22.03.22.

Discrete logarithm computation with generic algorithms.

- Section 9.2 of Morain
- chapter 5 of Washington
  - Shanks Baby step Giant Step (5.2.1)
  - Pollard ρ : inspired by Floyd path finding algorithm in a graph.
  - Pohlig - Hellman : 5.2.3.

These algorithm apply to generic groups G: it can be the multiplicative subgroup of a finite field, or the group of points of an elliptic curve over a fin. fi

Notation: G is a group with a multiplication: $x, y \in G \to x \cdot y \in G$.

-> we have exponentiation in G: x^n where $n \in \mathbb{Z}$,

$x^0 = 1$ (the neutral element of G: this is 1 in $\mathbb{F}_q^\times$, this is $\mathcal{O}$ on $E(\mathbb{F}_p)$).

$x^{-n} = (1/x)^n$ and inversion is in the usual sense in $\mathbb{F}_q^\times$,

inversion is $-P = (x_P, -y_P)$ on an elliptic curve.

Discrete logarithm (D-log): given G, a generator g, and a target $h \in G$, compute x in $\{0, 1, \dots, \# G - 1\}$ such that $g^x = h$ in G.

Example: • if $G = \mathbb{F}_p^\times$ the multiplicative subgroup of a finite field where p is prime then $\# G = p-1$ and the discrete log is such that $g^{\log_g h} = h$ in $\mathbb{F}_p^\times$ (that is mod p).

• if $G = E(\mathbb{F}_p)$, then $\# G = \# E(\mathbb{F}_p) = p+1-t$ where t is the trace of the curve, and if P is a generator of $E(\mathbb{F}_p)$ (of order $p+1-t = q$), then the d-log is such that $Q \in E(\mathbb{F}_p)$ has d-log $[\log_P Q] P = Q$ where exponentiation becomes scalar multiplication.

in Sage Math, P.order() computes ~~discrete log~~. the order of the point (costly).

for a point $G \in E$ and another point $P \in E$,

G.discrete_log (P) computes the discrete log of P in basis G (costly too).

Pohlig Hellman method.

Group homomorphism. Let G a multiplicative group of order N.
Suppose we can factor N into distinct prime factors

$$N = \prod_{p_i \text{ prime}} p_i^{e_i} = p_1^{e_1} \cdot p_2^{e_2} \cdots p_n^{e_n}$$

One can map $g, h \in G$ to subgroup by exponentiation to the cofactors of the subgroups:

$$g \longmapsto g^{N/p_i^{e_i}} \quad , \quad h \longmapsto h^{N/p_i^{e_i}}$$

so we can solve independently the discrete log problem in distinct subgroups of order a prime power $p_i^{e_i}$:

$$g_i = g^{N/p_i^{e_i}} \quad , \quad h_i = h^{N/p_i^{e_i}}, \qquad \text{compute } \log_{g_i}(h_i) \text{ in } \{0, 1, \dots, p_i^{e_i} - 1\}$$

$\to$ we have $\boxed{\log_g h \mod p_i^{e_i}}$

if we have n processors, we can parallelize over n tasks the d-log computation
$\to$ it means that the complexity in time is not function of N but function of $\max p_i^{e_i}$.

Now, we are left with the computation of $\log_{g_i}(h_i) \mod p_i^{e_i}$.

$\to h_i = g_i^{a}$ where $a = a_0 + a_1 p_i + a_2 p_i^2 + \dots + a_{e_i - 1} p_i^{e_i - 1}$ in basis p_i.

ex: $p_i^{e_i} = 3^2$, $a = 7 = \underset{a_0}{\underline{1}} + \underset{a_1}{\underline{2}} \times 3$ for example.

How to compute the a_j?

Let's cancel the $a_1, a_2, \dots a_{e_i - 1}$ to compute a_0.

$$h_i^{p_i^{e_i - 1}} = g_i^{a \cdot p_i^{e_i - 1}} \Big\} = a_0 p_i^{e_i - 1} + a_1 p_i^{e_i} + a_2 p_i^{e_i + 1} + \dots + a_{e_i - 1} p_i^{2(e_i - 1)}$$

but what is the order of g_i? g_i has order $p_i^{e_i}$ $\to$ $g_i^{p_i^{e_i}} = 1$ hence all the $a_1, \dots$ cancel.

$$h_i^{p_i^{e_i - 1}} = g_i^{a_0 p_i^{e_i - 1}} = \left(g^{p_i^{e_i - 1}}\right)^{a_0} \text{ and } g^{p_i^{e_i - 1}} \text{ has order } p_i.$$

$\to$ we are left with the computation of a d-log mod p_i.

Once we know a_0, we have:

known

$$h_i = g_i^{a_0 + \underbrace{a_1 p_i + a_2 p_i^2 + \dots + a_{e_i-1} p_i^{e_i-1}}_{\text{unknowns}}}$$

$$\vec{h}_1 = h_i / g_i^{a_0} \quad , \quad h_1 = g_i^{a_1 p_i + a_2 p_i^2 + \dots + a_{e_i-1} p_i^{e_i-1}}$$

raise h_1 to the power $p_i^{e_i-2}$ to cancel all the $a_2, a_3, \dots a_{e_i-1}$ coefficients

$\rightarrow$ we have $h_1^{p_i^{e_i-2}} = \left(g_i^{p_i^{e_i-1}}\right)^{a_1}$ $\rightarrow$ compute a d-log $a_1 \bmod p_i$

the next step will set $h_2 = h_1 / g_i^{a_0 + a_1 p_i}$ and compute the d-log $a_2 \bmod p_i$.

in total: the procedure computes sequentially e_i times a d-log mod p_i.

the complexity is e_i (time to compute a d-log mod p_i).

and it is not parallel this time.

In conclusion, Pohlig-Hellman has complexity

$$\max_{1 \le i \le n} \left(e_i \cdot (\text{complexity of d-log mod } p_i) \right) \text{ where } N = \prod_{i=1}^{n} p_i^{e_i}.$$

We will see just after that the complexity of computing a d-log mod p_i is in $O(\sqrt{p_i})$ with generic algorithms.

See Galbraith book Sect. 13.2 and Alg. 13 for more details.

Baby-Step Giant Step to compute a discrete log in a cyclic subgroup.
time-memory trade-off.

$h = g^a$, and we want to compute $a \bmod p$.

let $m = \lceil \sqrt{p} \rceil$ that is the ceiling of $\sqrt{p}$.
then $a = a_0 + a_1 m$ where $m = \lceil \sqrt{p} \rceil$ and $0 \leq a_0, a_1 < m$.
(write the Euclidean division of a by m to get a_0 and a_1.)

$$h = g^{a_0 + a_1 m} = g^{a_0} g^{a_1 m} = g^{a_0} / (g^{-m})^{a_1}$$

hence $h \cdot \left(\frac{1}{g^m}\right)^{a_1} = g^{a_0}$ baby steps: enumerate all g^i for $0 \leq i \leq m$

giant steps: enumerate all $h \cdot (1/g)^j$ for $j = 0, 1, \ldots$ until a match is found.
Requires comparison and search in the baby step database to be as fast as possible.

Algo 14 in Galbraith book p. 273.

input: $g, h \in G$ of order p, output: a such that $h = g^a$, or bottom $\perp$. (fail)
$m = \lceil \sqrt{p} \rceil$ (\lfloor \rfloor in latex, $\to \lfloor \ \rfloor$, \lceil \rceil is $\lceil \ \rceil$).
easily searched structure L (hash table, binary tree). $\to$ search in constant time.
$f = 1$
for $i = 0$ to m do : store (f, i) in L (with f the key of the hash table for ex.)
$f \leftarrow f \cdot g$ inserting costs $O((\log p)^2)$ bit op. since comparison costs $O(\log p)$ bit op.
end for L contains all the g^i, $0 \leq i \leq m$. it costs m mult. $\to \boxed{\sqrt{p}}$.
$u = g^{-m} = (1/g)^m$ storage $(\log p) \sqrt{p}$.
$y = h$ $j = 0$
while (y does not match a key in L) do
  $y = y \cdot u$, $j = j+1$
end while
if $\exists (f, i)$ in L such that $f = y$ then
  return $i + mj$
else
  return $\perp$
end if.

Theorem 13.3.1. let G be a group of order p. Suppose the elements of G are represented with $O(\log p)$ bits and that the group operations can be performed in $O((\log p)^2)$ bit operations. The BSGS algo for DLP in G has running time $O(\sqrt{p}(\log p)^2)$ bit operations. The algo requires $O(\sqrt{p} \log p)$ bits of storage.

22.03.22

What is a hash table? (What is a binary search tree?)

set of data of the same type
here: g^i for $0 \leq i \leq m$
$g^i \in G$

$\rightarrow$ hash function $G \to \{0, 1, \dots, m\}$

$f(g_i)$ = some encoding to $\mathbb{N}$

$H(g_i) = f(g_i) \bmod (m+1)$

initialize an array T of size $N := m+1$.

for each new item g^i, compute $H(g^i)$. We would like to store it at index $H(g^i)$.

$\rightarrow$ what do we do if there is a collision?

$T[H(g^i)]$ is actually a linked list: we store buckets of values that are distinct but sharing the same hash function.

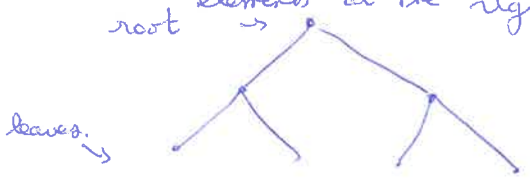
$T[n] \longmapsto$ NULL at initialisation.

$T[n] \longmapsto$ | @1 | (g^i, i) | $\longrightarrow$ | @2 | (g^ℓ, ℓ) | etc..

finding an element in T is done in constant time:
just compute $H(h_j)$ and test whether $T[H(h_j)]$ is NULL or some linked list. Search in the linked list.

Binary search tree.

elements at the left are smaller than the root,
elements at the right are larger (greater) than the root.



one can consider lexicographical ordering for example.

22.03.22.

Pollard ρ and random walks. Galbraith's book Chapter 14.

- Birthday paradox
- Floyd algorithm of cycle path finding in a graph
- Pollard ρ.

Theorem (Birthday paradox) 14.1.1 in Galbraith book.

Let S be a set of N elements. If elements are sampled uniformly at random from S then the expected number of samples to be taken before some element is sampled twice is less than $\sqrt{\pi N/2} + 2 \approx 1.253\sqrt{N}$.

a repeat, match or collision : something sampled twice.

The number of elements that need to be selected from S to get a collision with probability $1/2$ is $\sqrt{2 \log(2) N} \approx 1.177\sqrt{N}$.

Application: $\sqrt{2\log(2) 365} \approx 22,49$, $\sqrt{\pi\, 365/2} = 23,94$

-> with 23 people, the probability to have a birthday collision is ~~…~~ 1/2.

Note that this is about a collision: the birthday date is not set. (We are not fixing any value in S).

Floyd algorithm ("Hare & tortoise") Robert W. Floyd, US, Turing award 1978. 1936 – 2001

Let S be a sequence (a linked list, the values taken by a function) equiped with a "next" to get the next element of the sequence.
The end point is Null (None in Python).

```
def has_cycle(node s):
    if s is None:
        return False
    tortoise = s
    hare = s.next()
    while (tortoise != hare):
        if hare is None:
            return false
        hare = hare.next
        if hare is None:
            return False
        hare = hare.next
        tortoise = tortoise.next
    return True
```

The hare goes twice faster than the tortoise.

- if there is no cycle (no collision) then the hare will reach None (the end)
- if there is a cycle:
  - the tortoise gets closer to the cycle at each step
  - the hare, once in the cycle, will never hit a "None"
  - so at some point, both will be going inside the cycle, and then the hare will hit the tortoise at some point.

Pollard ρ algorithm applies this idea to finding the discrete log in a cyclic group G   G of order p

find a_i, b_i, a_j, b_j such that

$$g^{a_i} h^{b_i} = g^{a_j} h^{b_j} \quad \text{and } b_i \not\equiv b_j \bmod p.$$

$$\Leftrightarrow g^{a_i - a_j} = h^{b_j - b_i} \Leftrightarrow g^{(a_i - a_j)/(b_j - b_i) \bmod p} = h.$$

$$\log_g h = \frac{a_i - a_j}{b_j - b_i} \bmod p.$$

Alg 16 p290 (Galbraith ~~book~~ ch. 14).

input: $g, h \in G$

output: a s.t. $h = g^a$, or $\perp$

1. choose a function walk
2. $x_1 = g$. $a_1 = 1$, $b_1 = 0$
3. $(x_2, a_2, b_2) = \text{walk}(x_1, a_1, b_1)$
4. while $x_1 \neq x_2$:
5. $\quad (x_1, a_1, b_1) = \text{walk}(x_1, a_1, b_1)$ tortoise
6. $\quad (x_2, a_2, b_2) = \text{walk}(\text{walk}(x_2, a_2, b_2))$ twice fast hare
7. end while
8. if $b_1 \equiv b_2 \bmod p$ then
9. $\quad$ return $\perp$
10. else:
11. $\quad$ return $(a_2 - a_1)(b_1 - b_2)^{-1} \bmod p$
12. endif.

what do we choose for the walk function?

Define n_S subsets S_i of G of the $\approx$ same size $\#G/S$.

Precompute $g_j = g^{u_j} h^{v_j}$ for $0 \leq j \leq n_S - 1$. for uniformly randomly chosen $0 \leq u_j, v_j < p$.

$x_1 = g$.

original ρ walk: $x_{i+1} = f(x_i) = \begin{cases} x_i^2 & \text{if } S(x_i) = 0 \\ x_i \cdot g_j & \text{if } S(x_i) = j,\ j \in \{1, \dots, n_S - 1\} \end{cases}$

additive ρ walk: $x_{i+1} = f(x_i) = x_i\, g_{S(x_i)}$

epact: smallest i s.t. $x_{2i} = x_i$. expected value $0.823 \sqrt{\pi p/2} = \underbrace{\zeta(2)/2}_{\text{precomputed}} \sqrt{\pi p/2}$. zeta(2)

Complexity of historical ρ: $(3.093 + o(1))\sqrt{p}$ group op.

Best method: $(\sqrt{\pi/2} + o(1))\sqrt{p} \approx (1.253 + o(1))\sqrt{p}$. group operations.

$a_{i+1} = \begin{cases} 2a_i \bmod p & \text{if } S(x_i) = 0 \\ a_i + u_{S(x_i)} \bmod p & \text{if } S(x_i) > 0 \end{cases}$

$b_{i+1} = \begin{cases} 2 b_i \bmod p & \text{if } S(x_i) = 0 \\ b_i + v_{S(x_i)} \bmod p & \text{if } S(x_i) > 0 \end{cases}$