Math 776 Graph Theory Lecture Note 6
Kruskal's algorithm, Dijksta's algorithm, and Huffman algorithm

Lectured by Lincoln Lu
Transcribed by Lincoln Lu

1 Minimum spanning trees and Kruskal's algorithm

Definition 1 *A* **weighted** *graph* (G, w) *is a graph* G *together with a map* $w\colon E(G) \to \mathbb{R}^+ = [0, \infty)$.

For any edge e, $w(e)$ is called the weight of e. For any subgraph H, the total weight of H is

$$w(H) = \sum_{e \in E(H)} w(e).$$

In particular, we are interested in a spanning tree with minimum total weight. Such a tree is called a minimum spanning tree. Given a weighted graph (G, w), the Kruskal's algorithm produces a minimum spanning tree of G as follows.
Kruskal's Algorithm:

```
Kruskal(G, w){
        //input: a weighed graph (G, w).
        //output: a minimum spanning tree.
        //H: (temperate variable) an acyclic spanning subgraph.

        E(H) = ∅
        Sort edges of G according to their weights.
        while( |E(H)| < |V(G)| − 1){
                Find the next cheapest edge e joins two components of H.
                E(H) = E(H) ∪ {e}.
        }
        return H;
}
```

Theorem 1 (Kruskal 1956) *In a connected weighted graph* (G, w)*, Kruskal's algorithm constructs a minimum spanning tree.*

Proof: Let $n = |V(G)|$. Initially, H consists of n isolated vertices. At each iteration, the number of edges of H increases by one, while the number of connected components of H decreases by one. When the program exits the "while" loop, H has $n - 1$ edges and 1 connected component. Thus H is a spanning tree.

Let T^* be the resulting tree. We will prove $w(T^*) \leq w(T)$ for any spanning tree T.

For any tree T, let $m(T)$ be the number of edges in T that is not in T^*. We will inductively prove that $w(T^*) \leq w(T)$ for all T satisfying $m(T) \geq 0$.

If $m(T) = 0$, we have $T = T^*$. Thus, $w(T) = w(T^*)$, and we are done.

We assume $w(T^*) \leq w(T)$ for all trees T with $m(T) = k$.

For any tree T with $m(T) = k + 1$, let e be the first edge chosen for T^* that is not in T. Adding e to T creates a cycle C. Let e' be an edge in C but not in T^*. From the choice of edge e, we have

$$w(e) \leq w(e').$$

We note that $T + e - e'$ is a spanning tree with

$$m(T + e - e') = m(T) - 1 = k.$$

By inductive hypothesis, we have

$$\begin{aligned} w(T^*) &\leq w(T + e - e') \\ &\leq w(T) + w(e) - w(e') \\ &\leq w(T). \end{aligned}$$

The inductive step is completed.

Therefore, T^* is a minimum spanning tree. □

Worst-case running time analysis:

Let $V = |V(G)|$ and $E = |E(G)|$. The running time of Kruskal's algorithm for a graph G depends on the implementation of disjoint-set data structure. There is an implementation of the disjoint-set data structure such that the union and query takes only $o(\log E)$.

Initialization takes time $O(V)$, and the time to sort the edges is $O(E \log E)$. There is $O(E)$ operations of query and union on the disjoint-set structure, which in total take $O(E \log E)$ time. The total running time for Kruskal's algorithm is $O(E \log E)$.

2 Shortest distances and Dijkstra's algorithm

The distance $d(u, v)$ between two vertices u and v in a graph G is the minimum number of edges among all u, v-paths.

For a weighted graph (G, w) and two vertices u, v, the distance $d(u, v)$ is the minimum weight $w(P)$ among all u, v-paths.

Dijkstra's algorithm calculates distances from one vertex.

Dijkstra's Algorithm:

Dijkstra(G, w, u)\{
    //input: a weighed graph (G, w), a starting vertex u.
    //output: the distance $d(u, v)$ for all v.

```
        //S: (temperate variable) the set of vertices to which a shortest path from u is known.
        //t(u):tentative distance from u to v.
        //initializtion:
        S = {u}
        Extend weight function to non-edges. w(xy) = ∞ for any non-edge xy.
        t(u) = 0, t(z) = w(uz) for z ≠ u.

        while( S ≠ V(G)){
                Select a vertex v outside S with minimum value t(v).
                For all z ∉ S, t(z) = min{t(z), t(v) + w(vz)}.
        }
        For all v, let d(u, v) = t(v).
        Return d(u, v).
}
```

Theorem 2 *Given a weighted (di)graph (G, w) and a vertex $u \in V(G)$, Dijkstra's algorithm computes $d(u, z)$ for each $z \in V(G)$.*

Proof: Let S_k be the set S at k-th iteration, and $t_k(z)$ be value of $t(z)$ at k-th iteration. We will prove the following claim:
Claim: For $k = 1, 2, \ldots$, we have

1. for $z \in S_k$, $t_k(z) = d(u, z)$, and
2. for $z \notin S_k$, $t_k(z)$ is the least length of a u, z-path reaching z directly from S_k.

We use induction on k. Initial step: $k = 1$. We have $S_1 = \{u\}$, $t_1(u) = 0 = d(u, u)$, and $t_1(z) = w(uz)$. The claim holds.

Suppose that the claim holds for k. So we have $t_k(z) = d(u, z)$ for $z \in S_k$, and $t_k(z)$ is the least length of a u, z-path reaching z directly from S_k.

Now we consider a set $S_{k+1} = S_k \cup \{v\}$. Here v is the last vertex added into S_k. We have $t_k(v) \leq t_k(z)$, for any $z \notin S_k$.

If $z \in S_{k+1}$ and $z \neq v$, we have $t_{k+1}(z) = t_k(z) = d(u, z)$, by inductive hypothesis.

If $z = v$, $t_{k+1}(v) = t_k(v)$. We need show $t_k(v) = d(u, v)$. On one hand, we consider a shortest u, v-path P with weight $d(u, v)$. Suppose z be the first vertex on P not in S_k. Let P_{uz} be the sub-path of P from u to z. We have

$$d(u, v) = w(P) \geq w(P_{uz}) \geq t_k(z) \geq t_k(v).$$

On the other hand, $t_k(v) \geq d(u, v)$ by definitions of $t_k(v)$ and $d(u, v)$. Together, we have

$$d(u, v) = t_k(v).$$

For any $z \notin S_{k+1}$, we consider a minimum u, z-path P reaching z directly from S_{k+1}. If P doesn't contain v, P is a u, z-path reaching z directly from S'. Thus, $w(P) = t_k(z)$. If P contains v, then $w(p) = t_k(v) + w(vz)$. Thus, $w(P) = \min\{t_k(z), t_k(v) + w(vz)\}$. We finish the inductive step.

Worst-case running time analysis:

Let $V = |V(G)|$ and $E = |E(G)|$. Initialization takes time $O(V^2)$. There are V iterations. Each iteration takes time $O(V)$. The total running time for Dijkstra's algorithm is $O(V^2)$. □

3 Binary trees and Huffman's algorithm

A *rooted tree* is a tree with one vertex r chosen as *root*. For each vertex v, let $P(v)$ be the unique v, r-path. The *depth* of v is the length of path $P(v)$. The *height* of a rooted tree T is the largest depth of any vertex in T. The vertices of $P(v) \setminus \{v\}$ are the ancestors of v. The parent of v is the only neighbor of v which is on $P(v)$. The children of v are other neighbors of v other than its parent.

A *binary* tree is a tree where each vertex has at most two children, and each child of a vertex is designated as its left child and right child.

Binary trees permit storage of data for quick access. We store one item at a leaf and access it by following the path from the root. Suppose n items have access probabilities/frequencies $p_1, p_2, \ldots, p_n$. The expect access time is $l(T) = \sum_{i=1}^{n} p_i l_i$, where l_i is the depth of the leaf where i-th items are stored.

Given access probabilities among n items, we want to place them at leaves of a binary tree such that the expect access time is minimized.

A path in a binary tree can be encoded by a binary string with 0 when we move to a left child and 1 when we move to right child. These binary strings are prefix-free, since no two of items are stored on two vertices from a path from the root. The binary tree with items stored on leaves gives a set of prefix-free binary codes for these items. The expected access time is proportional to the length of encoded messages.

Huffman's Algorithm:

Huffman(C){
    //input: C contains n nodes with associated frequencies $p_1, p_2, \ldots, p_n$.
    //output: a binary tree with minimum expected access time.

    if(C has one node){
        return C as a binary tree with single node.
    }
    else {
        Find two nodes u and v with least frequencies, say p_{n-1} and p_n.
        Let $q_{n-1} = p_{n-1} + p_n$. Create a new node w with frequency q_{n-1}.
        Let $C' = C \setminus \{u, v\} \cup \{q_{n-1}\}$.
        Let T' = Huffman(C').
        Attach nodes u and v as children of w.
        Let T be the result binary tree.
    }
    return T;
}

Example 1 *Consider 6 items with frequencies* $2, 3, 10, 20, 25, 40$. *The output of Huffman's algorithm is the following binary tree (see Figure* **??***). The corresponding Huffman codes are given as follows.*

$$11000, 11001, 1101, 111, 10, 0.$$

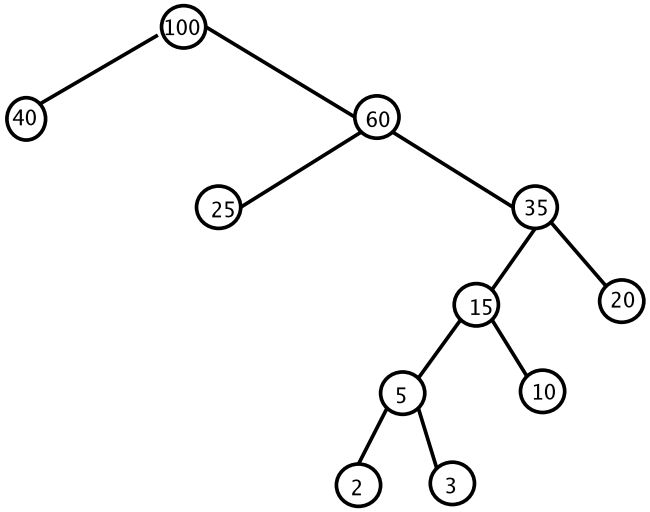

Figure 1: An example of Huffman coding.

Theorem 3 (Huffman) *The Huffman's algorithm computes correctly the prefix-free codings with minimum expected code length.*

Proof: Let C be the set of frequencies and $f(C)$ be the minimum access time among any binary trees with the set C as leaves. Let $T(C)$ be the tree constructed by Huffman's algorithm. The theorem says $l(T(C)) = f(C)$. We use induction on the number of items.

It is trivial when $n = 1$ since there is only one tree.

Suppose that $l(T(C)) = f(C)$ for any C with n nodes.

For a set C of $n + 1$ nodes, let u be the node with least frequency and v be the node with second least frequency in C. Let T be an optimized binary tree with leaf set C. We observe that

1. Except for the root, all non-leaf nodes have two children.

2. u and v have the maximum depth.

If there is an non-leaf node x with only one child node y. Delete the node x and connect y to the parent node of x. It is clearly that the new tree will have smaller access time. Contradiction.

Now let x be the node of T with maximum depth. Since the parent of x have two children. The other child of x's parent is also a leaf. We denote it by y. We have $l(x) = l(y) \geq \max\{l(u), l(v)\}$. We also assume that $p(y) \geq p(x)$. In particular, we have $p(u) \geq p(x)$ and $p(v) \geq p(y)$. Exchange node x with u and node y with v. Let T' be the resulting tree. We have

$$\begin{aligned} l(T') &= l(T) - p(u)l(u) - p(v)l(v) - p(x)l(x) - p(y)l(y) \\ &\quad + p(u)l(x) + p(v)l(y) + p(x)l(u) + p(y)l(v) \\ &= l(T) - (p(x) - p(u))(l(x) - l(u)) - (p(y) - p(v))(l(y) - l(v)) \\ &\leq l(T) \\ &\leq f(C). \end{aligned}$$

Thus, T' is also an optimized tree. Without loss of generality, we can assume that u and v are siblings. Let w be the parent nodes of u and v. Associate w with the frequency $p(u) + p(v)$. Delete u and v from the tree T resulting a binary tree T'' for $C' = C \setminus \{u, v\} \cup \{w\}$. By inductive hypothesis, we have $f(C') = l(T(C'))$. Thus,

$$\begin{aligned} l(T(C)) &= l(T(C')) + p(u) + p(v) \\ &= f(C') + p(u) + p(v) \\ &\leq l(T'') + p(u) + p(v) \\ &\leq l(T') \\ &\leq f(C). \end{aligned}$$

On the other hand, we have $l(T(C)) \geq f(C)$ by the definition of $f(C)$. Hence, we get $l(T(C)) = f(C)$. We finish the inductive step.

The proof of this theorem is finished. □

Worst-case running time analysis:

At each iteration, it find two nodes with least frequencies. If we use a priority queue to store the nodes, it requires two "pop" operations and one "insert" operation. These operations can be done in $O(\log n)$ time. Let $f(n)$ be the worst-case running time of Huffman's algorithm. We have

$$f(n) = f(n-1) + O(\log n).$$

It implies that $f(n) = O(n \log n)$.

Remark: Kruskal's algorithm is a greedy algorithm; Dijkstra's algorithm is a dynamical algorithm; and Huffman's algorithm is a recursive algorithm.