Greedy Algorithms

Data Structures and Algorithms
Andrei Bulatov

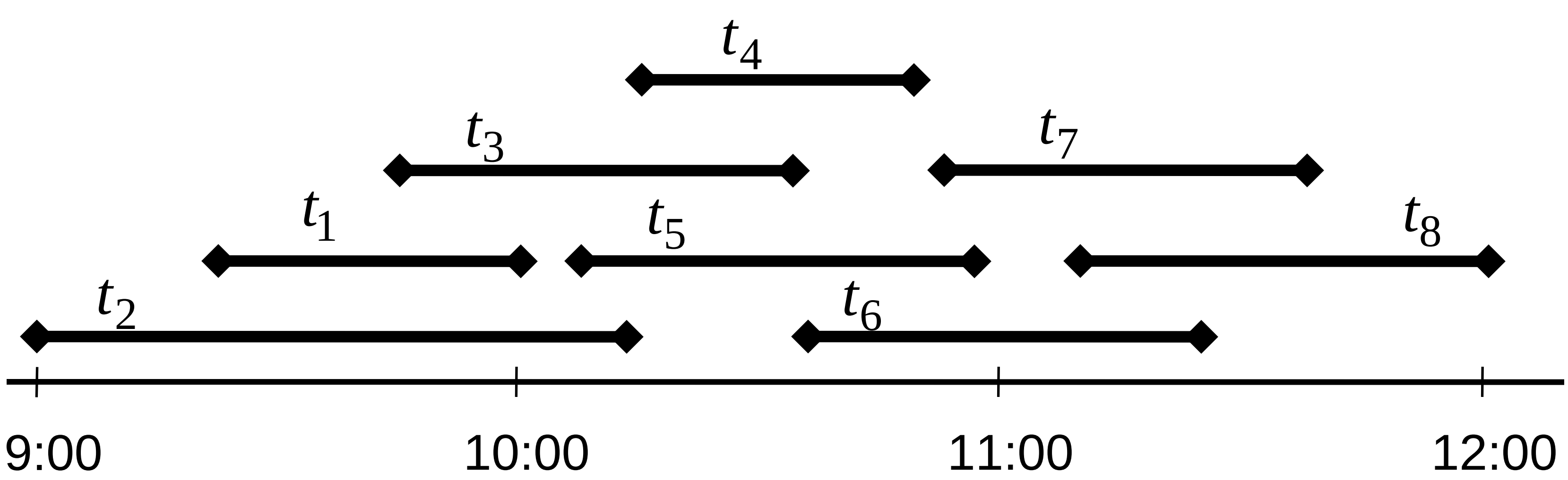
"Greed … is good. Greed is right.
Greed works."

"Wall Street"

Interval Scheduling

Consider the following problem (Interval Scheduling)

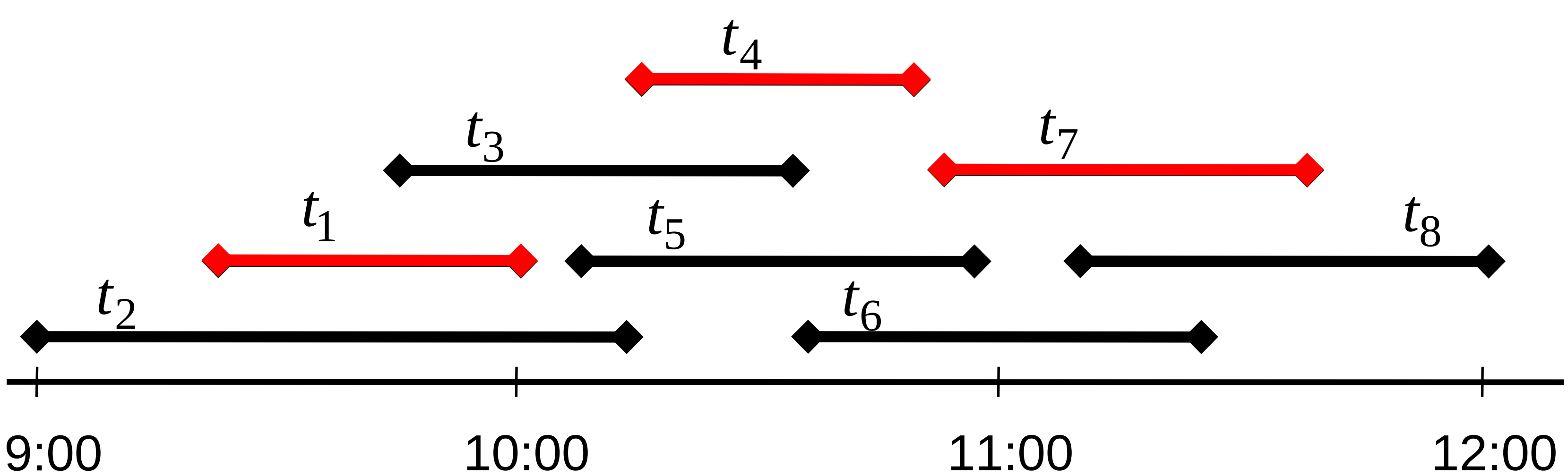
There is a group of proposed talks to be given. We want to schedule as many talks as possible in the main lecture room. Let $t_1, t_2, \ldots, t_m$ be the talks, talk t_j begins at time b_j and ends at time e_j. (No two lectures can proceed at the same time, but a lecture can begin at the same time another one ends.) We assume that $e_1 \le e_2 \le \ldots \le e_m$.



Greedy Algorithm

Greedy algorithm:

At every step choose a talk with the earliest ending time among all those talks that begin after all talks already scheduled end.



Greedy Algorithm (cntd)

Input: `Set R of proposed talks`

Output: `Set A of talks scheduled in the main lecture hall`

```
set A:=∅
while R≠∅
    choose a talk i∈R that has the smallest finishing time
    set A:=A∪{i}
    delete all talks from R that are not compatible with i
endwhile
return A
```

Theorem

The greedy algorithm is optimal in the sense that it always schedules the most talks possible in the main lecture hall.

Optimality

Proof

By induction on n we prove that if the greedy algorithm schedules n talks, then it is not possible to schedule more than n talks.

Basis step. Suppose that the greedy algorithm has scheduled only one talk, t_1. This means that every other talk starts before e_1, and ends after e_1. Hence, at time e_1 each of the remaining talks needs to use the lecture hall. No two talks can be scheduled because of that.

Inductive step. Suppose that if the greedy algorithm schedules k talks, it is not possible to schedule more than k talks.

We prove that if the algorithm schedules k + 1 talks then this is the optimal number.

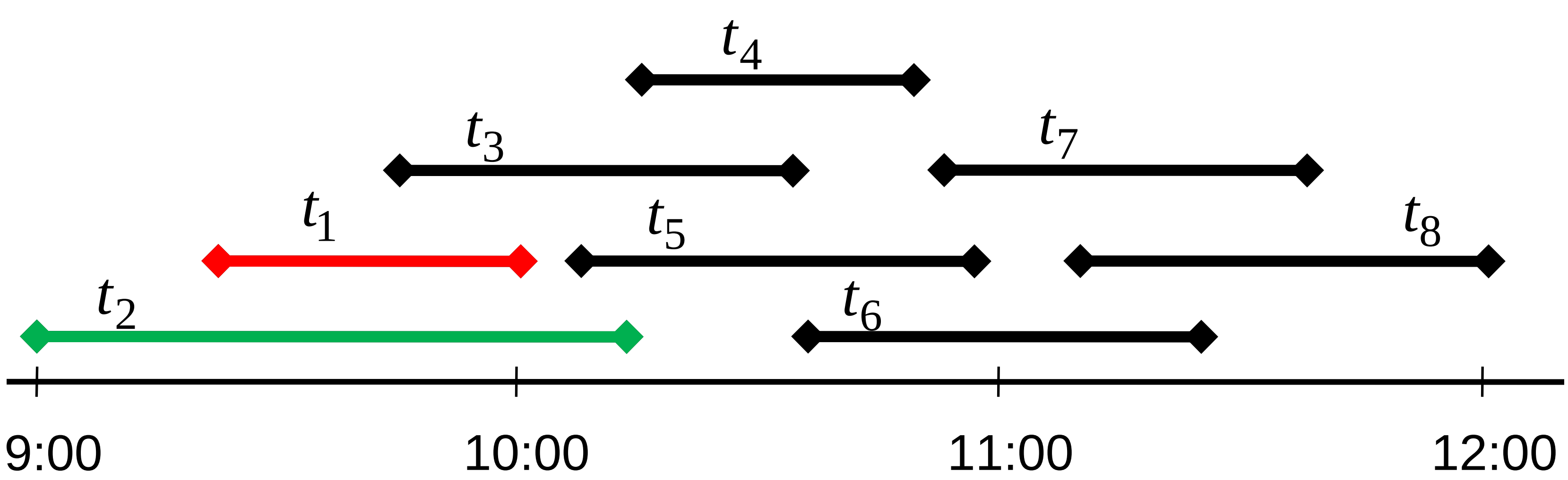
Optimality (cntd)

Suppose that the algorithm has selected k + 1 talks.

First, we show that there is an optimal scheduling that contains t_1

Indeed, if we have a schedule that begins with the talk t_i, i > 1, then this first talk can be replaced with t_1.

To see this, note that, since $e_1 \le e_i$, all talks scheduled after t_1 still can be scheduled.



Optimality (cntd)

Once we included t_1, scheduling the talks so that as many as possible talks are scheduled is reduced to scheduling as many talks as possible that begin at or after time e_1.

The greedy algorithm always schedules t_1 , and then schedules k talks choosing them from those that start at or after e_1

By the induction hypothesis, it is not possible to schedule more than k such talks. Therefore, the optimal number of talks is k + 1.

QED

Shortest Path

Suppose that every arc e of a digraph G has length (or cost, or weight, or …) len(e)

Then we can naturally define the length of a directed path in G, and the distance between any two nodes

The s-t-Shortest Path Problem

Instance:

Digraph G with lengths of arcs, and nodes s,t

Objective:

Find a shortest path between s and t

Single Source Shortest Path

The Single Source Shortest Path Problem

Instance:

Digraph G with lengths of arcs, and node s

Objective:

Find shortest paths from s to all nodes of G

Greedy algorithm:

Attempts to build an optimal solution by small steps, optimizing locally, on each step

Dijkstra's Algorithm

Input: digraph G with lengths len, and node s

Output: distance d(u) from s to every node u

Method:

let S be the set of explored nodes

for each $v \in S$ let d(v) be the distance from s to v

set S:={s} and d(s):=0

while S≠V do

pick a node v not from S such that the value

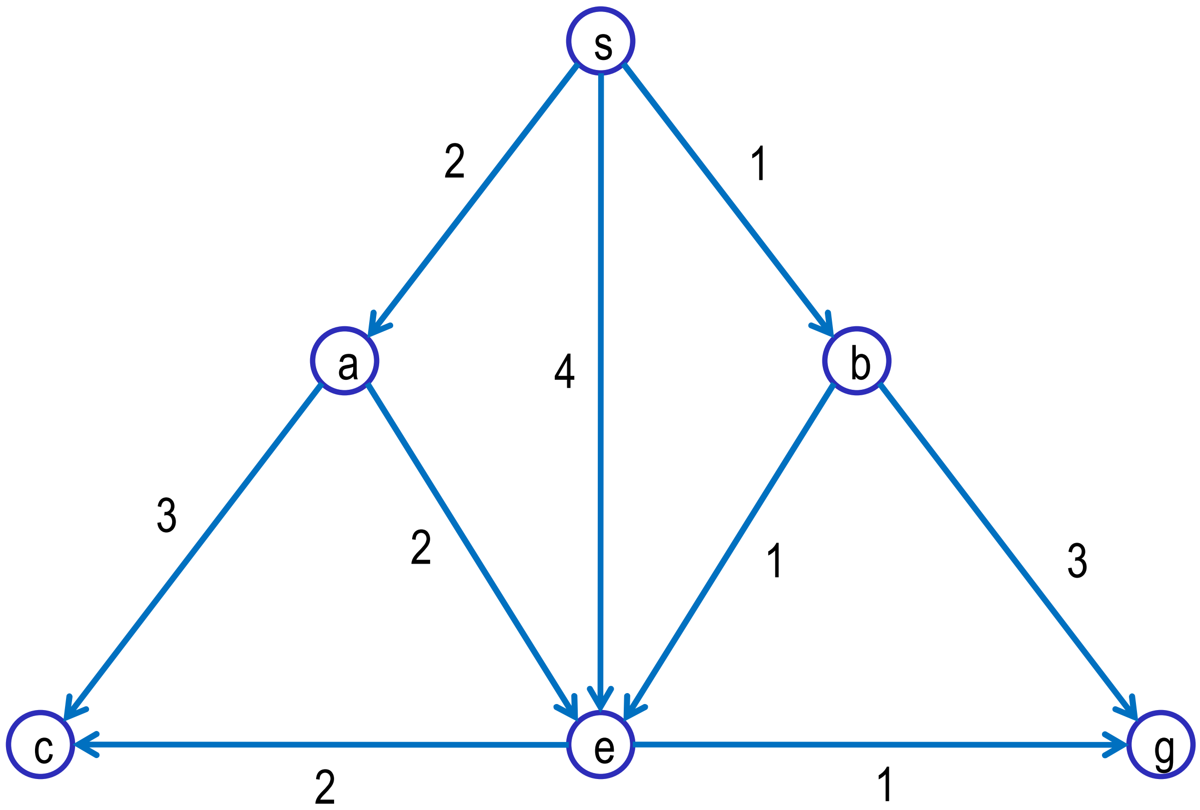
$$d'(v) := \min_{e=(u,v), u \in S} \{d(u) + len(e)\}$$

is minimal

set S:=S∪{v}, and d(v):=d'(v)

endwhile

Example



Questions

What if G is not connected?

there are vertices unreachable from s?

How can we find shortest paths from s to nodes of G?

Dijkstra's Algorithm

Input: `digraph G with lengths len, node s`

Output: `distance d(u) from s to every node u and predecessor P(u) in the shortest path`

Method:

```
set S:={s}, d(s):=0, and P(s):=null
while S≠V do
  pick a node v not from S such that the value
```

$$d'(v) := \min_{e=(u,v),u\in S}\{d(u)+len(e)\}$$

```
  is minimal
  set S:=S∪{v} and d(v):=d'(v)
  set P(v):= u (providing the minimum)
endwhile
```

Dijkstra’s Algorithm Analysis: Soundness

Theorem

For any node v the path s, … P(P(P(v))), P(P(v)), P(v), v is a shortest s – v path

Method: Algorithm stays ahead

Soundness

Proof

Induction on $|S|$

Base case: If $|S| = 1$, then $S = \{s\}$, and $d(s) = 0$

Induction case:

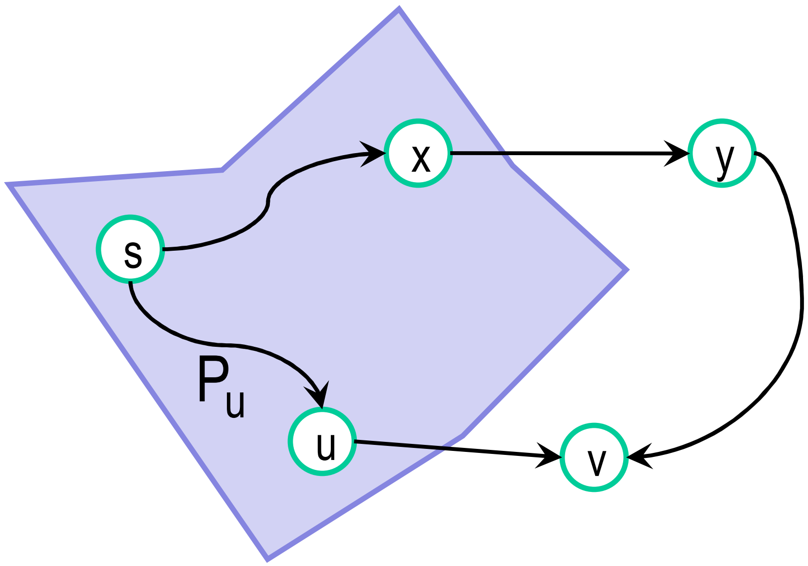
Let P_u denote the path $s, \ldots P(P(P(u))), P(P(u)), P(u), u$

Suppose the claim holds for $|S| = k$, that is for any $u \in S$ P_u is the shortest path

Let v be added on the next step.

Consider any path P from s to v other than P_v

Soundness (cntd)



There is a point where P leaves S for the first time

Let it be arc (x,y)

The length of P is at least the length of P_x + the length of (x,y) + the length of y – v

However, by the choice of v

$$len(P_v) = len(P_u) + len(u,v) \le len(P_x) + len(x,y) \le len(P)$$

QED

Running Time

Let the given graph have n nodes and m arcs

n iterations of the while loop

Straightforward implementation requires checking up to m arcs that gives O(mn) running time

Improvements:

For each node v store $d'(v) := \min_{e=(u,v), u \in S} \{d(u) + len(e)\}$ and update it every time S changes

When node v is added to S we need to change deg(v) values

m changes total

O(m+n) `calls' Properly implemented this gives O(m log n)

Recall heaps and priority queues

Spanning Tree

Design and Analysis of Algorithms

Andrei Bulatov

The Minimum Spanning Tree Problem

Let G = (V,E) be a connected undirected graph

A subset $T \subseteq E$ is called a spanning tree of G if (V,T) is a tree

If every edge of G has a weight (positive) c_e then every spanning tree also has associated weight $\sum_{e \in T} c_e$

The Minimum Spanning Tree Problem

Instance

Graph G with edge weights

Objective

Find a spanning tree of minimum weight

Prim's Algorithm

Input: graph G with weights c_e

Output: a minimum spanning tree of G

Method:

```
choose a vertex s
set S:={s}, T:=∅
while S≠V do
  pick a node v not from S such that the value
```

$$\min_{e=(u,v),u\in S} c_e$$

```
  is minimal
  set S:=S∪{v} and T:=T∪{e}
endwhile
```

Kruskal's Algorithm

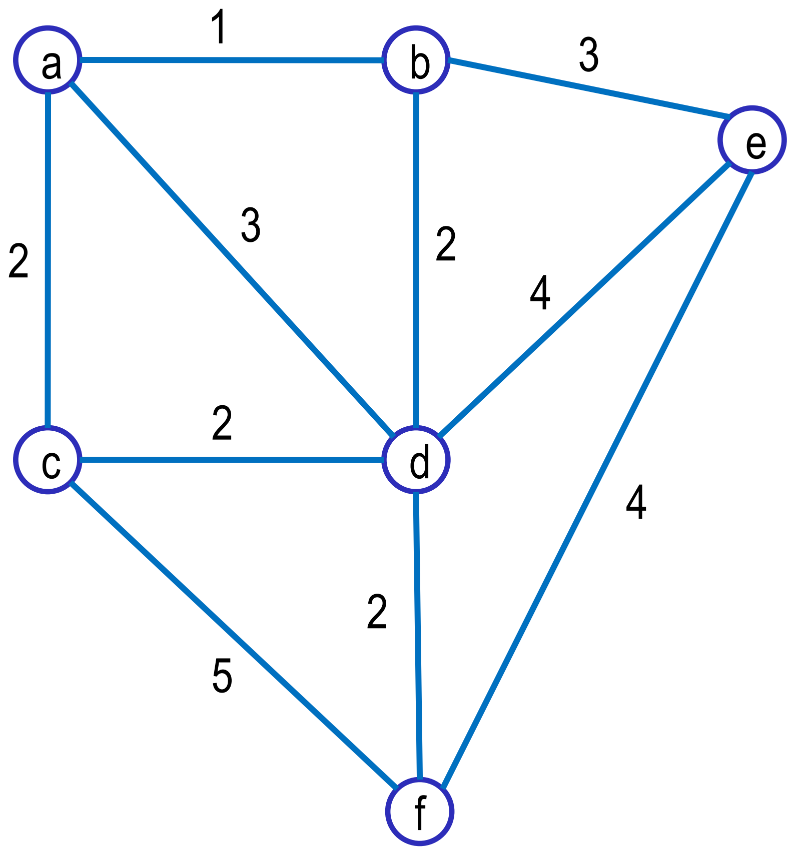
Input: `graph G with weights` c_e

Output: `a minimum spanning tree of G`

Method:

```
T:=∅
while |T|<|V|-1 do
  pick an edge e with minimum weight such that
    it is not from T and
    T∪{e} does not contain cycles
  set T:=T∪{e}
endwhile
```

Example



Kruskal's Algorithm: Soundness

Lemma (the Cut Property)

Assume that all edge weights are different. Let S be a nonempty subset of vertices, $S \neq V$, and let e be the minimum weight edge connecting S and $V - S$. Then every minimum spanning tree contains e

Use the exchange argument

Proof

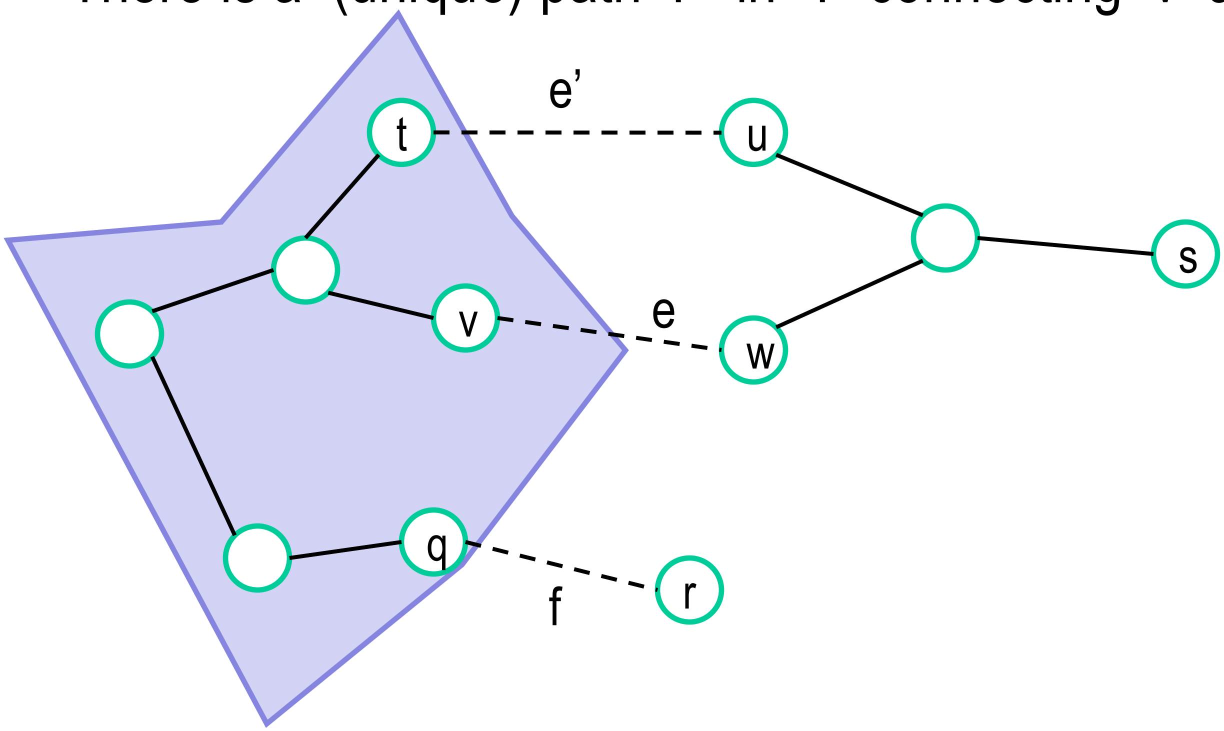
Let T be a spanning tree that does not contain e

We find an edge e' in T such that replacing e' with e we obtain another spanning tree that has smaller weight

Kruskal's Algorithm: Soundness (cntd)

Let e = (v,w)

There is a (unique) path P in T connecting v and w



Let u be the first vertex on this path not in S, and let e' = tu be the edge connecting S and V – S.

Kruskal’s Algorithm: Soundness (cntd)

Replace in T edge e' with e

$$T' = (T - \{e'\}) \cup \{e\}$$

T' remains a spanning tree

but lighter

QED

Kruskal's Algorithm: Soundness (cntd)

Theorem

Kruskal's algorithm produces a minimum spanning tree

Proof

T is a spanning tree

It contains no cycle

If (V,T) is not connected then there is an edge e such that $T \cup \{e\}$ contains no cycle.

The algorithm must add the lightest such edge

Kruskal's Algorithm: Soundness (cntd)

Proof (cntd)

T has minimum weight

We show that every edge added by Kruskal's algorithm must belong to every minimum spanning tree

Consider edge e = (v,w) added by the algorithm at some point, and let S be the set of vertices reachable from v in (V,T), where T is the set generated at the moment

Clearly $v \in S$, but $w \notin S$

Edge (v,w) is the lightest edge connecting S and V – S

Indeed if there is a lighter one, say, e', then it is not in T, and should be added instead

QED

Prim's Algorithm: Soundness (cntd)

Theorem

Prim's algorithm produces a minimum spanning tree

Proof: DIY

Kruskal's Algorithm: Running Time

Suppose G has n vertices and m edges

Straightforward:

We need to add n – 1 edges, and every time we have to find the lightest edge that doesn't form a cycle

This takes n · m · n · n, that is $O(mn^3)$

Using a good data structure that stores connected components of the tree being constructed we can do it in O(m log n) time