More Probability: Recurrences

Admin

- Updated problem 4b to be more to the point—make sure you have the most recent version
- Assignment 8 due Friday
- Don’t update your Macs just yet probably

Card Guessing: Memoryless

- To entertain your family you have them shuffle deck of n cards and then turn over one card at a time. Before each card is turned, you predict its identity. You have no psychic abilities or memory to remember cards
- Your strategy: guess uniformly at random
- How many predictions do you expect to be correct?
- Let X denote the r.v. equal to the number of correct predictions and X_i denote the indicator variable that the ith guess is correct
- Thus, $X = \sum_{i=1}^{n} X_i$ and $\mathrm{E}[X] = \mathrm{E}[\sum_{i=1}^{n} X_i] = \sum_{i=1}^{n} \mathrm{E}[X_i]$
- $\mathrm{E}[X_i] = 0 \cdot \Pr(X_i = 0) + 1 \cdot \Pr(X_i = 1) = \Pr(X_i = 1) = 1/n$
- Thus, $\mathrm{E}[X] = 1$

Card Guessing: Memoryfull

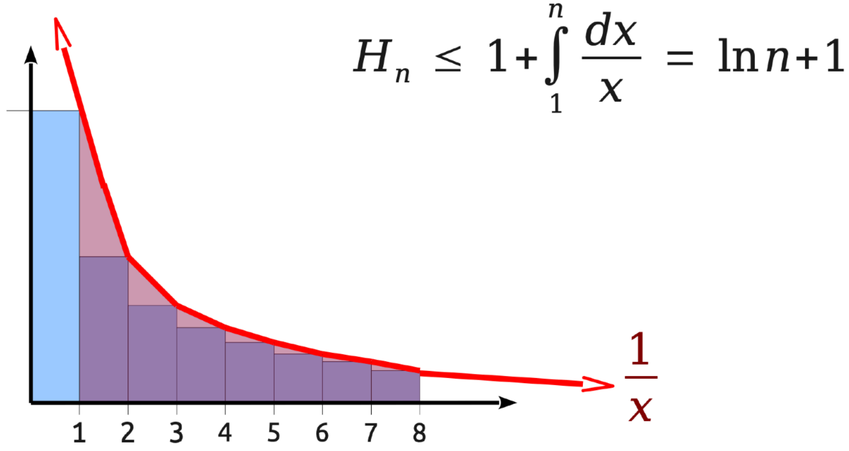
- Suppose we play the same game but now assume you have the ability to remember cards that have already been turned
- Your strategy: guess uniformly at random among cards that have not been turned over
- Let X denote the r.v. equal to the number of correct predictions and X_i denote the indicator variable that the ith guess is correct
- Thus, $X = \sum_{i=1}^{n} X_i$ and $\mathrm{E}[X] = \mathrm{E}[\sum_{i=1}^{n} X_i] = \sum_{i=1}^{n} \mathrm{E}[X_i]$
- $\mathrm{E}[X_i] = \Pr(X_i = 1) = \frac{1}{n - i + 1}$
- Thus, $\mathrm{E}[X] = \sum_{i=1}^{n} \frac{1}{n - i + 1} = \sum_{i=1}^{n} \frac{1}{i}$

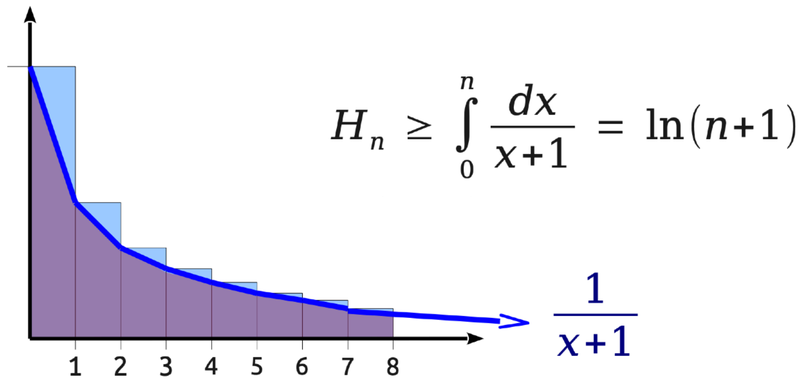
Harmonic Numbers

- The nth harmonic number, denoted H_n is defined as

$$H_n = \sum_{i=1}^{n} \frac{1}{i}$$

- **Theorem**. $H_n = \Theta(\log n)$
- Proof Idea. Upper and lower bound area under the curve





Card Guessing: Memoryfull

- Suppose we play the same game but now assume you have the ability to remember cards that have already been turned
- Your strategy: guess uniformly at random among cards that have not been turned over
- Let X denote the r.v. equal to the number of correct predictions and X_i denote the indicator variable that the ith guess is correct
- Thus, $X = \sum_{i=1}^{n} X_i$ and $E[X] = E[\sum_{i=1}^{n} X_i] = \sum_{i=1}^{n} E[X_i]$
- $E[X_i] = \Pr(X_i = 1) = \frac{1}{n - i + 1}$
- Thus, $E[X] = \sum_{i=1}^{n} \frac{1}{n - i + 1} = \sum_{i=1}^{n} \frac{1}{i} = \Theta(\log n)$

Probability and Recurrences

Bernoulli Expectation

- Let's say a sequence random variables $X_1, X_2, \ldots$ is 1 with probability p, and 0 with probability $1 - p$
- (Recall: these are called **Bernoulli random variables**)
- In expectation, what is the value of the first i such that $X_i = 1$?
    - Number of coin flips until heads $(p = 1/2)$
    - Number if times I roll a die until I get a 1 $(p = 1/6)$

- One way to solve it is to just do the sum:

$$\sum_{i=1}^{\infty} i(1-p)^{i-1}p$$

Bernoulli Expectation (using the sum)

$$\sum_{i=1}^{\infty} i(1-p)^{i-1}p = \sum_{i=1}^{\infty}\sum_{k=1}^{i}(1-p)^{i-1}p =$$

$$\sum_{k=1}^{\infty}\sum_{i=k}^{\infty}(1-p)^{i-1}p = \sum_{k=1}^{\infty}p(1-p)^{k-1}\sum_{i=0}^{\infty}(1-p)^{i} =$$

$$\sum_{k=1}^{\infty}p(1-p)^{k-1}\frac{1}{1-(1-p)} = \sum_{k=1}^{\infty}(1-p)^{k-1} = \sum_{k=0}^{\infty}(1-p)^{k} = \frac{1}{p}$$

Bernoulli Expectation

- Let's say a sequence random variables $X_1, X_2, \ldots$ is 1 with probability p, and 0 with probability $1 - p$
- In expectation, what is the value of the first i such that $X_i = 1$?
- Let's rewrite this recursively. What is $\mathrm{E}(\text{FindBernoulli}(p))$?

FindBernoulli(p):

$$X \leftarrow \begin{cases} 1 \text{ with prob. } p \\ 0 \text{ with prob. } (1-p) \end{cases}$$

If $X = 1$

Return 1

If $X = 0$

Return $1+$ FindBernoulli(p)

Bernoulli Expectation

FindBernoulli(p):

$$X \leftarrow \begin{cases} 1 \text{ with prob. } p \\ 0 \text{ with prob. } (1-p) \end{cases}$$

If $X = 1$

Return 1

If $X = 0$

Return $1+$ FindBernoulli(p)

$$\mathrm{E}(F) = p + (1-p)(1 + \mathrm{E}(F))$$

$$\mathrm{E}(F) = 1/p$$

Bernoulli Expectation Formal Recursion

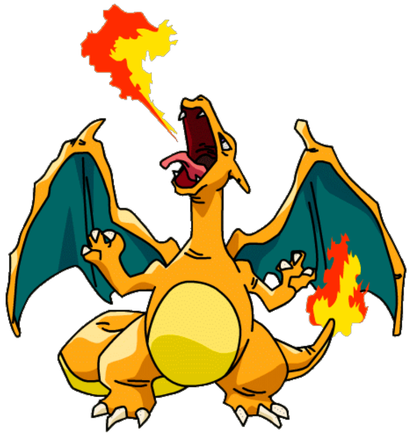
You don't need to do this proof every time you use recursion. But, it can help if you're unsure of correctness

- Let X^p be a random variable indicating # flips until heads (with prob p)
- $$\mathrm{E}(X^p) = \sum_{i=1}^{\infty} i(1-p)^{i-1}p$$
- We can then write
- $$\mathrm{E}(X^p) = \sum_{i=1}^{\infty} i(1-p)^{i-1}p = p + \sum_{i=2}^{\infty} i(1-p)^{i-1}p = p + \sum_{i'=1}^{\infty} (1+i')(1-p)^{i'}p$$
- $$\mathrm{E}(X^p) = p + (1-p)\sum_{i'=1}^{\infty} (1+i')(1-p)^{i'-1}p = p + (1-p)\mathrm{E}(X^p+1)$$
- $$\mathrm{E}(X^p) = p + (1-p)(\mathrm{E}(X^p)+1)$$

Coupon/Pokemon Collector Problem

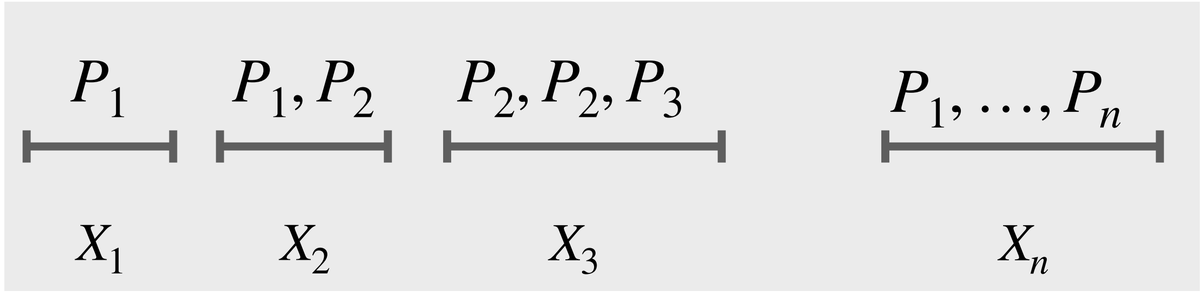
Gotta' Catch 'Em All

- Suppose there are n different types of Pokemon cards
- In each trial we purchase a pack that contains a Pokemon card
- We repeat until we have at least one of each type of card, how many packs does it take in expectation to collect all?
- Let X be the r.v. equal to the number of packs bought until you first have a card of each type. **Goal**: compute $E[X]$
- We break X into smaller random variables
- **Idea**: we make progress every time we get a card we don't already have

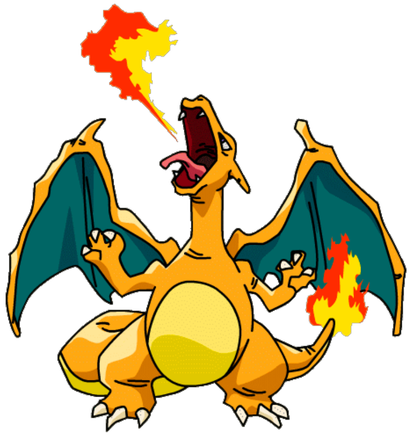


Pokemon Collector Problem

- Let X_i denote the "length of the ith phase", that is, the number of packs bought during the ith phase (ith phase ends as soon as we see the ith distinct card)
- Thus, $X = \sum_{1=1}^{n} X_i$

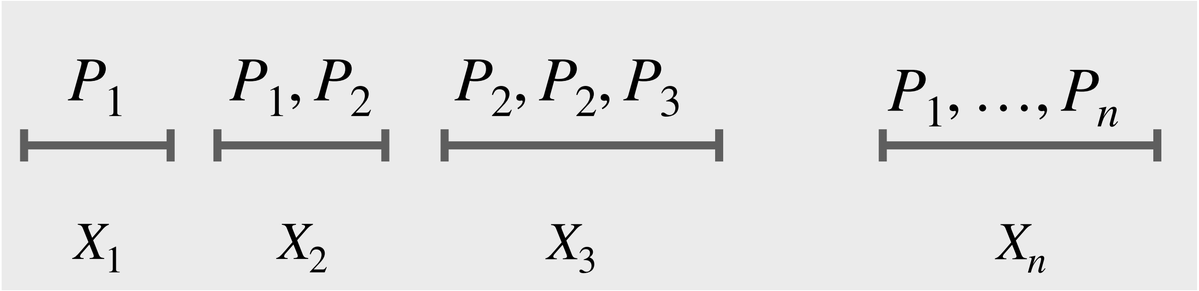


- Each phase can be though of as flipping a biased coin until we see a head, where seeing a head = getting a new card



Pokemon Collector Problem

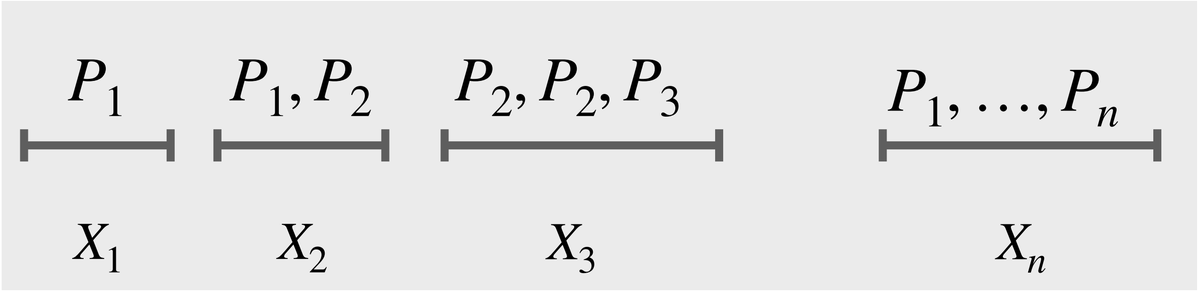
- $E[X_i]$ is the expected number of coin flips until success (expectation of a geometric r.v.)



- We know, $E[X_i] = 1/p_i$ where p_i is the probability of success/ probability of seeing a heads during a coin flip in the ith phase
- Before the ith phase starts, we don't have $n - i + 1$ Pokemon
- Each of the n Pokemon are equally likely to be in a pack
- $p_i = \dfrac{n - i + 1}{n}$

Pokemon Collector Problem

- We know, $E[X_i] = 1/p_i$ where p_i is the probability of success/ probability of seeing a heads during a coin flip in the ith phase

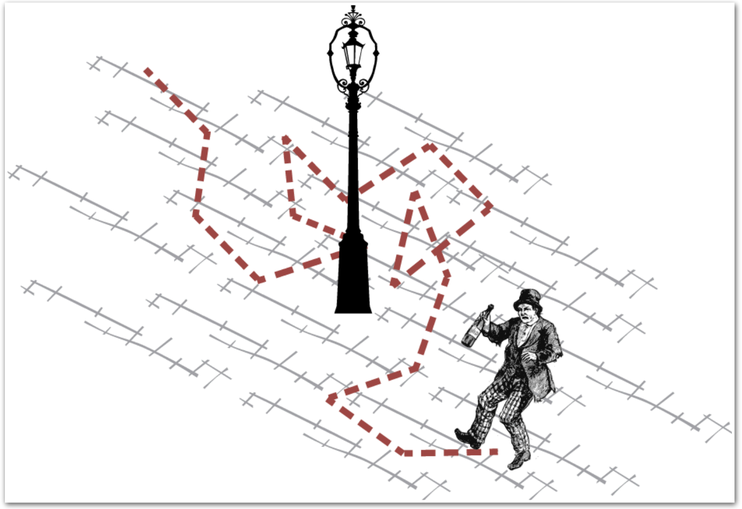


- $p_i = \dfrac{n - i + 1}{n}$
- $E[X_i]$ = Expected[number of flips until first heads] $= 1/p_i$
- $E[X] = E[\sum_{i=1}^{n} X_i] = \sum_{i=1}^{n} E[X_i] = \sum_{i=1}^{n} \dfrac{n}{n - i + 1} = \sum_{i=1}^{n} \dfrac{n}{i} = nH_n = \Theta(n \log n)$

Random Walks and Recurrences

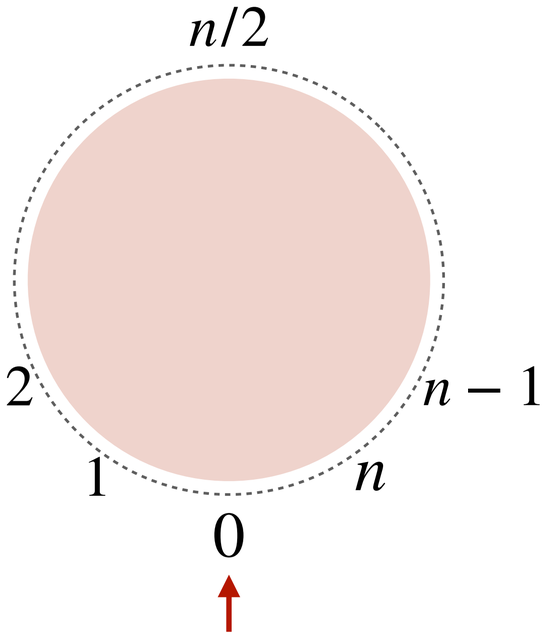
Random Walks

- A drunkard stumbles out of a bar. Each second, he either staggers 1 step to the left or staggers 1 step to the right, with equal probability. His home lies x steps to his left, and a canal lies y steps to his right.
- **Questions.** What is the probability that the drunkard arrives safely at home instead of falling into the canal? What is the expected duration of his journey, however it ends?
- The drunkard's meandering path is called a **random walk**
- Random walks are important as they model various phenomenon:
    - In Physics, random walks model gas diffusion
    - Google search engine uses random walks through the graph of web links to determine the relative importance of website
    - In finance theory, random walks can serve as a model for the fluctuation of market prices.



Pass the Candy

- We have n students labelled $1,\ldots,n$ and a professor labelled 0 sitting around in a circle
- Initially the professor has a candy bowl
- He withdraws a piece of candy and then passes the bowl either to the left or right, with equal probability
- Each person who receives the bowl takes a piece of candy if they do not already have one; then passes it on randomly
- The ***last*** person to receive the candy wins the game
- Which player is most likely to win?
- Guess? Seems like 1 and n are almost always going to be eliminated right away. Seems like $n/2$ is most likely to win—but by how much?



Simpler Problem

- Suppose the players $A, S_1, \ldots, S_k, B$ are arranged in a line instead and S_1 initially has a the candy
- As before, whenever a player gets the bowl they take a candy and pass it left or right with equal probability
- What is the probability that A gets the candy before B?
- Let P_k be the probability that A gets the candy before B.
- Base case. Suppose $k = 1$, then $P_1 = 1/2$
- Suppose $k > 1$. In the first step there are two possibilities: the bowl either moves left to A or right to S_2

$$
\begin{aligned}
P_k &= \Pr(\text{first step is left}) \\
&\qquad \cdot \Pr(A \text{ gets candy before } B \mid \text{first step is left}) \\
&\quad + \Pr(\text{first step is right}) \\
&\qquad \cdot \Pr(A \text{ gets candy before } B \mid \text{first step is right}) \\
&= \frac{1}{2} \cdot 1 + \frac{1}{2} \cdot \Pr(A \text{ gets candy before } B \mid \text{first step is right})
\end{aligned}
$$

Simpler Problem

- $$\begin{aligned} P_k &= \Pr(\text{first step is left}) \\ &\quad \cdot \Pr(A \text{ gets candy before } B \mid \text{first step is left}) \\ &\quad + \Pr(\text{first step is right}) \\ &\quad \cdot \Pr(A \text{ gets candy before } B \mid \text{first step is right}) \\ &= \frac{1}{2} \cdot 1 + \frac{1}{2} \cdot \Pr(A \text{ gets candy before } B \mid \text{first step is right}) \end{aligned}$$

- Recurrence. $P_k = \frac{1}{2} \cdot 1 + \frac{1}{2} \cdot P_{k-1} \cdot P_k$ and $P_1 = \frac{1}{2}$ (Base case)
- Solve it using guess and check, and prove by induction
- $P_2 = \frac{1}{2 - P_1} = \frac{2}{3}$, $P_3 = \frac{1}{1 - P_2} = \frac{3}{4}$
- $P_k = \frac{k}{k+1}$ (Verify this is correct by induction)

New starting configuration

$\Pr(S_1 \text{ gets candy before } B) = P_{k-1}$

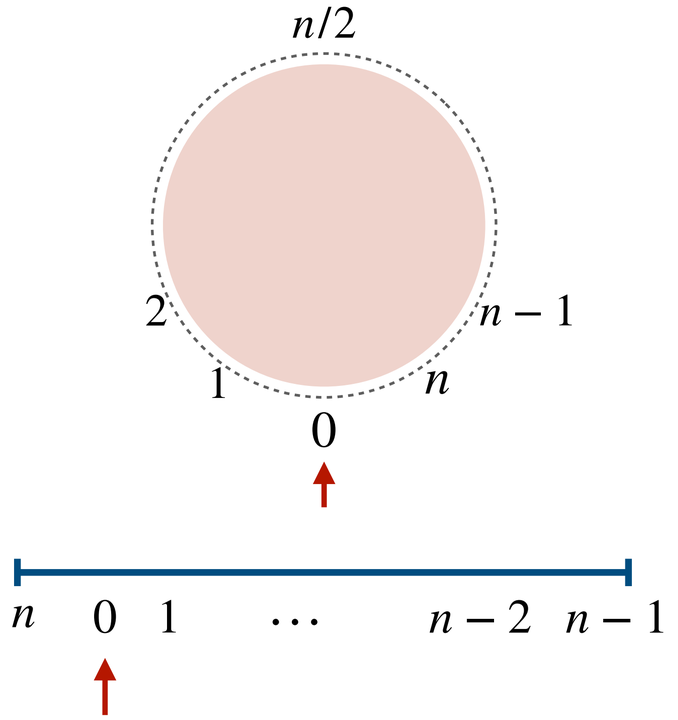
If S_1 gets candy, we are back in the initial configuration and A gets the candy before B with probability P_k.

Back to the Candy Game

- Consider player n on the right side of the professor. Only way n can win is the candy travels clockwise all the way around to player $n-1$ before n ever touches it
- Thus, if we cut the circle and arrange on a line as shown, n wins if and only if $n-1$ gets the bowl before n
- This fits our previous simpler problem model where we want to know the probability that B gets candy before A and $k = n-2$
- Pr($n-1$ gets candy before n) $= 1 -$ Pr(n gets candy before $n-1$)

$$= 1 - \frac{n-2}{(n-2)+1} = \frac{1}{n}$$

- Student n wins with probability $1/n$!
- We can extend this argument to show each student wins with probability $1/n$ --- we would never have guessed this!



n wins only if $n-1$ gets candy before n

Randomized Algorithm I:
Karger's Min-Cut

Randomized Min Cut

- **Global min-cut problem.**
  Given an undirected, unweighted graph $G = (V, E)$, find a cut (A, B) of minimum cardinality (that is, min # of edges crossing it).
- **Applications.** Network reliability, network design, circuit design, etc.
- **Poly-time network-flow solution** (by reduction to min s-t cut).
  - Replace every undirected edge (u, v) with $u \rightarrow v$ and $v \rightarrow u$, each of capacity 1
  - Fix any $s \in V$ and compute min s-t cut for every other node $t \in V - \{s\}$
  - $(n - 1)$ executions of min s-t cut
- Gives impression that finding global min cut is harder than finding a min s-t cut, which is not true
- Deceptively simple and efficient randomized algorithm [Karger 1992]

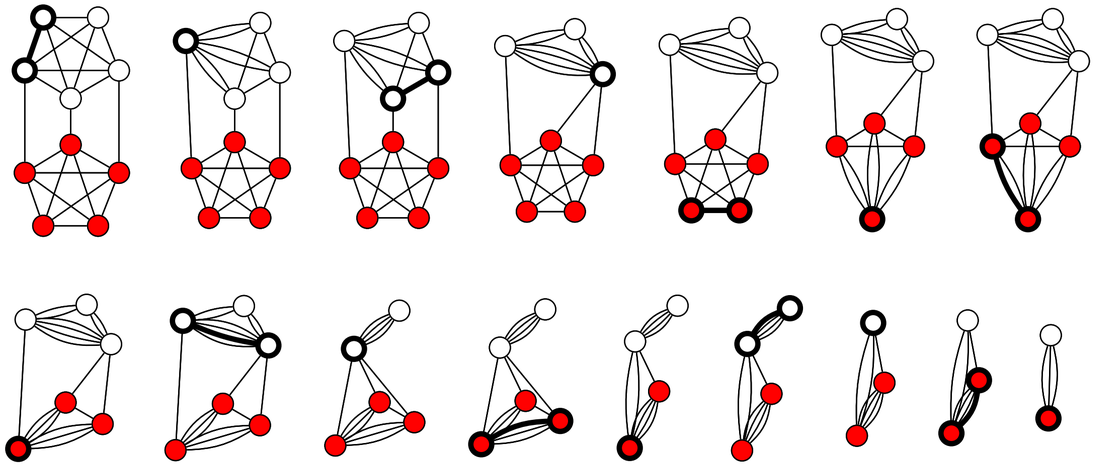
Karger's Min Cut

- Uses a primitive called *edge contraction*
- Contract edge e in G, denoted $G \leftarrow G/e$
    - Replace u and v by single new super-node w
    - Preserve edges, updating endpoints of u and v to w
    - Keep parallel edges, but delete self-loops
- An edge can be contracted in $O(n)$ time, assuming the graph is represented as an adjacency list

Karger's Min Cut

- Algorithm tries to *guess* the min cut by randomly contracting edges
- Running time $O(n^2)$ (why?)
- Correctness:
  How often, if ever, does it return the min cut?

```
GuessMinCut(G):
    for i ← n downto 2
        pick a random edge e in G
        G ← G/e
    return the only cut in G
```

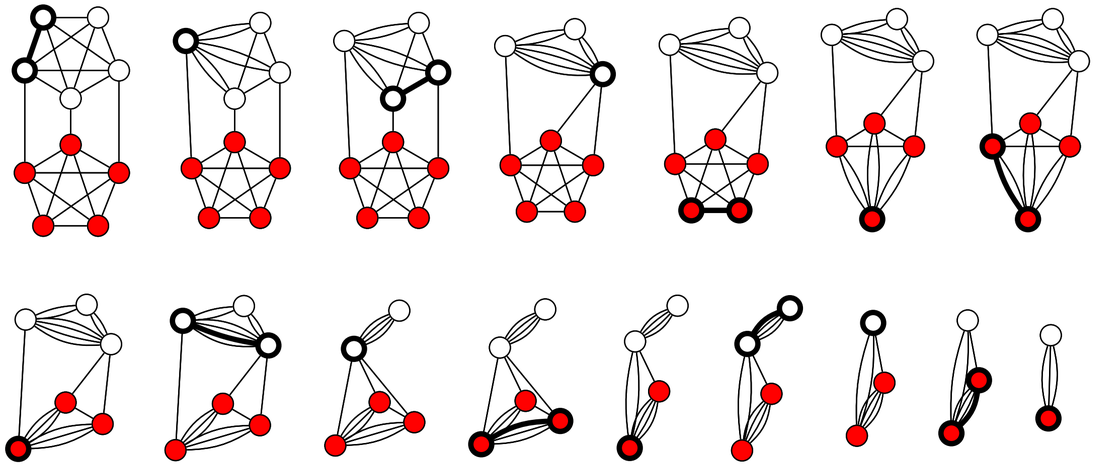


Reference: Thore Husfeldt

Observations:

If the minimum cut has size/cardinality k

- Each vertex must have degree at least k, and thus the graph must have at least $nk/2$ edges
- Any cut in the contracted graph is a cut in the original graph
- Let $C = (S, V - S)$ be any cut, if algorithm never contracts an edge crossing this cut, then it will produce the cut C



Reference: Thore Husfeldt

Karger's Analysis

- Let C be any arbitrary min cut of cardinality k
- If we pick an edge in G uniformly at random, what is the probability of picking an edge in C
    - $m \geq nk/2$
    - Pr(picking an edge in C) = $\frac{k}{m} \leq \frac{k}{nk/2} = \frac{2}{n}$
- The probability we don't screw up in the 1st step $\geq 1 - \frac{2}{n}$
- After the first edge is contracted, the algorithm proceeds recursively (with independent random choices) on the $(n-1)$-vertex graph

Karger's Analysis

- Let $P(n)$ denote the probability that the algorithm returns the correct min cut on an n-vertex graph, then
- $P(n) \geq \left(1 - \frac{2}{n}\right) \cdot P(n-1)$, with base case $P(2) = 1$
- Expanding the recurrence:
  - $P(n) \geq \frac{n-2}{n} \cdot \frac{n-3}{n-1} \cdot \frac{n-4}{n-2} \cdot \cdots \cdot \frac{2}{4} \cdot \frac{1}{3}$
- Terms cancel out to get: $P(n) \geq \frac{2}{n(n-1)} = \binom{n}{2}^{-1}$

Amplifying Success Probability

- Thus, a single execution of Karger's min cut algorithm finds the min cut with probability at least $1/\binom{n}{2}$, which is low
    - But, we can amplify our success probability!
- Run the algorithm R times (using independent random choices) and pick the best min-cut among them
- What is probability we don't find the min cut after R repetitions?
    - $\left(1 - 1/\binom{n}{2}\right)^{R}$

Amplifying Success Probability

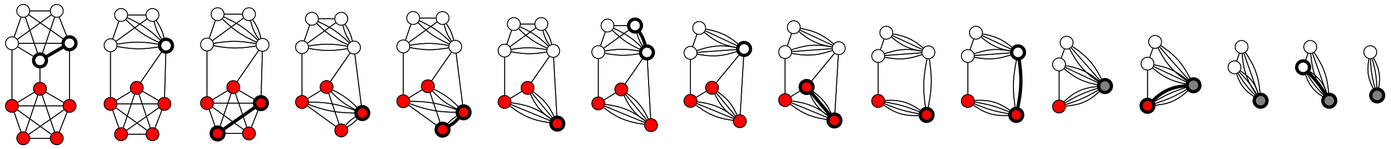
- If we execute $R = \binom{n}{2}$ times, the probability of failure is
  - $\left(1 - 1/\binom{n}{2}\right)^{\binom{n}{2}} \leq \frac{1}{e}$
- If we run the algorithm $R = \binom{n}{2} c \ln n$ times, we can make the failure probability polynomially small: $\left(\frac{1}{e}\right)^{c \ln n} = \frac{1}{n^c}$
- Karger's algorithm finds the min-cut with high probability (w.h.p.)

Useful Inequality: $\left(1 - \frac{1}{x}\right)^x \leq \frac{1}{e}$ for $x \geq 1$

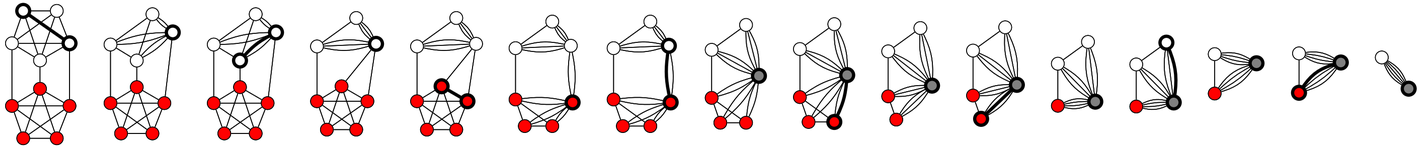
An algorithm is correct **with high probability (w.h.p.)** with respect to input size n if it fails with probability at most $\frac{1}{n^c}$ for any constant $c > 1$.

Example Execution

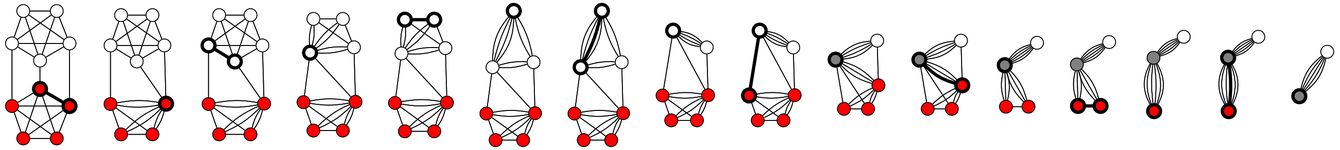
trial 1



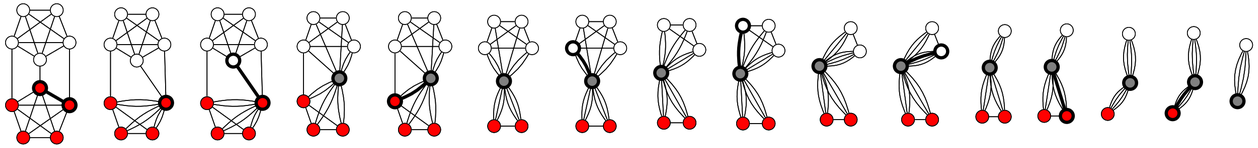
trial 2



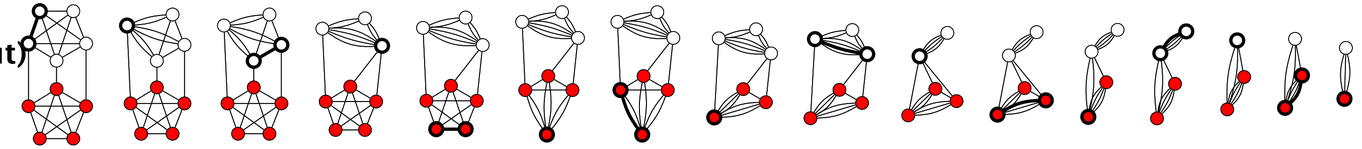
trial 3



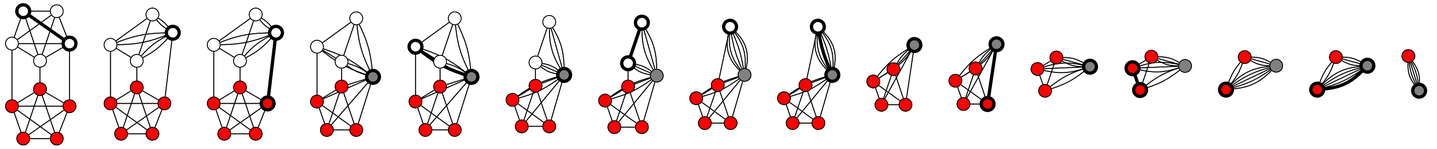
trial 4



trial 5
(finds min cut)



trial 6



...

Reference: Thore Husfeldt

Karger's Running Time

- Thus, Karger's algorithm finds the min-cut with high probability (w.h.p.)
- Running time: we perform $\Theta(n^2 \log n)$ iterations, each $O(n^2)$ time
  - $O(n^4 \log n)$ time
  - Faster than naive-flow-techniques, nothing to get excited about
- Improves to $O(n^2 \log^3 n)$ by guessing cleverly! [Karger-Stein 1996]
- **Idea**: Improve the guessing algorithm using the observation:
  - As the graph shrinks, the probability of contracting an edge in the minimum cut increases
  - At first the probability is very small: $2/n$ but by the time there are three nodes, we have a $2/3$ chance of screwing up!

Takeaways

- Notice: Karger's algorithm had *one-sided error*:
  - Might produce a cut that is not min cut
- You can increase the success rate of a Monte Carlo algorithm with one-sided errors by iterating it multiple times and taking the best solution
  - If the probability of success is $1/f(n)$, then running it $O(f(n)\log n)$ times gives a high probability of success
- If you're more intelligent about how you iterate the algorithm, you can often do much better than this
- Next, we'll see an example of a Las Vegas algorithm
  - Randomized selection and quick sort

Randomized Algorithm II

Randomized Selection

Randomized Selection

- **Problem.** Find the kth smallest/largest element in an unsorted array
- Recall our selection algorithm

Select (A, k):

If $|A| = 1$: return $A[1]$

Else:

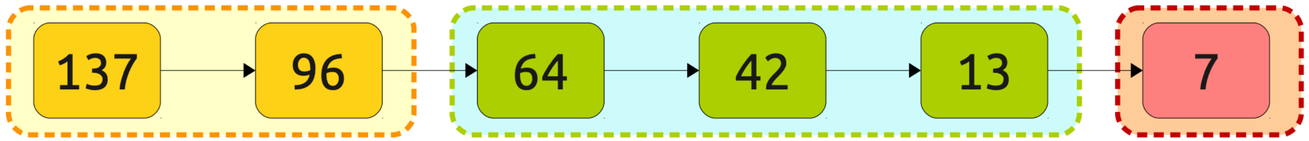
- Choose a pivot $p \leftarrow A[1, \ldots, n]$; let r be the rank of p
- $r, A_{<p}, A_{>p} \leftarrow$ Partition$((A, p)$
- If $k == r$, return p
- Else if $k < r$: Select $(A_{<p}, k)$
- Else: Select $(A_{>p}, k - r)$

Selection with a Good Pivot

- Recall: we called the pivot “good” if it reduced the array size by at least a constant
  - Which would give a recurrence $T(n) \leq T(\alpha n) + O(n)$ for some constant $\alpha < 1$
  - Expands to a decreasing geometric series $T(n) = O(n)$
- In the deterministic algorithm, how did we find a good pivot?
  - Split array into groups of 5
  - And computed the median of group medians
  - The pivot guaranteed that $n \to 7n/10$
- **Here is a silly idea:** What if we pick the pivot uniformly at random?

Analyzing Rand. Selection

- Normally, we'd write a recurrence relation for a recursive function
- But the array size in later recursive call depends on the random choice of pivots in earlier calls
- We use a different accounting trick for running time
- Randomized selection makes at most one recursive call each time:
  - Group multiple recursive call in "phases"
  - Sum of work done by all calls is equal to the sum of the work done in all the phases



Analyzing in Phases

- Idea: let a “phase” of the algorithm be the time it takes for the array size to drop by a constant factor (say $n \to (3/4) \cdot n$)
- If array shrinks by a constant factor in each phase and linear work done in each phase, what would be the running time?
- $T(n) = c(n + 3n/4 + (3/4)^2 n + \ldots + 1) = O(n)$
- If we want a $1/4$th, $3/4$th split, what range should our pivot be in?
  - Middle half of the array (if n size array, then pivot in $[n/4, 3n/4]$)
  - What is the probability of picking such a pivot?
    - $1/2$
  - Phase ends as soon as we pick a pivot in the middle half
    - Expected # of recursive calls until phase ends? 2

Expected Running Time

- Let the algorithm be in phase j when the size of the array is
  - At least $n\left(\frac{3}{4}\right)^{j}$ but not greater that $n\left(\frac{3}{4}\right)^{j+1}$
- Expected number of iterations within a phase: 2
- Let X_j be the expected number of steps spent in phase j
- $X = X_0 + X_1 + X_2 \ldots$ be the total number of steps taken by the algorithm
- Within a phase, the algorithm does work linear in the size of the array in one iterations and thus, $E[X_j] \leq 2cn\left(\frac{3}{4}\right)^{j}$
- Expected running time:

$$E[X] = \sum_j E[X_j] \leq 2cn \sum_j \left(\frac{3}{4}\right)^{j} \leq 8cn = O(n)$$

Randomized Algorithm III

Randomized QuickSort

Randomized Quicksort

- Recall deterministic Quicksort
- Depending on the choice pivot, could be $O(n^2)$
- What if we pick the pivot uniformly at random?
    - Can get expected running time as $O(n \log n)$

Quicksort(A)**:**

If $|A| < 3$: Sort(A) directly

Else: choose a pivot element $p \leftarrow A$

$A_{<p}, A_{>p} \leftarrow$ Partition around p

Quicksort$(A_{<p})$

Quicksort$(A_{>p})$

Modified Rand. Quicksort

- Before we analyze quick sort with uniform random pivot
- Consider the following modification
  - Pick pivot p randomly
  - Partition array around p
  - If p is a bad pivot (say, $\max\{|A_{<p}|, |A_{>p}|\} > (3/4)|A|$), we throw it our and pick another pivot
  - Else, we recursively call Quicksort on the partitions
- We know that expected number of trials before we get a good pivot is 2 and a good pivot gives a $1/4,3/4$ split
- This immediately gives us expected running time as $O(n \log n)$

Randomized Quicksort

- Suppose we don't throw out bad pivots (its wasteful anyway)
- Can we still show the expected running time is the same
    - Intuitively bad pivots don't hurt asymptotically, because they only occur $1/2$ the time
- We analyze quicksort using another accounting trick
    - Only two types of work:
        - Work making recursive calls (lower order term, turns out)
        - Work partitioning the elements
- How many recursive calls in the worst case?
    - $O(n)$

Randomized Quicksort

- We thus need to bound the work partitioning elements
- Partitioning an array of size n around a pivot element p takes exactly $n - 1$ comparisons
- We won't look at partitions made in each recursive calls, which depend on the choice of random pivot
- **Idea:** Account for the total work done by the partition step by summing up the total number of comparisons made
- Two ways to count total comparisons:
    - Look at the size of arrays across recursive calls and sum
    - Look at all pairs of elements and count total # of times they are compared (easier to do in this case)

Counting Total Comparisons

- Just for analysis, let B denote the sorted version of input array A, that is, $B[i]$ is the ith smallest element in A
- Define random variable X_{ij} as the number of times Quicksort compares $B[i]$ and $B[j]$
- Observation: $X_{ij} = 0$ or $X_{ij} = 1$, why?
    - $B[i]$, $B[j]$ only compared when one of them is the current pivot; pivots are excluded from future recursive calls
- Let $T = \sum_{i=1}^{n} \sum_{j=i+1}^{n} X_{ij}$ be the total number of comparisons made by randomized Quicksort

Expected Running Time

- Goal: $E[T] = E\left[\sum_{i=1}^{n}\sum_{j=i+1}^{n} X_{ij}\right] = \sum_{i=1}^{n}\sum_{j=i+1}^{n} E[X_{ij}]$
- $E[X_{ij}] = \Pr[X_{ij} = 1]$
- When is $X_{ij} = 1$? That is, when are $B[i]$ and $B[j]$ compared?
- Consider a particular recursive call. Let rank of pivot p be r.
  - Case 1. One of them is the pivot: $r = i$ or $r = j$
  - Case 2. Pivot is between them: $r > i$ and $r < j$
  - Case 3. Both less than the pivot: $r > i, j$
  - Case 4. Both greater than the pivot: $r < i, j$

Comparisons for Each Case

- Case 1. $r = i$ or $r = j$
  - $B[i]$ and $B[j]$ are compared once and one of them is excluded from all future calls
- Case 2. $r > i$ and $r < j$
  - $B[i]$ and $B[j]$ are both compared to the pivot but not to each other, after which they are in different recursive calls: will never be compared again
- Case 3. $r > i, j$ and Case 4. $r < i, j$
  - $B[i]$ and $B[j]$ are not compared to each other, they are both in the same subarray and may be compared in the future
- **Takeaway:** $B[i]$, $B[j]$ are compared for the 1st time when one of them is chosen as pivot from $B[i], B[i+1], \ldots, B[j]$ & never again

Expected Running Time

- $\Pr[X_{ij} = 1] = \Pr$(one of them is picked as pivot from $B[i], B[i+1], \ldots, B[j]$
- $\Pr[X_{ij} = 1] = \dfrac{2}{j - i + 1}$
- $E[T] = \sum_{i=1}^{n} \sum_{j=i+1}^{n} E[X_{ij}] = 2 \sum_{i=1}^{n} \sum_{j=i+1}^{n} \dfrac{1}{j - i + 1}$

Expected Running Time

- $B[i]$ and $B[j]$ are compared iff one of them is the first pivot chosen from the range $B[i], B[i+1], \ldots, B[j]$
- $\Pr[X_{ij} = 1] = \dfrac{2}{j - i + 1}$
- $E[T] = \displaystyle\sum_{i=1}^{n} \sum_{j=i+1}^{n} E[X_{ij}] = 2 \sum_{i=1}^{n} \sum_{j=i+1}^{n} \frac{1}{j - i + 1}$
- For fixed i, inner sum is
  $\dfrac{1}{2} + \dfrac{1}{3} + \dfrac{1}{4} + \ldots \dfrac{1}{n - i + 1} \leq \displaystyle\sum_{\ell=2}^{n} \frac{1}{\ell} = O(\log n)$
- Thus, expected running time
  $E[T] = O(n \log n + n) = O(n \log n)$

From # of recursive calls

Quick Sort Summary

- Las Vegas algorithms like Quicksort and Selection are always correct but their running time guarantees hold in expectation
- We can actually prove that the number of comparisons made by Quicksort is $O(n \log n)$ **with high probability**
    - This means the the probability that the running time of quicksort is more than a constant factor away from its expectation is very small (polynomially small in n)
    - Called **concentration bounds**
- Can prove by yet another accounting trick:
    - Counting how many times a pivot and non-pivot elements are compared during the execution

Acknowledgments

- Some of the material in these slides are taken from
  - Kleinberg Tardos Slides by Kevin Wayne (https://www.cs.princeton.edu/~wayne/kleinberg-tardos/pdf/04GreedyAlgorithmsI.pdf)
  - Jeff Erickson's Algorithms Book (http://jeffe.cs.illinois.edu/teaching/algorithms/book/Algorithms-JeffE.pdf)
  - Hamiltonian cycle reduction images from Michael Sipser's Theory of Computation Book