Greedy Algorithms

Yet another algorithm design paradigm.

1 – Divide and Conquer

2 – Dynamic Programming

3 – Greedy

↦ Making locally optimal decisions at each stage

– Will see this at work in a couple of examples.

Caution: - greedy does not always work (just like life)

- there is a well investigated understanding of when greedy works and there are interesting takeaways, but we won't discuss that.

(Matroid based optimization problems)

Examples:

① An algorithm for minimum spanning tree on graphs.

② An algorithm for lossless data compression.

Minimum Spanning Tree

Input: Undirected graph $G = (V, E)$. [either assume connected or detect connectivity]
Edge weights l_e (could be negative)

Output: A tree $T = (V, E')$, $E' \subseteq E$ that minimizes weight$(T) = \sum_{e \in E'} l_e$

Tree: Undirected graph that is connected and acyclic

Spanning: Vertex set of T equals V.

Properties of Trees:

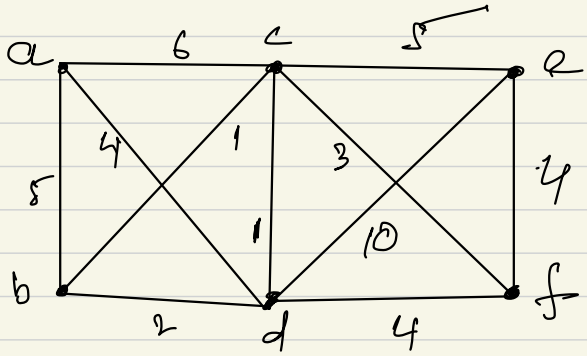
1. A tree on n vertices has exactly $(n-1)$ edges.
2. Any connected undirected graph with n vertices and $(n-1)$ edges is a tree.
3. An undirected graph is a tree iff there is a unique path between any pair of vertices.

Proofs: Homework.

Greedy Algorithm for minimum spanning tree.

(Want to minimize the total weight of the tree, so)

Repeatedly add the minimum weight edge in the graph that does not produce a cycle in the tree.



Sort the edges as per their weight.

bc, cd, bd, cf, df, ef, ad, ab, ce, ac, de

2. what edges are added to the tree 2

3. what is the final spanning tree.

- Many known algorithms for MST based on greedy.
- share some similarity

Kruskal's MST

① $E' = \{\}$

② Sort edges in E by increasing order of their weight

③ for all $(u,v) \in E$ in this order

④ if $\{(u,v)\} \cup E'$ is acyclic

$E' := E' \cup \{(u,v)\}$.

How do you check (5): whether or not adding an edge makes a graph acyclic?

Running time:

Correctness of Kruskal's Algorithm:

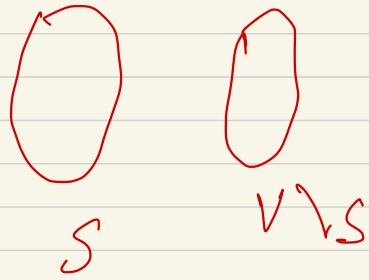
- correctness is typically harder for greedy algorithms than the ones based on Divide and conquer / DP
- On the other hand, these algorithms will be easier to come up with, compared to Div and Conq. / DP.

Cut property

Suppose X is a subset of edges in a Min spanning Tree of G.

Let $S \subseteq V$ s.t there is no $S - V \setminus S$ edge in X.

Let e be the edge of min wt between $S - V \setminus S$.

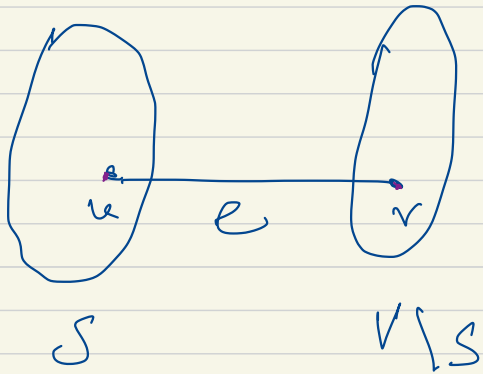


Then, $X \cup \{e\}$ is a part of some minimum spanning tree.

(Not necessarily the same spanning tree that X came from)

Pf:

T - MST containing the set X of edges



If $e \in T$ → done

Else, $e \notin T$.

Consider $\tilde{T} = T \cup \{e\}$

Is $\tilde{T}$ a tree?

$T \cup \{e\}$ has a cycle.

This cycle MUST contain e. why?

It must also contain another edge e' across $S - V \setminus S$ cut. why?

Set $T' = T \cup \{e\} - \{e'\}$

Note: Weight of $T' \leq$ wt of T. why?

Claim: T' is a tree.

Pf: Properties of Tree - - - .

But T is a min wt spanning tree.
$\Rightarrow$ $wt(T') = wt(T) \Rightarrow T'$ min wt spanning tree. ☐

So, what?

How does this imply correctness of Kruskal's algorithm.

Claim: At each stage i in Kruskal's, set E_i of edges selected satisfies:
- $\exists$ a min-wt spanning tree T_i in the graph
  s.t $E_i \subseteq T_i$.

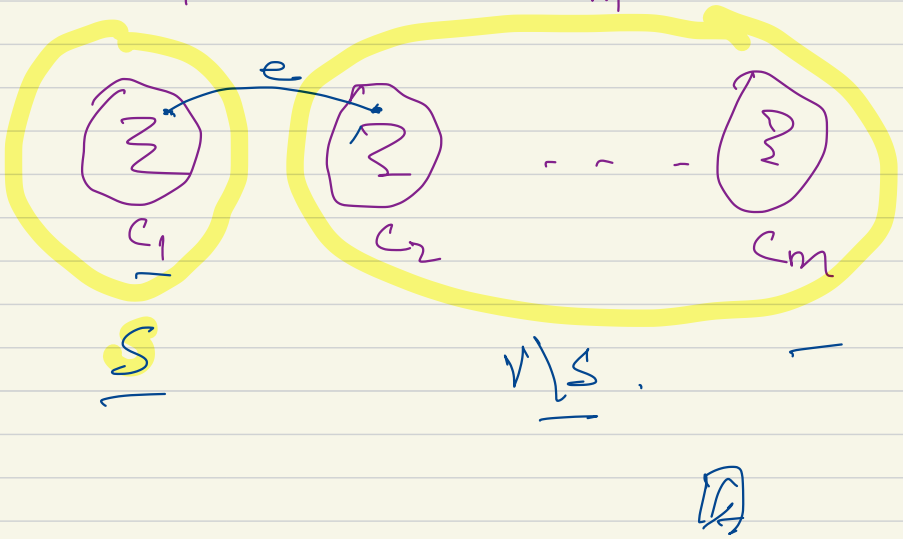
Pf: Proof via induction on i.

where do we use the cut property?

$E_i := E_{i-1} \cup \{e\}$

connected components of subgraph E_{i-1}

$C_1 \quad C_2 \quad \cdots \quad C_m$



Cut prop $\Rightarrow$ E_i is a subset of edges in some MST in G.

[This kind of cut and paste argument is very useful for these problems.]

Faster Implementation of Kruskal.

In every iteration:
- finding the smallest wt edge that does not produce a cycle.
- Maintain each connected component separately
- find min of edge with end points in different connected components.
- Update the connected

components.

- Union-Find Data structure.
- speeds up Kruskal to $O((m+n)\log n)$ time.

Homework: Read up on union find data structure.

Other well known greedy algorithms.

Prim's (Discovered by many — including Dijkstra's.

1. $E' = \{\}$
2. $S - \{s\}$ — arbitrary.
3. while there is an edge (u,v) with $u \in S$, $v \notin S$
   $(u^*, v^*) :=$ such an edge of min wt
   Add v^* to S
   add (u^*, v^*) to E'.
4. Return E'.

Exercice: Prove the correctness of Points algorithm.
and bound its running time.