一、Hashing

* perfect hash function → 所有 hash value 都不相等 (No collision)

* good
  1. efficient
  2. 分散平均
  3. minimum collision

* Handle collision
  - seperate chaining → linked list
  - open addressing →
    1. linear probing
    2. Quadratic probing

· linear probing → 先到 home bucket，若滿，往下找第一個空的

· quadratic probing → 先到 home bucket，若滿往下找第 n^2 個 ex. $1^2, 2^2, 3^2 \ldots$

* Double hashing → $h(k,i) = (h_1(k) + i \cdot h_2(k)) \% n$，i 從 1 開始

二、binary tree

* full binary tree：每個 node 都有 0 / 2 個 child

* perfect binary tree：所有的 internal node 都有 2 children，每個 level 都填滿

* complete binary tree：除最後 level，其他 level 都填滿，最後 level 所有 node 都集中在左邊

* Theorem 3.1 (complete binary tree)
① level i 最多 2^i 個 node，$i \geq 0$
→ 若為 perfect binary tree，node 數就是 2^i 個
② 高度 h 最多 $2^{h+1}-1$ 個 node，$h \geq 0$

3.2
③ 已知有 n node → 高度多少：$\lfloor \log_2 n \rfloor$

proof 3.1 ② $1 + 2 + 2^2 + 2^3 + \cdots + 2^{h-1} + 2^h = 2^{h+1} - 1$

proof 3.2.
from Theorem 3.1(2)，$n < 2^{h+1}$
from Theorem 3.1 and definition for full binary tree，$2^h \leq n$
$\because 2^h \leq n < 2^{h+1}$，$\therefore h \leq \log_2 n < h+1$
故 $h = \lfloor \log_2 n \rfloor$.

三、Heap

* A heap is a array 且可以被視為一個 complete binary tree

* max heap：上到下，大到小

  min heap：上到下，小到大

* Insert
  ① 加在最下方的空位，往上做 max heap

* Extract
  ① 刪 root 的值，把最後的 node 值放到 root，
  接著再做 max heap

* build a max heap
  · 最後 level 的 internal node 輪流做 down heap
  再往上以此類推

四、binary search tree（BST）

* 左邊 subtree 都比右邊 subtree 小

* Node* find (const int val) → Iterative search

```
{
  Node* try = root;
  while( try != nullptr)
  {
    if ( val < try → myval)
      try = try → left;
    else if ( val > try → myval)
      try = try → right;
    else
      return try;
  }
  return nullptr;
}
```

* Node* find (const int val) → Recursive search

```
{ return recursiveTreesearch ( root, val); }
Node* recursiveTreesearch ( Node* tryNode, const int val)
{
  if ( tryNode == nullptr)
    return nullptr;
  if ( val < tryNode → val)
    return recursive.... ( tryNode → left, val)
  else if ( val > tryNode → val)
    return recursive.... ( tryNode → right, val)
  else
    return tryNode;
}
```

Insertion

Deletion

tree traversal

- DFS. recursive

① inorder:

```
if (node != nullptr)
{
    inorder(node->left);
    cout << node->data;
    inorder(node->right);
}
```

② preorder.

```
if (node != nullptr)
{
    cout << node->data;
    preorder(node->left);
    preorder(node->right);
}
```

③ postorder:

```
if (   "   )
{
    postorder(node->left);
       "     (node->right);
    cout << node->data;
}
```

五. red-black tree.

- iterative traversal

① IterPreorder

```
{
    stack<Node*> s;
    s.push(node);
    while(!s.empty())
    {
        node = s.top();
        s.pop();
        cout << node->data;
        if (node->right != nullptr)
            s.push(node->right);
        if (node->left != nullptr)
            s.push(node->left);
    }
}
```

- BFS → 一個 Level 一個 Level 跑

六 Asymptotic notation

* Asymptotic Upper Bound : $O(\ )$

- $g: Z^+ \to R^+$ be a function

  $O(g) = \{ f: Z^+ \to R^+ \mid \exists c \in R^+, n_0 \in Z^+,$
  使得 $\forall n \geq n_0, f(n) \leq cg(n) \}$

  $\to f = O(g)$ 表 f is a member of $O(g)$

  故 $f = O(g)$ if and only if $\exists c \in R^+, n_0 \in Z^+$
  s.t. $\forall n \geq n_0, f(n) \leq cg(n)$

ex. $10n^2 - 6n + 4 = O(n^2)$

$\because 10n^2 - 6n + 4 \leq 11n^2, \forall n \geq 4$

$(-6n + 4 \leq n^2, \quad -6n + 4 \leq 4 \leq n^2, \quad n \geq 4)$

proof:

① $f = O(g)$ if and only if $\exists c \in R^+, n_0 \in Z^+$ s.t. $\forall n \geq n_0, f(n) \leq cg(n)$

② $f \neq O(g)$ if and only if $\forall c \in R^+, n_0 \in Z^+, \exists n_0 \geq n_0$ s.t. $f(n) > cg(n)$

③ $\forall c \in R^+, n_0 \in Z^+$, 設 $n = \max\{\lceil c \rceil, n_0\}$, 則 $6n + 7 > n \geq c = c \cdot 1$

$\therefore 6n + 7 \neq O(1)$

④ $\forall c \in R^+, n_0 \in Z^+$, 設 $n = \max\{\lceil c \rceil, n_0\}$, 則 $8n^2 + 2n + 3 > n^2 \geq cn$

$\therefore 8n^2 + 2n + 3 \neq O(n)$

* Theorem 1.1 If $f(n) = a_k n^k + \cdots + a_1 n + a_0$, 則 $f(n) = O(n^k)$

$\to$ proof:

$f(n) \leq |a_k| n^k + |a_{k-1}| n^{k-1} + \cdots + |a_1| n + |a_0|$

$\leq |a_k| n^k + \cdots + |a_1| n^k + |a_0|$

$= (|a_k| + \cdots + |a_1| + |a_0|) n^k$, for $n \geq 1$.

$\therefore f(n) = O(n^k)$ #

* Asymptotic Lower Bound.

- $\Omega(g) = \{ f: \mathbb{Z}^+ \to \mathbb{R}^+ \mid \exists c \in \mathbb{R}^+, n_0 \in \mathbb{Z}^+$
  $\text{s.t. } \forall n \geq n_0,\ f(n) \geq c\,g(n) \}$

  $\to f = \Omega(g)$ if and only if $\exists c \in \mathbb{R}^+, n_0 \in \mathbb{Z}^+$
  s.t. $\forall n \geq n_0,\ f(n) \geq c\,g(n)$

ex. $3n^2 - 6n + 4 = \Omega(n^2) \because 3n^2 - 6n + 4 \geq 2n^2$ for $n \geq 6$.
$3n^2 - 6n + 4 = \Omega(n) \because 3n^2 - 6n + 4 \geq 2n$ for $n \geq 6$
$3n^2 - 6n + 4 = \Omega(1) \because 3n^2 - 6n + 4 \geq 2$ for $n \geq 6$.

- proof:

① $f = \Omega(g)$ if and only if $\exists c \in \mathbb{R}^+, n_0 \in \mathbb{Z}^+$ s.t. $\forall n \geq n_0, f(n) \geq c\,g(n)$

② $f \neq \Omega(g)$ if and only if $\forall c \in \mathbb{R}^+, n_0 \in \mathbb{Z}^+, \exists n \geq n_0$ s.t. $f(n) < c\,g(n)$

③ $\forall c \in \mathbb{R}^+, n_0 \in \mathbb{Z}^+$, 設 $n = \max\{\lceil \frac{10}{c} \rceil, n_0\}$
則 $\frac{10}{c} \leq n$，因此 $10 \leq cn$，$2n + 7 < 10n \leq cn^2$
故 $2n + 7 \neq \Omega(n^2)$

- Theorem 1.2. If $f(n) = a_k n^k + \cdots + a_1 n + a_0$ and $a_k > 0$, then $f(n) = \Omega(n^k)$

proof
$$\begin{aligned} f(n) &\geq a_k n^k - |a_{k-1}| n^{k-1} - \cdots - |a_1| n - |a_0| \\ &\geq a_k n^k - |a_{k-1}| n^{k-1} - \cdots - |a_1| n^{k-1} - |a_0| n^{k-1} \\ &\geq a_k n^k - (|a_{k-1}| + \cdots + |a_1| + |a_0|) n^{k-1} \\ &\geq \frac{a_k}{2} n^k, \text{ for } n \geq 2(|a_{k-1}| + \cdots + |a_1| + |a_0|)/a_k. \end{aligned}$$
$\therefore f(n) = \Omega(n^k)$

* $n \geq 2(|a_{k-1}| + \cdots + |a_1| + |a_0|)/a_k$.

proof

$\frac{1}{2} a_k n \geq (|a_{k-1}| + \cdots + |a_1| + |a_0|)$

$\frac{1}{2} a_k n^k \geq (|a_{k-1}| + \cdots + |a_1| + |a_0|) n^{k-1}$

$a_k n^k - \frac{1}{2} a_k n^k \geq$ 〃

湊結果

$a_k n^k - (|a_{k-1}| + \cdots + |a_1| + |a_0|) n^{k-1} \geq \frac{1}{2} a_k n^k$

Asymptotic Tight Bound

- $f = \Theta(g)$ if and only if $\exists c \in R^+, n_0 \in Z^+$
  s.t. $\forall n \geq n_0$, $c_1 g(n) \leq f(n) \leq c_2 g(n)$

ex. $10n^2 - 6n + 4 = \Theta(n^2)$
$10n^2 - 6n + 4 \neq \Theta(n)$
$10n^2 - 6n + 4 \neq \Theta(1)$

- Theorem 1.3 : $f = \Theta(g)$ if and only if $f = O(g)$ and $f = \Omega(g)$

1. only if part) trivial
2. if part)
   $\because f = O(g)$ $\therefore \exists c_1 \in R^+, n_1 \in Z^+$ s.t. $\forall n \geq n_1$, $f(n) \leq c_1 g(n)$
   $\because f = \Omega(g)$ $\therefore \exists c_2 \in R^+, n_2 \in Z^+$ s.t. $\forall n \geq n_2$, $f(n) \geq c_2 g(n)$
   設 $n_0 = \max\{n_1, n_2\}$
   則 $f(n) \leq c_1 g(n)$ 且 $f(n) \geq c_2 g(n)$, $\forall n \geq n_0$.
   故 $f = \Theta(g)$

- Theorem 1.4 : If $f(n) = a_k n^k + \dots + a_1 n + a_0$ 且 $a_k > 0$ 則 $f(n) = \Theta(n^k)$

  By theorem 1-1 $f(n) = O(n^k)$
  theorem 1.2 $f(n) = \Omega(n^k)$
  theorem 1.3 $f(n) = \Theta(n^k)$

String searching Algorithm. (pattern matching.)

The KMP. Algorithm.

```
void kmp.( int* failure)
{
  bool found = false;
  size_t posS = 0, posP = 0;
  while( posS < strlen(str))
  {
    if ( pat[posP] == str[posS])
        ++posS, ++posP;
    if ( posP == strlen(pat))
    {
      found = true;
      cout << "found pattern at index" << posS - posP << endl;
      posP = failure[posP - 1];
    }
    else if ( pat[posP] != str[posS])
        if ( posP = 0)  ++posS;
        else   posP = failure[posP-1];

  }
  if ( !found) cout << "Not found\n";
}
```

```
void compfailure( int* failure)
{
  size_t i = 0;
  failure[0] = 0;
  size_t j = 1;
  while ( j < strlen(pat))
    if ( pat[j] == pat[i])
    {
      ++i;
      failure[j] = i;
      ++j;
    }
    else
    {
      if ( i == 0)
      {
        failure[j] = 0;
        ++j;
      }
      else
        i = failure[i-1];
    }
}
```

◎ time complexity of kmp is $O(strlen(str))$

proof

設 $n = strlen(str)$

∵ posS 不會減少，可得 ++posS 不執行超過 n 次

並且每次執行 ++posP 時 ++posS 也會執行，故 ++posP 也不會執行超過 n 次

根據 failure function 的定義

① $failure[posP-1] \le posP-1 < posP$，

可得 $posP = failure[posP-1]$ 使得 posP 的值減少

② $failure[posP-1] \ge 0$，

可得 posP 不可能為負

∵ posP 初始值為 0，故 $posP = failure[posP-1]$ 的執行次數不會超過 ++posP 的執行次數

∴ $posP = failure[posP-1]$ 不會執行超過 n 次

因此 the time complexity of kmp 是 $O(n) = O(strlen(str))$

◎ the time complexity of compFailure is $O(strlen(pat))$

proof

設 $m = strlen(pat)$

∵ j 值不會減少，可得 ++j 不執行超過 m 次

並且每次 ++i 執行時 ++j 也會執行，故 ++i 執行次數也不超過 m 次

根據 failure function 的定義

① $failure[i-1] \le i-1 < i$，

可得 $i = failure[i-1]$ 會使 i 的值減少

② $failure[i-1] \ge 0$

可得 i 不可能為負

故 $i = failure[i-1]$ 執行的次數不超過 ++i 執行次

可得 $i = failure[i-1]$ 不會執行超過 m 次

因此 the time complexity of compFailure is $O(m) = O(strlen(pat))$