Randomized Algorithms (Ch13)

- **Randomization**
- **Contention Resolution**
- **Global Min-cut**
- **Linearity of Expectation**
- **Maximum 3-Satisfiability**
- **Derandomization**
- **Monte Carlo and Las Vegas Algorithms**
- **Universal Hashing**
- **Closest Pair Problem**
- **Cache Maintenance Problem**
- **Chernoff Bounds**
- **Load Balancing**

Randomization

- **Algorithmic design approaches**

  Greedy

  Divide-and-conquer

  Dynamic programming

  Network flow

  Randomization

- **Randomization: Allow fair coin flip (access to a pseudo-random number generator in practice) in unit time to decide the next step.**

- **Randomization can lead to simplest, fastest, or only known algorithm for a particular problem. Examples: Symmetry-breaking protocols, graph algorithms, quicksort, hashing, load balancing,...**

Preliminaries for randomization

Experiments and outcomes

Experiment	Outcome
Toss a coin	$\{$**heads,tails**$\} = \{$**H,T**$\}$
Roll a dice	$\{1, 2, 3, 4, 5, 6\}$
Roll two dices	$\{1, .., 6\} \times \{1, .., 6\}$ **or** $A \subseteq \{1, .., 6\}$**,** $\lvert A\rvert \leq 2$
Buy 3 lottery tickets (out of 100,000)	**3-element subsets of** $\{1, .., 100, 000\}$



Sample space and events

- **Set S of all outcomes of an experiment is called the sample space.**
- **Any subset A of S is called an event.**
- **Examples**

Experiment	Sample space	event
Toss 2 coins	$S = \{\text{H,T}\} \times \{\text{H,T}\}$	**get exactly 1 H** $A = \{\text{(H,T),(T,H)}\}$
Roll 2 dice	$S = \{1, .., 6\} \times \{1, .., 6\}$	**sum of dice is 6** $A = \{(1,5), (2,4), (3,3), (4,2), (5,1)\}$

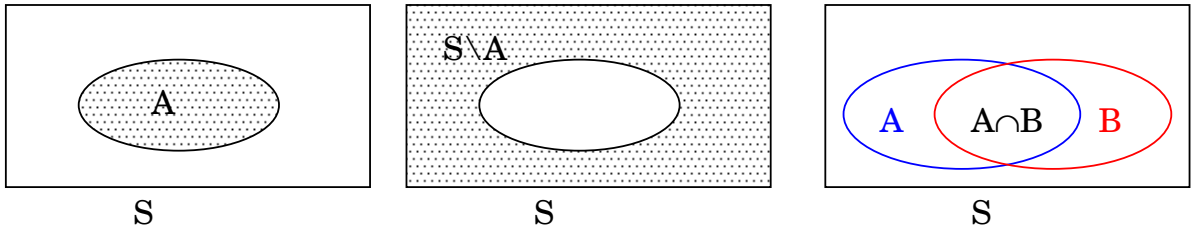
Probability

- **Equal likehood assumption: In an experiment, each of all possible outcomes has the same likehood of occurrence, or the same probability of occurrence.**
- **Under the equal likehood assumption, let S be the sample space for an experiment. For $a \in S$ and $A \subseteq S$,**

$$\Pr[\{a\}] = \Pr[a] = \frac{1}{|S|} \quad \text{probability that } a \text{ occurs}$$
$$\Pr[A] = \frac{|A|}{|S|} \quad \text{probability } A \text{ occurs}$$

- **Examples**
  - **Probability of getting H (heads) in tossing a coin:**
    Sample space $S = \{$H,T$\}$, event $A = \{$H$\}$, $\Pr[A] = \frac{|A|}{|S|} = \frac{1}{2}$.
  - **Probability of getting even number in rolling a dice:**
    $S = \{1, 2, 3, 4, 5, 6\}$**, event** $A = \{2, 4, 6\}$**,** $\Pr[A] = \frac{|A|}{|S|} = \frac{3}{6} = \frac{1}{2}$**.**

- **Probability is a measure quantifying the likehood that an event A in a sample space S will occur, a function $\Pr : S \to [0,1]$ s.t. for any $A \subseteq S$, $\Pr[A] = \sum_{a \in A} \Pr[a]$ satisfying:**
  $\Pr[\emptyset] = 0$, $\Pr[S] = 1$, $0 \leq \Pr[A] \leq 1$ **for all** $A \subseteq S$ **and**
  $\Pr[A \cup B] = \Pr[A] + \Pr[B]$ **for any disjoint** $A, B \subseteq S$.
- **Example:** $S = \{a_1, a_2, .., a_n\}$, $\Pr[a_i] = p_i \geq 0$, $\sum_{i=1}^{n} p_i = 1$.
  For $A \subseteq S$, $\Pr[A] = \sum_{a_i \in A} p_i$.
  Equal likehood assumption: $\Pr[a_i] = 1/n$ **for every** $1 \leq i \leq n$.
- **For a sample space S and events $A, B \subseteq S$,**
  $\Pr[\overline{A}] = 1 - \Pr[A]$, **where** $\overline{A} = S \setminus A$.
  Union Bound
  $\Pr[A \cup B] = \Pr[A] + \Pr[B] - \Pr[A \cap B] \leq \Pr[A] + \Pr[B]$.



Discrete random variable

- **A variable that takes discrete values with certain probability, (e.g.,** $X = 0, 1, .., n$ **with** $\Pr[X = i] = p_i$**).**
- **Example,** X **=amount of money one wins by buying 1 lottery ticket:**

  There are 1000 tickets, 1 wins $\$10,000$**, 10 wins** $\$100$**, the rest wins nothing;**

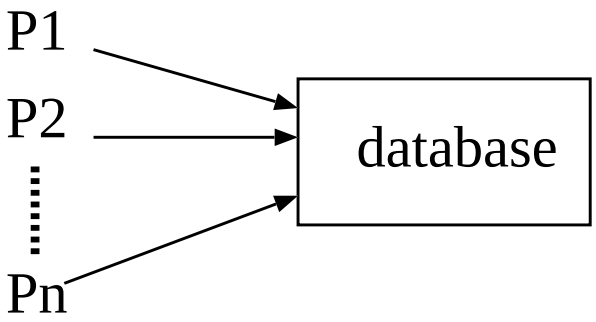
  $\Pr[X = 10,000] = \frac{1}{1000}$**,** $\Pr[X = 100] = \frac{1}{100}$**,** $\Pr[X = 0] = \frac{989}{1000}$**.**
- **Let** X **be a discrete random variable taking values** $v_1, .., v_k$**. The expectation of** X **is** $E[X] = v_1 \Pr[X = v_1] + .. + v_k \Pr[X = v_k]$**.**

  Example:

$$\begin{aligned} E[X] &= E[\text{money one wins}] \\ &= 10000 \Pr[X = 10000] + 100 \Pr[X = 100] + 0 \Pr[X = 0] \\ &= 10000 \frac{1}{1000} + 100 \frac{10}{1000} = 11. \end{aligned}$$

Contention Resolution

- **Contention resolution in a distributed system.**
  - **Instance: Given n processes $P_1, .., P_n$, each competing for access to a shared database. If two or more processes access the database at the same time, all processes are locked out.**
  - **Object: A protocol to ensure all processes get through on a regular basis.**
  - **Restriction: Processes can not communicate.**
- **Challenge: Need a symmetry-breaking paradigm.**



- **A randomized protocol for contention resolution**

  Each process requests access to the database at time t with probability $p = 1/n$.

- **Claim 1: Let $S(i,t)$ be the event that process i succeeds in accessing the database at time t. Then $\frac{1}{en} \leq \Pr[S(i,t)] \leq \frac{1}{2n}$.**

  Proof. By independence, $\Pr[S(i,t)] = p(1-p)^{n-1} = \frac{1}{n}(1-\frac{1}{n})^{n-1}$. Since $\frac{1}{e} \leq (1-\frac{1}{n})^{n-1} \leq \frac{1}{2}$, the claim holds. □

- **Useful facts from calculus. As n increases from 2,**

  $\lim_{n\to\infty}(1+\frac{1}{n})^n = e$**,** $e = 2.718....$

  $(1-\frac{1}{n})^n$ **converges monotonically from 1/4 up to** $1/e$**.**

  $(1-\frac{1}{n})^{n-1}$ **converges monotonically from 1/2 downto** $1/e$**.**

- **Claim 2: The probability that process i fails to access the database in en rounds is at most $1/e$; after $en(c \ln n)$ rounds, the probability is at most n^{-c}.**

  Proof. Let $F(i,t)$ be the event that process i fails to access the database in rounds 1 through t. By independence and Claim 1, $\Pr[F(i,t)] \leq (1 - \frac{1}{en})^t$.

  Choose $t = \lceil en \rceil$,

  $$\Pr[F(i,t)] \leq (1 - \frac{1}{en})^{\lceil en \rceil} \leq (1 - \frac{1}{en})^{en} \leq \frac{1}{e}.$$

  Choose $t = \lceil en \rceil \lceil c \ln n \rceil$,

  $$\Pr[F(i,t)] \leq (\frac{1}{e})^{c \ln n} = n^{-c}.$$

  □

- **Claim 3: The probability that all processes succeed within $2en \ln n$ rounds is at least $1 - 1/n$.**

  Proof. Let $F(t)$ be the event that at least one of the n processes fails to access the database in any of the rounds 1 through t. By Union Bound (Given events $E_1, .., E_n$, $\Pr[\cup_{i=1}^{n} E_i] \leq \sum_{i=1}^{n} \Pr[E_i]$; taking $E_i = F(i,t)$) and Claim 2,

$$\Pr[F(t)] = Pr[\cup_{i=1}^{n} F(i,t)] \leq \sum_{i=1}^{n} \Pr[F(i,t)] \leq n(1 - \frac{1}{en})^t.$$

  Choose $t = 2\lceil en \rceil \lceil \ln n \rceil$ gives $\Pr[F(t)] \leq nn^{-2} = 1/n$. □

Global Min-Cut

- **Global min-cut problem:**

  Instance, a connected graph $G(V, E)$.

  Goal, find a cut (A, B) (a set of edges whose removal partitions G into two separated components A and B) of minimum cardinality.

- **Applications: Partition items in a database, identify clusters of related documents, network reliability, network design, circuit design, TSP solvers, etc.**

- **Network flow based solution: Replace every edge $\{u, v\}$ with two arcs $(u, v), (v, u)$; fixed a node s and computes min $s - t$ cut for every $t \in V(G) \setminus \{s\}$.**

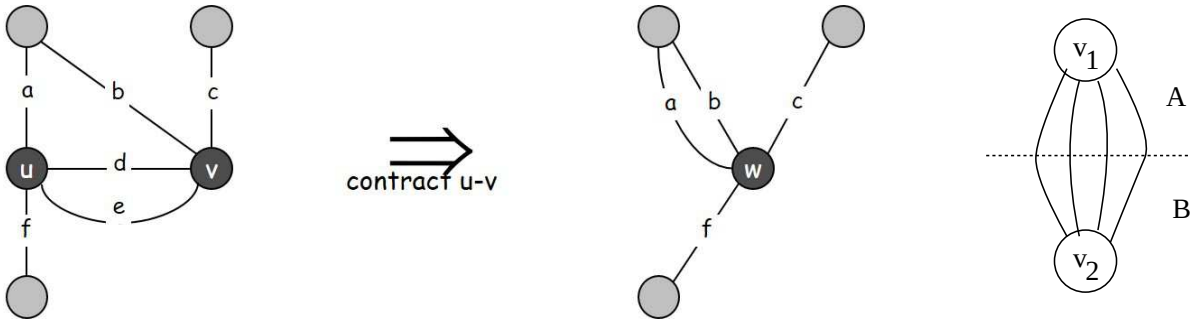
- **Flow based solution uses $O(n)$ calls to a min $s - t$ cut solver.**

  An early intuition is that global min-cut is harder than min $s - t$ cut.

  This intuition is not true due to a randomized algorithm [Karger 1995].

Contraction algorithm [Karger 1995]

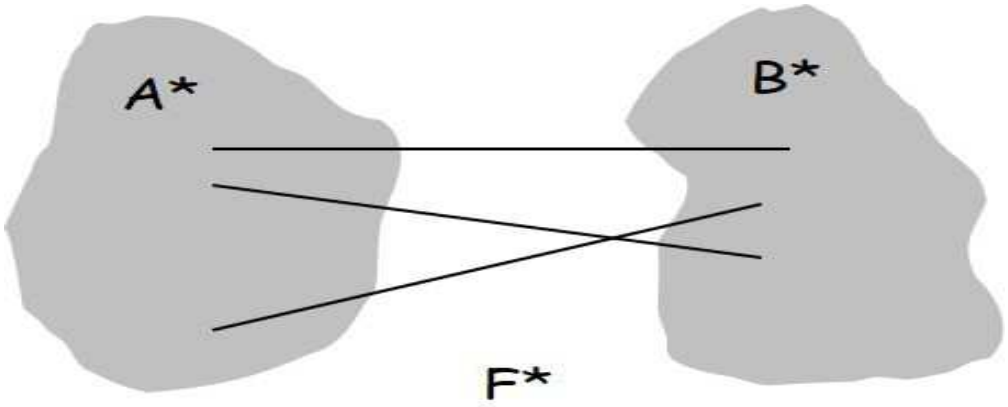
1. **Select an edge $e = \{u, v\}$ uniformly at random.**
2. **Contract edge e**
   - **replace u and v by a new node w;**
   - **for each edge $\{x, y\}$ with $y = u$ or $y = v$, replace y with w;**
   - **keep parallel edges and delete self-loops.**
3. **Repeat until graph has two nodes v_1 and v_2.**
4. **Return the cut (the set of nodes contracted to form v_1).**

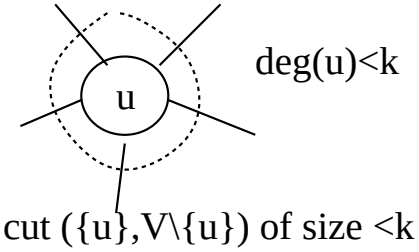


Claim: Given a graph G of n nodes, the contraction algorithm returns a min-cut with probability at least $\frac{2}{n^2}$.

Proof. Consider a global min-cut (A^*, B^*) of G. Let F^* be edge-cut set that gives (A^*, B^*) and $|F^*| = k$.

In the 1st step, an edge in F^* is contracted with probability $k/|E|$. For each node $u \in V$, $\deg(u) \geq k$, as otherwise (A^*, B^*) is not a min-cut, implying $|E| \geq (\frac{1}{2})kn$. Thus, an edge in F^* is contracted in the 1st step with probability at most $\frac{k}{|E|} = \frac{k}{(1/2)kn} = 2/n$.





Let E_j be the event that an edge in F^* is not contracted in iteration j.

$$\begin{aligned}
& \Pr[E_1 \cap E_2 \cap .. \cap E_{n-2}] \\
= \quad & \Pr[E_1] \times \Pr[E_2|E_1] \times .. \times \Pr[E_{n-2}|E_1 \cap .. \cap E_{n-3}] \\
\geq \quad & (1-\frac{2}{n})(1-\frac{2}{n-1})(1-\frac{2}{n-2})(1-\frac{2}{n-3})\cdots(1-\frac{2}{4})(1-\frac{2}{3}) \\
= \quad & (\frac{n-2}{n})(\frac{n-3}{n-1})(\frac{n-4}{n-2})(\frac{n-5}{n-3})\cdots(\frac{2}{4})(\frac{1}{3}) \\
= \quad & \frac{2}{n(n-1)} \geq \frac{2}{n^2}.
\end{aligned}$$

□

- **Amplification: To amplify the probability of finding a min-cut, run the contraction algorithm many times.**
- **Claim: If the contraction algorithm is repeated $n^2 \ln n$ times with independent random edge selections, the probability of failing to find the global min-cut is at most $1/n^2$.**

  Proof. By independence, the failure probability is at most

$$(1 - \frac{2}{n^2})^{n^2 \ln n} = ((1 - \frac{2}{n^2})^{n^2/2})^{2 \ln n} \leq (e^{-1})^{2 \ln n} = \frac{1}{n^2}.$$

□

- **Running time: one execution of contraction algorithm takes $O(m)$ time, $O(n^2 \ln n)$ runs of the algorithm takes $O(mn^2 \log n)$ time.**
- **Improve running time to $O(n^2 \log n)$ [Karger-Stein 1996]:**
  - **Early runs of the algorithm are less risky than later runs, the probability of contracting an edge in min-cut hits 0.5 when $n/(\sqrt{2})$ nodes remain.**
  - **Run contraction algorithm until $n/(\sqrt{2})$ nodes remain to get graph H.**
  - **Run contraction algorithm twice on H and return the best of two cuts.**
- **Best known running time: $O(m \log^3 n)$ [Karger 2000]; faster than the best known max-flow algorithm or deterministic global min-cut algorithm.**

Linearity of Expectation

- **The expectation of a discrete random variable X taking values $v_0, v_1, ..$ is defined as:**

$$E[X] = \sum_{j=0}^{\infty} v_j \Pr[X = v_j].$$

- **Example, waiting for a first success: Coin turns up heads (H) with probability p and tails (T) with probability $1 - p$. How many independent flips X are needed until getting the first H.**

$$\begin{aligned} E[X] &= \sum_{j=0}^{\infty} j \Pr[X = j] = \sum_{j=1}^{\infty} j(1-p)^{j-1}p \\ &= \frac{p}{1-p} \sum_{j=1}^{\infty} j(1-p)^j = (\frac{p}{1-p})(\frac{1-p}{p^2}) = \frac{1}{p}. \end{aligned}$$

This result is known as waiting-time bound.

Properties of expectation

- **If X is a 0/1 random variable, $E[X] = \Pr[X = 1]$.**

  Proof.

$$E[X] = \sum_{j=0}^{\infty} v_j \Pr[X = j] = \sum_{j=0}^{1} j \Pr[X = j] = \Pr[X = 1].$$

□

  ($\Pr[X = j] = 0$ **for** $j \geq 2$**,** $j \Pr[X = j] = 0$ **for** $j = 0$**)**

- **Linearity of expectation: Given two random variables X and Y defined over the same sample space, $E[X + Y] = E[X] + E[Y]$.**

- **Application: Decouple a complex calculation into simpler ones.**

Guessing cards

- **Game: Shuffle n distinct cards, turn them over one at a time, try to guess each card.**
- **Memoryless guess: No psychic abilities, cannot even remember what has been turned over already; guess a card from full deck uniformly at random.**
- **Claim: The expected number of correct guesses is 1.**

Proof. Let $X_i = 1$ if the ith guess is correct, otherwise $X_i = 0$.

Let X be the number of correct guesses, that is, $X = X_1 + .. + X_n$.

$$E[X_i] = \Pr[X_i = 1] = \frac{1}{n}.$$

$$E[X] = \sum_{i=1}^{n} E[X_i] = \sum_{i=1}^{n} \frac{1}{n} = 1.$$

□

- **Guess with memory: Guess a card uniformly at random from cards not seen yet.**
- **Claim: The expected number of correct guesses is $\Theta(\log n)$.**

Proof. Let $X_i = 1$ if the ith guess is correct, otherwise $X_i = 0$.

Let X be the number of correct guesses, that is, $X = X_1 + .. + X_n$.

$$E[X_i] = \Pr[X_i = 1] = \frac{1}{n-(i-1)}.$$

$$E[X] = \sum_{i=1}^{n} E[X_i] = \sum_{i=1}^{n} \frac{1}{n-(i-1)} = \frac{1}{n} + .. + \frac{1}{2} + 1 = H(n).$$

$\ln(n+1) \leq H(n) \leq 1 + \ln n.$ □

- **Coupon collector: Each box of cereal contains a coupon. There are n different types of coupons. Assume all boxes are equally likely to contain each coupon, how many boxes does one need to open before you have at least 1 coupon of each type.**
- **Claim: The expected number of steps is $\Theta(n \log n)$.**

Proof. Let phase j be the time between one has j and $j+1$ distinct coupons. Then $\Pr[\text{get a new coupon at each open in phase } j] = (n-j)/n$.

Let X_j be the number of steps one spends in phase j. Then $E[X_j] = 1/p = n/(n-j)$.

Let X be the number of steps in total, i.e. $X = X_0 + X_1 + .. + X_{n-1}$.

$$E[X] = \sum_{j=0}^{n-1} E[X_j] = \sum_{j=0}^{n-1} \frac{n}{n-j} = n \sum_{i=1}^{n} \frac{1}{i} = nH(n) = \Theta(n \log n).$$

□

Randomized algorithm for stable matching problem

- **Given a set $M = \{m_1, .., m_n\}$ employers and a set $W = \{w_1, .., w_n\}$ students. each $m \in M$ has a preference list on students defined by bijection $f_m : W \to \{1, .., n\}$, each $w \in W$ has a preference list on employers defined by bijection $g_w : M \to \{1, .., n\}$.**
- **A matching S between M and W is stable if**
  1. **S is a perfect matching; and**
  2. **for any $(m, w'), (m', w) \in S$, $f_m(w') > f_m(w)$ or $g_w(m') > g_w(m)$.**
- **Problem: find a stable matching S.**

Randomzed-Stable-Matching

Input: M**,**W **and preference lists for every** $w \in W$**. Output: A stable matching** S**.**

$S = \emptyset$**; mark every** $m \in M$ **and every** $w \in W$ **unmatched;**

while $\exists$ **unmatched** $m \in M$ **do**

  select an unmatched m **uniformally at random;**

  select a $w \in W$ **uniformally at random;**

  if w **is not matched then** $S = S \cup \{(m, w)\}$ **in** S**; mark** m **and** w **matched;**

  else

    let (m', w) **be the pair in** S**;**

    if $g_w(m) > g_w(m')$ **then**

      remove (m', w) **from** S**;** $S = S \cup \{(m, w)\}$**; mark** m' **unmatched;**

return S

The algorithm finds a stable matching in $O(n \log n)$ **expected running time.**

Proof. By a similar analysis for G-S stable matching algorithm, the algorithm finds a stable matching. The expected number of iterations of the while loop is the same as that of collecting all of n types of coupons, which is $O(n \log n)$, □

Maximum 3-Satisfiability

- **Max 3-Sat: Given a 3-CNF, find an assignment that satisfies as many clauses as possible. The problem is NP-hard.**
- **A simple random assignment: Assign each variable $x_i = 1$ with probability $1/2$ independently.**
- **A clause of 3 literals is satisfied with probability $7/8$.**
- **Lemma: Given a 3-SAT formula of n variables and m clauses, the expected number of clauses satisfied by a random assignment is $7m/8$.**

  Proof. Let Z_j be the random variable with $Z_j = 1$ if clause C_j is satisfied, otherwise $Z_j = 0$. Let Z be the number of clauses satisfied.

$$E[Z] = \sum_{j=1}^{m} E[Z_j] = 7m/8.$$

□

- **Corollary: For any instance of 3-SAT of n variables and m clauses, there exists a truth assignment that satisfies at least $7m/8$ clauses.**

  Proof. Let $s_1, .., s_r, r = 2^n$, be the all possible assignments to the n Boolean variables. Let m_i be the number of clauses satisfies by s_i and $m^* = \max_{i=1}^{r} m_i$. Let Z be the number of clauses satisfied by a randomly selected assignment. Then

$$E[Z] = 7m/8 = \sum_{i=1}^{r} m_i \Pr[Z = s_i] \leq m^* \sum_{i=1}^{r} \Pr[Z = s_i] = m^*.$$

  Thus, there exits an assignment satisfying at least $7m/8$ clauses. □

- **Probabilistic method Prove the existence of a non-obvious property by showing that a random construction produces it with positive probability.**

- **Is it possible to get a $(7/8)$-approximation algorithm for the Max 3-Sat problem based the Corollary? It is possible, but there is a hurdle: the probability that a random assignment satisfies $\geq 7m/8$ clauses is not high.**
- **Lemma: The probability that a random assignment satisfies at least $7m/8$ clauses is at least $1/(8m)$.**

  Proof. Let p_j be the probability that j clauses are satisfied and $p = \sum_{j \geq 7m/8} p_j$ be the probability that at least $7m/8$ clauses are satisfied. Then

$$\frac{7m}{8} = E[Z] = \sum_{j \geq 0} j p_j = \sum_{j < 7m/8} j p_j + \sum_{j \geq 7m/8} j p_j.$$

If $7m/8$ is an integer, then

$$\begin{aligned}
\frac{7m}{8} &= \sum_{j\geq 0} jp_j = \sum_{j<7m/8} jp_j + \sum_{j\geq 7m/8} jp_j \\
&= \sum_{j\leq 7m/8-1} jp_j + \sum_{j\geq 7m/8} jp_j \\
&\leq \sum_{j\leq 7m/8-1} (7m/8-1)p_j + \sum_{j\geq 7m/8} mp_j \\
&= (\frac{7m}{8}-1)\sum_{j\leq 7m/8-1} p_j + m\sum_{j\geq 7m/8} p_j \leq (\frac{7m}{8}-1)\cdot 1 + mp.
\end{aligned}$$

From this, $mp \geq 1$ and $p \geq 1/m \geq 1/(8m)$.

If $7m/8$ is not an integer, then $\lfloor 7m/8 \rfloor \leq 7m/8 - (1/8)$ and

$$\begin{aligned}
\frac{7m}{8} &= \sum_{j \geq 0} jp_j = \sum_{j < 7m/8} jp_j + \sum_{j \geq 7m/8} jp_j \\
&= \sum_{j \leq \lfloor 7m/8 \rfloor} jp_j + \sum_{j \geq \lceil 7m/8 \rceil} jp_j \\
&\leq \sum_{j \leq (7m/8)-(1/8)} (\frac{7m}{8} - \frac{1}{8})p_j + \sum_{j \geq 7m/8} mp_j \\
&= (\frac{7m}{8} - \frac{1}{8}) \sum_{j < 7m/8} p_j + m \sum_{j \geq 7m/8} p_j \leq (\frac{7m}{8} - \frac{1}{8}) \cdot 1 + mp.
\end{aligned}$$

From this, $mp \geq (1/8)$ and $p \geq 1/(8m)$. □

- **For a 3-CNF of n variables and m clauses, the probability that a random assignment satisfies at least $7m/8$ clauses is at least $1/(8m)$.**
- **One idea to increase the probability of satisfying $\geq 7m/8$ clauses is to repeatedly generate random assignments until one of them satisfies $\geq 7m/8$ clauses (Johnson's algorithm).**
- **Theorem: Johnson's algorithm is a $(7/8)$-approximation algorithm.**

  Proof. By the previous lemma, one assignment succeeds with probability $\geq 1/(8m)$. By the waiting-time bound, the expected number of trials to find the satisfying assignment is at most $8m$. □

Derandomization: a deterministic $(7/8)$-approximation algorithm for Max 3-Sat.

- **Set each variable x_i of $x_1, .., x_n$ to 1 or 0 in the order $1 \leq i \leq n$.**
- **Let $E[Z|x_1 = b], b \in \{0, 1\}$, be the expected number of satisfied clauses with $x_1 = b$ and every of $x_2, .., x_n$ is assigned 1 or 0, each with probability $1/2$ independently.**

  If $E[Z|x_1 = 1] \geq E[Z|x_1 = 0]$ then set $x_1 = 1$ else set $x_1 = 0$.
- **Assume $x_1, .., x_i$ have been set ($x_1 = b_1, .., x_i = b_i, b_1, .., b_i \in \{0, 1\}$). Let $E[Z|x_1 = b_1, .., x_i = b_i, x_{i+1} = b], b \in \{0, 1\}$, be the expected number of satisfied clauses with $x_{i+1} = b$ and every of $x_{i+2}, .., x_n$ is assigned 1 or 0, each with probability $1/2$ independently.**

  If
  $E[Z|x_1 = b_1, .., x_i = b_i, x_{i+1} = 1] \geq E[Z|x_1 = b_1, .., x_i = b_i, x_{i+1} = 0]$
  then set $x_{i+1} = 1$ else set $x_{i+1} = 0$.

Compute $E[Z|x_1 = b_1, .., x_i = b_i]$**.**

$$E[Z|x_1 = b_1, .., x_i = b_i] = \sum_{j=1}^{m} E[Z_j|x_1 = b_1, .., x_i = b_i]$$

$$= \sum_{j=1}^{m} \Pr[C_j \textbf{ satisfied}|x_1 = b_1, .., x_i = b_i].$$

$\Pr[C_j$ **satisfied**$|x_1 = b_1, .., x_i = b_i] = 1$ **if** C_j **is satisfied by one of** $x_1 = b_1, .., x_i = b_i$**, otherwise** $\Pr[C_j$ **satisfied**$|x_1 = b_1, .., x_i = b_i] = (1 - (\frac{1}{2})^q)$**, where** q **is the number of literals not assigned a value yet.**

Example: For $C_j = (x_1 \vee \overline{x}_5 \vee x_7)$**,**

$\Pr[C_j$ **satisfied**$|x_1 = 1, x_2 = 0, x_3 = 1] = 1$**,**

$\Pr[C_j$ **satisfied**$|x_1 = 0, x_2 = 0, x_3 = 1] = 1 - (\frac{1}{2})^2 = \frac{3}{4}$**, and**

$\Pr[C_j$ **satisfied**$|x_1 = 0, .., x_5 = 1, .., x_7 = 0] = 0$**.**

Given a 3-CNF of n variables and m clauses, the algorithm finds a truth assignment satisfying at least $(7m/8)$ clauses in $O(mn)$ time.

Proof. We show $E[Z|x_1 = b_1, .., x_n = b_n] \geq E[Z] \geq (7m/8)$ by induction. For $i = 1$,

$$\begin{aligned} E[Z] &= E[Z|x_1 = 1]\Pr[x_1 = 1] + E[Z|x_1 = 0]\Pr[x_1 = 0] \\ &= \frac{1}{2}(E[Z|x_1 = 1] + E[Z|x_1 = 0]). \end{aligned}$$

If $E[Z|x_1 = 1] \geq E[Z|x_1 = 0]$ then the algorithm sets $x_1 = 1$ and $E[Z|x_1 = 1] \geq E[Z] \geq (7m/8)$.

If $E[Z|x_1 = 0] > E[Z|x_1 = 1]$ then the algorithm sets $x_1 = 0$ and $E[Z|x_1 = 0] \geq E[Z] \geq (7m/8)$.

Assume the statement is true for $i \geq 1$ ($E[Z|x_1 = b_1, .., x_i = b_i] \geq E[Z] \geq (7m/8)$), we prove it for $i + 1$.

$$\begin{aligned} E[Z|x_1 = b_1, ..x_i = b_i] &= E[Z|x_1 = b_1, .., x_i = b_i, x_{i+1} = 1] \Pr[x_{i+1} = 1] \\ &\quad + E[Z|x_1 = b_1, .., x_i = b_i, x_{i+1} = 0] \Pr[x_{i+1} = 0] \\ &= \frac{1}{2} E[Z|x_1 = b_1, .., x_i = b_i, x_{i+1} = 1] \\ &\quad + \frac{1}{2} E[Z|x_1 = b_1, .., x_i = b_i, x_{i+1} = 0]. \end{aligned}$$

If $E[Z|x_1 = b_1, .., x_i = b_i, x_{i+1} = 1] \geq E[Z|x_1 = b_1, .., x_i = b_i, x_{i+1} = 0]$ then the algorithm sets $x_{i+1} = 1$ and
$E[Z|x_1 = b_1, .., x_i = b_i, x_{i+1} = 1] \geq E[Z|x_1, .., x_i = b_i] \geq E[Z] \geq (7m/8)$.

If $E[Z|x_1 = b_1, .., x_i = b_i, x_{i+1} = 0] > E[Z|x_1 = b_1, .., x_i = b_i, x_{i+1} = 1]$ then the algorithm sets $x_{i+1} = 0$ and
$E[Z|x_1 = b_1, .., x_i = b_i, x_{i+1} = 0] \geq E[Z|x_1, .., x_i = b_i] \geq E[Z] \geq (7m/8)$.

It takes $O(m)$ time to assign a value to each x_i. The total running time is $O(mn)$, □

Two properties of the randomized Max 3-SAT algorithm

- **Each variable is set independently**
- **Conditional expectations can be computed in polynomial time**

The derandomization of the randomized Max 3-SAT algorithm is an example of method of conditional expectations to derandomize random algorithms with the two properties.

- **Extensions of Max 3-SAT**
  - **MAX-SAT: allow one, two, or more literals per clause.**
  - **Weighted MAX-SAT: each clause is given a non-negative weight, find an assignment to maximize the sum of the weights of the satisfied clauses.**
- **Theorem [Asano-Williamson 2000]: There exists a 0.784-approximation algorithm for MAX-SAT.**
- **Theorem [Karloff-Zwick 1997,Zwick+computer 2002]: There exists a $(7/8)$-approximation algorithm for the MAX-3SAT with each clause having at most 3 literals.**
- **Theorem [Håstad 1997]: Unless P=NP, no ρ-approximation algorithm for MAX-3SAT (hence MAX-SAT) for any $\rho > 7/8$.**

A randomized algorithm for k-SAT [Schöning 1999]

Random-k-SAT

Input a k-CNF Φ of n variables and m clauses;

Let $f : \{x_1, .., x_n\} \rightarrow \{0, 1\}$ be a randomly selected truth assignment;

Repeat $O(n)$ times

  If Φ is satisfied by f then output f and terminate

  else

    Let C be an unsatisfied clause;

    Select one literal l of C uniformly at random;

    Update f by flipping the value of l

Algorithm Random-k-SAT, given a satisfiable k-CNF Φ of n variables, finds a satisfying assignment for Φ with probability at least $(\frac{k}{2(k-1)})^n$.

Proof. Let f^* be a satisfying assignment for Φ and f be a randomly selected assignment. Let $d_H(f, f^*) = |\{i|f(x_i) \neq f^*(x_i)\}|$ be the Hamming distance between f and f^*. For each $0 \leq d \leq n$, $\Pr[d_H(f, f^*) = d] = \binom{n}{d}(\frac{1}{2})^n$. Start from f with $d_H(f, f^*) = d$, after one update, $\Pr[d_H(f, f^*) = d-1] \geq 1/k$, $\Pr[d_H(f, f^*) = d+1] \leq 1-(1/k)$, and by a Markov chain analysis, the probability that the algorithm finds a satisfying assignment is at least $(\frac{1}{k-1})^d$. The probability that the algorithm finds a satisfying assignment is

$$\sum_{d=0}^{n} \binom{n}{d}(\frac{1}{2})^n(\frac{1}{k-1})^d = (\frac{k}{2(k-1)})^n.$$

□

An $O^*((\frac{2(k-1)}{k})^n)$ time randomized algorithm for k-SAT ($O^*((\frac{4}{3})^n)$ time for 3-SAT) can be obtained by repeating Algorithm Random-k-SAT. There is an $O^*((\frac{2(k-1)}{k}+\epsilon)^n)$ time deterministic algorithm for k-SAT ($O^*(1.334^n)$ time for 3-SAT) [Moser and Scheder 2010]. O^* notation hides the polynomial factor $\mathrm{Poly}(n, m)$.

Monte Carlo and Las Vegas Algorithms

- **Monte Carlo algorithm: Guaranteed to run in a specific time (e.g., poly-time), likely to find a correct answer (not guaranteed).**

  Example: Contraction algorithm for global min-cut, Random-k-SAT for k-SAT.

- **Las Vegas algorithm: Guaranteed to find a correct answer, likely to run in a specific time (e.g., poly-time) (not guaranteed).**

  Example: Johnson's MAX-3-SAT algorithm.

- **It is always possible to convert a Las Vegas algorithm into a Monte Carlo algorithm (simply stop the algorithm at some point). No general approach is known for converting a Monte Carlo algorithm into a Las Vegas one.**

- **One-sided error:**

  If the correct answer is NO, always return NO.

  If the correct answer is YES, return YES with probability at least $1/2$.

- **Problem class RP [Monte Carlo]: Set of decision problems solvable with one-sided error in poly-time.**

- **Problem class ZPP [Las Vegas]: Set of decision problems solvable in expected poly-time.**

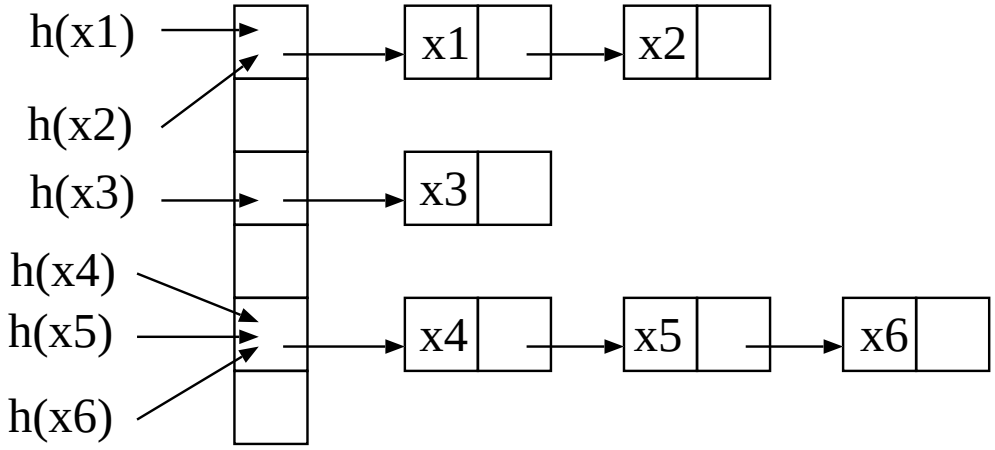
- **Theorem: P$\subseteq$ZPP$\subseteq$RP$\subseteq$NP.**

- **Fundamental open problems: Does P=ZPP? Does ZPP=RP? Does RP=NP?**

Universal Hashing

- **Dictionary: Given a set U of possible elements, maintain a subset $S \subseteq U$ so that inserting, deleting and searching in S is efficient.**
- **Naive approach: Create a table T of size $|U|$ and access $T[x]$ when processing $x \in U$. Challenge: U can be very large and a table of size $|U|$ is infeasible.**
- **Hashing:**
  - **Create a smaller table, put some elements of U in the table.**
  - **To process an $x \in U$, not use x itself, but a function $h(x)$ computed from x to access the table.**
  - **The function is called hash function, the table is called hash table, $h(x)$ is called hash value of x.**
  - **Example: create an array A of size n and a hash function $h : U \to \{0, 1, .., n-1\}$. When processing element x, access array element $A[h(x)]$.**

- **Collision in hashing: $h(x) = h(y)$ but $x \neq y$. A collision is expected after $\Theta(\sqrt{n})$ random insertions of elements into a hash table of size n.**
- **Separate chaining: $A[i]$ stores linked list of elements x with $h(x) = i$.**
- **Time of accessing an element depends on the length of chains.**



- **Good hash functions: Map m elements uniformly at random to n hash slots. Average length of chain = m/n. Choose $m \approx n$, $O(1)$ expect access time.**
- **It is difficult to use a fixed hash function to achieve $O(1)$ access time without the info of the elements (keys) distribution.**
- **Randomized approach: Create a set H of hash functions and select a hash function $h \in H$ at random.**
- **A universal family of hash functions is a set H of hash functions: $U \to \{0, 1, .., n-1\}$ s.t.**
  - **for any $x \neq y$, $\Pr_{h \in H}[h(x) = h(y)] \leq 1/n$;**
  - **selection $h \in H$ at random can be done efficiently; and**
  - **$h(x)$ can be computed efficiently.**

$h_i: U \rightarrow \{0,1\}$, n=2

		a	b	c	d	e	f
Pr=1/2	$h_1(x)$	0	1	0	1	0	1
Pr=1/2	$h_2(x)$	0	0	0	1	1	1

$H = \{h_1, h_2\}$

$Pr_{h \in H}[h(a) = h(b)] = 1/2$

$Pr_{h \in H}[h(a) = h(c)] = 1$

$Pr_{h \in H}[h(a) = h(d)] = 0$

...

not universal

		a	b	c	d	e	f
Pr=1/4	$h_1(x)$	0	1	0	1	0	1
Pr=1/4	$h_2(x)$	0	0	0	1	1	1
Pr=1/4	$h_3(x)$	0	0	1	0	1	1
Pr=1/4	$h_4(x)$	1	0	0	1	1	0

$H = \{h_1, h_2, h_3, h_4\}$

$Pr_{h \in H}[h(a) = h(b)] = 1/2$

$Pr_{h \in H}[h(a) = h(c)] = 1/2$

$Pr_{h \in H}[h(a) = h(d)] = 1/2$

$Pr_{h \in H}[h(a) = h(e)] = 1/2$

$Pr_{h \in H}[h(a) = h(f)] = 0$

...

universal

- **Lemma: Assume a hash function h is selected at random from a universal family H of hash functions and used to hash m keys into a table T of size n. For a key x not in S, the expected length of the list that x hashed to is at most m/n.**

  Proof. Let S be the set of keys in T. For any key $s \in S$, define random variable $X_s = 1$ if $h(x) = h(s)$, otherwise 0. Let X be a random variable counting the total number of collisions with any key in S.

$$\begin{aligned} E_{h \in H}[X] &= E[\sum_{s \in S} X_s] = \sum_{s \in S} E[X_s] = \sum_{s \in S} \Pr[X_s = 1] \\ &\leq \sum_{s \in S} \frac{1}{n} = \frac{|S|}{n} \leq \frac{m}{n}. \end{aligned}$$

  □

- **Corollary: Using a universal hashing and collision resolution by chaining in a table of size n, it takes expected $\Theta(m)$ time to handle any sequence of m table operations containing $O(n)$ insertions.**

- **Constructing a universal family of hash functions**
  - **Let p be a prime number. Identify each element $u \in U$ by a key $x = (x_{r-1}, .., x_0)$ of r-digit base-p integer ($x = \sum_{i=0}^{r-1} x_i p^i, 0 \leq x_i < p$).**
    Example: for $p = 2$ and $x = (x_3, x_2, x_1, x_0) = (1, 0, 1, 1)$,
    $x = 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 = 11$.
  - **Hash function. Let A be the set of all r-digit base-p integers. For each $a = (a_{r-1}, .., a_0) \in A$, define**

$$h_a(x) = (\sum_{i=0}^{r-1} a_i x_i) \bmod p.$$

    Example: for $p = 3$, $a = (2, 0, 2, 1)$ and $x = (1, 2, 0, 1)$,
    $h_a(x) = (2 \cdot 1 + 0 \cdot 2 + 2 \cdot 0 + 1 \cdot 1) \bmod 3 = 3 \bmod 3 = 0$.
  - **Hash function family: $H = \{h_a : a \in A\}$ ($h_a : U \to \{0, 1, .., p-1\}$).**

Theorem: $H = \{h_a : a \in A\}$ **is a universal family of hash functions.**

Proof. For two distinct keys $x = (x_{r-1}, .., x_0)$ and $y = (y_{r-1}, .., y_0)$, we show that $\Pr[h_a(x) = h_a(y)] \leq 1/p$ ($\frac{|\{h_a | h_a(x)=h_a(y)\}|}{|H|} \leq \frac{1}{p}$).
Since $x \neq y$, $x_j \neq y_j$ for some j. Assume $h_a(x) = h_a(y)$. Then
$h_a(x) - h_a(y) = \sum_{i=0}^{r-1} a_i x_i - \sum_{i=0}^{r-1} a_i y_i = \sum_{i=0}^{r-1} a_i(x_i - y_i) = 0$.
Thus, $h_a(x) = h(y)$ iff

$$a_j(y_j - x_j) \equiv \sum_{i \neq j} a_i(x_i - y_i) \bmod p.$$

Let $z = (y_j - x_j)$ and $q = \sum_{i \neq j} a_i(x_i - y_i)$. The equation becomes
$a_j z \equiv q \bmod p$ with a_j randomly selected. We show that $a_j z \equiv q \bmod p$ has at most one solution $0 \leq a_j < p$, and thus $\Pr[h_a(x) = h_a(y)] \leq 1/p$.

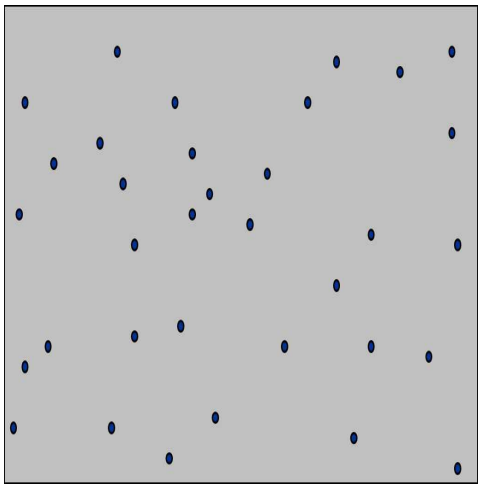
Assume there are two different solutions a_j and a'_j. Then $(a_j - a'_j)z \equiv 0 \bmod p$.
From $z \not\equiv 0 \bmod p$, $(a_j - a'_j) \not\equiv 1 \bmod p$ and z is not divisible by p. Hence,
$(a_j - a'_j)$ is divisible by p. Since p is prime, $(a_j - a'_j) = 0$. □

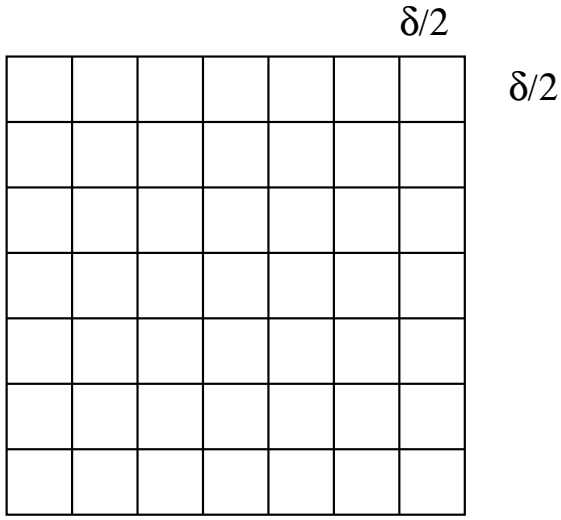
Summary of universal hash

- **Given a set U of elements, maintain a subset $S \subseteq U$ of size m in a table so that each table operation is efficient.**
- **Select prime p with $m \leq p < 2m$ and create $H = \{h_a, a \in A\}$.**
- **Space used: $\Theta(m)$.**
- **Expected number of collisions per table operation is at most 1, implying $O(1)$ expected time per table operation.**

Closest Pair Problem

- **Given n points $p_1, .., p_n$, $p_i = (x_i, y_i)$ in the 2-dimensional Euclidean plane, find a pair of points with the smallest Euclidean distance $\delta = \min_{1 \le i < j \le n} d(p_i, p_j)$ between them.**
- **Assume all points are in the unit square P: $0 \le x_i, y_i \le 1$.**
- **Parititon the unit square P into N^2 $(\delta/2) \times (\delta/2)$ subsquares $P[s,t]$, $1 \le s, t \le N$, for some $0 < \delta < 1$, $N = \lceil 1/(\delta/2) \rceil$.**





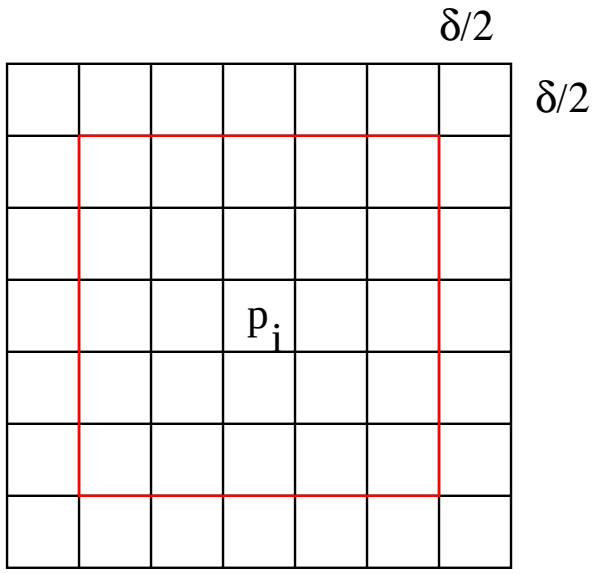
- **If two points p_i, p_j are in a same subsquare, then $d(p_i, p_j) < \delta$.**

  Proof: $d(p_i, p_j) \leq \sqrt{(\delta/2)^2 + (\delta/2)^2} = \delta/\sqrt{2} < \delta$.

- **Let $p_i = (x_i, y_i)$ be in subsqure $P[s, t]$. If subsquare $P[s', t']$ has p_j with $d(p_i, p_j) \leq \delta$, then $|s - s'| \leq 2$ and $|t - t'| \leq 2$.**

  Proof: Assume that $|s - s'| > 2$. Then $|y_i - y_j| > 2(\delta/2) = \delta$. Similarly, $|t - t'| > 2$ implies $|x_i - x_j| > \delta$. Since $d(p_i, p_j) \geq \max\{|x_i - x_j|, |y_i - y_j|\}$, $d(p_i, p_j) > \delta$.

  Let $S(s, t) = \{P[s', t'] || s - s'| \leq 2, |t - t'| \leq 2\}$. Then $|S(s, t)| \leq 25$.



Randomized algorithm for closest pair problem

Order the n **points in a random sequence** $p_1, .., p_n$**;** $\delta = d(p_1, p_2)$**;**

MakeDictionary to store a subset of $(\delta/2) \times (\delta/2)$ **subsquares** $P[s,t]$**;**

/* Create hash table to store subsquares containing a point processed so far

for $i = 1, ..., n$ **do**

find the subsquare $P[s,t]$ **containing** p_i**;**

Let $\delta' = \min_{p_j \in P[s',t'] \ in \ S(s,t), j<i} d(p_i, p_j)$**; /*** $P[s',t']$ **in dictionary has a** p_j

if $\delta' < \delta$ **then**

delete the current dictionary;

MakeDictionary to store a subset of $(\delta'/2) \times (\delta'/2)$ **subsquares** $P[s,t]$**;**

$\delta = \delta'$**;**

for $j = 1, .., i$ **do**

find the subsquare $P[s,t]$ **containing** p_j**;**

insert $P[s,t]$ **into the new dictionary; /* tagged with** j

endfor

else insert $P[s,t]$ **into the current dictionary; /* tagged with** i

endfor

Analysis of the algorithm

- **The algorithm correctly keeps the closest pair of $p_1, .., p_i$ for every $1 \leq i \leq n$. It has $O(n)$ distance computation, $O(n)$ operations to find subsquare $P[s,t]$, and $O(n)$ MakeDictionary operations.**
- **Let X_i be a random variable s.t. $X_i = 1$ if $\delta' < \delta$ when p_i is processed, $X_i = 0$ otherwise. The total number of insertions is $n + \sum_{1 \leq i \leq n} iX_i$.**
- $\Pr[X_i = 1] \leq 2/i$.

  Proof: Let p, q be the two points in $p_1, .., p_i$ s.t. $d(p,q) = \min_{1 \leq i' \neq j \leq i} d(p_{i'}, p_j)$. Since $\delta' < \delta$, either $p = p_i$ or $q = p_i$. As $p_1, .., p_i$ are in a random sequence, $\Pr[p = p_i$ **or** $q = p_i] \leq (1/i) + (1/i) = 2/i$.
- **Expected number of Insertion operations:**

  $E(X) = n + \sum_{1 \leq i \leq n} iE(X_i) \leq n + 2n = 3n$.

- **The algorithm has $O(n)$ expected time plus $O(n)$ expected dictionary operations.**
- **In expectation, the algorithm uses $O(n)$ hash-function computations and $O(n)$ additional time for finding the closest pair of points.**

  Proof idea: Applying a universal hashing scheme, the expected total time for lookup the hash table is $O(n)$ (details omitted).

Cache Maintenance Problem

- **An algorithm S requests a sequence of data blocks $d_1, .., d_m$, data blocks may not be distinct. A cache can keep at most $k < m$ data blocks. For $1 \leq i \leq m$, when the algorithm requests block d_i,**
  - **if d_i is in cache, S accesses d_i in cache (cache hit);**
  - **if d_i is not in cache and cache has less than k data blocks (cache not full), load d_i into cache and S accesses d_i in cache (cache missing);**
  - **if d_i is not in cache and cache has k blocks (cache full), evict one block from cache and load d_i into cache, and A accesses d_i in cache (cache missing).**
- **Optimization goal: minimize the total number of evictions.**

A greedy algorithm (Furthest-in-future)

- **When a request to a block d not in cache and cache is full, evict a block in cache that comes furthest in the future in the sequence.**
- **Example, assume cache has size $k = 6$ and the request sequence is:**

$$a, b, c, d, e, f, g, a, b, c, e, d, a, b, b, a, c, d, e, a, f, a, d, e, f, g, h$$

After the 1st six requests, cache holds a, b, c, d, e, f and is full.

The next request is to block g, block f in cache comes furthest in the future in the sequence, so evict f and put g into cache, and cache holds a, b, c, d, e, g right after the request to g.

Furthest-in-future algorithm S_{FF}

- **Let $d_1, .., d_n$ be the request sequence. For each d_j, let n_j be the smallest index such that $j < n_j$ and $d_j = d_{n_j}$ (n_j is defined $+\infty$ if $d_j \neq d_l$ for every $l > j$).**

  $C = \emptyset$. /* Initially, cache C is empty

  For $i = 1$ to n

  if $d_i = d_j$ for some $d_j \in C$ then change the index j to i

  else if $|C| < k$ then $C = C \cup \{d_i\}$

  else $\{d_{j^*} = \arg\max_{d_j \in C} n_j$; $C = (C \setminus \{d_{j^*}\}) \cup \{d_i\}\}$;

  endif

  endif

  endfor

Blocks	a	b	a	b	c	b	c	a	a	b
Cache k=2	a	a	a	a	c	c	c	a	a	a
		b	b	b	b	b	b	b	b	b

Theorem FF: Algorithm furthest-in-future S_{FF} has the minimum number of evictions.

Proof. For a sequence $\sigma = d_1, .., d_m$ of blocks and $1 \leq i \leq m$, let σ_i be the subsequence of the first i blocks $d_1, .., d_i$. For a schedule S, let $f_S(\sigma_i)$ be the number of evictions by S on σ_i, $f_S(\sigma_m) = f_S(\sigma)$.

A schedule is reduced if it only puts a block in cache when the block is requested, otherwise unreduced.

When a data block is put into cache at the time it is not requested, we count this as an unreduced event.

The proof is divided into two parts. Part 1: for any unreduced schedule S, there is a reduced schedule S' s.t. $f_{S'}(\sigma) \leq f_s(\sigma)$. Part 2: For any reduced schedule S, $f_{S_{FF}}(\sigma) \leq f_S(\sigma)$.

- Proof for Part 1

  Claim: Any unreduced schedule S can be transformed to a reduced schedule S' s.t. $f_{S'}(\sigma) \leq f_S(\sigma)$.

  We prove the claim by induction on the number i of unreduced events.

  For $i = 0$, claim holds. Assume claim true for $0 \leq i - 1$ and prove it for i. Let d be the block put in cache at the time t in the ith unrduced event. Then d is either evicted at time $t' > t$ before the next request to d or d is requested at time $t' > t$ before d is evicted. We simply transform S to S' by removing the operation to put d into cache at time t.

- Proof for Part 2

  Claim: For any reduced schedule S, $f_S(\sigma) \geq f_{S_{FF}}(\sigma)$.

  We prove the following statement by induction: Let S be any reduced schedule that $f_S(\sigma_i) \geq f_{S_{FF}}(\sigma_i)$ for $i \geq 0$. Then there is a reduced schedule S' that S and S' have same cache contents and $f_S(\sigma_j) \geq f_{S'}(\sigma_j) \geq f_{S_{FF}}(\sigma_j)$ for some j with $i+1 \leq j \leq m$.

  For $i = 0$ and any schedule S, $f_S(\sigma_i) = f_{S_{FF}}(\sigma_i)$. When access block $d_{i+1} = d_1$, let S' follow what S does. Then S and S' have same cache contents and $f_S(\sigma_{i+1}) = f_{S'}(\sigma_{i+1}) \geq f_{S_{FF}}(\sigma_{i+1}) = 0$. The statement is true and we get the induction base.

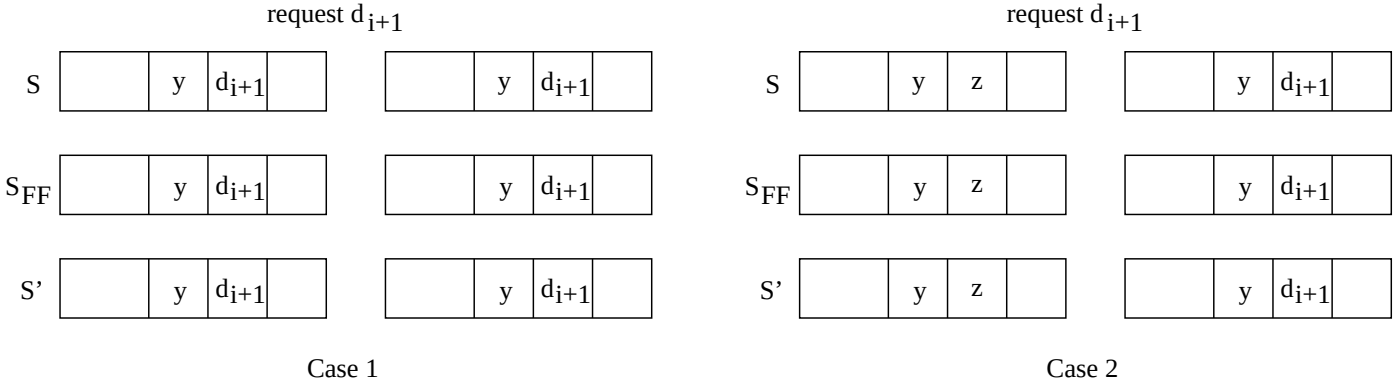
(Proof for Part 2 continued)

Assume the statement holds for $i \geq 0$ and prove it for $i + 1$.

When access d_{i+1}, S and S' have same cache contents. There are 3 cases:

1. d_{i+1} is in cache and no eviction by S;
2. d_{i+1} is not in cache, S and S_{FF} evict a same block from cache;
3. d_{i+1} is not in cache, S evicts block z and S_{FF} evicts $y \neq z$.

For Cases 1 and 2, let S' follow S. Then S and S' have same cache contents and $f_S(\sigma_{i+1}) = f_{S'}(\sigma_{i+1}) \geq f_{S_{FF}}(\sigma_{i+1})$.



(Proof for Part 2 continued)

For Case 3, first let S' follow S_{FF} to evict y. Let j be the minimum index s.t. $i+2 \leq j$ and one of the following events happens:

(A) before z is put back to cache, S evicts y for a request to $x \neq y$;

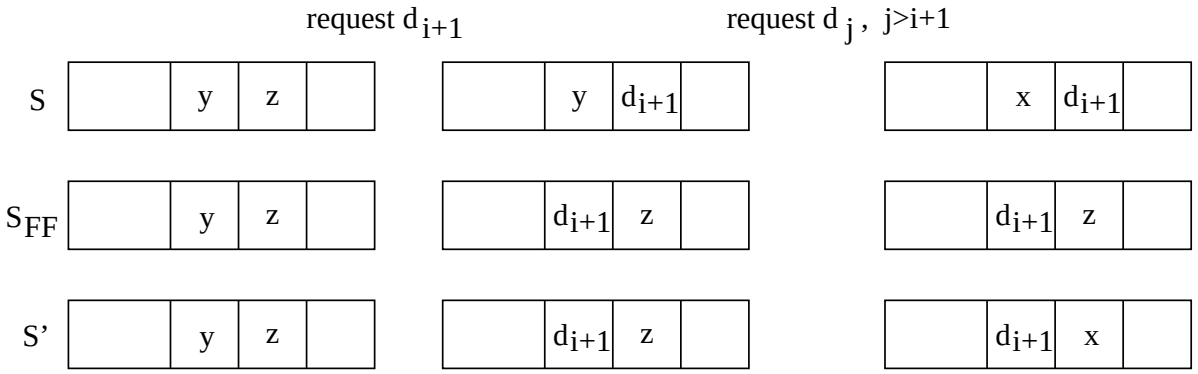
(B) S evicts some w for a request to z.

Then S' follows S on $d_{i+2}, .., d_{j-1}$.

When a request for d_j:

(A) $d_j = x \neq y$, z not in cache, and S evicts y to put x in cache.

In this case, S' does not have x in cache. So, S' evicts z to put x in cache. After this, S' and S have same cache contents and $f_S(\sigma_j) = f_{S'}(\sigma_j) \geq f_{S_{FF}}(\sigma_j)$.



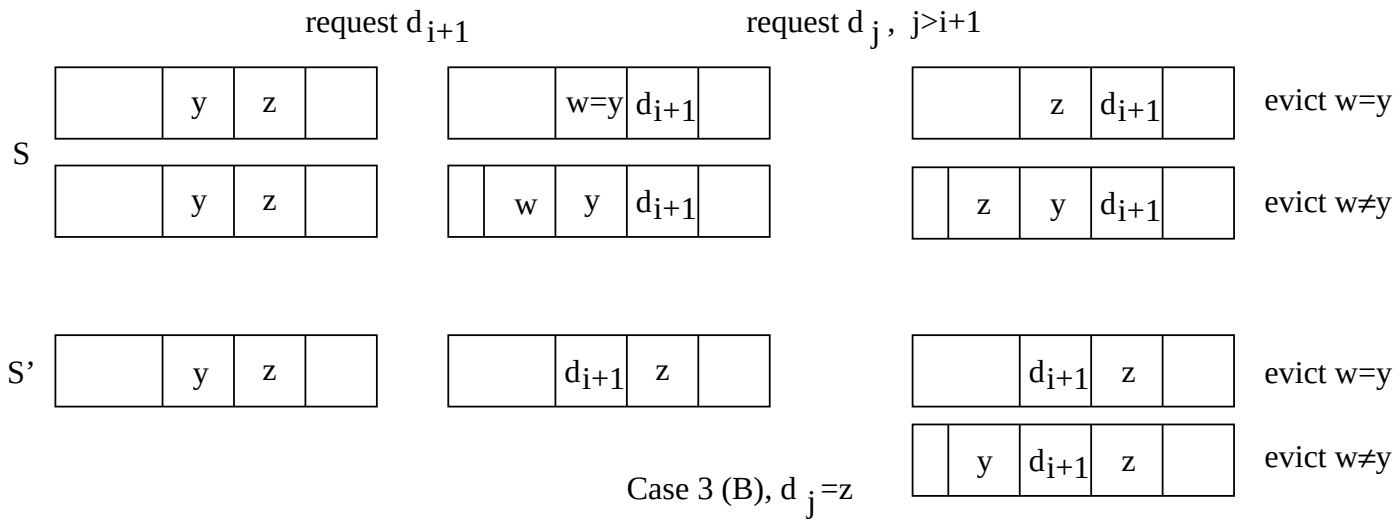
Case 3 (A), $d_j = x \neq y$

(B) $d_j = z$ and S evicts w.

If $w = y$, then S' does not evict any block. After this, S and S' have same cache contents and $f_S(\sigma_j) > f_{S'}(\sigma_j) \geq f_{S_{FF}}(\sigma_j)$. If $w \neq y$, then S' evicts w and put y into cache. After this, S and S' have same cache contents. S' becomes an unreduced schedule. By the proof for Part 1, we can tranform S' into reduced. Then $f_S(\sigma_j) \geq f_{S'}(\sigma_j) \geq f_{S_{FF}}(\sigma_j)$.

Summarize above, the statement is proved.

From the proofs for Part 1 and Part2, the theorem holds.



□

An online algorithm

- **When algorithm A requests block d_i, no knowledge on future blocks $d_{i+1}, .., d_m$.**
- **Least-recently-used (LRU) algorithm**

  For $k < i \leq m$, let $d_{j_1}, .., d_{j_k}$ be the blocks in cache and $t_1, ..., t_p$ be the sequence of accesses to the blocks in cache. For each d_{j_l} in cache, let q_{j_l} be the maximum index s.t. $q_{j_l} \leq p$ and d_{j_l} is accessed in $t_{q_{j_l}}$.

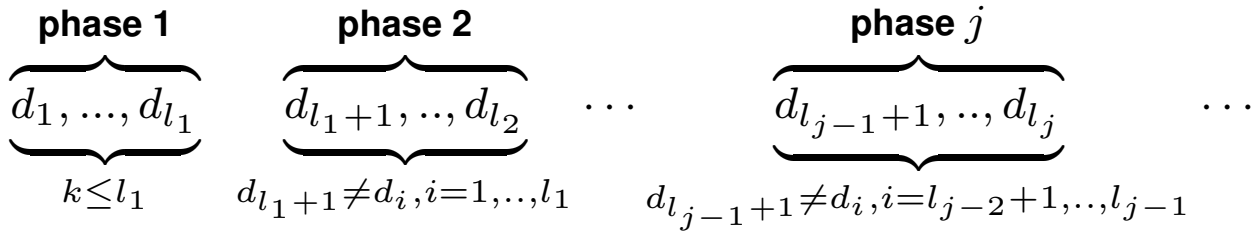
  For $1 \leq i \leq m$ do

  if cache is full then let $j^* = \arg\min_{q_{j_l}} d_{j_l}$, evict d_{j^*} and load d_i to cache.
- **LRU algorithm has at most $k \cdot \text{opt} + k$ evictions, where opt is the value of optimal solution (proof later).**

Blocks	a	b	a	b	c	b	c	a	a	b
Cache k=2	a	a	a	a	c	c	c	c	c	b
		b	b	b	b	b	b	a	a	a

A general template, marking algorithms

- **Consider requests to data blocks $d_1, d_2, .., d_m$ as phases of requests, each phase requested k distinct blocks, a new phase starts when a block different from the k blocks is requested.**



- **At start of a phase, every block is unmarked (not recently requested).**

  When a block d is requested in the phase, d is marked (recently requested).

  When an eviction is needed, evict an unmarked block.

  When all blocks in cache become marked and an eviction is needed, finish the current phase and start a new phase.

- **Marking algorithms, evict a block not recently requested.**

  Initially, every block not requested is considered not recently requested

  $i = 1$;

  while $i \leq m$ **do**

    unmark all blocks in cache C; /***start of a phase,**

      mark every block in C **not recently requested**

    while ($C \neq$ full **or** $\exists d \in C$ **unmarked) and** $i \leq m$ **do**

      mark d_i; /***mark** d_i **recently requested**

      if $C \neq$ full **and** $d_i \notin C$ **then load** d_i **into** C

      else if ($C =$ full **and** $\exists d \in C$ **unmarked) and** $d_i \notin C$ **then**

        evict an unmarked block; load d_i **into** C; $i = i + 1$

      endif

    endif

    endwhile /* end of a phase

  endwhile

- **LRU algorithm is a marking algorithm**

Analysis of marking algorithm

- **In each phase, there are at most k evictions.**

  Proof: After k evictions, every block in C becomes marked.

- **If there are r phases, there are at most kr evictions.**

- **An optimal algorithm has at least $r-1$ evictions.**

  Proof: Let S_i be the subsequence of blocks processed in the ith phase. Then for each S_i with $1 \leq i \leq r-1$, S_i has $k+1$ distinct blocks and an optimal algorithm must make at least one eviction.

- **For any marking algorithm, the number of evictions is at most $k \cdot opt + k$.**

  Proof: $kr = k(r-1) + k \leq k \cdot \mathrm{opt} + k$.

- **An example: assume $m/(k+1) = l$ is an integer and $d_1, .., d_m = S_1, S_2, .., S_l$, where each $S_i = d_1, .., d_k, d_{k+1}$. Then $\mathrm{opt} = m/k$ and the number of evictions by LRU is $m - k = k(m/k) - k$.**

Randomized marking algorithm

- **Initially, every block not requested is considered not recently requested**

  $i = 1$;
  while $i \leq m$ **do**
    unmark all blocks in cache C; /***start of a phase,**
      mark every block in C **not recently requested**
    while ($C \neq$ full **or** $\exists d \in C$ **unmarked) and** $i \leq m$ **do**
      mark d_i; /***mark** d_i **recently requested**
      if $C \neq$ full **and** $d_i \notin C$ **then load** d_i **into** C
      else if ($C =$ full **and** $\exists d \in C$ **unmarked) and** $d_i \notin C$ **then**
        evict an unmarked block selected uniformly at random;
        load d_i **into** C; $i = i + 1$
      endif
      endif
    endwhile /* end of a phase
  endwhile

Analysis of randomized marking algorithm

- **A block d requested in phase j is called fresh if d is not marked (requested) in phase $j-1$, otherwise called stale. Let c_j be the number of fresh blocks in phase j. Let r be the number of phases. Let $f^*(\sigma)$ be the number of evictions by an optimal schedule.**
- **Claim 1:** $f^*(\sigma) \geq (1/2)\sum_{j=2}^{r} c_j$.

  Proof. Let $f_j^*(\sigma)$ be the number of evictions in phase j by the optimal schedule. Then $f^*(\sigma) = \sum_{j=1}^{r} f_j^*(\sigma)$. In any phase $j \geq 1$, there are k requests to k distinct blocks and there are requests to c_{j+1} distinct fresh blocks in phase $j+1$. So there are at least $k + c_{j+1}$ requests to distinct blocks in phases j and $j+1$. This implies that the optimal schedule has at least c_{j+1} evictions in phases j and $j+1$. So, $f_j^*(\sigma) + f_{j+1}^*(\sigma) \geq c_{j+1}$. Then,

$$2f^*(\sigma) = 2\sum_{j=1}^{r} f_j^*(\sigma) \geq \sum_{j=1}^{r-1}(f_j^*(\sigma) + f_{j+1}^*(\sigma)) \geq \sum_{j=1}^{r-1} c_{j+1}.$$

  □

- **Claim 2: Let X be the number of evictions by the randomized algorithm. For every sequence σ, $E[X] \leq H_k \sum_{j=2}^{r} c_j$, H_k is the harmonic function.**

  Proof. There are k requests to unmarked blocks in each phase, c_j blocks are fresh and $k - c_j$ are stale. Let X_j be the number of evictions in phase $j \geq 2$. Then $X_j \geq c_j$ since each fresh block is not marked in phase $j - 1$ and is not in cache when requested in phase j.

  For a stale block d, d may be in cache when requested in phase j. If d is in cache then no eviction happens, otherwise an eviction happens. What is the probability an eviction happens?

  At the beginning of a phase, all k blocks are stale by the initialization of a phase. For the lth request to a stale block d initially in cache, when d is requested, $l - 1$ initially stale blocks become marked and $k - l + 1$ blocks remain stale.

Let c be the number of requests to fresh blocks before d is requested. Then c initially stale blocks are evicted and not in cache. Each of the $k-l+1$ has the same probability not in cache. So $\Pr[d \text{ not in cache}] = \frac{c}{k-l+1} \leq \frac{c_j}{k-l+1}$. Then

$$E[X_j] \leq c_j + \sum_{l=1}^{k-c_j} \frac{c_j}{k-l+1} \leq c_j(1 + \sum_{l=c_j+1}^{k} \frac{1}{l}) = c_j(1+H_k - H_{c_j}) \leq c_j H_k.$$

From this,

$$E[X] = \sum_{j=2}^{r} E[X_j] \leq H_k \sum_{j=2}^{r} c_j .$$

□

- **Theorem: The expected number of evictions by the randomized marking algorithm is $O((\log k) \cdot f^*(\sigma))$.**

  Proof. The theorem follows from Claims 1 and 2. □

Chernoff Bounds

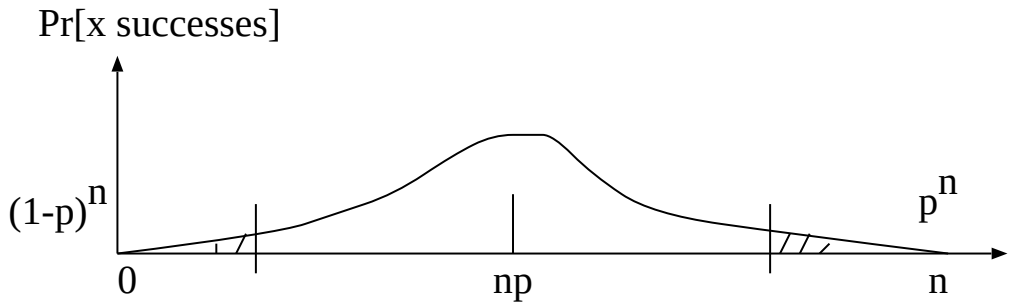
- **Theorem (above mean): Assume $X_1, .., X_n$ are independent $0-1$ random variables. Let $X = X_1 + .. + X_n$. Then for any $\mu \geq E[X]$ and any $\delta > 0$,**

$$\Pr[X > (1+\delta)\mu] < \left(\frac{e^{\delta}}{(1+\delta)^{1+\delta}}\right)^{\mu}$$

  Intuition: Sum of independent $0-1$ random variables is tightly centred on the mean.

- **Looser but more convenient bound in practice**

$$\Pr[X > (1+\delta)\mu] \leq \left(\frac{1}{e}\right)^{\delta^2\mu/(2+\delta)}$$



number of successes in n 0-1 trials, each trial has success probability p.

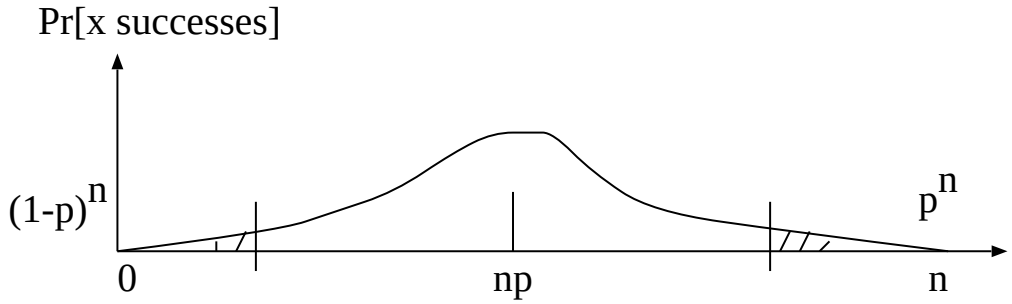
- **Theorem (below mean): Assume $X_1, .., X_n$ are independent $0-1$ random variables. Let $X = X_1 + .. + X_n$. Then for any $\mu \leq E[X]$ and any $0 < \delta < 1$,**

$$\Pr[X < (1-\delta)\mu] < \left(\frac{e^{-\delta}}{(1-\delta)^{1-\delta}}\right)^{\mu}$$

Intuition: Sum of independent $0-1$ random variables is tightly centred on the mean.

- **Looser but more convenient bound in practice**

$$\Pr[X < (1-\delta)\mu] \leq \left(\frac{1}{e}\right)^{\delta^2 \mu/2}$$



number of successes in n 0-1 trials, each trial has success probability p.

Load Balancing

- **Load balancing problem**

  Instance: System in which m jobs arrive in a stream and need to be processed immediately on n identical processors.

  Objective: Find an assignment to balance the workload across processors.

- **Centralized controller: Assign jobs in round-robin manner. Each processor receives at most $\lceil m/n \rceil$ jobs.**

- **Decentralized controller: Assign jobs to processors uniformly at random.**

  How likely is it that some processor is assigned "too many" jobs?

- **Analysis Let X_i be the number of jobs assigned to processor i.**

  Let $Y_{ij} = 1$ if job j assigned to processor i, otherwise 0.

  Then $E[Y_{ij}] = 1/n$, $X_i = \sum_{1 \le j \le m} Y_{ij}$ and $\mu = E[X_i] = m/n$.

  Applying Chernoff bounds with $\delta = c - 1$ gives

$$\Pr[X_i > c\frac{m}{n}] < \left(\frac{e^{c-1}}{c^c}\right)^{m/n}.$$

  Let $\gamma(n)$ be the number x s.t. $x^x = n$, and choose $c = e\gamma(n)$.

$$\begin{aligned}\Pr[X_i > c\frac{m}{n}] &< \frac{e^{(c-1)\frac{m}{n}}}{c^{c\frac{m}{n}}} < \left(\frac{e}{c}\right)^{c\frac{m}{n}} \\ &= \left(\frac{1}{\gamma(n)}\right)^{e\gamma(n)\frac{m}{n}} < \left(\frac{1}{\gamma(n)}\right)^{2\gamma(n)\frac{m}{n}} = \frac{1}{n^{2m/n}}.\end{aligned}$$

  Union bound implies that with high probability at least $1 - 1/n$ no processor receives more than $e\gamma(n)\frac{m}{n} = \Theta(\frac{\log n}{\log\log n}\frac{m}{n})$ jobs.

Theorem: Assume the number of jobs is $m = 16n \ln n$ jobs. Each of the n processor handles $\mu = 16 \ln n$ jobs on average. With high probability, each processor will have at least half and at most twice the average load.

Proof. Let X_i and Y_{ij} be defined as in the analysis. Applying Chernoff bounds with $\delta = 1$ gives

$$\begin{aligned} \Pr[X_i > 2\mu] &< \left(\frac{e}{4}\right)^{16 \ln n} < \left(\frac{1}{e^2}\right)^{\ln n} = \frac{1}{n^2}. \\ \Pr[X_i < \frac{1}{2}\mu] &< e^{-\frac{1}{2}(\frac{1}{2})^2 (16 \ln n)} = \frac{1}{n^2}. \end{aligned}$$

Union bound implies that each processor has load between half and twice the average with probability at least $1 - 2/n$. □