**ວຽກກຸ່ມວິຊາ Algorithm**

**ກຸ່ມທີ່ 4**

ທ້າວ ກະເຢັ້ງເຮີ ຈົ່ງລື

ທ້າວ ກູ່ ວົງພະຈັນ

ທ້າວ ຄອນສະຫວັນ ແສງເລກ

ທ້າວ ສອນພາວັນ

ທ້າວ ສີນໄຊ ໄຊບຸນມີ

ທ້າວ ກູ້ວ່າງ ຊົວເຍ່ຍວື

**ບົດທີ 7 ການສ້າງ Heap**

1. **ຄວາມໝາຍຂອງ Heap ຫຼື Priority queue.**

Heap ເປັນໂຄງສ້າງຂໍ້ມູນທີ່ມີລັກສະນະພື້ນຖາມຂອງການເກັບມຽ້ນຂໍ້ມູນແບບ Binary Tree ແລະ ການເກັບມຽ້ນຂໍ້ມູນຕ້ອງລຽງໃຫ້ເຕັມແຕ່ລະຊັ້ນ.

* ປະເພດຂອງ Heap
* Max Heap ແມ່ນ Heap ທີ່ທຸກໂໜດຈະມີຂໍ້ມູນໃນພໍ່ແມ່ມີຄ່າຫຼາຍກ່ວາຫຼືເທົ່າກັບຂໍ້ມູນໃນ Subtree ທັງສອງຟາກ.
* Min Heap ແມ່ນ Heap ທີ່ທຸກໂໜດຈະມີຂໍ້ມູນໃນພໍ່ແມ່ມີຄ່ານ້ອຍກ່ວາຫຼືເທົ່າກັບຂໍ້ມູນໃນ Subtree ທັງສອງຟາກ.

1. **ນິຍາມຂອງ Heap**

Heap ເປັນໂຄງສ້າງຂໍ້ມູນແບບ Binary Tree ທີ່ມີຄຸນລັກສະນະດັ່ງນີ້:

1. ຕ້ອງເປັນ Complete Binary Tree ຫຼື Nearly Complete Binary Tree ເທົ່ານັ້ນ
2. ຕ້ອງມີລັກສະນະເປັນ Max Heap ຫຼື Min Heap.

Nearly Complete Binary Tree

Complete Binary Tree

Nearly Complete Binary Tree

Nearly Complete Binary Tree

Nearly Complete Binary Tree

Nearly Complete Binary Tree

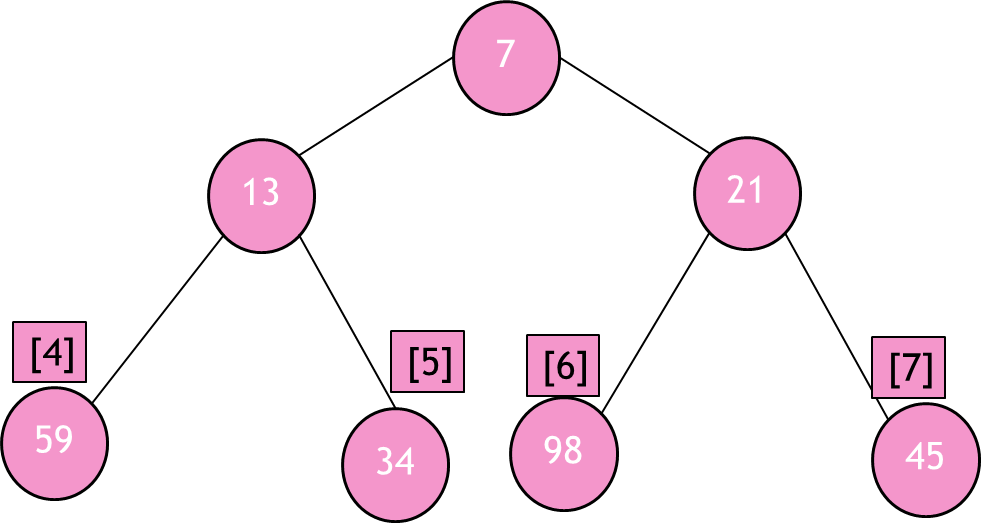
Tree

Complete Binary Tree Nearly Complete Binary Tree

1. **ໂຄງສ້າງຂໍ້ມູນ Heap**

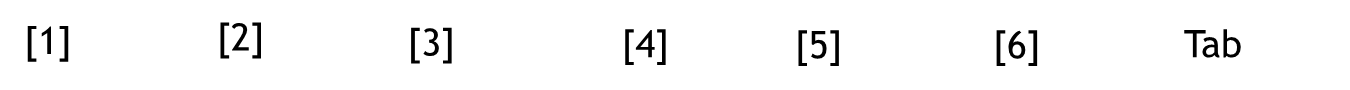
Heap ສາມາດສ້າງມາຈາກ Array ໜຶ່ງມິຕິ ໂດຍກຳໜົດຈຳນວນບ່ອນເກັບຂໍ້ມູນໃນ Array ໃຫ້ເທົ່າກັບຈໍານວນຂໍ້ມຸນທີ່ຕ້ອງການເກັບໃນ Heap ຫຼື ຈະສ້າງຈາກ Link list ກໍ່ໄດ້ ເຊິ່ງຈຳນວນໂໜດໃນ ກໍ່ຈະຕ້ອງມີຈຳນວນເທົ່າກັບຈຳນວນຂໍ້ມູນໃນ Heap ເຊັ່ນ: ການໃຊ້ Array ຫຼື Link list ເຮັດໃຫ້ການຈັດເກັບຂໍ້ມູນຂອງ Heap ບໍ່ສິ້ນເບືອງເນື້ອທີ່ໃນໜ່ວຍຄວາມຈໍາ ເເລະ ປະມວນຜົນໄດ້ໄວ.

ຕົວຢ່າງ: ການສ້າງ Heap ຈາກ Array ໜື່ງມີຕີ.



Graphical user interface

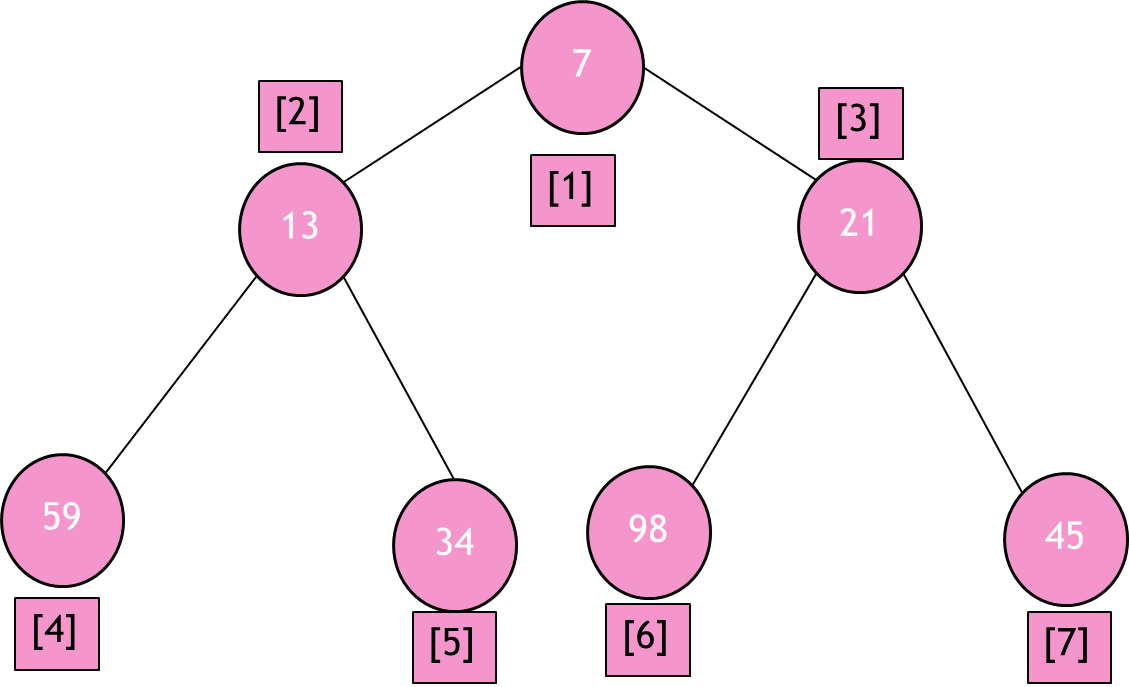
Description automatically generated



* ຄວາມສຳພັນຂອງ index ເເລະ ຕຳເເໜ່ງໃນ Heap

1. ສຳຫຼັບໂໜດທີ່ມີຄ່າ index ຂອງ Array ຕໍາເເໜ່ງທີ່ I.
2. ໂໜດລູກດ້ານຊ້າຍຈະມີຕຳເເໜ່ງຢູ່ທີ່ index: 2i.
3. ໂໜດລຸກດ້ານຂວາຈະມີຕຳເເໜ່ງຢູ່ທີ່ index: 2i+1
4. ສຳລັບໂໜດທີ່ມີຄ່າ Index ຕໍາເເໜ່ງ i ຈະມີຕໍາເເໜ່ງໂໜດພໍ່ເເມ່ຢູ່ທີ່ index:(i/2).
5. ສຳລັບໂໜດລູກທາງດ້ານຊ້າຍທີ່ມີຕຳເເໜ່ງ index: j, ຖ້າໂໜດລູກທາງດ້ານຂວາມີຂໍ້ມູນຈະມີເເຫ່ງ index: j+1, ໃນທາງກົງກັນຂ້າມ ຖ້າໂໜດລູກທາງດ້ານຂວາເກັບຢູ່ທີ່ index: k, ໂໜດລູກທາງດ້ານຊ້າຍຈະເກັບຕຳເເໜ່ງ index: k-1,
6. ຖ້າ Heap ດັ່ງກ່າວເປັນ Complete Binary tree ທີ່ມີຈໍານວນໂໜດທັງໝົດເທົ່າ n ໂໜດຕຳເເໜ່ງຂອງ Leaf node ຕົວທຳອິດຈະມີ index:(n/2)+1.
7. ຖ້າHeap ດັ່ງກ່າວເປັນ Complete Binary tree ທີ່ມີຈໍານວນໂໜດທັງໝົດເທົ່າ n ໂໜດຕຳເເໜ່ງຂອງໂໜດຕົວສຸດທ້າຍທ່ີ່ບໍ່ທັນເປັນ Leaf node index:(n/2).

ຕົວຢ່າງ: ການຊອກຫາຄວາມສຳພັນ.



1. **ການສ້າງ Heap**

ການດໍາເນີນການທີ່ສາມາດກະທໍາໄດ້ກັບໂຄງສ້າງຂໍ້ມູນຂອງ Heap ປະກອບດ້ວຍ:

1. ການສ້າງ Heap (Build Heap)
2. ການເພີ່ມໂໜດເຂົ້າໄປໃນ Heap (Insert Node)
3. ການລືບໂໜດອອກຈາກ Heap (Delete Root)
   1. ການສ້າງ Heap (Build Heap)

ເຮົາສາມາດສ້າງ Heap ໄດ້ໂດຍ 2 ແບບຄື: ການສ້າງ Heap ຈາກ Tree ແລະ ການສ້າງ Heap ຈາກ Node.

* + 1. ການສ້າງ Heap ຈາກ Tree

ເປັນການສ້າງຂື້ນມາຈາກ Tree ທີ່ມີຂໍ້ມູນເກັບໄວ້ແລ້ວ. ໂດຍ Tree ຈະປ່ຽນເປັນ Heap ໄດ້ຕ້ອງການມີຄຸນລັກສະນະເປັນ Complete Binary Tree ຫຼື Nearly Complete Binary Tree ຕາມກົດຂໍ້ທີ່ 1 ຂອງ Heap.

1. ຊອກຫາ Index ຂອງຕຳແໜ່ງສຸດທ້າຍເພື່ອທີ່ຈະຮູ້ວ່າຈຳນວນໂໜດທັງໜົດມີເທົ່າໃດ?
2. ຊອກຫາໂໜດພໍ່ແມ່ຂອງໂໜດສຸດທ້າຍ.
3. ປຽບທຽບຂອງມູນຂອງໂໜດພໍ່ແມ່, ໂໜດລູກດ້ານຊ້າຍ ແລະ ດ້ານຂວາໂດຍພີຈາລະນາດັ່ງນີ້:

* ຖ້າສ້າງ Max Heap ໃຫ້ປຽບທຽບຄ່າຂອງສາມໂໜດ, ໂໜດໃດມີຄ່າຫຼາຍກ່ວາໝູ່ໃຫ້ປ່ຽນເປັນໂໜດພໍ່ແມ່ຂອງກຸ່ມນັ້ນ.
* ຖ້າສ້າງ Min Heap ໃຫ້ປຽບທຽບຄ່າຂອງສາມໂໜດ, ໂໜດໃດມີຄ່ານ້ອຍກ່ວາໝູ່ໃຫ້ປ່ຽນເປັນໂໜດພໍ່ແມ່ຂອງກຸ່ມນັ້ນ.

1. ຖ້າມີການສັບປ່ຽນບ່ອນຂອງໂໜດພໍ່ແມ່ກັບໂນດລູກກັບດ້ານໃດດ້ານໜື່ງໃຫ້ຂ້າມໄປເຮັດຂ້າງຕອນທີ່ 5, ແຕ່ຖ້າມີການສັບປ່ຍນຈະປ່ຍນມາເປັນໂນດລູກແທນ.
2. ພີຈາລະນາໂໜດຖັດໄປໂດຍຫາຕຳແໜ່ງຈາກຂວາໄປຫາຊ້າຍຈາກລຸ່ມຂື້ນເທີງເມື່ອຫາຕຳແໜ່ງໄດ້ແລ້ວໃຫ້ກັບໄປເຮັດຂັ້ນຕອນ 3 ແລະ 4.
3. ເຮັດຊໍ້າຂັ້ນຕອນ 3,4 ແລະ 5 ໄປເລື້ອຍໆຈົນເຖີງ Root ໂໜດ.
4. ຖ້າ Root ໂໜດມີການສັບປ່ຽນຄ່າກັບໂໜດລູກດ້ານໃດດ້ານໜື່ງໃຫ້ປຽບທຽບຄ່າຕາມຫຼັກການຂໍ້ 3 ລົງມາເລື້ອຍໆຕາມໂໜດລູກດ້ານນັ້ນຈົນເຖີງລີບໂໜດ.

ຕົວຢ່າງ: ການສ້າງ Max Heap ຈາກ Tree

12

12

48

27

48

21

21

27

12

48

21

27

|  |  |  |  |
| --- | --- | --- | --- |
| 48 | 27 | 12 | 21 |

ຄ່າໃນ Array ຄື:

ຕົວຢ່າງ: ການສ້າງ Min Heap ຈາກ Tree

7

8

8

12

7

12

9

9

|  |  |  |  |
| --- | --- | --- | --- |
| 7 | 8 | 12 | 9 |

ໄດ້ຄ່າໃນ Array ຄື:

* + 1. ການສ້າງ Heap ຈາກ Node

ມີຂັ້ນຕອນດັ່ງນິ້:

1. ເພີ່ມໂໜດທຳອີດເຂົ້າໄປຊື່ງຈະເຮັດໃຫ້ເກີດ Heap ຂຶ້ນມາ.
2. ເພີ່ມໂໜດໃໝ່ເຂົ້າໄປໃນຕຳແໜ່ງຈາກເທີງລົງລຸ່ມ, ຈາກຊ້າຍຫາຂວາ.
3. ປຽບທຽບຄ່າຂອງໂໜດທີ່ເພີ່ມເຂົ້າໄປໃໝ່ກັບຄ່າຂອງໂໜດພໍ່ແມ່ ແລະ ພີ່ນ້ອງຂອງມັນດ້ວຍພີຈາລະນາດັ່ງນີ້:

3.1) ຖ້າຕ້ອງການສ້າງ Max Heap ໃຫ້ປຽບທຽບຄ່າຂອງ 3 ໂໜດຕົວໃດມີຄ່າຫຼາຍກ່ວາໝູ່ໃຫ້ປ່ຽນເປັນໂໜດພໍ່ແມ່.

3.2) ຖ້າຕ້ອງການສ້າງ Min Heap ໃຫ້ປຽບທຽບຄ່າຂອງ 3 ໂໜດຕົວໃດມີຄ່ານ້ອຍກ່ວາໝູ່ໃຫ້ປ່ຽນເປັນໂໜດພໍ່ແມ່.

4) ຖ້າບໍ່ມີການສັບປ່ຽນບ່ອນກັນໃຫ້ຂ້າມໄປເຮັດຂັ້ນຕອນທີ່ 5, ແຕ່ຖ້າມີ ແລະ ຕໍາແໜ່ງນັ້ນບໍ່ແມ່ນ Root ໃຫ້ປຽບທຽບຂໍ້ມູນຂອງລູກກັບຕົວໂໜດໃໝ່ຕາມຫຼັກການຂໍ້ 3 ໂດຍການກວດສອບແບບນັ້ນຂືນໄປເລື້ອຍໆຕາມເບື້ອງທີ່ມີການສັບປ່ຽນຄ່າ.

5) ເພີ່ມໂໜດເຂົ້າໄປແລ້ວເຮັດຕາມຂັ້ນຕອນ 2,3 ແລະ 4 ໄປເລື້ອຍໆຈົນກ່ວາເພີ່ມໂໜດເຂົ້າໝົດ.

ຕົວຢ່າງ: ການສ້າງ Max Heap ຈາກ Node

ໃຫ້: {4,9,3,12}

9

9

4+++++++++++++++++++++++++++++++++++++++++

4

3

4+++++++++++++++++++++++++++++++++++++++++

4+++++++++++++++++++++++++++++++++++++++++

9

12

9

9

3

9

3

12

3

4+++++++++++++++++++++++++++++++++++++++++

4+++++++++++++++++++++++++++++++++++++++++

4+++++++++++++++++++++++++++++++++++++++++

12

|  |  |  |  |
| --- | --- | --- | --- |
| 12 | 9 | 3 | 4 |

ຄ່າ Min Heap ທີ່ເກັບໄວ້ໃນ Array ຄື:

ຕົວຢ່າງ: ການສ້າງ Min Heap ຈາກ Node

ໃຫ້: {4,9,3,12}

3

4

4

4

9

4

3

9

9

12

|  |  |  |  |
| --- | --- | --- | --- |
| 3 | 9 | 4 | 12 |

ຄ່າ Min Heap ທີ່ເກັບໄວ້ໃນ Array ຄື:

* 1. ການເພີ່ມໂໜດເຂົ້າໃນ Heap

ໃຊ້ຫຼັກການດຽວກັນກັບການສ້າງ Heap ຈາກໂໜດດ້ວຍການເພີ່ມໂໜດທີ່ຕ້ອງການເຂົ້າໄປໃນລີບໂໜດຕົວສຸດທ້າຍແລ້ວສັບປ່ຽນບ່ອນໃຫ້ຮັກສາຄຸນລັກສະນະຂອງ Heap.

ການສ້າງ Min Heap ຈາກ ໂໜດ ໃຫ້: {4,9,3,12} ຫຼັງຈາກນັ້ນໃຫ້ເພີ່ມ 5 ເຂົ້າໄປ.

3

4

4

4

4

9

3

9

9

5

12

3

4

5

12

9

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 3 | 5 | 4 | 12 | 9 |

ຄ່າ Min Heap ທີ່ເກັບໄວ້ໃນ Array ຄື:

4.3 ການລືບ Root Node ອອກຈາກ Heap

ມີຂັ້ນຕອນດັ່ງນີ້:

1. ລືບຄ່າ Root Node ອອກໄປ.
2. ເອົາຄ່າໂໜດສຸດທ້າຍມາໃສ່ແທ່ນທີ່.
3. ປຽບທຽບຄ່າຂອງ Root Node ໃໝ່ກັບລູກທາງສອງດ້ານດັ່ງນີ້:

3.1) ຖ້າຕ້ອງ Max Heap ໃຫ້ປຽບທຽບຄ່າຂອງ 3 ໂໜດຕົວໃດມີຄ່າຫຼາຍກ່ວາໝູ່ຈະເປັນພໍ່ແມ່ຂອງກຸ່ມນັ້ນ.

3.2) ຖ້າຕ້ອງ Min Heap ໃຫ້ປຽບທຽບຄ່າຂອງ 3 ໂໜດຕົວໃດມີຄ່ານ້ອຍກ່ວາໝູ່ຈະເປັນພໍ່ແມ່ຂອງກຸ່ມນັ້ນ.

4) ຖ້າມີການສັບປ່ຽນບອ່ນກັນລະຫວ່າງໂໜດພໍ່ແມ່ ແລະ ໂໜດລູກດ້ານໃດໜື່ງໂໜດພໍ່ແມ່ເດີມຈະລົງມາເປັນໂໜດລູກແທນຖ້າຕຳແໜ່ງດັ່ງກ່າວຍັງບໍ່ແມ່ນລີບໂໜດໃຫ້ປຽບທຽບໂໜດພໍ່ແມ່ກັບໂໜດລູກຕາມວິທີໃນຂັ້ນຕອນ 3 ລົງມາເລື້ອຍໆຈົນເຖີງລີບໂໜດ.

ຕົວຢ່າງ: ການລືບ Root Node ອອກຈາກ Heap

9

ລືບ

3

4

5

4

5

4

5

12

9

12

9

12

4

5

5

4

9

9

12

12

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 4 | 9 | 5 | 12 |  |

ຄ່າ Min Heap ທີ່ເກັບໄວ້ໃນ Array ຄື:

**ບົດທີ 8 ການຄົ້ນຫາຂໍ້ມູນ (Searching)**

1. **ນິຍາມການຄົ້ນຫາຂໍ້ມູນ**

ການຄົ້ນຫາຂໍ້ມູນແມ່ນການຊອກຫາຂໍ້ມູນຕົວທີຕອ້ງການ ໃນໂຄງສ້າງຂໍ້ມູນໃດຫນື່ງວ່າມີຫລືບໍ ໂດຍຄຳຕອບທີໃດ້ຮັບຈະມີຢູ່ 2 ແບບທີເປັນໄປໃດ້ຄື:

* ສະເລັດ (Successful) ຊື່ງເປັນການຄົ້ນນຫາແລ້ວພົບຂໍ້ມູນຕົວທີ່ຕອ້ງການໃນໂຄງສ້າງຂໍ້ມູນດັ່ງກ່າວ
* ບໍສຳເລັດ (Unsuccessful) ຊື່ງເປັນການຄົ້ນນຫາແລ້ວບໍພົບຂໍ້ມູນຕົວທີ່ຕອ້ງການໃນໂຄງສ້າງຂໍ້ມູນດັ່ງກ່າວ
* Algorithm ໃນການຄົ້ນນຫາຂໍ້ມູນມີີຫລາຍແບບການທີຈະເລືອກໃຊ້ໂຕໃດນັ້ນສາມາດຕັດສີນໃຈດວ້ຍປັດໃຈເລົ່ານີ້:
* ລັກສະນະໂຄງສ້າງຂໍ້ມູນເປັນເປັນແບບ Array, Link, List, Tree…
* ລັກສະນະການເກັບຂໍ້ມູນໃນໂຄງສ້າງຂໍ້ມູນເປັນຂໍ້ມູນມີການລຽງລຳດັບຫລືຂໍ້ມູນບໍລຽງດັບ

ປະລິມານຂໍ້ມູນໃນໂຄງສ້າງຂໍ້ມູນຫລາຍ ຫລື ຫນອ້ຍສຳໃດ.

1. **ປະເພດຂອງ Algorithm ໃນການຄົ້ນຫາຂໍ້ມູນ**

ປະເພດຂອງ Algorithm ທີ່ໃຊ້ໃນການຄົ້ນຫາຂໍ້ມູນສາມາດຈຳແນກຕາມບ່ອນເກັບຂໍ້ມູນ ຫຼື ສື່ໃນການຈັດເກັບຂໍ້ມູນໄດ້ 2 ປະເພດຄື:

1. ການຄົ້ນຫາພາຍໃນ (Internal searching) ເປັນການຄົ້ນຫາຂໍ້ມູນທີ່ເກັບຢູ່ໃນໜ່ວຍຄວາມຈຳ.
2. ການຄົ້ນຫາພາຍນອກ (External Searching) ເປັນການຄົ້ນຫາຂໍ້ມູນທີ່ເກັບຢູ່ໃນສື່ທີ່ຄອມພີວເຕີສາມາດຕີດຕໍ່ໄດ້.
3. **ຫຼັກການຄົ້ນຫາຂໍ້ມູນ (Search principle)**

ການຄົ້ນຫາເປັນການດຳເນີນການທີ່ສຳຄັນຢ່າງໜື່ງ ໃນລະບົບງານຄອມພີວເຕີ້ ບໍ່ວ່າຂໍ້ມູນຈະຢູ່ໃນໜ່ວຍຄວາມຈຳໂດຍມີໂຄງສ້າງເປັນເເຖວລຳດັບ ເເລະ ລາຍການເເບບເສັ້ນຊື່ ດັ່ງທີ່ໄດ້ຮຽນຜ່ານມາເເລ້ວ ຈະສາມາດນຳມາປະມວນຜົນໄດ້ ເມື່ອເຮົາຕ້ອງການຂໍ້ມູນຕົວໃດ ມາປະມວນຜົນ ເຮົາຈະສັ່ງໄຫ້ຄອມພີວເຕີໄປຄົ້ນຫາຂໍ້ມູນຕົວນັ້ນ ເຊີ່ງການຄົ້ນຫາຈະຊ້າຫຼືໃວ ກໍຂື້ນຢູ່ກັບໂຄງສ້າງຂອງຂໍ້ມູນ ເເລະ ວິທີນການຄົ້ນຫາຂໍ້ມູນ ເຊີ່ງເຮົາຈຳເປັນຕ້ອງປ້ອນຂໍ້ຄວາມບາງຢ່າງ ຫຼື ຄ່າຂອງຂໍ້ມູນທີ່ຈະຄົ້ນຫາເເກ່ຄອມພີວເຕີ ເຊີ່ງເຮົາຈະເອີ້ນສີ່ງນັ້ນວ່າຄີເມື່ອຄອມພີວເຕີຮັບຄີທີ່ຈະຄົ້ນຫາມາປຽບທຽບກັບຄີຂອງຂໍ້ມູນທີ່ຕ້ອງການຄົ້ນຫາໂດຍລຳດັບຂອງການປຽບທຽບຂື້ນຢູ່ກັບ Algorithm ທີ່ໃຊ້ໃນການຄົ້ນຫາຂໍ້ມູນ.

1. **ການຄົ້ນຫາຂໍ້ມູນແບບ Linear Searching**

ການຄົ້ນຫາຂໍ້ມູນແບບ Linear Searching ເປັນການຄົ້ນຫາຂໍ້ມູນໂດຍການປຽບທຽບກັນລະຫວ່າງຂໍ້ມູນທີ່ຕ້ອງການຄົ້ນຫາ ແລະ ຂໍ້ມູນທີ່ຢູ່ໃນໂຄງສ້າງຂໍ້ມູນໂດຍຈະປຽບທຽບເທື່ອລະຕົວໄປເລື້ອຍໆຈົນພົບຂໍ້ມູນທີ່ຕ້ອງການແຕ່ຖ້າຫາກຄົ້ນຫາໄປຮອດຕຳແໜ່ງສຸດທ້າຍແລ້ວບໍ່ພົບຂໍ້ມູນທີ່ຕ້ອງການຜົນລັບທີ່ໄດ້ກໍຈະສະແດງບໍ່ພົບຂໍ້ມູນທີ່ຄົ້ນຫາແຕ່ຖ້າຫາກພົບຂໍ້ມູນທີ່ຕ້ອງການກໍ່ຈະເກັບຕຳແໜ່ງທີ່ຢູ່ຂອງຂໍ້ມູນນັ້ນແລ້ວເອົາອອກມາສະເເດງຜົນ.

ແຕ່ລະ Algorithm ທີ່ມີການຄົ້ນຫາແບບ Linear Searching ຈະມີເທັກນິກຕ່າງໆສະເພາະຕົວໃນການເລືອກກຸ່ມຂໍ້ມູນເພື່ອເອົາມາປຽບທຽບໃນແຕ່ລະຄັ້ງເຊິ່ງແບ່ງອອກເປັນຫຼາຍເທັກນິກດ້ວຍກັນ, ແຕ່ລະເທັກນິກກໍ່ຈະເໜາະສົມກັບລັກສະນະ ແລະ ຄຸນສົມບັດຂອງໂຄງສ້າງຂໍ້ມູນທີ່ແຕກຕ່າງກັນ.

ຕົວຢ່າງ:

ຕ້ອງການຄົ້ນຫາຂໍ້ມູນ 23

[1] [2] [3] [4] [5] [6] [7] [8] [9] [10]

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 7 | 10 | 23 | 20 | 13 | 3 | 18 | 32 | 29 | 17 |

* ຈາກຕຳແໜ່ງທີ່ 1 ຂໍ້ມູນທີ່ໄດ້ຄື 7 ເມື່ອປຽບທຽບກັບຂໍ້ມູນທີ່ໄດ້ກັບຄ່າທີ່ຕ້ອງການຄົ້ນຫາແມ່່ນ 7 ≠ 23 ການເຮັດວຽກຈະຖັດໄປຄົ້ນຫາອີກຕຳແໜ່ງຕໍ່ໄປ.

[1] [2] [3] [4] [5] [6] [7] [8] [9] [10]

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 7 | 10 | 23 | 20 | 13 | 3 | 18 | 32 | 29 | 17 |

* ຈາກຕຳແໜ່ງທີ່ 2 ຂໍ້ມູນທີ່ໄດ້ຄື 10 ເມື່ອປຽບທຽບກັບຂໍ້ມູນທີ່ໄດ້ກັບຄ່າທີ່ຕ້ອງການຄົ້ນຫາແມ່່ນ 10 ≠ 23 ການເຮັດວຽກຈະຖັດໄປຄົ້ນຫາອີກຕຳແໜ່ງຕໍ່ໄປ.

[1] [2] [3] [4] [5] [6] [7] [8] [9] [10]

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 7 | 10 | 23 | 20 | 13 | 3 | 18 | 32 | 29 | 17 |

* ຈາກຕຳແໜ່ງທີ່ 3 ຂໍ້ມູນທີ່ໄດ້ຄື 23 ເມື່ອປຽບທຽບກັບຂໍ້ມູນທີ່ໄດ້ກັບຄ່າທີ່ຕ້ອງການຄົ້ນຫາແມ່່ນ 23 ≠ 23 ຈະຈົບການເຮັດວຽກ.

**ຜົນລັບທີ່ໄດ້ເມື່ອຈົບການເຮັດວຽກຄື:**

ຄົ້ນພົບຂໍ້ມູນ 23 ໃນຕໍາແໜ່ງທີ່ 3

ໂດຍເຮັດວຽກທັງໝົດ 3 ຮອບ

1. **ການຄົ້ນຫາເເບບລຽງຕາມລຳດັບ (Sequential Search)**

Sequential Search ການຄົ້ນຫາຂໍ້ມູນ ເເບບລຳດັບຂັ້ນຕອນ ໂດຍຈະຄົ້ນຫາຕັ້ງເເຕ່ໂຕທຳອິດ ໄປເທື່ອລະໂຕ ຈົນກວ່າຈະພົບຂໍ້ມູນທີ່ຕ້ອງການ ຫຼືປຽບທຽບໄປຈົນເຖີງໂຕສຸດທ້າຍການຄົ້ນຫາດ້ວຍວິທີນີ້ເປັນວິທີທີ່ງ່າຍທີ່ສຸດ algorithm ໃນການຄົ້ນຫາທີ່ບໍ່ຊັບຊ້ອນສາມາດໃຊ້ກັບທັງຂໍ້ມູນທີ່ລຽງລຳດັບເເລ້ວ ຫຼື ຂໍ້ມູນທີ່ຍັງບໍ່ໄດ້ລຽງລຳດັບກໍໄດ້ເຊັ່ນກັນໂດຍຜົນຈາກການຄົ້ນຫາຂໍ້ມູນຈະມີຄວາມເປັນໄປໄດ້ 2 ເເບບຄື:

1. ພົບຕຳເເຫນ່ງຂອງຂໍ້ມູນທີ່ຕ້ອງການ(Successful Search).
2. ບໍ່ພົບຂໍ້ມູນທີ່ຕ້ອງການ (Unsuccessful Search).

ຕົວຢ່າງ: ການຄົ້ນຫາຂໍ້ມູນເເບບ Sequential Search ຈາກໂຄງສ້າງຂໍ້ມູນຕໍ່ເເບບ Array ເຊີ່ງເຮົາຈະຄົ້ນຫາຊຸດຂໍ້ມູນທີ່ເປັນເລກ 5:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **2** | **1** | **4** | **5** | **3** |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **2** | **1** | **4** | **5** | **3** |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **2** | **1** | **4** | **5** | **3** |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **2** | **1** | **4** | **5** | **3** |

****

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **2** | **1** | **4** | **5** | **3** |

****

1. **ການຄົ້ນຫາແບບດັດສະນີແບບລຽງລຳດັບ (Indexed Sequential Search).**

ເປັນການຄົ້ນຫາຂໍ້ມູນ ທີ່ໃຊ້ກັບຂໍ້ມູນທີ່ ໄດ້ລຽງລຳດັບຂໍ້ມູນເເລ້ວ ເຊີ່ງມີລັກສະນະການເຮັດວຽກຄ້າຍຄືກັບເທັກນິກ Sequential Searching ມັກໃຊ້ກັບໂຄງສ້າງຂໍ້ມູນເເບບ Array ເເລະ Link List ໂດຍເເບາງຂໍ້ມູນອອກເປັນຊ່ວງເເລ້ວສ້າງຕາຕະລາງ Index ຂື້ນມາເພື່ອເຮັດໜ້າທີ່ໃນການເກັບຂໍ້ມູນຕົວສຸດທ້າຍໃນເເຕ່ລະຊ່ວງ ເເລະ ມີ Pointer ຊີ້ໄປຫາຕຳເເໜ່ງຂອງຂໍ້ມູນເຫຼົ່ານັ້ນ ໃນໂຄງສ້າງຂໍ້ມູນເມື່ອຕ້ອງການຄົ້ນຫາຂໍ້ມູນກໍກວດສອບກ່ອນວ່າ ຂໍ້ມູນທີ່ຕ້ອງການຢູ່ໃນຊ່ວງໃດ ຈາກນັ້ນກໍປຽບທຽບຂໍ້ມູນທີ່ຕ້ອງການກັບຂໍ້ມູນໃນຊ່ວງນັ້ນເທົ່ານັ້ນ ບໍ່ໄດ້ປຽບທຽບທັງໝົດຄືກັບ Sequential Searching.

ເທັກນິກດັ່ງກ່າວນີ້ ສາມາດໃຊ້ງານໄດ້ດີກັບໂຄງສ້າງຂໍ້ມູນທີ່ມີປະລິມານຂໍ້ມູນຫຼາຍ ເເຕ່ໂຄງສ້າງຖານຂໍ້ມູນຕ້ອງໄດ້ລຽງລຳດັບໄວ້ເເລ້ວ ອາດຈະເປັນການລຽງເເຕ່ນ້ອນຫາໃຫຍ່ ຫຼື ໃຫຍ່ຫານ້ອຍກໍໄດ້. ເປັນລັກສະນະໜື່ງທີ່ມີລັກສະນະການເຮັດວຽກທີ່ລຽບງ່າຍ ເເລະ ມີປະສິດທິພາບດີ ໃນໂຄງສ້າງທີ່ມີຂໍ້ມູນຫຼາຍ.

ຕາຕະລາງindex Data / Array

|  |  |
| --- | --- |
| 2 | 1 |
| 14 | 2 |
| 12 | 3 |
| 14 | 4 |
| 20 | 5 |
| 21 | 6 |
| 23 | 7 |
| 26 | 8 |
| 27 | 9 |
| 30 | 10 |
| 35 | 11 |
| 42 | 12 |

K\_index ptr\_index

|  |  |
| --- | --- |
| 14 |  |
| 23 |  |
| 42 |  |

ຮູບ: ລັກສະນະການຈັດການໂຄງສ້າງຂໍ້ມູນສຳລັບເທັກນິກເເບບ index Sequential Searching

ໃນການໃຊ້ງານໂຕຈີງ ຂໍ້ມູນອາດຈະບໍ່ເເມ່ນໂຕເລກສະເໝີໄປ ຈື່ງຈຳເປັນຕ້ອງກຳໜົດໂຕເລກເພື່ອອ້າງອີງເເທນຂໍ້ມູນເເຕ່ລະຕົວດັ່ງນີ້:

Key / Data

|  |  |
| --- | --- |
| 1 | ທ ກອນ |
| 2 | ນ ຂັນ |
| 3 | ທ ຈິດ |
| 4 | ທ ດາ |
| 5 | ທ ພອນ |
| 6 | ນ ສີ |
| 7 | ນ ຈັນ |

ຈະມີໂຕເລກລຳດັບເພື່ອເເທນຂໍ້ມູນ

ທີ່ບໍ່ເເມ່ນໂຕເລກດັ່ງນີ້:

1. **ການຄົ້ນຫາແບບໄບນາຣີ (binary search)**

ການຄົ້ນຫາຂໍ້ມູນດອຍເທັກນີກ binary searchເປັນເທັກທີ່ຄຸນນະພາບສູງ ແລະ ສາມາດໃຊ້ງານກັບໂຄງສ້າງຂໍ້ມູນໃດ້ຫຼາຍແບບເຊັ່ນ: array, link list, binary tree ໂດຍທີ່ໂຄງສາ້ງຂໍ້ມຸນນັ້ນຈະຕອ້ງເກັບຂໍ້ມຸນໄວ້ແບບລຽງລຳດັບ ແລະ ສາມາດໃຊ້ງານໃດ້ກັບໂຄງສ້າງຂໍ້ມຸນທີ່ມີປະລິມານຂໍ້ມູນຫຼວງຫຼາຍດຽວກັນກັບເທັກນີກ indexed sequential search.

ການຄົ້ນຫາແບບໄບນາຣີ binary search ເປັນການປຽບທຽບຂໍ້ມູນທີ່ຕອ້ງກັນກັບຄ່າເຄີ່ງກາງໃນໂຄງສາ້ງຂໍ້ມູນວ່າແມ່ນຕົວທີ່ຕອ້ງກັນ ຫຼື ບໍ ຖ້າບໍ່ແມ່ນຈະພິຈາລະນາວ່າ target ມີຄ່າໃຫຍ່ກວ່າຫຼື ນອ້ຍກວ່າ, ຖ້ານອ້ຍກວ່າຈະຕັດທັງຫຼັງຂື້ນຖີ້ນໄປ, ຖ້າໃຫຍ່ກວ່າຈະຕັດທັງໜ້າຂື້ນຖີ້ນໄປ, ແລ້ວປຽບທຽບກັບຕົວເຄີ່ງກາງຂອງສວ່ນໄໝ່ນັ້ນ ໄປເລື້ອຍໆຈົນພົບຕົວທີ່ຕອ້ງກັນ ຫຼື ຈັນໝົດຂໍ້ມູນ.

ການຄົ້ນຫາດວ້ຍເທັກນີກນີ້ເປັນການແກ້ຈຸດອອ່ນຂອງການຄົ້ນຫາຂໍ້ມູນເທັກນີດແບບ sequential search ເພາະຖ້າຫາກຂໍ້ມູນທີ່ຕອ້ງການ ຢູ່ທັງຫຼັງສຸດເຮົາຈະຕອ້ງເສຍເວລາໃນການຄົ້ນທາງຫຼາຍ, ແຕ່ການຄົ້ນຫາແບບໄບນາຣີເປັນການຄົ້ນຫາທີ່ໃຊ້ໃດ້ກັບຊຸດຂອງຂໍ້ມູນທີ່ມີການລຽງລຳດັບໄວ້ແລ້ວເທົ່ານັ້ນ ເຊີ່ງສາມາດຫຼຸດຈຳນວນຂໍ້ມູນລົງໃດ້ເທື່ອລະປະມານເຄີ່ງໜື່ງ ຂອງປະລິມານຂໍ້ມູນທີ່ເຫຼືອ

ດັ່ງນັ້ນ ຕົວປຽ່ງທີ່ໃຊ້ໃນການເຮັດຈະມີຢູ່ 3 ຕົວປຽ່ນ, ປະກອດດວ້ຍຕົວປຽ່ນ low ແມ່ນຄ່າຕຳສຸດ, ຕົວປຽ່ນ mid ແມ່ນຄ່າເຄີ່ງກາງ ແລະ ຕົວປຽ່ນ high ແມ່ນສູງສຸດເຊີ່ງທັງສາມຄ່ານີ້ເປັນຕົວແທນຄ່າຕຳແໜ່ງ index ໃນ array ທີ່ຈະເອົາພິຈາລະນາ ດັ່ງຮູບ:

Low high

|  |  |  |
| --- | --- | --- |
| < target | Mid | >target |

ສວ່ນການຄຳນວນຫາຄ່າໃນແຕ່ລະຮອດນັ້ນ ສາມາດຄຳນວນໃດ້ຈາກສູດຂັ້ນລຸ່ມ ແລະ ເນື່ອງຜົນທີ່ໃດ້ຈາກການຄຳນວນ ບາງຄັ້ງອາດຈະມີຄ່າເສດເກີດຂື້ນ(ເປັນເລກເສດ) ຖ້າເປັນດັ່ງນັ້ນໃຫ້ປັດເສດຖີ້ມ ເອົາແຕ່ຖວ້ນ

Mid= (Low + high) / 2

ຂັ້ນຕອນການເຮັດວຽກຂອງ binary search algorithm

1. ກຳນົດຄ່າເລີ່ມຕົ້ນໃຫ້ກັບ low ແລະ high ໂດຍ low=1 and high=ຂະໜາດຂອງໂຄງສາ້ງຂໍ້ມູນ
2. ຄຳນວນຄ່າ Mid ຈາກ low ແລະ high
3. ປຽບທຽບຂໍ້ມູນໃນຕຳແແໜ່ງເຄີ່ງກາງຂອງໂຄງສ້າງຂໍ້ມູນກັບ target ວ່າເທົ່າກັນບໍ
4. ຖ້າເທົ່າກັນສະແດງວ່າສຳເລັບແລ້ວມີ target ຢູ່ໃນໂຄງສ້າງຂໍ້ມູນນີ້ ແລະ ສະແດງຜົນອອກມາວ່າພົບ target ແລ້ວເກັບຕຳແໜ່ງທີ່ພົບໄວ້ ແລະ ຈົບການຄົ້ນຫາ
5. ຖ້າບໍເທົ່າກັນຈະປຽບທຽບວ່າຄ່າ target ມີຄ່າຫຼາຍກວ່າຫຼື ໜອ້ຍກວ່າ ຄ່າເຄີ່ງກາງ
6. ຖ້າໜອ້ຍກວ່າກຳນົດໃຫ້ high= mid-1 ສວ່ນ low ຮັກສາໄວ້ຄືເກົ່າ
7. ຖ້າຫຼາຍກວ່າກຳນົດໃຫ້ low=mid-1 ຮັກສາ high ໄວ້ຄືເກົ່າ
8. ກວດສອບວ່າ Low ມີຄ່າ >high ບໍ
9. ຖ້າບໍແມ່ນກັບໄປເຮັດວຽກຂັ້ນຕອນທີ່ 2 ຄືນອີກ ເຮັດແນວນັ້ນເລື້ອຍໆຈົນກວ່າອອກຈາກ algorithm ໄດ້
10. ຖ້າແມ່ນສະແດງວ່າບໍ່ພົບ ບໍ່ສຳເລັດ ໃຫ້ສະແດງຜົນອອກມາວ່າ ບໍ່ຂໍ້ມູນທີ່ຕອ້ງການຢູ່ໃນໂຄງສ້າງຂໍ້ມູນນນີ້ ແລ້ວຈົບການຄົ້ນຫາາ

ດັັ່ງນີ້ ຈາກ algorithm ຈະເຫັນໃດ້ວ່າສາມາດອອກຈາກ algorithm binary search ໃດ້ກໍ່ເມື່ອ

* **ສຳເລັບ** ແມ່ນການຄົ້ນພົບຂໍ້ມູນທີ່ຕອ້ງການໃນໂຄງສ້າງຂໍ້ມູນ ແລະ ມີຜົນໃດ້ຮັບອອກມາເປັນຕຳ

ແໜ່ງທ່ີ່ຢູ່ທີ່ພົບຂໍ້ມູນ ຊີ່ງເຮົາສາມາດເອົາຕຳແໜ່ງທີ່ຢູ່ນັ້ນໄປກະທຳຢ່າງໃດຢ່າງໜື່ງ່ໃດ້ເຊັ່ນ: ອາດຈະໄປແ້ໄຂຂໍ້ມູນ ຫຼື ລືບຂໍ້ມູນເປັນ.

* **ບໍ່ສຳເລັບ** ແມ່ນ array [mid] ບໍ່ເທົ່າກັບ target ເລີຍ ຈົນກະທັ້ງຄ່າ low ມີຄ່າຫຼາຍກວ່າ high.

ຕົວຢ່າງ: 1 ການສະແດງການຄົ້ນຫາຂໍ້ມູນແບບໄບນາຣີ ໂດຍສົມມຸດວ່າໃຫ້ຂໍ້ມູນຊຸດໜື່ງດັ່ງນີ້

1 2 3 4 5 6 7 8 9

ຖ້າຕອ້ງການຫາ 4 ຈາກຂໍ້ມູນຊຸດນີ້ມີລຳດັບຂັ້ນຕອນດັ່ງນີ້

**ຂັ້ນທີ່ 1** ນຳ 4 ໄປປຽບທຽບກັບຄ່າກາງຂອງຂໍ້ມູນຊຸດທີ່ 5 ປະກົດວ່າ 4 ນອ້ຍກວ່າ 5

**ຂັ້ນທີ່ 2** ນຳ 4 ໄປປຽບທຽບກັບຄ່າກາງຂອງຄີ່ງຊຸດທີ່ມີຄ່ານອ້ຍເຊີ່ງປະກອບດອ້ຍ 1 2 3 4 5 ຄ່າກາງຄື 3 ປະກົດວ່າ 4 ຫຼາຍກວ່າ 3

**ຂັ້ນທີ່ 3** ນຳ 4 ໄປປຽບທຽບກັບຄ່າກາງຂອງຄີ່ງຊຸດທີ່ມີຄ່າຫຼາຍເຊີ່ງປະກອບດອ້ຍ 4 5 ຄ່າກາງຄື 4 ປະກົດວ່າ 4 ເທົ່າກັບ 4 ຄົ້ນພົບຂໍ້ມູນທີ່ຕອ້ງການ

ຖ້າມີຂໍ້ມູມ n ລາຍການຈຳນວນເທື່ອຂອງການປຽບທຽບນອ້ຍທີ່ສຸດເທົ່າກັບ 1 ຄັ້ງ.