K^{th} largest / k largest elements

Input: $A[1..n]$ Output: k largest elements of A.

Naive: (i) Sort A and output $A[n-k+1..n]$

(ii) Build a max heap from A, output k items by executing DeleteMin k times.

Neither algorithm is feasible is n is very large, $A[1..n]$ is a stream.
- External K^{th} largest element problem.

Algorithm idea:

Create a min heap of size k that holds the k largest elements seen so far.

When processing the next element $A[i]$:

if $A[i] >$ min element in heap then

DeleteMin()

Insert(A[i]).

Implementation:

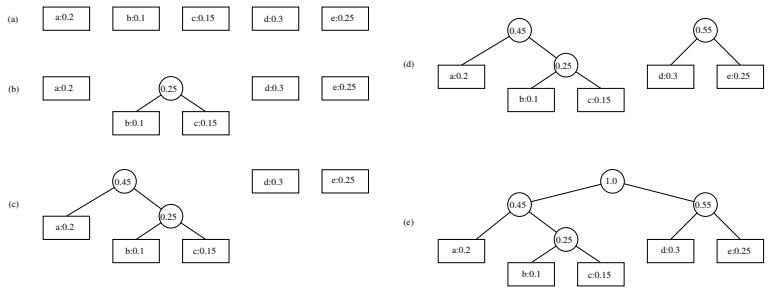
Heap $h[1..K]$ that stores K largest elements - BuildHeap with $A[1..k]$

for $i \leftarrow k+1$ to n do { if $A[i] > h[1]$ then

$h[1] \leftarrow A[i]$; percolateDown(1) }

Illustration of Huffman coding algorithm

Characters $\{a, b, c, d, e\}$ with frequencies $\{0.2, 0.1, 0.15, 0.3, 0.25\}$



Encoding: $\{a = 00, b = 010, c = 011, d = 10, e = 11\}$.
Average character encoding size is
$0.2 * 2 + 0.1 * 3 + 0.15 * 3 + 0.3 * 2 + 0.25 * 2 = 2.25$.

Kruskal's algorithm

for each $u \in V$ **do** $MakeSet(u)$
Let F be an empty forest; $wmst \leftarrow 0$
Sort edges by non-decreasing order of weight
for each edge $e = (u, v)$ in this sorted order **do**
    $ru \leftarrow Find(u)$ // representative of u's component
    $rv \leftarrow Find(v)$ // representative of w's component
    if $ru \neq rv$ **then**
        // u and v are in different components
        $F \leftarrow F \cup \{e\}$; $wmst \leftarrow wmst + w(u, v)$
        $Union(ru, rv)$ // Merge the two components
Output MST F and its weight $wmst$

Disjoint-set data structures

A new abstract data type: collection of disjoint subsets of a given set V that support the following operations. Each set is represented by (any) one of its elements.

1. Make-Set(x): create a new set whose only member (and representative) is x.
2. Find(x): find the representative of the set containing x. If x and y are in the same set, then Find(x) = Find(y). Otherwise Find(x) $\neq$ Find(y).
3. Union(x, y): Replace the sets S_x and S_y represented by x and y respectively, by $S_x \cup S_y$, and return a representative of the newly created set. Union should be called only with representatives of sets (this operation is called Link in the textbook).

Implementation of operations

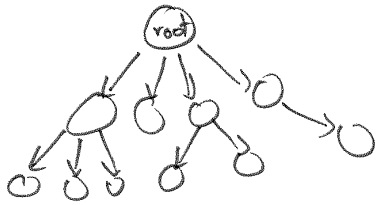
```
Make-Set (x)
{ x.p := x
  x.rank := 0
}
Find (x)
{ if x != x.p then
     x.p := Find (x.p)
  return x.p
}
```

```
Union (x, y)
{ if x.rank > y.rank then
     y.p := x
  else if y.rank > x.rank then
     x.p := y
  else  // x.rank = y.rank
     x.p := y
     y.rank ++
}
```

MST in directed graphs (Edmond's branching algorithm)

- Greedy algorithm does not work in directed graphs
- In directed graphs: rooted tree < outgoing tree / incoming tree

Out-tree: Acyclic subgraph in which:
- root node has no incoming edges
- path from root to every node
- all non-root nodes have exactly one incoming edge.



← Minimum weight rooted spanning tree = Branching

Problem: Input: Directed graph $G=(V,E)$, $w: E \to \mathbb{Z}^+$, root $r \in V$
Output: Spanning tree $T=(V,E')$, $E' \subseteq E$, rooted at r, of min weight.

Theorem 1:
(i) Consider any $u \in V$, $u \neq r$
Suppose we decrease the weight of all edges into u by Δ.
Then the weight of MST decreases by Δ.

Proof: Consider a spanning tree T rooted at r.
T has exactly one edge into u.
T's weight decreases by Δ.
Transformation: New weight (T) = Old Weight$(T) - \Delta$.
MST under old weight are MST with new weights also.

Theorem 2: If G has a rooted spanning tree (rooted at r) of weight 0, then it is an MST.
Proof: All edges have nonnegative weight.
$w(T) = 0 \Rightarrow T$ is MST.

Algorithm to find an MST in directed graphs

Input: Directed graph $G = (V, E)$, nonnegative weight function W on its edges, root $r \in V$.
Output: Directed tree rooted at s (outgoing tree), of minimum weight.

1. Transform weights so that every node except r has an incoming edge of weight 0:

   for $u \in V - \{r\}$ **do**
   Let d_u be the weight of a minimum weight edge into u.
   for $(p, u) \in E$ **do**
   $w(p, u) \leftarrow w(p, u) - d_u$

   Reduction in weight of MST by above transformation $= \sum d_u$, where $u \in V - r$.

2. Let $G_0 = (V, Z)$ be the subgraph of G containing all edges of 0-weight, i.e., $Z = \{e \in E : w(e) = 0\}$. Run DFS/BFS in G_0, from r. Note that we are using only edges of G with 0-weight. If all nodes of V are reached from r, then return this DFS/BFS tree as MST.

3. If there is no spanning tree rooted at r in G_0, then there is a 0-weight cycle. Find a 0-weight cycle as follows:

   (a) Find a node z that is not reachable from r in G_0, in the above search.

   (b) Walk backward from z in G_0, using incoming edges at each node visited. Every node except r has a 0-edge coming into it, and so this walk can keep going forever. Since r has no path to z using 0-edges, this walk will never get to r. There are only a finite number of nodes. So, some node x will be repeated on this walk. The path from x to itself on this walk is composed of 0-weight edges, and this gives a 0-weight cycle C.

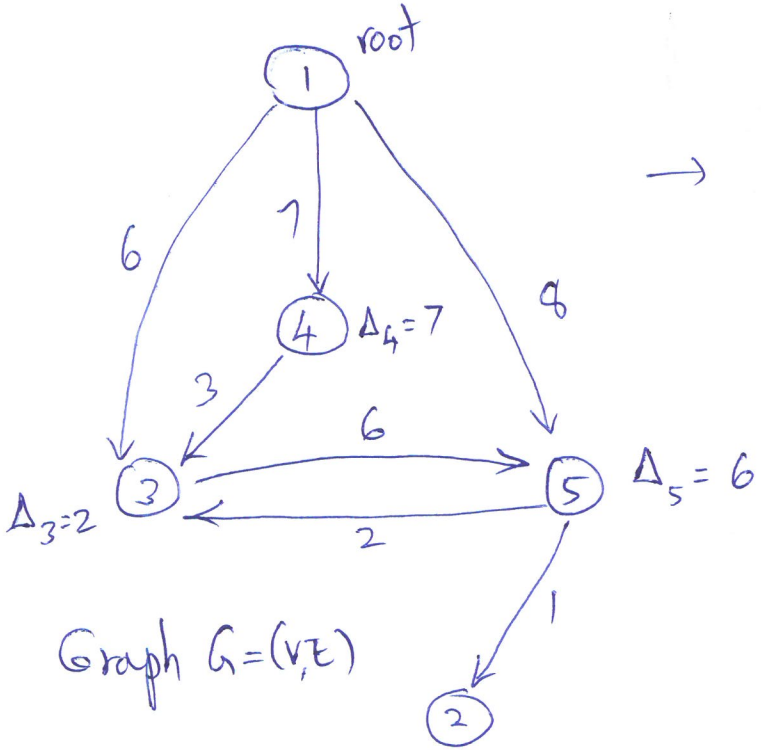
4. Shrink cycle C into a single node c. There may be many edges from the nodes of C to a node u outside the cycle. These are replaced by a single edge. For each edge (a, u) in G, with $a \in C$ and $u \notin C$, introduce the edge (c, u) of weight $w(a, u)$.

   Similarly, for edges of G that are going into C, do the following. For each edge (u, a) in G, with $u \notin C$ and $a \in C$, introduce the edge (u, c) of weight $w(u, a)$.

   If there are multiple edges (c, u), keep just one edge with minimum weight, and record the corresponding edge of G. Similarly, process multiple edges (u, c) by replacing each multi-edge by a single edge of minimum weight.

   The new graph has fewer nodes than the original graph, and the MSTs of the two graphs have equal weight.

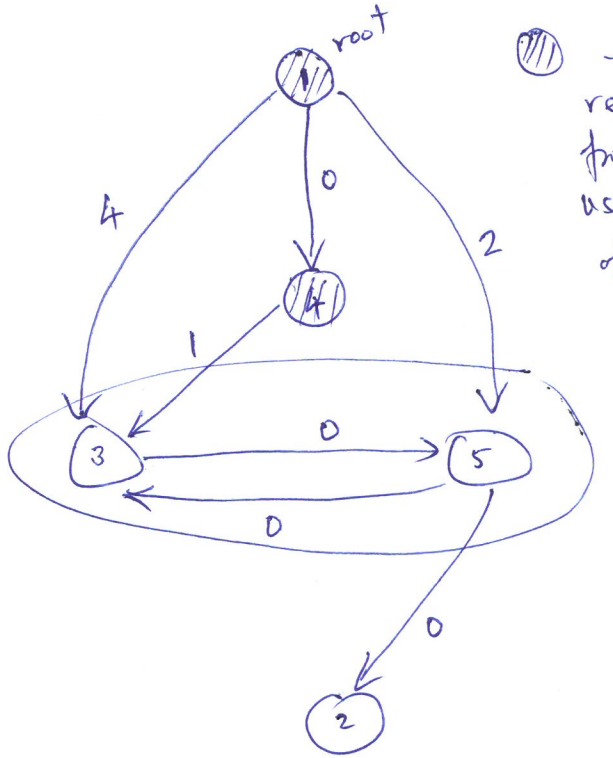
5. Recursively find an MST of the smaller graph. This MST has exactly one edge into c, and this edge corresponds to some actual edge (u, a) in the graph before shrinking, where $a \in C$. Now, expand node c, and include the edges of the 0-weight cycle C. Since the total weight of the cycle is 0, adding it to the MST does not increase its weight. But node a will have 2 incoming edges: edge (u, a) from the MST, and one edge from the cycle. Delete the edge coming into node a in the cycle, to get an MST of the original graph. Return this MST.



Graph $G = (V, E)$

$\Delta_2 = 1$

$\sum \Delta_u = 1 + 2 + 7 + 6 = 16$

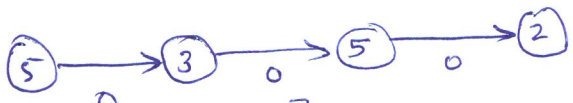


- nodes reachable from root using edges of 0 weight

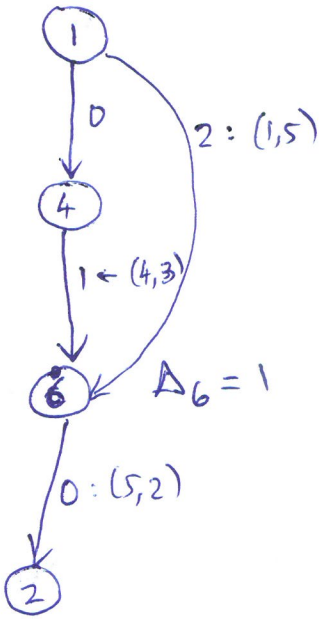
shrink C

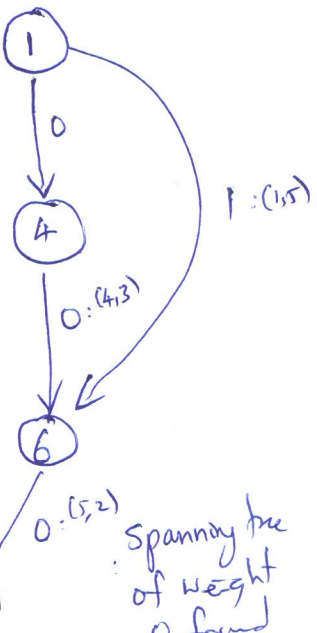
C = Node 6

z = 2 is not reachable from root using 0-edges.



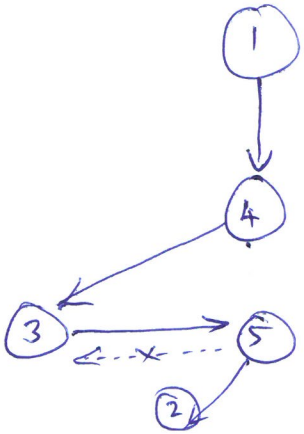
Node repeats – cycle found



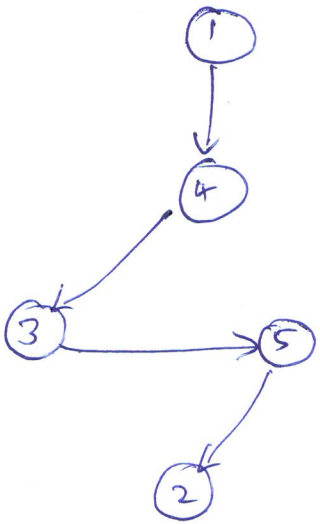


Spanning tree of weight 0 found

Expand 6



Output



MST of G

Weight = $16 + 1 = 17$

$\sum \Delta_u$ (2,3,4,5) + Δ_6