W3U2: Kruskal's Algorithm Part 2

C Pandu Rangan

Recap

- Kruskal's Algorithm

Outcomes

- Partition ADT
- Name Array Representation

Proof

We will now prove that A is indeed a MST.

Again, we prove this by contradiction.

Let $x_1, x_2, \cdots x_{n-1}$ be the edges in A in increasing order of weights.

Recall that all the edge weights are distinct.

If (T, A) is not a MST, let T' be a MST with $w(T') < w(T)$ and let $y_1, y, \cdots y_{n-1}$ be the edges of T' in increasing order of weights.

Proof (contd)

Let j be the first index such that

$x_j \neq y_j$ and

$x_j = y_j$ for $1 \leq i \leq j-1$

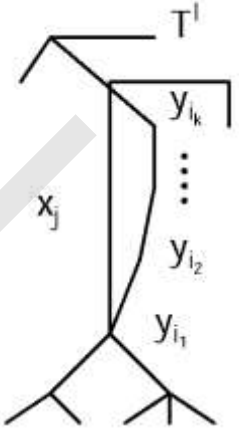
Since x_j was chosen greedily by the algorithm,

We have

$w(x_j) < w(y_j)$

We now construct a spanning tree T'' that is cheaper than T'.

Proof (contd)



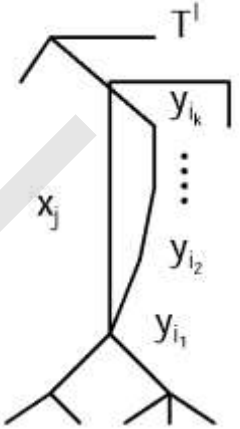
When we add the edge x_j to T', a path $\langle y_{i_1}, y_{i_2}, \cdots y_{i_k} \rangle$ in T' and x_j forms a unique cycle C as shown in figure.

If $\langle y_{i_1}, y_{i_2}, \cdots y_{i_k} \rangle \subseteq \{y_1, y_2, \cdots y_{j-1}\}$

Then $y_{i_1} = x_{i_1}, y_{i_2} = x_{i_2}, \cdots, y_{i_k} = x_{i_k}$ because

$y_1, \cdots y_{j-1} = x_1, \cdots x_{j-1}$

Proof (contd)



Thus, $x_j, x_{i_1}, x_{i_2}, \cdots, x_{i_t}$ would form the cycle C.

This is impossible as $x_j, x_{i_1}, x_{i_2}, \cdots, x_{i_t}$ are all edges of T and T is a tree.

Hence the path $\langle y_{i_1}, y_{i_2}, \cdots y_{i_t} \rangle$ must contain an edge outside the set $\{y_1, y_2, \cdots y_{j-1}\}$.

That means the path has an edge y_t of T' where $t \geq j$.

Observe that ,

$$w(y_t) \geq w(y_j) > w(x_j) \cdots\cdots (1)$$

Proof (contd)

Now define

$T'' = T' + x_j - y_t$

T'' is a spanning tree as the cycle C is now broken by the removal of y_t.

$w(T'') = w(T') + w(x_j) - w(y_t)$

$< w(T')\ by\ (1)$

This contradicts minimality of T'.

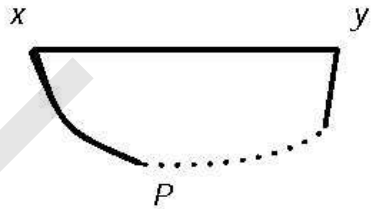
Hence a spanning tree with a cost smaller than the cost of the tree output by Kruskal's algorithm can not exist.

This implies that the output of Kruskal's algorithm is indeed an MST.

Implementation

- While the description of the Kruskal's algorithm is simple, its implementation is far from direct. Any naïve attempt to implement it may lead to exponential time code or very inefficient code.
- The key question asked in the algorithm, namely, (A'') is e forming a cycle with any subset of edges in A'', is not all easy to answer unless we use some clever data structures. This graph theoretic question itself should be seen in a different way to arrive at an efficient implementation.

Implementation (contd)



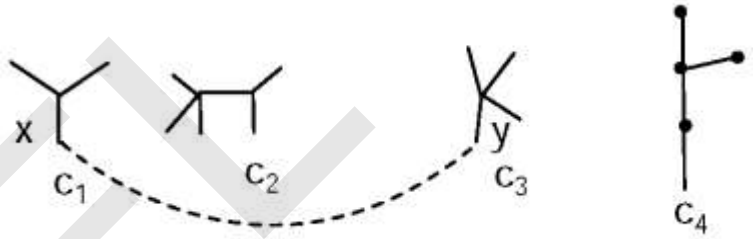
- We begin with an observation that an edge $e = (x, y)$ will form a cycle with some of the steps in A if x and y are connected by a path P consisting of edges only from A
- All edges of P are in A.
- Hence, instead of checking if an edge $e = (x, y)$ forms a cycle, we can check if there is a path connecting the end vertices x and y in the graph $T = (V, A)$.
- Equivalently, we may check if x and y are in the same connected component of T.

Implementation (contd)

Assuming there is a function
Find (x) that returns the name of the connected component of T that contains x, we may express our plan as follows:

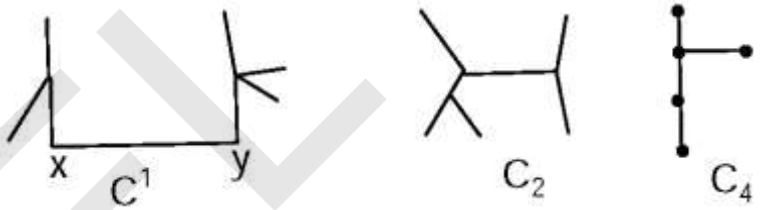
Let $e = (x, y)$
If Find $(x) \neq$ Find (y)
Add e to A
Else 'ignore e'

Where e is added to A, we need to update the information related to the connected components of A.



For example, assume that currently T has four connected components and $x \in C_1$ and $y \in C_3$ for an edge (x, y), as shown in the picture

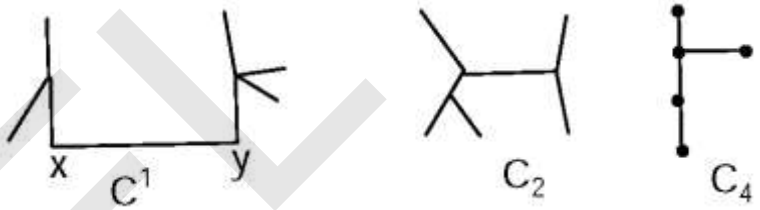
Implementation (contd)



After adding (x, y) to A, there will be only three connected components for the updated T. The components C_1 and C_3 will be merged into a bigger component say C' as shown in the picture.

In general, when we add an edge to A, the two connected components containing the end vertices of the edge will be merged to a single new component. Thus, the number of connected components in the updated T is one less than the number of connected components of previous T.

Implementation (contd)



- We give a name to this operation. Union (X, Y), where X and Y are names of two connected components of T. This results in a new connected component consisting of the union of the vertices of the components of X and Y. Note that union operation is performed only when we add an edge to A.

Pseudocode

We may now define the pseudocode we wrote earlier as follows.

\\ $G = (V, E)$ is a connected, undirected, edge weighted graph. $w(e)$ denotes the weight of the edge $e \in E$.

\\ A denotes the set of edges collected by the algorithm. $T = (V, A)$ is the subgraph G induced by the edge set A.

\\ for a vertex x, find (x) returns the name of the connected component of T containing x.

\\ Union (X, Y) merges the connected components X and Y of T into a new connected component and returns the name of the new component. The connected components X and Y are automatically removed.

\\ L is the list of edges in E in increasing order of their weights.

Pseudocode (contd)

Initialize:

$A = \emptyset$,

$e =$ First edge of L;

Initialize the names of the connected components of

$T = (V, A)$

Process:

While (NOT END OF L)

Let $e = (x, y)$

$X =$ Find (x);

$Y =$ Find (y);

If $(X \neq Y)$

Add e to A

Union (X, Y)

$e =$ Next (e, L)

Output:

$T = (V, A)$

\\ T is a MST of G

Graph theory basics

Extending the notion of adjacency, we define the notion of connectivity.

Two vertices $x \; and \; y$ are said to be connected if there is a path from $x \; to \; y$ in G. If every pair of vertices are connected, then G is said to be a connected graph. A graph that is not connected is called a disconnected graph.

In a disconnected graph, each maximally connected subgraph is called its connected component. A disconnected graph is made up of two or more connected components. A connected graph can be viewed as a graph that is made up of one connected component and that single component is the whole graph itself.

Graph theory basics (contd)

Graph theory basics (contd)

A graph is said to be acyclic if it has no cycles.
A connected acyclic graph is called a Tree
A general acyclic graph may be disconnected and each of its connected components is both connected and acyclic and hence is a tree.
Thus, a generic acyclic graph is a collection of trees and hence is known by another natural name, Forest.

Graph theory basics (contd)

Graph theory basics (contd)

There is another interesting way to look at connected components and this is quite useful in arriving at an efficient implementation of Kruskal Algorithm.
Let is define a relation on the vertex set of G as follows.
Two vertices u & v are related, denoted by uRv, iff there is a path from $u\ to\ v$. We also define $uRu\ \forall u \in V$. If G is undirected, the relation is an equivalence relation. Hence, the relation R induces a partition on the vertex set V. (This is a well-known fundamental property of equivalence relations). Each set of the partition corresponds to a connected component of G. Each connected component is a maximal set with respect to connectivity in the sense that any vertex not in the set is not reachable by a path from any vertex in the set and any vertex in a set is reachable from any other vertex from the same set.

Graph theory basics (contd)

Partition ADT

Let $V = \{1,2,3 \ldots n\}$ be a finite set with $n \geq 1$.

A k-partition of V is a collection of subsets $S_1, S_2, \ldots S_k$ of V such that

$S_i \cap S_j = \emptyset, i \neq j$

$\bigcup_{i=1}^{k} S_i = V$

An n- partition of V is a collection of n singletons

$\{1\}, \{2\}, \ldots \{n\}$

The 1-partition of V is simply the whole set V.

Partition ADT (contd)

- The identifier or a name of a set is any element of the set. Since any element of V belongs to exactly one set in the partition, we may use any element of a set to serve as an identifier of the set in a partition.
- For example, for the set $\{i\}$ in an n-partition, the integer i may serve as its name.

Name array representation

Implementation of union, fine operations on a partition.

Let $V = \{1,2, \ldots n\}$, let $P = \{S_1, S_s, \cdots S_k\}$ be a k-partition of V. Let us use the minimum element in S_i to act as the name or identifier of the set S_i.

Eg: $\{1,2,4,7\}$, $\{3,8,10\}$, $\{5,6,9,11\}$, $\{12,14\}$, $\{13\}$

↓ ↓ ↓ ↓ ↓

Names: 1 3 5 12 13

Name [1]=1
Name [2]=1
Name [3]=3
Name [4]=1

Name [7]=1
Name [8]=3
Name [9]=5
Name [10]=3

Name [13]=13
Name [14]=12

Union Find Operations

Find (x) \\ $x \in V$.
Return name $[x]$
Union (A, B) \\ A, B are names of two sets
in the current partition
\\ Assume W.L.O.G, $A < B$
For $i = 1$ to n
If (Name $[i] == B$)
Name $[i] = A$
\\ every element of B is moved to the set A
Complexity of Find (x) ----------$O(1)$
Complexity of Union (A, B) ----------$O(n)$

Thank You