Randomized Algorithms

Does the Universe Have True Randomness?

- Even if it does not, we can still model our uncertainty about things using probability.
- Randomness is an essential tool in modelling and analyzing nature.
- It plays a key role in computer science.
    - Speeding up computation: statistics via sampling
    - Cryptography: a secret is only as good as the entropy/uncertainty in it.
    - Machine Learning: data is generated by some probability distribution.

Randomness and Algorithms

- How can randomness come into the picture?
- Given some algorithms that solves a problem
  - ~~Input is chosen randomly~~
  - Algorithm can make random choices

Randomness and Algorithms

```
def f(x):
    y = Bernoulli(0.5)
    if(y == 0):
        while(x > 0):
            print(“What up?”)
            x = x - 1
    return x+y
```

For a fixed input (e.g. $x = 3$)

the output can vary

the running time can vary.

Types of Randomized Algorithms

- Randomized algorithms always gamble with either correctness or running time.

- Given an array with n elements (n even). A[1... n]. Half of the array contains 0s, the other half contains 1s.
  Goal: Find an index that contains a 1.

```
repeat:
    k = RandInt(n)
    if A[k] = 1, return k
```

doesn't gamble with correctness, gambles with running time.

```
repeat 300 times:
    k = RandInt(n)
    if A[k] = 1, return k
return "Failed"
```

gambles with correctness, doesn't gamble with running time.

Randomness and Algorithms

```
**repeat** 300 times:
    k = RandInt(n)
    **if** A[k] = 1, **return** k
**return** "Failed"
```

$$Pr[Failure] = \frac{1}{2^{300}}$$

Worst-case running time: O(1)

This is called Mont Carlo algorithm.
Gambles with correctness not time.

Randomness and Algorithms

```
repeat:
    k = RandInt(n)
    if A[k] = 1, return k
```

$$Pr[Failure] = 0$$

Worst-case running time: cannot bound

Expected running time: O(1)

This is called Las Vegas algorithm.
Gambles with time not correctness.

Types of Randomized Algorithms

- Given an array with n elements (n even). A[1 ... n]. Half of the array contains 0s, the other half contains 1s.
  Goal: Find an index that contains a 1.

	Correctness	Run-time
Deterministic	always	$\Omega(n)$
Monte Carlo	w/ high prob	$O(1)$
Las Vegas	always	$O(1)$ w/ high prob

Mont Carlo Algorithm

- Let $f : \Sigma^* \to \Sigma^*$ be a computational problem.
- Suppose A is a randomized algorithm such that

$$\forall x \in \Sigma^*, Pr[A(x) \neq f(x)] \leq \epsilon$$

$$\forall x \in \Sigma^*, \#\text{ steps A(x) takes is} \leq T(|x|)$$

- Then we say A is a T(n)-time Monte Carlo algorithm for f with ϵ probability of error.
- Example: Min Cut

Las Vegas Algorithm

- Let $f : \Sigma^* \to \Sigma^*$ be a computational problem.
- Suppose A is a randomized algorithm such that

  $\forall x \in \Sigma^*, A(x) = f(x)$

  $\forall x \in \Sigma^*$, $\mathbb{E}$[# steps A(x) takes] $\leq T(|x|)$

- Then we say A is a T(n)-time Las Vegas algorithm for f.

Quiz

- How can we make a randomized sorting algorithm based on quick sort?

Quick Sort

- On input $S = (x_1, \ldots, x_n)$
- If n=1, return S
- Pick uniformly at random a "pivot" x_m
- Compare x_m to all other x's
- Let $S_1 = \{x_i : x_i < x_m\}, S_2 = \{x_i : x_i > x_m\}$
- Recursively sort S_1 and S_2

Quick Sort

- This is a Las Vegas algorithm
    - Always gives the correct answer
    - Running time can vary depending on our luck
- The expected run time is $\leq 2n\ ln\ n = O(n\ log\ n)$

Coupon Collector Problem

- Story: "Coupon collecting" is the activity of buying cereal-packets, each of which will have a coupon inside. There are n different types of "coupon" and the goal is to collect one copy of each, then stop buying.

- How many packets do we (expect to) need to buy?

- Assumptions:

  - Items are randomly and identically distributed in packets (one card per packet). I.e. when buying a box the probability of any particular card being inside is 1/n.

How to Analyze CC Problem?

- Could we evaluate expected number of purchases to get card i (Y_i) and sum them?

CC Analysis

- Let X be a random variable defined to be the number of trials required to collect at least one of each type of coupon.
- Let $C_1, C_2, \ldots, C_X$ be the sequence of trials where $C_i \in \{1,\ldots,n\}$
- Call C_i a success if the coupon C_i was not drawn in any of the first i-1 selections. Clearly C_1 and C_X are always successes.

CC Analysis

- We divide the sequence into epochs, where epoch i begins with the trial following the ith success and ends with the trial on which we obtain the (i+1)st success.

- Define the random variable X_i, for i=0, …,n-1, to be the number of trials in the ith epoch, so that

$$X = \sum_{i=0}^{n-1} X_i$$

- Let p_i denote the probability of success on any trial of the ith epoch (the probability of drawing one of the n-i remaining coupon types

$$p_i = \frac{n-i}{n}$$

Reminder: Geometric Distribution

- The probability distribution of the number *X* of Bernoulli trials needed to get one success, supported on the set {1,2,…}.

- When the probability of success is p, the expected value for the number of independent trials to get the first success, and the variance are:

$$\mathbb{E}[X] = \frac{1}{p} \qquad Var[X] = \frac{1-p}{p^2}$$

CC Analysis

- X_i is geometrically distributed with parameter p_i
- The expected value of X_i is $\frac{1}{p_i}$ and its variance is $\frac{1-p_i}{p_i^2}$
- Recall $X = \sum_{i=0}^{n-1} X_i$
- By linearity of expectation we have

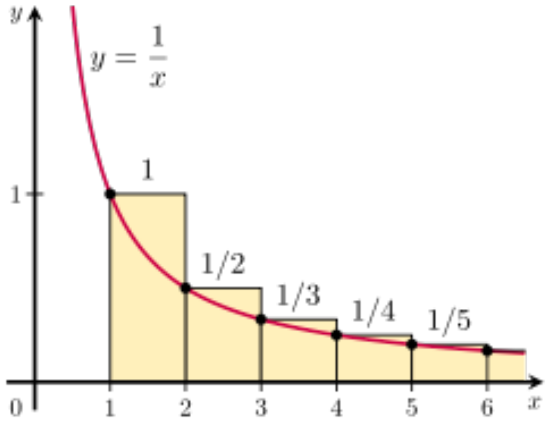
$$\mathbb{E}[X] = \sum_{i=0}^{n-1} \mathbb{E}[X_i] = \sum_{i=0}^{n-1} \frac{n}{n-i} = n \sum_{i=1}^{n} \frac{1}{i} = nH_n$$

- The nth Harmonic number H_n is asymptotically equal to $ln(n) + \Theta(1)$, implying that

$$\mathbb{E}[X] = nln(n) + O(n)$$

Reminder: Harmonic Sum

$$H_n = 1 + \frac{1}{2} + \ldots + \frac{1}{n} \approx \int_1^n \frac{1}{x} dx = ln(n)$$



- A good approximation is H(n) ≈ ln(n) +γ where γ ≈ 0.58 (Euler-Mascheroni constant).

CC Analysis

- Since the X_i's are independent, we can determine the variance of X as

$$\sigma_X^2 = \sum_{i=0}^{n-1} \sigma_{X_i}^2 = \sum_{i=0}^{n-1} \frac{ni}{(n-i)^2} = \sum_{i=1}^{n} \frac{n(n-i)}{i^2} = n^2 \sum_{i=1}^{n} \frac{1}{i^2} - nH_n$$

Beyond this Analysis

- In analyzing the performance of a randomized algorithm, we often like to show that the behavior of the algorithm is good almost all the time. For example, it is more desirable to show that the running time is small with high probability, not just that it has a small expectation.

- The next step (not covered here) is to derive sharper estimates of the typical value of X. More precisely, we would like to show that the value of X is unlikely to deviate far from its expectation, or is sharply concentrated around its expected value.

Min-Cut Problem

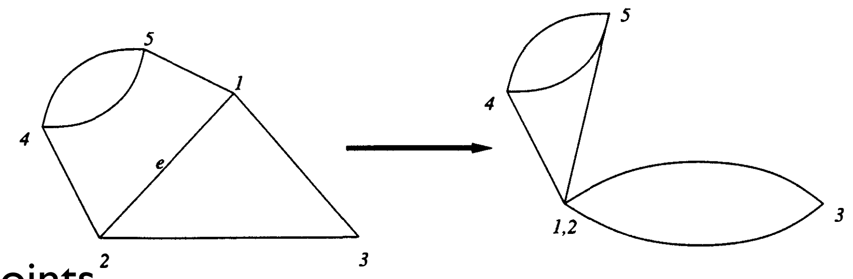
Does the Universe Have True Randomness?

- Even if it does not, we can still model our uncertainty about things using probability.
- Randomness is an essential tool in modelling and analyzing nature.
- It plays a key role in computer science.
    - Speeding up computation: statistics via sampling
    - Cryptography: a secret is only as good as the entropy/uncertainty in it.
    - Machine Learning: data is generated by some probability distribution.

Min-Cut Problem

- Let G be a connected, undirected multi-graph with n vertices. A cut in G is a set of edges whose removal results in G being broken into two or more components. A min-cut is a cut of minimum cardinality.

A Simple Algorithm for Min-Cut



- Repeat the following steps
  - pick an edge uniformly at random
    and merge the two vertices at its end-points.
  - Retain all edges between pairs of newly formed vertices.
  - Remove edges between vertices that are merged (contraction process).
    - With each contraction, the number of vertices of G decreases by one.
    - An edge contraction does not reduce the min-cut size in G. Why?
- Continue until only two vertices remain; at this point, the set of edges between these two vertices is a cut in G and is output as a candidate min-cut.
- Easier than the deterministic algorithms based on the network-flow.

A Simple Algorithm for Min-Cut

- Does this algorithm always find a min-cut?
- Let k be the size of a min-cut C for graph G.
    - G has at least kn/2 edges. Why?
- We will bound from below the probability that no edge of C is ever contracted during an execution of the algorithm, so that the edges surviving till the end are exactly the edges in C.

Analysis of Min-Cut Algorithm

- O_i : the event of not picking an edge of C at the ith step, for $1 \leq i \leq (n-2)$
- What is the maximum probability that the edge randomly chosen in the first step is in C?

Analysis of Min-Cut Algorithm

- O_i : the event of not picking an edge of C at the ith step, for $1 \leq i \leq (n-2)$
- Probability that the edge randomly chosen in the first step is in C is at most k/(nk/2) = 2/n, so that $P(O_1) \geq \frac{2}{n}$
- If O_1 occurs during the first step, during the second step there are at least k(n-1)/2 edges, so the probability of picking an edge in C is at most 2/(n-1), so that $P(O_2 | O_1) \geq (1 - \frac{2}{n-1})$
- At the ith step, the number of remaining vertices is n-i+1. The size of the min-cut is k, so the graph has at least k(n-i+1)/2 edges.

$$P(O_i | \cap_{j=1}^{i-1} O_j) \geq 1 - 2/(n-i+1)$$

Review: Probability of (In)dependent Events

- Independent events

$$P(O_1 \cap O_2) = P(O_1) \times P(O_2)$$

- Convenient way to compute the intersection probability of a collection of events that is not independent:

$$P(O_1 \cap O_2) = P(O_1 \mid O_2) \times P(O_2) = P(O_2 \mid O_1) \times P(O_1)$$

$$P(\cap_{i=1}^{k} O_i) = P(O_1) \times P(O_2 \mid O_1) \times P(O_3 \mid O_1 \cap O_2) \times \ldots \times P(O_k \mid \cap_{i=1}^{k-1} O_i)$$

Analysis of Min Cut Algorithm

- The probability of no edge of C is ever picked

$$P(\cap_{j=1}^{n-2} O_i) \geq \Pi_{i=1}^{n-2}\left(1 - \frac{2}{n-i+1}\right) = \frac{2}{n(n-1)}$$

- The probability of discovering a particular min-cut is larger than $\frac{2}{n^2}$
- The algorithm may err in declaring the cut it outputs to be a min-cut.
- Suppose we were to repeat the above algorithm $\frac{n^2}{2}$ times, making independent random choices each time. The probability that a min-cut is not found in any of the attempts

$$\left(1 - \frac{2}{n^2}\right)^{\frac{n^2}{2}} < 1/e$$

- Further executions will make the failure probability arbitrarily small but increases the running time.

Analysis of Min-Cut Algorithm

- We were able to bound the probability of an incorrect solution.
- Monte Carlo algorithm: if the algorithm is run repeatedly with independent random choices each time, the failure probability can be made arbitrarily small, at the expense of running time.

Stable Marriage Problem

Does the Universe Have True Randomness?

- Consider a society in which there are n men (A,B,C, ...) and n women (a,b,c, …).
- A monogamous, heterosexual marriage M is a 1-1 correspondence between men and women.
- Each person has a preference list organized in a decreasing order of desirability.
- A marriage is said to be unstable if there exist two married couples X-x and Y-y such that X desires y more than x, and y desires X more than Y.
  - The pair X-y is said to be dissatisfied under this marriage.
- A marriage M in which there are no dissatisfied couples is called a stable marriage.

Stable Marriage: Example

- Consider the following preference lists.
    - A:abcd B:bacd C:adcb D:dcab
    - a:ABCD b:DCBA c:ABCD d:CDAB
- Is the marriage M given by A-a, B-b, C-c, and D-d stable?

Stable Marriage: Example

- Consider the following preference lists.
    - A:abcd B:bacd C:adcb D:dcab
    - a:ABCD b:DCBA c:ABCD d:CDAB
- C-d is a dissatisfied couple, thus, M is unstable. However, if C and d marry each other, and c and D marry each other, we obtain the stable marriage A-a, B-b, C-d, D-e.
- The stable marriage problem has applications matching medical graduates to residency positions in hospitals.

An Algorithm for Stable Marriage Problem

- It can be shown that for every choice of preference lists there exist at least one stable marriage.

Proposal Algorithm

- Assume that the men are numbered in some arbitrary manner.
- The lowest numbered unmarried man X proposes to the most desirable woman on his list who has not already rejected him (aka x).
- x will accept the proposal if she is currently unmarried, or if her current mate Y is less desirable to her than X.
- Repeat; terminate when every person has been married.
- Partial marriage at each stage.
- Does this algorithm always terminates with a stable marriage? Why?

Proposal Algorithm

- The final marriage M is stable.

- Let X-y be a dissatisfied pair, where in M they are paired as X-x and Y-y. Since X prefers y to x, he must have proposed to y before getting married to x. Since y either rejected X, or accepted him only to jilt him later, Y must be more desirable to her than X. Therefore, y must prefer Y to X, contradicting the assumption that y is dissatisfied.

Average-case Analysis of

- What is the maximum number of proposals between n men/ women?

- T_P : the average number of proposals made during the execution of the Proposal Algorithm.

    - How to analyze T_P?

Average-case Analysis of

- What is the maximum number of proposals between n men/ women?
- T_P : the average number of proposals made during the execution of the Proposal Algorithm.
    - Extremely difficult to analyze
    - The choice of the proposer at any step is conditioned by the history of the process. The choice of the woman at each step also depends on the past proposals of the current proposer.

Principle of Deferred Decision

- The idea is to not assume that the entire set of random choices is made in advance. Rather, at each step of the process we consider only the random choices that must be revealed to the algorithm.

- This principle can be used to simplify the average-case analysis of the Proposal Algorithm.

Clock Solitaire

- In this game we start with a standard deck of 52 cards, randomly shuffled.
- The pack is divided into 13 piles of 4 cards each.
- Each pile is arbitrarily labeled with a distinct member of {A,2,3,…,J,Q,K}.
- On the first move we draw a card from the pile labeled K.
- At each subsequent move, a card is drawn from the pile whose label is the face value of the card drawn at the previous move.
- The game ends when an attempt is made to draw a card from an empty pile.
- We win the game only if, on termination, all 52 cards have been drawn.

Clock Solitaire: Prob of Winning

- What is the probability of winning the game?

Clock Solitaire: Prob of Winning

- The game always terminates in an attempt to draw a card from the K pile
  - The last card drawn has to be a K, because there are 4 cards of each denomination, and except for the K pile, each pile initially has 4 cards.
- A naive view of the probability space: all possible ways of dealing out the cards. Each point in this space corresponds to some partition of the 52 cards into 13 distinct piles, with an ordering defined on the 4 cards in each pile.
  - At each move of the game we introduce a new source of dependency.

Clock Solitaire: Prob of Winning

- The idea of the Principle of Deferred Decisions is to not assume that the entire set of random choices is made in advance. Rather, at each step of the process we fix only the random choices that must be revealed to the algorithm.

- At each draw any unseen card is equally likely to appear. Thus, the process of playing this game is equivalent to repeatedly drawing a card uniformly at random from a deck of 52 cards.

- A winning game corresponds to the situation where the first 51 cards drawn in this fashion contain exactly 3 Kings.

- The probability of the 52nd card drawn being a King is exactly 1/13. This is also the probability of winning the game.

    - Recall the game always terminates in an attempt to draw a card from the K pile.

Proposal Algorithm: Average Case

- We do not assume that the men have chosen their (random) preference list in advance.
- Suppose that men do not know their lists to start with. Each time a man makes a proposal, he chooses a woman uniformly at random from the set of all n women, including those to whom he has already proposed. (Amnesiac Algorithm — T_A: the number of proposals)
- There are some wasted proposals in the Amnesiac Algorithm.
  - T_A stochastically dominates T_P : for all m, $Pr[T_A > m] >= Pr[T_P > m]$. For an upper bound, analyze the distribution of T_A.

Proposal Algorithm: Average Case

- To analyze T_A we need to only count the total number of proposals made.
- Each proposal is independently made to one of the n women chosen uniformly at random.
- The algorithm terminates with a stable marriage once all women have received at least one proposal each.
- Does this remind you of a problem we learned before?