ເອກະສານ

ວິຊາ ໂຄງສ້າງຖານຂໍ້ມູນ ແລະ ຂັ້ນຕອນວິທີ 2

ກຸ່ມ I

ສອນໂດຍ: ອ.ຈ ຈິດນາວັນ

ผู้สົมຮູ້ຮ່ວມຄິດໃນຄັ້ງນີ້

ນ.ແຕັກ

ທ.ເອກພະໄຊ

ທ.ເຊັ່ງວ່າງ

ທ.ຣືວ່າງ

ທ.ຫຼ້າເບກແກ້ວ

ທ.ອາລຸນໄຊ

ນ.ໜົ່ມ

ທ.ວົງວິຈິດ

ຄຳນຳ

ໃນບົດນີ້ເປັນພຽງແຕ່ວຽກບ້ານໜຶ່ງໃນວິຊາ ໂຄງສ້າງຖານຂໍ້ມູນ ແລະ ຂັ້ນຕອນວິທີ ສຳລັບການພັດທະນາເວັບໄຊ້ 2 ເທົ່ານັ້ນ, ເຊິ່ງແມ່ນ ອ.ຈ ຈິດນາວັນ ເປັນຜູ້ມອບໝາຍໃຫ້ ແລະ ຂ້າພະເຈົ້າ ທ.ເອກພະໄຊ ເພົ້າລັດສະໝີ ຕາງໜ້າກຸ່ມທີ I ຈາກຫ້ອງ **2CW1** ໄດ້ພາກັນຊອກຫາຂໍ້ມູນເພື່ອຈະນຳໃຊ້ເຂົ້າໃນເອກະສານສະບັບນີ້ ບໍ່ວ່າຈະເປັນຈາກແຫຼ່ງຕ່າງໆ, ເນື້ອໃນທັງໝົດໃນເອກະສານສະບັບນີ້ເປັນພຽງແຕ່ຄວາມເຂົ້າໃຈ ແລະ ຄວາມຮູ້ຂອງພວກເຮົາເທົ່ານັ້ນ ເຊິ່ງພວກເຮົາ ກຸ່ມທີ I ບໍ່ໄດ້ມີເຈດຕະນາທີ່ຈະປ່ຽນແປງ ຫຼື ບິດເບືອນ ຄວາມຖືກຕ້ອງຂອງບົດຮຽນ ດັ່ງນັ້ນ, ພວກເຮົາຫວັງວ່າ ອາຈານ ແລະ ໝູ່ເພື່ອນທຸກຄົນທີ່ໄດ້ອ່ານເອກະສານສະບັບນີ້ຈະພິຈາລະນາ, ກວດສອບ ແລະ ແນະນຳໃນສິ່ງທີ່ພວກເຮົາຜິດພາດ ຫຼື ຂາດຫາຍໄປໃນເອກະສານສະບັບນີ້ດ້ວຍ.

ຮຽນມາດ້ວຍຄວາມເຄົາລົບທ່ານຜູ້ອ່ານທຸກຄົນ

ກຸ່ມທີ I

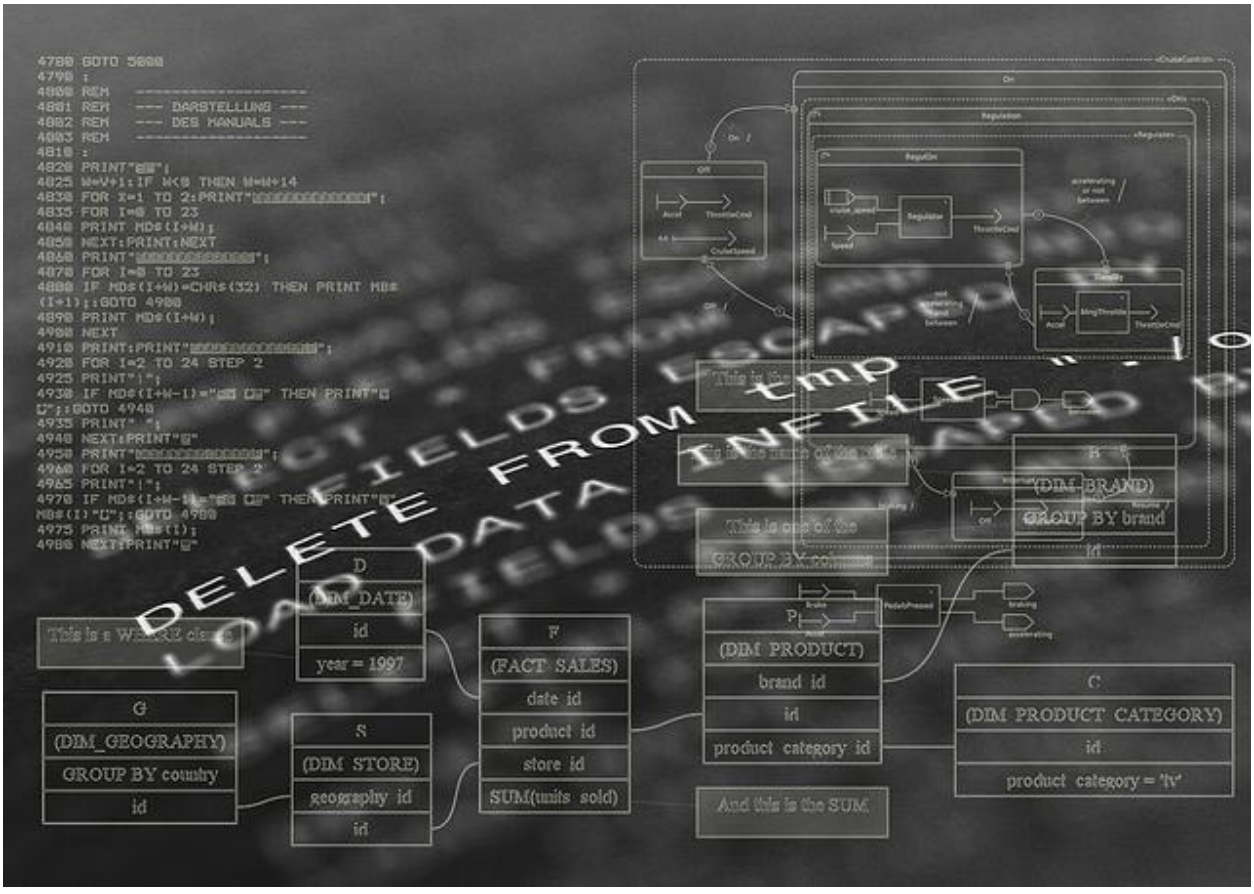
<u>ວິຊາ: ໂຄງສ້າງຖານຂໍ້ມູນ ແລະ ຂັ້ນຕອນວິທີ</u>

ບົດທີ I

ໂຄງສ້າງເບື້ອງຕົ້ນກ່ຽວກັບໂຄງສ້າງຂໍ້ມູນ ແລະ ຂັ້ນຕອນວິທີ

I. <u>ຄວາມຮູ້ພື້ນຖານກ່ຽວກັບໂຄງສ້າງຂໍ້ມູນ.</u>

ໂຄງສ້າງຂໍ້ມູນ ແມ່ນ ຄວາມສຳພັນລະຫວ່າງຂໍ້ມູນທີ່ຢູ່ໃນໂຄງສ້າງນັ້ນໆ ລວມທັງຂະບວນການໃນການຈັດການຂໍ້ມູນ ເຊັ່ນ: ເພີ່ມ, ແກ້ໄຂ, ລົບ,...



ຕົວຢ່າງໂຄງຂໍ້ມູນປະເພດຕ່າງໆໄດ້ແກ່:

1. Array: ແມ່ນປະເພດຂອງຂໍ້ມູນທີ່ສາມາດເກັບຂໍ້ມູນປະເພດດຽວກັນແບບເປັນລຳດັບໄດ້ ໂດຍຂໍ້ມູນນັ້ນຈະຢູ່ໃນໂຕແປ ທີ່ເອີ້ນວ່າ Array ຂໍ້ມູນແຕ່ລະໂຕຂອງ ອາເລ່ນັ້ນຈະເອີ້ນວ່າ Element ແລະ ຂໍ້ມູນແຕ່ລະ Element ຈະມີໝາຍເລກເພື່ອໃຊ້ໃນການອ້າງອີງ ຈຶ່ງເອີ້ນຕົວເລກນີ້ວ່າ Index.

   ການປະກາດ Array

```
type[] name;
type[] name = new type[size];
type[] name = new type[] {value1, value2, ...};
```

2. String: ແມ່ນປະເພດຂໍ້ມູນປະເພດຂໍ້ຄວາມ ຫຼື ການເອົາຕົວອັກສອນຫຼາຍໆອັກສອນມາຕໍ່ກັນ ຫຼື ເອີ້ນວ່າ Array ຂອງຕົວອັກສອນ ໂດຍຄວາມຍາວຂອງມັນສາມາດປ່ຽນແປງໄດ້ຕາມຄ່າທີ່ກຳໜົດໃຫ້ກັບໂຕແປ.

   ການປະກາດໂຕແປ String

```
$str1 = "This is string declaration with double quote.";
$str2 = 'This is string declaration with single quote.';
```

3. Stack: ແມ່ນຢູ່ໃນພື້ນຖານຂອງຫຼັກການແບບ **Last In First Out**(LIFO) ໝາຍຄວາມວ່າສິ່ງທີ່ເພີ່ມເຂົ້າມາໃໝ່ສຸດຈະຖືກລົບອອກກ່ອນ.

   ຄຳສັ່ງພື້ນຖານຂອງ Stack

   - push()- ເປັນການເພີ່ມ Element ລົງໄປໃນສ່ວນເທິງສຸດຂອງ Stack
   - pop()- ເປັນການລົບ Element ທີ່ຢູ່ສ່ວນເທິງສຸດຂອງ Stack
   - peek()- ເປັນການເບິ່ງ Element ທີ່ຢູ່ສ່ວນເທິງສຸດຂອງ Stack
   - isFull()- ເປັນການກວດສອບວ່າ Stack ເຕັມ ຫຼື ບໍ່
   - isEmpty()- ເປັນການກວດສອບວ່າ Stack ຫວ່າງ ຫຼື ບໍ່

4. Queue: ເປັນ Linear Data Structure ເຊິ່ງຈະຄ້າຍຄືກັນກັບ Stack, Queue ແມ່ນຢູ່ໃນພື້ນຖານຂອງຫຼັກການແບບ **First In First Out**(FIFO) ໝາຍຄວາມວ່າ ສິ່ງໃດທີ່ເຂົ້າກ່ອນຈະໄດ້ອອກກ່ອນ ແລະ ສິ່ງໃດທີ່ເຂົ້າມາກ່ອນກໍຈະໄດ້ອອກກ່ອນ.

ຄຳສັ່ງພື້ນຖານຂອງ Queue

- Enqueue()- ເປັນການເພີ່ມຂໍ້ມູນລົງໄປໃນ Queue
- Dequeue()- ເປັນການລົບຂໍ້ມູນລົງໄປໃນ Queue
- Peek()- ເປັນການເບິ່ງຂໍ້ມູນທີ່ຢູ່ຕຳແໜ່ງ Front ຂອງ Queue
- isFull()- ເປັນການກວດສອບວ່າ Queue ເຕັມ ຫຼື ບໍ່
- isEmpty()- ເປັນການກວດສອບວ່າ Queue ຫວ່າງ ຫຼື ບໍ່

II. ປະເພດຂອງໂຄງສ້າງຂໍ້ມູນ.

ປະເພດຂອງໂຄງສ້າງຂໍ້ມູນທີ່ນຳໃຊ້ໃນປັດຈຸບັນແບ່ງອອກເປັນ 2 ປະເພດຄື:

1. ໂຄງສ້າງຂໍ້ມູນທາງກາຍະພາບ.

ເປັນໂຄງສ້າງຂໍ້ມູນທີ່ນຳໃຊ້ທົ່ວໄປໃນພາສາຄອມພິວເຕີ ເຊິ່ງແບ່ງອອກເປັນ 2 ປະເພດຄື:

1.1 ຂໍ້ມູນເບື້ອງຕົ້ນເຊັ່ນ: Int, Float, Character

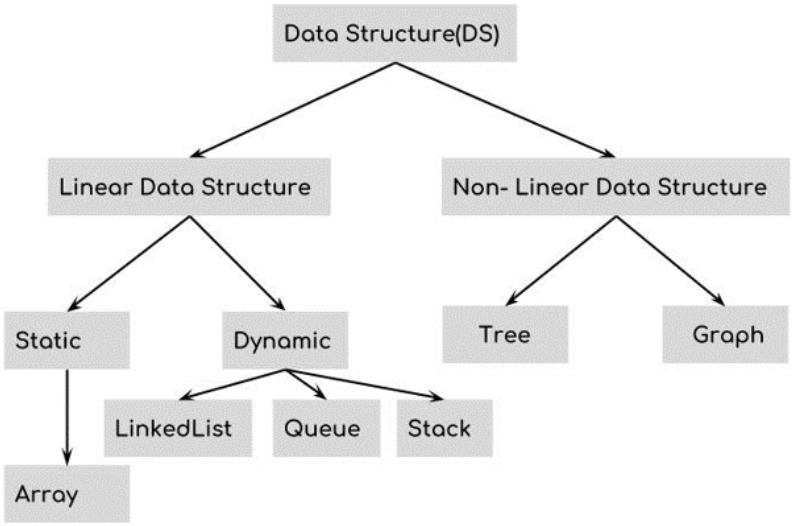
- Int: ແມ່ນຊະນິດຂໍ້ມູນທີ່ເປັນຕົວເລກຖ້ວນ
- Float: ແມ່ນຊະນິດຂໍ້ມູນທີ່ເປັນຕົວເລກຈຳນວນຈິງ ຫຼື ຕົວເລກທີ່ມີຈຸດ
- Char: ແມ່ນຊະນິດຂໍ້ມູນທີ່ເປັນຕົວອັກສອນ

1.2 ຂໍ້ມູນໂຄງສ້າງເຊັ່ນ: Field, Record, File

- Field: ໝາຍເຖິງການນຳເອົາຂໍ້ມູນທີ່ສຳຄັນມາໄວ້ໃນອັນດຽວ
- Record: ໝາຍເຖິງການນຳເອົາຫຼາຍໆ Field ມາຮວມເຂົ້າກັນ
- File: ໝາຍເຖິງການນຳເອົາ Record ຫຼາຍໆ Record ທີ່ກ່ຽວຂ້ອງກັນໃນດ້ານໃດດ້ານໜຶ່ງມາລວມເຂົ້າກັນ

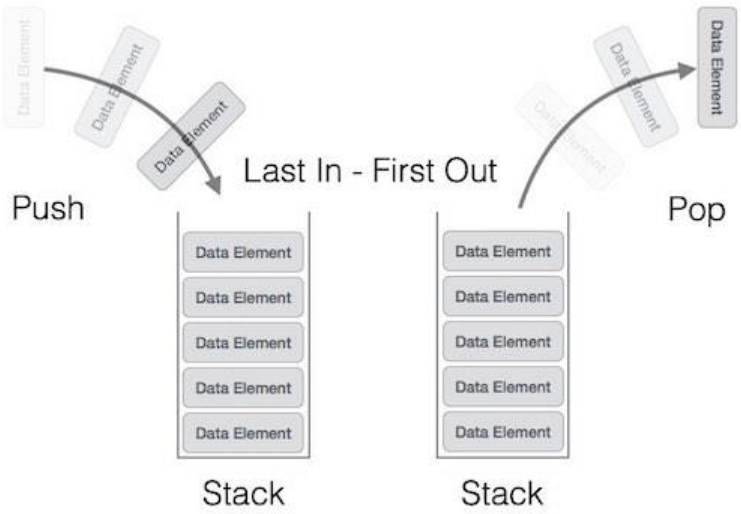
2. ໂຄງສ້າງຂໍ້ມູນທາງຕັກກະ.

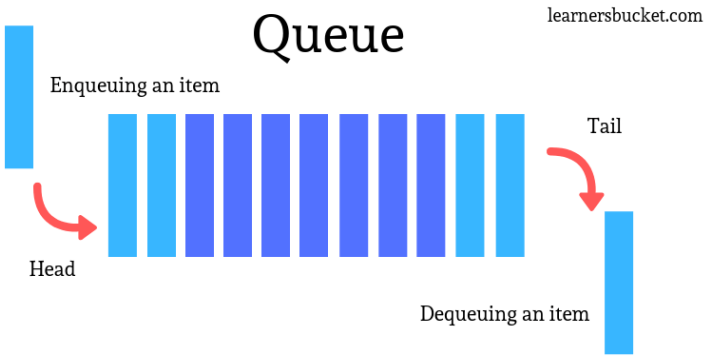
ເປັນໂຄງສ້າງທີ່ເກີດຈາກຈິນຕະນາການຂອງຜູ້ໃຊ້ ເພື່ອນຳໃຊ້ໃນການແກ້ໄຂບັນຫາໃນໂປຣແກຣມທີ່ສ້າງຂຶ້ນ ແບ່ງອອກເປັນ 2 ປະເພດຄື:



2.1 Linear Data Structure

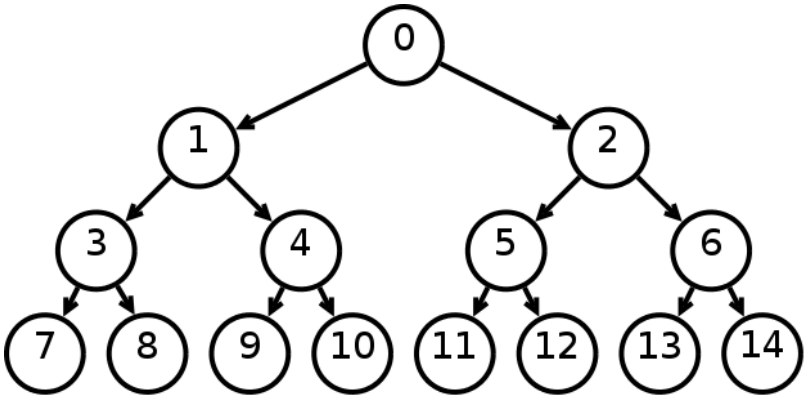
ເປັນໂຄງສ້າງຂໍ້ມູນທີ່ຄວາມສຳພັນຂອງຂໍ້ມູນຈະລຽງຕໍ່ກັນ ເຊັ່ນ: List, Stack, Queue, String ເປັນຕົ້ນ.





2.2 Non-Linear Data Structure

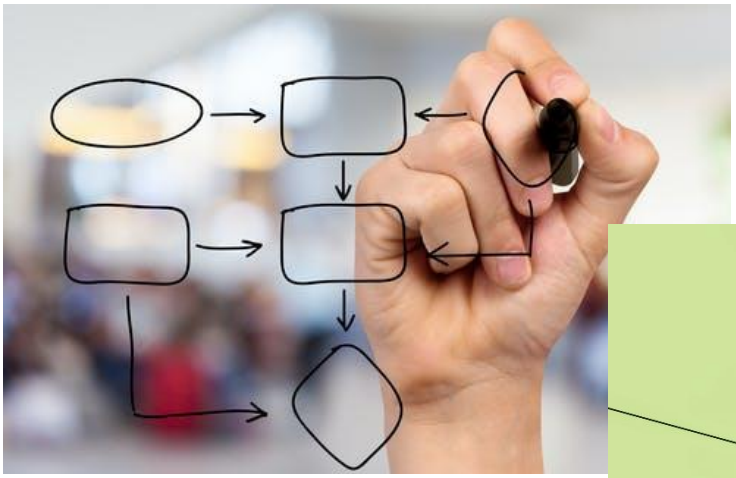
ເປັນໂຄງສ້າງຂໍ້ມູນທີ່ຂໍ້ມູນແຕ່ລະຕົວສາມາດມີຄວາມສຳພັນກັບຂໍ້ມູນອື່ນໄດ້ຫຼາຍຕົວ ເຊັ່ນ: Tree, Graph

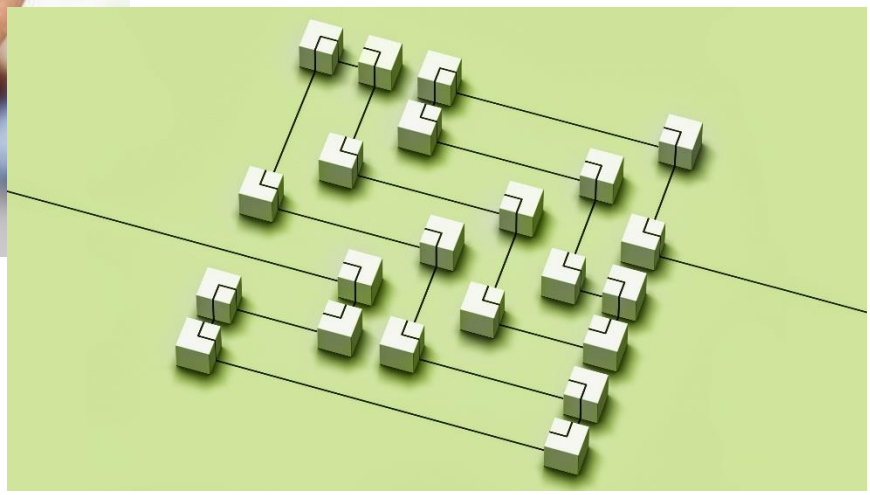


III. ການເລືອກໃຊ້ໂຄງສ້າງຂໍ້ມູນ.

1) ໂຄງສ້າງຂໍ້ມູນນັ້ນຈະຕ້ອງສ້າງຄວາມສຳພັນໃຫ້ກັບຂໍ້ມູນຊຸດນັ້ນໆ**ໄດ້ຢ່າງສົມບູນທີ່ສຸດ**

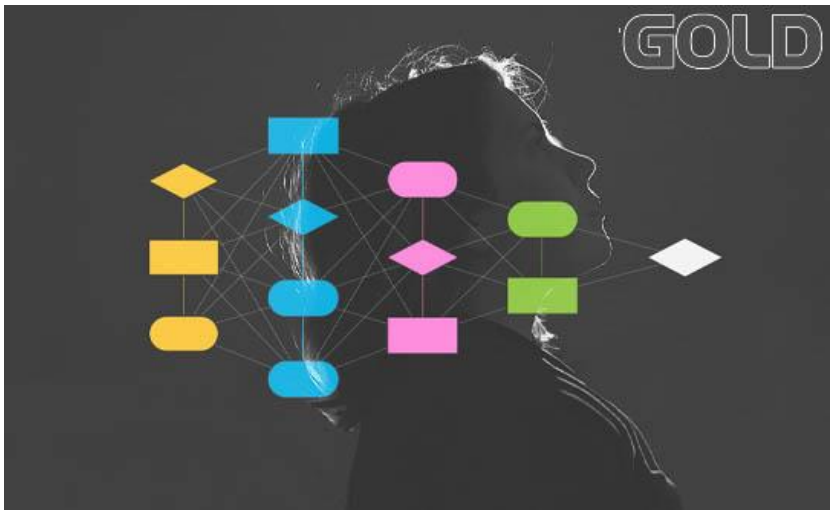
2) ໂຄງສ້າງຂໍ້ມູນນັ້ນຕ້ອງ**ງ່າຍຕໍ່ການດຳເນີນການ**ໃນຂະບວນການ





IV. ຂັ້ນຕອນວິທີ (Algorithm).

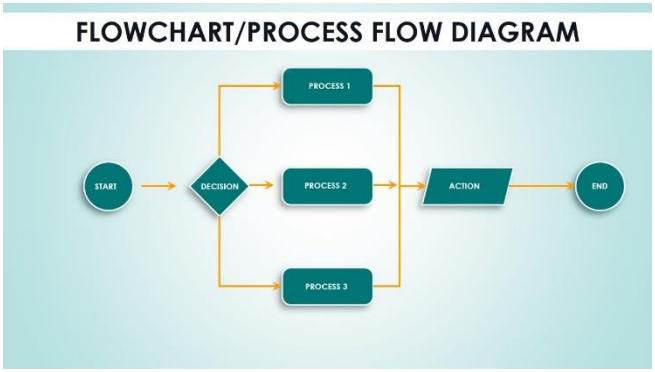
ແມ່ນວິທີການແກ້ບັນຫາຕ່າງໆຢ່າງມີລະບົບ ມີລຳດັບຂັ້ນຕອນ ຕັ້ງແຕ່ຕົ້ນຈົນຈົບ, ສາມາດຂຽນໄດ້ຫຼາຍແບບ, ການເລືອກໃຊ້ຕ້ອງເລືອກໃຊ້ຂັ້ນຕອນທີ່ເໝາະສົມ, ກະທັດຮັດ ແລະ ມີປະສິດທິພາບທີ່ສຸດ.

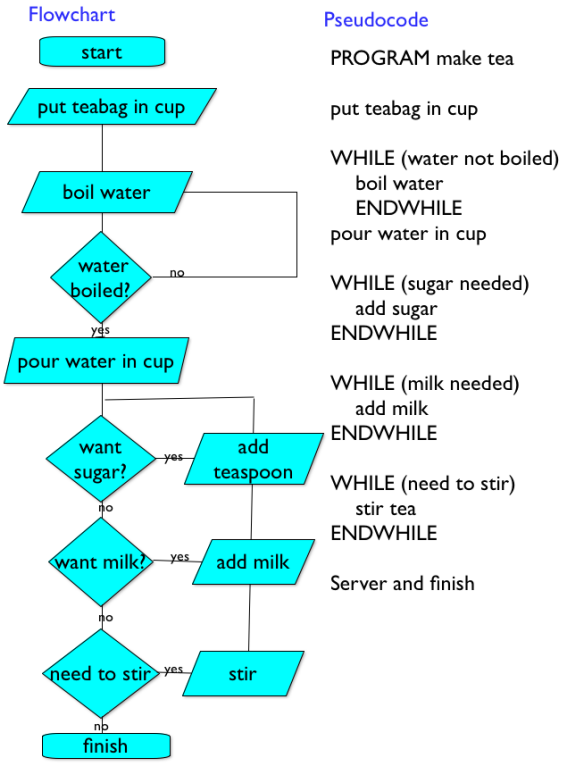


- ➢ ຂັ້ນຕອນວິທີທີ່ດີ ຕ້ອງມີຄຸນສົມບັດດັ່ງລຸ່ມນີ້:
  - ✓ ມີຄວາມຖືກຕ້ອງ
  - ✓ ໃຊ້ເວລາໃນການປະຕິບັດງານໜ້ອຍທີ່ສຸດ
  - ✓ ສັ້ນກະທັດຮັດ ມີສະເພາະຂັ້ນຕອນທີ່ຈຳເປັນ
  - ✓ ນຳໃຊ້ໜ່ວຍຄວາມຈຳໜ້ອຍທີ່ສຸດ
  - ✓ ມີຄວາມຢືດຢຸ່ນໃນການໃຊ້ງານ
  - ✓ ໃຊ້ເວລາໃນການພັດທະນາໜ້ອຍທີ່ສຸດ
  - ✓ ງ່າຍຕໍ່ການທຳຄວາມເຂົ້າໃຈ

1. ການສະແດງຂັ້ນຕອນວິທີ

ການສະແດງຂັ້ນຕອນວິທີ ແມ່ນການສະແດງຂັ້ນຕອນທີ່ໄດ້ນຳໃຊ້ ບໍ່ວ່າຈະເປັນ ການຂຽນຜັງງານ (Flowchart), ລະຫັດຈຳລອງ ຫຼື ລະຫັດທຽມ ແລະ ພາສາທຳມະຊາດ.





2. ພາສາຂັ້ນຕອນວິທີ

ເປັນພາສາ ສຳລັບຂຽນຂັ້ນຕອນວິທີ ມີຮູບແບບທີ່ສັ້ນ ກະທັດຮັດ ແລະ ມີຂໍ້ກຳນົດດັ່ງຕໍ່ໄປນີ້:

- ຕົວປ່ຽນຈະຕ້ອງຂຽນແທນດ້ວຍຕົວອັກສອນ ຫຼື ຕົວອັກສອນປະສົມກັບຕົວເລກ
- ການກຳນົດຄ່າໃຫ້ກັບຕົວປ່ຽນຈະໃຊ້ເຄື່ອງໝາຍເທົ່າກັບ =
- ນິພົນທີ່ເປັນການຄຳນວນຈະມີລຳດັບຂັ້ນຂອງການຄຳນວນຕາມລຳດັບຄື: ວົງເລັບ, ຂຶ້ນກຳລັງ, ຄູນ, ຫານ, ບວກ ແລະ ລົບ ລຳດັບຄວາມສຳຄັນແມ່ນແຕ່ຊ້າຍຫາຂວາ

  ນິພົດທີ່ເປັນຕັກກະສາດ

  ນິພົດທີ່ເປັນຕັກກະສາດ ຈະໃຊ້ເຄື່ອງໝາຍໃນການປຽບທຽບ ຄື:

  == ເທົ່າກັບ

  != ບໍ່ເທົ່າກັບ

  > ໃຫຍ່ກວ່າ

  < ນ້ອຍກວ່າ

  >= ໃຫຍ່ກວ່າ ຫຼື ເທົ່າກັບ

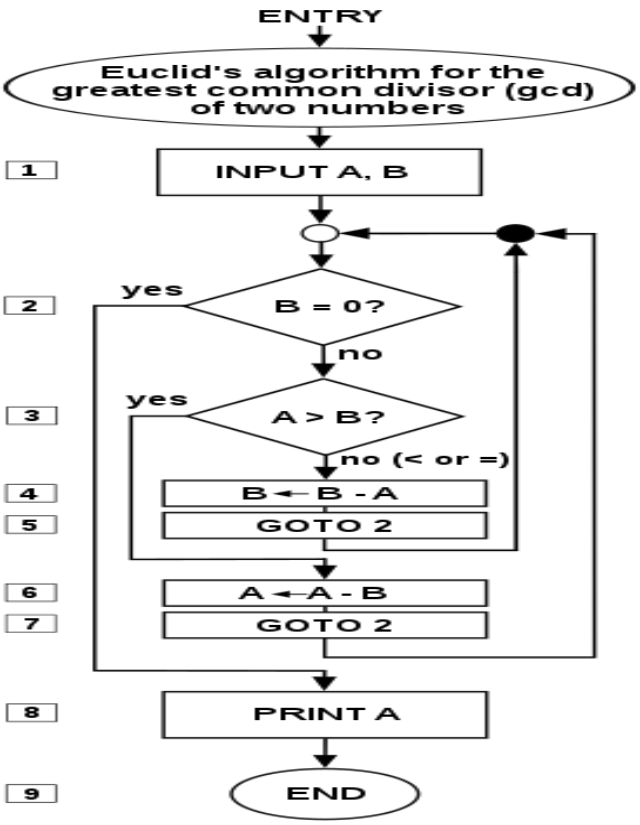
  <= ນ້ອຍກວ່າ ຫຼື ເທົ່າກັບ

ບົດທີ 2:

ການວິເຄາະປະສິດທິພາບຂອງຂັ້ນຕອນວິທີ

I. ການວິເຄາະປະສິບທິພາບຂອງຂັ້ນຕອນວິທີ

ປະສິດທິພາບຂອງອາກໍຣິທຶມ ໂດຍທົ່ວໄປມີມາດຕະຖານການວັດຜົນສອງແບບໂດຍແບບທຳອິດແມ່ນ ການໃຊ້ໃນພື້ນທີ່ວ່າງ (space Utilization) ເປັນຈຳນວນຂອງຫນ່ວຍຄວາມຈຳທີ່ຕ້ອງໃຊ້ເພື່ອໃຫ້ງານສຳເລັດ ,ປະກອບດ້ວຍພື້ນທີ່ເກັບຄຳສັ່ງທີ່ເກັບຂໍ້ມູນ ແລະພື້ນທີ່ສະພາວະສິ່ງແວດລ້ອມແຕກຕ່າງຄືແບບທີ່ 2 ປະສິດທິພາບຂອງເວລາພາວະເວລາ (time Efficiency) ເປັນຈຳນວນເວລາທີ່ໃຊ້ໃນການປະມວນຜົນ



ຄຸນລັກສະນາຂອງຂັ້ນຕອນວິທີທີ່ດີມີຄື:

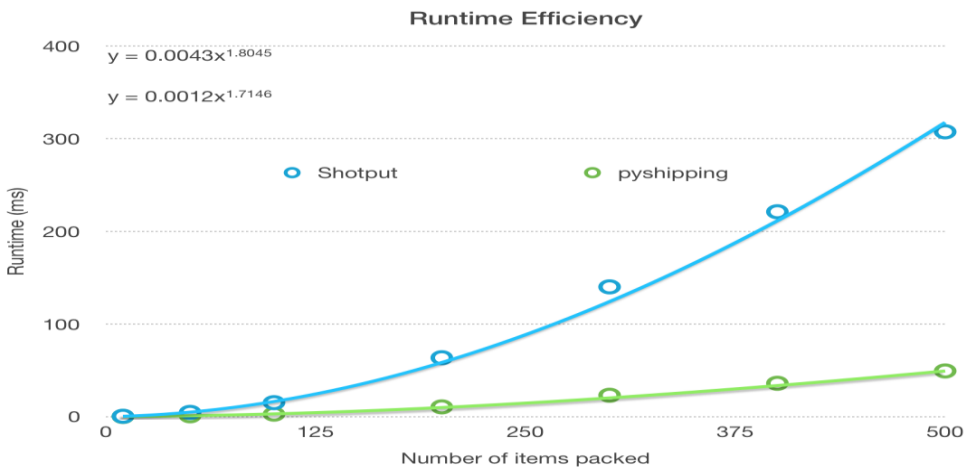
1. ເຮັດວຽກໄດ້ຢ່າງຖືກຕ້ອງ
2. ເຮັດວຽກໄດ້ໄວ
3. ໃຊ້ຊັບພະຍາກອນຢ່າງຄຸ້ມຄ່າ
4. ບໍ່ສັບຊ້ອນ (ເຂົ້າໃຈງ່າຍ)
5. ສາມາດນຳໄປໃຊ້ໄດ້ກັບວຽກທີ່ຫຼາກຫຼາຍ

Example:

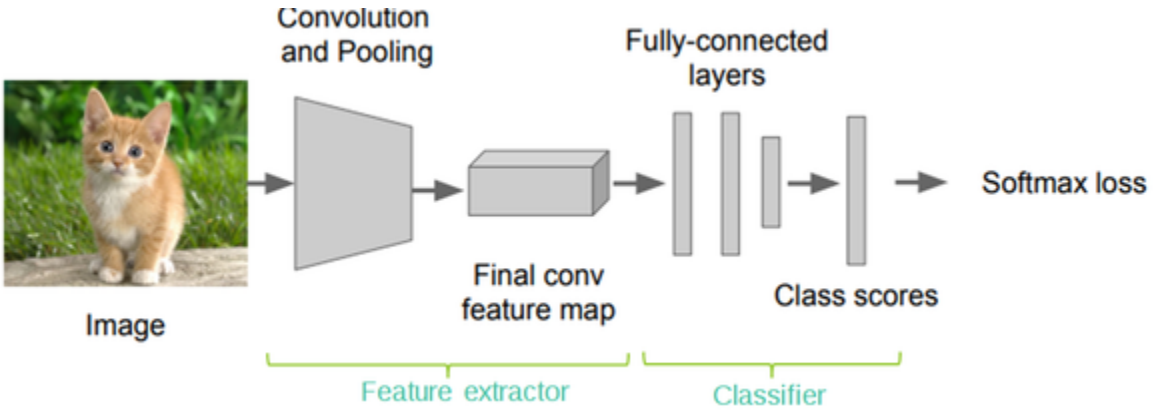
```
Sum of Two Numbers:

step 1 - START ADD
step 2 - get values of a & b
step 3 - c ← a + b
step 4 - display c
step 5 - STOP
```

II. ປະສິດທິພາບຂອງຂັ້ນຕອນວິທີ

ປະສິດທິພາບຂອງ ຂັ້ນຕອນວິທີ ຫຼື ອານກໍຣິດທຶມ (Algorithm) ແມ່ນ ການປະເມີນຄ່າ ຊັບພະຍາກອນທີ່ຈຳເປັນຕ້ອງໃຊ້ໃນການທຳງານ ເຊັ່ນ: ເວລາ ຫຼື ໜ່ວຍຄວາມຈຳ, ຂັ້ນຕອນວິທີ ສ່ວນຫຼາຍອອກແບບມາເພື່ອໃຫ້ສາມາດຮອງຮັບຈຳນວນຂໍ້ມູນນຳເຂົ້າ (Input) ໄດ້ບໍ່ຈຳກັດ ປົກກະຕິແລ້ວປະສິດທິພາບ ຫຼື ຄວາມສັບຊ້ອນຂອງຂັ້ນຕອນວິທີ ຈະວັດຈາກຄວາມສຳພັນຂອງ ຈຳນວນຂໍ້ມູນນຳເຂົ້າ ກັບ ເວລາທີ່ໃຊ້ໃນການທຳງານ ຫຼື ໜ່ວຍຄວາມຈຳທີ່ນຳໃຊ



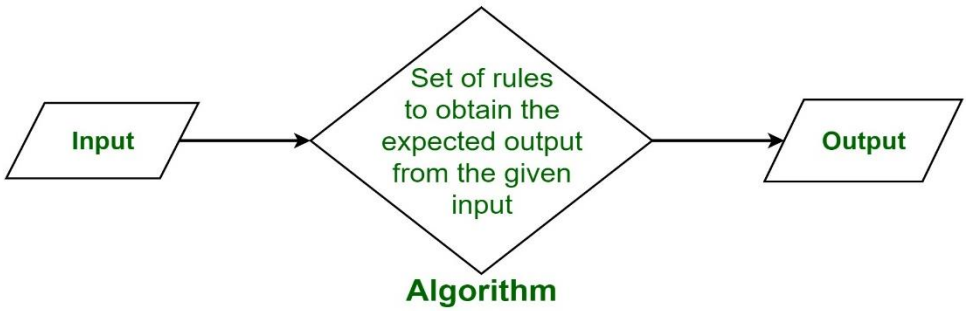
III. ວິທີວິເຄາະງານໃຊ້ຫນ່ວຍຄວາມຈຳທັງຫມົດ



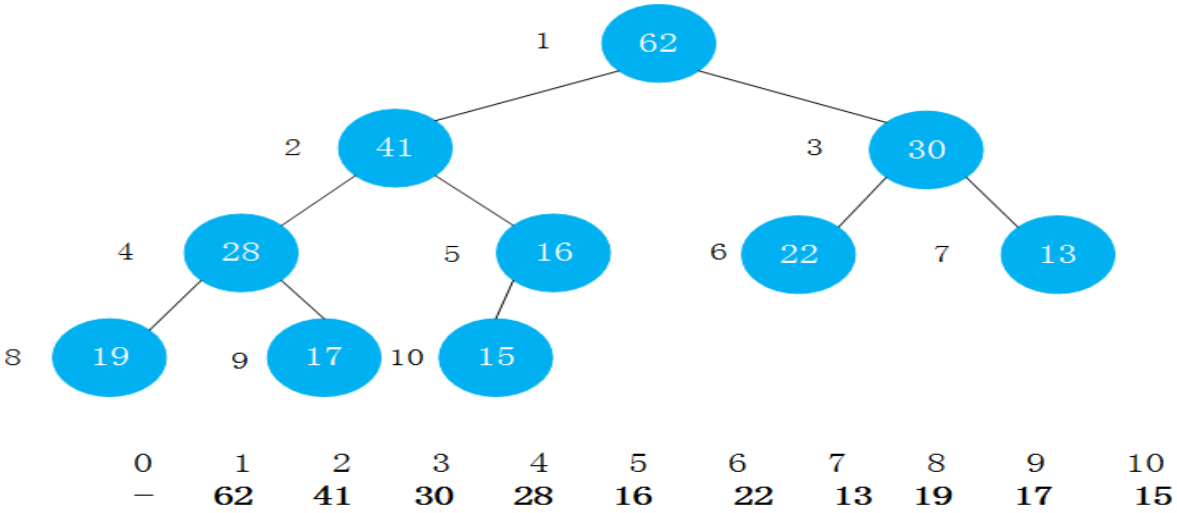
1. ການວິເຄາະໜ່ວຍຄວາມຈຳທັງໝົດທີ່ໂປຣແກຣມໃຊ້ໃນການປະມວນຜົນຂອງຂັ້ນຕອນວິທີ
2. ເພື່ອໃຫ້ຮູ້ເຖິງຂະໜາດຂໍ້ມູນທີ່ສາມາດປ້ອນ ຫຼື ຂໍ້ມູນເຂົ້າມາໃຫ້ຂັ້ນຕອນວິທີປະມວນຜົນແລ້ວບໍ່ເກີດຂໍ້ຜິດພາດ

IV. ອົງປະກອບຂອງການວິເຄາະການໃຊ້ໜ່ວຍຄວາມຈຳທີ່ໃຊ້ໃນການປະມວນຜົນ

- Instruction Space: ແມ່ນ ຂະໜາດໜ່ວຍຄວາມຈຳທີ່ຈຳເປັນຕ້ອງໃຊ້ໃນເວລາຄອມໄຟລເລີ້ ໂປຣແກຣມ

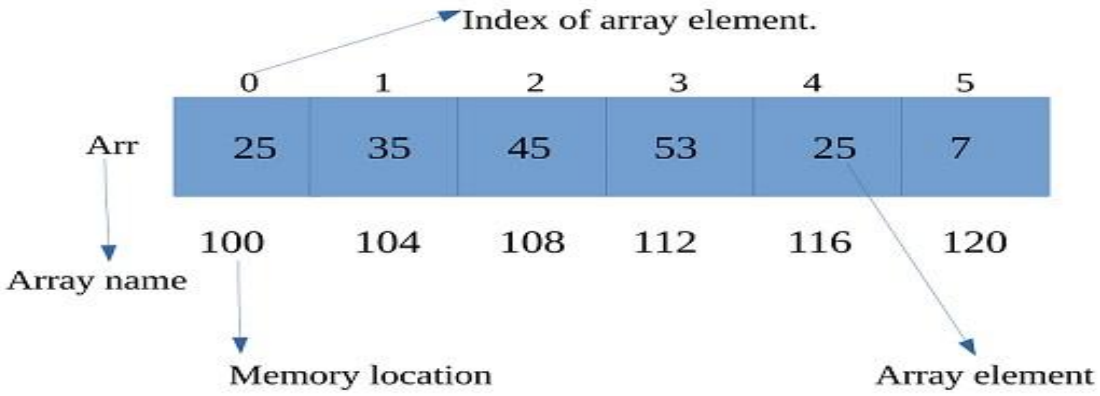


- Data Space: ແມ່ນ ຂະໜາດໜ່ວຍຄວາມຈຳທີ່ຈຳເປັນຕ້ອງໃຊ້ສຳລັບເກັບຂໍ້ມູນຄ່າຄົງທີ່ ແລະ ຕົວປ່ຽນທີ່ໃຊ້ໃນເວລາປະມວນຜົນໂປຣແກຣມ

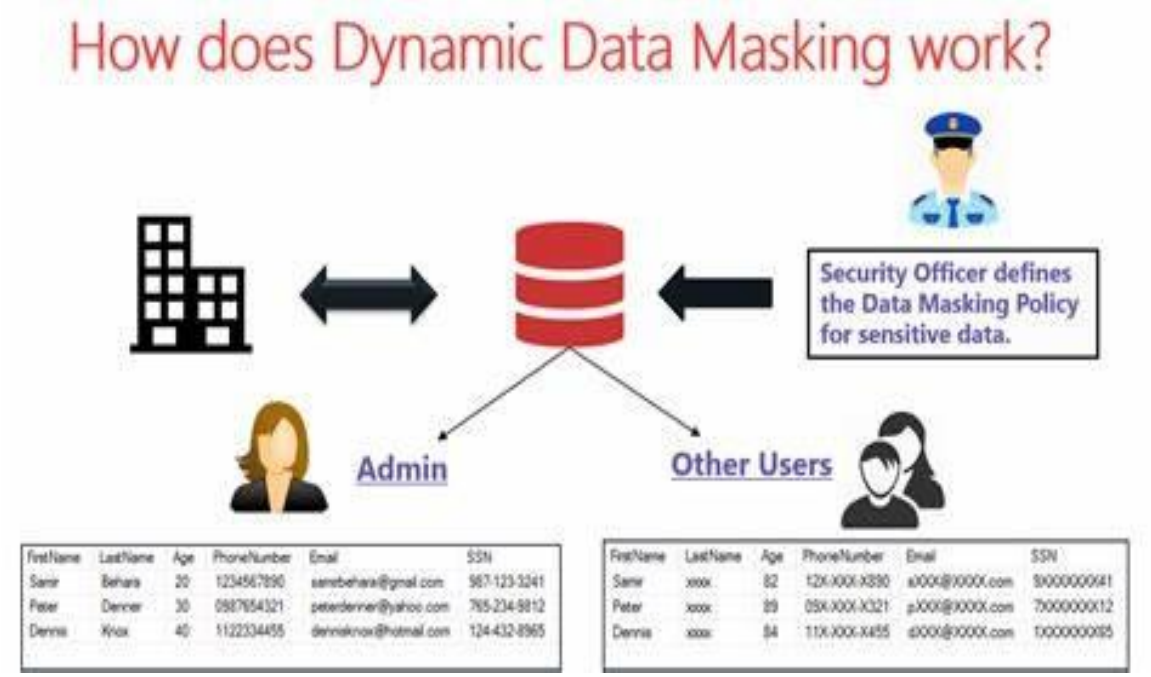


ເຊິ່ງແບ່ງອອກເປັນ 2 ປະເພດ ຄື:

- Static memory: ຂະໜາດໜ່ວຍຄວາມຈຳທີ່ຕ້ອງໃຊ້ໃນການປະມວນຜົນຢ່າງແນ່ນອນ ເຊັ່ນ: Array

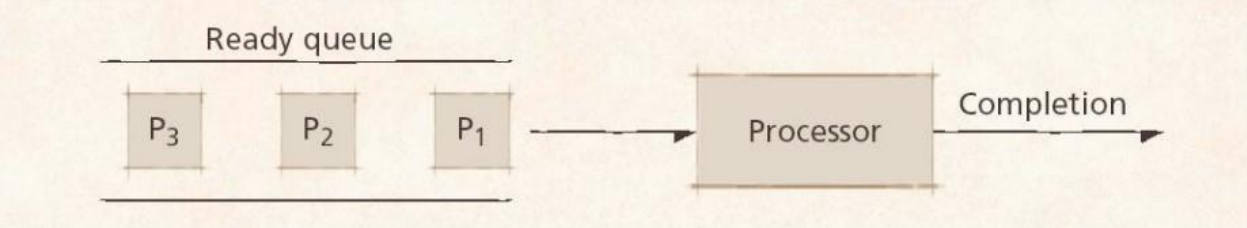


- Dynamic memory: ຂະໜາດໜ່ວຍຄວາມຈຳທີ່ຕ້ອງໃຊ້ໃນການປະມວນຜົນທີ່ບໍ່ແນ່ນອນ ໝາຍຄວາມວ່າ ຈະໃຊ້ໜ່ວຍຄວາມຈຳນີ້ກໍ່ຕໍ່ເມື່ອໂປຣແກຣມຕ້ອງການໃຊ້ງານເທົ່ານັ້ນ ໂດຍບໍ່ມີການຈອງເນື້ອທີ່ໄວ້ລ່ວງໜ້າ

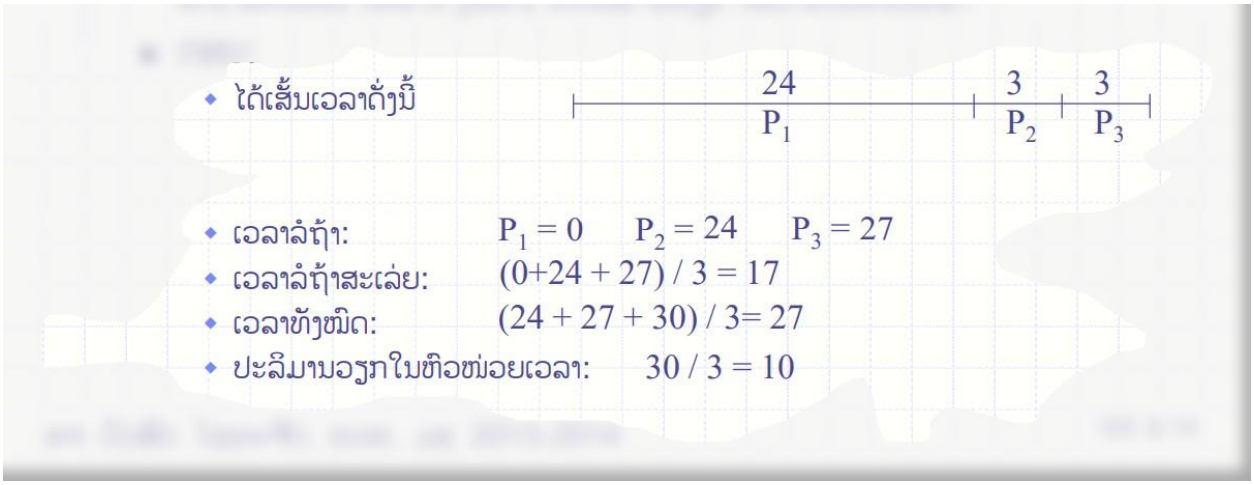


- Environment Stack Space: ແມ່ນໜ່ວຍຄວາມຈຳທີ່ໃຊ້ສຳລັບເກັບຂໍ້ມູນຜົນຮັບທີ່ໄດ້ຈາກການປະມວນຜົນ ເພື່ອລໍເວລາທີ່ຈະນຳກັບໄປໃຊ້ງານໃໝ່ໃນໂປຣແກຣມ ເຊິ່ງໜ່ວຍຄວາມຈຳປະເພດນີ້ຈະເກີດຂຶ້ນເມື່ອມີການໃຊ້ງານເທົ່ານັ້ນ

V. ການວິເຄາະເວລາທີ່ໃຊ້ໃນການປະມວນຜົນ



- ເພື່ອວິເຄາະຂັ້ນຕອນທີ່ຕ້ອງໃຊ້ໃນການດຳເນີນການປະມວນຜົນຂັ້ນຕອນວິທີ
- ເພື່ອໃຊ້ປະເມີນເວລາທີ່ໃຊ້ໃນການປະມວນຜົນ.
- ຮູ້ເຖິງປະສິດທິພາບການທຳງານຂອງໂປຣແກຣມ ແລະ ແກ້ໄຂປັນຫາໄດ້ຢ່າງຖືກຕ້ອງ
- ສາມາດເລືອກໃຊ້ຄອມພິວເຕີທີ່ເໝາະສົມກັບການປະມວນຜົນຂັ້ນຕອນວິທີ



ຫຼັກການພິຈາລະນາເວລາທີ່ໃຊ້ໃນການປະມວນຜົນ

- ❖ ເມື່ອປະມວນຜົນໂປຣແກຣມໃນເຄື່ອງຄອມພິວເຕີທີ່ມີປະສິດທິພາບໄວກວ່າ ກໍ່ຈະປະມວນຜົນຂໍ້ມູນໄວກວ່າ
- ❖ ເມື່ອຣັນໂປຣແກຣມທີ່ມີຜົນການເຮັດວຽກດຽວກັນ ໂປຣແກຣມທີ່ມີໂຄດນ້ອຍກວ່າຈະເຮັດວຽກໄດ້ໄວກວ່າ
- ❖ ຕົວປ່ຽນທີ່ມີຂະໜາດໜ່ວຍຄວາມຈຳນ້ອຍກວ່າຈະປະມວນຜົນໄດ້ໄວກວ່າ

ตัวอย่าง: กรณีที่ 1

ขบวนการ :	P1	P2	P3	P4
เวลามาถึง:	0	2	4	5
เวลาใช้ CPU :	7	4	1	4

จัดตารางสำหรับ preemptive SJF:

P_1	P_2	P_3	P_2	P_4	P_1

เวลาในการรอคอย : 9 1 0 2

เวลารอคอยเฉลี่ย : $(9 + 1 + 0 + 2)/4 = 3$

เวลาทั้งหมด : $9 + 7$ $1 + 4$ $0 + 1$ $2 + 4$

เวลาโดยเฉลี่ย : $(16 + 5 + 1 + 6)/4 = 7$

ประเภทของเวลาที่ในการประมวลผล

- Compile time เป็นเวลาที่ใช้ในการตรวจไวยากรณ์การเขียน (Syntax)
- Run time หรือ Execution time เป็นเวลาที่คอมพิวเตอร์ใช้ในการประมวลผลขั้นตอนวิธี เกี่ยวข้องกับข้อมูล, จำนวนตัวแปรที่ใช้ในโปรแกรม และ จำนวนวงรอบ

ตัวอย่าง: วิเคราะห์เวลาที่ใช้ในการประมวลผลของโปรแกรมต่อไปนี้:

- int n = 20; ← กำหนดค่า 1 ครั้ง
- int total = 0; ← กำหนดค่า 1 ครั้ง
- while(n!=20) { ← เปรียบเทียบค่า n+1 ครั้ง
- total += n; ← คำนวณ n ครั้ง
- ++n; ← คำนวณ n ครั้ง
- } // จบคำสั่ง while
- System.out.println("Total = " + total); ← แสดงผล 1 ครั้ง

ຄຳອະທິບາຍ

ກຳນົດໃຫ້ f(n) ແທນປະສິດທິພາບໃນການວິເຄາະເວລາທີ່ໃຊ້ໃນການປະມວນຜົນ

n ແທນຈຳນວນຮອບໃນການເຮັດວຽກ

ຈະໄດ້ວ່າຂັ້ນຕອນວິທີນີ້ມີປະສິດທິພາບ

f(n) = 1+1+(n+1)+n+n+1 = 3n+4

ຕົວຢ່າງ: ວິເຄາະເວລາທີ່ໃຊ້ໃນການປະມວນຜົນຂອງໂປຣແກຣມວົນຊ້ຳລຸ່ມນີ້:

```
factorial(in n:int):int                    ← ຖືກເອີ້ນໃຊ້ n ຄັ້ງ
    if(n == 0)                             ← ກວດສອບເງື່ອນໄຂ n ຄັ້ງ
        return 1                           ← ຄືນຄ່າ 1 ຄັ້ງ
    else return n*(factorial(n-1))         ← ເອີ້ນໃຊ້ຕົວເອງ n ຄັ້ງ
```

- ພິຈາລະນາຕາມຈຳນວນຮອບທີ່ວົນຊ້ຳ ຖ້າກຳນົດ n = 3 ຈະຕ້ອງທຳການວົນຊ້ຳ 3 ຄັ້ງ
- ໃຊ້ເນື້ອທີ່ໃນໜ່ວຍຄວາມຈຳເທົ່າກັບ 3*4 = 12 byte
- ສະຫຼຸບໄດ້ວ່າ ຖ້າກຳນົດໃຫ້ຫາຄ່າ Factorial n ຈະຕ້ອງໃຊ້ເນື້ອທີ່ໜ່ວຍຄວາມຈຳເທົ່າກັບ n*4

ບົດທີ 3:

ປະສິດທິພາບຂອງ Algorithm

I. ປະສິດທິພາບຂອງ Algorithm

ປະສິດທິພາບຂອງອະກໍຣິທຶມແມ່ນອາໄສການພິຈາລະນາຈາກເວລາທີ່ໃຊ້ໃນການປະມວນຜົນ ແລະ ຈາກຈຳນວນຂອງໜ່ວຍຄວາມຈຳທີ່ໃຊ້ໃນການປະມວນຜົນນັ້ນແມ່ນບໍ່ຍຸດຕິທຳພຽງພໍການປຽບທຽບຂອງອະກໍຣິທຶມວ່າອັນໃດດີກວ່າກັນນັ້ນຕ້ອງມີວິທີເໝາະສົມ ແລະ ຍຶດຕິທຳ.

Efficiency of Algorithms

- Efficiency: Amount of resources used by an algorithm
  - Space (number of variables)
  - Time (number of instructions)
- When design algorithm must be aware of its use of resources
- If there is a choice, pick the more efficient algorithm!

ປະສິດທິພາບຂອງອະກໍຣິທຶມໄດ້ຖືກພິຈາລະນາຈາກ 2 ປັດໃຈຄື:

1. ການວິເຄາະ (Analysis) ໃຊ້ວິເຄາະວິທີການເຮັດວຽກຂອງອະກໍຣິທຶມ
2. ການວັດແທກ (Measure) ໃຊ້ເພື່ອວັດແທກຈາກການທົດລອງຕົວຈິງ

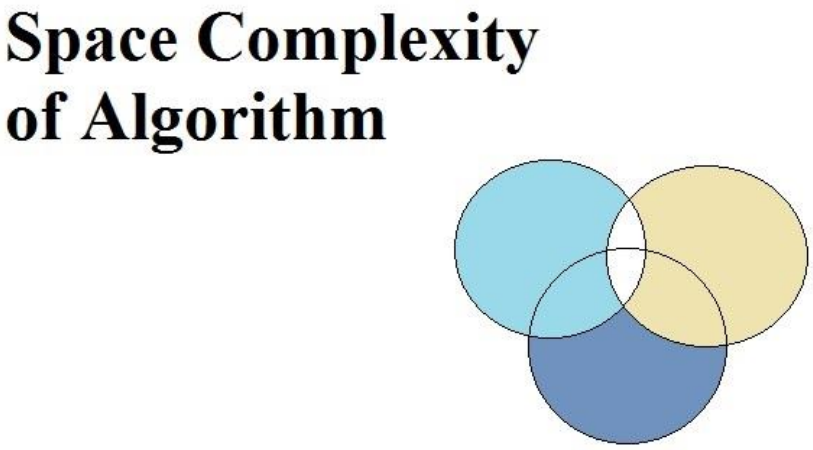
ປະສິດທິພາບຂອງອະກໍຣິທຶມໂດຍທົ່ວໄປມີມາດຕະຖານການວິເຄາະ 2 ແບບຄື:

1. ການວິເຄາະໂດຍໃຊ້ໜ່ວຍຄວາມຈຳທີ່ຕ້ອງໃຊ້ໃນການປະມວນຜົນ (Space Complexity).
2. ການວິເຄາະເວລາທີ່ໃຊ້ໃນການປະມວນຜົນ (Time Complexity).

Algorithm	Time complexity	Space complexity
GRNAs	$O(l^2 + \|P\|)$	$O(l^2 + \|P\|)$
TIRNA	$O(k^2 \log k^2)$	$O(k^2)$
SPM	$O(n^3m^3)$	$O(n^2m^2)$
LM	$O(n^3m^3)$	$O(n^2m^2)$
inRNAs	$O(k^4w)$	$O(k^2)$
RNAup	$O(n^3m)$	$O(n^2)$
EBM	$O(n^3m^3)$	$O(n^2m^2)$
App	$O(n^3m^3)$	$O(n^2m^2)$
Pairfold	$O(k^3)$	$O(k^2)$
IntaRNA	$O(nm + nl^3)$	$O(nm)$
ripalign	$O(N^6)$	$O(N^4)$
PETcofold	$O(Mll^3)$	
RactIP	$O(n^5)$	

II. ການວິເຄາະໂດຍໃຊ້ໜ່ວຍຄວາມຈຳທີ່ຕ້ອງໃຊ້ໃນການປະມວນຜົນ (Space Complexity)

ແມ່ນການວິເຄາະທີ່ເຮັດໃຫ້ຮູ້ຈັກວ່າອະກໍຣິທຶມນັ້ນສາມາດຮອງຮັບຂໍ້ມູນທີ່ສົ່ງເຂົ້າມາປະມວນຜົນໄດ້ຫຼາຍສຸດເທົ່າໃດ, ສາລັບອະກໍຣິທຶມທີ່ຕ້ອງປະມວນຜົນໃນຄອມພິວເຕີທີ່ໃຊ້ຮ່ວມກັນຫຼາຍຄົນຜ່ານລະບົບເຄືອຂ່າຍຈຳເປັນຕ້ອງຮູ້ຈຳນວນຂອງໜ່ວນຄວາມຈຳເພື່ອບໍ່ໃຫ້ກະທົບກັບການເຮັດວຽກຂອງຄົນອື່ນ ແລະ ຍັງສາມາດເລືອກສະເປັກຂອງຄອມພິວເຕີຊຶ່ງຈະນຳໂປຣແກຣມໄປຕິດຕັ້ງໄດ້ຢ່າງເໝາະສົມ.



III. ການວິເຄາະເວລາທີ່ໃຊ້ໃນການປະມວນຜົນ (Time Complexity)

ก. ຫຼັກການໃນການພິຈາລະນາເວລາທີ່ໃຊ້ໃນການປະມວນຜົນ

ໂປຣແກຣມຈະສາມາດປະມວນຜົນໄດ້ໄວກວ່າເມື່ອເຮັດວຽກຢູ່ເທິງເຄື່ອງທີ່ມີຄວາມໄວໃນການປະມວນຜົນສູງກວ່າ, ຖ້າໃຊ້ Compiler ຕົວດຽວກັນ Code ທີ່ສັ້ນກວ່າຈະໃຊ້ເວລາໃນການປະມວນຜົນທີ່ສັ້ນກວ່າ, ມີຕົວປ່ຽນທີ່ມີຈຳນວນ byte ນ້ອຍກວ່າຈະປະມວນຜົນໄດ້ໄວກວ່າ.

ຂ. ເວລາທີ່ໃຊ້ໃນການປະມວນຜົນຂອງໂປຣແກຣມ

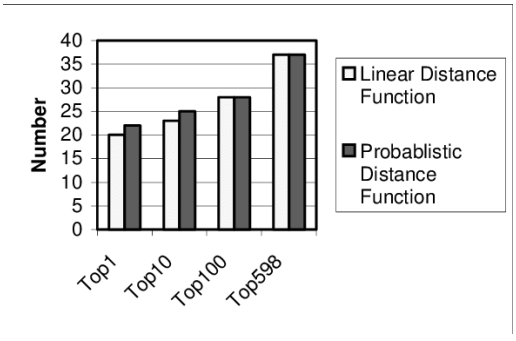
- Compile Time ໄລຍະເວລາໃນການກວດສອບໄວຍະກອນຂອງ Code.
- Run Time / Execution Time ເວລາທີ່ເຄື່ອງຄອມພິວເຕີ້ທີ່ໃຊ້ໃນການເຮັດວຽກເຊິ່ງຂຶ້ນກັບຈຳນວນຕົວປ່ຽນ ແລະ ຄຸນລັກສະນະຂອງຕົວປ່ຽນທີ່ໃຊ້ໃນໂປຣແກຣມ.

ຂະໜາດຂອງຂໍ້ມູນທີ່ໄດ້ຮັບເຂົ້າມາ

ແມ່ນຈຳນວນຂອງຂໍ້ມູນທີ່ຮັບເຂົ້າມາເຮັດວຽກຈະມີຜົນກະທົບຕໍ່ເວລາໃນການໃຊ້ງານການປະມວນຜົນຂໍ້ມູນຂອງລາຍການຂຶ້ນກັບຈຳນວນຂອງຂໍ້ມູນທີ່ໃນລາຍການ. ດັ່ງນັ້ນໃນເວລາເຮັດວຽກຂອງອະກໍຣິທຶມຈະສະແດງໃນຮູບແບບ T(n) ຂອງຂໍ້ມູນທີ່ໄດ້ຮັບຂໍ້ມູນມາມີຂະໜາດ n.

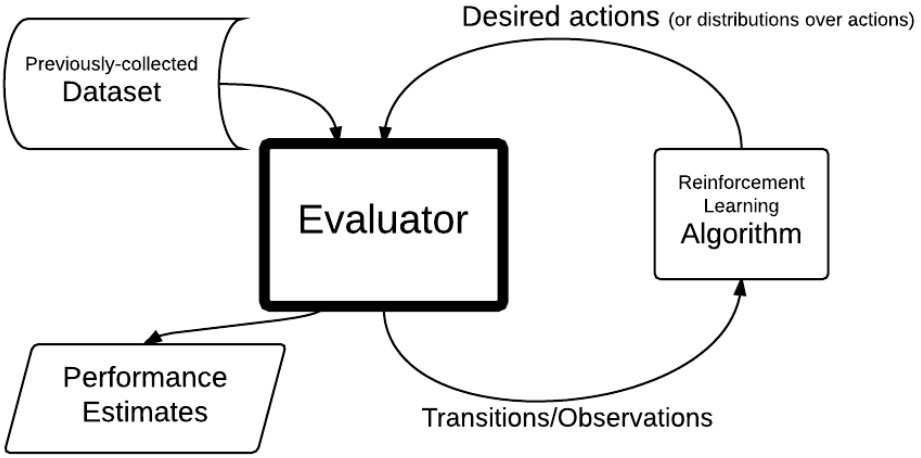
ການປຽບທຽບ

ການປຽບທຽບແມ່ນອາໄສການພິຈາລະນາຈາກເວລາທີ່ໃຊ້ໃນການປະມວນຜົນ ແລະ ຈາກຈຳນວນຂອງໜ່ວຍຄວາມຈຳທີ່ໃຊ້ໃນການປະມວນຜົນນັ້ນແມ່ນບໍ່ຍຸດຕິທຳພຽງພໍ. ການປຽບທຽບອະກໍຣິທຶມວ່າອັນໃດດີກວ່າກັນນັ້ນຕ້ອງມີວິທີເໝາະສົມ ແລະ ຍຸດຕິທຳ.



ການປະເມີນ

ວິທີການປະເມີນປະສິດທິພາບຂອງອະກໍຣິທຶມໂດຍການວິເຄາະສະເພາະອະກໍຣິທຶມພຽງຢ່າງດຽວເປັນ ວິທີທີ່ໃຊ້ສັນຍາລັກ.



Step Count

ແມ່ນຄ່າຕຳລາທີ່ໃຊ້ອະທິບາຍພຶດຕິກຳການປະມວນຜົນຂອງອະກໍຣິທຶມເປັນຄ່າຕຳລາທາງຄະນິດສາດທີ່ ສະແດງເຖິງຄວາມສຳພັນລະຫວ່າງປະລິມານຂໍ້ມູນທີ່ກຳລັງຖືກປະມວນຜົນກັບເວລາທີ່ຕ້ອງໃຊ້ໃນການປະມວນຜົນ ຂໍ້ມູນນັ້ນ.

ການຊອກຫາຄ່າເວລາທີ່ໃຊ້ໃນການປະມວນຜົນຂອງອະກໍຣິທຶມໃດໜຶ່ງມີ 2 ວິທີຄື:

1. Operation Count

ເປັນວິທີການຄຳນວນເວລາໃນການປະມວນຜົນໂດຍພິຈາລະນາຈາກ Operation ທີ່ຢູ່ພາຍໃນອະ ກໍຣິທຶມນັ້ນຊຶ່ງເປັນການນັບຈຳນວນການປະມວນຜົນ Operation ຕ່າງໆທັງໝົດໃນອະກໍທຶມ.

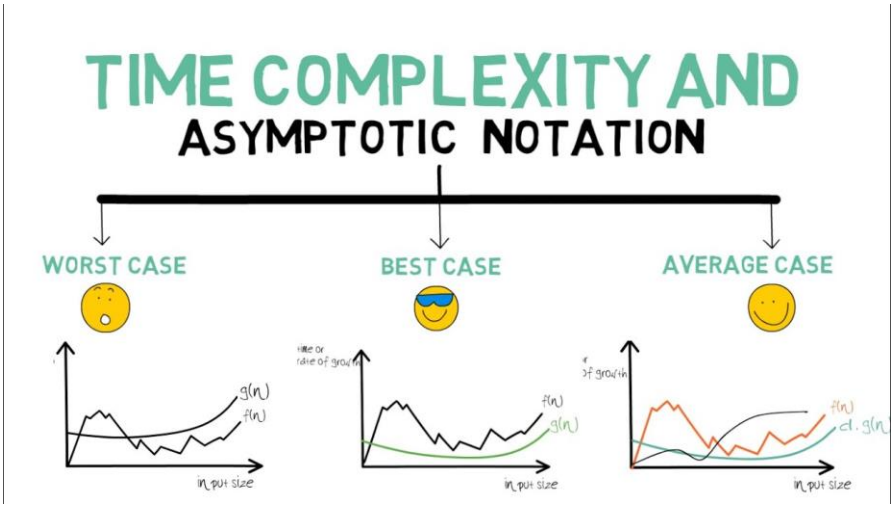
```
Algorithm prefixAverages2(X, n)
    Input array X of n integers
    Output array A of prefix averages of X        #operations
    A ← new array of n integers                   n
    s ← 0                                         1
    for i ← 0 to n − 1 do                         n
        s ← s + X[i]                              n
        A[i] ← s / (i + 1)                        n
    return A                                      1
```

2. Step Count

ເປັນວິທີຄຳນວນເວລາໃນການປະມວນຜົນໂດຍພິຈາລະນາຈຳນວນວຽກທັງໝົດທີ່ອະກໍຣິທຶມເຮັດ ຫຼື ເປັນການຊອກຫາວ່າຈະຕ້ອງປະມວນຜົນຄຳສັ່ງທັງໝົດທີ່ຢູ່ພາຍໃນອະກໍຣິທຶມນັ້ນທັງໝົດຈັກຄັ້ງ.

ສັນຍາລັກ

ສັນຍາລັກເປັນຕຳລາຂອງເວລາທີ່ໃຊ້ໃນການປະມວນຜົນຂອງອະກໍຣິທຶມໃດໜຶ່ງໂດຍຈະພິຈາລະນາຈາກຄ່າເມື່ອອະກໍຣິທຶມນັ້ນມີປະລິມານຂໍ້ມູນທີ່ຫຼາຍທີ່ສຸດ.



- ## ຜົນປະໂຫຍດຂອງສັນຍາລັກ Asymptotic

- ເພື່ອໃຊ້ຊັບພະຍາກອນເວລາທີ່ຕ້ອງການຈະໃຊ້ໃນການປະມວນຜົນຂອງອະກໍຣິທຶມໃດໜຶ່ງໂດຍສັງເຂບໄດ້ໃນປະລິມານຂໍ້ມູນຕ່າງໆ.
- ໃຊ້ໃນການປຽບທຽບປະສິດທິພາບຂອງອະກໍຣິທຶມທີ່ເຮັດວຽກຢ່າງດຽວ.

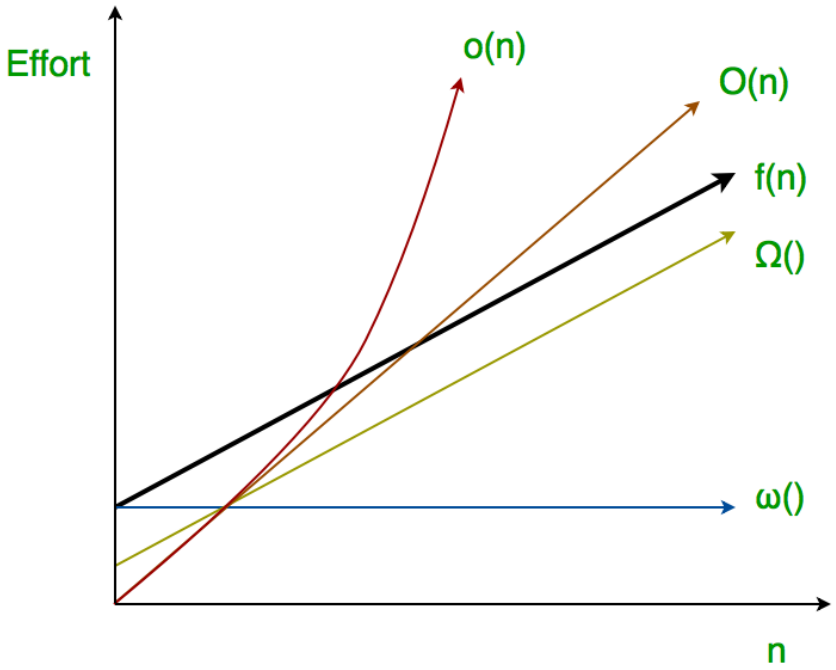
- ## ການວິເຄາະສັນຍາລັກຈາກອະກໍຣິທຶມ

ການວິເຄາະສັນຍາລັກຈາກອະກໍຣິທຶມແມ່ນການພິຈາລະນາຄຳສັ່ງຕ່າງໃນອະກໍຣິທຶມຊອກຫາຄ່າຂອງສັນຍາລັກ Asymptotic ໃດໜຶ່ງທີ່ຕ້ອງການ.

- ## ວິທີອ່ານຄ່າຂອງອະກໍຣິທຶມ

ແມ່ນການອ່ານວ່າອະກໍຣິທຶມອັນໃດມີປະສິດທິພາບດີກວ່າແມ່ນຄ່າຈາກຕຳລາຂອງອະກໍຣິທຶມນັ້ນ ໂດຍຕຳລາທີ່ຄ່າໜ້ອຍກວ່າຈະມີປະສິດທິພາບຫຼາຍກວ່າ. ປະເພດຂອງສັນຍາລັກມີຄື:

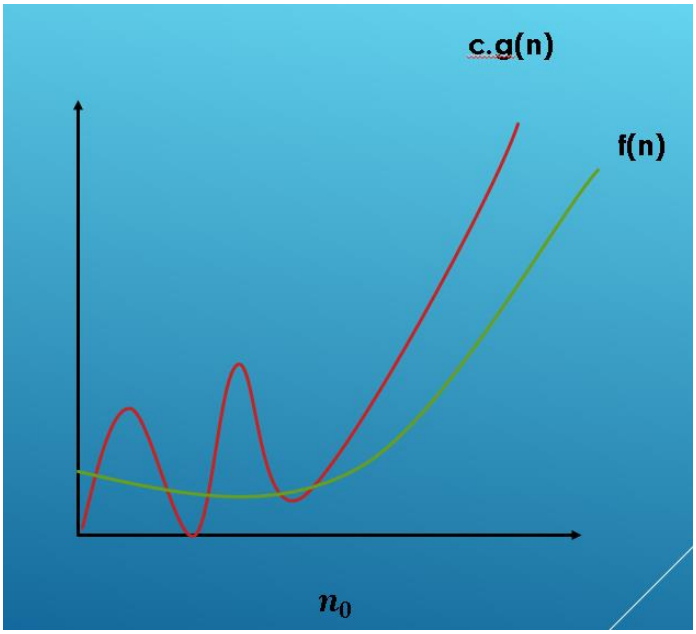
1. Big O ໃຊ້ເຄື່ອງໝາຍ O()
2. Omega ໃຊ້ເຄື່ອງໝາຍ Ω()
3. Theta ໃຊ້ເຄື່ອງໝາຍ Θ()
4. Little O ໃຊ້ເຄື່ອງໝາຍ o()



Big O

ແມ່ນການຊອກຫາຂອບເຂດເທິງຂອງຕຳລາຂອງເວລາ (Upper-bound function)
ສຳລັບອະກໍຣິທຶມໃດໜຶ່ງ

f(n) = O(g(n)) **ກໍຕໍ່ເມື່ອມີຄ່າຄົງທີ່** C>0 **ແລະ** n_0 >0 **ທີ່ເຮັດໃຫ້** f(n) ≤ cg(n) **ສຳຫຼັບທຸກໆຄ່າຂອງ** n, **ເມື່ອ** $n \geq n_0$



ຕົວຢ່າງ: ຊອກຫາຄ່າ Big O ຂອງອະກໍຣິທຶມທີ່ມີຕຳລາເວລາດັ່ງນີ້:

$f(n) = 4n+7$

ແກ້

ເມື່ອ $n \geq 7$

->$4n + 7 \leq 4n + n \leq 5n$

->$4n + 7 \leq 5n$

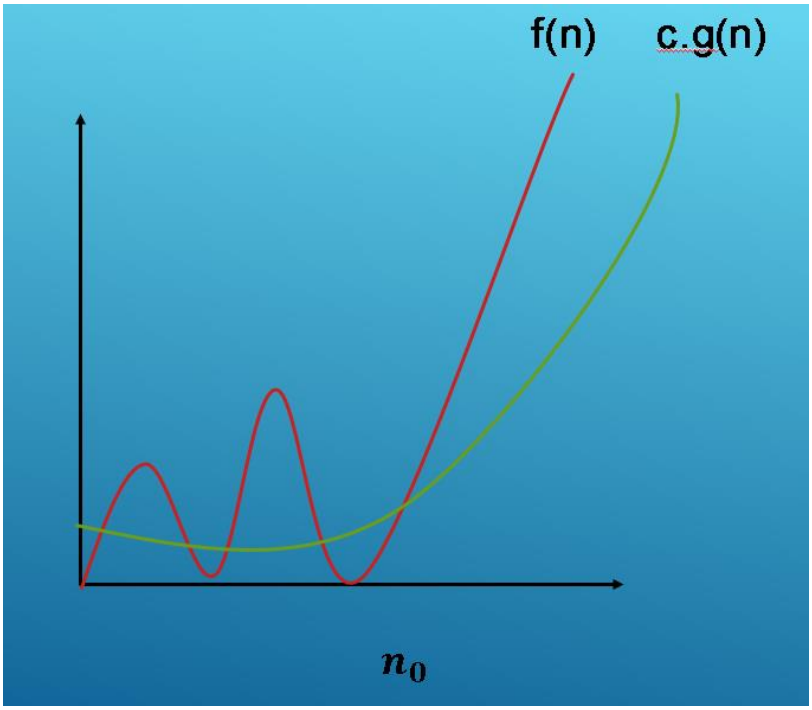
->$f(n) \leq 5n$

ດັ່ງນັ້ນ $f(n) = O(n)$ ສຳລັບ $n \geq n_0$, ເມື່ອ $c=5$ ແລະ $n_0 = 7$

Omega

ເປັນສັນຍາລັກທີ່ກົງກັນຂ້າມກັບ Big O ເປັນຂອງເຂດລຸ່ມຂອງຕຳລາຂອງເວລາຂອງອະກໍຣິທຶມໃດໜຶ່ງວ່າເວລາທີ່ໃຊ້ໃນການນປະມວນຜົນໜ້ອຍທີ່ສຸດແມ່ນເທົ່າໃດ.

$f(n) = \Omega(g(n))$ ກໍຕໍ່ເມື່ອມີຄ່າຄົງທີ່ $C>0$ ແລະ $n_0 >0$ ທີ່ເຮັດໃຫ້ $f(n) \geq cg(n)$ ສຳລັບທຸກໆຄ່າຂອງ n, ເມື່ອ $n \geq n_0$



ต์วอย่าง: ຊອກຫາຄ່າຂອງ Omega ຂອງຕຳລາຂອງເວລາຕຳລາຂອງເວລາຕໍ່ໄປນີ້ f(n) = 5n+17

แท้

5n+17 > 5n

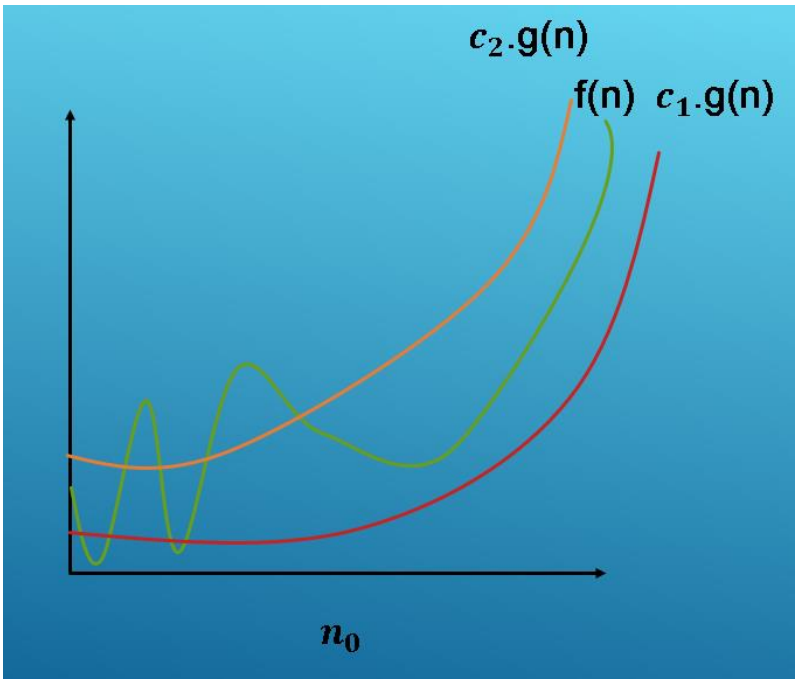
f(n) > 5n

ดั่งนั้ນ ເຮົາໄດ້ f(n) = Ω(n) ∀n, **ເມື່ອ** c=5

Theta

ເປັນການຊອກຫາຄ່າເຄິ່ງກາງຂອງຕຳລາຂອງເວລາ ໝາຍຄວາມວ່າຊອກຫາຂອບເທິງ ແລະ ຂອບລຸ່ມຂອງຕຳລາດັ່ງກ່າວ

f(n) = Θ(g(n)) **ກໍຕໍ່ເມື່ອມີຄ່າຄົງທີ່** c_1 >0, c_2 >0 **ແລະ** n_0 > 0 **ທີ່ເຮັດໃຫ້** $c_1 g(n) \leq f(n) \leq c_2 g(n)$ **ສຳລັບທຸກໆຄ່າຂອງ** n, **ເມື່ອ** $n \geq n_0$



ຕົວຢ່າງ: ຊອກຫາຄ່າຂອວ Theta ຂອງຕຳລາຂອງເວລາຕໍ່ໄປນີ້

$f(n)=9n+8$

ແກ້

ເມື່ອ $n \geq 8 \rightarrow 9n+8 \leq 10n$

$9n \leq f(n) \leq 10n$

$f(n)= \Theta(n)\ \forall n \geq 8$, ເມື່ອ $c_1 = 9$ ແລະ $c_2 = 10$

Little O

ແມ່ນມີລັກສະນະຄ້າຍຄືກັບສັນຍາລັກຂອງ Big O, ເປັນການຊອກຫາຄ່າຂອງຕຳລາ $f(n)$ ທີ່ມີຄ່ານ້ອຍກວ່າຕາລາ $g(n)$ ຢ່າງຂາດຕົວ $f(n) = o(g(n))$ ກໍຕໍ່ເມື່ອ $f(n) = O(g(n))$ ແລະ $f(n) \neq \Theta(g(n))$

ຕົວຢ່າງ: ຊອກຫາຄ່າຂອງ Little O ຂອງຕຳລາຂອງເວລາຕໍ່ໄປນີ້

$f(n) = 4n+5$

ແກ້

$f(n) = O(n^2)$ ແລະ $f(n) \neq \Theta(n^2)$ ເມື່ອ $n_0 > 25$

ສະນັ້ນ $f(n) = o(n^2)$

ຂໍຂອບໃຈ