Lecture Notes #15

Minimum Spanning Tree: Prim’s Algorithm

(For details, see pp. 570 of the textbook.)

Like Kruskal’s algorithm, Prim’s algorithm is a special case of Generic-MST algorithm.

Input: a connected weighted graph \( G = (V, E) \).
Output: \( A \), the set of edges in a minimum spanning tree of \( G \).

procedure \textit{MST-Prim}(G, w, r)
{ 
    for each node \( u \in V \) of \( G \) do 
    { 
        \( Key[u] := \infty \); 
        \( \pi[u] := \text{nil} \) 
    } 
    \( key[r] := 0; \) 
    \( Q := V; \) 
    while \( Q \neq \emptyset \) do 
    { 
        \( u := \text{Extract-Min}(Q); \) 
        for each node \( v \in \text{Adj}[u] \) do 
            if \( v \in Q \) and \( w(u, v) < key[v] \) then 
            { 
                \( \pi[v] := u; \) 
                \( key[v] := w(u, v); /* perform Heap-Decrease-Key implicitly */ \) 
            } 
    } 
}
key[v]: min weight of any edge connecting v to a node in the (partially constructed) tree.
π[v]: the parent of v in the tree.
r: the root of the tree.
Q: min-priority queue of all nodes not in the tree based on key field.
Adj[u]: adjacency list of node u.

During the execution of MST-Prim, the set A of tree edges in Generic-MST is kept implicitly as

\[ A = \{(v, π[v]) | v ∈ V - \{r\} - Q\}. \]

The nodes already included in the MST are those in \( V - Q \).
The cut that respects A: \( (V - Q, Q) \).
Edge \((u, π[u])\) is the light edge crossing this cut. Removing \( u \) from \( Q \) adds \( u \) to \( V - Q \), thus adding \((u, π[u])\) to \( A \).
By the theorem about the Generic-MST, MST-Prim produces an MST.

Figure 1 shows how Prim's algorithm works on an example.
Figure 1: The execution of Prim’s algorithm. Solid shaded edges are taken into $A$. 
Min-Heap: a Brief Review

(For details, refer to Chapter 6 of the textbook.)

A min-heap is a priority queue of tree structure implemented by a linear array.

Min-heap property: \( v \)’ parent \( \leq v \).

The following min-heap procedures are used in Prim’s Algorithm.

```latex
procedure Parent(i)
{ return \( \lceil \frac{i}{2} \rceil \) }

procedure Left(i)
{ return 2i }

procedure Right(i)
{ return 2i + 1 }

procedure Build-Min-Heap(A)
{
heapsize[A] := length[A];
for i = \( \lceil \text{length}[A]/2 \rceil \) downto 1 do
  Min-Heapify(A, i)
}

procedure Min-Heapify(A, i)
{
l := Left(i);

r := Right(i);

if \( l \leq \text{heapsize}[A] \) and \( A[l] < A[i] \)
then smallest := l else smallest := i;

if \( r \leq \text{heapsize}[A] \) and \( A[r] < A[\text{smallest}] \)
then smallest := r;

if smallest \neq i then
{ exchange A[i] \leftrightarrow A[\text{smallest}];
  Min-Heapify(A, \text{smallest})
}
}
```
procedure Extract-Min(A)
{
    if heap-size[A] < 1 then error “heap underflow”;
    min := A[1];
    Min-Heapify(A,1);
    return min
}

The following procedure is used to decrease the value of A[i] to a new value key, and maintain the min-heap property.

procedure Heap-Decreas-Key(A, i, key)
{
    if key > A[i] then error “new key is larger than current key”;
    A[i] := key;
    while i > 1 and A[Parent(i)] < A[i] do
    {
        i := Parent(i)
    }
}

Time Complexity of Prim’s Algorithm

Operations prior to while-loop:
    \(O(|V|)\) time using Build-Min-Heap.

Extract-Min: Each takes \(O(|V| \lg |V|)\) time. Total: \(O(|V| \lg |V|)\).
    \(v \in Q\) test: Each takes \(O(1)\) time if a flag is associated with each node. flag[v] = 1 if
\(v \in Q\), flag[v] = 0 if \(v \notin Q\).
    if-then statement: \(O(|V| \lg |V|)\) time for Heap-Decreas-Key operation.
    for-loop: \(O(|E|)\) iterations, since the length of all adjacency lists is \(2|E|\). Total: \(O(|E| \lg |V|)\)
    time.

Conclusion: Using min-heap, Prim’s algorithm has time complexity \(O(|E| \lg |V|)\), same as the time of Kruskal’s algorithm.