0.1 Generating Partitions

- *Algorithm:* name (algo. 1)
- *Input:* An positive integer n
- *Complexity:* $\mathcal{O}(n^2)$
- *Data structure compatibility:* N/A
- *Common applications:* The Durfee square can be used to prove many partition identities in the field of combinatorics.

Problem. Generating Partitions

Given an positive integer n, we want to find the number of different tuples $(a_1, a_2, ..., a_k)$, such that two constrains are satisfied. Firstly, $a_1 + a_2 + ..., a_k = n$. Secondly, $a_1 \geq a_2 \geq ... \geq a_k > 0$[0].

Description

A first look at the problem

Given an positive integer n, we want to find the number of different tuples $(a_1, a_2, ..., a_k)$, such that two constrains are satisfied. Firstly, $a_1 + a_2 + ..., a_k = n$. Secondly, $a_1 \geq a_2 \geq ... \geq a_k > 0$. Some variation of this problem can be find by setting additional constrains like k is even number. We define a partition function $p(n)$ as the number of tuples we find in the problem. In this paper, we will focus on the partition problem without additional constrins to make our life easier. For example, we can find out there are only one partition (1) for $n = 1$. For $n = 5$, there are 7 partitions, (1, 1, 1, 1, 1), (1, 1, 1, 2), (1, 1, 3), (1, 4), (1, 2, 2), (5), (2, 3). Here we define part as the summand in each partition, for example $1 + 1 + 3$ is the part for tuple (1, 1, 3). The order of elements in the tuples does not matter in our problem.

The idea of this algorithm

To save time for future calculation, we use dynamic programming to store the values used in former calculation. Notice that in the for loop, for example, we can use the value of array[j - i]. This is quite similar to Bellman-ford that we learned in chapter 4 of the slides[0].

Time complexity

In this algorithm, there are two levels of for loops, the time complexity is $\mathcal{O}(n) \times \mathcal{O}(n) = \mathcal{O}(n^2)$, the first loop to set the value of array does not matter. Comparing to the size of input, which is $\mathcal{O}(\log n)$, the time complexity would be exponential. Some approximation algorithms are proposed to obtain the value with less accuracy within polynomial time. Also we can directly calculate the asymptotics of the number by

$$\log p(n) \sim C\sqrt{n}$$

where C is a constant and $C = \pi\sqrt{\frac{2}{3}}$[0].

Or we can express the formula in another way as

$$p(n) \sim \frac{1}{4n\sqrt{3}} \exp\left(\pi\sqrt{\frac{2n}{3}}\right)$$

.

Another way to tackle this problem

We can use generating function to calculate the value as well[0].

$$\sum_{n=0}^{\infty} q(n)x^n = \prod_{k=1}^{\infty}\left(1+x^k\right) = \prod_{k=1}^{\infty}\frac{1}{1-x^{2k-1}}$$

, by using this equation, we can get the number of partition by writing down the coefficient before the term x^n. For example, to find the number of partitions when $n = 8$, we need to calculate

$$\begin{gathered}\left(1+x+x^2+x^3+x^4+x^5+x^6+x^7+x^8\right)\left(1+x^2+x^4+x^6+x^8\right)\left(1+x^3+x^6\right)\\ \left(1+x^4+x^8\right)\left(1+x^5\right)\left(1+x^6\right)\left(1+x^7\right)\left(1+x^8\right)\\ = 1+x+2x^2+3x^3+5x^4+7x^5+11x^6+15x^7+22x^8+\cdots+x^{56}\end{gathered}$$

Then we can get the number by reading the coefficient of x^8, which is 22.

Further application

We can define the rank of a certain partition of the positive integer n as the largest integer k so that we have at least k numbers in partition that is greater or equal to k. Furthermore, two diagrams, namely Ferrers diagram and Young diagram are used for visualization representation of this kind of problem. Durfee square can be defined as the $k \times k$*squarecountingfromthelargestelementinthepartition*, where k stands for the rank we give.

Algorithm 1: Partition

Input : a positive integer n
Output: the number of partition, where partition is defined in problem definition

```
int array[n] for inti = 1; i <= n; i++ do
    array[i] = 1
end for
for inti = 1; i <= n; i++ do
    for intj = i; j <= n; j++ do
        array[j] = array[j] + array[j - i]
    end for
end for
return array[n]
```

References.

[0] George E Andrews. *The theory of partitions*. 2. Cambridge university press, 1998 (cit. on p. 1).

[0] Manuel. *VE477 – Introduction to Algorithms (lecture slides)*. 2020 (cit. on p. 1).

[0] whitman.edu. *Partitions of Integers*. [Online; accessed 5-October-2020]. 2020. URL: https://www.whitman.edu/mathematics/cgt_online/book/section03.03.html (cit. on p. 2).

[0] Wikipedia contributors. *Partition (number theory)*. [Online; accessed 5-October-2020]. 2020. URL: https://en.wikipedia.org/wiki/Partition_(number_theory) (cit. on p. 1).

0.2 Primality testing (AKS)

- *Algorithm:* name (algo. 2)
- *Input:* A positive integer $n > 1$
- *Complexity:* $\mathcal{O}(\log^{21/2} n)$
- *Data structure compatibility:* N/A

- *Common applications:* Primality testing is useful in the field of Cryptography and complexity theorem.

Problem. Primality testing (AKS)

Given a positive integer $n > 1$, we want to decide whether it is a prime number.

Description

A first look at the problem

In this problem, we are going to discuss an algorithm that can decide whether an integer $n > 1$ is a prime or not(which means composite). In the field of Cryptography, many protocols are designed based on large prime numbers. The question comes up: can we do primality testing in polynomial time? Note that the size of input is not the integer n but the bits that it has, which is $\mathcal{O}(\log n)$. So brute force is never recommended since it requires to test all $1 < i < \lfloor\sqrt{n}\rfloor$ and whether $i|n$. Since we need to compare the time complexity with input size, it will be an exponential-time algorithm.

Some algorithm not that good In algorithm for this problem, we need to focus on four important characters: general, deterministic, unconditional, and polynomial[0]. Here general means that all the input can be calculated, deterministic is the opposite of random algorithm, unconditional means that the proof of its correctness does not depend on any unproved theorem, lastly, polynomial is the requirement of time complexity. People are able to design algorithms that satisfy three conditions out of four, but no one can satisfy all until the idea from Agrawal Kayal Saxena.

Although there are many algorithms proposed that is not perfect, they focus on different characters and is still important. I will introduce them by some examples. Lucas–Lehmer primality test has a limitation that is only used for Mersenne numbers[0]. In real life application, we are interested in other integers as well, so we need to explore more. Miller-Rabin algorithm is introduced in VE475. It is a random algorithm, which means we can not be 100 percent sure about its result. If the algorithm says n is composite, then it must be composite. But when it says that it is a prime, it has a probability. So to increase the probability, we need to use this algorithm with different parameters[0]. Another algorithm, Miller test, can be determined, but we need to assume the correctness of an unproven lemma Riemann's Hypothesis[0].

AKS

This is an algorithm proposed by three Indian professors in 2002. Notice that it is an NP and co-NP problem. It is trivial to prove that this is in co-NP, just take an factor p of the number n as the certificate, we can easily calculate the GCD of p and n and get the decision. Pratt has proven that it belongs to NP in 1974[0]. The key idea in AKS is the equivalence of being prime and a module equation[0].

Key idea: given $n > 1$ to be an integer, a is another positive integer that is co-prime to n. Then n is prime if and only if

$$(X + a)^n \equiv X^n + a \mod n$$

Here is a little proof of this idea. We discuss two cases. First, if n is prime, then we recall binomial expansion and look at the coefficients carefully. Recall the definition of binomial coefficients.

$$\binom{n}{i} = \frac{n(n-1)\cdots(n-i+1)}{i!}$$

$\binom{n}{i}$ are dividable by n when i is greater than 0. For a^n, we can use Fermat little theorem and get $a^n \equiv a^{n-1} \times a \equiv a \mod n$. Another case is that when n is a composite, assume there is a prime factor of n and satisfies $p^k \| n$. Note that based on Fundamental Theorem of Arithmetic, we can always find such q and k. Then

we consider the coefficient $\binom{n}{p}$, but there is an additional p in $p!$, so there is only k-1 p remaining, but we need k[0]. So this term will not satisfy the module equation. Since an integer greater than 1 can be either prime or composite, the proof is done.

Time complexity

The step that judge the perfect power cost $\mathcal{O}(\log^3 n)$[0]. So we do not need to worry about it. In the next step, we need to find the r, this step takes $\mathcal{O}(\log^7 n)$[0]. So the for loop afterwards requires to calculate $\mathcal{O}(\log^5 n)$ times gcd and thus $\mathcal{O}(\log^5 n)$. GCD takes another $\mathcal{O}(\log n)$, which leads to $\mathcal{O}(\log^6 n)$. The if afterwards does not matter, so we need to care about the next for loop. This last step requires $\mathcal{O}(r^{3/2} \times \log^3 n) = \mathcal{O}(\log^{21/2} n)$[0]. Compare the complexity before, we get the overall complexity as $\mathcal{O}(\log^{21/2} n)$.

Correctness

Too difficult.

Algorithm 2: AKS

Input : an integer $n > 1$
Output: Prime or Composite

```
if n is a perfect power then
    return Composite
end if
find smallest r where o_r(n) > log^2 n
for a ≤ r do
    if 1 ¡ gcd(a, n) ¡ n then
        return Composite
    end if
end for
if n ≤ r then
    return Prime
end if
for a ≤ ⌊√φ(r) log n⌋ do
    if ((X + a)^n ≠ X^n + a (mod X^r − 1, n)) then
        return Composite
    end if
end for
return Prime
```

References.

[0] Manindra Agrawal, Neeraj Kayal, and Nitin Saxena. "PRIMES is in P". In: *Annals of mathematics* (2004), pp. 781–793 (cit. on pp. 3, 4).

[0] JW Bruce. "A really trivial proof of the Lucas-Lehmer primality test". In: *The American mathematical monthly* 100.4 (1993), pp. 370–371 (cit. on p. 3).

[0] Manuel. *VE475 – Introduction to Cryptography (lecture slides)*. 2020 (cit. on p. 3).

[0] Gary L Miller. "Riemann's hypothesis and tests for primality". In: *Journal of computer and system sciences* 13.3 (1976), pp. 300–317 (cit. on p. 3).

[0] Vaughan R Pratt. "Every prime has a succinct certificate". In: *SIAM Journal on Computing* 4.3 (1975), pp. 214–220 (cit. on p. 3).

[0] whitman.edu. *Partitions of Integers*. [Online; accessed 5-October-2020]. 2020. URL: https://www.whitman.edu/mathematics/cgt_online/book/section03.03.html (cit. on p. 4).

[0] Wikipedia contributors. *AKS primality test*. [Online; accessed 5-October-2020]. 2020. URL: https://en.wikipedia.org/wiki/AKS_primality_test (cit. on p. 3).

0.3 Gradient descent

- *Algorithm:* Gradient Descent(algo. 3)
- *Input:* A vector
- *Complexity:* None.
- *Data structure compatibility:* None
- *Common applications:* Artificial intelligence

Problem. Gradient descent

The purpose of optimization is to minimize our target function, for example,

$$\min f(x) \tag{0.3.1}$$

However, for some large scale function, analytical solution $x^\star = (A^\mathsf{T}A)^{-1}A^T Y$ is unsolvable and time consuming ($\mathcal{O}(n^3)$). So the descent method is raised to solve optimization.

Description

Descent Method

To solve equation 0.3.1, a sequence of points $x^{(k)}$ is produced to approach the optimum.

$$x^{(k+1)} = x^{(k)} + t^{(k)}\Delta x^{(k)} \quad \text{with} \quad f\left(x^{(k+1)}\right) < f\left(x^{(k)}\right) \tag{0.3.2}$$

The general descent method is shown in Algorithm 3

Algorithm 3: General descent method

Input : a starting point $x \in$ **dom** f
Output: optimal x to minimize $f(x)$

```
while Stopping criterion is not satisfied do
    Determine a descent direction Δx.
    Line search. Choose a step size t > 0.
    Update. x = x + tΔx
end while
return x
```

Gradient Descent Algorithm

In this section, we will decide the direction Δx in Algorithm3. From convexity

$$f\left(x^{(k+1)}\right) \leq f\left(x^{(k)}\right) + \nabla f\left(x^{(k)}\right)\Delta x^{(k)} \tag{0.3.3}$$

So,

$$f\left(x^{(k+1)}\right) < f\left(x^{(k)}\right) \Rightarrow \nabla f\left(x^{(k)}\right)\Delta x^{(k)} < 0 \tag{0.3.4}$$

A natural choice is gradient: $\Delta x^{(k)} = \nabla f\left(x^{(k)}\right)$.

Line Search: Backtracking

This section will find the step size t. t can be described by

$$t = \text{argmin}_t f(x + t\Delta x) \tag{0.3.5}$$

We use two parameters $\alpha \in (0, 0.5)$, $\beta \in (0, 1)$. Starting from $t = 1$, repeat $t = \beta t$ until

$$f(x + t\Delta x) < f(x) + \alpha t \nabla f(x)^{\mathsf{T}} \Delta x \tag{0.3.6}$$

See Figure 1.

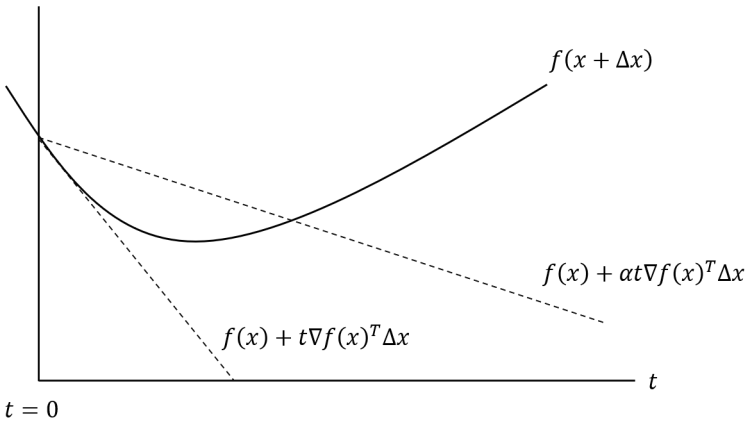

Figure 1: Backtracking

Constrained Optimization

Consider the following optimization problems that include inequality constraints,

$$\begin{array}{ll} \text{minimize} & f_0(x) \\ \text{subject to} & f_i(x) \leq 0, \quad i = 1, \cdots, m \\ & Ax = b \end{array} \tag{0.3.7}$$

We use central path. The equivalent problem of problem 0.3.7 is

$$\begin{array}{ll} \text{minimize} & tf_0(x) + \phi(x) \\ \text{subject to} & Ax = b \end{array} \tag{0.3.8}$$

which has the same minimizers. We assume problem 0.3.8 can be solved by GD and has the unique solution for any $t > 0$. We use $x^\star(t)$ to denote the solution to problem 0.3.8, and we call it central point. We define the set of $x^\star(t)$ the central path. $x^\star(t)$ should satisfy,

$$Ax^\star(t) = b. \quad f_i(x^\star(t)) < 0, \quad i = 1, 2, \cdots, mao \tag{0.3.9}$$

There exits $\hat{\nu} \in R^p$

$$0 = t\nabla f_0(x^\star(t)) + \nabla\phi(x^\star(t)) + A^T\hat{\nu} \tag{0.3.10}$$

$$= t\nabla f_0(x^\star(t)) + \sum_{i=1}^{m} \frac{1}{-f_i(x^\star(t))} + A^T\hat{\nu} \tag{0.3.11}$$

A geometric explanation is shown in Figure 2.

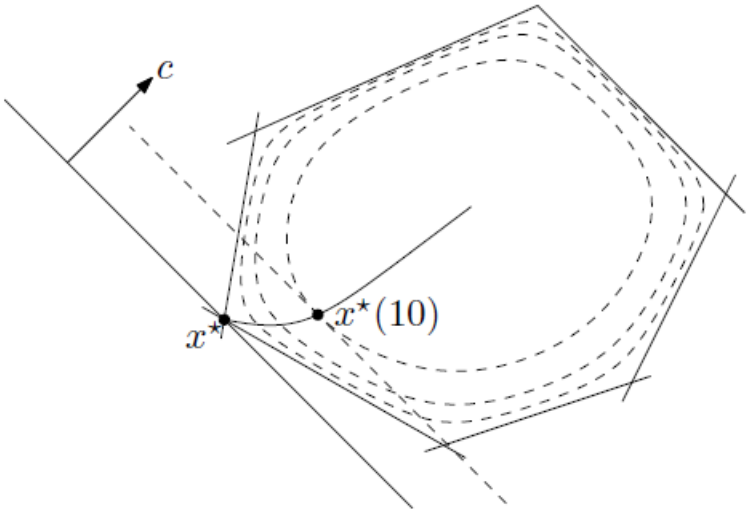

Figure 2: Central path for an LP with n = 2 and m = 6

0.4 Faster R-CNN

- *Algorithm:* Faster R-CNN
- *Input:* A RGB 3 channel picture
- *Complexity:* None
- *Data structure compatibility:* None
- *Common applications:* Artificial intelligence

Problem. Faster R-CNN

Object detection is the task of detecting instances of objects of a certain class within an image. A sample of object detection is shown in Figure 3. Faster R-CNN[**fasterrcnn**], stading for Faster Region-based Convolutional

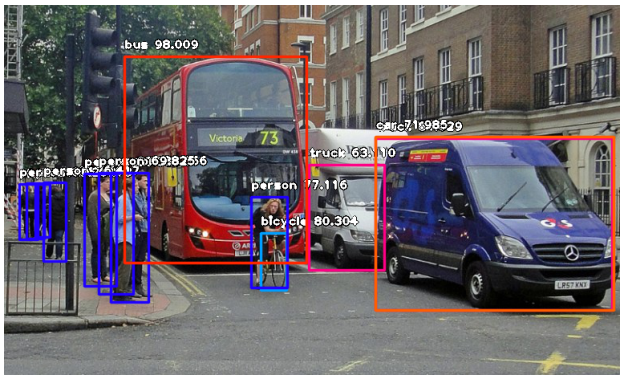

Figure 3: The result of object detection

Network, is a great milestone in the object detection. Its predecessor R-CNN[**rcnn**] first raise some region of interest by using selective search algorithm whose speed is far from practice. Faster R-CNN raises a brand new architecture to perform object detection. The sketch of its architecture is shown in Figure 4

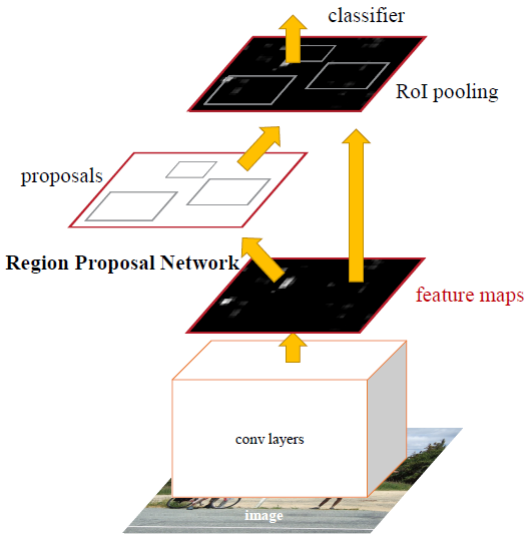

Figure 4: The structure of Fast R-CNN

Description

Conv Layers

Faster R-CNN use VGG network[**vgg**] to generate feature. It's a common backbone used in computer vission, so we don't talk about it here.

Region Proposal Network(RPN)

Talking about RPN, anchors are the most import mechanism. After we got feature maps by VGG net. We generate 9 anchors boxes shown in Figure 5. After that, it will use regression to get 4 parameters x_1, y_1, x_2, y_2

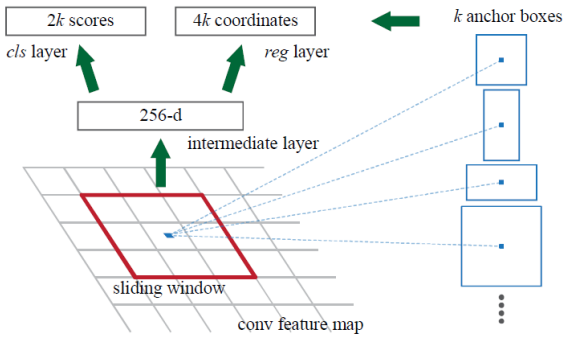

Figure 5: Region Proposal Network(RPN)

for the bounding box of the target and 1 score for whether the bounding box includes a target.

The RoI pooling layer

After we crop the image from the region proposed by RPN, the cropped image cannot be directly send to VGG net soon. Because the image size is not match with the input size of VGG. So, Faster R-CNN first project the region to $H/16 \times W/16$ which is the scale of the feature map. Then, it divided the feature map to $H_{pool} \times W_{pool}$ grid where it is the scale of our target size. At last we do max pooling.

Classification

Classification use VGG net as common. The structure is shown in Figure 6.

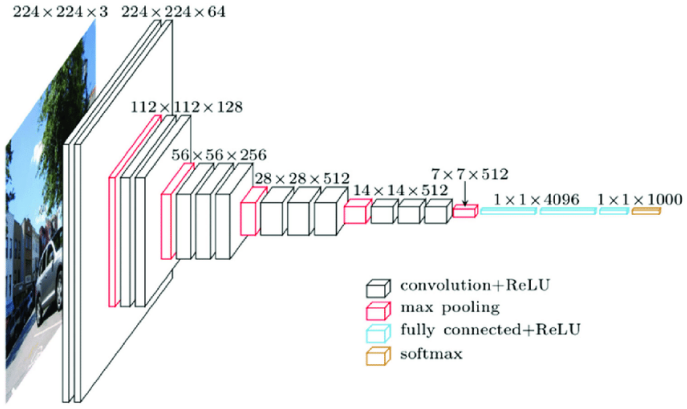

Figure 6: VGG net)

0.5 Image cropping

- *Algorithm:* Cropping (algo. 4)
- *Input:* Image to be cropped.
- *Complexity:* $\mathcal{O}(mn)$ depending on the size
- *Data structure compatibility:* Array.
- *Common applications:* Image processing.

Problem. Image cropping

Image cropping is an elementary operation on an image of removing an outer part of the image and leaving a smaller rectangular region. It can help remove the unwanted parts, improve image framing, highlight a part of the image or produce uniform datasets.

Description

Image cropping is the removal of an outer part of the image. It can be performed physically as well as digitally. It would change the basic framing, ratio and structure of the image itself. The following Figure 7 gives an example of image cropping.

The process of cropping a image is simple as explained literally. We will make this operation more mathematically accurate before discussing the algorithm. Suppose we have an bitmap image of size $a \times b$, and we hope to crop a $m \times n$-sized rectangular in the bottom-right part of the point (x, y) from it. To reach this aim, just create a new array of size $m \times n$, and assign the corresponding color to each point. You may see the process in the pseudo-code in Algorithm 4.

To assign value for each pixel in the $m \times n$ rectangular, it is easy to see that the algorithm has a time complexity of $\mathcal{O}(mn)$.

(a) Pic. 1

(b) Pic. 2

Figure 7: Group of pictures

Algorithm 4: Image Cropping, t

Input : Image I as a 2-dim array of size $a \times b$, the target part size $m \times n$ and reference point (x, y)
Output: The new cropped image T'

```
I ← array of size m × n
for i ← 1to m do
    for j ← 1to n do
        I'[i][j] ← I[x + i][y + j]
    end for
end for
return I'
```

0.6 PNG Encoding and Decoding

- *Algorithm:* Filtering (algo. 5), Huffman coding (algo. 7), LZ77 (algo. 8)
- *Input:* Origin raw image as pixelwise color data.
- *Complexity:* Not the main topic.
- *Data structure compatibility:* Array, Huffman tree
- *Common applications:* Lossless image compression.

Problem. PNG Encoding and Decoding

The portable network graphic (PNG) format is a bitmap image file format which is portable and supports lossless compressed [0]. It is developed to replace Graphics Interchange Format (GIF), but posses more features that GIF does not have [0]. To increase space efficiency, it do filtering on th original bitmap data and perform DEFLATE algorithm for compression. This allows PNGs to carry image information with much smaller size, compared to formats like GIF.

Description

The most instinctive way to store an image file is to direct save the pixel information as a large 2 or 3-dimensional array (for gray-scale and RGB images). However, this method occupies large space and is not suitable for transmission, especially through the internet. PNG brings an encoding method to significantly compress the original information. It first filters on the raster graphics and then apply an in-the-box compression algorithm called DEFALTE. We will first talk about these two aspects.

Filtering

Before introducing the principles of filtering, we will discuss instinct of coming up with filtering. For a series of data, if they are linearly correlated, i.e. the difference between consecutive elements are low, then those differences would have high probability to be lower than the original data and many of them are duplicated. This provides convenience for further compressions. For example, a transformation on an integer series is shown below:

$$a = [2, 3, 4, 6, 7, 8, 6, 5, 4, 3, 1] \Rightarrow b = [2, 1, 1, 2, 1, 1, -2, -1, -1, -2]$$

We may see that an evenly distributed array is transformed into duplicated integers like ± 1 or ± 2. This make the data more compressible. In this simple case, we calculates the difference of neighbouring elements, and such method is called Delta encoding. PNG make use of adjusted Delta encodings, which is named "filtering", where it calculates the differences of the current pixel with a predictor based on the pixel to the left, above and above-left [0]. The brief process of filtering is described in Algorithm 5. There are 5 type of filtering in PNG, which are shown in Table 1. Denote the current pixel as X, and the left, top and top-left pixel as L, T and TL.

Algorithm 5: Filtering

Input: Image lines L_1, L_2 with length n, where L_1 is right above L_2
Output: The filtered line of L_2

$L_2 = []$
$L_2'[1] = L_2[1]$
for $i \leftarrow 2$ **to** n **do**
  $L_2'[i] = L_2 - Predictor(L_2[i-1], L_1[i], L_1[i-1])$
end for
return L_2'

Type	Description	Expression
0	No filtering	X
1	With left	$X - L$
2	With top	$X - T$
3	With Avg of left and top	$X - \lfloor (L + T)/2 \rfloor$
4	Paeth predictor	$X - P(L, T, TL)$

Table 1: Types of PNG filtering

The type 4, Paeth predictor is due to Alan W. Paeth [0], which first calculate a simple linear composition of the three pixels, then choose the closest pixel to it from the 3 original ones as the predictor. A simple psuedocode illustrates it in Algorithm 6 [0].

Despite the way of demonstration here, true filtering process the data as byte instead of pixels, regardless of the bit depth or color type. By line scanning on the image, byte streams are created, where filtering algorithm will be working on.

Each filter has its best performance cases, and the encoder programs would automatically choose the best one given the image rows. Such a row-by-row choice improves the compression. This heuristic method of improving is designed by Lee Daniel Crocker, one of the member of PNG working group, who performed empirical tests on many images during the creation of PNG format [0].

Algorithm 6: PaethPredictor

```
Function PaethPredictor(X, L, T, TL):
    P = L + T − TL;
    pl = |P − L|; pt = |P − T|; ptl = |P − TL|;
    if pl is minimum then
        return L
    else if pt is minimum then
        return T
    else
        return TL
    end if
end
```

DEFLATE compression

DEFLATE is a lossless compression file format, designed by Phil Katz. Primary for personal use of a archiving tool, but was later specified as standard in RFC 1996 [0]. Technically, it's a combination of Huffman coding and LZSS compression algorithm. The former increase the space efficiency by attributing shorter coding for more frequently appeared characters. The latter achieves compression by reusing duplicated parts in the data. We are discussing these 2 algorithms in the following parts.

Huffman code is a type of prefix code used for lossless compression, while Huffman coding describes the algorithm of finding such a code, designed by David A. Huffman [0]. Prefix code is defined on a system such that no element is the prefix of another. For example, the set $\{1, 2, 12\}$ doesn't satisfy the prefix property, as 1 is the prefix of 12. This means that a Huffman code can be generated by a binary trees, called Huffman tree, and the Huffman coding contains the way to generate a Huffman tree and develop a coding system from it. It is shown in the Algorithm 7. It takes in the frequency statistics and returns the binary codes for each character.

Here offer an example of encoding the characters with frequency of

$$[a : 8,\ b : 6,\ c : 1,\ d : 3,\ e : 9]$$

which would gives a Huffman tree in Figure 8. The left path corresponds to 0 and right to 1, so we can obtain the coding table shown in Table 2. Note that no element is the prefix of another. Since if code c_1 is the prefix of code c_2, the node of c_1 should be an ancestor of that of c_2, which cannot be a leaf itself, so no character would be encoded as c_2. This guarantee the possibility of decoding.

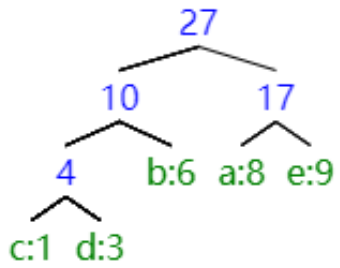

Figure 8: Example Huffman Tree

We need 3 bits originally to encode each of the 5 characters, which would take up $3 \times (8 + 6 + 1 + 3 + 9) = 81$ bits. However, with the Huffman code offerred in Table 2, the new size is $2 \times (8 + 6 + 9) + 3 \times (1 + 3) = 58$ bits, which saves $(81 - 58)/81 = 28.4\%$ space. We may see that the key of compression is to allocate shorter codes for more frequent characters.

Algorithm 7: Huffman Coding

Input : Character frequency statistics as a dictionary F
Output: Codes for each character as a dictionary C

```
Function HuffmanTree(F):
    Q ← a min-queue
    foreach character ch do
        create new node n
        n.char = ch; n.value = F[ch]
        Q.Insert(n, n.value)
    end foreach
    while Q.size > 1 do
        create a new node n
        n.left ← Q.ExtractMin()
        n.right ← Q.ExtractMin()
        n.value ← n.left.value + n.right.value
        Q.Insert(n, n.value)
    end while
    root = Q.ExtractMin()
    return root
end
Function GenerateCode(node):
    if node == root then return ""                    /* can be improved with dynamic programming */;
    else if n == n.parent.left then return GenerateCode(node.parent) + "0";
    else if n == n.parent.right then return GenerateCode(node.parent) + "1";
end
tree ← HuffmanTree(F)
foreach unique character ch do
    n ← the leaf node such n.ch == ch C[ch] = GenerateCode(n)
end foreach
return
```

char	code
a	10
b	01
c	000
d	001
e	11

Table 2: Example Huffman Coding

Before LZSS, we would briefly illustrate its predecessor, LZ77, which is a lossless compression algorithm named after the author A. Lempel and J. Ziv as well as the publication year 1977 [0]. It maintains a dictionary of a string buffer, and compresses the string data by representing the sub-strings as references to that in the dictionary. This dictionary slides as encoding and decoding, and is thus called a *sliding window*. The references is encoded as a length-distance pair (l, d), which means the following string of length l is exactly the same as the characters d distance behind in the uncompressed stream.

However, in this design, the authors suggests all strings be encoded in the dictionary, even though it has no matches in the dictionary. A reference to 1 or 2 characters occupies more bits than the original string, and this might end up in cases where a processed data is even longer than the original one [0]. To solve this, in 1982, James A. Storer and Thomas Szymanski proposed omitting such cases for better compression, and that scheme is named as LZSS [0]. Besides, to indicate the difference between literals and dictionary references, LZSS also introduces a 1-bit flag to label the following data chunks. The simplified scheme is shown in Algorithm 8 [0].

Algorithm 8: LZSS

Input : Unencoded input
Output: Encoded output

```
Initialize dictionary D                                   /* Already gotten a dictionary anyway */
output ← empty
string ← part of uncoded string of maximum match length allowed
while input is not entirely encoded do
    search in string for longest matching string in D
    if match found m And m longer than reference then
        output.add encoded flag True
        output.add distance and length (d, l)
    else
        output.add encoded flag False
        output.add first uncoded symbol string[1]
    end if
    Copy symbols newly written encoded output to D
    Shift string by length of newly written encoded output
end while
```

Decoding

The process of decoding is simply a reversion of the encoding, but to completely read a PNG file, we would need specifically designed decoders. They should first read the image head and critical data chunks which indicate the encoding details. A PNG file starts with an 8-byte signature `IHDR` as the Table 3 specifies [0].

Width	4 bytes	Height	4 bytes
Bit Depth	1 byte	Color Type	1 bytes
Compression method	1 byte	Filter method	1 byte
Interlace method	1 byte		

Table 3: PNG Image Head `IHDR`

After the header there comes a series of data chunks. A typical data chunk is made of 4 parts: 4-byte *length* the chunk length, 4-byte chunk type/name, *length*-byte chunk data and 4-byte cyclic redundancy code/checksum (CRC) for data authentication [0]. Chunks are divided in to critical ones and ancillary ones. The critical chunks store the major information of the image for decoding, including `IHDR` the file head, `PLTE` the palette/color list, `IDAT` the image (might be split among multiple chunks) and `IEND` all-0 bytes as the image end. On the other hand, the ancillary chunks are not necessary and can be skipped if the decoder cannot understand. It involves other image information like transparency, color space, histogram, text information, time stamp and so on.

After the decoder understand the detailed ways of filtering and compression, it is only a reverse of encoding to decode the PNG data. The decoding of LZSS result is easy by using the same dictionary and decide whether to transform the string by checking the encoded flag. If encoded flag is `True`, then look up the dictionary and copy the string referred to to the current position. The Huffman encoded bit-string can also be directly recovered to the original string by looking up the codes. While filtering is just a linear combinations of the current and neighbouring pixels, and can be restored by re-calculating the bytes from the last scan line back to the first.

Characters of PNG

One of the main advantages of PNG format is the minimum compression loss and the portability. Even if the size of the image is made to be small, the image quality is not damaged. Another major compressed image file format JPEG can also achieve good compression effects, but as a lossy compression method, some of the information would be definitely lost. And compared with its ascender GIF, PNG shows better compression ability in most cases.

Another point is that PNG support more color as well as transparency. As in GIF and JPEG only 8-bit indexed color are allowed, while PNG offers more choices, including 8 or 16-bit per channel true color. The alpha channel in PNG also makes transparent images possible.

PNG also supports addition of meta-file, that is, to add whatever text information you like to the file. It is a common practice in digital cameras and smart phones to add time stamp, location, shooting parameters, author and even copyrights. This function would be useful in some cases.

References.

[0] Thomas Boutell and T Lane. "Png (portable network graphics) specification version 1.0". In: *Network Working Group* (1997), pp. 1–102 (cit. on p. 10).

[0] Lee Daniel Crocker. "PNG: The portable network graphic format". In: *Dr Dobb's Journal-Software Tools for the Professional Programmer* 20.7 (1995), pp. 36–45 (cit. on p. 11).

[0] Peter Deutsch. *RFC1951: DEFLATE compressed data format specification version 1.3*. 1996 (cit. on p. 12).

[0] Michael Dipperstein. *LZSS (LZ77) Discussion and Implementation*. Mar. 2015 (cit. on p. 13).

[0] David A Huffman. "A method for the construction of minimum-redundancy codes". In: *Proceedings of the IRE* 40.9 (1952), pp. 1098–1101 (cit. on p. 12).

[0] Colt McAnlis. *How PNG Works*. medium.com. Apr. 2016. URL: https://medium.com/@duhroach/how-png-works-f1174e3cc7b7 (cit. on p. 11).

[0] Alan W. Paeth. *Image file compression made easy*. Dec. 1991 (cit. on p. 11).

[0] James A Storer and Thomas G Szymanski. "Data compression via textual substitution". In: *Journal of the ACM (JACM)* 29.4 (1982), pp. 928–951 (cit. on p. 13).

[0] W3C. *Portable Network Graphics (PNG) Specification (Second Edition)*. Nov. 2003. URL: https://www.w3.org/TR/PNG/ (cit. on pp. 11, 14).

[0] Jacob Ziv and Abraham Lempel. "A universal algorithm for sequential data compression". In: *IEEE Transactions on information theory* 23.3 (1977), pp. 337–343 (cit. on p. 13).