Goals:

- Describe input size, runtime in terms of input size
- Improve understanding of P, NP

Questions:

- Will cover in class

NP = Nondeterministic Polynomial Time

A problem is in NP if

- Yes-No Problem
- There is a polynomial time algorithm M s.t. *(in $|x|$)* *("verifier")*
  - If x is a Yes Instance $\rightarrow \exists y$ s.t. $M(x,y) = 1$ *("witness")*
  - If x is a No Instance $\rightarrow \forall y$ $M(x,y) = 0$

P = Polynomial Time

A problem is in P if

- Yes-No Problem
- There is a polynomial time algorithm M s.t. *(in $|x|$)*
  - If x is a Yes Instance $\rightarrow M(x) = 1$
  - If x is a No Instance $\rightarrow M(x) = 0$

$|x|$ is the number of bits needed to write down instance x. "Input size"

Runtime $\rightarrow$ we care about how runtime scales with input size.

Example 3SAT instances

$X_1 = (u_1 \vee u_2) \wedge (\neg u_1 \vee \neg u_3 \vee \neg u_4) \wedge (u_2 \vee u_3 \vee u_4) \wedge (u_1 \vee \neg u_2 \vee \neg u_3) \wedge$
$(\neg u_2 \vee \neg u_3 \vee \neg u_4) \wedge (\neg u_1 \vee \neg u_3 \vee u_4) \wedge (\neg u_1 \vee u_2 \vee \neg u_3) \wedge (u_1 \vee \neg u_3)$

$X_2 = (u_1 \vee u_2 \vee u_3 \vee u_4)$

$X_3 = (u_1) \wedge (\neg u_1)$

Knapsack: (Is there a solution with value at least V?)

$X = (C_1, C_2, C_3, \ldots C_n, V_1, V_2, V_3, \ldots V_n, C, V)$ $\quad |x|$?

To write down a number m, need $\log_2 m$ bits.

But usually ignore # of bits needed to write down a number, unless runtime scales explicitly with that number

$|x| = \log_2 C_1 + \log_2 C_2 + \cdots \log_2 C_n + \log_2 V_1 \cdots + \log V_n + \log_2 C + \log_2 V$

(each term $O(1)$, except $\log_2 C \rightarrow \log_2 C$)

Runtime $= O(nC)$

$|x| = n \cdot O(1) + \log_2 C$

$|x| = O(n + \log_2 C)$

Case 1: $C = n$ $\rightarrow |x| = O(n + \log_2 n) = O(n)$

Runtime $= O(n \log n)$ $\leftarrow$ polynomial

If $C = n$, Knapsack is in P

Case 2: $C = 2^n$ $\rightarrow |x| = O(n + \log_2 2^n) = O(n+n) = O(n)$

Runtime $= O(n 2^n)$ $\leftarrow$ exponential

If $C = 2^n$, Knapsack is not known if in P.

Group Work

If $X = (x_1 \vee \neg x_2 \vee \neg x_3) \wedge \cdots \wedge (x_1 \vee x_{n-1} \vee x_n)$

- m clauses
- n variables
- 3 variables per clause

- What is $|X|$ in terms of m, n?
- Design a brute force algorithm to solve 3-SAT. What is runtime in terms of $|X|$?
- What makes 2-SAT $\in$ P? Any ideas for an algorithm?

10: Bellman-Ford (Shortest Path)

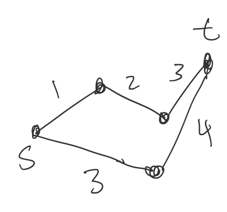
Thursday, April 15, 2021
4:52 PM

Shortest Path Problem

Input: Graph $G = (V, E)$, $w: E \rightarrow \mathbb{R}$, $s, t \in V$, $|V| = n$

Output: Shortest path from s to t in G

(Shortest $\downarrow$ sum of edge weights on path)



3 approaches

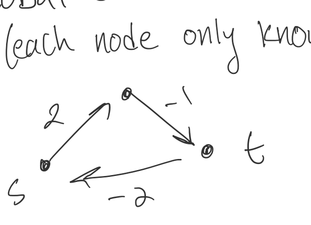
- Dynamic Programming
- Greedy
- Brute Force

} Which approach is used depends on type of graph

Bellman-Ford: used for graphs with

- directed or undirected edges
- positive + negative weights
- no negative cycles
- Global or distributed description of G (each node only knows neighbors)

Negative Cycle:



With modification: can create alg to detect neg. cycles

Applications

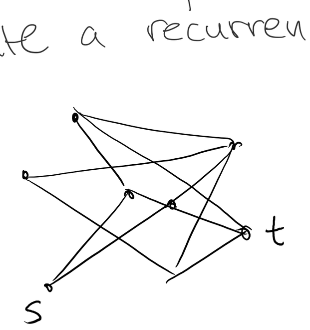
- Data routing in distributed networks
- Bartering: Apple $\xrightarrow{-1}$ orange $\xrightarrow{+1.5}$ banana
  - Can get orange and \$1 for apple
  - Can get banana for orange and \$1.50
- arbitrage = find inefficiency in currency trading market to make \$. (Neg cycle detection version)

Harm/Benefit?

Designing D.P.

Think about options for final choice in our solution/strategy.

Create a recurrence in terms of smaller subproblem + final choice.



Final Choice?