Greedy Algorithms

Hengfeng Wei

hengxin0912@gmail.com

June 10, 2015

Outline

Outline

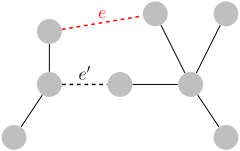
Outline

Three properties of MST

MST Property

- $G = (V, E, w)$; T is an MST of G
- $e \in G \setminus T$; $T + \{e\}$ creates a cycle C

Then, e is one of the maximum-weight edge in C.



Proof.

By contradiction (for "⇒").

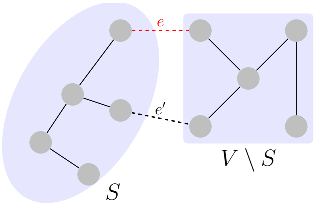
- Suppose $\exists e' \in C : w(e') > w(e)$
- $T' = T + \{e\} - \{e'\}$
- $w(T') < w(T)$

Three properties of MST

Cut Property [Problem 6.2.4 (a)]

- Graph $G = (V, E)$
- A cut $(S, V \setminus S)$ where $S, V - S \neq \emptyset$
- Let e be the minimum-weight edge across the cut

Then e must be in *some* MST of G.



Cut Property [Problem 6.2.4 (a)]

Proof.

Basic idea: $e \notin T \Rightarrow e \in T'$.

- $T + \{e\}$ to construct a cycle C
- $\exists e' \in C$ such that e' crossing the cut; $w(e') \geq w(e)$
- $T' = T + \{e\} - \{e'\}$
- $w(T') \leq w(T) \Rightarrow w(T') = w(T) \Rightarrow T'$ is an MST

$\square$

Three properties of MST

Cut Property [Another Version]

- Graph $G = (V, E)$; X is part of an MST T.
- A cut $(S, V \setminus S)$ *respecting* X (X does not cross $(S, V \setminus S)$)
- Let e be a minimum-weight edge crossing $(S, V \setminus S)$

Then, $X + \{e\}$ is part of some MST.

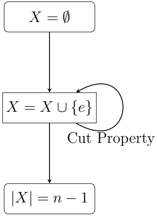
Proof.

Same As Above. □

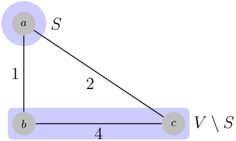
Three properties of MST

Remark:

- Kruskal's and Prim's algorithm in terms of Cut Property.
- exchange argument



Divide-and-conquer algorithm for MST [Problem 6.2.15]

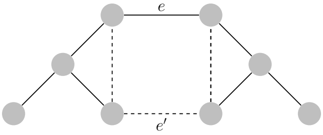


Three properties of MST

Cycle Property [Problem 6.2.4 (b), 6.2.6 (b)]

- $G = (V, E, w)$
- Let C be any cycle in G
- e is a maximum-weight edge in C

Then $\exists$ MST $T : e \notin T$.



Question.

What if all edge weights are distinct?

Three properties of MST

Cycle Property [Problem 6.2.4 (b), 6.2.6 (b)]

Proof.

Basic idea: $e \in T \Rightarrow e \notin T'$.

- $T - \{e\}$ creates cut $(S, V \setminus S)$
- $\exists e' \in C$ crossing the cut; $w(e') \leq w(e)$
- $T' = T - \{e\} + \{e'\}$
- $w(T') \leq w(T) \Rightarrow T'$ is an MST

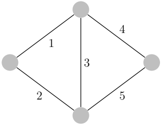
□

Remark:

- Why don't we pick any $e' \in C$?
- "Anti-Kruskal" (reverse-delete; also by Kruskal) [Problem 6.2.6 (c)]

Three properties of MST

1. The MST contains the minimum-weighted edge in every cycle [Problem 6.2.4 (c)].

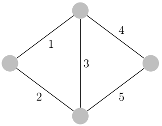


2. If e does not belong to any cycle, then e is in every MST [Problem 6.2.6 (a)].
   - Bridge!
   - OR: $T + \{e\}$ produces cycle

Three properties of MST

[Problem 6.2.1]

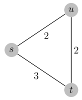
1. ✗ $|E| > |V| - 1$, e is the unique maximum edge $\Rightarrow$ e does not belong to any MST.
2. ✓ If G has a cycle with a unique maximum edge e, then e cannot be part of any MST. (Prove: Cycle property)
3. ✓ Let e be any edge of minimum edge in G. Then e belongs to some MST. (Prove: Cut property)
4. ✓ If the minimum edge is unique, then it belongs to every MST.
5. ✗ If G has a cycle with a unique minimum edge e, then e belongs to every MST.



Three properties of MST

[Problem 6.2.1]

6. ✗The shortest-path tree computed by Dijkstra's algorithm is necessarily an MST.
7. ✗The shortest path between two nodes is necessarily part of some MST.



8. ✓Prim's algorithm works correctly when there are negative edges.
9. ✓If e belongs to some MST, then e is a minimum edge across some cut.
10. ✓$w > 0$; Vertex s; shortest-path tree of s and some MST share a common edge [Problem 6.1.5]
11. ✓$w'(e) = (w(e))^2$ [Problem 6.2.2]

Outline

1. Minimum Spanning Tree
   - MST Property, Cut Property, and Cycle Property
   - Updating MST
   - Variants of MST
2. Greedy Algorithms on Intervals
3. Optional Greedy Algorithms

Updating MST [Problem 6.1.1, 6.1.6]

G and an MST T

1. $w(e)$ is decreased: $w'(e) = w(e) - k$
2. $w(e)$ is increased

Solution for (1).

- $e \in T$: no need to update $T' = T$.
  $w'(T') = w(T) - k \Rightarrow w'(T') < w(T)$.
  To prove that T' is an MST of G':
  Suppose $\exists T'' : T''$ is an ST of G' and $w'(T'') < w'(T')$.
  - $e \notin T''$: $w(T'') = w'(T'') < w'(T') < w(T)$
  - $e \in T''$: $w(T'') = w'(T'') + k < w'(T') + k = w(T)$
- $e \notin T$: $T' = T + \{e\} - \{e'\}$; e' is the maximum-weight edge in cycle and $w(e') > w(e)$
  - $e \notin T''$: $w(T'') = w'(T'') < w'(T') < w(T)$
  - $e \in T''$: ???

To verify it: `http://cs.stackexchange.com/q/43309/4911`

Critical edge: [Problem 6.1.8]

To find critical edge e: remove it, $w(T)$ increases

An MST T:

- $e \notin T$: not critical
- $e \in T$:
    - $T - \{e\}$ to produce a cut $(S, V \setminus S)$
    - $\forall e' \neq e$ across $(S, V \setminus S)$, $w(e') > w(e)$

Algorithm.

Using Kruskal's algorithm to find such e's: unique minimum-weight edge crossing cut.

Proof.

No missing critical edges: during Kruskal's algorithm. $\square$

Question.

Prim's algorithm?

Adding vertex to MST [Problem 6.1.2]

- $G = (V, E)$; an MST T
- $G' = (V', E')$: $V' = V + \{X\}, E' = E + E_X$; E_X: incident edges to X
- To find an MST T' of G'

Algorithm.

1. - Adding E_X into G one at a time ($T + \{e\} \Rightarrow C$)
   - $O(n) \times O(n) = O(n^2)$ (not $O(n) \times O(m)$)
2. - There *exists* an MST of G' that includes no edges in $G \setminus T$
   - Run MST alg. on $G'' = (V + \{X\}, T + E_X)$
   - $O(n \lg n)$
3. - "On Finding and Updating Spanning Tress and Shortest Paths", 1975
   - "Algorithms for Updating Minimum Spanning Trees", 1978
   - $O(n)$

Outline

1. Minimum Spanning Tree
   - MST Property, Cut Property, and Cycle Property
   - Updating MST
   - Variants of MST
2. Greedy Algorithms on Intervals
3. Optional Greedy Algorithms

Feedback Edge Set (FES): [Problem 6.1.4]

1. maximum spanning tree
2. (minimum) feedback edge set:
   - a set of edges which, when removed from the graph, leave an acyclic graph G'
   - assuming G is connected $\Rightarrow$ G' is connected
   - feedback *arc* set: "cycle" $\Rightarrow$ circular dependency

Algorithm.

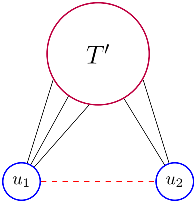
- G' is connected + acyclic $\Rightarrow$ G' is an ST
- FES $\Leftrightarrow$ $G \setminus$ Max-ST

MST with specified leaves: [Problem 6.1.7]

- $G = (V, E), U \subset V$
- finding an MST with U as leaves

Algorithm.

- $G' = G \setminus U$
- MST T' for G'
- $\forall u \in U$, attach it to T'



ST with specified edges: [Problem 6.1.10]

- $G = (V, E), S \subset E$ (no cycle in S)
- finding an MST with E as edges

Algorithm.

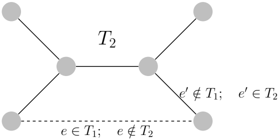
- contract each isolated component of S to a *super-vertex*
- $G \to G'$
- find MST of G'

Proof.

Prove by contradiction. □

Unique MST [Problem 6.2.7]

Distinct weights $\Rightarrow$ unique MST.



Proof.

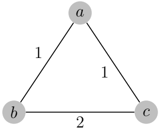
By contradiction: two MSTs $T_1 \neq T_2$.

- $\Delta E = \{e \mid e \in T_1 \setminus T_2 \vee e \in T_2 \setminus T_1\}$
- $e = \min \Delta E$. Suppose $e \in T_1 \setminus T_2$
- $T_2 + \{e\} \Rightarrow C$
- $\exists (e' \in C) \notin T_1 : w(e') < w(e)$ (MST property)
- $e' \in T_2 \setminus T_1 \Rightarrow e' \in \Delta E$

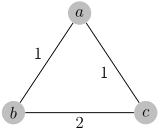
□

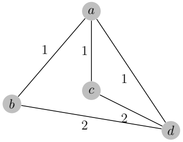
Conditions for Unique MST [Problem 6.2.5]

1. Example: unique MST, with equal weights



2. Counterexamples:
   1. ✗cut: minimum-weight edge across any cut is unique
   2. ✗cycle: maximum-weight edge in any cycle is unique





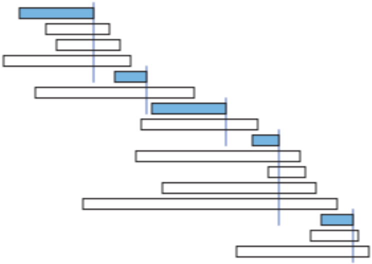
Outline

Points cover intervals [Problem 6.2.10, 6.2.12]

every interval contains at least one point in P

Algorithm and Example.

1. sorted by finishing times
2. pick the first uncovered interval



Points cover intervals [Problem 6.2.10, 6.2.12]

Every interval contains at least one point in P (*"stab"*).

Proof.

contradiction ($m < k$) + mathematical induction + exchange argument ($g_j \Rightarrow o_j$)

- $g_1, g_2, \ldots, g_m, \ldots, g_k$; $o_1, o_2, \ldots, o_m$ (**sorted**)
- Base case: $o_1 \Rightarrow g_1$
  - $o_1 \leq g_1$
  - o_1 covers $I \Rightarrow g_1$ covers I (by contradiction again!)
- Inductive step:
  - $g_1, g_2, \ldots, g_{j-1}, g_j, \ldots, g_m, \ldots, g_k$; $g_1, g_2, g_{j-1}, o_j, \ldots, o_m$
  - $o_j \Rightarrow g_j$
  - consider the next I for g to cover
- $m < k \Rightarrow o$ is not a solution.

□

Question.

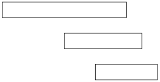
- points-cover-intervals vs. interval-scheduling

Tiling path [Problem 6.2.11]

- X: a set of intervals; finding $Y \subseteq X$ to cover all intervals.
- To minimize $|Y|$.

Algorithm:

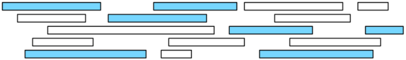
- heuristics: longest (overlapping wasted); starts/finishes first



- greedy:
  - you need to cover the first interval
  - what then?
  - always pick the interval reaching furthest right

Tiling path [Problem 6.2.11]

- X: a set of intervals; finding $Y \subseteq X$ to cover the real line.
- To minimize $|Y|$.



Proof.

- $o_1, o_2, \ldots, o_m$ sorted/identified by finishing times
- contradiction + induction + exchange argument

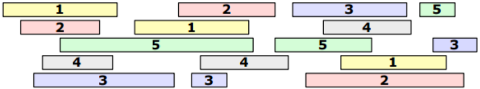
$\square$

Interval coloring/partition [Problem 6.2.13]

- applications: scheduling conflicting
  - lectures (intervals) : (as few as possible) classrooms;
  - jobs : machines

Algorithm and Example.

- sorted all times
- lock/unlock colors



Interval coloring/partition [Problem 6.2.13]

proof

1. Observation: #colors $\geq$ depth D of intervals
   - $t : I_t = \{I \mid t \in I\}$
   - $D = \max_t |I_t|$
2. Greedy algorithm: #colors $= D$
   - no two overlapping intervals are assigned the same color (color is locked)
   - each interval I is colored
     - at least one color is free for I

Base stations [Problem 6.2.16]

- houses: $x_1 < x_2 < \cdots < x_n$
- base stations: $s_1 < s_2 < \cdots < s_k$
- coverage of base station: t

Algorithm.

No overlapping coverage allowed.

- the first station: $s_1 = x_1 + t$
- remove the houses covered by s_1
- recurs on other houses

Proof.

- $o_1 \Rightarrow g_1$: $o_1 \leq x_1 + t$
- $o_j \Rightarrow g_j$: g_j is the rightmost position to cover the first uncovered house
- $m < k$: contradiction.

Rest stop [Problem 6.2.17]

- rest stops: $x_1 < x_2 < \cdots < x_n$
- one charging for $100km$
- to minimize the times of charging

Algorithm.

Go as far as possible.

- the farthest rest stop he can reach
- recurs on the rest of rest stops

Proof.

- g and o consist of positions of rest stops

Relation with "tiling path" [Problem 6.2.11]. $\square$

Total service time [Problem 6.2.20]

- a server : n customers
- service time for customer i: t_i
- to minimize $T = \sum_{i=1}^{n}(\sum_{j=1}^{i} t_j)$

Algorithm.

Sorted by increasing service times

Proof.

- To prove: T is minimized $\Leftrightarrow$ any solution $o_1, o_2, \ldots, o_n$ is increasingly sorted.
- contradiction + exchange argument

$\square$

Outline

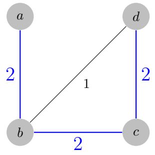
(Optional) Bottleneck spanning tree: [Problem 6.2.14]

$G = (V, E)$ has a set $\mathcal{T}$ of spanning trees of G; the most expensive edge is as cheap as possible:

$$\min_{T \in \mathcal{T}} \left(\max_{e \in T} w(e) \right)$$

Solution

- MST $\Rightarrow$ BST
  - $e \in T, e' \in T', w(e') < w(e)$
  - $T - \{e\} \Rightarrow (S, V \setminus S) : \forall e''$ across $(S, V \setminus S), w(e'') \geq w(e) > w(e')$
  - T' is not an ST.
- MST $\not\Leftarrow$ BST



(Optional) Bottleneck spanning tree: [Problem 6.2.14]

$G = (V, E)$; a set $\mathcal{T}$ of spanning trees of G:

$$\min_{T \in \mathcal{T}} \left(\max_{e \in T} w(e) \right)$$

$\Theta(m)$ algorithm BST(G):

$$T(m) = T(m/2) + \Theta(m)$$

1. $E = E_A \cup E_B \ (\forall e \in E_A, \forall e' \in E_B : w(e) \leq w(e'))$
2. If G_{E_A} is connected, BST(G_{E_A}) recursively
3. If G_{E_A} is not connected, $\text{BST}\left((G_{E_A})_\eta \cup G_{E_B}\right)$ recursively

Check: `https://en.wikipedia.org/wiki/Minimum_bottleneck_spanning_tree`

Is e in Some MST? [Problem 6.1.11]

$O(m+n)$ to decide that is e in some MST?

Algorithm.

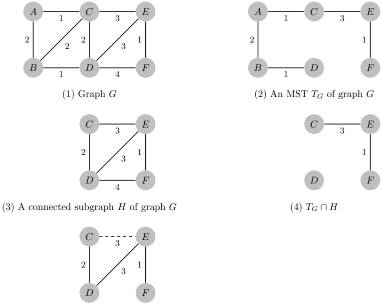
- removing all edges of weight $\geq w(e)$
- checking connectivity

Proof.

Theorem: Edge $e=(u,v)$ does not belong to any MST $\iff$ u and v can be joined by a path consisting of edges of weight $< w(e)$. $\square$

MST in Subgraph [Problem 6.2.3]

- T is an MST of G; $H \subseteq G$ connected
- To prove: $T \cap H$ is part of some MST of H



(1) Graph G

(2) An MST T_G of graph G

(3) A connected subgraph H of graph G

(4) $T_G \cap H$

(5) Some MST T_H of H

Proof.

Check: `http://cs.stackexchange.com/a/43142/4911` $\square$

Second MST [Problem 6.1.3]

Find a second MST.

Algorithm.

The second MST differs from MST by *a single* edge exchange.

Proof.

Complicated. □


THANK YOU