# Data Structure and Algorithm

## Class 10

Seongjin Lee

Updated: 2017-03-06
DSA_2017_10

insight@gnu.ac.kr
http://resourceful.github.io
Systems Research Lab.
GNU

# Table of contents

# Review: Kruskal's Algorithm

# Kruskal's algorithm

Kruskal's algorithm builds a minimum cost spanning tree T by adding edges to T one at a time.

- The algorithm selects the edges for inclusion in T in nondecreasing order of their cost.
- An edge is added to T if it does not form a cycle with the edges that are already in T.
- Since G is connected and has n>0 vertices, exactly n-1 edges will be selected for inclusion in T.

# The code: Kruskal's Algorithm

# Kruskal's algorithm

```c
// code adopted from "C 언어로 쉽게 풀어쓴 자료구조", 천인국, 생능출판사

#include <stdio.h>

#define MAX_VERTICES 100 // Max number of vertices

int parent[MAX_VERTICES]; // parent of the vertex
int vcnt[MAX_VERTICES];  // number of vertices in the subset
```

# Kruskal's algorithm

```
// initializing the set for union-find operation
void set_init(int n){
    int i;
    for (i = 0; i<n; i++) {
        parent[i] = -1;
        vcnt[i] = 1;
    }
}
```

# Kruskal's algorithm cntd

```
// return the parent in the set with the vertex (iterative)
int set_find(int vertex){
    int p, s, i = -1;
    for (i = vertex; (p = parent[i]) >= 0; i = p) // repeat till the root
        ; // if it is the root than no-operation
    s = i;  // set representative node in the set
    for (i = vertex; (p = parent[i]) >= 0; i = p)
        parent[i] = s; // set the parent of all nodes as s
    return s;
}
```

# Kruskal's algorithm cntd

```
// merge the two sets with the nodes
void set_union(int u, int v){
    if (vcnt[u] < vcnt[v]) {
        parent[u] = v;
        vcnt[v] += vcnt[u];
    }
    else {
        parent[v] = u;
        vcnt[u] += vcnt[v];
    }
}
```

# Kruskal's algorithm cntd

```
typedef struct {
    int u;  // src vertex
    int v;  // dst vertex
    int cost; // used as the cost of the edge
} element;

// min heap to find the edge with the least cost
#define MAX_ELEMENT 100
typedef struct {
    element heap[MAX_ELEMENT];
    int heap_size;
} HeapType;
```

# Kruskal's algorithm

```
// Initialize the heap
void init(HeapType *h){
    h->heap_size = 0;
}
```

```
// insert an item into the heap
void insert_min_heap(HeapType *h, element item){
    int i;
    i = ++(h->heap_size);

    // iteratively comparing with the parent of the node
    while ((i != 1) && (item.cost < h->heap[i / 2].cost)) {
        h->heap[i] = h->heap[i / 2];
        i /= 2;
    }
    h->heap[i] = item; // add new item
}
```

# Kruskal's algorithm cntd I

```
// delete an item from the heap
element delete_min_heap(HeapType *h){
    int parent, child;
    element item, temp;

    item = h->heap[1];
    temp = h->heap[(h->heap_size)--];
    parent = 1;
    child = 2;
    while (child <= h->heap_size) {
        // find the smaller child of current parent
        if ((child < h->heap_size) &&
            (h->heap[child].cost) > h->heap[child + 1].cost)
            child++;
        if (temp.cost <= h->heap[child].cost) break;
```

# Kruskal's algorithm cntd II

```
        // move to next level
        h->heap[parent] = h->heap[child];
        parent = child;
        child *= 2;
    }
    h->heap[parent] = temp;
    return item;
}
```

```c
// inserting an edge to the heap wrapper
void insert_heap_edge(HeapType *h, int u, int v, int weight){
    element e;
    e.u = u;
    e.v = v;
    e.cost = weight;
    insert_min_heap(h, e);
    //print_heap(h);
}
```

# Kruskal's algorithm cntd

```c
// insert edges to the heap
void insert_all_edges(HeapType *h){
    insert_heap_edge(h, 0, 1, 4);
    insert_heap_edge(h, 0, 7, 8);
    insert_heap_edge(h, 1, 2, 8);
    insert_heap_edge(h, 1, 7, 11);
    insert_heap_edge(h, 2, 3, 7);
    insert_heap_edge(h, 2, 5, 4);
    insert_heap_edge(h, 2, 8, 2);
    insert_heap_edge(h, 3, 4, 9);
    insert_heap_edge(h, 3, 5, 14);
    insert_heap_edge(h, 4, 5, 10);
    insert_heap_edge(h, 5, 6, 2);
    insert_heap_edge(h, 6, 7, 1);
    insert_heap_edge(h, 6, 8, 6);
    insert_heap_edge(h, 7, 8, 7);
}
```

## Kruskal's algorithm cntd

```
// kruskal minimum spanning tree
void kruskal(int n){
    int edge_accepted = 0; // number of edges in the spanning tree
    HeapType h;    // heap data structure
    int uset, vset;  // the set for the u and v in the graph
    element e;    // heap element
    int tcost = 0;  // total cost of the spanning three

    init(&h);     // initialize the heap
    insert_all_edges(&h); // insert the edges into the heap
    set_init(n);    // union−find set initialize
```

# Kruskal's algorithm cntd

```
    while (edge_accepted < (n − 1)) // # of edges in MST should be less
        than n−1
    {
        e = delete_min_heap(&h); // take the min element from the heap
        uset = set_find(e.u);  // find the set with the vertex u
        vset = set_find(e.v);  // find the set with the vertex v
        if (uset != vset) {   // if they are in different sets
            printf("(%d,%d) %d \n", e.u, e.v, e.cost);
            tcost += e.cost;
            edge_accepted++;
            set_union(uset, vset); // merge the two sets
        }
    }
    printf("Total Minimum Spanning Tree Cost is %d\n", tcost);
}
```