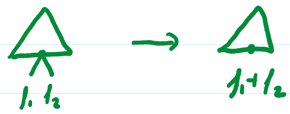
Review:

Greedy Algorithms:

1) Scheduling

2) Huffman Codes

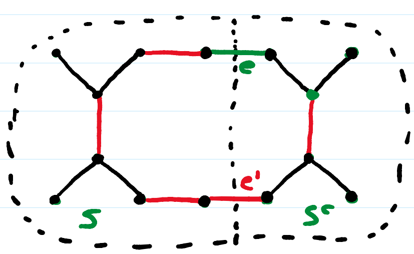


3) Minimum Spanning Trees (MST)

The Cut Property:

Let $S \subseteq V$, and let $X \subseteq E$ be part of a MST T s.th. 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'



$T = \{ -, - \}$

$T' = T \cup \{e\} \setminus \{e'\}$

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

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

Kruskal's Alg: Order edges by weight, and in each step add next edge which does not create a cyle

Prim(G, w)

$\forall u \in V$ cost(u) $= \infty$, prev(u) = nil
Pick any $u_0 \in V$
cost(u_0) = 0
$\forall v \in V$ insert key (v, cost(v))
while queue non empty
  v = Delete Min
  $\forall \{v,u\} \in E$
    if cost(u) > w(v,u)
      cost(u) = w(v,u)
      prev(u) = v
      DecreaseKey(u)

$n = |V|$, $m = |E|$

O(n) insert, delete
O(m) decrease key

Running Time

$O((n+m)\log n)$

Union Find Data Structure:

- makeset(x) makes singleton containing x
- Find(x) which set does x belong to
- union(x,y) merges sets contain. x,y

Kruskal(G, w)

For all $v \in V$ makeset(v)
$X = \{\}$
Sort edges in E by w(·)
$\forall \{u,v\} \in E$ in that order
  if Find(u) $\neq$ Find(v)
    add {u,v} to X
    union(u,v)
return X

n	makeset	x	O(1)
2m	Find	x	O(log n)
n−1	union	x	O(log n)

+ sort $O(m \log m)$

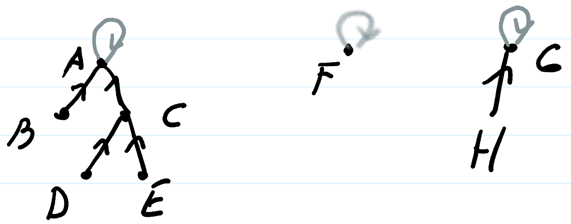
Running Time

$O((m+n)\log n)$

Union Find Data Structure

We need data structure for finite sets

Choose trees, label set by its root



$\pi(x) =$ "parent of x"

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

$\{A, B, C, D, E\}$; $\{F\}$; $\{G, H\}$

makeset(x)

$\pi(x) = x$

$\text{rank}(x) = 0$

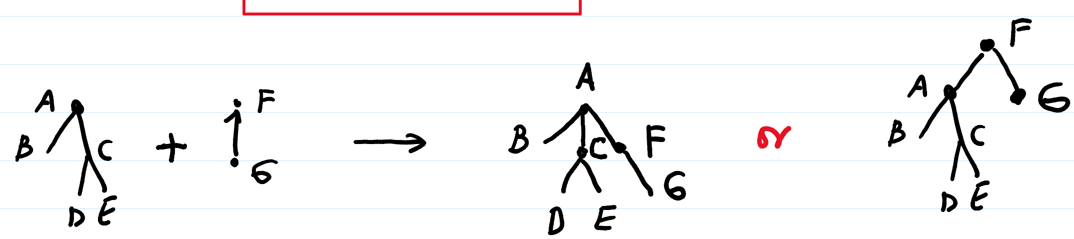
find(x)

while $\pi(x) \neq x$

  $x = \pi(x)$

return x

Union:



deeper trees make find(x) slower

union(x,y)

$r_x = \text{find}(x)$ $r_y = \text{find}(y)$

if $r_x = r_y$: return

if $\text{rank}(r_x) \leq \text{rank}(r_y)$

  $\pi(r_x) = r_y$

  if $\text{rank}(r_x) = \text{rank}(r_y)$:

    $\text{rank}(r_y) = \text{rank}(r_x) + 1$

else: $\pi(r_y) = r_x$

Example 1:

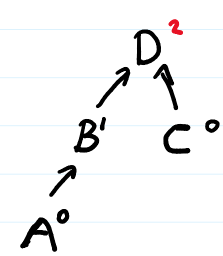
makeset(A), … makeset(D)

A^0, B^0, C^0, D^0 ← - - - rank

union(A, B): $\begin{array}{c} B^1 \\ \uparrow \\ A^0 \end{array}$ union(C, D): $\begin{array}{c} D^1 \\ \uparrow \\ C^0 \end{array}$

union(A, C):

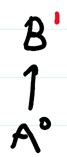
$\begin{array}{c} B^1 \\ \uparrow \\ A^0 \end{array} + \begin{array}{c} D^1 \\ \uparrow \\ C^0 \end{array} \longrightarrow$

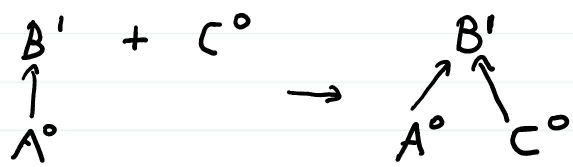


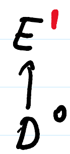
Example 2:

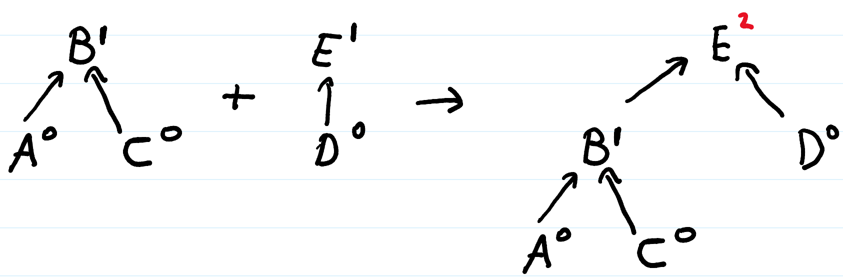
makeset(A), … makeset(E°)

A°, B°, C°, D°, E°

union(A, B): 

union(A, C): 

union(D, E) 

union(A, D): 

Running Time:

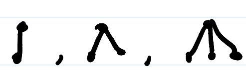
makeset: $O(1)$ find: $O(\text{rank of root})$

union: $O(\text{rank root})$

Claim: If rank(x) = k and x is the root

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

Pf of Claim

$k=0$ $k=1$ 

$k \to k+1$ The only way to create a rank $k+1$ tree is to merge two rank k trees ∎

Corollary: If we have n element, the rank is allways $\leq \lfloor \log_2 n \rfloor$

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

4) Horn Formulae

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

SAT-Formula:

Any expression F that can be obtained from $x_1, \cdots x_n$ by iteratively applying AND, OR or NOT $(\wedge, \vee, \bar{\cdot})$

Eample:
- a the weather is nice
- b I am inside
- c my NeurIPS paper was rejected
- d You are happy with my teaching

$$F = (a \vee b) \wedge \bar{c} \wedge d$$

More complicate example

$$F = \overline{\overline{(a \vee b) \wedge \bar{c} \wedge d} \wedge (\overline{a \vee b})}$$

Conjunctive Normal Form

$L = \{x_1, \bar{x}_1, \cdots, x_n, \bar{x}_n\}$ Literals

positive L.

negative L.

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

$$F = C_1 \wedge \cdots \wedge C_m$$

where each clause C_i is an OR of literals.

Proof by induction

$F = F_1 \wedge F_2$ ✓

$F = F_1 \vee F_2$ $\quad F_1 = \bigwedge_{i=1}^{m} C_i \quad F_2 = \bigwedge_{j=1}^{m'} D_j$

$= \bigwedge_{i,j} (C_i \vee D_j)$ ✓

$F = \overline{C_1 \wedge \cdots \wedge C_m} = \bigvee_{i=1}^{m} \overline{C}_i$

$C_i = \ell_1 \vee \cdots \vee \ell_k$

$\overline{C}_i = \overline{\ell}_1 \wedge \cdots \wedge \overline{\ell}_k$ ■

SAT-Problem: Given a CNF-Formula F, find True, False assignments to the variables s.th. 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

$x \vee y = \{\overline{x} \Rightarrow y\} = \{\overline{y} \Rightarrow x\}$

→ Representation as a directed graph

→ Problem reduces to finding SCCs

Horn-SAT: Clauses have at most one positive literal

1) $C = \{\overline{x}_1 \vee \cdots \vee \overline{x}_k\}$ "pure negative"

2) $C = \{\overline{x}_1 \vee \cdots \vee \overline{x}_k \vee x\} = \{(x_1 \wedge \cdots \wedge x_k) \Rightarrow x\}$ "Implications"

2a) $C = \{x\} = \{\Rightarrow x\}$ "Unit Clause Implications"

Notation:

$C_1, C_2, \cdots$ instead of $C_1 \wedge C_2 \wedge \cdots$

Greedy:

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

Example:

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

Start with all False

$xyzw = FFFF, \ TFFF, \ TTFF, \ TTFW$

If we also had the clause $\{\bar{x} \vee \bar{w}\}$, the formula would not be satisf.

Horn (F)

Set $x_1, \cdots, x_n$ to False
While $\exists$ non satisfied implication clause C,
    set right hand variable in C to True
If all purely negative clause are satisfied
    return assignement
Else: Return "F not satisfiable"

Correctness

Claim: If Horn(F) sets $x = T$, the $x = T$ in all satisfying assignments

Proof by induction

$N = \#$ of variables set to T

N=1 There must have been a clause

$C = \{x\}$ Why?

$\Rightarrow x = T$ in all satisf. ass.

N→N+1: Assume $x_1, \cdots x_N$ are set to T in Horn & are true in all sat. assignments

case 1: Horn finds no further clause which is unsat $\Rightarrow$ we are done

Case 2: $\exists$ clause

$$x_{i_1} \wedge \cdots \wedge x_{i_\ell} \Rightarrow x_\ell$$

$x_{i_1}, \cdots x_{i_\ell} = T$ in Horn

〃 in all sat. ass.

$\Rightarrow x_\ell = T$ 〃

Proof of Correctness:

- If Horn finds satisf. assignment
  $\Rightarrow$ there exists ——

- If Horn outputs No ——
  $\Rightarrow \exists$ pure clause

  $$C = (\bar{x}_1 \vee \cdots \bar{x}_k) = \text{unsat.}$$

  $\Rightarrow$ Horn has set $x_1 = \cdots x_k = T$
  $\Rightarrow$ $x_1 = \cdots x_k = T$ in all sat. ass.
  $\Rightarrow$ F is not satisfyable

Running Time

Set $x_1, \cdots, x_n$ to False
While $\exists$ non satisfied implication clause C,
    set right hand variable in C to True
If all purely negative clause are satisfied
    return assignement
Else: Return "F not satisfiable"

$m_+ = \#$ of implication clauses

While Loop runs $\le m_+$ times

Each run take

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

$\Rightarrow$ quadratic running time $O(m|F|)$

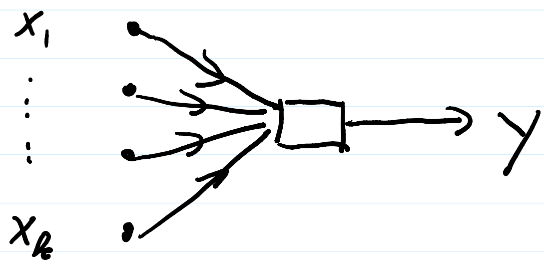
Checking negative Clauses

$$\sum_{i=m_++1}^{m} |C_i| \le |F|$$

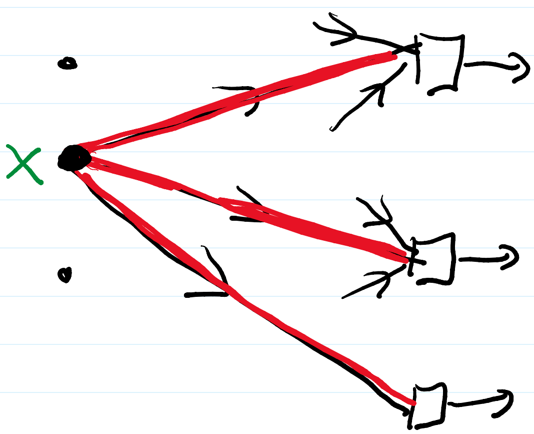
Graph Representation

Variable nodes •, clause nodes □

$$C = \{(x_1 \wedge x_2 \wedge \cdots \wedge x_k) \Rightarrow y\}$$



When we set a variable x to true



- delete edges going out from x
- remove x from clauses
- If clause becomes $\{\Rightarrow y\} = \{y\}$ put y in Queue to be set to True

Q receives $\leq m_+$ injects

$\#$ of updates for the clauses

$\leq |E| \leq |F| \Rightarrow$ linear time!

Algorithms so Far

Divide & Conquer:

Integer Multiplication $O(n^{\log_2 3}) = O(n^{1.58})$
Matrix Multiplication $O(n^{\log_2 7}) = O(n^{2.81})$
Merge Sort $O(n \log n)$
FFT $O(n \log n)$

Simple Graph Algorithms

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

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 $O(n \log n)$
MWS tree (Kruskal & Prim) $O((n+m) \log n)$
Horn Formulae $O(|F|)$
Greedy Set Cover (later, only find approx. min)

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

Dynamic Programming

Very powerful, versatile tool