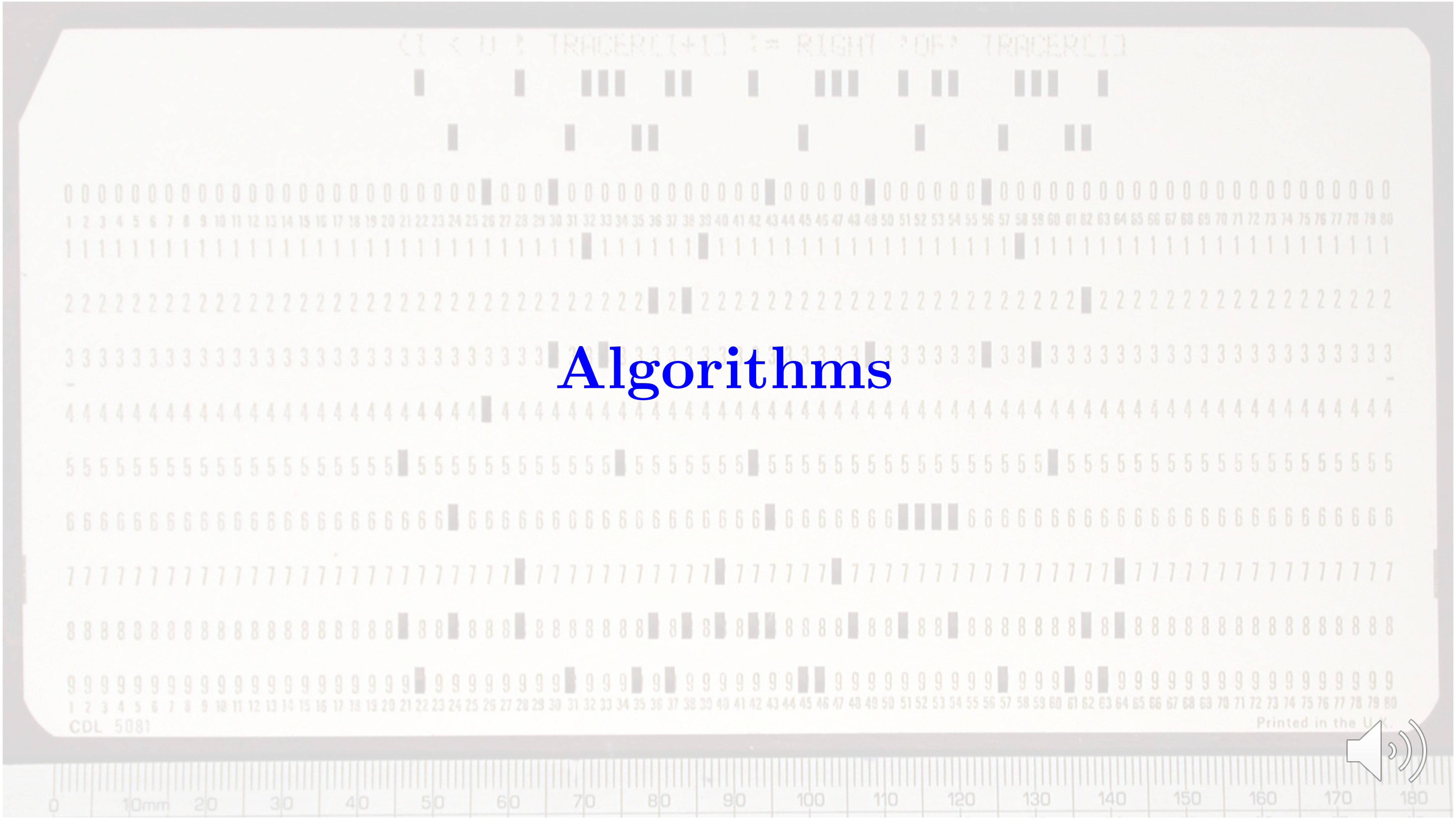


Algorithms

Single Source Shortest Paths (SSSP)

Input: Directed graph $G = (V, E)$, edge lengths c_e [possibly negative], source vertex $s \in V$.

Goal: For all destinations $v \in V$, compute the length of a shortest s-v path

Single Source Shortest Paths (SSSP)

What graphs have negative weight edges?

Imagine a road system which, instead of using distance to represent edge weight, you use change in elevation.

In this case, the weight of the edges (roads) going downhill might be represented with negative weights.

In such case, can you have negative cycles?

RECURRENCE

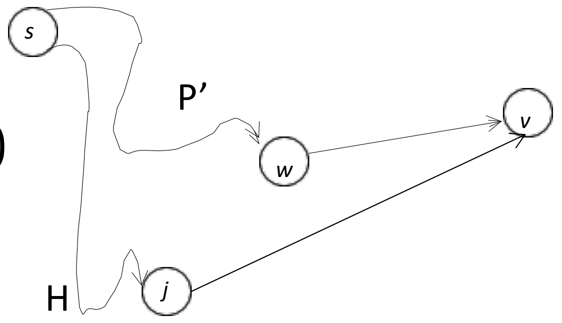
Notation: Let $L_{i,v}$ = minimum length of a s-v path with $\leq i$ edges.

- Defined as $+\infty$ if no s-v paths with $\leq i$ edges

Recurrence: For every $v \in V$, $i \in \{1,2,\ldots n\text{-}1\}$

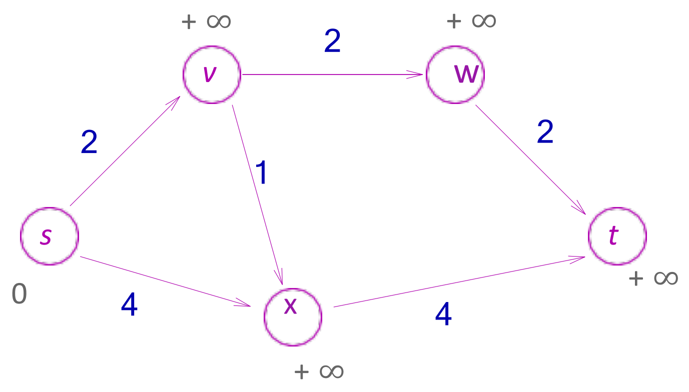
$$L_{i,v} = \min \begin{cases} L_{(i-1),v} & \text{Case 1} \\ \min_{(w,v)\in E}\{L_{(i-1),w} + c_{wv}\} & \text{Case 2} \end{cases}$$

Correctness: Brute-force search from the only (1+in-deg(v)) candidates (by the optimal substructure lemma).



Example

$$A[v, i] = \min \begin{cases} A[v, i-1] \\ \min_{(w,v)\in E}\{A[w, i-1] + c_{wv}\} \end{cases}$$



i = 0

i = 1

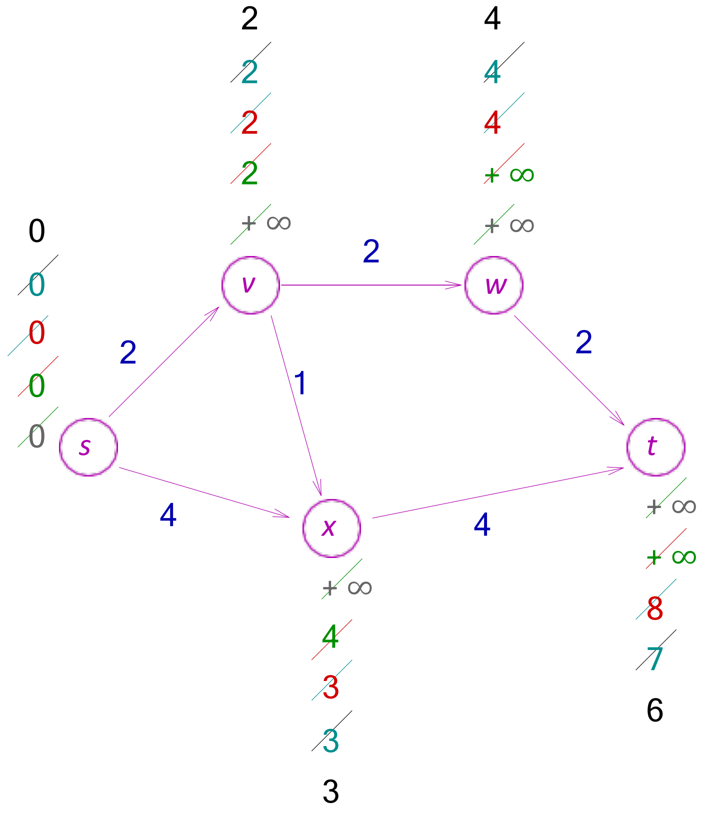
i = 2

i = 3

i = 4

Example

$$A[v, i] = \min \begin{cases} A[v, i-1] \\ \min_{(w,v)\in E}\{A[w, i-1] + c_{wv}\} \end{cases}$$



i = 0
i = 1
i = 2
i = 3
i = 4

Complexity

Question: What is the running time of the Bellman-Ford algorithm? [Pick the strongest true statement.] [m = # of edges, n = # of vertices]

A) $O(n^2)$

[✓ B)] $O(mn)$

C) $O(n^3)$

D) $O(m^2)$

Reason: Total work is $O(\underbrace{n}_{\text{\# iterations of outer loop (i.e. choices of } i)} \times \underbrace{\sum indeg(v)_{v \in V}}_{\text{work done in one iteration } = m})$ $= O(mn)$

Can we Stop Early?

Note: Suppose for some $j < n - 1$, $A[v,j] = A[v,j-1]$ for all vertices v

$\Rightarrow$ For all v, all future $A[v, i]$'s will be the same

WHY? ASSUME A[W,I+1] DENOTES THE PATH P OF LENGTH I+1, THEN THE PATH P' OF LENGTH I IA THE SHORTEST TO SOME NODE T. CONTRADICTION

$\Rightarrow$ Can safely halt (since $A[v, n-1]$'s = correct shortest-path distances)

All Pair shortest Path Problem

- Input: Directed graph $G = (V, E)$ with edge costs c_e for each edge
- $e \in E$, [No distinguished source vertex.]
- Goal: Compute the length of a shortest $u \rightarrow v$ path for all pairs of vertices $u, v \in V$

COMPLEXITY

Question: How many invocations of a single-source shortest-path subroutine are needed to solve the all-pairs shortest path problem? [n = # of vertices]

A) 1

B) $n-1$

[✓ C)] n

D) n^2

Running time (nonnegative edge costs):

$$n\cdot \text{Dijkstra} = O(nm\log n) = \begin{cases} O(n^2\log n) \text{ if } m=\Theta(n) \\ O(n^3\log n) \text{ if } m=\Theta(n^2) \end{cases}$$

Running time (general edge costs):

$$n\cdot \text{Bellman-Ford} = O(n^2 m) = \begin{cases} O(n^3) \text{ if } m=\Theta(n) \\ O(n^4) \text{ if } m=\Theta(n^2) \end{cases}$$

Practice: Problem solutions

Problem For a set of variables $x_1, \dots, x_n$, you are given some equality constraints, of the form $x_i = x_j$ and some disequality constraints, of the form $x_i \neq x_j$. Is it possible to satisfy all of them?

For instance, the constraints $x_1 = x_2$, $x_2 = x_3$, $x_3 = x_4$, $x_1 \neq x_4$ cannot be satisfied.

Describe an algorithm that takes as input m constraints over n variables and decides whether the constraints can be satisfied. Provide algorithm complexity.

Hint: One possible option is to consider a graph representation of this problem where each node is a variable (e.g. x_i) and an edge represents an "equality constraint".

Practice: Problem solutions

Following are 3 solutions from different class groups. Review these solutions and let me know if you find any problems.

Practice: Class solutions

- Idea: consider a graph representation of this problem where each node is a variable (e.g. x_i) and an edge represents an "equality constraint". If we have a constraint xi not equal to xj, then xj should not be accessible from xi (no path between them)
- Algorithm

```
Let A be a 2-dim matrix initialized to -1.
For all i,j
    Set A[i,j]=1 if there is a constraint xi≠xj or xj≠xi
    Set A[i,j]=0 if xi=xj or xj=xi
#we only need to fill half the array
For all i in 1,…,n
    Run BFS(xi) while checking for every node xj added to the BFS tree if A[i,j] or A[j,i] is 1. If so return "unsatisfiable"
Return Satisfiable
```

- Complexity

  O(nm) for the array

  O(n(n+m)) for BFS

Practice: Class solutions

algorithm:-

- have each variable be represented as a node $(1, \dots n)$ } $O(n)$
- loop through the "=" constraints, $x_p = x_q$, then add edge from the node with the lower rank to the parent of the node with the higher rank (if same, it doesn't matter and increment rank)
- loop through "≠" constraint, $x_p \neq x_q$, if parent(p) = parent(q), return false
- return true

$n \rightarrow$ variables

$q \rightarrow$ equal constraint

$p \rightarrow$ not equal constraint

= constraints

$$\left\{\begin{array}{lll} \text{makesets} & \rightarrow & O(n) \\ \text{unions} & \rightarrow & O(n-1) \\ \text{find} & \rightarrow & O(2q \log n) \end{array}\right\} \rightarrow O(n + q \log n)$$

$+$

≠ constraints

$$\left\{\begin{array}{lll} \text{finds} & \rightarrow & O(2p \log n) \end{array}\right\} \rightarrow O(2p \log n)$$

$\downarrow$

$O(n + q \log n + 2p \log n)$

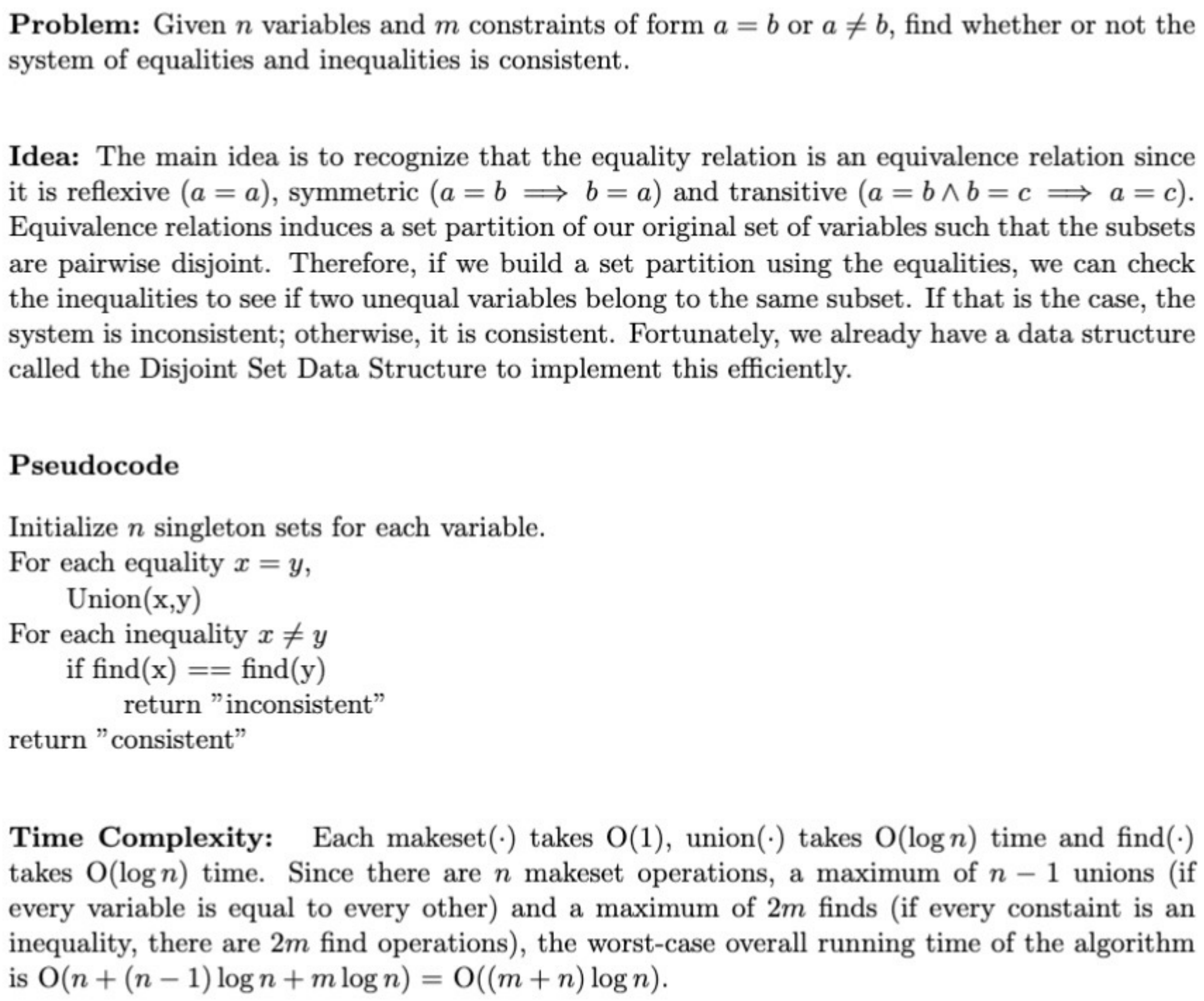
$O(n + (q+p) \log n)$

$q + p$ = number of constraints = m

$O(n + m \log n)$

correctness, if $x_p \neq x_q$, and assume set D are the variables equal to x_p and set E are the variables equal to x_q. Then, they should have different parents

Practice: Class solutions



Problem: Given n variables and m constraints of form $a = b$ or $a \neq b$, find whether or not the system of equalities and inequalities is consistent.

Idea: The main idea is to recognize that the equality relation is an equivalence relation since it is reflexive ($a = a$), symmetric ($a = b \implies b = a$) and transitive ($a = b \wedge b = c \implies a = c$). Equivalence relations induces a set partition of our original set of variables such that the subsets are pairwise disjoint. Therefore, if we build a set partition using the equalities, we can check the inequalities to see if two unequal variables belong to the same subset. If that is the case, the system is inconsistent; otherwise, it is consistent. Fortunately, we already have a data structure called the Disjoint Set Data Structure to implement this efficiently.

Pseudocode

```
Initialize $n$ singleton sets for each variable.
For each equality $x = y$,
    Union(x,y)
For each inequality $x \neq y$
    if find(x) == find(y)
        return "inconsistent"
return "consistent"
```

Time Complexity: Each makeset(·) takes O(1), union(·) takes $O(\log n)$ time and find(·) takes $O(\log n)$ time. Since there are n makeset operations, a maximum of $n - 1$ unions (if every variable is equal to every other) and a maximum of $2m$ finds (if every constaint is an inequality, there are $2m$ find operations), the worst-case overall running time of the algorithm is $O(n + (n-1)\log n + m \log n) = O((m+n)\log n)$.