Review:

Greedy Algorithms:

1) Scheduling

2) Huffman Codes

3) Minimum spanning Trees (MST)

The Cut Property:

Let $S \subseteq V$ and let $X \subseteq E$ be part of a MST $T$ s.t. $X$ has no edge from $S$ to $\bar{S} = V \setminus S$

If $e$ is a lightest edge from $S$ to $\bar{S}$

then $X \cup e$ is part of some MST $T'$

Prims Alg: Maintain a tree $(S_t, X_t)$, in each step adding a vertex $v_{t+1}$ that minimize

$\text{cost}(1v) = \min_{u \in S_t} w_{uv}$

Kruskal's Alg: Order edges by weight, and in each step add next edge which does not create a cycle
Prim (G, w)

\[ \forall u \in V \quad \text{cost}(u) = \infty, \quad \text{prev}(u) = n1 \]

Pick any \( u_0 \in V \)
\[ \text{cost}(u_0) = 0 \]

\[ \forall v \in V \quad \text{insert key} (v, \text{cost}(v)) \]

while queue non empty
\[ v = \text{Delete Min} \]

\[ \forall \{v, u\} \in E \]
if \( \text{cost}(u) > w(v, u) \)
\[ \text{cost}(u) = w(v, u) \]
\[ \text{prev}(u) = v \]
\[ \text{Decrease Key}(u) \]

Union Find Data Structure:

- \text{makeSet}(x) \quad \text{makes singleton containing } x
- \text{Find}(x) \quad \text{which set does } x \text{ belong to}
- \text{union}(x, y) \quad \text{merges sets contain } x, y

Kruskal (G, w)

For all \( v \in V \), \text{makeSet}(v)

\[ X = \{ \} \]

Sort edges in \( E \) by \( w(\cdot) \)

\[ \forall \{u, v\} \in E \text{ in that order} \]
if \( \text{Find}(u) \neq \text{Find}(v) \)
add \( \{u, v\} \) to \( X \)
\[ \text{union}(u, v) \]

return \( X \)

\[ n = |V|, \quad m = |E| \]

\[ O(n) \quad \text{insert, delete} \]
\[ O(m) \quad \text{decrease key} \]

Running Time
\[ O((n+m) \log n) \]

\[ n \quad \text{makeSet} \quad x \quad O(|V|) \]
\[ 2m \quad \text{Find} \quad x \quad O(|E| \log |V|) \]
\[ n-1 \quad \text{union} \quad x \quad O(|E| \log |V|) \]
\[ + \text{sort } O(m \log m) \]

Running Time
\[ O((m+n) \log n) \]
**Union Find Data Structure**

We need data structure for finite sets

Choose trees, label set by its root

\[ \pi(x) = "\text{parent of } x" \]

\[ \text{rank}(x) = \text{high of tree under } x \]

\{A, B, C, D, E, F, G\}, \{F, I\}, \{6, H\}

\[
\begin{align*}
\text{makeSet}(x) & & \text{findSet}(x) \\
\pi(x) &= x & \text{while } \pi(x) \neq x \\
\text{rank}(x) &= 0 & x = \pi(\pi(x)) \\
& & \text{return } x
\end{align*}
\]

**Union:**

\[ A \bigcup C \bigcup G \rightarrow A \bigcup C \bigcup F \bigcup G \]

Deeper trees make \text{findSet}(x) slower

\[
\begin{align*}
\text{union}(x, y) & \\
\pi_x = \text{findSet}(x), \pi_y = \text{findSet}(y) \\
\text{if } \pi_x = \pi_y : & \text{ return } \\
\text{if } \text{rank}(\pi_x) \leq \text{rank}(\pi_y) : & \pi(\pi_x) = \pi_y \\
\text{if } \text{rank}(\pi_x) = \text{rank}(\pi_y) : & \text{rank}(\pi_y) = \text{rank}(\pi_x) + 1 \\
\text{else} : & \pi(\pi_y) = \pi_x
\end{align*}
\]

**Example:**

\[ \text{makeSet}(A), \ldots \text{makeSet}(D) \]

\[ A^0, B^0, C^0, D^0 \leftarrow \text{rank} \]

\[ \text{union}(A, B) : B^1 \]

\[ \text{union}(C, D) : D^1 \]

\[ \text{union}(A, C) : \]

\[ A^0 \bigcup A^0 \bigcup C^0 \rightarrow B^1 \bigcup C^0 \]
Example 2:

\texttt{makeSet(A), \ldots makeSet(E)}

\begin{align*}
A^* & \quad B^* \quad C^* \quad D^* \quad E^* \\
\text{union}(A, B) : & \quad B' \quad & \text{union}(A, C) : & \quad B' + C^* \\
\text{union}(D, E) : & \quad E' & \quad D^* \\
\text{union}(A, D) : & \quad B' + E' \quad & \quad E'' \\
\end{align*}

\textbf{Running Time:}

\begin{align*}
\text{makeSet: } & \quad \theta(n) \\
\text{find: } & \quad \Theta(\text{rank of root}) \\
\text{union: } & \quad \Theta(\text{rank root}) \\
\end{align*}

\textbf{Claim:} If \( \text{rank}(x) = 2 \) and \( x \) is the root

\( \Rightarrow \) tree under \( x \) has at least \( 2^k \) nodes

\textbf{Pf of Claim}

\begin{align*}
\text{\( k = 0 \)} & \quad \Rightarrow \quad 2^1 = 1 \\
\text{\( k \rightarrow k+1 \) The only way to create a rank 9+1} \\
\text{tree is to merge two rank 2 trees} \\
\end{align*}
**Corollary:** If we have a set of elements, the rank is always \( \leq \lfloor \log_2 n \rfloor \)

\( \Rightarrow \) Run time of union, find is \( O(\log n) \)

4) **Horn Formulae**

\( x_1, \ldots, x_n \) Boolean variables (can be set to TRUE or FALSE)

**SAT-Formula:**

Any expression \( F \) that can be obtained from \( x_1, \ldots, x_n \) by iteratively applying AND, OR or NOT \((\land, \lor, \neg)\)

**Example:**

- a) The weather is nice
- b) I am inside
- c) My NeurIPS paper was rejected
- d) You are happy with my teaching

\[ F = (a \lor b) \land \neg c \land d \]

**More complicated example**

\[ F = (a \lor b) \land \neg c \land d \land (\neg a \lor b) \]

**Conjunctive Normal Form**

\[ L = \{ x_1, \overline{x}_1, \ldots, x_n, \overline{x}_n \} \] Literals

- \( \uparrow \) positive \( L \),
- \( \uparrow \) negative \( L \).

**Claim:** Any SAT formula can be written in conjunctive Normal Form (CNF), i.e. as

\[ F = C_1 \land \ldots \land C_m \]

where each clause \( C_i \) is an OR of literals.
Proof by induction

\[ F = F_1 \land F_2 \]

\[ F = F_1 \lor F_2 \quad F_1 = \bigwedge_{i=1}^{m} C_i; \quad F_2 = \bigwedge_{j=1}^{n} D_j; \]

\[ = \bigwedge_{i, j} (C_i \lor D_j) \]

\[ F = \overline{C_1 \land \cdots \land C_m} = \bigvee_{i=1}^{m} \overline{C_i} \]

\[ C_i = \ell_1 \lor \cdots \lor \ell_a \]

\[ \overline{C_i} = \overline{\ell_1} \land \cdots \land \overline{\ell_a} \]

**SAT-Problem:** Given a CNF-Formula \( F \), find True, False assignments to the variables s.t. \( F \) is TRUE ("satisfied")

In short: Find satisfying assignment to CNF formula \( F \)

- Hard in general
- Few exceptions

**2-SAT:** Each clause has 2 literals

\[ x \lor y = \{ \overline{x} \Rightarrow y \} = \{ \overline{y} \Rightarrow x \} \]

\( \Rightarrow \) representation as a directed graph

\( \Rightarrow \) Problem reduces to finding SCCs

**Horn-SAT:** Clauses have at most one positive literal

1) \( C = \{ \overline{x}_1 \lor \cdots \lor \overline{x}_a \} \) "pure negative"

2) \( C = \{ \overline{x}_1, \cdots, \overline{x}_a \} \lor x \Rightarrow x \) "Implications"

2a) \( C = \{ x \lor y \} = \{ \Rightarrow x \lor y \} \) "Unit clause implications"
**Notation:**

$C_1, C_2, \ldots$ instead of $C_1 \land C_2 \land \ldots$

**Greedy:**

- set all variables to $F$
- change variable if we are forced to

**Example:**

$(w \land y \land z) \Rightarrow x, (x \land y) \Rightarrow w, x \Rightarrow y, \Rightarrow x$

Start with all False

$xyzw = FFFF, TFFF, TFFF, TFFW$

If we also have the clause $\{x \lor w\}$, the formula would not be satisfiable.

**Horn ($F$)**

- Set $x_1, \ldots, x_n$ to False
- While there is a non satisfied implication clause $C$,
  - set right hand variable in $C$ to True
- If all purely negative clause are satisfied
  - return assignment
- Else: Return "F not satisfiable"
Correctness

Claim: If Horn(F) sets \( x = T \), then \( x = T \) in all satisfying assignments.

Proof by induction

\( N = \# \) of variables set to \( T \)

\( N=1 \): There must have been or clause \( C = \neg x \)

\[ \Rightarrow x = T \] in all sat. ass.

\( N \to N+1 \): Assume \( x_1, \ldots, x_N \) are set to \( T \) in \( \text{Horn} \) and are true in all sat. assignments.

**Case 1:** Horn finds no further clause which is unsat \( \Rightarrow \) we are done

**Case 2:** \( \exists \) clause

\[ x_{i_1} \land \ldots \land x_{i_e} \Rightarrow x_e \]

\[ x_{i_1}, \ldots, x_{i_e} = T \] in \( \text{Horn} \)

\[ \downarrow \]

\( \Rightarrow x_e = T \)

\[ \downarrow \]
Proof of Correctness:

- If Horn finds satisfiable assignment
  \( \Rightarrow \) there exists \( \exists \)

- If Horn outputs No

  \( \Rightarrow \exists \) pure clause

  \( C = (\overline{x_1} \lor \cdots \lor \overline{x_a}) = \text{unsat.} \)

  \( \Rightarrow \) Horn has set \( x_1 = \cdots x_a = T \)

  \( \Rightarrow x_1 = \cdots x_a = T \) in all sat. ass.

  \( \Rightarrow F \) is not satisfiable
Running Time

Set $x_1, \ldots, x_n$ to False

While some satisfied implication clause $C$,
set right hand variable in $C$ to True

If all purely negative clause are satisfied
return assignment
Else: Return "Not satisfiable"

$m_+ = \# \text{ of implication clauses}$
While loop runs $\leq m_+$ times

Each run take

$$\leq \sum_{i=1}^{m_+} |C_i| \leq |F| = \sum_{C \in F} |C|$$

⇒ quadratic running time $O(m|F|)$

Checking negative clauses

$$\sum_{j=m+1}^{m} |C_j| \leq |F|$$
Graph Representation

Variable nodes \( \bullet \), clause nodes \( \square \)

\[ C = \{(x_1 \land x_2 \land \cdots \land x_k) \Rightarrow y_j\} \]

When we set a variable \( x \) to true

- delete edges going out from \( x \)
- remove \( x \) from clauses
- if clause becomes
  \[ \{\Rightarrow y_j = 1 \}
  \]
  put \( y \) in Queue to be set to True

\( Q \) receives \( \leq m \) injects

\# of updates for the clauses

\[ \leq |E| \leq |F| \Rightarrow \text{linear time!} \]
Algorithms so Far

Divide & Conquer:

- Integer Multiplication $O(n \log_2^3) = O(n^{1.58})$
- Matrix Multiplication $O(n \log_2^2) = O(n^{2.81})$
- Merge Sort $O(n \log n)$
- FFT $O(n \log n)$

Simple Graph Algorithms

- DFS, connected components $O(n+m)$ $n=|V|$, $m=|E|$
- topological search, SCC $O(n+m)$

Single Source Shortest Paths

- DFS $O(n+m)$
- Dijkstra $O((n+m) \log n)$
- Bellman-Ford $O(nm)$
- DAG-SSSP $O(n+m)$

Greedy

- Scheduling $O(n)$
- Huffman Coding $O(n \log n)$
- TWS Tree (Kruskal & Prim) $O((n+m) \log n)$
- Horn Formulae $O(1F1)$
- Greedy Set Cover (later, only find approx. min)

- Important basic algorithms, fast
- Not a very general tool

Dynamic Programming

- Very powerful, versatile tool