Chapter 5 Greedy algorithms

Joe Song

October 24, 2019

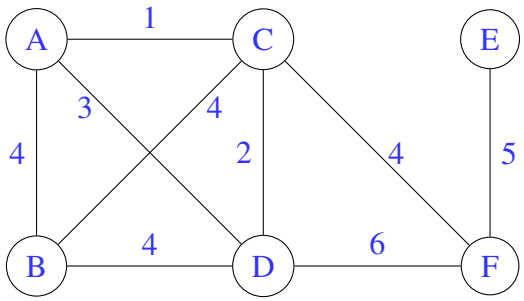
The notes are based on Chapter 5 of Dasgupta, Papadimitriou and Vazirani. Algorithms. 2008. McGraw-Hill. New York.

Contents

1 Minimum spanning trees

Tree ≡ Undirected, acyclic, and connected graph

How to connect the nodes in the graph with the least total sum of edge weights?



Property 1 *Removing a cycle edge cannot disconnect a graph.*

This is because two nodes on a cycle would still be connected in an undirected graph if only one edge on the cycle is removed.

Properties of trees:

- **Property 2** A tree on n nodes has $n-1$ edges.

  Proof:

  1. Start with nodes as n connected components without any edges;
  2. Pick an edge and merge two connected components. The number of components is reduced by 1. No other edges exist between the two components.
  3. Repeat this procedure exactly $n-1$ time and all edges are picked exactly once. Therefore the total number of edges is $n-1$.

- **Property 3** Any connected, undirected graph $G=(V,E)$ with $|E|=|V|-1$ is a tree.

  Proof:

  1. If there is a cycle in the graph, we can remove an edge on the cycle. The remaining graph would be connected.
  2. We will get a tree if we repeat step 1. That tree has $n-1$ edges.
  3. As the original graph has $n-1$ edges, it has to be exactly the same tree obtained above.

- **Property 4** An undirected graph is a tree if and only if there is a unique path between any pair of nodes.

Proof.

("Only if") Tree $\rightarrow$ unique path: If there are more than one paths between two nodes, we would have cycles in the graph. The graph would not be a tree. A contradiction.

("if") Unique path $\rightarrow$ tree: A unique path between any pair of nodes suggests the graph has no cycles and is connected. Thus it is a tree.

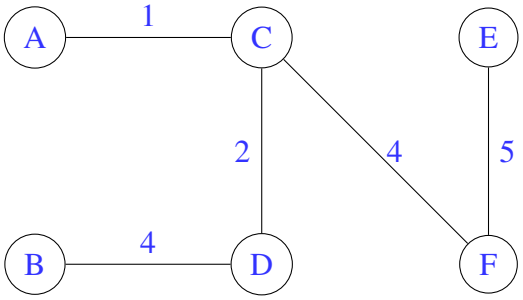
Definition of *minimum spanning tree* (MST):

Input: An undirected graph $G = (V, E)$; edge weights w_e.

Output: A tree $T = (V, E')$, with $E' \subset E$, that minimizes

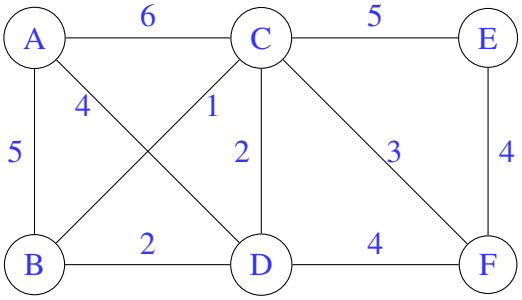
$$weight(T) = \sum_{e \in E'} w_e$$

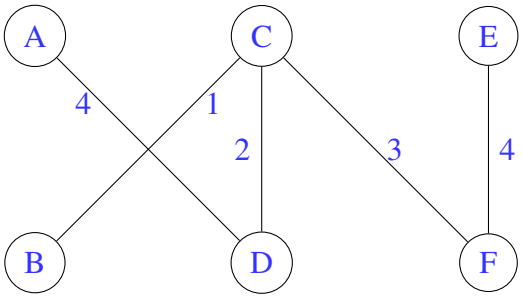
One minimum spanning tree of a cost of 16:



1.1 A greedy approach

Repeatedly add the next lightest edge that doesn't produce a cycle.

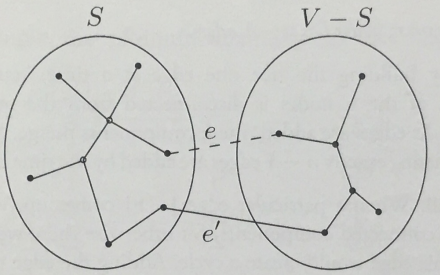




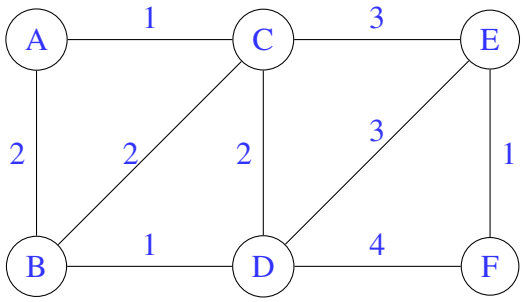
1.2 The cut property

Cut Property *Suppose edges X are part of a minimum spanning tree of $G = (V, E)$. Pick any subset of nodes S for which X does not cross between S and $V - S$, and let e be the lightest edge across this partition. Then $X \cup \{e\}$ is part of some MST.*

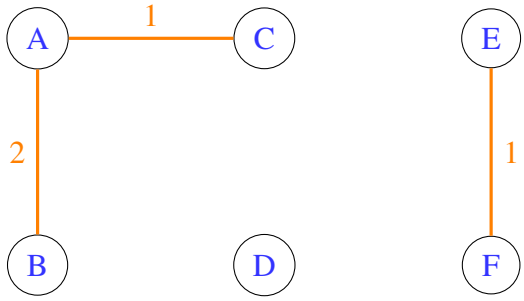
Figure 5.2 $T \cup \{e\}$. The addition of e (dotted) to T (solid lines) produces a cycle. This cycle must contain at least one other edge, shown here as e', across the cut $(S, V - S)$.



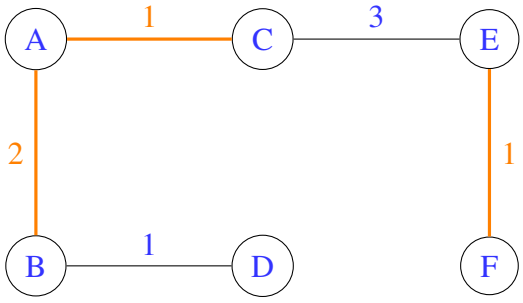
Input graph:



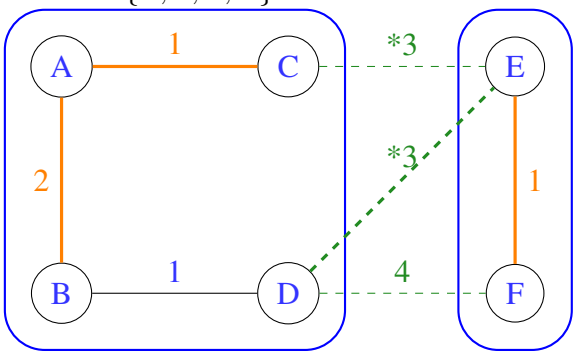
A set X of three edges part of a MST T:



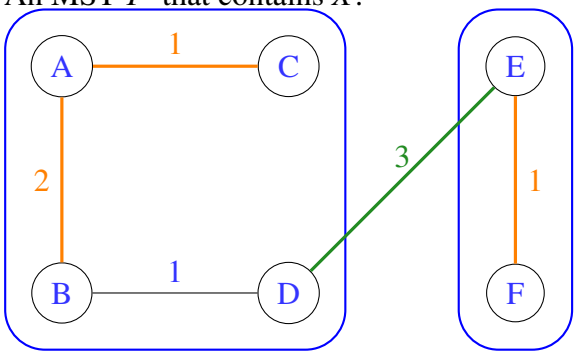
An MST T that contains X:



Pick $S = \{A, B, C, D\}$. The "cut":



An MST T' that contains X:



Proof. If e is already on some known MST, we are done.

If e is not on a known MST, then there must another cross edge e' on an MST T', such that $w(e') \geq w(e)$.

Replacing e' by e leads to a tree T whose total weight is

$$w(T) = w(T') - w(e') + w(e) \leq w(T')$$

As T' is a already an MST, it must be that $w(T) = w(T')$, i.e., T is another MST. □

1.3 Kruskal's algorithm

---

procedure Kruskal(G, w)
Input: A connected undirected graph $G = (V, E)$ with edge weights w_e
Output: A minimum spanning tree defined by the edges X

for all $u \in V$:
    makeset(u)

$X = \{\}$
sort the edges in E by weight
for all edges $(u, v) \in E$, in increasing order of weight:
    if find$(u) \neq$ find(v):
        add edge (u, v) to X
        union(u, v)

Run time:

- $|V|$ make-set() operation
- $2|E|$ find() operations
- $|V|-1$ union() operations, because we add exactly $|V|-1$ edges to X.

1.4 A data structure for disjoint sets

We use a *directed tree* to represent a set. Each node x in the tree has a

- $\pi(x)$: pointer to the parent node of x

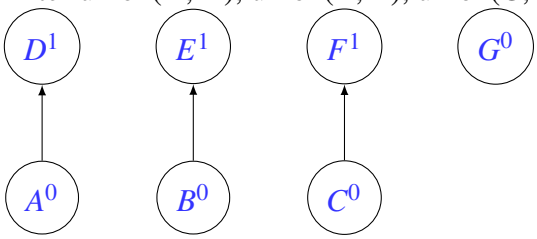
  If x is a root node, $\pi(x) = x$.

- rank(x): the height of the subtree rooted at x

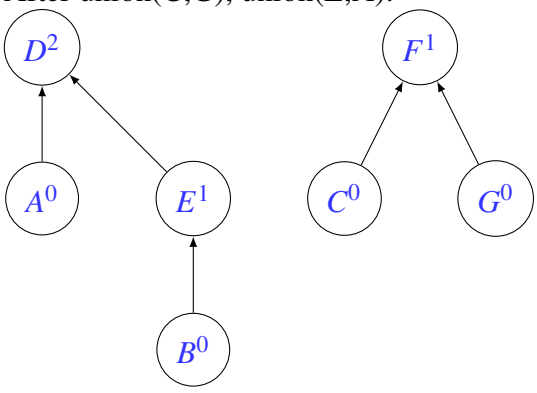
The *name* of the set or tree is the root node of the tree.

After makeset(A), ..., makeset(G):

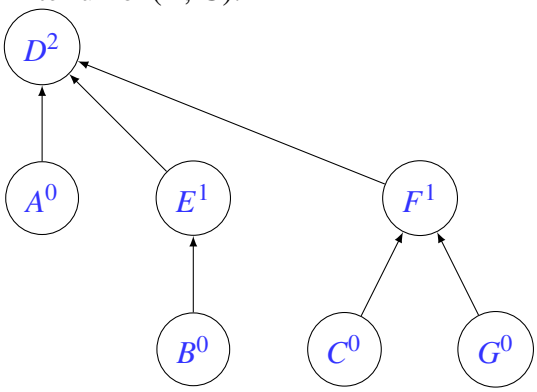
After union(A, D), union(B, E), union(C, F):



After union(C,G), union(E,A):



After union(B, G):



---

procedure makeset(x)
$\pi(x) = x$
rank(x) $= 0$

---

procedure find(x)
while $x \neq \pi(x)$: $x = \pi(x)$
return x

---

procedure union(x, y)
r_x = find(x)
r_y = find(y)
if $r_x = r_y$: return
if rank(r_x) $>$ rank(r_y):
  $\pi(r_y) = r_x$
else:
  $\pi(r_x) = r_y$
  if rank(r_x) == rank(r_y): rank(r_y) = rank(r_y) + 1

Properties of disjoint sets:

Property 1 For any $x \neq \pi(x)$, $\text{rank}(x) < \text{rank}(\pi(x))$.

By definition of rank.

Property 2 Any root node of rank k has at least 2^k nodes in the tree.

By induction. A rank k node is created by merging two trees of rank $k-1$.

Property 3 If there are n elements overall, there can be at most $n/2^k$ nodes of rank k.

From Property 2, # nodes under a rank k node $\geq 2^k$. Therefore, if the total number of nodes is n, # rank k nodes $\leq n/2^k$.

Property 3 suggests that the tree can have at most one $k = \log n$ node, i.e., the height of the tree can not be more than $\log n$. Therefore, we can determine the run time for each operation as follows.

Run time:

- make-set() takes constant time and we do it n times
- find() takes $\log n$ time (the maximum height of a tree). We do it at most $2|E|$ times in the Kruskal's algorithm.
- union() is $O(\log n)$ (two find() operations). We do it at most $|V|-1$ times in the Kruskal's algorithm.
- sort all edges in the graph takes $O(|E|\log|E|)$, approximately $O(|E|\log|V|)$, if the number of edges is a polynomial function of the number of nodes in the graph.
- This implies that the Kruskal's algorithm has a runtime of $O((|E|+|V|)\log|V|)$.

1.5 Prim's algorithm

$$$ Idea: Use the subtree formed by X to be S and all other nodes not explored as $V-S$. Include node one by one to X, based on the *cut property*.

---

procedure Prim(G, w)
Input: A connected undirected graph $G = (V, E)$ with edge weights w_e
Output: A minimum spanning tree defined by the array prev

for all $u \in V$:
    cost(u) = ∞
    prev(u) = nil

pick any initial node u_0
cost(u_0)=0

H=make-queue(V) (priority queue, using cost-values as keys)
while H is not empty:
    v = delete-min(H)
    for each $(v,z) \in E$:
        if cost(z) $> w(v,z)$:
            cost(z) $= w(v,z)$
            decrease-key(H,z)
            prev(z) $= v$

Run time:

- $|V|$ insertion in the make-queue() operation
- $|V|$ delete-min() operations
- $|E|$ decrease-key() operations

If we use a binary heap for the priority queue, the Prim's algorithm will have $O(|V| + |E|) \log |V|)$ worst case run time.

2 Huffman coding

Example: MP3 audio compression:

1. Sample at 44,100 Hz. 50-minute symphony will generate $50 \times 60 \times 44,100 \approx 130$ million samples
2. Quantize to discrete values (in an alphabet) that are close to the original
3. Encode the discrete values in binary and
4. Compress the binary number

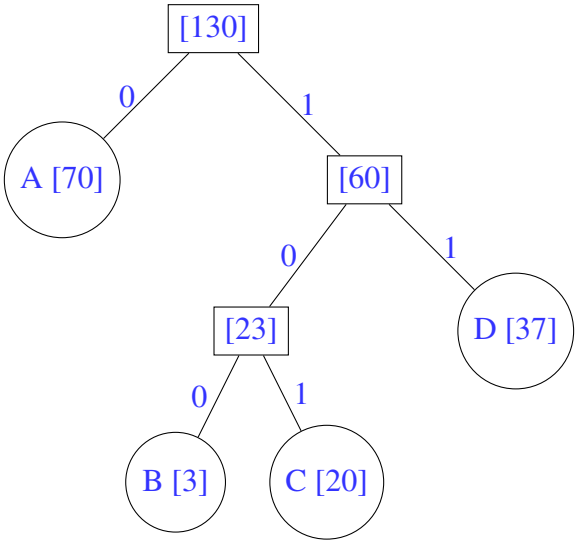
Assume the alphabet has only four symbols A,B,C,D. Use binary 00, 01, 10, 11 coding, we get 260 megabits.

If we know the frequency is A – 70 million, B – 3 million, C – 20 million, D – 37 million, then we can consider using different number of bits for each symbol.

However, $\{0, 01, 11, 001\}$ does not work because of its ambiguity in decoding 001.

Instead, we can use a *full* binary tree which satisfies the *prefix-free* property:

Prefix-free property: no codeword can be a prefix of another word.



$\Longrightarrow$

Symbol	Codeword
A	0
B	100
C	101
D	11

Here the Huffman code has a total cost of

$$1 \times 70 + 3 \times 3 + 3 \times 20 + 2 \times 37 = 70 + 60 + 23 + 37 + 3 + 20 = 213 \text{ megabits}$$

saving $260 - 213 = 47$ megabits.

Thus in general, we have

$$\begin{aligned} &\text{cost of tree} && (1)\\ &= \sum_{i=1}^{n} f_i \times \text{depth of } i\text{th symbol in tree} && (2)\\ &= \sum \text{frequencies of all leaves (ovals) and internal nodes (squares) except the root} && (3) \end{aligned}$$

$$$ Idea: Create a tree that is corresponding to the frequencies of each symbol using a greedy strategy: use smallest frequencies as leave nodes, merge them, and then repeat.

---

procedure Huffman(f)
Input: An array $f[1, \ldots, n]$ of frequencies

Output: An encoding tree with n leaves

let H be a priority queue of integers, ordered by f
for $i = 1$ to n: insert(H, i)
for $k = n+1$ to $2n-1$:
    i=delete-min(H), j=delete-min(H)
    create a node numbered k with children i, j
    $f[k] = f[i] + f[j]$
    insert(H, k)

Correctness: We argue that the bottom of an optimal Huffman coding tree must include a pair of nodes with the least frequencies. If such nodes are not at the bottom, one can always switch them to the bottom to have a tree with lower cost, which contradicts with the original tree being optimal.

Running time: $O(n \log n)$ if a binary heap is used.