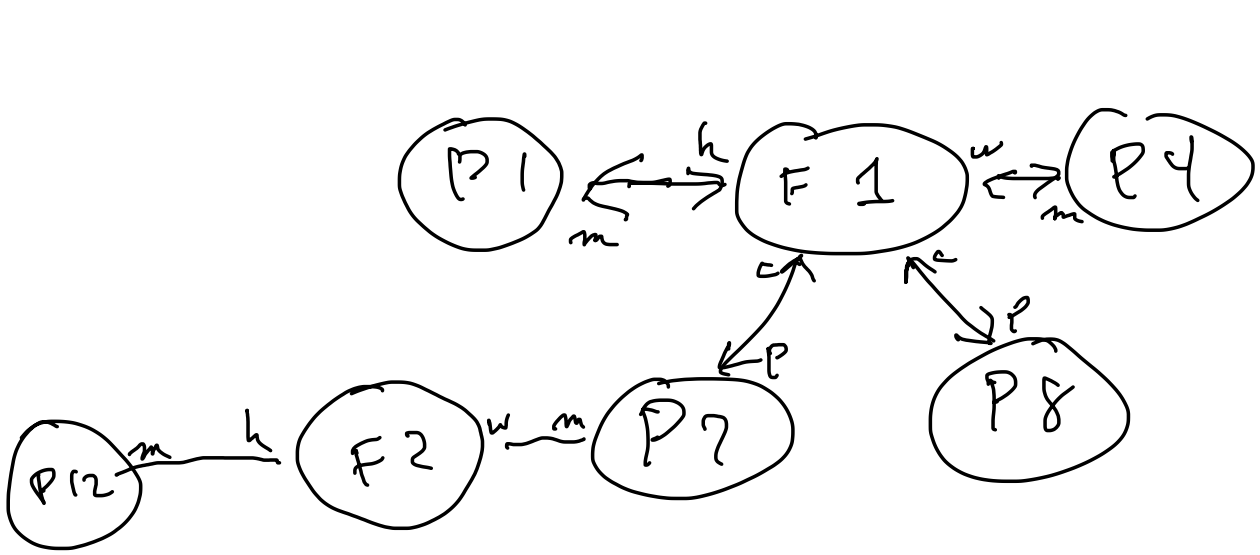
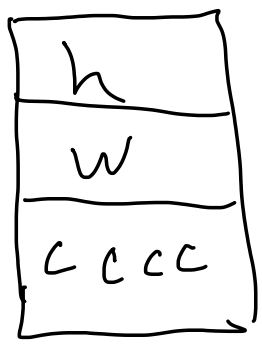
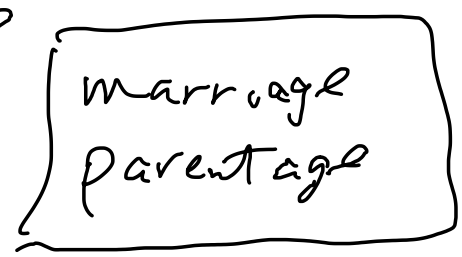
Data Structures #5



F:



P



Bipartite graph:

families only point to persons.
persons only point to families.
edges always connect a person + a family.

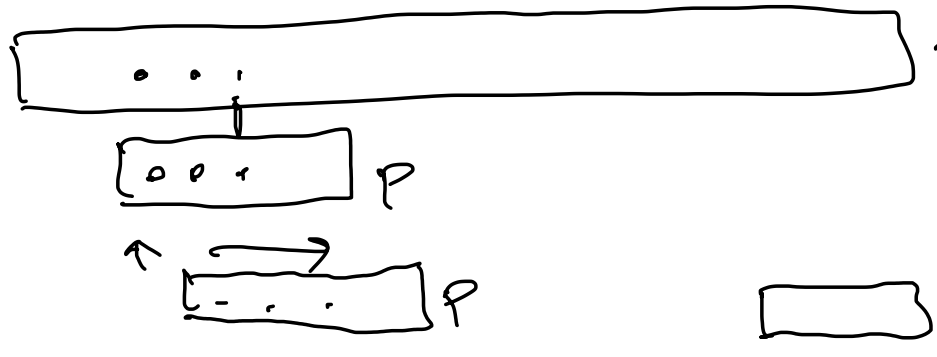
Pattern matching. — Text search problem.

find a match for a pattern (string) p in a text t

t = hello world result = 7
p = orl
return value: -1 if not found.
index of start of pattern if found.

$|p| = m$ $|t| = n$

① Brute force.



Example:

t 0 1 0 0 0 1

p 0 0 1
   0 0 1
   0 0 1
   0 0 1

Complexity: (comparisons)

best case: $n-m$ ↑ $\Theta(n)$

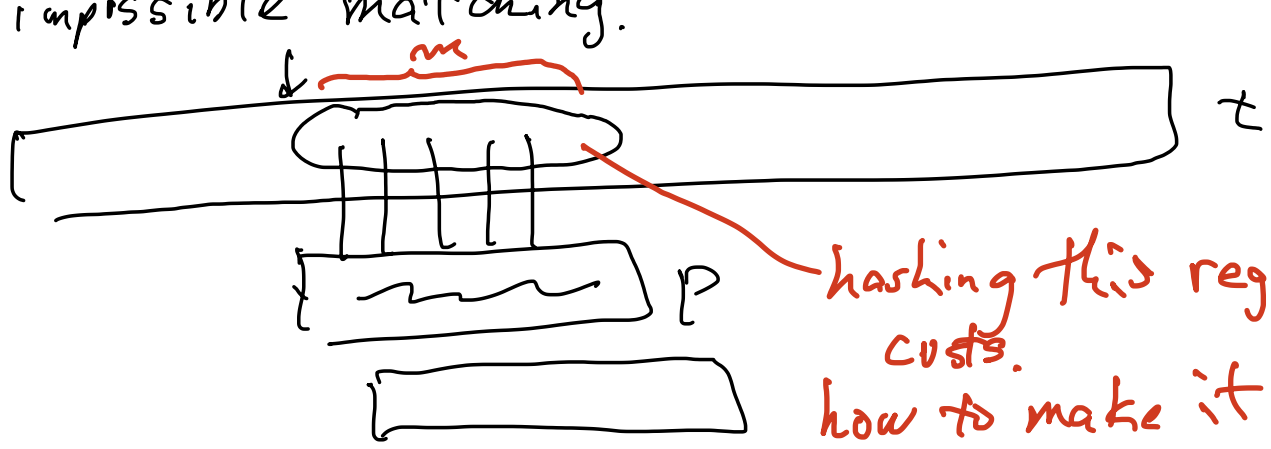
worst case: $(n-m)m$ ↑ $\Theta(n \cdot m)$

In practice: 1.1 n comparisons.

---

② Rabin-Karp 1987

Idea: do a quick hash-based check every time you slide the pattern over to avoid impossible matching.



Stage 1 of the development.

"fingerprinting"

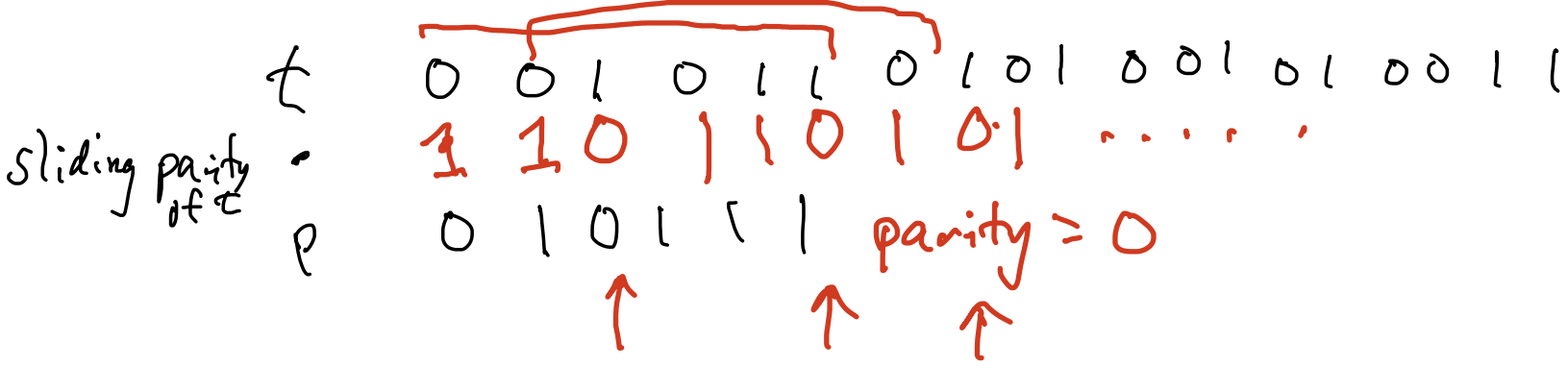
based on parity: (# of 1 bits in a string) mod 2.

assume t, p are just 0,1 strings.

$$\text{parity of } p = \left(\sum_{0 \le j < m} p[j] \right) \bmod 2.$$

↑ implement by & 01

window ↙



sliding parity of t = tParity$_j$

tParity$_0$ = direct computation

$$t\text{Parity}_{j+1} = \left(t\text{Parity}_j + t[j] + t[j+m]\right) \bmod 2$$

Stage 2: use XOR (^) instead of parity.

Now we expect full comparison of characters $1/128$ th of the positions of the window.

Cost: $O(n)$ parity updates, each constant time.

Stage 3: Reduce work not to $\frac{1}{2}$ (parity) or $\frac{1}{128}$ (XOR) but $1/q$ for a very high q.

Hash function for window starting at j

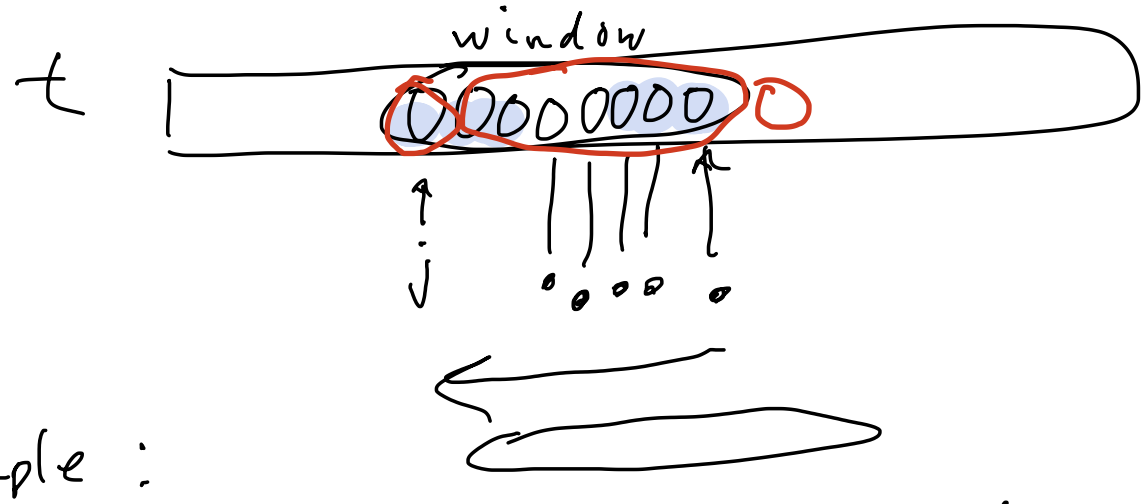
$$\left(\sum_{0 \le i < m} 2^{m-1-i}\, t[j+i]\right) \bmod q.$$

Experience: q should be prime $> m$.

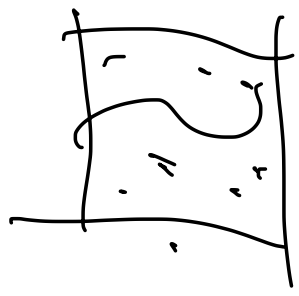
Update $t\text{Parity}_{j+1} = t[j+m] + 2\left(t\text{Parity}_j - 2^{m-1}\, t[j]\right) \bmod q.$ ← still expensive

Use shifting to multiply by powers of 2.

Chances of getting a mismatch if hash agrees is very low.
⇒ skip the inner loop, just assume success.



Example:
$m = 10$
$n = 1000$ choose q: prime, close to mn^2
$q = 999\,991$
probability of "mistake" = "false positive"
= "hash collision" = $\frac{1}{q}$
Monte-carlo: omit the point-by-point verification
not worth the benefit.
Las Vegas version: check anyway.



Experience: $O(n)$ comparisons.
work: $7n$
not worth it.

(3) Knuth-Morris-Pratt 1970~1977.

t: Tweedledee and Tweedledum

p: Tweedledum

p	Tweedledum
1	1 2 3 4 5 6 7 8 9 10

slide window a large amount.

t: p a p p a p p a p p a r

p: p a p p a r

p	p a p p a r
1	1 2 3 3 6

p a p p a r

p a p p a r

Precompute a "shift table" based on p.

t: p a p p a p p a p p a s s s s s s

p p a p p a

p a p p a r

p a p p a r

p a p p a r

p a p p a r

How to build the shift table:

omitted. $\Theta(m)$

---

(4) Boyer-Moore 1977

step 1: always search from end of pattern

t

p

Step 2: occurrence heuristic

↑ method that is likely to improve performance.

At a mismatch: say see letter α in t,

shift p so that the rightmost occurrence of α in p lines up with that α in t.

(dont slide left)

(if α is absent from p, slide p to one place beyond α)

t = conundrum
p = rum
rum
rum

t = conundrum
p = drum
drum
drum

---

t = conundrum
p = natu
natu

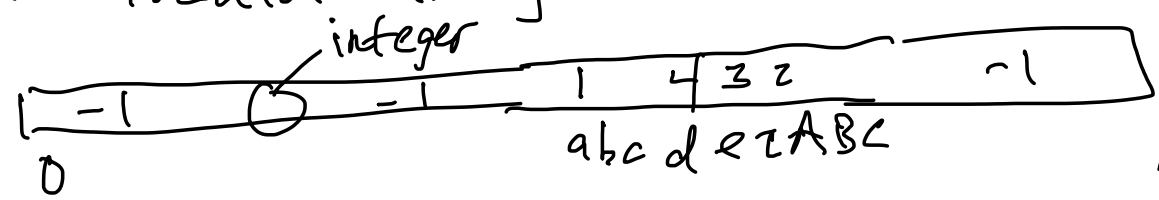
---

t = detective
p = date
date
date

---

precompute a "location" array

location



pattern: dated

```
for (charVal = 0; charVal < 128; charVal += 1){
        location[charVal] = -1;
}
for (pIndex = 0; pIndex < m; pIndex += 1){
        location[p[pIndex]] = pIndex;
}
```

On mismatch, t Index += max(1, pIndex − location[α])
(see α in text)

case: α not in P, pIndex = m−1

location[α] = −1

max(1, m−1−(−1)) = m

case: α not in P, pIndex = j

partial shift: j + 1

case α is in P.

shift enough so last instance of α in P aligns with α in T.

Match Heuristic: Use a shift table, like Knuth-Morris-Pratt.

Nigel Horspool's heuristic:

at a mismatch, consider β, the characters in T where we started matching. Shift pattern so β in T aligns with its rightmost occurrence in P. [not counting last place]

t d e t e c t i v e

p d a t e d a t e

Typical complexity:

$O(n \cdot m)^{n/m}$

Other pattern-matching developments

approximate matching (Udi Manber)

agrep

exact matching: grep ← print

global, regular expression

example: (in ed)

g/date/p

% grep date FILE ...

% egrep (more complex expressions)

% zgrep (for compressed files)

Advanced: regular expression (Perl regexp)

- exact strings: conandrum
- don't care symbol: con.ndr.m (. = don't care)
- character classes: c[ou-s]nundrum
- predefined character classes:

\w	word-like	\W	not word-like
\s	space-like	\S	not space
\d	digit	\D	not digit

- Unicode character classes:

\p{ASCII}

\p{digit}

\p{Final_Punctuation}

- pseudo-characters

  ^ start of text

  $ end of text example: ^drum$

- building expressions from others

  alternation: drum|com

  repetition um+ : 1 or more times

  um* : 0 or more times.

  {wo}th\s+the

  K{2,4}

- capture groups

  (drum|com)(fio)

  $1 $2

- zero-width assertions.

  (?=con) condition

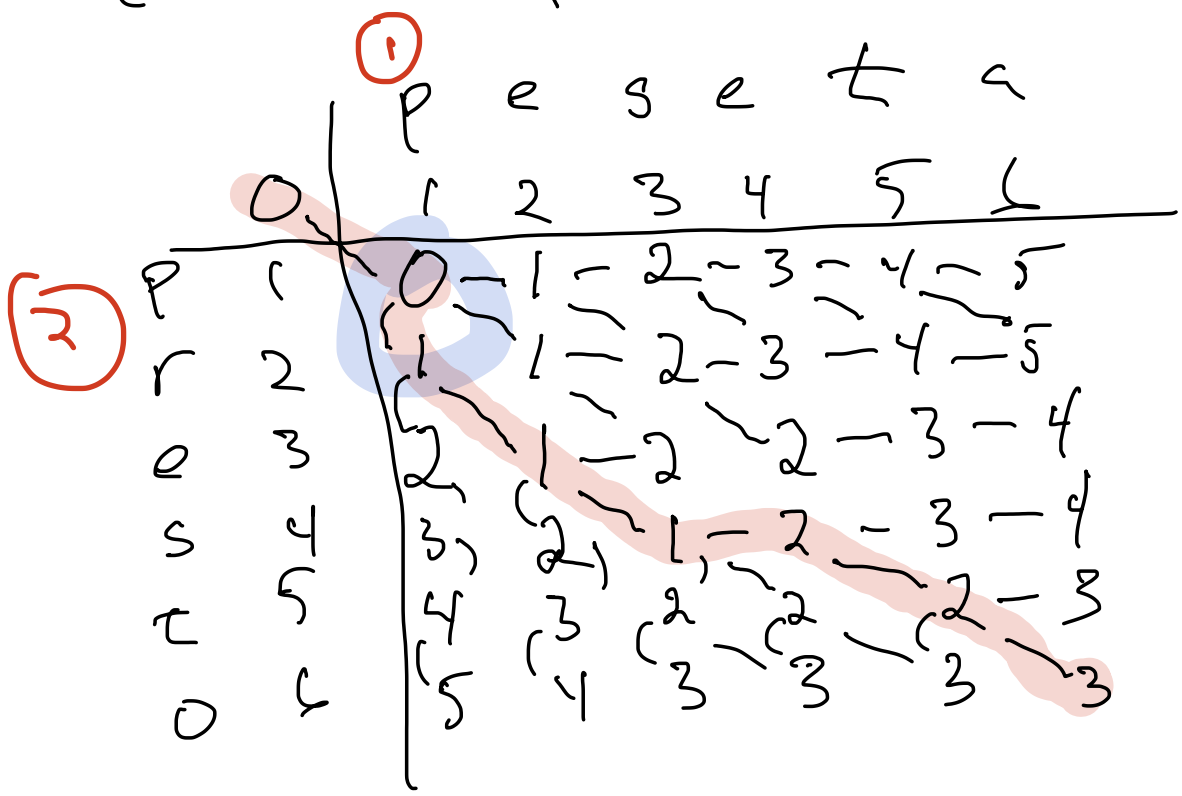
↳ man perlre

perlunicode

Edit distance: how hard is it to change one string into another by: ① delete char, ② insert char ③ replace char.

ghost $\xrightarrow{D}$ host $\xrightarrow{I}$ houst $\xrightarrow{R}$ house

peseta → presto

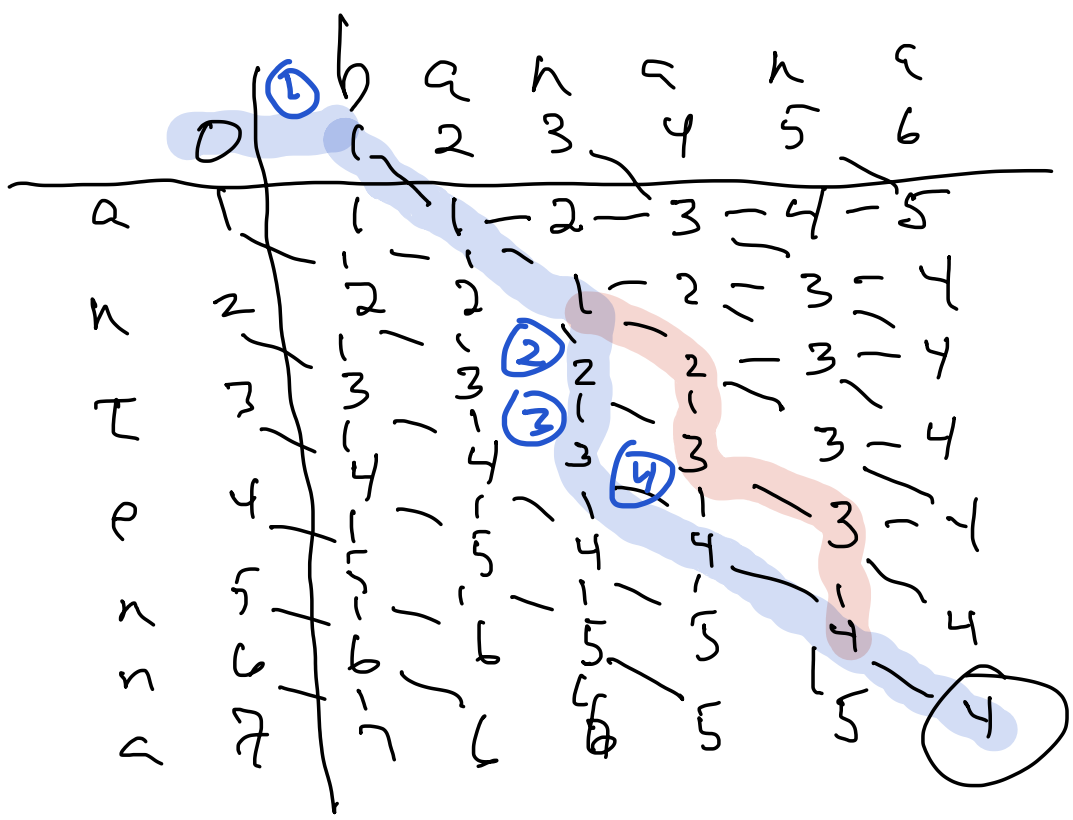


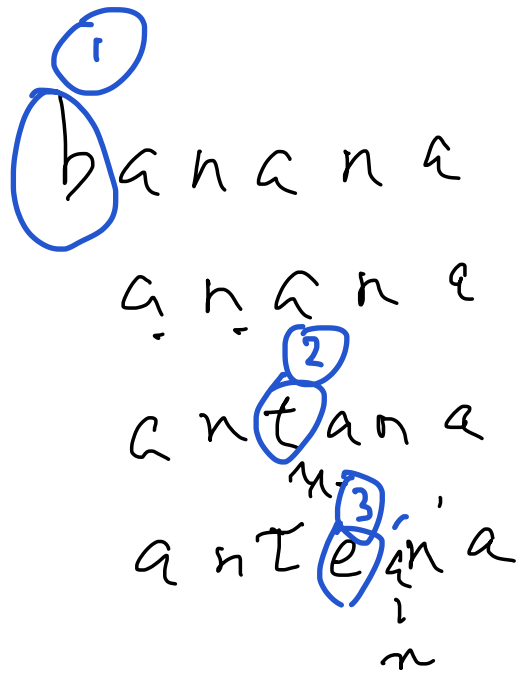
p e s e t a

p r e s i d e n t o

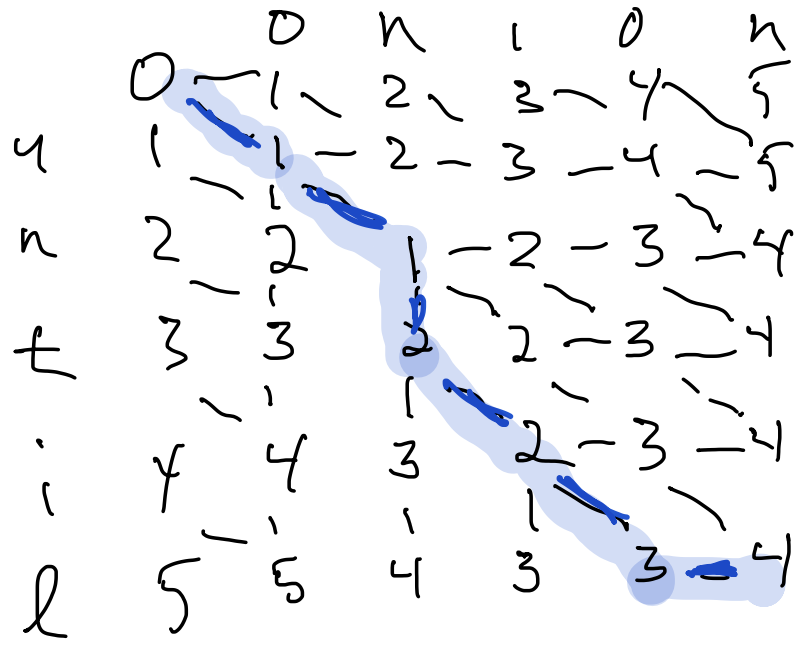
Complexity: lengths are n, m

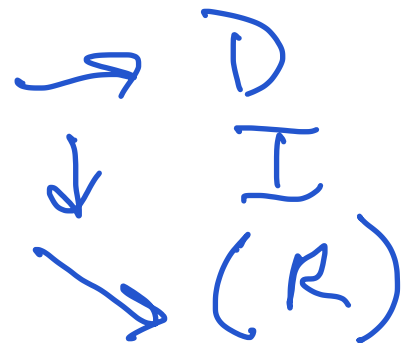
$\Theta(n\,m)$





Onion → until





onion
union
untion
untiln
until

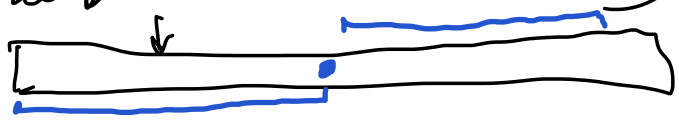
Category: dynamic programming
solve all smaller problems first.

- Divide + conquer
  - if problem is trivial, solve it.
  - divide problem into (a) smaller problems.
    - of size (n/b)
  - do those easier problems
  - combine the answers [at cost (n^k)]

In general $C_n = n^k + a\,C_{n/b}$

Examples

① jth smallest element of array

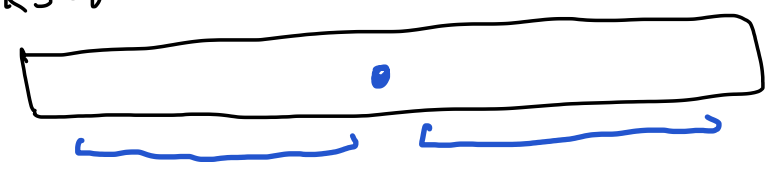


a = 1
b = 2
k = 1

$a \quad b^k$

$1 < 2 \quad : \Theta(n^k) = \Theta(n)$

② Quicksort



$a = 2$

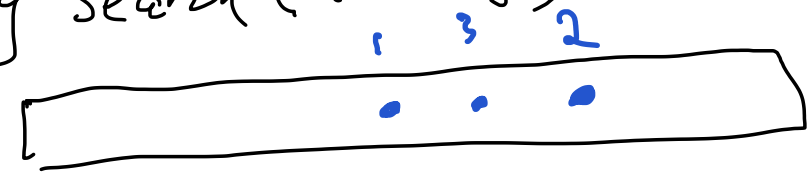
$b = 2$

$k = 1$

$a \quad b^k$

$2 = 2 : \Theta(n^k \log n) = \Theta(n \log n)$

③ Binary search (array)



$a = 1$

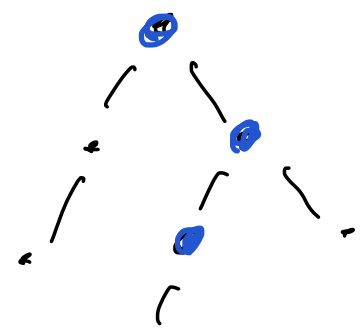
$b = 2$

$K = 0$

$a \quad b^k$

$1 = 1 : \Theta(n^k \log n) = \Theta(\log n)$

④ Binary search (tree)



$a = 1$

$b = 2$ $\Theta(\log n)$

$K = 0$

⑤ Multiplication (Karatsuba)

$x = a \cdot 10^{n/2} + b$ $\quad a = 3$

$y = c \cdot 10^{n/2} + d$ $\quad b = 2$

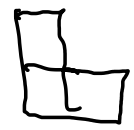
$k = 1$

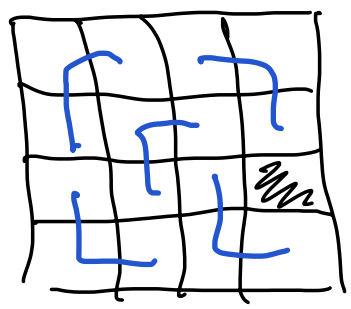
$a \quad b^k$

$3 > 2 \quad \Theta(n^{\log_b a}) = \Theta(n^{\log_2 3})$

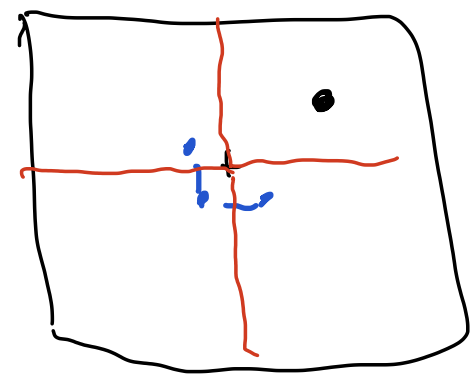
$\cong \Theta(n^{1.5})$

(5) Tile an $n \times n$ board with triminos missing 1 cell. ($n = 2^p$)





n



$a = 4$
$b = 2$
$k = 0$

$a = b^k$
$4 > 1$

$\Theta(n^{\log_b a}) = \Theta(n^{\log_2 4})$
$= \Theta(n^2)$
(if n is the side of the board)

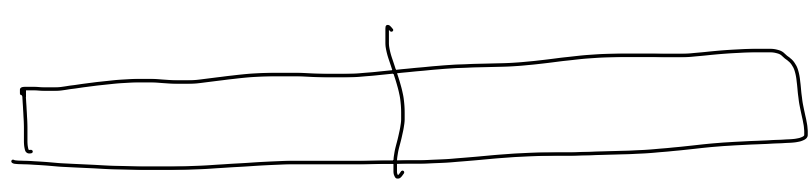
if n = # of cells:

$a = 4$
$b = 4$
$k = 0$

$a = b^k$
$4 > 1$

$\Theta(n^{\log_b a})$
$= \Theta(n^{\log_4 4})$
$= \Theta(n)$

(6) Mergesort



$a = 2$
$b = 2$
$k = 1$

$a = 2^{b^k}$ $\Theta(n^k \log n) = \Theta(n \log n)$

- Greedy: enlarge a partial solution by best single step.

(1) Compute coins for amount
use biggest coins first
76¢: 25, 25, 25, 1
not optimal: coins 1, 6, 10,
to get 12¢: 10, 10, 1, 1
6, 6
1, 5, 10 coins: greedy optimal.

oz : T : ... : C : p : g : ... : 9
weights on barbell:

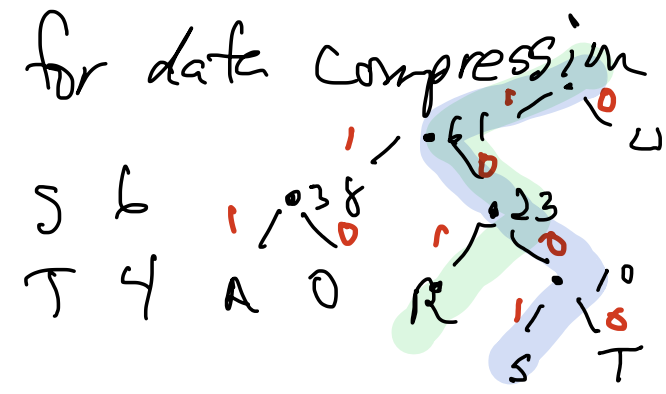
(2) Kruskal's algorithm,
greedily add edges

(3) Prim's algorithm
greedily add vertices

(4) Dijsbra's algorithm for all shortest paths
expand the shortest path so far.

(5) Huffman codes for data compression

frequencies
_	60
A	22
O	16
R	13



text with all those letters, at those frequencies ⇒ 121 chars

7 bits/char = 847 bits

code table:

␣	0
A	111
O	110
R	101
S	1001
T	1000

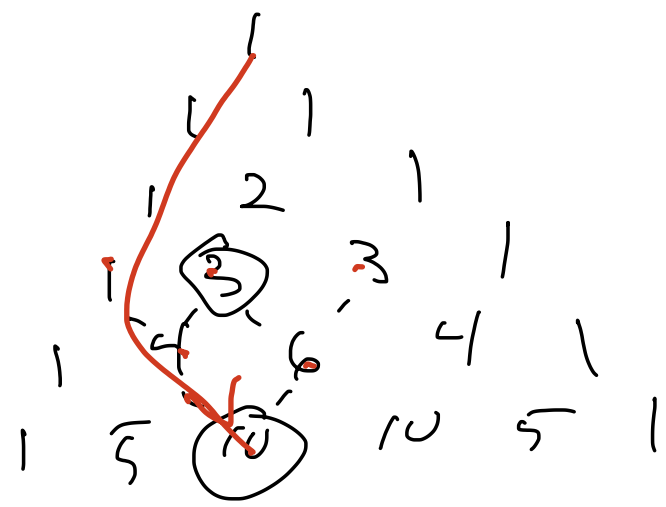
$\underbrace{60.}_{␣} + \underbrace{22.}_{A} \cdot 3 + \cdots$

= 253 bits

(6) Add a millim floats, all in [0 .. 1] with best precision.

repeatedly: add two smallest numbers, put result back in the set.

Dynamic: $\binom{10}{5}$



Fibonacci Numbers

1 1 2 3 5 8 13

good

bad

Hashing

Basic idea: store data ~~that~~ has key K in an array at index $h(K)$

h: hash function.

Handling collisions:

open addressing: (clustering)

find an alternative index

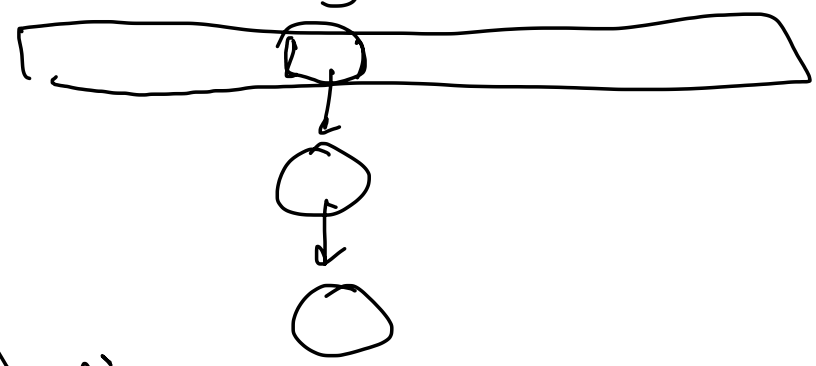
linear probing $p_i = h(K) + i$

quadrating probing $p_i = h(K) + i^2$

add-the-hash $p_i = h(K)(i+1)$

double hashing $p_i = h_1(K) + i\,h_2(K)$

external chaining



Good hash function

fast

uniform (equally likely to result in 0..s-1)

How big should the array be?

open addressing: 3x # of insertions.

external chaining: 1x # of insertions

If too small:

rehash into larger table.

Cryptographic hashes (digests)

one-way function from strings to bits

(strings: arbitrary length; bits: example: 256)

purpose:

- store passwords.
- noticing plagiarism.
- for authentication (signatures)
- intrusion detection.

Graphs

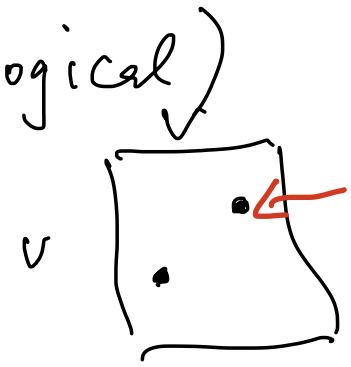
vertex/vertices, edges

directed / undirected	weighted / unweighted	dense / sparse

representations

special-purpose (genealogical)

adjacency matrix



adjacency list

Algorithms

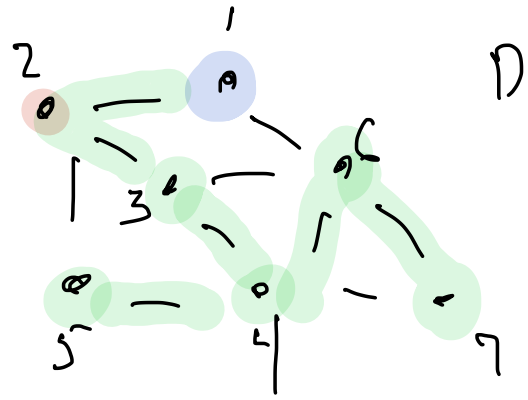
compute degree of each vertex

Depth-first search (DFS)

keeping track of what vertices have been visited

check for connectivity

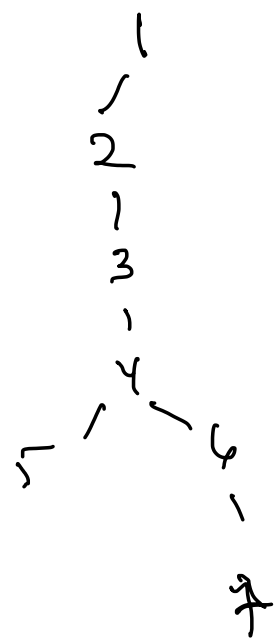
Breadth-first search (BFS)



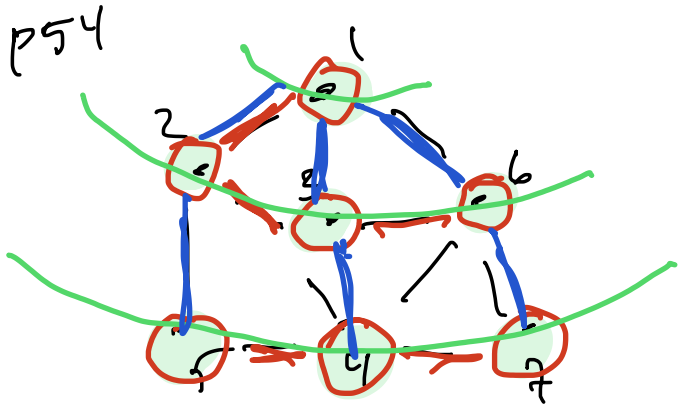
DFS

p52

navigation tree



p54



if weighted,
queue → heap.
p55

Dijkstra's algorithm: all shortest paths from a starting vertex

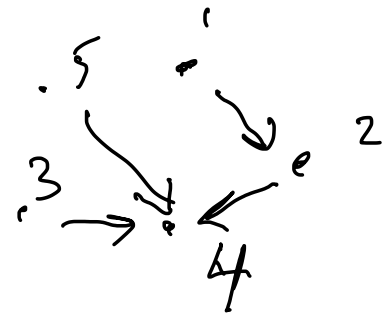
---

Spanning trees (subset of edges connecting vertices)

Prim: add best vertex (closest to growing tree)

Kruskal: add best edge (shortest, without introducing a cycle)

Union-Find



u 1, 2
u 4, 3
u 5 4
u 1 3

Numerical algorithms

Euclid GCD (greatest common divisor)

a	39	12	3
b	12	3	0

Fast exponentiation

$243^{745} \mod 452$

a

represent in binary (1)011101001

$(((((((((1^2 \cdot a)^2)^2 a)^2 a)^2)^2 a)^2)^2)^2 a$

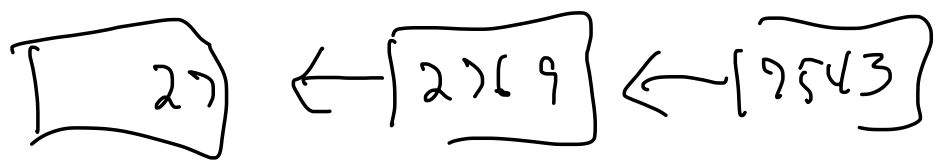
14 multiplies

14 mods

Integer multiplication

Bignums: linked list of smaller chunks.

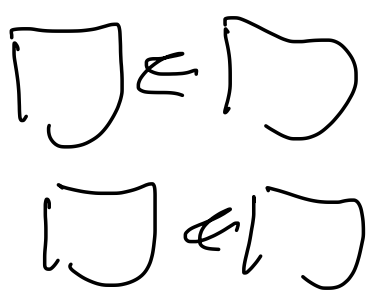
7543276927



n cells

Karatsuba's method.

if $n = 2$,

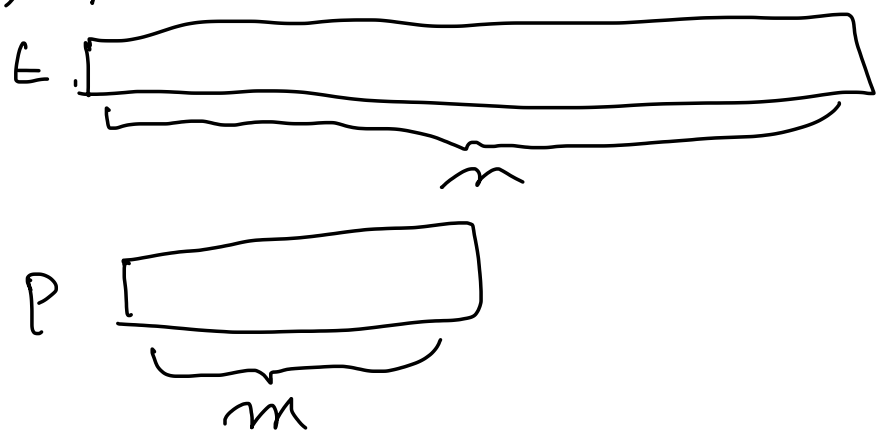


4 multiplies

$\Rightarrow$ 3 multiplies

$C_n = n + 3C_{n/2}$ $\quad \Theta(n^{\log_2 3}) \approx \Theta(n^{1.6})$

Strings, pattern matching



1) Brute force: try p at every location in t

2) Rabin-Karp: only try p is signature of p matches signature of the slice in t.
   Constant-time update of slice's signature

3) Knuth-Morris-Pratt
   on failure, slide p based on a shift table

4) Boyer-Moore
   test p in reverse order, shift based on occurrence heuristic.
   shift table
   Horspools

Edit distance

Categories
- greedy: each step, do the best thing
- divide and conquer: subdivide into smaller problem: Recursion theorem
- dynamic programming: solve all smaller problems first

Tractibility $O(n)$ good $O(n^3)$ ok. $O(n^{20})$ tractable $O(2^n)$ bad.