CSE 5095 Notes

Author : Michael Tedford

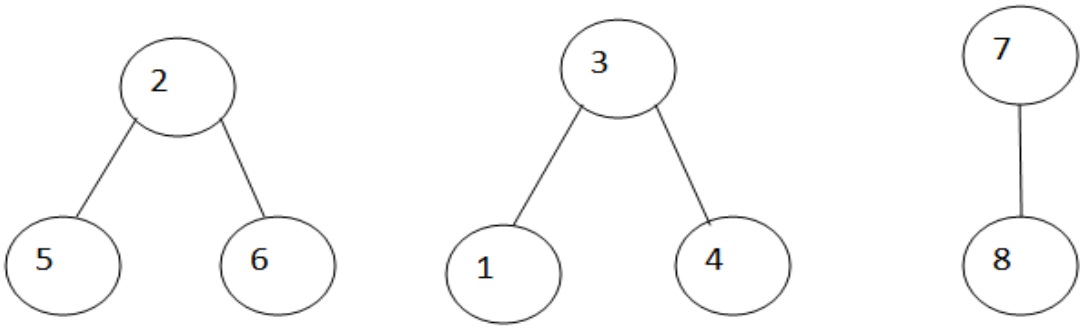
Class Date : 18th February 2014

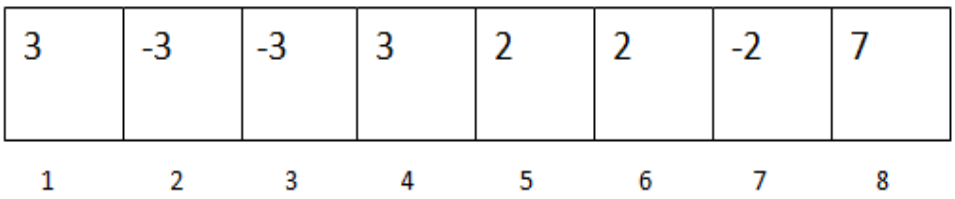
Union-Find: Consider the n singleton sets $\{1\}, \{2\}, \ldots, \{n\}$. Consider a sequence of Union and Find operations performed on these sets. Union(A, B) is the operation of computing the union of two sets named A and B. This operation is destructive, i.e., after this operation is perfumed, these two sets do not exist any more. Find(x) is the operation of finding the name of the set that the element x belongs to. An efficient data structure for performing these operations will find numerous applications. For example, Kruskal's algorithm for minimum spanning tree (MST) employs this data structure.

We can represent the sets using an array: $P[1:n]$.

Example:

Consider the sets $\{2, 5, 6\}\{1, 3, 4\}\{7, 8\}$. These sets can be represented as trees as shown below. All of these trees can be stored in one array $P[1:n]$.





In each node of the array, store the array index corresponding to the parent of the node. If the node is a root, then store the size of the tree * -1.

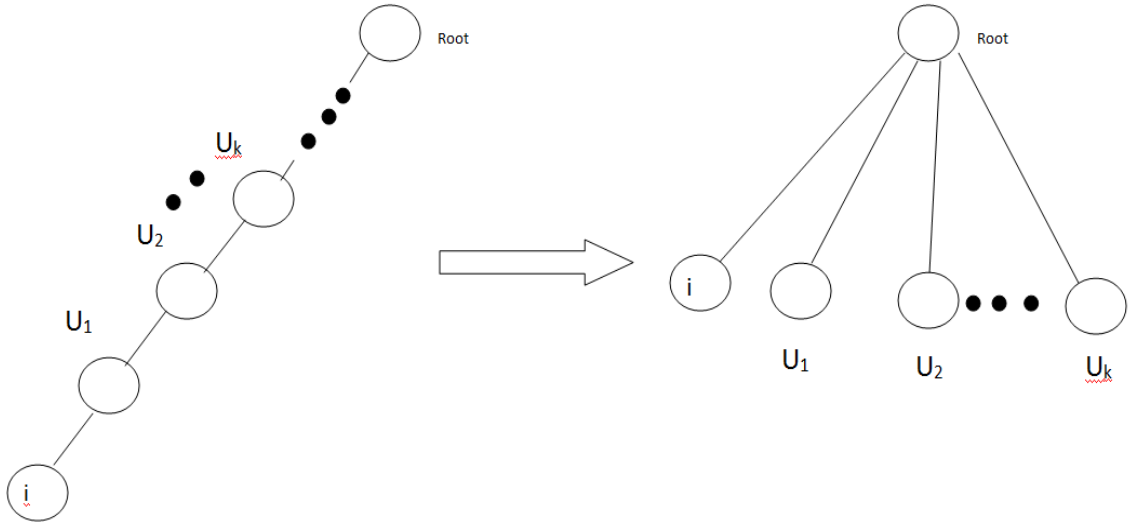
In this case, the following algorithms can be used for performing the Union and Find operations.

Weighted Union(i, j):
If $P[i] < P[j]$ /* i has more nodes than j */
then $\{P[i] := P[i] + P[j]; \quad P[j] := P[i];\}$

Find:

Trace the parent pointers.

Collapsing-Find(i): While tracing the parent pointers of the node i, let all the nodes along the way point directly to the root. Specifically, if $u_1, u_2, \ldots, u_k$ are the intermediate nodes between i and the root of the tree that the node i is in, then after identifying the root, let each of the nodes $u_1, u_2, \ldots, u_{k-1}$ and u_k point to the root.



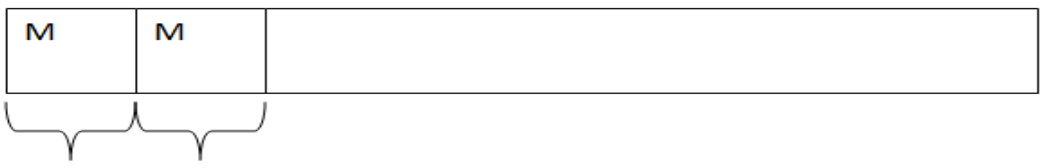
Theorem: Tarjan and van Leeuwen (1974)
Any sequence of m Union-Find operations (when $m > 2n$) will take $O(m\ \alpha(m))$ time if we use weighted union and collapsing find.
$\alpha(m) \rightarrow$ Inverse Ackermann's Function.

Kruskal's Algorithm: Let the input be $G(V, E)$. Let $|V| = n$ and $|E| = m$.

1. Sort the edges in nondescending order of edge weights. Let this sequence be $e_1, e_2, \ldots, e_m$.
2. Generate n trees with a single node in each tree. Let this forest be F.
3. for $1 \leq i \leq m$ do: throw the edge e_i into F if e_i does not cause a cycle.

Let $e_i = (a, b)$
We can check if (a, b) causes a cycle:
If $Find(a) = Find(b)$ then (a, b) causes a cycle
Runtime $= O(m\ \ log(m) + m\ \alpha(m))$

Out-of-core MST: Consider the case of $M = \Omega(n)$. Here is a simple MST algorithm due to (Dementiev, et al. 2004).
The edges are in the disk (in nonsorted order).



$F := \emptyset$;
Repeat:
Bring the next set S of edges; /*the size of S = M */
Compute the M.S.F of $S \cup F$;
$F := MSF(F \cup S)$;

Until all the edges have been processed.

Note: The above algorithm makes only one pass through the data. (Dementiev, et al. 2004) note that this algorithm does not perform well in practice.

Out-of-core MST - General Algorithm: There are several algorithms that do not make the above assumption that $M = \Omega(n)$. Many of these algorithms are based on Borůvka's algorithm.

Borůvka's Algorithm:
Let $G(V, E)$ be a given undirected weighted graph. For every node, identify the lightest edge. These edges belong to a MST. Let S be the set of these edges. Using only these edges find the connected components. Replace each connected component with a node and keep all the edges that go from one component to another. Edges within any component are eliminated. Let $G'(V', E')$ be the resultant graph. Recursively identify the MST T' of G'. Finally output the edges of S and the edges in T'. It is easy to see that the run time of this algorithm (in-core) is $O(|E| \log |V|)$ since in each iteration of the algorithm the number of nodes decreases by a factor of at least 2.

Example: In the example shown in Figure 1, a MST has the edges: $(1, 4), (4, 5), (5, 6), (1, 3), (2, 3), (2, 7)$.

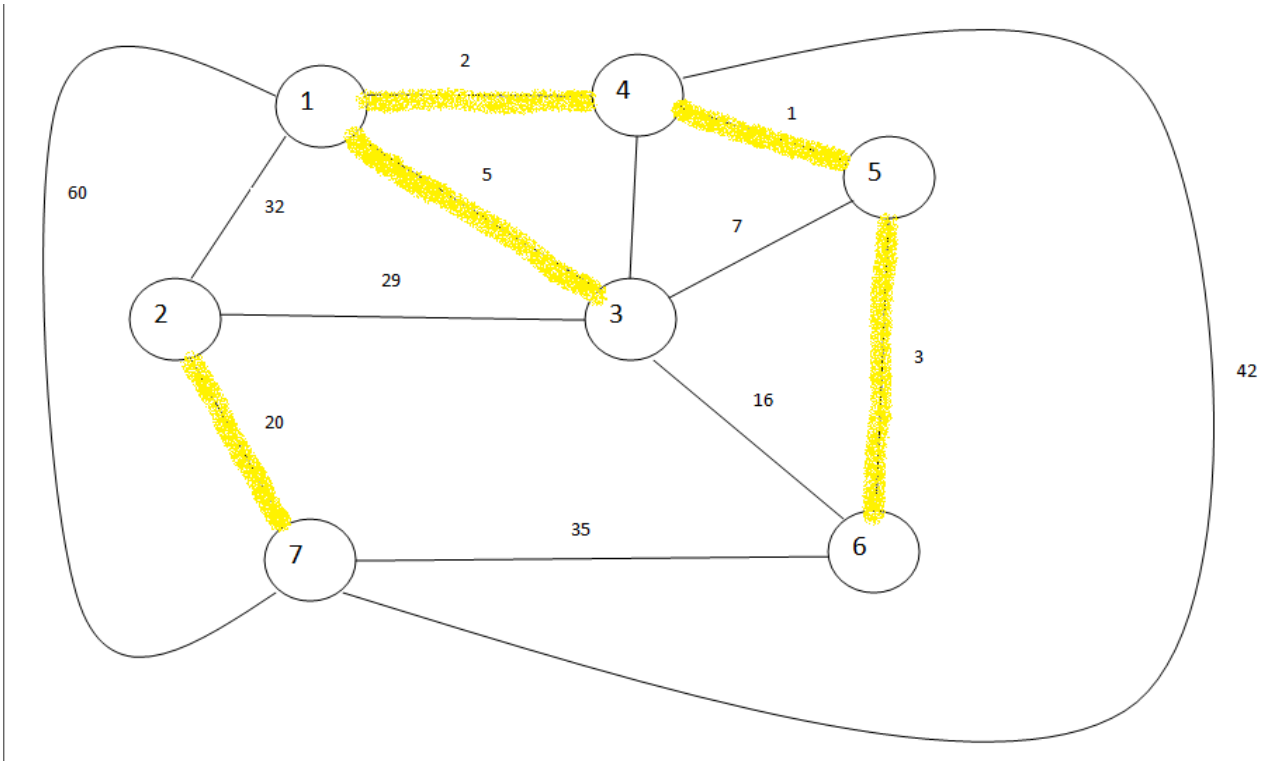
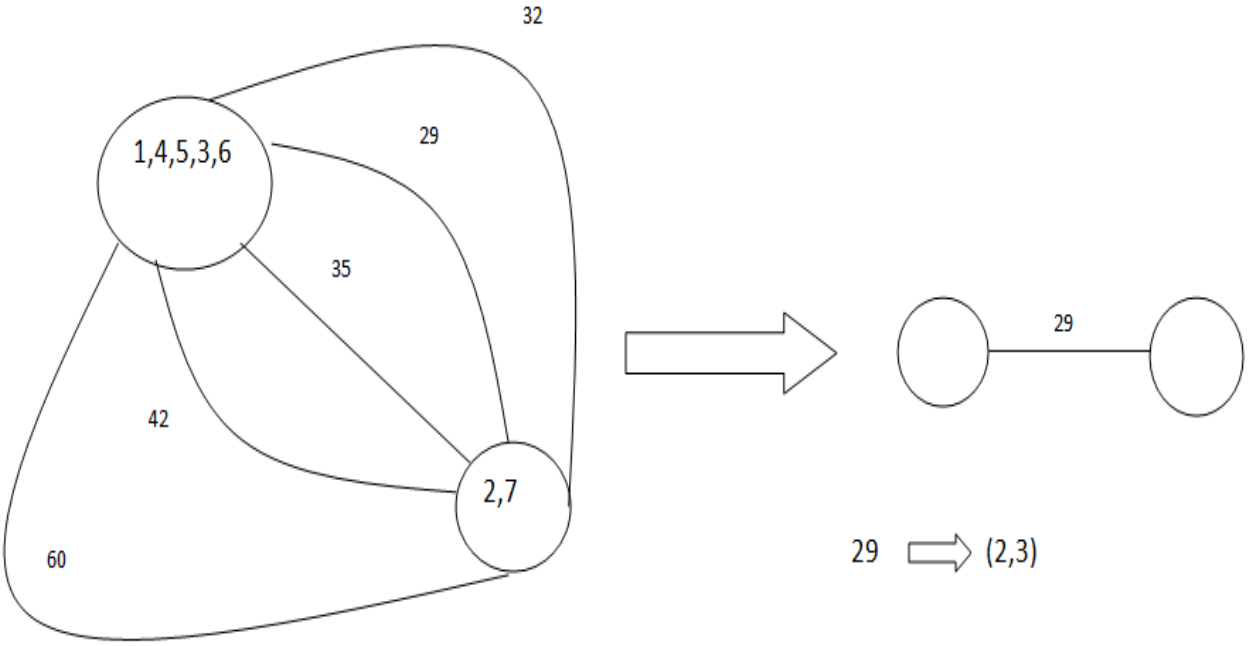


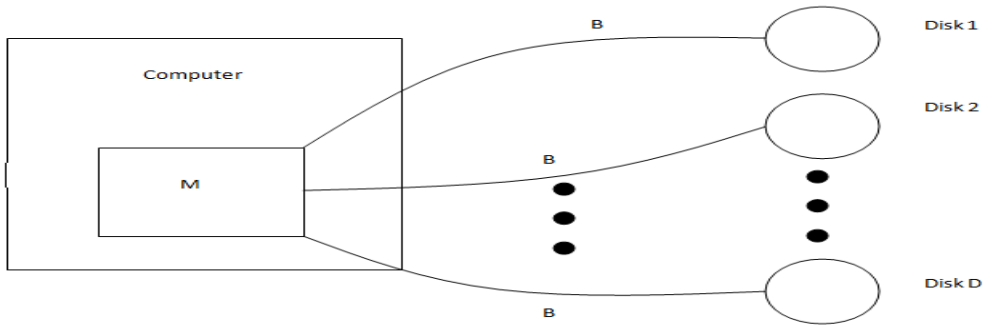
Figure 1: Borůvka's Algorithm – an example


32
1,4,5,3,6
29
35
42
2,7
60
29
29 (2,3)

(**Arge, et al., 2004**)
Each phase of Boruvka's algorithm can be done in $O(s(m))$ I/O operations, where $s(m)$ is the I/O complexity of sorting m elements.
$s(m) = O\left(\frac{m}{B}\frac{\log(m/B)}{\log(M/B)}\right)$
$m = |E|$ $M =$ memory size (core)
$\rightarrow$ MST can be done in $O(s(m)\log\frac{n}{M})$ I/O operations.

(Arge, et al. 2004) show that MST can be found in $O(s(m+n)\log\log(n/M))$ I/O operations. This algorithm is deterministic. An open question is if there is a deterministic algorithm that takes $O(s(m+n))$ I/O operations.
(**Abello, et al., 2002**) have implemented the optimal sequential randomized algorithm (of Klein, Tarjan, and Karger 1993) in the out-of-core model to get a randomized I/O complexity of $O(s(m+n))$.

Parallel Disks Model (PDM) :



In one parallel I/O, we can bring one block of data from each disk.

Theorem:
Sorting of n keys will need $\Omega(\frac{N}{DB}\frac{log(N/B)}{log(M/B)})$ I/O operations.

Note: In the PDM model, we normally assume that $M = \Theta(BD)$.