Median-Finding

~~or~~ input = ?

1. all values are same $O(n)$ to do first split

$S_L = \emptyset$ S_v $S_R = \emptyset$

Randomized Algorithms

2. all values different - go through all levels of recursion $O(n)$ using analysis from before

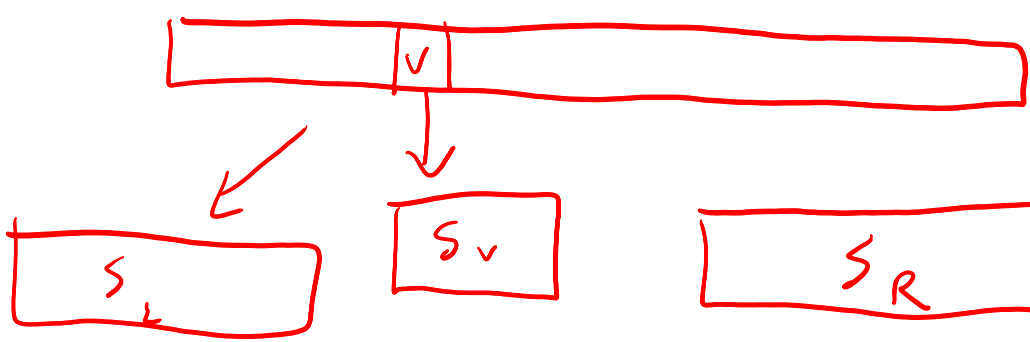
3. possibilities in between $O(n)$

Deterministic vs. Randomized Algorithms

- A deterministic algorithm always makes the same sequence of actions when given the same input

same output, same running time, etc.

- A randomized algorithm bases its behavior not just on input but also random choices



"good" v

is in 25^{th} – 75^{th} % of values

Expected Running Time

over all inputs

[0,1]

over all possible configs. of algo.

- We can define worst-case expected running time for randomized algorithms

for each input

- $T_{wc}(N) = \max\{E[T(X)] : \text{all inputs } X \text{ of size } N\}$

$T_{EWC}(N)$

have to reason about algorithm

Expected running time on a specific input

- Note how this is different from the previous definition of expected running time!

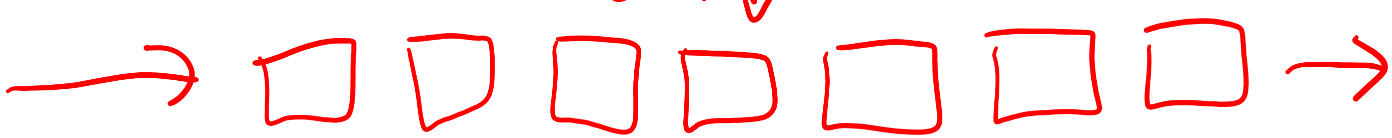
over all inputs (deterministic algorithm)

Sampling from a Stream of Items

~~Res~~ Reservoir sampling

- Suppose that you have a very long stream of objects, and you want to select 1 item from the stream in such a way that *all items have equal chance of being selected*
- You don't know how long the stream is, and it is too big to ~~store everything!~~ → can store a constant amount of information → much smaller than stream
- Ideas?

You



Reservoir Sampling

Why does this ensure that each k ∈ stream has equal prob. of $\frac{1}{N}$ being chosen?

```
def
ReservoirSampling(stream):
count = 1
for k in stream:
  if random() < 1.0/count:
    chosen = k
  count += 1
Return chosen
```

in range (0,1)

What is the probability the last element is chosen? $1/N$

Second to last? $\frac{1}{(N-1)} \cdot \frac{(N-1)}{N} = \frac{1}{N}$

Third to last? $\frac{1}{(N-2)} \cdot \frac{(N-2)}{N} = \frac{1}{N}$

⋮

The Marriage Match Problem

know M ahead of time

- Suppose that you are trying to find a spouse, and have a sequence of *M* people to choose from.
- Each candidate partner has some value to you, but you don't know the value until you spend some time dating them.

  You $\leftarrow$ $\boxed{1}$ $\boxed{2}$ $\boxed{3}$ ... $\boxed{M}$

- Rules:
  - Can only date one person at a time
  - Once you choose a spouse, you cannot go on to date other people
  - Once you break up with someone, you cannot go back to them at a later point
- Goal: Maximize the chances of selecting the highest-value spouse
  - (note that this is a slightly strange goal)

The Marriage Match Problem

$e = 2.718...$

- There is a strategy that finds the best candidate with probability at least $1/e$ (approximately 37%). Regardless of M
  - –This is amazing!
- Proof? Exercise for your homework!
- (Note: The algorithm here is not random, but the input stream is, so the strategy for analyzing has some similarities to those for randomized algorithms)

The Theory of Computational Complexity

Algorithm Running Time

- We have seen a lot of algorithms with polynomial running time
- This means that the running time is $O(n^k)$, where k is some fixed constant that does not depend on n
- Is $O(n \log n)$ polynomial?

input

algorithm is considered to have polynomial running time

$A = O(n \log n)$

$A = O(n^2)$

Algorithm Running Time

- This class is about designing efficient, correct algorithms

time, space usage

right answer

- Many techniques:
  - Divide-and-conquer
  - Greedy
  - Dynamic programming
  - Linear programming, network flows, etc.
- What is the inefficient alternative? brute force

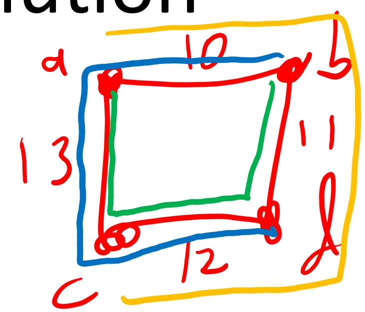
Search Problems

- A typical problem has exponentially many possible solutions
  - A graph with n vertices has up to 2^{n-2} spanning trees
  - There are up to n! possible bipartite matchings
  - In a normal graph, exponentially many paths from s to t
- Search problem: ~~Given~~ There exist exponentially many possible solutions, find one that satisfies some requirements

Search vs. Optimization

input = original input + b

- Search problem: Find solution satisfying some requirement, doesn't have to be best!
  - Find spanning tree with weight at most *b*
  - Find bipartite matching with flow greater than *b*
  - Etc.
- Optimization problem: Find *best* solution
  - Find minimum spanning tree 36
  - Find heaviest bipartite matching 35
  - Etc. 33



- Why are these basically equivalent? b = 46

Search vs. Optimization

MST problem

shortest path

search

all algs deterministic

- P (Polynomial): The class of all problems that can be solved in polynomial running time
- NP (Non-deterministic Polynomial Time): The class of search problems with solutions that can be *verified* in polynomial time.

checked

- if you have a black-box alg that claims to solve the problem, can you check whether it correctly solved the problem on a particular input?

Examples of P vs. NP: Satisfiability (SAT)

clause

negation of a literal

Formula → $(x \vee y \vee z) \wedge (x \vee \overline{y}) \wedge (y \vee \overline{z}) \wedge (z \vee \overline{x}) \wedge (\overline{x} \vee \overline{y} \vee \overline{z})$

literal

- P: The subproblem where all clauses have two literals (2SAT) is in P.
- NP: The subproblem where all clauses have three literals (3SAT) is in NP, and not P!

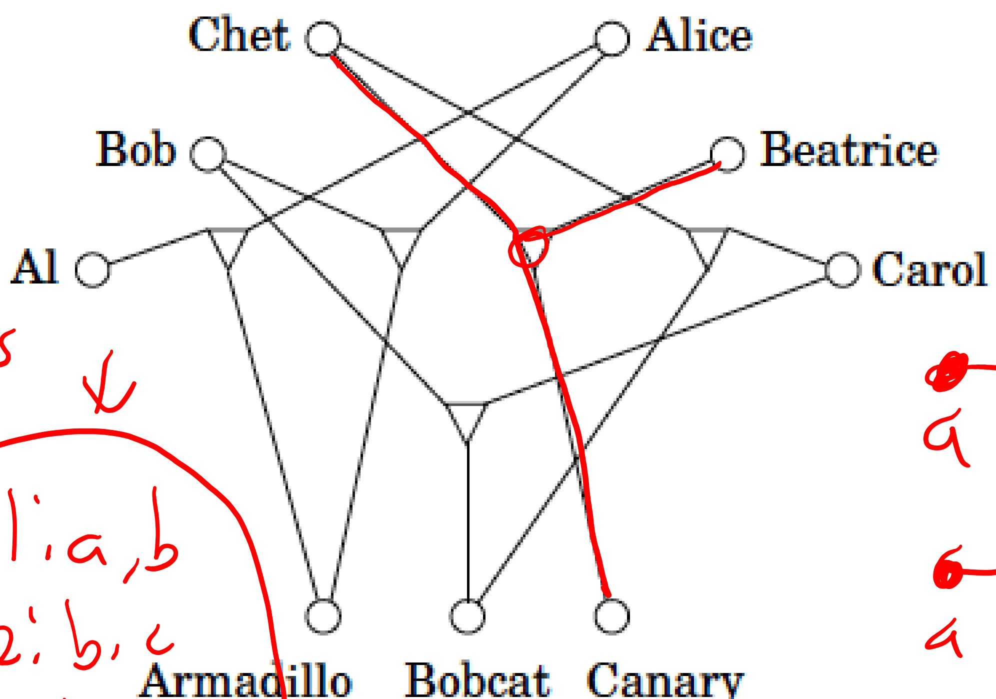
$(x \vee y) \wedge (\bar{x} \vee \bar{z})$ ← valid 2SAT input

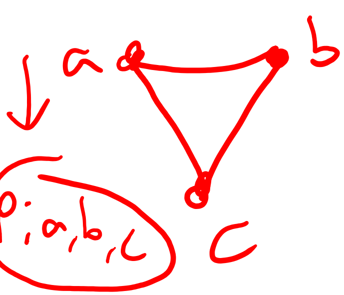
$\begin{cases} x = T \\ y = T \\ z = F \end{cases}$

Examples of P vs. NP: Network Matching

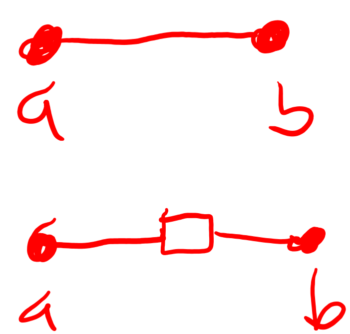
- Maximum bipartite matching is in P. → network flows
- Maximum tripartite matching, where each hyperedge (*a*, *b*, *c*) has a weight, is in NP, and not P!

a and b are co-authors





p1: a,b
p2: b,c
p3: a,c



Examples of P vs. NP: Linear Programming

→ solution can be non-integral

- General linear programming (like we saw) is in P.
- Integer linear programming is in NP, and not P!

Is P = NP?

- Whether P = NP is an open question!
- Most computer scientists think that P ≠ NP.
  assume
- But we don't have a proof…
- One of the most important open questions in computer science

P = NP?

Reductions

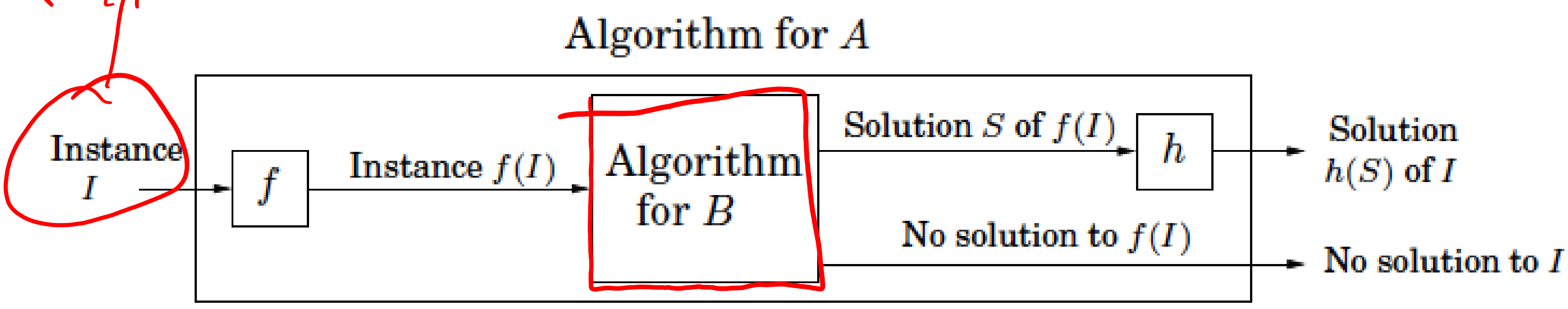
bipartite matching → NF

input

- Problem A **reduces** to Problem B if you can convert every instance of Problem A to an instance of Problem B, and convert the solution to Problem B back to the original solution, in *polynomial time*.

to prob A

For prob A



- Is reduction transitive?