[CSE3081(2반)] 알고리즘 설계와 분석

2020학년도 2학기

강의자료

(2020.12.03 목요일)

서강대학교 공과대학 컴퓨터공학과
임 인 성 교수

본 강의에서 제작하여 제공하는 **PDF 파일, 동영상, 그리고 예제 코드 등의 강의 자료**의 저작권은 특별히 명기되어 있지 않은 한 서강대학교에 있습니다.

본인의 학습 목적 외에 공개된 장소에 올리거나 타인에게 배포하는 등의 행위를 금합니다. 협조 부탁합니다.

[주제 6]
Graph Algorithms

Prim's Minimum Spanning Tree Algorithm

- **Idea**
  - In each step, find and add **an edge of the least possible weight** that connects the (current) tree to a non-tree vertex.
- **Algorithm**

Given $G = (V, E)$,

Begin with a tree $T^0 = (V^0, E^0)$ where $V^0 = \{v_1\}$ and $E_0 = \{\}$.

repeat { // $T^i = (V^i, E^i)$ → $T^{i+1} = (V^{i+1}, E^{i+1})$

Select a vertex v in $V - V^i$ that is **nearest** to V^i.

// Let v is from the edge (u, v), where u in V^i.

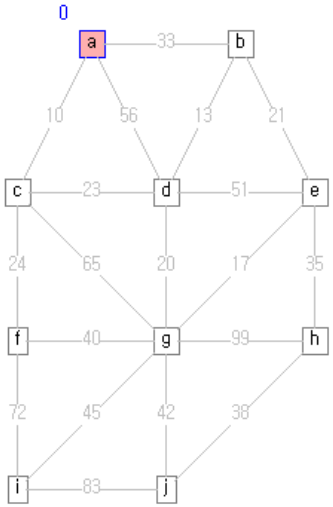
Update T in such a way that

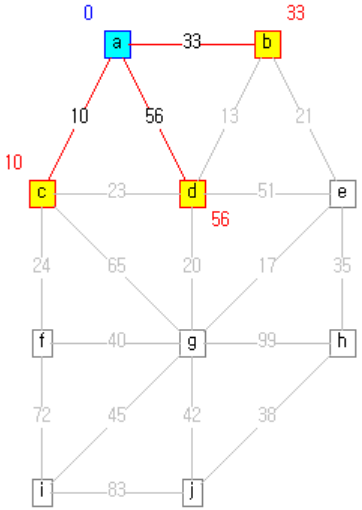
$V^{i+1} = V^i + \{v\}$, and $E^{i+1} = E^i + \{(u, v)\}$.

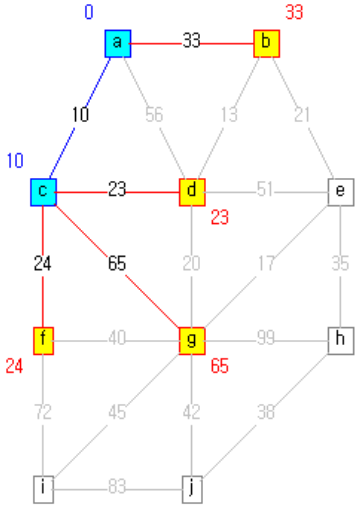
until (an MST is found)

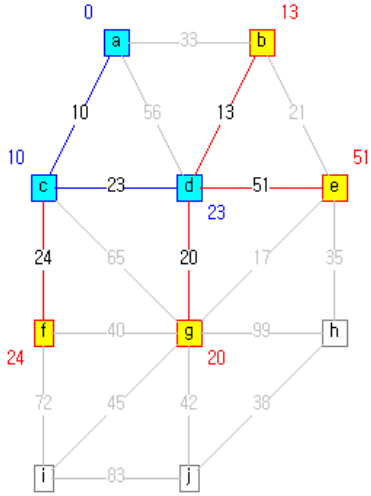
- **A key issue in implementation**
  - Tree vertices와 non-tree vertices들을 어떻게 관리할 것인가?
  - Tree vertices와 non-tree vertices들 간의 최소 비용 edge를 어떻게 (효율적으로) 찾을 것인가?

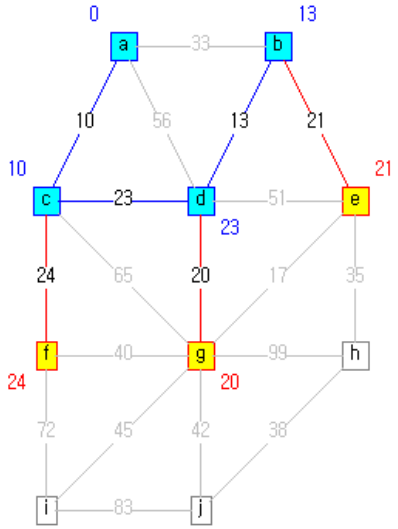
From Prof. Kenji Ikeda's Home Page

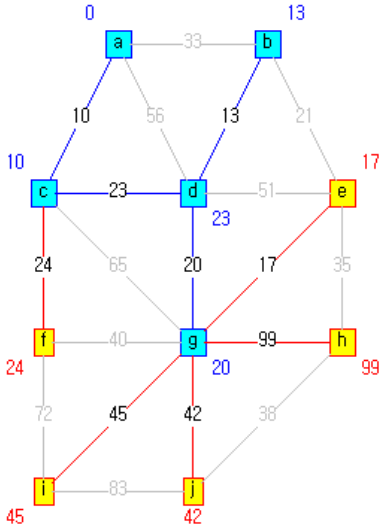


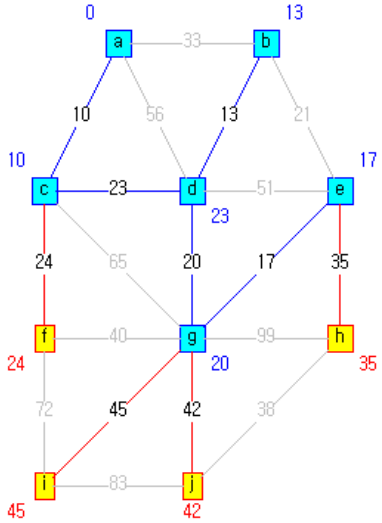


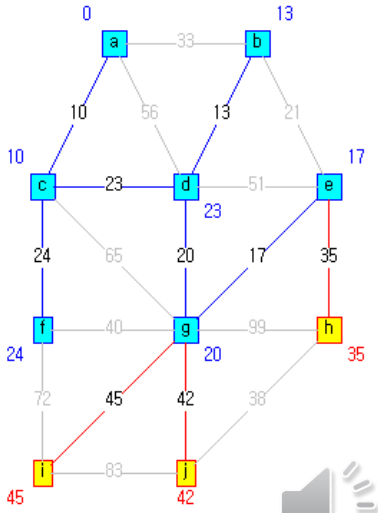












Prim's Minimum Spanning Tree Algorithm

- **Idea**
  - In each step, find and add **an edge of the least possible weight** that connects the (current) tree to a non-tree vertex.
- **Algorithm**

Given $G = (V, E)$,

Begin with a tree $T^0 = (V^0, E^0)$ where $V^0 = \{v_1\}$ and $E_0 = \{\}$.

repeat { // $T^i = (V^i, E^i)$ → $T^{i+1} = (V^{i+1}, E^{i+1})$

Select a vertex v in $V - V^i$ that is **nearest** to V^i.

// Let v is from the edge (u, v), where u in V^i.

Update T in such a way that

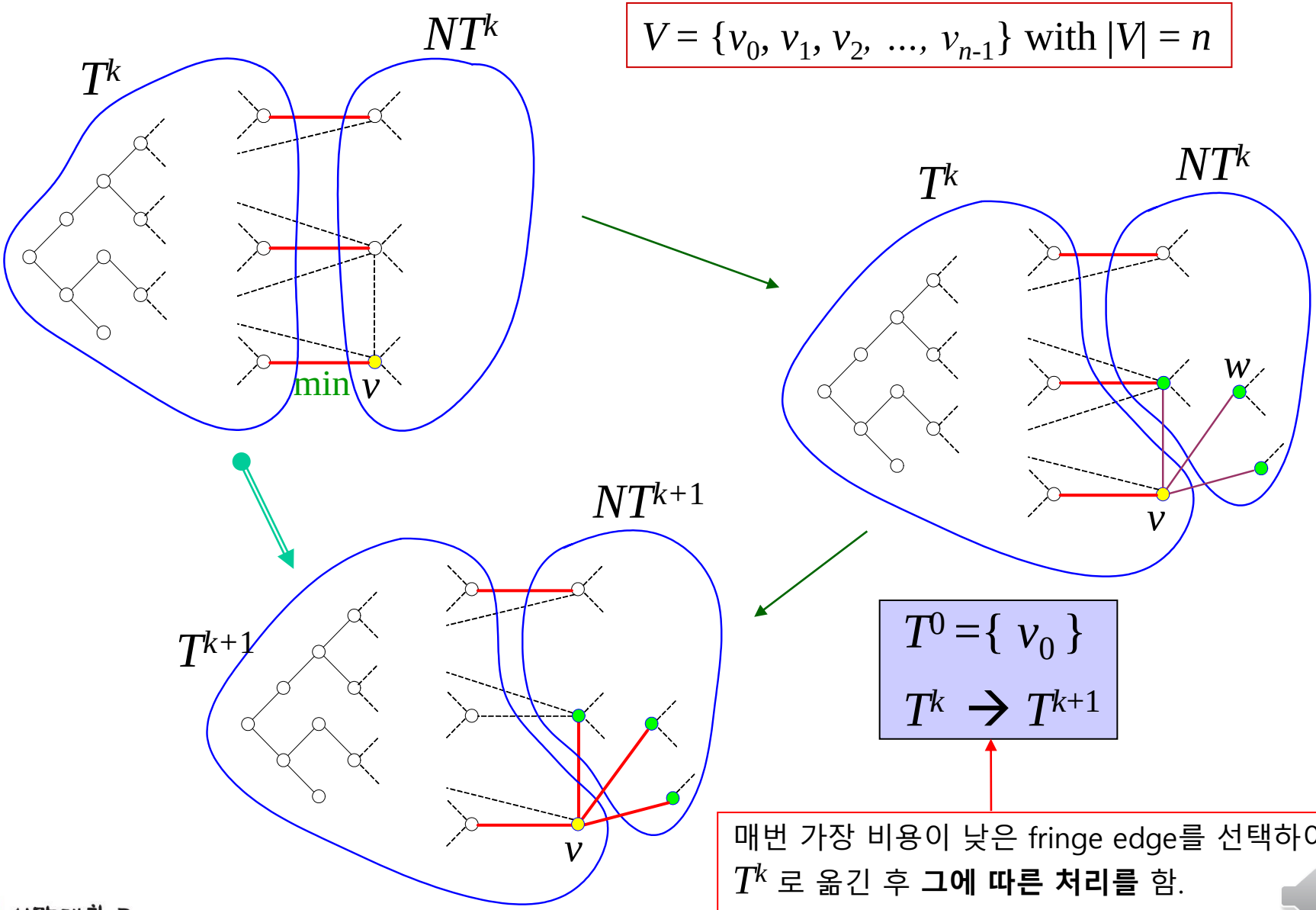
$V^{i+1} = V^i + \{v\}$, and $E^{i+1} = E^i + \{(u, v)\}$.

until (an MST is found)

- **A key issue in implementation**
  - Tree vertices와 non-tree vertices들을 어떻게 관리할 것인가?
  - Tree vertices와 non-tree vertices들 간의 최소 비용 edge를 어떻게 (효율적으로) 찾을 것인가?

Inductive Description of the Prim's Algorithm

$V = \{v_0, v_1, v_2, \ldots, v_{n-1}\}$ with $|V| = n$



$T^0 = \{ v_0 \}$

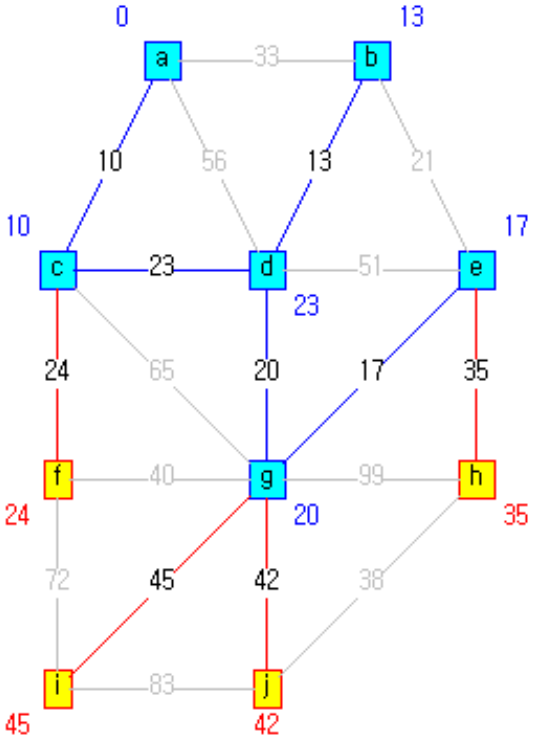
$T^k \rightarrow T^{k+1}$

매번 가장 비용이 낮은 fringe edge를 선택하여 T^k 로 옮긴 후 **그에 따른 처리를** 함.

An O(n²) Implementation: Adjacency Matrix 사용

- **st[i]:** T로 선택된 vertex i의 parent vertex 번호 저장
- **fr[i]:** NT에 있는 vertex i에서 T에 있는 vertex 중 가장 가까운 vertex의 번호
- **wt[i]:** NT에 있는 vertex i에 대해 그 vertex에서 **fr[i]**까지의 거리

n = | V |



a = 0, b = 1, c = 2, d = 3, e = 4, f = 5, g = 6, h = 7, i = 8, j = 9

v	st[v]	fr[v]	wt[v]
0 (a)	0	0	maxWT
1 (b)	3	3	13
2 (c)	0	0	10
3 (d)	2	2	23
4 (e)	6	6	17
5 (f)	-1	2	24
6 (g)	3	3	20
7 (h)	-1	4	35
8 (i)	-1	6	45
9 (j)	-1	6	42
10			maxWT

dummy vertex

dummy weight

```
static int fr[maxV];
#define P G->adj[v][w]
void GRAPHmstV(Graph G, int st[], double wt[]) {
  int v, w, min, n = G->V;
  for (v = 0; v < n; v++) {
     st[v] = -1; fr[v] = v; wt[v] = maxWT;
  }
  wt[n] = maxWT;
  for (min = 0; min != n; ) {
     v = min; st[min] = fr[min];
     for (w = 0, min = n; w < n; w++)
        if (st[w] == -1) {
           if (P < wt[w]) {
              wt[w] = P; fr[w] = v;
           }
           if (wt[w] < wt[min]) min = w;
        }
  }
}
```

st[i]: T로 선택된 vertex i의 parent vertex 번호 저장
fr[i]: NT에 있는 vertex i에서 T에 있는 vertex 중 가장 가까운 vertex의 번호
wt[i]: NT에 있는 vertex i에 대해 그 vertex에서 **fr[i]**까지의 거리

- Check to see whether adding the new edge brought any nontree vertex closer to the tree.
- Find the next edge to add to the tree.

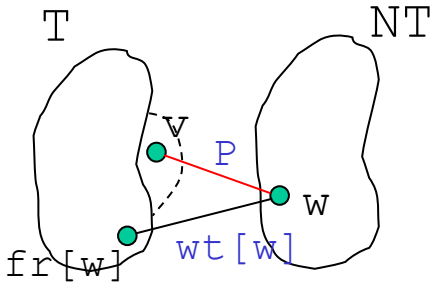
*** 언제 끝날까?**

*** 모든 계산이 끝난 후 wt[i]는 어떤 정보를 가지고 있을까?**

다음으로 선택된 vertex 번호

아직 선택되지 않은 모든 vertex w에 대해, v = min이 선택된 것에 대한 update 수행

wt[w]를 update하면서, 동시에 가장 작은 wt 값을 가지는 vertex 번호 min을 계산



An O(e log n) Implementation: Adjacency List 사용

- **Observations**

n = | V |, e = | E |

  - The inner for-loop in the $O(n^2)$ implementation visits all the vertices to update wt[] array and to find the minimum.

$0 \le e \le \frac{n(n-1)}{2}$

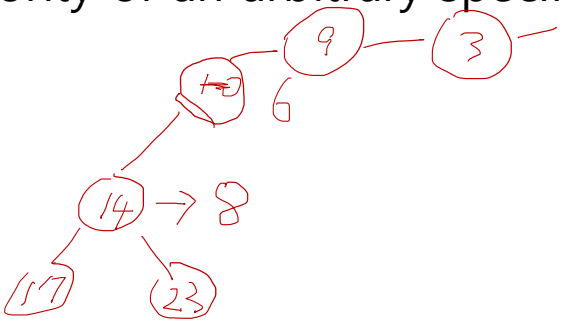
$O(n) \le e \le O(n^2)$

  - An $O(e \log n)$ time implementation is possible.
    - If the graph is dense, … $n^2 \log n$
    - If the graph is sparse, … $n \log n$

$\Downarrow$

$\frac{n^2}{\log n}$

  - We need to employ the **priority queue** that allows
    - to insert a new item (**PQinsert(w)),**
    - to delete the minimum item (w = **PQdelmin()**), and
    - to change the priority of an arbitrary specified item (**PQdec(w)**).



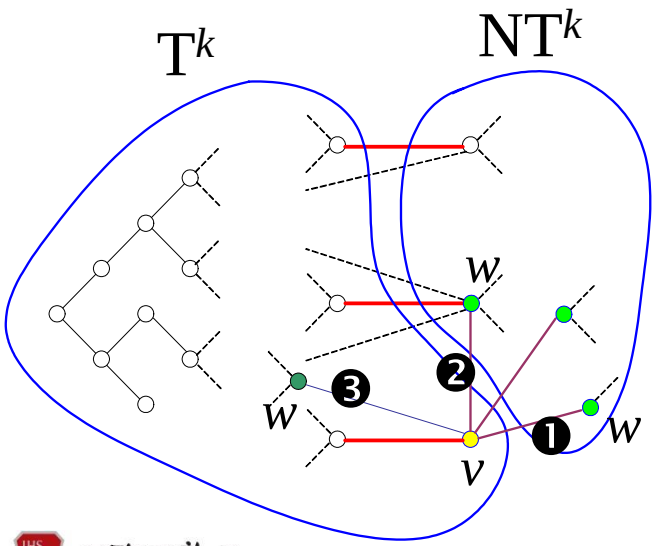
```
typedef struct node *link;
struct node { int v; double wt, link next; };
struct graph { int V; int E; link *adj; };
typedef struct graph *Graph;
```

st[i]: T로 선택된 vertex i의 parent vertex 번호 저장
fr[i]: NT에 있는 vertex i에서 T에 있는 vertex 중 가장 가까운 vertex의 번호
wt[i]: NT에 있는 vertex i에 대해 그 vertex에서 **fr[i]**까지의 거리

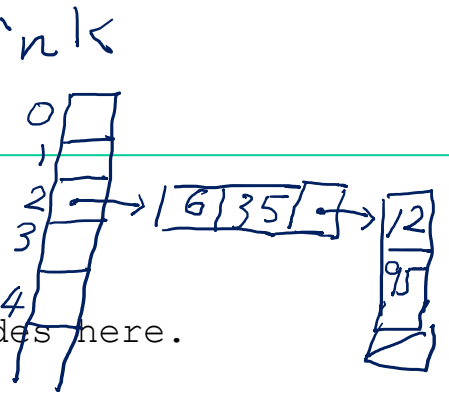
❶ `(fr[w] == -1)`

❷ `(fr[w] != -1 && st[w] == -1)`

❸ `(fr[w] != -1 && st[w] != -1)`




```
#define GRAPHpfs GRAPHmst
static int fr[maxV];
static double *priority;
// Put the priority queue codes here.
#define P t->wt
void GRAPHpfs(Graph G, int st[], double wt[]) {
   link t; int v, w;
   PQinit();
   priority = wt;
   for (v = 0; v < G->V; v++) {
      st[v] = -1; fr[v] = -1;
   }
   fr[0] = 0; PQinsert(0);
   while (!PQempty()) {
      v = PQdelmin(); st[v] = fr[v];
      for (t = G->adj[v]; t != NULL; t = t->next)
         if (fr[w = t->v] == -1) {
            wt[w] = P; PQinsert(w); fr[w] = v;
         }
         else if ((st[w] == -1) && (P < wt[w])) {
            wt[w] = P; PQdec(w); fr[w] = v;
         }
   }
}
```



$n = |V|, e = |E|$

O(n)

O(nlogn)

O(logn)

O(elogn)

O(logn) each