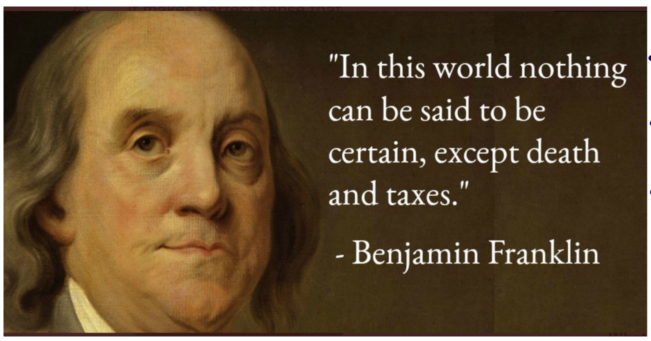
• Algorithms under Uncertainty.

• What makes life interesting?

Decision-making under uncertainty.

Uncertainty is omnipresent!



- Whom to marry?
- Which classes to take?
- Should I buy or rent?
- Which stocks to buy?

Algorithms under uncertainty:

- Decision maker don't have complete knowledge of the input.
- Each time step, decision maker needs to take a decision based only on knowledge of the past.

Several approaches:

- Online Algorithms.
- Online Learning.
- Online convex optimization.

Prerequisite:
Basics of prob. theory & linear programming.

r

• Online learning: Answer a sequence of questions given (partial) knowledge of the correct answers to previous questions and possibly additional available information.

Online Learning

for $t = 1, 2, \ldots$
  receive question $\mathbf{x}_t \in \mathcal{X}$ instance domain
  predict $p_t \in D \supseteq \mathcal{Y}$
  receive true answer $y_t \in \mathcal{Y}$ target domain
  suffer loss $l(p_t, y_t)$

$x_1 \quad p_1 \quad y_1$
$x_2 \quad p_2 \quad y_2$
$\vdots$

Goal: Minimize cumulative loss (regret) suffered along its run.

Example (Predicting whether it is going to rain tomorrow:)

day t, the question x_t can be encoded as a vector of meteorological measurements

the learner should predict if it's going to rain tomorrow output a prediction

in $[0, 1]$, $D \neq \mathcal{Y}$. $\{0,1\}$

$0.7 \quad 0 \quad 0.7$
$0.6 \quad 1 \quad +\,0.4$

loss function: $\ell(p_t, y_t) = |p_t - y_t|$

which can be interpreted as the probability to err if predicting that it's going to rain with probability p_t

• Online Convex Optimization (OCO):

Online Convex Optimization (OCO)

input: A convex set S
for $t = 1, 2, \ldots$
  predict a vector $\mathbf{w}_t \in S$
  receive a convex loss function $f_t : S \to \mathbb{R}$
  suffer loss $f_t(\mathbf{w}_t)$

$$\text{Regret}_T(u) = \sum_{t=1}^{T} f_t(w_t) - \sum_{t=1}^{T} f_t(u)$$

Best hypothesis

Example (Prediction from expert advice)

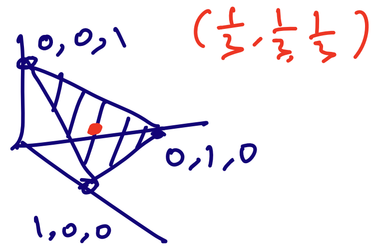
The decision maker has to choose among the advice of n given experts. i.e., the n-dimensional simplex $\mathcal{X} = \{x \in \mathbb{R}^n, \sum_i x_i = 1, x_i \geq 0\}$.

$g_t(i)$: the cost of the i'th expert at iteration t

g_t: the cost vector of all n experts

The cost function is given by the linear function $f_t(w) = g_t^T x$.

There are n stocks.



- This lecture & next few weeks, we focus on Online Algorithms.

- Decisionmaker don't have complete knowledge of the input.
- Input arrives over time : part by part.
- Each time step, decisionmaker needs to take a decision : immediate & irrevocable, based only on knowledge of the past.

- Performance Measure :

Competitive Ratio.

An online algorithm ALG is c-competitive if there is a constant α s.t. for all finite input sequences I,

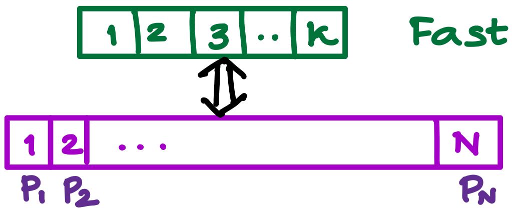
$$ALG(I) \leq c \cdot OPT(I) + \alpha .$$

- For additive constant $\alpha = 0$, we call ALG is strictly c-competitive.

- Mostly we are interested in efficient (polytime) algorithms.
- But hardness also lies in the information theoretic barrier!

- **Paging Problem:**

Consider two-level computer memory system.



Fast memory

$K < N$

slow memory / storage disc
(stores fixed set of pages)
fixed-sized memory units

→ At each time step t a request for some page p_{i_t} comes.

→ A hit occurs if p_{i_t} is already in cache.

→ Otherwise, a miss occurs;
Then the system incurs one page fault and p_{i_t} must be fetched from slow memory to cache.

→ Goal: Decide which k pages to retain in the cache, at each point of time, to minimize misses / maximize hits.

- Typically, for paging
  fast memory = RAM,
  slow memory = Disk.

- In Caching (where page is called Block),
  fast memory = Cache,
  slow memory = RAM.

We use caching/paging interchangeably.

- Multiple abstract cost model: $f \ll s$

General: Cost f for reading fast memory,
Cost s for fetching from slow memory.

- Page fault model: $\boxed{f = 0,\ s = 1.}$ — Our model!
- Full access model: $f = 1,\ s = s.$

Candidate algorithms:

Online

- **LRU** (LEAST-RECENTLY-USED): When eviction is necessary, replace the page whose most recent request was earliest.
- **CLOCK** (CLOCK-REPLACEMENT): An approximation to LRU in which a single "use bit" replaces the implicit (time of last access) timestamp of LRU.[1]
- **FIFO** (FIRST-IN/FIRST-OUT): Replace the page that has been in the fast memory longest.
- **LIFO** (LAST-IN/FIRST-OUT): Replace the page most recently moved to the fast memory.
- **LFU** (LEAST-FREQUENTLY-USED): Replace the page that has been requested the least since entering the fast memory.
- **LFD** (LONGEST-FORWARD-DISTANCE): Replace the page whose next request is latest.

Offline

Assume $k = 3$.

Requests: $P_4, P_2, P_1, P_4, P_3, P_2, P_1, P_4$.

At $t = 3$. $\boxed{P_4 \mid P_2 \mid P_1}$

$t = 4$. $\boxed{P_4 \mid P_2 \mid P_1}$

$t = 5$. Need to evict a page.

- LRU: P_2
- FIFO: P_4
- LIFO: P_1
- LFU: P_2 / P_1.
- LFD: P_4.

Note: All are
Demand paging algorithms, i.e., unless there is a page fault they never evict a page from the cache.

HW: Prove that any online/offline paging algorithm can be modified to be demand paging without increasing the overall cost on any request sequence.

Corollary 5 *There is a paging algorithm such that* Daniely-Mansour '19

- *Its competitive ratio is* $2H_k$ → compares w. best offline algo
- *Its regret on any interval* I *is* → compares w. single fixed cache

$$O\left(k\sqrt{|I|\log(T)} + \sqrt{k|I|\log\left(\binom{n}{k}T\right)}\right) = O\left(k\sqrt{|I|\log(nT)}\right)$$

Let us rephrase Corollary 5 in terms of the paging problem. It guarantees an online algorithm for which the number of cache misses is at most a $2H_k$ larger than what is achieved by the optimal offline schedule. Likewise, for long enough segments, i.e., longer than $\Omega(k^2\log(nT))$, it guarantees that the number of cache misses is not much larger compared to the best single "fixed" cache C^*. Recall that for a fixed cache C^*, whenever we have a request for a page $i \notin C^*$ we first fetch i evicting $j \in C^*$ and then evict i and fetch back j, for a total cost of two.

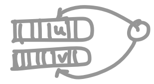
§ Optimal offline paging algorithm: LFD.

Intuition: Any optimal offline paging algo $\mathcal{A}$ can be modified to act like LFD without degrading its performance.

Claim: Let σ be any request sequence.
$\forall i \in [|\sigma|]$, we can construct $\mathcal{A}_i$ s.t. → offline algo
(i) $\mathcal{A}_i$ processes first $i-1$ requests same as $\mathcal{A}$.
(ii) If i'th request results in a page fault, $\mathcal{A}_i$ evicts from its fast memory the page with LFD.
(iii) $\mathcal{A}_i(\sigma) \le \mathcal{A}(\sigma)$.

Claim implies LFD is optimal.

Start with $\mathcal{A} = OPT \xrightarrow[i=1]{\text{claim w/}} OPT_1 \xrightarrow[i=2]{\text{claim w/}} \dots \to OPT_n = LFD$.



Proof of Claim:

Assume just after processing i'th request, fast memories of $\mathcal{A}$ & $\mathcal{A}_i$ contains page sets $\underbrace{X}_{\text{common } k-1 \text{ pages}} \cup \{v\}$, $X \cup \{u\}$ resp.

$\mathcal{A}_i$ has evicted v
$\mathcal{A}$ " " u

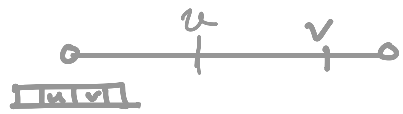
w.l.o.g. assume $v \neq u$ (& i th request results in page-fault)

> Until v is requested, for subsequent requests, $\mathcal{A}_i$ mimics $\mathcal{A}$ except for evicting u if $\mathcal{A}$ evicts v.

Note: # common pages is always $\geq k-1$.
If # common pages becomes k (i.e., if $\mathcal{A}$ evicts v) then $\mathcal{A}_i$ continues same as $\mathcal{A}$.

However, if v is requested before $\mathcal{A}$ evicts v, then $\mathcal{A}_i$ incurs page fault, but not $\mathcal{A}$.

However, when v was evicted by $\mathcal{A}_i$ it must have LFD, so $\exists$ at least one request for u after the i'th page fault.



That incurs a page fault to $\mathcal{A}$ but not to $\mathcal{A}_i$.

$\therefore$ # page faults for $\mathcal{A}_i$ $\leq$ # page faults for $\mathcal{A}$. (after servicing v)

& after servicing v, $\mathcal{A}$ and $\mathcal{A}_i$ identify. $\square$

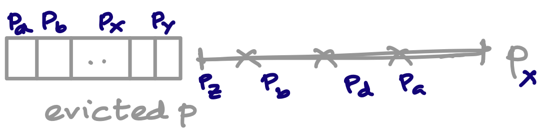
- Polytime solvable!
- Hardness comes from lack of information.

§ Lower Bound of k for deterministic algorithms for paging.

Claim: For any finite sequence σ chosen from a set of $k+1$ pages, $LFD(\sigma) \leq |\sigma|/k$.

$\rightarrow$ LFD faults at most once every k requests.

[If LFD evicts a page p_x, all other pages in cache must be requested prior to next request for p_x]



Pf of lower bound: pages: $p_1, p_2, \dots, p_{k+1}$.

W.l.o.g. assume an online (deterministic) algo $\mathcal{A}_0$ holds $p_1, \dots, p_k$.

We define request sequence σ inductively:

$r_1 = p_{k+1}$,

r_{i+1} = unique page not in $\mathcal{A}$'s cache just after servicing $r_1, \dots, r_i$.

$$(p_1 p_2 p_3) \xrightarrow[p_4]{} p_1 p_4 p_3 \xrightarrow[p_2]{} p_2 p_4 p_3 \xrightarrow[p_1]{}$$

Note: $\mathcal{A}$ faults on each request & we can make $|\sigma| \to \infty$.

$\therefore$ $\mathcal{A}(\sigma) = |\sigma|$, $OPT(\sigma) = LFD(\sigma) \leq |\sigma|/k$.

$\Rightarrow$ $C.R.(\mathcal{A}) \geq \dfrac{|\sigma|}{|\sigma|/k} = k$.

- Paging is not hard due to computational reasons, but due to information theoretic reasons.

• **The (h,k)-paging problem:** $h, k \in \mathbb{Z}$, $h \leq k$.

Measure performance of online paging algo of cachesize $= k$, relative to optimal offline algo with cache $h \leq k$.

weaker adversary!

Belady's anamoly: Interestingly, on some input sequence some algorithms (e.g. FIFO) may perform better when it has smaller fast memory.

(HW: On properties of different algorithm)

§ Marking Algorithms:

• **k-phase partition:** Fast memory size $= k$

Divide σ into phases. Phase 0 = empty sequence.

Phase i is the maximal sequence following phase $i-1$ that contains $\leq k$ distinct page requests.

So phase $i+1$ begins with $(k+1)$th distinct pg rq.

$k=2$ $\quad \overset{P_0}{} \;\vdots\; p_1 \, \overset{P_1}{p_3} \, p_1 \;\vdots\; p_2 \, p_3 \, \overset{P_2}{p_2} \, p_3 \, p_2 \;\vdots\; \overset{P_3}{p_4} \, p_1 \;\vdots\; p_2 \, \overset{P_4}{p_3}$

• Associate a bit "mark" for each page.
  Bit is set → Marked, else → Unmarked.

• Beginning of each k-phase:
  unmark all pages currently in fast memory.

• During a k-phase:
  Mark a page when it is first requested during the phase.

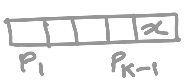
• A marking algorithm never evicts a marked page from its fast memory!

LRU, CLOCK, and many more algorithms belong to the class of marking algorithms.

- We'll show any marking algorithm attains optimal competitive ratio in the page-fault model.

Thm: LRU is marking algorithm.

- For contradiction, assume LRU evicts a marked page x during some k-phase.
- Consider the first request for x during this k-phase. Immediately after serving x: it is marked as most recently used page.
- So to evict x by LRU, there should be $\geq k+1$ different pages to be requested in this phase. [k pages including x + the page which got x evicted]



we need k other pages to be more recently used.

- Contradiction with defn of phase!

§ Theorem: Any marking algo ALG is $k/(k-h+1)$ competitive.

→ Any request sequence σ & its k-phase partition.

Claim: For any phase i, ALG incurs $\leq k$ pg faults

→ There are k distinct pg references in each phase. Once a pg is accessed, it is marked & cant be evicted till end of phase.

So, ALG cant fault twice on same page.

Let q be first request of phase i, consider sequence σ' starting w. second req. of phase i upto and including first req of phase $(i+1)$.

OPT has $h-1$ pages excluding q.

There are k distinct requests in σ'.

So, OPT must incur $\geq k-(h-1) = k-h+1$ faults.

(We can ignore last phase by additive $O(k)$ term)

In each phase alg incurs fault $\leq k$ times.

$$ALG(\sigma) \leq \frac{k}{k-h+1} \cdot OPT(\sigma) + k. \quad \blacksquare$$

upper bound from last phase.

Power of Randomization:

• Adversary Models.

'Natural' model in deterministic case:
- knows the algo & chooses worst-case input to maximize C.R.

• Advantage of randomness : Adversary may not know the outcome of the random choice.

• Adversary constructs a request sequence σ & pays a cost.

① OBL (oblivious) : Weak Adversary
- constructs σ in advance.
- pays optimally.
- Doesn't know actions (due to outcomes of random choices) by Algo.

② ADON (adaptive-online) : Medium Adversary
- Generate $\sigma(t)$ based on Algo's action so far. (on $\sigma(1) \dots \sigma(t-1)$).
- serves current request online.

[So it knows its own strategy for generating σ, Description of online algo & its action so far; & then need to perform in online manner].

③ ADOF (adaptive-offline) : Strong Adversary
- Choose next request based on Algo's action so far.
- Pays optimal offline cost (on σ that is created online by adversary).

- Competitive ratio:

$$\forall \sigma,\ \mathbb{E}[\mathrm{ALG}(\sigma)] \leq c \cdot \underbrace{\mathrm{OBL}(\sigma)}_{\mathrm{OPT}(\sigma)} + \underbrace{\alpha}_{\text{some constant indep of } \sigma} =: r(\mathrm{OBL}(\sigma)).$$

Expectation is taken over the random choices made by ALG.

Note: ADV(σ) itself becomes a RV for ADON/ADOF. So a more careful definition is needed there.

- Adversary: (Q, S).

S: servicing component (answer requests created by Q)

Q: requesting component (creates request)

S for OBL/AD-OF: Optimal offline algo. (so completely determined by Q).

Q for OBL: Fix sequence σ that may depend on the decision-makers online algo.

Q for AD-ON or AD-OF: Q is a sequence of functions $q_i : \underbrace{A_1 \times A_2 \times \cdots \times A_{i-1}}_{\text{Prev. actions}} \to R$.

Let cost of ALG against ADV: ALG(ADV) = ALG(Q) where σ is constructed by Q via interaction w. ALG.

Similarly, adversary cost against ALG = ADV(ALG).

AD-OF: γ-comp against AD-OF if $\forall$ AD-OF Q:

$$\mathbb{E}[ALG(Q)] = \mathbb{E}_x\left[ALG_x[\sigma(ALG_x, Q)]\right]$$

randomness

$$\leq \gamma\, \mathbb{E}_x\left[OPT[\sigma(ALG_x, Q)]\right] = \gamma\, \mathbb{E}\left[OPT[\sigma(ALG, Q)]\right].$$

AD-ON: γ-comp against AD-ON if $\forall$ AD-OF (Q, S)

$$\mathbb{E}[ALG(ADV)] = \mathbb{E}_x[ALG_x(\sigma(ALG_x, Q))]$$

$$\leq \gamma\,(\mathbb{E}_x[ADV(ALG_x)]) = \gamma\,(\mathbb{E}[ADV(ALG)]).$$

§ Relating the adversaries: (Ch 7.3 in B-Y)

Theorem: If there exists a randomized online algorithm that is c-competitive against adaptive offline adversaries, then there also exists a c-competitive deterministic online algorithm.

Theorem: If A is c-competitive against adaptive online adversaries and there exists a d-competitive algo against oblivious adversaries, then there exists a cd-competitive algorithm against adaptive offline adversaries.

Corollary: If A is c-competitive against adaptive online adversaries, then there exists a c^2-competitive deterministic algorithm.

• Randomized Algorithm:

• Algorithm RAND: Whenever a page fault occurs, evict a page chosen randomly and uniformly among all fast memory pages.

→ RAND can be shown to be $\frac{k}{k-h+1}$ - competitive.

§ The optimal algorithm: MARK.

Never evicts a marked pg.

- Initially, all the pages are marked.
- If there is a request for a page p,
  - If p in cache, but unmarked, then mark p.
  - Else if p is not in cache, then p is brought into cache, replacing a randomly and uniformly chosen page from the set of all unmarked pages. Mark p.

    If all pages in cache are marked when p is about to be brought in, then they are all unmarked first.

    Mark p.

k-memory bit

• Thm: C.R. of MARK (against obl. adversaries) is $O(\log k)$.

Proof: W.l.o.g. assume MARK & adversary start with the same set of initial pages in the cache (else we have an additive $O(k)$ term).

Fix a request sequence σ & its k-phase partition.

For each phase i, at start pages in cache are called old pages.

Non-old page requests " new pages.

Obs : (i) Each request is either old/new pg.

(ii) Repeated requests dnt contribute to online cost.

(iii) Beginning of phase, all old pages are unmarked. → k distinct requests give k marked pages. → New pg not in cache, so all are unmarked.

Consider phase i. Has k distinct pg requests.

m_i : # new pages requested in this phase.

Worst possible sequence σ : Request all new pages (m_i page faults), followed by $(k-m_i)$ (first) requests to old pages (that were evicted). → Adv doesnot know which pages are being evicted.

Key! Obs : In i'th phase, j'th requested old page is in cache w.p. $= \underbrace{(k-m_i-(j-1))}_{\text{\#old unmarked pgs in cache}} / \underbrace{(k-(j-1))}_{\text{\#old unmarked pages}}$.

only j-1 requested ones are marked

Hence, $\Pr[\text{fault event}] = 1 - \frac{k-m_i-(j-1)}{k-(j-1)} = \frac{m_i}{k-j+1}$.

$$\mathbb{E}\left[\begin{matrix}\text{faults during}\\ \text{i'th phase}\end{matrix}\right] = \underbrace{m_i}_{\text{from new}} + \sum_{j=1}^{k-m_i} \frac{m_i}{k-j+1} \leftarrow \text{from old}$$

$$= m_i + m_i(H_k - H_{m_i}) \leq m_i H_k .$$

Here, $H_K = \sum_{i=1}^{K} \frac{1}{i}$.

(HW: Show $\ln k \le H_K \le 1 + \ln k$.
Intuition: $\int_1^K \frac{1}{x} dx \approx \ln K$).

Hence, MARK pays $\le \sum_{i=1}^{p} m_i H_K$.

What is OPT(σ): During i th and $(i-1)$st phases, $\geq K + m_i$ distinct pg requests are made.

pg faults during i & $i-1$'th phase $\geq m_i$.

$\therefore$ OPT(σ) $\geq \sum_{i=1}^{p} F_i$ [F_i = # fault in i th phase, P = # phases]

$\geq \frac{1}{2}\left[(F_1+F_2)+(F_2+F_3)+\cdots\right]$

$\geq \frac{1}{2}\sum_{i=1}^{p} m_i$.

C.R. $\le \sum_{i=1}^{p} m_i H_K \Big/ \left(\frac{1}{2}\sum_{i=1}^{p} m_i\right) = 2H_K$. ∎

§ Lower bounds for randomized algorithms:

Yao's lower bound principle:

- Uses L.B. for deterministic algorithms on a distribution to bound L.B. for rand algo.

$C_A(\sigma)$ be the cost of a deterministic online algorithm A with input σ.

Fact: Any randomized μ-comp algo against oblivious adversaries is simply a prob. distr. over the set of deterministic algos $\mathcal{F}$.

$$\mathop{\mathbb{E}}_{A \in \mathcal{F}} \{C_A(\sigma)\} \leq \mu \, C_{OPT}(\sigma), \ \forall \sigma \qquad \cdots (1)$$

Consider some distr. D_I over input σ.

$$\mathop{\mathbb{E}}_{\sigma \in D_I} \left[\mathop{\mathbb{E}}_{A \in \mathcal{F}} \{C_A(\sigma)\} \right] \leq \mu \cdot \mathop{\mathbb{E}}_{\sigma \in D_I} [C_{OPT}(\sigma)]$$

Exchanging expectations (Fubini's thm):

$$\mathop{\mathbb{E}}_{A \in \mathcal{F}} \left[\mathop{\mathbb{E}}_{\sigma \in D_I} \{C_A(\sigma)\} \right] \leq \mu \cdot \mathop{\mathbb{E}}_{\sigma \in D_I} [C_{OPT}(\sigma)] \qquad \cdots (*)$$

Now, $\min_{A_D \in \mathcal{D}} \mathbb{E}_{\sigma \in D_I} \{C_{A_D}(\sigma)\}$, where $\mathcal{D}$ is the set of all det. algo

$$\leq \mathbb{E}_{A \in \mathcal{F}} \left[\mathbb{E}_{\sigma \in D_I} \{C_A(\sigma)\} \right]$$

$$\leq \mu \cdot \mathop{\mathbb{E}}_{\sigma \in D_I} [C_{OPT}(\sigma)]$$

$$\Rightarrow \mu \geq \min_{A_D \in \mathcal{D}} \frac{\mathop{\mathbb{E}}_{\sigma \in D_I} [C_{A_D}(\sigma)]}{\mathop{\mathbb{E}}_{\sigma \in D_I} [C_{OPT}(\sigma)]}.$$

Ratio of expected cost of best deterministic algo & the optimal offline algo.

Note: Choice of D_I is arbitrary. Creative part is finding at "good" D_I.

To show L.B. for paging, we need two facts.

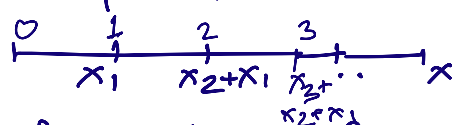
Coupon Collector's Problem.

Suppose each box of cereal contains one of n different coupons. Once you obtain one of every type of coupon, you can send in for a prize. Assuming coupon in each box is chosen indepen-dently and uniformly at random, how many boxes of cereal you need to buy before you obtain at least one of every type of coupon.

- Let X be #boxes bought untill we have all types of coupons.

  Let X_i denote #new boxes bought while you had exactly $(i-1)$ different coupons, then clearly

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



  When exactly $(i-1)$ coupons have been found, the prob. of obtaining a new coupon is:

  $$P_i = 1 - \frac{i-1}{n}.$$

  Hence, X_i is a geom RV with parameter p_i.

  $$\mathbb{E}[X_i] = \frac{1}{p_i} = \frac{n}{n-i+1}.$$

  * with replacement

  $$\therefore \mathbb{E}[X] = \mathbb{E}\left[\sum_{i=1}^{n} X_i\right] = \sum_{i=1}^{n} \mathbb{E}[X_i]$$

  $$= \sum_{i=1}^{n} \frac{n}{n-i+1} = n \sum_{j=1}^{n} \frac{1}{j} = n\,H(n).$$

[Section E in Borodin et al.]

• Renewal Theory :

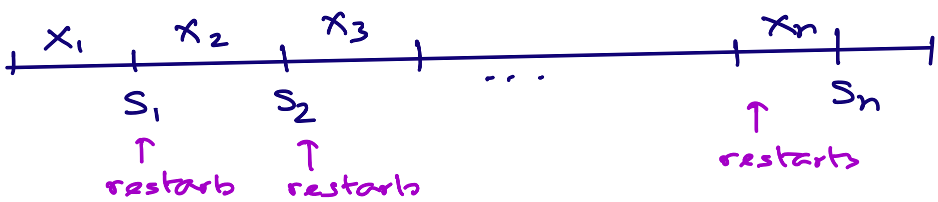
A renewal process represents process that counts # times a certain event restarts itself within some time interval.

$\{X_i : i \in \mathbb{N}\}$ be sequence of +ve iid RV.
Let $\mathbb{P}[X_i = 0] < 1$ and $S_n = \sum_{i=1}^{n} X_i$, $n \in \mathbb{N}$
$S_0 = 0$. $N(t) = \sup\{n : S_n \leq t\}$.

Formally, renewal process is a stochastic process $\{N(t) : t \geq 0\}$.

Remember, stochastic process $\{X(t) : t \in T\}$ is a family of RV where T is an index set (typically time in online algo?) & $X(t)$ is the state of the process at time t.

Renewal occurs at time t, if $S_n = t$ for some n.



After each renewal, the process begins again.

S_n : time of n'th renewal.

X_n : time between $(n-1)$ & n'th renewal. $= S_n - S_{n-1}$

$N(t)$: Number of renewals in the interval $[0, t]$.

Renewal function $M(t) = \mathbb{E}[N(t)]$.

Elementary renewal theorem:

Given a sequence of iid RVs X_i, $i \geq 1$, with $0 < \mathbb{E}[X_1] \leq \infty$, then

$$\lim_{t \to \infty} \frac{M(t)}{t} = \frac{1}{\mathbb{E}[X_1]}.$$

• L.B. for randomized algo for paging is $\ln K$.

→ Let universe size be $K+1$.

Distribution D: Each input at time i, σ_i is uniformly distributed over $K+1$ elements.

Pr of fault on any online algo $\mathcal{A}$ on $\sigma_\ell = 1/_{K+1}$ for $\ell \in [j]$.

For an input $\sigma = \{\sigma_i\}_{i=1}^{j}$;

$$\min_{\mathcal{A}} \mathbb{E}_D \{F_{\mathcal{A}}(\sigma)\} = \sum_{i=1}^{j} 1/_{K+1} = \frac{j}{K+1}. \quad \cdots \circledast$$

#faults of $\mathcal{A}$ on σ

Now we compute U.B. on $\mathbb{E}_D\{F_{OPT}(\sigma)\}$.

We partition σ into phases P_i, where P_i consists of requests made at time $[\theta_i, \theta_i+1, \ldots, \theta_{i+1}-1]$, where $\theta_1 = 1$, and

$$\theta_{i+1} = \min\{r : \{\sigma_{t_i}, \sigma_{t_i+1}, \ldots, \sigma_r\} = [K+1]\}.$$

Thus P_i contains exactly K distinct file requests are made. $|P_i| = \theta_{i+1} - \theta_i$.

OPT = LFD, hence if one fault is incurred in a phase then no more faults can occur.

∴ # faults incurred by OPT until time j
$\leq$ # completed phases by time j. $+1$

Hence, $\mathbb{E}[F_{OPT}(\sigma)] \leq 1 + \mathbb{E}[\max\{\ell : \theta_\ell \leq j\}]$. (1)

Now D is uniform & iid, $|P_i| = \theta_{i+1} - \theta_i$ is also iid.

So apply elementary renewal theorem, with renewal intervals. $X_i = P_i = \theta_{i+1} - \theta_i$,

$$\lim_{j\to\infty} \left(1 + \mathbb{E}\left[\max\{\ell : \theta_\ell \le j\}\right]\right)/j = 1/\mathbb{E}[|P_1|]. \quad ②$$

$\approx$ when $|\sigma| \to \infty$, the number of completed phases normalized with $|\sigma|$ converges to the reciprocal of the expected length of a phase.

$$\lim_{j\to\infty} \frac{F_{OPT}(\sigma)}{j} \le 1/\mathbb{E}[|P_1|]. \quad \dots ③$$

From defn of phase, $|P_1|+1$ is identically distr. as the finish time T_c in the coupon collector problem.

Hence,
$$\begin{aligned}\mathbb{E}(|P_1|) &= (K+1)H_{K+1} - 1 \\ &= (K+1)H_K + 1 - 1 \\ &= (K+1)H_K. \quad \dots(4)\end{aligned}$$

Therefore, $\mathbb{E}[F_{OPT}(\sigma)] \le j/\mathbb{E}[P_1] \le \dfrac{j}{(K+1)H_K}$ $\cdot(**)$

Also, $\min_{A} \mathbb{E}_D\{F_A(\sigma)\} = \dfrac{j}{K+1}$.

$$\therefore \mu \ge \min_{A_D \in \mathcal{D}} \frac{\mathbb{E}_{\sigma\in D_I}[C_{A_D}(\sigma)]}{\mathbb{E}_{\sigma\in D_I}[C_{OPT}(\sigma)]}$$

$$\underset{(\text{from } * \,\&\, **)}{\ge} \frac{j/K+1}{j/(K+1)H_K} \ge H_K \ge \ln K. \blacksquare$$