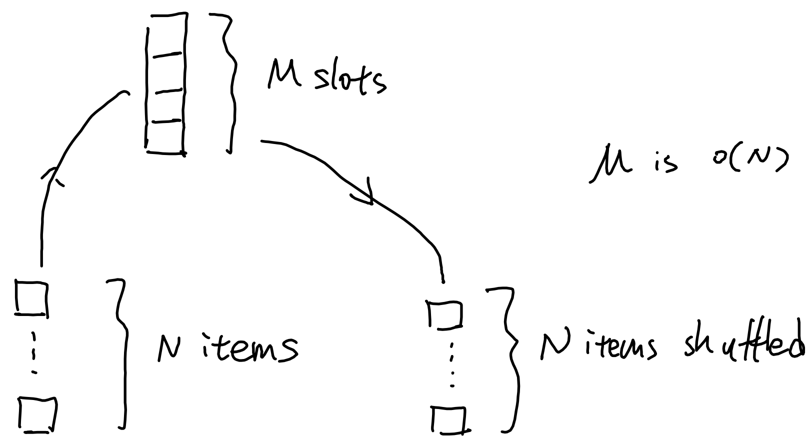
Stream Shuffle Problem

How much entropy can we get on a data stream of length N, with M slots of random access memory ?



M is $o(N)$

A lossy upper bound is $\underbrace{\sum_{j=1}^{N} \ln j}_{\uparrow \text{ uniform dist. for } N! \text{ items}} \sim O(N \ln N)$

What kind of entropy are we looking at ?

We have a random process $X_t(\omega) : \{0,1\}^{[1\ldots t]}$ and $\sum_{i=1}^{t} (X_t(\omega))_i \leq M$

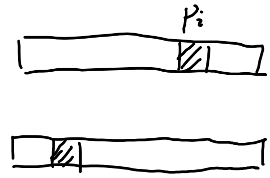
$X_{t+1}(\omega) \curvearrowleft X_t(\omega)$

This works like removing one element and add a new element.

Formally, we have this statement : $\underbrace{d_H}_{\text{Hamming Distance}}(X_{t+1}(\omega), X_t(\omega)) = 2 \ \& \ (X_{t+1}(\omega))_{t+1} = 1$

First, we prove that if $X_t(\omega)$ is initally filled (which is $\sum_{i=1}^{t}(X_t(\omega))_i = M$) then the entropy of this process is the same as the shuffled stream, i.e. the selected item.

$X_0, X_1, \ldots, X_{t-1}$



The probability mass is $p_{X_{t+1}|X_t}(x_{t+1}|x_t)$

which is the probability to remove an element.

(invalid moves are just zero)

How to enumerate the elements in this space?

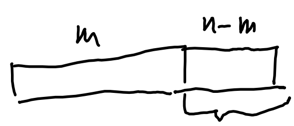
Suppose $\Omega_{m,n}$ represents the space with m memory slots and n stream items.

Then the size of probability space $|\Omega_{m,n}|$ will be

- case 1: $m \geq n$

  $$|\Omega_{m,n}| = n!$$

- Case 2: $m < n$



$$|\Omega_{m,n}| = m! \, \underline{m^{(n-m)}}$$

for $[n-m+1 \ldots n]$, they have m choices

we randomly remove items from memory slots, so there are $m!$ items to be removed

Question: Is the entropy of the shuffle the entropy of the distribution on $\Omega_{m,n}$? (Not exactly)

$$H(p_{X,Y}) = -\sum_{x,y} p(x,y) \log p(x,y)$$

$$= -\sum_{x,y} p(y|x) \cdot p(x) \cdot (\log p(y|x) + \log p(x))$$

$$= -\sum_{x,y} p(y|x) \cdot p(x) \cdot \log p(y|x) - \sum_{x,y} p(y|x) \cdot p(x) \cdot \log p(x)$$

$$= -\sum_{x} p(x) \cdot H(p_{Y|X}(\cdot|x)) - \sum_{x} p(x) \log(p(x))$$

It seems like we will use this twice.

First, let X be the choices (evictions) made in the first $(n-m)$ outputs.

Let Y be the evictions made in the last m outputs.

According to the formula, not matter what X is instantiated after the $(n-m)$ steps,

We just maximize $P_{Y|X}(\cdot|X)$, and it is simple: just uniformly pick items from all non-empty slots.

For now, let's just worry about the replacement part (that $(n-m)$ thing)

We notice that the picking of slots can be independent, and when it is uniform it maximizes the entropy.

Also, there is a one-one correspodence between

Question: If we are not doing shuffle, but instead we want to perform sampling

Grad Algorithm Lecture 2

Problem Set : piazza.com ↙ whitelist that as (non-spam)

↑ ↑

Discuss Solutions ☑ Syllabus

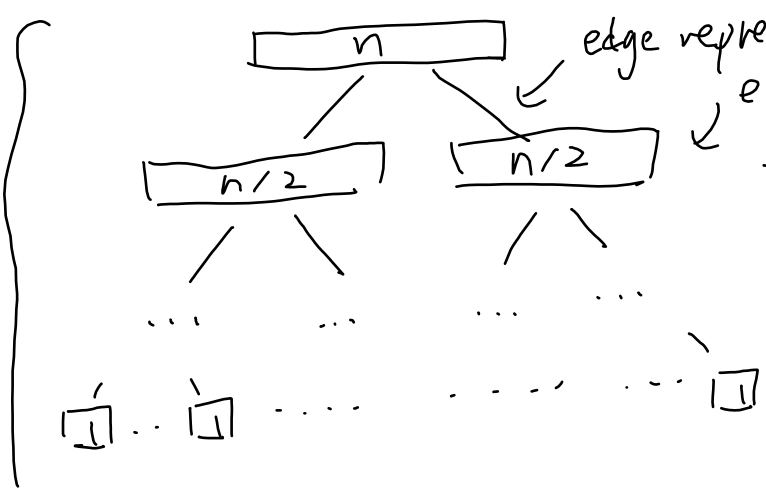
↑

Home Page

Review : Merge Sort

$$T(n) = \begin{cases} 2 \cdot T(\frac{n}{2}) + O(n) & n > 1 \\ O(1) & n = 1 \end{cases}$$

Recursion Tree :



edge represents #s. of calls

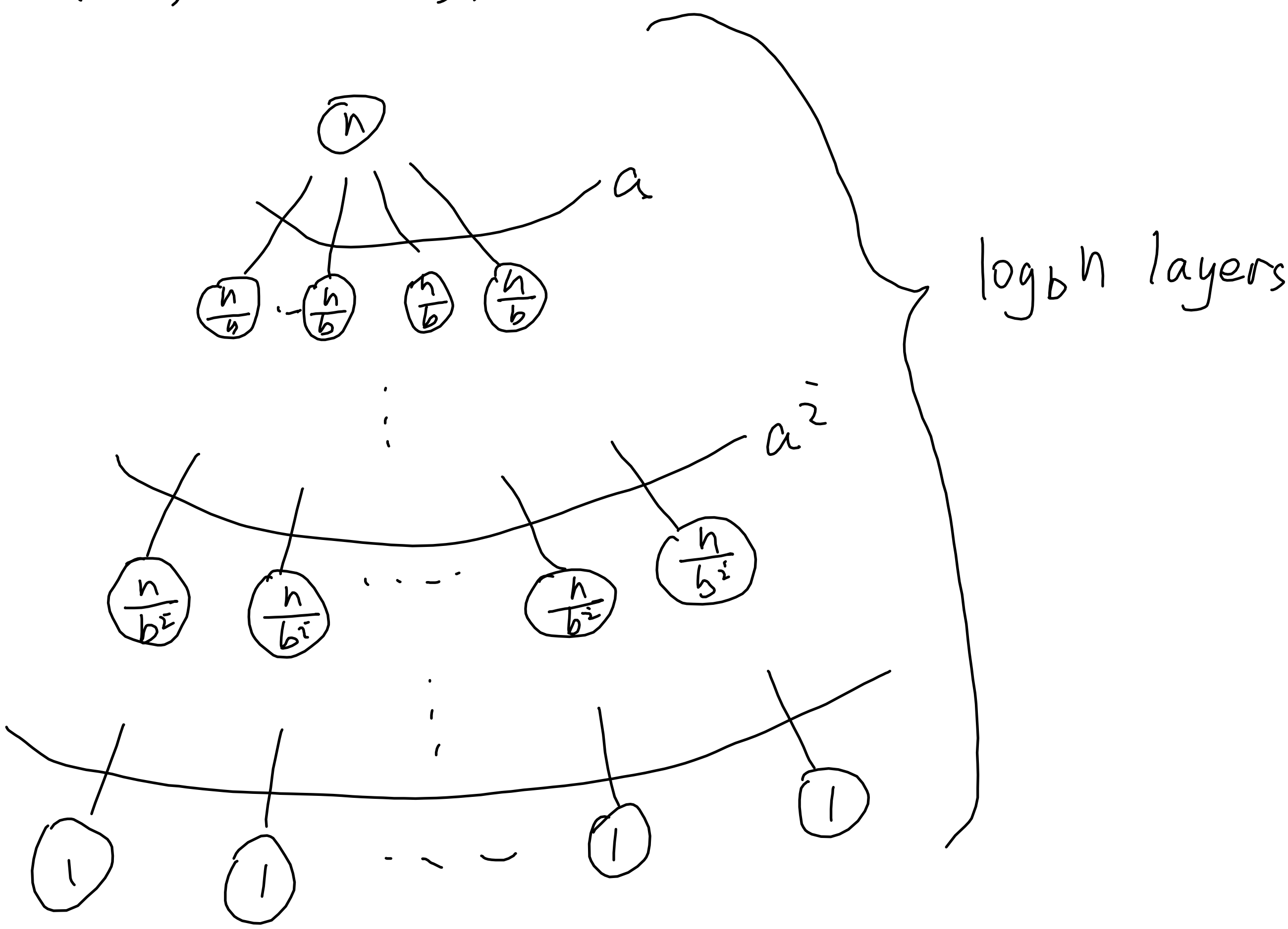
each node is a call to the function

lg n height

$$\underbrace{O(n) + 2\,O(n/2) + 4\,O(n/4) + \cdots + n\,O(1)}_{\lg n \text{ terms}}$$

Generalize a little!

$$T(n) = a \cdot T\left(\frac{n}{b}\right) + O(n^c)$$



$$T(n) = \sum_{i=0}^{\lfloor \log_b n \rfloor} a^i \cdot O\left(\frac{n^c}{b^{ic}}\right)$$

$$= \sum_{i=0}^{\lfloor \log_b n \rfloor} \frac{a^i}{b^{ic}} \cdot O(n^c)$$

not if $a = b^c$

$$= \frac{1 - \left(\frac{a}{b^c}\right)^{\lfloor \log_b n \rfloor}}{1 - \frac{a}{b^c}} \cdot O(n^c) = \begin{cases} O(n^{\log_b a}) & a > b^c \\ \log n \, O(n^c) & a = b^c \\ O(n^c) & a < b^c \end{cases}$$

Another D&C

$A = [3 \ 1 \ 2 \ 8 \ 6 \ 4 \ 5]$

Find maximum $A[i] - A[j]$ where $i < j$

Polynomial Multiplication

$$f(x) = \sum_{i=0}^{d} x^i f_i$$

$$f(x) \cdot g(x) = \sum_{i=0}^{d_f+d_g} x^i \left(\sum_{j=0}^{i} f_j g_{i-j} \right)$$

Naive implementation gives $O(n^2)$ ← where n = degree

Polynomial Multiplication is convolution

Applications & Examples :

3-SUM :

Input : 3 sets $A, B, C \subseteq \{1, 2, \dots n\}$

Is there $a \in A$, $b \in B$ s.t. $a+b \in C$

$$f_A(x) = \sum_{a \in A} x^a \qquad f_B(x) = \sum_{b \in B} x^b$$

$$f_C(x) = f_A(x) \cdot f_B(x)$$

Conjecture : For $|A| \leq n$ w/o upper bound of values

$|B| \leq n$

$|C| \leq n$

Algo returns true if there is $c \in C$ with

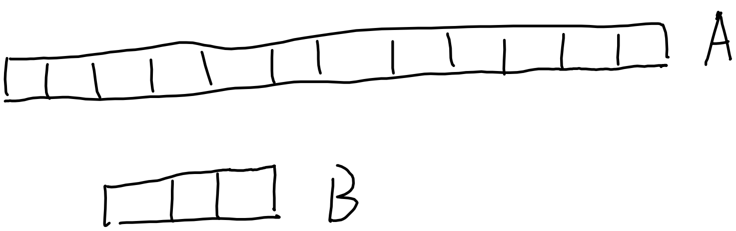
$$\sum_{k=0}^{c} f_{c-k}\, g_k \neq 0 \iff \exists k : k \in B \wedge c-k \in A \wedge c \in C$$

Image detection & Pattern Matching

Task: Picture A ← larger

Picture B ← smaller

Find B in A (simplified, 1-d)



$$f_A(x) = \sum_{i=0}^{n} x^i A[i]$$

$$f_B(x) = \sum_{i=0}^{m} x^i B[i]$$

FFT Algorithm

Representation of Polynomials p

1) coefficient

2) point representation (degree d), we have $d+1$ points $\{x_1, \dots x_i, \dots, x_{d+1}\}$, and $\{y_i : y_i = p(x_i)\}$ evaluated at x_i

If f and g are two polynomials in point representation, (evaluated on the same set of points)

compute $h(x) = f(x) \cdot g(x)$ will be just $h(x_i) = f(x_i) \cdot g(x_i)$

Remember: that h has degree $2d$

Evaluate one of the points on $f(x)$ always takes $O(n)$.

But we can share some computation between different points.

Ford-Fulkerson Method

Idea 1: start with some flow f

Gradually improve f by finding an augmenting path in a modified graph G^f ← residual graph

Idea 2: we can increase f_e up to capacity c_e

decrease $f_e > 0$

Residual Graph G^f:

1) same vertices V
2) Edges $e \in E$ have capacity $c_e - f_e$
3) Edges $-e \in E$ for $e \in E$ with capacity f_e

Alg:

1) Start with $f = (0, \dots, 0)$
2) Create G^f
3) Find s-t path p in G^f, break if no such path exists
4) $f \leftarrow f + p$, go back to step 2

Time:

$O((m+n) \cdot \text{"max-flow"})$

Thm : This Alg returns maximum flow.

Def : min s-t cut of a graph

$G = (V, E)$ edge capacity c_e, source s, sink t, $s \neq t$.

find $X \ni s \wedge X \not\ni t$, s.t. $\sum_{e \in X \times V \setminus X} c_e$

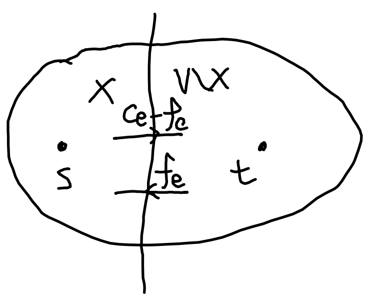
Obs : max s-t flow $\leq$ min s-t cut

Thm : max s-t flow = min s-t cut

Pf : If G^f has no s-t path then $\exists$ a s-t cut X

s.t. $\sum_{e \in (X, V \setminus X)} c_e = \sum_{(s,v) \in E} f_{sv}$

Let X denote all vertices reachable from s in G^f

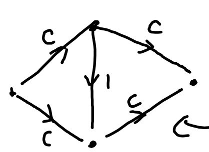


① $\forall e \in (X, V \setminus X)$ $f_e = c_e$ the edge weight $c_e - f_e$

② $\forall e \in (V \setminus X, X)$ $f_e = 0$

Alg : a polynomial algorithm finds s-t flow

in time $O((m+n) m \cdot \log C)$ C = maxnum capacity of edges



send maximum possible flow doesn't work if the path is bad ……

Modification : To find the augmenting path, don't try an arbitirary path. Find an augment path with capacity over M.

Run the algorithm for $M, M/2, M/4, \cdots, 1$

After iteration for M, the ditterence between current flow and optimal flow is upper-bounded by M.

Linear Programming

Continuous variables $x_1, \ldots, x_n$

max / min $\sum_{i=1}^{m} c_i x_i$

s.t. $\sum_{i=1}^{m} a_i^{(j)} x_i \le b^{(j)}$ for $j \in \{1, \ldots, m\}$

Theorem : LP can be solved in time the is polynomial of bits of inputs.

Test Optimality of x : show a linear combination of constraints such that its coefficient is $(c_1, \cdots, c_m)$

Homework 1

1.1 $n^c = O(d^n)$

case 1

$d > 1$ then $\forall c > 0 : \exists a : n \to \infty : n^c \le a d^n$

case 2

$d = 1$ then we need $c = 0$ for n^c to be constant

case 3

$d < 1$ then $n \to \infty : d^n \to 0$: no $c > 0$ feasible

1.2 $\exists a > 1 : n \to \infty : (\log(n))^c \le a^c n^d$

$c > 0$, so $(\cdot)^c$ is mono-incr, $\log(n) \le a n^{d/c}$

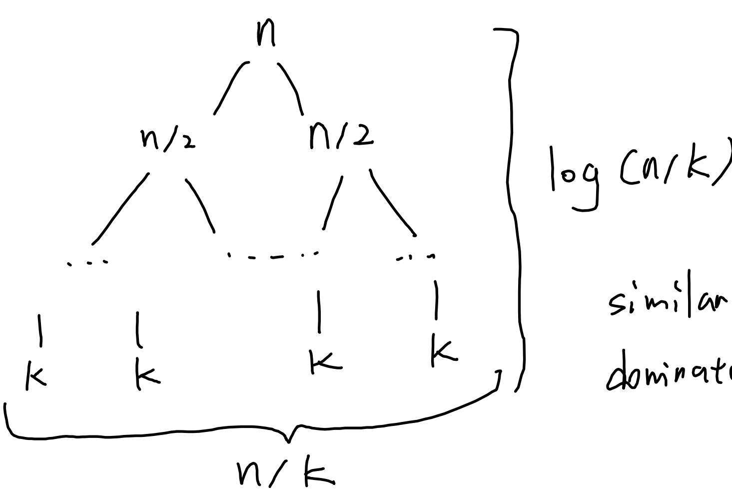
This requries $d/c > 0$, which always holds.

2.1 $T(n) = 4 \cdot T(n/2) + n^2$

Apply master theorem, $4 = 2^2$, so $T(n) = n^2 \log n$

2.2 $T(n, k) = 2T(n/2, k) + k$ $\quad T(k, k) = O(k \log k)$

We should suppose $n \ge k$ and $T(n, k)$ increases when $n \nearrow k \uparrow$



$T(n, k) = O((n/k) \cdot k \log k)$

$= O(n \cdot \log k)$

similar to maste theorem, the dominate term is $O((n/k)^{\log_2 2})$

2.3 $T(n) = T(n/2) + T(n/3) + n$

$$T(n) + an = T(n/2) + a(n/2) + T(n/3) + a(n/3)$$

so we have $a = -6$

$$F(n) = T(n) - 6n$$

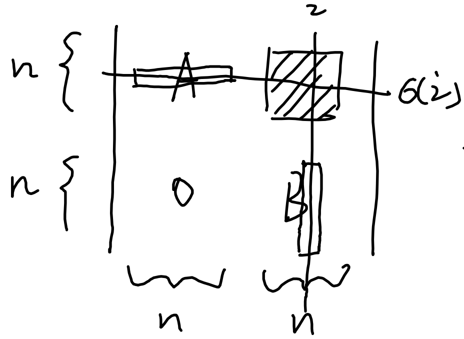
$$F(n) = F(n/2) + F(n/3)$$

1. Prove Cauchy-Binet Formula

Attempt 1:

$\det(A \times B) = (\det A)(\det B)$ ← when A, B are both $n \times n$

Which can be written as:



$= \prod_{\sigma} \operatorname{sgn}(\sigma) \cdots a_{i\sigma(i)} \cdots \cdots b_{i-n\,\sigma(i)-n} \cdots$

↓

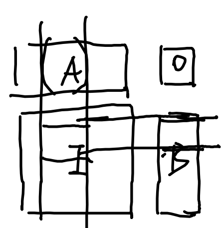
For a term to be non-zero, all elements must be selected from A, B, ▨. Selecting a term $(i, \sigma(i))$ in ▨ means $\forall j: a_{j\sigma(i)}$ cannot be selected, but each col $j \in [n]$ needs to select a $\sigma(j)$, so $\sigma(j) \in [n+1..2n]$, which selects a zero term.

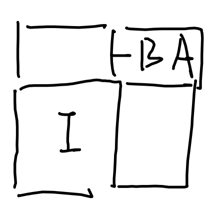
So $\det\begin{pmatrix} A & ▨ \\ 0 & B \end{pmatrix} = \det\begin{pmatrix} A & 0 \\ 0 & B \end{pmatrix} = \det(A)\det(B)$

$$\det\begin{pmatrix} A & I \\ 0 & B \end{pmatrix} = \det\begin{pmatrix} A & I \\ -BA & 0 \end{pmatrix} = \det\begin{pmatrix} -BA & 0 \\ A & I \end{pmatrix}$$

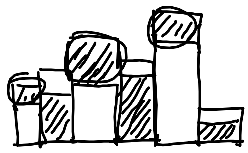
$$= \det(BA)$$

If it works for Cauchy-Binet formula, that will be wonderful!





$$d_{TV}(\mu, \nu) = \sup_A |\mu(A) - \nu(A)|$$



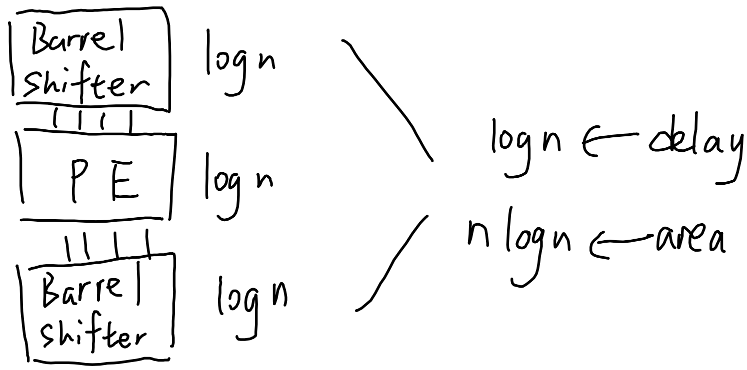
compute the max difference for each side

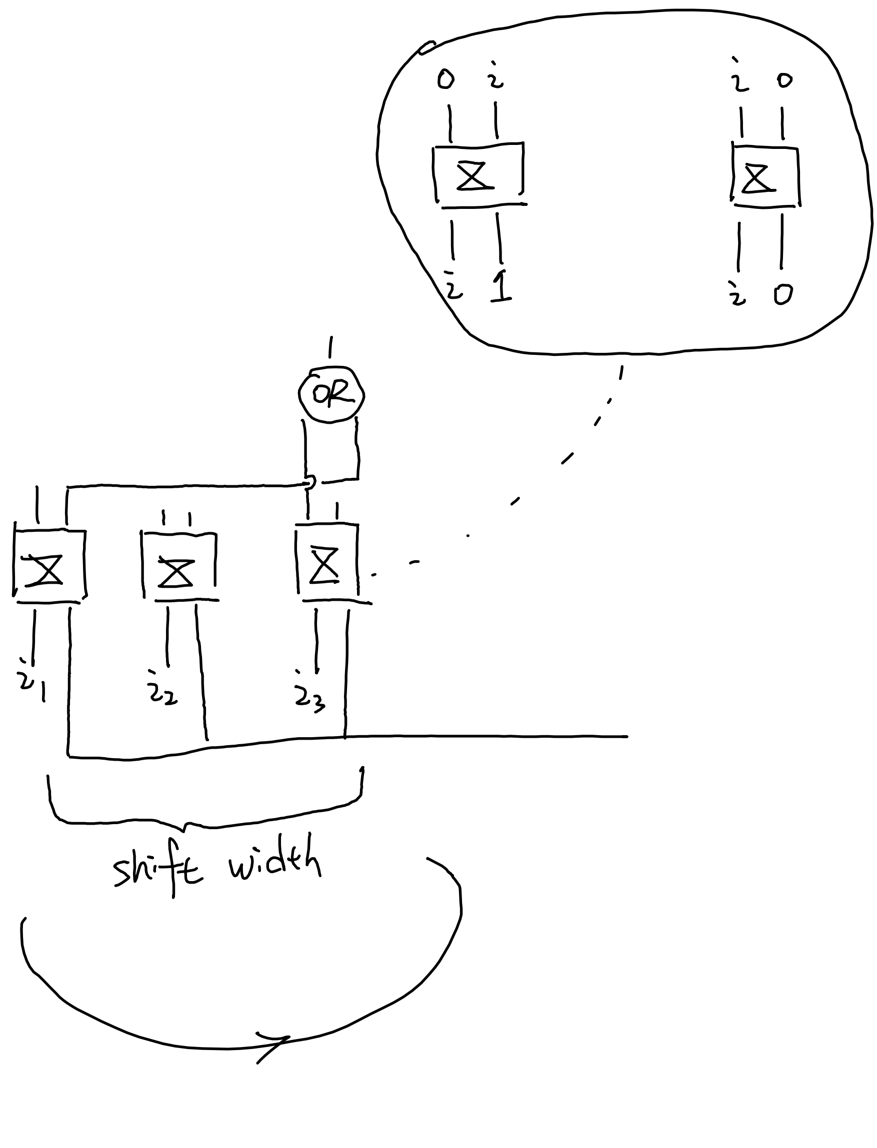
$$T_x(\varepsilon) = \min\{t : d_{TV}(P^t(x, \cdot), \pi) \le \varepsilon\}$$

$$\forall x : T_x(\varepsilon) = \min\{t : d_{TV}(P^t(x, \cdot), \pi) \le \varepsilon\}$$

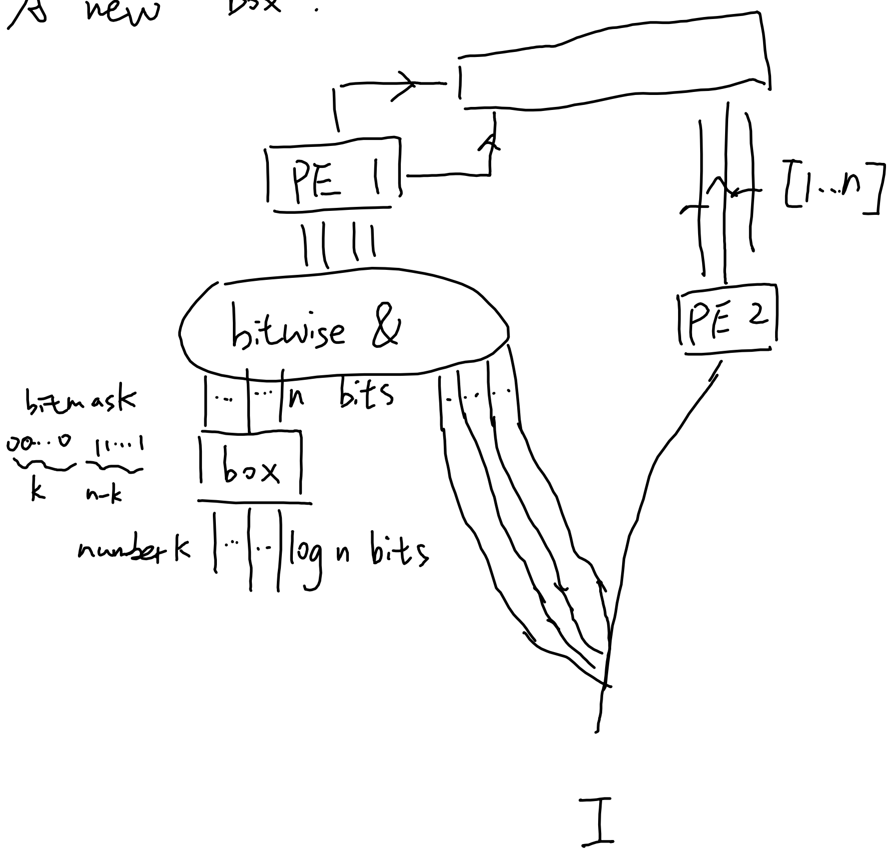
$$\le T_x(1/4) \log_2(1/\varepsilon)$$

Programmable Priority Encoder





A new box :



If a[k..n] has a one, then take a[k..n]

otherwise take a[0..n].

Register 1 cycle

SRAM 10ns

DRAM 40~60ns page mode 4kB

interleaved DRAM

Pipeline Design :

Other techniques

combine DRAM w. SRAM

wide word parallelism

Exact-match Lookups

(Fixed length)

two reasons

1. 25.6 μs Ethernet { buffering / stations cannot be far
2. limited #(memory accesses)
3. potential for hardware speedup
4. crucial for <u>bridging</u>

CSMA/CD ← Only one endnode transmits.

↓

RJ-45 P2P

Consider this scenerio :

these two endnodes hear the transmission end signal simultanausly

Exponential Backoff

Ethernet Bridge

Time is sliced into frame in Ethernet

$(1 + o(1))\frac{\log n}{\log\log n}$ ← max collision in hashmap

Perfect hashing

of elements in the bucket

$\mathrm{Binomial}(n, \frac{1}{n})$

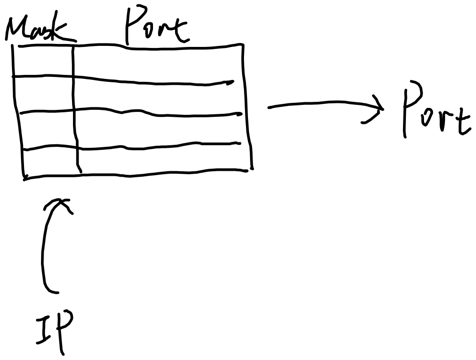
when $n \to \infty$

Poisson Distribution with avg 1.

d-left (Another kind of hash table)

$\log\log n / \log 2 + o(1)$

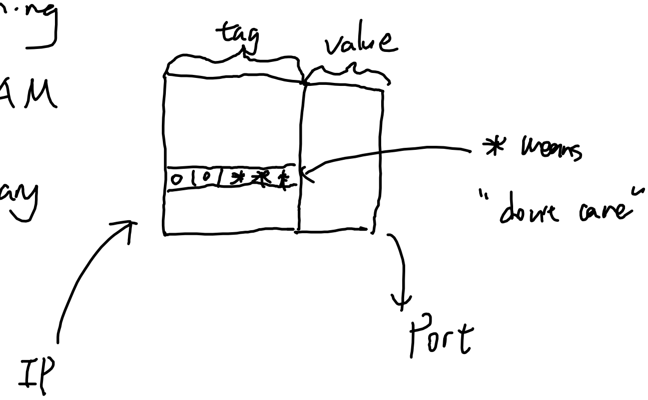
Longest Prefix Match



"Engineer Solutions"

1. Caching
2. TCAM

Ternary



trie-based algorithm

1. binary-encoded edges (unibit trie)
2. when it matches, walks down the edge

optimization:

1. mult-bit edges (typically 8 bits)
2. duplicate values over matching entries

   if multiple prefixes end at the same level, then the port for longer prefix is stored.

fixed stride optimization: determine optimal step length with k strides

$$dp[l, k] = \min_{1 \le j \le 32-l} \{ 2^j \cdot |forest(l)| + dp[l+j, k-1] \}$$

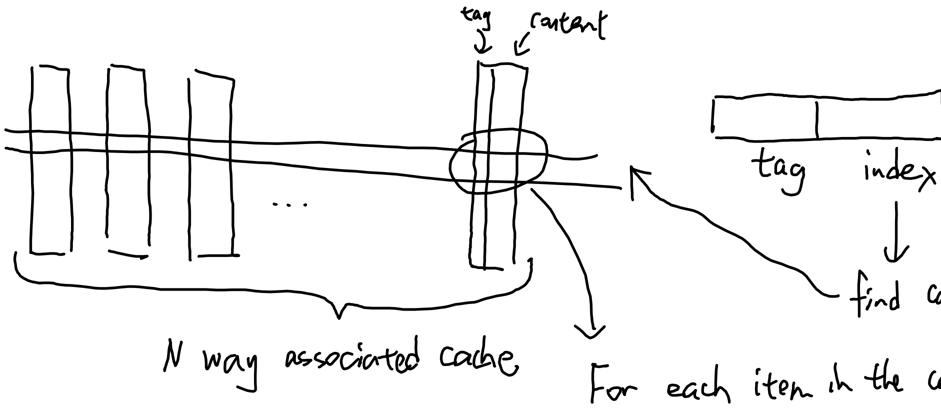
$$dp[l, 1] = 2^{32-l} \cdot \underbrace{|forest(l)|}$$

the number of prefixes

variable stride optimization: determine optimal step length

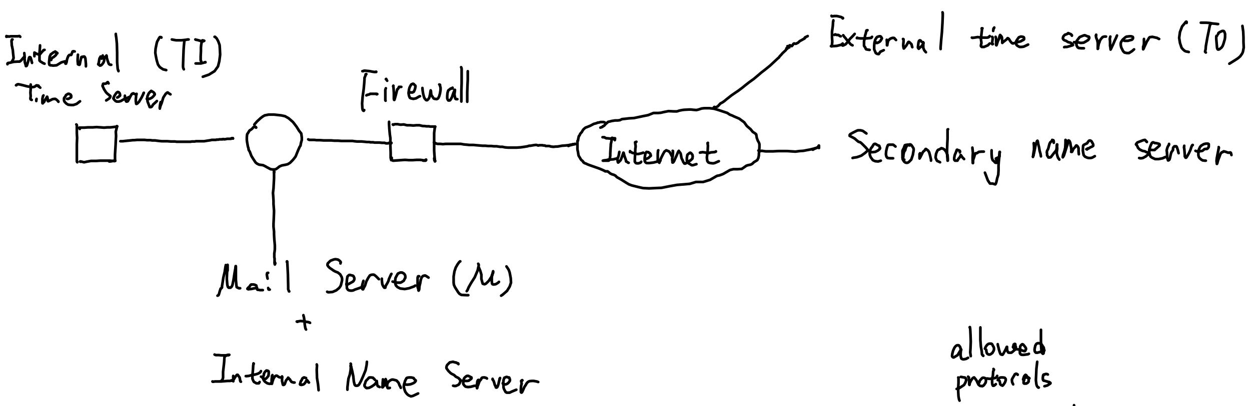
$$dp[r, k] = \min_{1 \le j \le 32 - level(r)} \{ 2^j + \sum_{t \in forest(r,j)} dp[t, k-1] \}$$

Set Associated Cache



For each item in the cache line, compare the tag with the tag on each item.

Firewall in 80s



D : Destination

S : Source

P : Port

A : Action

allowed protocols

DIP	SIP	DP	SP	Flags	ACT
M	*	25	*	*	allow incoming mail
				⋮	

EGT : extended grid of tries

Decision Tree for Firewall