길이라 하자.
- 만일 다음으로 짧은 최단 경로가 정점 u 까지의 경로라면 v에서 u로의 경로는 오직 S에 속한 정점들만을 통하게 된다.

- 이때 이 경로는 v에서 u까지의 최단 경로가 되어야만 한다. (그렇지 않으면 모순)
- 여기서 선택된 정점 u 는 S의 원소가 된다. 이때 v 에서 시작하여 S에 있는 속하지 않은 w 까지의 최단 경로의 길이는 감소할 수 있다(이유: u가 새로 S에 추가되었기 때문)
- 따라서 u 를 거쳐서 w 로 갈 경우를 현재의 dist[w]와 비교하여 더 작을 경우 dist[w]를 갱신하여야만 한다.
- 즉, dist[w] = min { dist[w], dist[u]+length[u, w] },
  단 w는 S에 포함 안된 정점

	0	1	2	3	4	5
0	0	50	45	10	∞	∞
1	∞	0	10	15	∞	∞
2	∞	∞	0	∞	30	∞
3	20	∞	∞	0	15	∞
4	∞	20	35	∞	0	∞
5	∞	∞	∞	∞	3	0

min{ ∞, 10+15 }

S	선택 정점(u)	dist 0	1	2	3	4	5
		0	50	45	10	∞	∞
0	3	0	50	45	10	25	∞
0, 3	4	0	45	45	10	25	∞
0, 3, 4	1	0	45	45	10	25	∞
0, 1, 3, 4	2	0	45	45	10	25	∞
0, 1, 2, 3, 4							

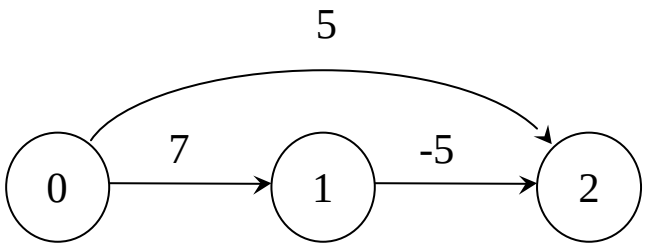
min{ 50, 25+20 }

최단경로(lec6-1)

```
1 void ShortestPath(int n, int v)
2 // dist[j],0 ≤ j< n은 n개의 정점을 가진 방향 그래프 G에서 정점 v로부터 정점 j 까지
3 // 의 최단 경로 길이로 설정됨. 간선의 길이는 length[j][j]로 주어짐.
4 {
5   for (int i=0; i<n; i++) {s[i] = false; dist[i] = length[v][i]; }// 초기화
6   s[v] = true;
7   dist[v] = 0;
8   for (i=0; i<n-2; i++) { // 정점 v로부터 n-1개 경로를 결정
9      int u = choose(n); // choose는 dist[u] = minimum dist[w]인 u를 반환
10                              // (여기서 s[w]=FALSE)
11    s[u] = true;
12    for (int w=0; w<n; w++)
13       if(!s[w])
14          if(dist[u] + length[u][w] < dist[w])
15                dist[w] = dist[u] + length[u][w];
16 } // for(i=0; ...)의 끝
17}
```

- 분석: 8행의 for 루프는 n-2번 실행

  for 루프 내의 9행은 최대 n 번 실행, 그리고 12번행 역시 최대 n 번 실행, 따라서 $O(n^2)$ 시간 걸린다.
- 간선이 음의 길이를 가질 때, 프로그램이 정확히 작동하지 않을 수 있다.



- 정점 2까지의 길이는 dist[2]=5 이지만 실제 2가 최단 경로의 길이이다.