23.1 Minimum Weight Spanning Trees

Throughout this section $G=(V,E)$ denotes an undirected graph. We say that G is a <u>weighted graph</u> if to every edge $(u,v) \in E$ there is associated a weight $w(u,v)$. i.e. there is a weight function

$$w : E \to \mathbb{R}.$$

This weight is sometimes interpreted as a cost or distance.

A graph H is called a <u>subgraph</u> of G if $V(H) \subseteq V(G)$ and $E(H) \subseteq E(G)$. H is called a <u>spanning subgraph</u> if $V(H)=V(G)$. A <u>spanning tree</u> in G is a spanning subgraph which is also a tree.

The weight of a subgraph H is the total weight of all its edges:

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

T is called a <u>minimum weight spanning tree</u> if no spanning tree has lower weight than T. i.e. for all spanning trees T':

$$w(T) \leq w(T').$$

EXERCISE
PROVE THAT G CONTAINS A SPANNING TREE IFF G IS CONNECTED.

EXERCISE
PROVE THAT IF A CONNECTED GRAPH G HAS DISTINCT EDGE WEIGHTS, THEN G CONTAINS A UNIQUE MINIMUM WEIGHT SPANNING TREE.

PROBLEM (MWST)
GIVEN A CONNECTED GRAPH $G=(V,E)$ AND A WEIGHT FUNCTION $W: E \to \mathbb{R}$, DETERMINE A MINIMUM WEIGHT SPANNING TREE IN G.

WE WILL STUDY TWO FAMOUS ALGORITHMS (KRUSKAL AND PRIM) WHICH SOLVE THIS PROBLEM.

(23.2) ALGORITHMS OF KRUSKAL & PRIM

BOTH ALGORITHMS FOLLOW A SO-CALLED 'GREEDY' STRATEGY. WHILE BUILDING A SPANNING TREE, SELECT NEW EDGES ACCORDING TO SOME LOCAL OPTIMUM CRITERION, I.E. AMONGST ALL 'SUITABLE' EDGES, CHOOSE ONE OF MINIMUM WEIGHT.

WE BEGIN WITH A HIGH LEVEL DESCRIPTION OF BOTH ALGORITHMS. THESE WILL BE MOST USEFUL FOR TRACING AND PROVING THINGS ABOUT THE ALGORITHMS. LATER WE'LL CONSIDER THE MORE DETAILED VERSIONS IN THE BOOK.

IN THE FOLLOWING, A DENOTES A SUBSET OF $E(G)$.

KRUSKAL

1.) $A \leftarrow \emptyset$
2.) while $|A| < |V| - 1$
3.) AMONGST ALL EDGES IN $E - A$ WHOSE ADDITION TO A WOULD NOT FORM A CYCLE, LET e BE ONE OF MINIMUM WEIGHT
4.) $A \leftarrow A \cup \{e\}$

THEOREM

WHEN KRUSKAL IS COMPLETE $T = (V, A)$ IS A MWST IN G.

WE WILL NOT PROVE THIS THEOREM EXCEPT TO NOTE THAT KRUSKAL OBVIOUSLY PRODUCES A SPANNING TREE. THIS FOLLOWS FROM THE TREENESS

THEOREM SINCE NO CYCLES ARE EVER CREATED AND WE STOP WHEN $|A| = |V| - 1$.

IN THE FOLLOWING ALGORITHM AGAIN $A \subseteq E(G)$ AND W DENOTES A SUBSET OF $V(G)$.

PRIM

1.) $A \leftarrow \emptyset$
2.) PICK ANY $r \in V$, $W \leftarrow \{r\}$
3.) WHILE $|A| < |V| - 1$
4.) AMONGST ALL EDGES IN $E - A$ WITH EXACTLY ONE END IN W (AND THE OTHER END IN $V - W$) LET e BE ONE OF MINIMUM WEIGHT.
5.) $A \leftarrow A \cup \{e\}$
6.) $W \leftarrow W \cup \{\text{'OTHER END' OF } e\}$

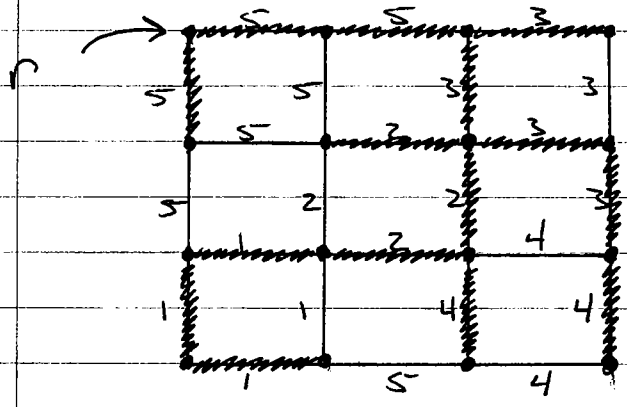
THEOREM

WHEN PRIM IS COMPLETE $T = (V, A)$ IS A MWST IN G (AND $W = V$).

AGAIN WE WILL NOT PROVE THIS, BUT IT IS OBVIOUS THAT PRIM PRODUCES A SPANNING TREE BY THE TREENESS THEOREM.

OBSERVE THAT BOTH ALGORITHMS WOULD BE DIFFICULT TO IMPLEMENT AS STATED SINCE IT IS NOT CLEAR HOW TO PERFORM KRUSKAL LINE 3 OR PRIM LINE 4. ALSO NEITHER ALGORITHM IS DETERMINISTIC AS STATED SINCE IT IS NOT SPECIFIED HOW TO SELECT EDGE e WHEN MORE THAN ONE CHOICE IS AVAILABLE.

EX. PRIM

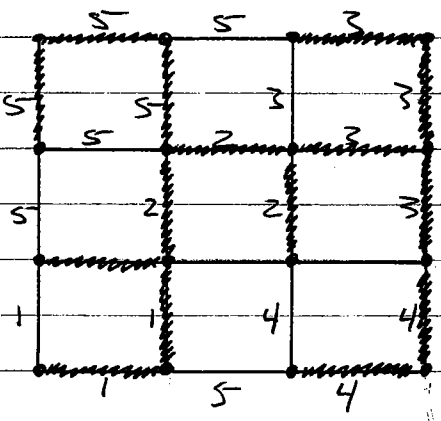


$w(T) = 44$

OBSERVE THAT PRIM GROWS A SINGLE TREE (W, A) FROM THE 'SEED' r UNTIL W BECOMES ALL OF V.

ON THE OTHER HAND KRUSKAL SIMULTANEOUSLY GROWS A NUMBER OF DISJOINT TREES UNTIL THEY ALL MERGE INTO (V, A).

EX. KRUSKAL



$w(T) = 44$

IN THE FOLLOWING DETAILED VERSION OF PRIM, EACH VERTEX u HAS TWO ATTRIBUTES: $P[u]$, THE PARENT OF u, AND $Key[u]$ WHICH IS USED TO SORT VERTICES. Q DENOTES A MIN-PRIORITY QUEUE WHICH STORES VERTICES ACCORDING TO THEIR KEY VALUES.

```
Prim(G, w, r)
1.) for each u ∈ V
2.)     Key[u] ← ∞
3.)     P[u] ← nil
4.) Key[r] ← 0
5.) Q ← V   (i.e. Build a min-PRIORITY FROM V)
6.) while Q ≠ ∅
7.)     u ← ExtractMin(Q)
8.)     for each v ∈ Adj[u]
9.)         if v ∈ Q AND w(u,v) < Key[v]
10.)            P[v] ← u
11.)            Key[v] ← w(u,v)   (i.e. DecreaseKey)
```

DURING EXECUTION THE SET $V - Q$ CONSTITUTES THE SET OF VERTICES OF THE TREE T BEING BUILT ($V - Q = W$ IN EARLIER VERSION)
SPECIFICALLY

$$V(T) = V - Q$$

$$E(T) = \{ (P[x], x) \mid x \in V(T) \text{ & } P[x] \neq nil \}$$

THUS WHEN PRIM(G, w, r) IS COMPLETE THE MWST T IS THE ROOTED TREE (WITH ROOT r) GIVEN IMPLICITLY BY THE PARENT FIELDS.

LOOP 6-11 MAINTAINS THE FOLLOWING INVARIANT:

> FOR EACH $v \in Q$, key[v] IS THE MINIMUM WEIGHT OF ANY EDGE WHICH JOINS v TO T. BY CONVENTION key[v] = ∞ IF NO SUCH EDGE IS KNOWN TO EXIST.

THUS P[v] NAMES THE (FUTURE) PARENT OF v IN T, i.e. P[v] IS THE END OF THAT MINIMUM WEIGHT EDGE WHICH LIES IN T, AND v IS THE SO CALLED 'OTHER END' OF THAT EDGE.

LINE 7 PICKS THE 'closest' VERTEX TO T THEN ADDS IT TO THE TREE BY DELETING IT FROM Q. THE EDGE (P[u], u) IS BEING ADDED TO THE TREE IMPLICITLY. SINCE T HAS CHANGED, THE KEYS OF SOME VERTICES OUTSIDE T MAY BE INCORRECT. OBVIOUSLY THE VERTICES AFFECTED BY THE CHANGE ARE THOSE ADJACENT TO THE NEWLY ADDED VERTEX u. LOOP 8-11 MAKES THE NECESSARY CHANGES TO P[v] AND key[v] FOR THOSE VERTICES v WHICH ARE BOTH OUTSIDE THE TREE (i.e. $v \in Q$) AND ADJACENT TO u.

Run Time →

We assume Q is implemented as a Min Heap.

Use Build-Heap to perform initialization steps 1-5 in time $O(|V|)$. Loop 6-11 executes $|V|$ times, and each call to ExtractMin costs $O(\lg|V|)$ time. Thus the total cost of all calls to ExtractMin is $O(|V|\lg|V|)$.

Loop 8-11 executes a total of $O(|E|)$ times. Within this loop the test $v \in Q$ can be performed in constant time. (Just keep a flag for each vertex recording whether or not it is in Q.) Line 11 involves a call to DecreaseKey which costs $O(\lg|V|)$ time. Thus the total cost of all such operations is $O(|E|\lg|V|)$

Therefore the total Run Time of Prim is

$$O(|V|\lg|V| + |E|\lg|V|).$$

However since G is assumed to be connected we have $|E| \geq |V| - 1$ (exercise), whence $|V| \leq |E| + 1 = O(E)$. Thus the Run Time of Prim is in the class:

$$O(|E|\lg|V|).$$