Minimum Spanning Trees

CE 377K

February 17, 2015

REVIEW

HW 2 coming soon

Trees and spanning trees

Big O notation

n and m

Basic search algorithm

Algorithm

1. Set $C(r) \leftarrow \{r\}$
2. Initialize the *scan eligible list* $SEL \leftarrow \{r\}$
3. Choose some node $i \in SEL$ and remove it from SEL.
4. "Scan" node i by repeating the following steps for each link $(i, j) \in A(i)$:
    1. If node j is not in $C(r)$, add it to SEL: $SEL \leftarrow SEL \cup \{j\}$
    2. Mark node j as reachable: $C(r) \leftarrow C(r) \cup \{j\}$
5. If SEL is empty, terminate. Otherwise, return to step 3.

Proof of correctness

To show that an algorithm is "correct" we must show two things:

- It must eventually terminate.
- When it terminates, it must have the correct answer.

How do we know that our search algorithm terminates?

At each iteration, one node is removed from SEL.

If the algorithm does not terminate, this means that some node must be added to SEL infinitely many times.

However, a node is only added to SEL if it is not already in $C(r)$, and then it is added to $C(r)$ immediately afterwards.

Therefore, no node can enter SEL more than once.

So the algorithm must terminate.

How do we know that it terminates with the right answer?

By contradiction, assume that when the algorithm terminates $C(r)$ is not the set of all nodes connected to r.

There are two possibilities. $C(r)$ either contains nodes which are not connected to r, or there are nodes connected to r not in $C(r)$.

For the first case, we can show that at any point in time $C(r)$ only contains nodes connected to r. (It does so upon initialization, and nodes are only added to $C(r)$ when there is a link to that node from another node connected to $C(r)$)

For the second case, assume that there is some path between r and i but $i \notin C(r)$ when the algorithm terminates. Find the last node j in the path which is in $C(r)$, so the next node in the path k is not in $C(r)$.

When j was added to $C(r)$, it was also added to SEL. Since the algorithm terminated, j must have been removed from SEL and scanned.

When this happened, k would have been added to both SEL and $C(r)$, a contradiction.

Complexity

We have already shown that nodes are scanned at most once.

When node i is scanned, we must perform $3|A(i)|$ steps (checking if a node is in SEL, adding it to SEL if not, and adding it to $C(r)$.)

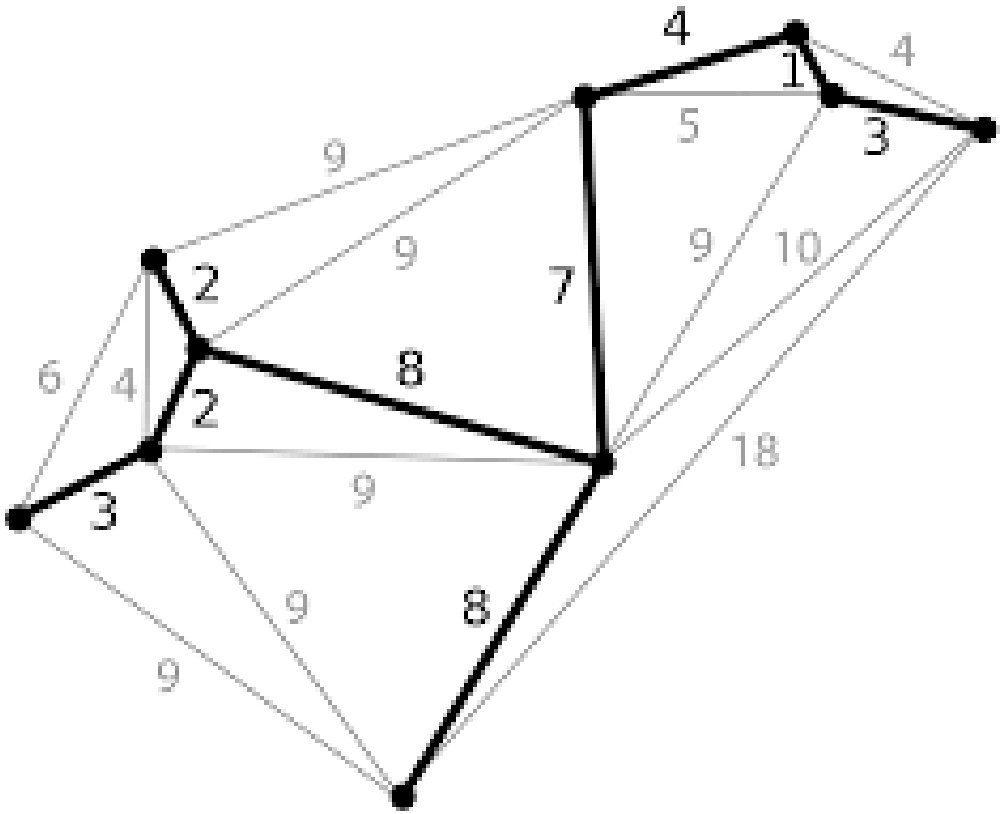
So at worst $\sum_{i \in N} 3|A(i)| = 3m$ computations are performed, which is $O(m)$.

As stated, the algorithm only determines whether or not paths exist between i and other nodes, it does not actually tell you what the paths are.

Is there some way to modify the algorithm to provide specific connecting paths as well?

MINIMUM SPANNING TREES

Minimum Spanning Tree



Identify a subset of links which form a spanning tree, where the total cost of the links in the tree is minimized.

Applications

- Building roads in rural areas
- Providing utilities and other infrastructure
- Espionage networks, passing messages between spies

In the minimum spanning tree problem, *the direction of all links is ignored*. (This is called an *undirected network*.) Connections can go in either direction.

How many links must be in a spanning tree?

In any spanning tree, adding a link will create exactly one cycle.

In a *minimum* spanning tree, any new link which is added must have a cost at least equal to the maximum cost of the other links in that cycle.

PRIM'S ALGORITHM

Example

Notation

Let $G = (N, A)$ be the original network; let $T = (N_T, A_T)$ be the links in the spanning tree.

At first T is empty, but over time N_T and A_T will grow.

T is called a *subnetwork* of G, because $N_T \subseteq N$ and $A_T \subseteq A$.

A link is *admissible* if exactly one of its end nodes is in N_T.

Algorithm

(Assumes that the network is connected.)

1. Arbitrarily choose some root note s.
2. Initialize $N_T \leftarrow \{s\}$, $A_T \leftarrow \emptyset$
3. Identify all of the admissible links; if there are none, terminate.
4. Choose an admissible link (u, v) with minimum cost. (Assume u is in N_T, but not v.)
5. Add this link to the tree: $N_T \leftarrow N_T \cup \{v\}$, $A_T \leftarrow A_T \cup (u, v)$
6. Return to step 3.

Correctness

Does it terminate?

Each iteration adds an admissible link to the tree. By doing so, one more node is added to the tree.

After $n-1$ admissible links have been added, $N_T = N$.

There are no more admissible links at this point, so the algorithm must terminate.

Correctness

When it terminates, do we have a minimum spanning tree?

There are three ways it can go wrong: at termination, T might not be a tree; or it might be a tree, but not a *spanning* tree; or it might be a spanning tree, but not a *minimum* cost one.

Is T a tree? Only admissible links are added; admissible links have one end node not in N_T, so no cycles are created.

Is T a *spanning* tree? Since the network is connected, if $N_T \neq N$ then there must be an admissible link.

Must T be a *minimum* spanning tree? Here is a proof sketch (with some handwaving).

Assume not, and let T' be a minimum spanning tree.

Let (u, v) be the first link chosen by the algorithm which is *not* part of T'.

Let (u, v') be a link connected to u in T (with $v \neq v'$)

Since Prim's algorithm chose (u, v), $c_{uv} \leq c_{uv'}$.

If $c_{uv} < c_{uv'}$, it could be swapped into T' to reduce its cost (eliminating another link on the cycle created.) However this is impossible since T' is a minimum spanning tree.

Therefore, $c_{uv} = c_{uv'}$ whenever the algorithm chooses a link not part of T', so its total cost is the same.

Complexity

There are $O(n)$ iterations (technically $n-1$).

At each iteration, we must identify all admissible links (of which there are at most m), and identify one with minimum cost (which again takes m steps).

So, Prim's algorithm is $O(nm)$.

There are more clever ways of identifying admissible links and finding one with minimum cost, which can reduce the running time to $O(m \log n)$ or $O(m+n \log n)$. These do so by avoiding "duplication of effort" in subsequent iterations.