0.1 GCD and Bezout's identity

- *Algorithm:* name (algo. 1)
- *Input:* two integers a and b
- *Complexity:* complexity of the algorithm, e.g. $\mathcal{O}(\log a)$
- *Data structure compatibility:* N/A
- *Common applications:* Number theory

Problem. GCD and Bezout's identity

Given two integers a and b, calculate the GCD of a and b as d. Bezout's identity is that there exist two integers x and y such that $ax + by = d$.

Description

1. Bezout's identity

Bezout's identity is a fundamental theorem in the field of number theory, which is described as follow: given two integers a and b, assume the greatest common divider(GCD) of them is d, there exist two integers x and y, such that $ax + by = d$.

I will give my own proof here. Since the GCD of a and b is d, we can assume $a = d \times p$ and $b = d \times q$, where $(p, q) = 1$. Then we need to find x and y such that $px + qy = 1$. This is equivalent to $p|qy - 1$, which is to find y such that $qy \equiv 1 \mod p$. Since $(y, p) = 1$, we can always find such q. Qed. In the last step we can also let $q = 0, 1, \cdots, p - 1$, then for these q, $qy \equiv r \mod p$. r is different as q becomes different. So there must be a q such that $qy \equiv 1 \mod p$.

Notice that Bezout's identity can be generalized into more than two integers. Given n integers $a_1, a_2, \cdots, a_n$, assume the GCD of them is d, then there exists integers $x_1, x_2, \cdots, x_n$, such that $d = a_1x_1 + a_2x_2 + \cdots + a_nx_n$. The proof using induction is not as straightforward as the proof for two-integer case. For finite field, we have similar Bezout's identity. Example of calculating GCD on finite field can be found in the slides provided on Friday's VE477 lecture[2].

This is my own naive proof using induction. For input of size two, we have already proved above. Now assume Bezout's identity is true for k inputs, and we need to prove that it is also true for $k + 1$ inputs. Assume the $k + 1$ inputs are $a_1, a_2, \cdots, a_k, a_{k+1}$, GCD of first k inputs as d, GCD of first $k + 1$ inputs as e, it is obvious that we can write $d = e \times f$ and $(a_{k+1}, f) = 1$. Using the induction hypothesis, there exists $x_1, x_2, \cdots, x_k$, such that $a_1x_1 + a_2x_2 + \cdots + a_kx_k = d$. We only need to find x_{k+1} such that $d + a_{k+1}x_{k+1} = e$. This is equivalent to $f + \frac{a_{k+1}}{e}x_{k+1} = 1$, since $(a_{k+1}, f) = 1$, because of the Bezout's identity with 2 inputs, there exist such x_{k+1}, Qed. For proof in finite field, maybe in the final exam, who knows.

2. GCD

When we need to take advantage of the Bezout's identity, we need to calculate the GCD of two different integers first. Here GCD problem is that given two integers a and b, calculate the largest positive integer d such that d divide both a and b. Notice that we can define the GCD of more than two integers in a similar way. The

commonly used algorithm for calculating GCD is Euclidean algorithm. Other common applications of GCD are extended Euclidean algorithm, linear Diophantine equations, and Chinese remainder theorem.

Euclidean algorithm is based on subtracting the smaller input from the larger one until one of the input becomes zero. We can speed up the process by using division instead of subtraction in some cases. The number of divisions required for Euclidean algorithm is $\mathcal{O}(\log(min(a, b)))$, where a, b are the input integers[1]. Thus we have the time complexity of $\mathcal{O}(\log(min(a, b)))$.

Algorithm 1: Euclidean algorithm

Input : two integers a and b
Output: the GCD of a and b

```
while b ≠ 0 do
    if a > b then
        a ≡ tmp mod b
        a = tmp
    end if
    else
        b ≡ tmp mod b
        b = tmp
    end if
end while
return a
```

References.

[1] H Grossman. "Discussions: On the number of divisions in finding a GCD". In: *The American Mathematical Monthly* 31.9 (1924), pp. 443–443 (cit. on p. 2).

[2] Manuel. *VE477 – Introduction to Algorithms (lecture slides)*. 2019 (cit. on p. 1).