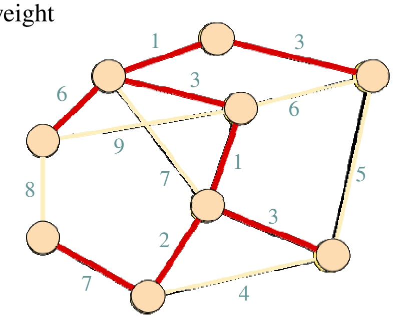
Algorithms

Dana Shapira
Lesson #6:
Minimum Spanning Trees (MST)

Minimum Spanning Tree

Given a graph $G = (V, E)$ and an assignment of weights $w(e)$ to the edges of G, a ***minimum spanning tree*** T of G is a spanning tree with minimum total edge weight

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

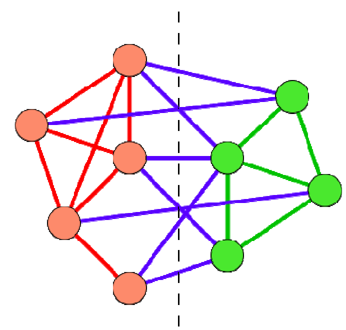


How To Build A Minimum Spanning Tree

- **General strategy:**
  1. Maintain a set of edges A such that (V, A) is a spanning forest of G and such that there exists a MST (V, F) of G such that $A \subseteq F$.
  2. As long as (V, A) is not a tree, find an edge that can be added to A while maintaining the above property.
- **Two greedy variants of this strategy:**
  - **Kruskal's algorithm**
  - **Prim's algorithm**

Cuts

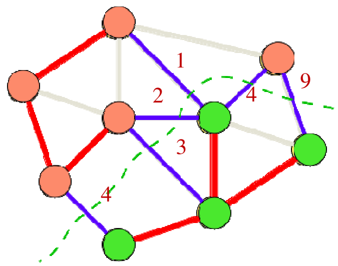
A ***cut*** (X, Y) of a graph $G = (V, E)$ is a partition of the vertex set V into two sets X and $Y = V \setminus X$.
An edge (v, w) is said to cross the cut (X, Y) if $v \in X$ and $w \in Y$.



A cut (X, Y) ***respects*** a set A of edges if no edge in A crosses the cut.

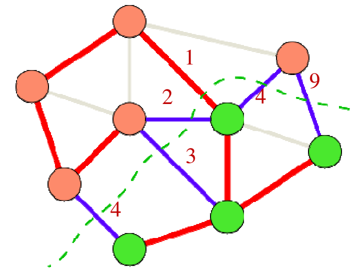
A Cut Theorem

Theorem: *Let A be a subset of the edges of some minimum spanning tree of G; let (X, Y) be a cut that respects A; and let e be a minimum weight edge that crosses (X, Y). Then A* $\cup\{e\}$ *is also a subset of the edges of a minimum spanning tree of G; edge e is* ***safe***.



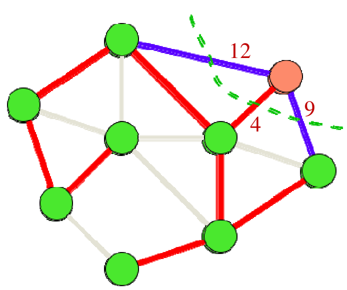
A Cut Theorem

Theorem: *Let A be a subset of the edges of some minimum spanning tree of G; let (X, Y) be a cut that respects A; and let e be a minimum weight edge that crosses (X, Y). Then A* $\cup\{e\}$ *is also a subset of the edges of a minimum spanning tree of G; edge e is* ***safe***.

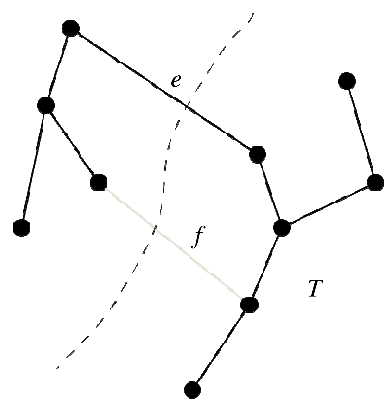


A Cut Theorem

Theorem: *Let A be a subset of the edges of some minimum spanning tree of G; let (X, Y) be a cut that respects A; and let e be a minimum weight edge that crosses (X, Y). Then A* $\cup\{e\}$ *is also a subset of the edges of a minimum spanning tree of G; edge e is* ***safe***.



A Cut Theorem



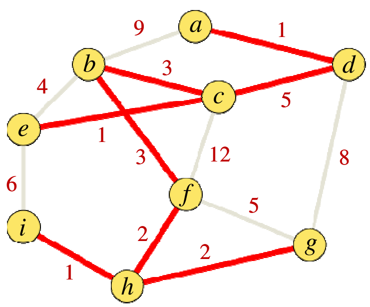
$w(e) \le w(f)$

$w(T') \le w(T)$

Kruskal's Algorithm

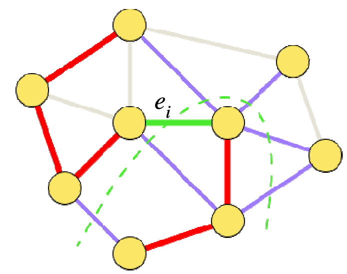
Kruskal(G)

$A \leftarrow \emptyset$
for every edge $e = (v, w)$ of G, sorted by weight
do if v and w belong to different connected components of (V, A)
then add edge e to A



(a, d):1(h, i):1 (c, e):1(f, h):2 (g, h):2 (b, c):3(b, f):3 (b, e):4 **(c, d):5**(f, g):5 (e, i):6 (d, g):8 (a, b):9 (c, f):12

Correctness Proof



Sorted edge sequence: $e_1, e_2, e_3, e_4, e_5, e_6, \ldots, e_i, e_{i+1}, e_{i+2}, e_{i+3}, \ldots, e_n$

Every edge e_j that cross the cut have a weight $w(e_j) \geq w(e_i)$ for j>i.

Hence, edge e_i is safe.

Union-Find Data Structures

- Union-find data structures solve the following problem:
- Given a set S of n elements, maintain a partition of S into subsets $S_1, S_2, \ldots, S_k$
- Support the following operations:
  - **Union(x, y):** Replace sets S_i and S_j such that $x \in S_i$ and $y \in S_j$ with $S_i \cup S_j$ in the current partition
  - **Find(x):** Returns a member $r(S_i)$ of the set S_i that contains x
- In particular, Find(x) and Find(y) return the same element if and only if x and y belong to the same set.
- It is possible to create a data structure that supports the above operations in $O(\alpha(n))$ amortized time, where α is the inverse Ackermann function.

Kruskal's Algorithm Using Union-Find Data Structure

Kruskal(G)

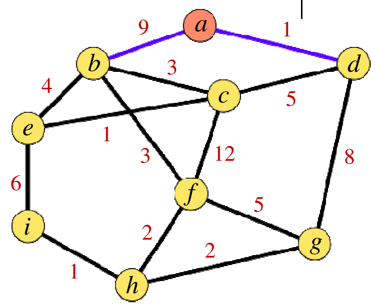
$A \leftarrow \emptyset$
Initialize a union-find data structure S that partitions V into singleton sets $\{v\}$.
for every edge $e = (v, w)$ of G, sorted by weight
do if Find(v) $\neq$ Find(w)
then add edge e to A
Union(v, w)

- **Analysis:**
- $O(m \log m)$ time for everything except the operations on S
- **Cost of operations on S:**
  - $O(\alpha(n))$ amortized time per operation on S
  - $n - 1$ Union operations
  - m Find operations
  - **Total: $O((n + m)\alpha(n))$ time**
- **Total running time: $O(m \log m)$**

Prim's Algorithm

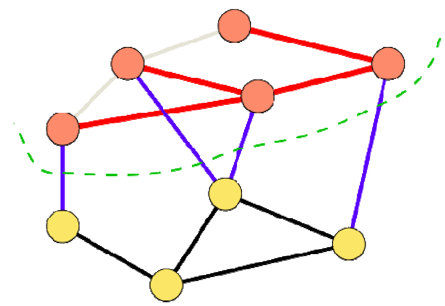
```
Prim(G)
    Q ← V
    for each u ∈ Q
      do key [u] ←∞
    s ← some vertex of G
    key [s] ← 0
    π [s] ← NULL

    while Q is not empty
      do u ← DeleteMin(Q)
      for each v ∈ Adj[u]
        then if  v ∈ Q and w(u,v) < key[v]
          then π [v] ← u
               key [v] ← w(u,v)
```



Correctness Proof

Observation: *At all times during the algorithm, the set of tree edges defines a tree that contains all visited vertices; priority queue Q contains all unexplored edges incident to these vertices.*



Corollary: *Prim's algorithm constructs a minimum spanning tree of G.*