**ບົດລາຍງານ**

**ວິຊາ:ໂຄງສ້າງຂໍ້ມູນແລະຂັ້ນຕອນວິທີ**

**ສອນໂດຍ:ອຈ ຈິດດາວັນ**

**ກຸ່ມ III**

**1.ພຣະ ສຸກອານັນ ທຳມະຈັກ**

**2.ສ.ນ ສະຖາພອນ ພັນທະນິສິດ (ບໍ່ໄດ້ລາຍງານ)**

**3.ທ້າວ ສຸລີນ ເກດຕິພູມ**

**4.ທ້າວ ສີນນຸສອນ ຄຳສຸກຖາວົງ**

**5.ທ້າວ ສະຖຽນ ວໍລະກຸມມານ**

**6.ທ້າວ ສຸກສະຫວັນ ບັນນະວົງ**

**7.ທ້າວ ພອນອານົງ ສູນທະນູວັດ**

**8.ທ້າວ ຈັນດີ ຈະທໍ່ຊົງ**

**ລາຍງານກ່ຽວກັບ**

**ໂຄ້ງສ້າງຂໍ້ມູນແບບຕົ້ນໄມ້ (Tree Data Structure)**

ໂຄ້ງສ້າງຂໍ້ມູນແບບຕົ້ນໄມ້ (Tree Data Structure) ຫຼືເອີ້ນວ່າ (Tree) ເປັນໂຄງສ້າງຂໍ້ມູນຮູບອບບຫນຶ່ງໃນລັກສະນະໂຄງສ້າງຂໍ້ມູນຊະນິດ (Non-Linear) ທີ່ສາມາດນຳໄປໃຊ້ໃນການຈັດການ ກັບຂໍ້ມູນໄດ້ຢ່າງປະສິດທິພາບ ສະມາໍິກແຕ່ລະໂຕໃນຕົ້ນໄມ້ສາມາດເຊື່ອມໂຍງໄປຍັງສະມາຊິກໂຕທັດໄປ (Successor) ໄດ້ຫລາຍກວ່າໜຶ່ງໂຕ ແລະ ເຊື່ອມໂຍງເຖິງກັນໃນລັກສະນະເປັນລະດັບຄ້າຍໆກັບສະແຕັກ ກິ່ງກ້ານສາຂາອອກໄປຂອງຕົ້ນໄມ້ ດັ່ງນັ້ນຄວາມສຳພັນຂອງສະມາຊິກຂໍ້ມູນໃນຕົ້ນໄມ້ ຈຶ່ງມີລັກສະນະລຳດັບຊັ້ນ (Hierarchical Relationship) ຄື ມີການເຊື່ອມໂຍງຂອງແຕ່ລະໂນດເປັນແບບທາງດຽວຈາກເທິງລົງລຸ່ມ.

ໂຄງສ້າງຂໍ້ມູນທຮີປະກອບດ້ວຍໂນດ ສຳລັບຈັດເກັບຂໍ້ມູນ ແລະ ກິ່ງຫຼືເສັ້ນທີ່ເຊື່ອມໂຍງໂນດເຂົາດ້ວຍກັນເອີ້ນວ່າ: ບຣານ (Branch) ໂນດຕ່າງໃນທຣີຈະຢູ່ໃນລະດັບທີ່ຕ່າງກັນດ້ວຍ້ລີມຕົ້ຍຈາກໂນດທຳອິດຂອງທຣີ ທີ່ເອີ້ນວ່າ ໂນດຮາກ (Root) ຈະຢູ່ໃນລະດັບທຳອິດ (Level 0) ໂນດລູກໆ (Children) ຂອງໂນດຮາກຈະຢູ່ມນລະດັບທີ່ 1 (Level 1) ສ່ວນໂນດລູກໆຂອງໂນດໃນລະດບທີ່ 1 ຈະຢູ່ໃນລະດັບທີ່ 2 (Level 2) ທຸກໂນດມນທຣີຈະມີຄວາມສຳພັນກັນໃນລັກສະນະເມືອນຄອບຄົວດຽວກັນນັ້ນຄື ແຕ່ລະໂນດຈະເຊື່ອມໂຍງໄປຍັງໂນດພໍ່ (Parent) ເຊິ່ງຢູ່ລະດັບທີເໜືອກວ່າໄດ້ 1 ໂນດ ແຕ່ລະໂນດຈະເຊື່ອມໄປຍັງໂນດລູກ (Child) ທີ່ຢູ່ລະັບຕຳກວ່າຫຼາຍໂນດ ໂນດລູກທັງ 2 ໂນດ (Children) ຂືີ້ນໄປທີ່ເຊື່ອມໂຍງໄປຍັງໂນດພໍ່ດຽວກັນ ຈະມີຄວາມສຳພັນກັນໃນລັກສະນະເປັນໂນດພີ່ນ້ອງກັນ (Siblings) ດັ່ງຮູບທີ່ 1.0 ເປັນຕົວຢ່າງໂຄງສ້າງຂໍ້ມູນທຣິທີ່ມີໂນດ A ເປັນໂນດຮາກໂນດ B E ແລະ F ເປັນໂນດພີ່ນ້ອງກັນເພາະຕ່າງກໍ່ເປັນໂນດຂອງໂນດ A ເມື່ອເພີ່ມລະດັບຂອງທຣີໄປເລື້ອຍໆ ກໍ່ຈະທີໂນດຂອງລູກຫຼານເພີ່ມຂື້ນ ດັ່ງນັ້ນໂນດ F I ແລະ J ເປັນໂນລູກຫຼານ ( Descendants) ຂອງໂນດ A ໂດຍມີສະຖານະເປັນໂນດລູກ (Child Node) ໂນດຫຼານ (Grandchild Node) ແລະໂນດເຫຼນ (Grand-grandchild Node) ຂອງໂນດ A ຕາມລຳດັບ ຫຼືເອີ້ນວ່າຮ່ວມກັນວ່າໂນດລູກຫຼານ ( Descendants) ຂອງໂນດ A ແລະ ໃນທາງກັບກັນໂນດ I F ແລະ A ເປັນໂນດບັດພະບຸລຸດ ( Ancestors) ຂອງໂນດ J ໂດຍມີສະຖານະເປັນໂນດພໍ່ (Parent) ໂນດປູຍ່າ-ຕາຍາຍ (Grandparent) ແລະ ໂນດທວດ (Grand-grandparent) ຂອງໂນດ J ຕາມລຳດັບ

**B**

**F**

**D**

**C**

**H**

**I**

**G**

**E**

**A**

**J**

**Child Node**

**ຂອງໂນດ**

**A**

**Level 0**

**Level 1**

**Gran**

**dchild Node**

**ຂອງໂນດ**

**A**

**Level 2**

**ໂນດຮາກຂອງທຣີ**

**Grand**

**-**

**grandchild Node**

**ຂອງໂນດ**

**A**

**Level 3**

ຮູບທີ່ 1.0 ຕົວຢ່າງໂຄງສ້າງຂໍ້ມູນຂອງທຣີ

ຈາກຮູຍທີ 1.0 ສາມາດອະທິບາຍລັກສະນະ ແລະ ຄຸນສົມບັດພື້ນຖານຂອງໂຄງສ້າງໄບນາລຣີໄດ້ດັ່ງນີ້:

1. ໂນດ A ເປັນໂນດຮາກຂອງທຣີ
2. ທຸກໂນດ (ຍົກເວັ້ນຮາກ) ຈະມີສະມາຊິກໂຕກ່ອນໜ້າ (Predecessor) ພຽງ 1 ໂນດດັ່ງນັ້ນໂນດ B ມີໂນດ A ເປັນ (Predecessor) ເປັນຕົ້ນ ແຕ່ສະມາຊິກໂຕຖັດໄປ (Successor) ອາດທີ 0 ຫຼື 1 ຫຼືຫຼາຍກວ່າໜຶ່ງໂນດກໍ່ໄດ້ດັ່ງນັ້ນໂນດ B ມິໂນດ E F ແລະ G ເປັນ Successor
3. ໂນດພໍ່ເປັນໂນດທີ່ມີ Successor ດັ່ງນັ້ນໂນດ B ຄື່ໂນດພໍ່ຂອງໂນດ E F ແລະ G
4. ໂນດລູກເປັນໂນດທີ່ມີ Predecessor ດັ່ງນັ້ນ B ຄືໂນດລູກຂອງໂນດ A
5. {B,E,F} ຄືກຸ່ມໂນດພີ່ນ້ອງກັນທີ່ມີໂນດ A ເປັນໂນດພໍ່ ເຊັ່ນດຽວກັບ {C,D} ແລະ {G,H,I} ເປັນກຸ່ມໂນດພີ່ນ້ອງທີ່ມີ B ແລະ F ເປັນໂນດພໍ່ຕາມລຳດັບ
6. ໂນດກິ່ງ (Branch Node) ຫຼືໂນດພາຍໃນ (Internal Node) ຄືໂນດໃດໆ ທີ່ມີີເສັ້ນເຊື່ອມໄປຍັງໂນດລູກໄດ້ແກ່ A B F ແລະ I
7. ໂນດພາຍນອກ (External Node) ຫຼືໂນດໃບ (Leaves) ຄືໂນດໃດໆ ທີ່ບໍ່ມີເສັ້ນເຊື່ອມໄປຍັງໂນດລູກໄດ້ແກ່ C D E G H ແລະ J
8. ຈຳນວນບຮານຂອງໂນດເອີ້ນວ່າ: ດີກຣີ (Degree) ເຊັ່ນ:

* ດີກຣີທັງໝົດຂອງທຣີຄື 9
* ດີກຣີຂອງໂນດ F ຄື 3
* ດີກຣີຂອງໂນດໃບ (Leaf Node) ຈະເທົ່າກັບ 0 ໄດ້ແກ່ໂນດ C,D,E,G,H ແລະ J

1. ເສັ້ນທາງ (Path) ຈາກໂນດ 1 ເຖີງໂນດ K ຄືລຳດັບຂອງໂນດ n1 , n2 ,…, nk ເມື່ອ ຄືໂນດພໍ່ຂອງໂນດ ແລະ 1 ≤ i < k
2. ຄວາມເລີກ (Depth) ຂອງໂນດຄືຄວາມຍາວຂອງ Path ຈາກໂນດຮາກເຖີງໂນດນັ້ນເຊັ່ນ

* ຄວາມເລີກຂອງໂນດ A ເທົ່າກັບ 0
* ຄວາມເລີກຂອງໂນດ J ເທົ່າກັບ 3

1. ຄວາມສູງ (Height) ຂອງໂນດຄືຄວາມຍາວຂອງ Path ຈາກໂນດເຖີງໂນດໃບ

* ຄວາມສູງຂອງໂນດ B ເທົ່າກັບ 1 ແລະ ຄວາມສູງຂອງໂນດ C ເທົ່າກັບ 0
* ຄວາມສູງຂອງໂນດ A ເທົ່າກັບ 3 ເຊິ່ງໂນດ A ຄືໂນດຮາກຈຶ່ງເທົ່າກັບຄວາມສູງຂອງໂນດທຣີ

1. ທຣີວ່າງ (Empty tree) ຄືໂນດທີ່ບໍ່ມີສະມາຊິກ ຫຼື ຈຳນວນໂນດໃນທຣີເທົ່າກັບ 0 ການດຳເນີນການພື້ນຖານກັບຊະນິດຂໍ້ມູນນາມທຣີ (ADT Tree Operations) ປິກອບດ້ວຍ

**ການດຳເນີນງານໜ້າທີ**

Size() ນັບ ແລະ ຄື່ນຄ່າຈຳນວນໂນດທີ່ຈັດເກັບໃນທຣີ

isEmpty() ກວດສອບວ່າທຣີວ່າງຫຼືບໍ່ ໂດຍຈະຄື່ນຄ່າຈິ່ງເມື່ອທຣີວ່າງ ແລະ ຄື່ນຄ່າບໍ່ຈິ່ງເມື່ອມີຂໍ້ມູນຈັດເກັບ

ໃນທຣີ

p.getItem() ຄົ້ນຄື່ນຄ່າຂໍ້ມູນທີ່ຈັດເກັບໃນໂນດ P ເມື່ອ P ຄືສະມາຊິກຂອງທຣີ

p.swapItem(q) ສະລັບຄ່າຂໍ້ມູນລະຫວ່າງໂນດ Q ແລະ Q ເມື່ອ Q ແລະ Q ຄືສະມາຊິກຂອງທຣີ

p.replaceItem(item) ແທ່ນຄ່າຂໍ້ມູນເກົ່າທີ່ຈັດເກັບໃນໂນດ P ດ້ວຍຂໍ້ມູນໃໝ່ (item)

p.isRoot() ກວດສອບວ່າໂນດ P ເປັນໂນດຮາກຂອງທຣີຫຼືບໍ່

p.isExternal() ກວດສອບວ່າໂນດ P ມີສະຖານະເປັນໂນພາຍນອກຫຼືບໍ່

p.parent() ຄົ້ນຄື່ນຄ່າໂນດພໍ່ຂອງໂນດ P (ຄວນກວດສອບຂໍ້ເວັ້ນໃນການເກີດຂໍ້ຜິດພາດກໍ່ລະນີເປັນ

ໂນດຮາກ)

p.children() ຄົ້ນຫາແລະຄື່ນຄ່າລາຍການໂນດລູກຂອງໂນດ P (ຄວນກວດສອບຂໍ້ຍົກເວັ້ນໃນການ

ເກີດຂໍ້ຜິດພາດກໍ່ລະນີ P ເປັນໂນດພາຍນອກ)

root() ຄົ້ນຫາແລະຄື່ນຄ່າໂນດຮາກຂອງທຣີ (ຄວນກວດສອບຂໍ້ຜິດພາດໃນກໍ່ລະນີທຣີວ່າງ)

1. **ນິຍາຍໄບນາລິທຣີ**

ໂຄງສ້າງຕົ້ນໄມ້ແບບມຮທຣີຍ່ອຍບໍ່ເກີນ 2​ ເອີ້ນວ່າ: ໄບນາລີທຣີ (Binary Tree) ມີລັກສະນະສະເພາະດັ່ງນີ້:

* 1. ແຕ່ລະໂນດຈະມີກິ່ງຫຼືເສັ້ນເຊື່ອມໂຍງໄປຍັງໂນດລູກໄດ້ບໍ່ເກີນ 2 ໂນດເທົ່ານັ້ນ ເອີ້ນວ່າໂນດລູກທາງຊ້າຍ (Left Child Node) ແລະໂນດລູທາງຂວາ (Right Child Node) ໄບນາລີທຣີອາດມີລັກສະນະເປັນທຣີວ່າງ ຫຼື ເປັນທຣີທີ່ມີ 1​​ 2​ 3 ຫຼື n ໂນດເຊັ່ນດຽວກັບທຣີທົ່ວໄປ ດັ່ງຮູບທີ່ 1.1 (ກ) ແຕ່ຖ້າໂນດໃດມີເສັ້ນເຊື່ອໃໄປຍັງໂນດລູກພຽງໂນດດຽວ ຈະຕ້ອງລະບຸວ່າເປັນໂນດທາງຊ້າຍຫຼືໂນດທາງຂວາ
  2. ກຸ່ມໂນດທັງໝົດທີ່ເຊື່ອມຈາກໂນດໜຶ່ງໆ ເອີ້ນວ່າ: ຕົ້ນໄມ່ຍ່ອຍ (Subtree) ແບ່ງເປັນຕົ້ນໄມ້ຍ່ອຍທາງຊ້າຍ (Left Subtree) ແລະ ຕົ້ນໄມ້ຍ່ອຍທາງຂວາ (Right Subtree) ດັ່ງຮູບທີ່ 1.1 (ຂ)

|  |  |  |  |
| --- | --- | --- | --- |
| **A** | **B** | **A** | **C**    **A** |
| **B**    **D**    **A** | **B**    **A** | **D** | **A**    **C**    **B** |

**B**

**C**

**E**

**D**

**I**

**F**

**A**

**H**

**G**

**Level 0**

**Level 1**

**Level 2**

**Level 3**

**Right Subtree Left Subtree**

(ກ) ໄບນາລີທຣີມີ 1​ 2 ແລະ 3 ໂນດ (ຂ) Right Subtree ແລະ Left Subtree

ຮູບທີ່ 1.1 ຕົວຢ່າງໂຄງສ້າງຂໍ້ມູນໄບນາລີທຣີ

ຈາກຮູບທີ່ 1.1 (ຂ) ໃນລະດັບທີ່ i ຂອງໄບນ່ລີທຣີ ຈະມີຈຳນວນໂນດທັງໝົດບໍ່ເກີນ ໂນດ ( )ເຊັ່ນໃນລະດັບທີ່ 3 ຈະມີຈຳນວນໂນດບໍ່ເກີນ -8 ໂນດດັ່ງນັ້ນຮາກໄບນາລີທຣີມີຄວາມສູງເທົ່າກັບ h ຈະພົບວ່າ

* *ມີຈຳນວນໂນດທີ່ເປັນໂນດພາຍນອກ* (External Nodes) ທັງໝົດບໍ່ເກີນ ໂນດ
* ໄບນາລີທຣີແບບສົມບູນທີ່ມີຈຳນວນໂນດໃນລະດັບ 0​ ເຖິງລະດັບທີ່ h ຄົບທັງໝົດຈະມີຈຳນວນໂນດທັງໝົດໃນທຣີ (n) ດັ່ງສົມຜົນ

n = 1 +2 + 4 + … + + = – 1

* ຄ່າຄວາມສູ່ງຂອງທຣີຈະຄຳນວນໄດ້ຈາກສົມຜົມ h = (n) ແລະຈຳນວນໂນດທີ່ເປັນໂນດພາຍນອກຈະໂນດທັງ ໝົດເທົ່າກັບ (n + 1) / 2

ການດຳເນີນງານກັບຊະນິດຂໍ້ມູນນາມມະທຳໄບນ່ລີທຣີ (ADT Binary Tree Operations) ເຊັ່ນ

**ການດຳເນີນງານໜ້າທີ່**

p.left() ຄື່ນຄ່າໂນດລູກທາງຊ້າຍຂອງໂນດ p (ຄວນກວດສອບຂໍ້ຍົກເວັ້ນໃນການເກີດຂໍ້ຜິດພາດກໍ່ລະນີ p

ເປັນໂນດພາຍນອກ)

p.right() ຄົ້ນຄື່ຄ່າໂນດລູກທາງຂວາຂອງໂນດ p

p.parent() ຄົ້ນຄື່ນຄ່າໂນດພໍ່ຂອງໂນດ p (ຄວນກວດສອບຂໍ້ຍົກເວັ້ນໃນການເກີດຂໍ້ຜິດພາດກໍ່ລະນີ p ເປັນ

ໂນດຮາກ)

p.hasLeft() ກວດສອບວ່າໂນດ p ມີໂນດໍູກທາງຂວາຫຼືບໍ່ ຫາກມີໃຫ້ຄື່ນຄ່າຈິ່ງ ຫາກບໍ່ມີໃຫ້ຄື່ນຄ່າບໍ່ຈິງ

p.hasRight() ກວດສອບວ່າໂນດ p ມີໂນດລູກທາງຂວາຫຼືບໍ່

p.isRoot() ກວດສອບສ່າໂນດ p ເປັນໂນດຮາກຫຼືບໍ່

p.isExternal() ກວດສອບວ່າໂນດ p ມີສະຖານະເປັນໂນດພໍ່ຫຼືບໍ່

size() ນັບແລະຄື່ນຄ່າຈຳນວນໂນດໃນທຣີ

sEmpty() ກວດສອບວ່າທຣີວ່າງຫຼືບໍ່

oot() ຄົ້ນຫາ ແລະ ຄື່ນຄ່າໂນດຮາກຂອງທຣີ (ຄວນກວດສອບຂໍ້ຍົກເວັ້ນໃນກໍ່ລະນີທຣີວ່າງ)

ການນຳໄບນາລີທຣີໄປໃຊ້ງານຈະຂື້ນຢູ່ກັບລັກສະນະງານີ່ນຳໄປໃຊ້ ເຊິ່ງມີຜົນໃຫ້ຂັ້ນຕອນໃນການເຮັດວຽກຂອງແຕ່ລະການປະຕິບັດໜ້າທີ່ກັບຂໍ້ມູນທີ່ຈັດເກັບໄບນາລີທຣີມີຄວາມແຕກຕ່າງກັນອອກໄປ ໃນຫົວຂໍ້ນີ້ຈະກວ່າເຖິ່ງໂຄງສ້າງຂໍ້ມູນໄບນາລີທຣີເພື່ອການຄົ້ນຫາຂໍ້ມູໃນ ແລະ ໂຄງສ້າງຂໍ້ມູນເອວີເເອວທຣີ (AVL Tree)

**1.2 ໄບນາລີທຣີເພື່ອການຄົ້ນຫາ**

ໄບນາລີທຣີເພື່ອການຄົ້ນຫາ ( Binary Search Tree) ເປັນໂຄງສ້າງຂໍ້ມູນໄບນາລີທຣີ ທີ່ເກັບຂໍ້ມູນແບບມີຄ່າຄີຂອງຂໍ້ມູນເພື່ອໃຊ້ໃນການຄົນຫາຂໍ້ມູນໄດ້ຢ່າງມີປະສິດທິພາບດັ່ງນີ້

1. ທຸກຄ່າຄີໃນທຣີຍ່ອຍທາງຂວາ < ຄ່າຄີຂອງໂນດຮາກ < ທຸກຄ່າຄີໃນທຣີຍ່ອຍທາງຂວາ ດັ່ງນັ້ນຄ່າຄີທັງໝົດຂອງໂນດລູກຫຼານໃນທຣີຍ່ອຍມາງຊ້າຍຂອງໂນດໃດໆ ຕ້ອງມີຄ່ານ້ອຍກວ່າຄ່າຄີຂອງໂນດນັ້ນ ແລະ ຄ່າຄີທັງໝົດຂອງໂນດລູກຫຼານໃນທຣີຍ່ອຍທາງຂວາໂນດໃດໆ ຕ້ອງມີຄ່າຫຼາຍກວ່າຄ່າຄີຂອງໂນດນັ້ນເຊັ່ນກັນ ດັ່ງຮູບທີ່ 1.3 (ກ) ຫາກໂນດມີຄ່າຄີເທົ່າກັບ y ສະແດງວ່າໂນດລູກທາງຊ້າຍຈະເກັບຄ່າຄີ x ໃດໆ ທີ່ມີຄ່ານ້ອຍກວ່າ y ແລະ ໂນດລູກທາງຂວາຈະເກັບຄ່າຄີ z ໃດໆ ທີ່ມີຄ່າຫຼາຍກວ່າຄ່າ y ຈາກຮູບທີ່ 1.3 (ຂ) ພົບວ່າໂນດຂອງຄ່າຄີ 30 ຈະມີຄ່າຄີຂອງໂນດລູກຫຼານໃນທຣີຍ່ອຍທາງຊ້າຍນ້ອຍກວ່າ 30 ທັງໝົດຄື 15 20 25
2. ຈຳນວນໂນດສູງສຸດຂອງທຣີທີ່ມີຄວາມສູງ h ຈະມີຄ່າເທົ່າກັບ -1 ເຊັ່ນ h ເທົ່າກັບ 2 ຈຳນວນໂນດສູງສຸດ ຈະເທົ່າກັບ -1 = 7 ໂນດ

**5**

**10**

**30**

**25**

**35**

**15**

**20**

**40**

**4**

**7**

**x<y**

**y<z**

**y**

(ກ) ຄຸນສົມບັດຂອງໄບນລີທຣີເພື່ອການຄົ້ນຫາ (ຂ) ຕົວຢ່າງໄບນາລີທຣີເພື່ອການຄົ້ນຫາ

ຮູທີ່ 1.3​ ຄຸນສົມບັດ ແລະ ຕົວຢ່າງໂຄງສ້າງຂໍ້ມູນໄບາລິທຣີເພື່ອການຄົ້ນຫາ

1. **ການເພີມໂນກໃຫມ່ໃນໄບນາລີທຣີເພື່ອການຄົ້ນຫາ**

ການປະຕິບັດການເພີມໂນດ ຂໍ້ມູນໃຫມ່ (Insert Operation) ໃນໂຄງສ້າງໄບນາລີທຣີເພື່ອການຄົ້ນຫາກໍ່ລະນີທຣີວ່າງ ຈະກຳນົດໂນດໃຫມ່ເປັນໂນດຮາກ ແຕ່ຫາກບໍ່ແມ່ນທຣີວ່າງຈະແທຣກໂນດໃຫມ່ດັີ່ງລຸ່ມນີ້

* 1. ກຳນົດໃຫ້ໂນດປັດຈຸບັນ (ເອີ້ນວ່າໂນດ x ) ຊີ້ທີ່ໂນດຫາກ
  2. ປຽບທຽບຄ່າຄີຂອງໂນດໃໝ໋ແຕ່ລພໂນດກັບໂນດ x
  3. ຫາກຄ່າຄີຂອງໂນດໃໝ່ ມີຄ່າຫຼາຍກວ່າຄີຂອງໂນດ x ແຍກເປັນ 2 ກໍ່ລະນີດັ່ງນີ້
  + ກໍລະນີ x ບໍ່ມີໂນດລູກທາງຂວາ (x ເປັນໂນດໃບ) ສະແດງວ່າພົບຕຳແໜ່ງໃນການແທຣກໂນດໃໝ່ ຈະແທຣກໂນດໃໝ່ໄວ້ທາງຂວາຂອງໂນດ x
  + ກໍ່ລະນີ x ມີໂນດລູກທາງຂວາ ຈະກຳນົດໃຫ້ x ຊີ້ທີ່ໂນດລູກທາງຂວາແລເວກັບໄປເຮັດຂໍ້ 2 ຊ້ຳເພື່ອຫາຕຳແໜ່ງທີຕ້ອງການແທຣກໂນດໃໝ່ຕໍ່ໄປ
  1. ຫາກມີຄີຂອງໂນດໃໝ່ມີຄ່ານ້ອຍກວ່າຄ່າຄີຂອງໂນດ x ແຍກເປັນ 2 ກໍ່ລະນີດັ່ງລຸ່ທນີ້
* ກໍ່ລະນີ x ບໍ່ມີໂນດລູກທາງຊ້າຍ (x ເປັນໂນດໃບ) ສະແດງວ່າພົບຕຳແໜ່ງໃນການແທຣກໂນດໃໝ່ຈະແທຣກໂນດໃໝ່ໄວ້ທາງຊ້າຍຂອງໂນດ x
* ກ່ລະນີ x ມີໂນດລູກທາງຊ້າຍ ຈະກຳນົດໃຫ້ໂນດ x ຊີ້ທີ່ໂນດລູກທາງຊ້າຍແລ້ວວົນກັບໄປເຮັດຂໍ້ 2 ຊ້ຳເພື່ອຫາຕຳແໜ່ງທີ່ຕ້ອງການແທຣກໂນດໃໝ່ຕໍ່ໄປ
  1. ວົນເຮັດຊ້ຳຂໍ້ 1 ຈົນກວ່າຈະແທຣກຂໍ້ມູນໃໝ່ຈົນຄົບຂໍ້ມູນນຳເຂົ້າທັງໝົດ ດັ່ງຮູບທີ່ 1.4 ສະແດງຕົວຢ່າງການແທຣກໂນດໃໝ່ຂອງຂໍ້ມູນ 10 30 40 35 20 47 ແລະ 5 ຕາມລຳດັບ

**10**

**30**

**35**

**20**

**40**

**47**

**10**

**10**

**30**

**10**

**30**

**40**

**10**

**30**

**35**

**40**

**10**

**30**

**35**

**20**

**40**

**10**

**30**

**35**

**20**

**40**

**47**

**5**

ຮູທີ່ 1.4 ການເພີ່ມຂໍ້ມູນໃນໄບນາລີເພື່ອການຄົ້ນຫາດ້ວຍຂໍ້ມູນ 10​ 30 40 35 20 42 ແລະ 5 ຕາມລຳດັບ

1. **ການລົບໂນດໄບນາລີເພື່ອການຄົ້ນຫາ**

ການປະຕິບັດການລົບໂນດ ( Delete Operation) ໃນທຣີ ຈະຕັດການເຊື່ອມໂຍງໂນດອອກຈາກໂຍງສ້າງໄບນາລີເພື່ອການຄົ້ນຫາ ກຳນົດໃຫ້ x ຊີ້ທີ່ໂນດທີ່ຕ້ອງການລົບອອກຈາກທຣີ ວິທີ່ການລົບຂື້ນຢູ່ກັບສະຖານະຂອງໂນດ x ທີ່ຕ້ອງການຈະລົບແຍກອອກເປັນ 3 ກໍ່ລະນີ ດັ່ງລຸ່ມນີ້

1. ກໍລະນີ x ຄືໂນດໃບ ໃຫ້ກຳນົດໂນດພໍ່ຂອງ x ຊີ່ໄປຍັງຄ່າ null ສ່ວນຈະເປັນການກຳນົດໃຫ້ຕົວຊີ້ທາງຊ້າຍຫຼືຊີ້ທາງຂວາເປັນຄ່າ null ຈະຂື້ນຢູ່ກັບສະຖານະຂອງໂນດ x ວ່າເປັນໂນດລູກທາງຊ້າຍຫຼືໂນດລູກທາງຂວາ

ດັ່ງຮູບທີ່1.5 ສະແດງຕົວຢ່າງລົບໂນດ 47 ເຊິ່ງເປັນໂນດໃບ

**15**

**10**

**30**

**35**

**20**

**40**

**47**

**5**

**25**

**x**

**15**

**10**

**30**

**3**

**5**

**20**

**40**

**5**

**25**

|  |
| --- |
| ເນື່ອງຈາກ 47 ເປັນໂນດລູກທາງຂວາຂອງໂນດ y ຈະກຳນົດໃຫ້ y->right = NULL |

**y (ໂນດພໍ່ຂອງ x)**

**ຮູບທີ່ 1.5 ກໍ່ລະນີໂນດທີ່ຕ້ອງການລົບເປັນໂນດໃບ (ໂນດ 47 )**

1. ກໍລະນີ x ມີໂນດລູກພຽງໂນດດຽວ ຈະໃຫ້ກຳນົດໃຫ້ຕົວຊີ້ຂອງໂນດພໍ່ຂອງ x ຊີ້ໄປຍັງໂນດລູກຂອງ x ດັ່ງຮູບທີ່1.6 ສະແດງຕົວຢ່າງການລົບໂນດ 40 ທີ່ມີລູກທາງຊ້າຍພຽງໂນດດຽວ

|  |
| --- |
| ເນື່ອງຈາກ 40 ເປັນໂນດລູກທາງຂວາຂອງໂນດ y (ຄ່າ 30) ແລະໂນດ 40 ມີລູກທາງຊ້າຍຢູ່ຫນຶ່ງໂນດ  (ຄ່າ 35) ເມື່ອລົບໂນດ 40 ຈະກຳນົດໃຫ້ y->right  = x->left ເພື່ອໃຫ້ 35 ເປັນລູກທາງຂວາຂອງ 30 ແທນ |

**15**

**10**

**30**

**35**

**20**

**40**

**5**

**25**

**x**

**15**

**10**

**30**

**20**

**3**

**5**

**5**

**25**

ຮູບທີ່ 1.6 ກໍ່ລະນີໂນດທີຕ້ອງການລົບ (ໂນດ 40 ) ມີໂນດລູກພຽງໂນດດຽວ (ໂນດ 42 )

1. ກໍລະນີໂນດ x ມີທັງໂນດລູກທາງຊ້າຍ ແລະ ທາງຂວາ

3.1) ຈາກໂນດທັງໝົດໃນທຣີຍ່ອຍມາງຂວາຂອງໂນດ x (x’ s Right Subtree) ໃຫ້ຕຳແໜ່ງໂນດ y ທີ່ມີຄ່າຄີນ້ອຍທີ່ສຸດ (Leftmost Node) ເອີ້ນວ່າໂນດຕາມຫຼັງ (Successor) ແບບອິນວໍເດີຂອງໂນດ x ໃນກໍ່ລະນີການເຂົ້າເຖິງຂໍ້ມູນໃນໄບນາລີທຣີເພື່ອການຄົ້ນຫາເປັນແບບລຽງລຳດັບຈາກນ້ອຍໄປຫາຫຼາຍ ເຊິ່ງເປັນລັກສະນະການເຂົ້າເຖິ່ງແບບອິນວໍເດີນັ້ນເອງ

3.2)​ ສຳເນົາ (Copy) ຂໍ້ມູນຈາກໂນດ y ໄປໄວ້ໃນໂນດ x ແລ້ວລົບໂນດ y ແບ່ງເປັນ 2 ກໍ່ລະນີດັ່ງນີ້

* ກໍ່ລະນີໂນດ y ຄືໂນດໃບ ໃຫ້ກຳນົດໂນດພໍຂອງ y ຊີ້ໄປຍັງຄ່າ null ໃນລັກສະນະດຽວກັນກັບຂໍ້ 1 ດັ່ງຕົວຢ່າງໃນຮູບທີ່ 1.7
* ກໍລະນີໂນດ y ມີໂນດລູກ 1​ ໂນດ ໃຫ້ກຳນົດໂນດພໍ່ຂອງ y ຊີ້ໄປຍັງໂນດລູກຂອງ y ໃນລັກສະນະດຽວກັບຂໍ້ 2 ດັ່ງຕົວຢ່າງໃນຮູບທີ່ 1.8

|  |
| --- |
| ເມື່ອຈະລົບໂນດ 30 ເຊິ່ງມີທັງໂນດລູກທາງຊ້າຍ  ແລໂນດລູກທາງຂວາມີການເຮັດວຽກດັັ່ງນີ້:   1. ຫາໂນດy ທີ່ເປັນ Inorder Successor ຂອງ 30 ເຊິ່ງກໍ່ຄືຄ່ານ້ອຍສຸດຂອງທຣີບ່ອຍມາງຂວາຂອງໂນດ 30 ໃນກໍ່ລະນີນີ້ຈະພົບ y ຄືຄ່າ 35 ດັ່ງຮູບທີ່ 10.7 (ກ) 2. ສຳເນົາຂໍ້ມູນຈາກໂນດ35 ໄປໄວ້ທີ່ໂນດ 30 3. ລົບໂນດ 35 ອອກຈາກທຣີແຍກເປັນ 2 ກໍ່ລະນີດັ່ງນີ້:    * ກໍ່ລະນີ 35 ບໍ່ມີໂນດລູກຈະກຳນົດໃຫ້ z->left = NULL 35 ດັ່ງຮູບທີ່ 10.7 (ຂ)    * ກໍ່ລະນີ 35 ມີໂນດລູກທາງຂວາຈະກຳນົດໃຫ້ z->left = y->right ດັ່ງຮູບທີ່ 10.8   ( ເນື່ອງຈາກໂນດ y ຄືລູກທາງ້າຍຂອງໂນດ z (ຄ່າ 40)) |

**x**

**x**

**I**

**(**

**norder**

**S**

**uccessor**

**ຂອງ**

**x**

**)**

**15**

**10**

**30**

**35**

**20**

**40**

**47**

**5**

**25**

**15**

**10**

**30**

**35**

**20**

**40**

**47**

**5**

**25**

**x (**

**ສຳເນົາຂໍ້ມູນອອກຈາກ**

**y)**

**15**

**10**

**3**

**5**

**35**

**20**

**40**

**47**

**5**

**25**

**y**

**(**

**I**

**norder**

**S**

**uccessor**

**ຂອງ**

**x**

**)**

**z**

**15**

**10**

**35**

**20**

**40**

**47**

**5**

**25**

(ຂ) ສຳເນົາຈາກ y ໄປ x ແລ້ວລົບໂນດ y

ຮູບທີ່ 1.7 ກໍ່ລະນີໂນດທີ່ຕ້ອງການລົບ (ໂນດ x) ມີໂນດລູກ 2 ໂນດ ແລະໂນດ y ບໍມີລູກ

**x (**

ສຳເນົາຂໍ້ມູນມາຈາກ

**y)**

**15**

**10**

**35**

**35**

**20**

**40**

**47**

**5**

**25**

**39**

**z**

**(**

**ໂນດພໍ່ຂອງ**

**y**

**)**

**15**

**10**

**3**

**5**

**20**

**40**

**47**

**5**

**25**

**39**

Y (Inorder Successor ຂອງ X )

ຮູບທີ່ 1.8 ກໍ່ລະນີໂນດທີ່ຕ້ອງການລົບ (ໂນດ x ) ມີໂນດລູກ 2 ໂນດ ແລະ ໂນດ y ມີໂນດລູກທາງຂວາ

1. **ການຄົ້ນຫາຄີໃນໄບນາລີເພື່ອການຄົ້ນຫາ**

ການປະຕິບັດກນເພື່ອກດັ່ານຄົ້ນຫາ ( Find Operation) ເປັນການຄົ້ນຫາຕຳແໜ່ງໂນດທີ່ມີຄ່າຄີຕົງກັບຄ່າຄີທີ່ຕ້ອງການຄົ້ນຫາຕົວຢ່າງເຊັ່ນ ຫາກຕ້ອງການລົບໂນດທີ່ມີຄ່າຄີເທົ່າກັບ k ຈະຕ້ອງເຮັດການຄົ້ນຫາຕຳແໜງໂນດທີ່ຕ້ອງການລົບດ້ວຍການເອີ້ນໃຊ້ການປະຕິບັດການຄົ້ນຫາ ໂດຍຈະເລີ່ມຕົ້ນທີ່ໂນດຮາກ ແລ້ວປຽບທຽບຄ່າຄີທີ່ຕ້ອງການຄົ້ນຫາກັບໂນດປັດຈຸບັນ ຫາກພົບຄີທີ່ຕ້ອງການຈະຢຸດການຄົ້ນຫາ ແຕ່ຫາກຍັງບໍ່ພົບຈະຄົ້ນຫາໄປຍັງໂນດຖັດໄປ ຈົນກວ່າຈະພົບໂນດທີ່ມີຄ່າຄີຕົງກັບຄີທີ່ຕ້ອງການຄົ້ນຫາ ຫຼືຈົນກວ່າຈະເຖິງໂນດໃບ ເຊິ່ງຈະກວ່າໂດຍລະອຽດໃນຫົວຂໍ້ຖັດໄປ

1. **ປະສິດທິພາບໃນການເຮັກວຽກກັບໂຄງສ້າງໄບນາລີທຣີເພຶ່ອການຄົ້ນຫາ**

ກຳນົດໃຫ້ n ຄືຈຳນວນຂໍ້ມູນທີ່ຈັດເກັບໃນໄບນາລິທຣີ ເມື່ອວິເຄາະຫາປະສິດທິພາບໃນການເຮັດວຽກການຕ່າງໆ ກັບ

ຂໍ້ມູນໃນໄບນາລີທຣີ ເພື່ອການຄົ້ນຫາ ແລະ ຈະເຫັນວ່າ

* ການຈັດເກັບຂໍ້ມູນໃນໄບນາລີທຣີຈະໃຊ້ເນື້ອທີ່ (Space) ເທົ່າກັບ O (n)
* ເວລາທີ່ໃຊ້ໃນການເຮັດວຽກ T(n) ຂອງແຕ່ລະການປະຕິບັດການຂອງໄຍນາລີທຣີບໍ່ວ່າຈະເປັນ find(),insert() ຫຼື remove() ຈະມີຄ່າເທົ່າກັບຄວາມສູງ (h) ຂອງທຣີຄື O(h) ແບ່ງເປັນ 2 ລັກສະນະດັ່ງນີ້
* ຫາກໄບນາລີທຣີມີລັກສະນະເປັນທຣີແບບສົມບູນ ( Balance Tree) ດັ່ງຮູບທີ່ 1.9 (ກ) ນັ້ນຄືໂນດຮາກມີຈຳນວນໂນດໃນທຣີຍ່ອຍທາງຊ້າຍເທົ່າກັບຈຳນວນໂນດໃນທຣີຍ່ອຍທາງຂວາ ຈະເຫັນວ່າວາມສູງຂອງທຣີຈະເທົ່າກັບ log(n) ດັ່ງນັ້ນ T(n) = O(log (n))
* ອາດຈະພົບຫັນກໍ່ລະນີທີ່ບໍ່ດີທີ່ສຸດ (Worst case) ເມື່ອການເກັບຂໍ້ມູນໃນໄບນາລີທຣີເປັນລັກສະນະອຽງຊ້ານຫຼືຂວາດ້ານດຽວ ດັ່ງຮູບທີ່ 1.9 (ຂ) ພົບວ່າຄວາມສູງຂອງທຣີ ຄື n ດັ່ງນັ້ນ T(n) = O(n)

ຈາການວິເຄາະປະສິດທິພາບດ້ນເວລາໃນການເຮັດວຽກກັບຂໍ້ມູນໃນໄບນາລີທຣີ ຈະເຫັນວ່າຄວາມສົມບູນ

ຂອງທຣີເປັນເລື່ອງສຳຄັນ ຈຶ່ງເຮັດໃຫ້ມີຜູ້ຄິດຄົ້ນໂຄງສ້າງຂໍ້ມູນທຣີທີ່ສາມາດຈັດເກັບຂໍ້ມູນໃນທຣີແບບສົມບູນ ເອີ້ນວ່າ ເອວີແອວທຣີ (AVL Tree) ເຊິ່ງຕັ້ງຊື່ຕາມຜູ້ຄິດຄົ້ນໂຄງສ້າງເອວີແອວທຣີ ຄື Georgy Adelson-Velsky ແລະ Evgenii Landis

(ກ) ທຣີແບບສົມບູນ (ຂ) ທຣີແບບອຽງຊ້າຍ ແລະ ອຽວຂວາດ້ານດຽວ

ຮູບທີ່ 1.9 ໂຄງສ້າງຂໍ້ມູນທຣີແບບສົມບູນ ແລະ ທຣີແບບອຽງຂ້າງດຽວ

**1.3 ເອວີແອວທຣີ (AVL Tree)**

ເປັນໄບນາລີທຣີເພື່ອການຄົ້ນຫາທີ່ມີວິືີ່ການປະຕິບັດການກັບທຣີໃຫ້ສາມາດຈັດເກັບຂໍ້ມູນໃນລັກສະນະສົມບູນຫຼື ເກືອບຈະສົມບູນ ດັ່ງຮູບທີ່ 2.0 ແລະ ໂຄງສ້າງເອວີແອວທຣີ ຈະມີຄຸນສົມບັດທີ່ສຳຄັນຄື “ ທຸກໂນດທີ່ບໍ່ແມ່ນໂນດໃບ ທຣີຍ່ອຍທາງຊ້າຍ ແລະ ທີ່ຍ່ອຍທາງຂວາຈະມີຄວາມສູງຕ່າງກັນຫຼາຍທີ່ສຸດພຽງ 1 “

ຮູບທີ່ 2.0 ສະແດງຕົວຢ່າງໂຄງສ້າງເອວີແອວທຣີທີ່ມີຄວາມສູງເທົ່າກັບ 4 ຈະເຫັນວ່າໂນດ 7​ 8 ມີຄວາມສູງຂອງທຣີຍ່ອຍທາງຊ້າຍ (ມີໂນດ 50 ເປັນໂນດຮາກຂອງທຣີຍ່ອຍ) ເທົ່າກັບ 2 ໃນຂະນະທີ່ຄວາມສູງຂອງທຣີຍ່ອຍທາງຂວາ (ມີໂນດ 80 ເປັນໂນດຮາກຂອທຣີຍ່ອຍ) ເທົ່າກັບ 1​

**17**

**78**

**32**

**88**

**50**

**48**

**44**

**60**

**4**

**2**

**3**

**2**

**1**

**1**

**1**

**0**

**10**

**20**

**38**

**0**

**0**

**82**

**90**

**0**

**0**

**56**

**67**

**0**

**0**

**46**

**49**

**0**

**0**

**1**

ຮູບທີ່ 2.0 ຕົວຢ່າງໂຄງສ້າງເອວີແອວທຣີ

**1.3.1 ການແທຣກໂນດໃໝ່ໃນເອວີແອວທຣີ**

ຈະໃຊ້ຫຼັການດຽວກັນກັບການແທຣກໂນດໃໝ່ໃນໄບນາລີທຣີເພື່ອການຄົ້ນຫາ ນັ້ນຄືການປຽບທຽບຄ່າຄີຈາກໂນດຮາກ ຫາກຄ່າຄີຂອງໂນດໃໝ່ນ້ອຍກວ່າ ຈະໄປຍັງໂນດລູກທາງຊ້າຍແຕ່ຫາກຫຼາຍກວ່າກໍ່ຈະໄປຍັງໂນດລູກທາງຂວາ ແລະ ປຽບທຽບກັບຄ່າຄີເລື້ອຍໆໄປ ຈົນກວ່າຈະເຫັນວ່າຕຳແໜ່ງໃນການແທຣກໂນດໃໝ່ ດັ່ງຮູບຕົວຢ່າງທີ່ 2.1 ເມື່ອແທຣກໂນດໃໝ່ (ໂນດ 11) ໃນໂຄງສ້າງເອວີແອວທຣີ ຮູບທີ່ 2.1 (ກ) ຈໄດ້ຜົນລັບຂອງການດຳເນີນການແທຣກໂນດໃໝ່ ດັ່ງຮູບທີ່ 2.1 (ຂ)

**3**

**2**

**1**

**4**

**5**

**6**

**15**

**16**

**1**

**4**

**13**

**12**

**3**

**2**

**1**

**4**

**5**

**6**

**15**

**6**

**14**

**13**

**12**

**1**

**1**

(ກ) ເອວີແອວທຣີ (ຂ) ດອວີແອວທຣີຫຼັງການແທຣກໂນດ 11

**3**

**2**

**1**

**4**

**5**

**6**

**15**

**16**

**7**

**14**

**13**

**12**

(ຄ) ເອວີແອວທຣີຂັ້ນຕອນການແທຣກໂນດໃໝ່ໃນເອວີແອວທຣີ

ຮູບທີ່ 2.1 ຕົວຢ່າງຂັ້ນຕອນການແທຣກໂນດໃໝ່ໃນເອວີແອວທຣີ

ແນວໃດກໍ່ຕາມ ເນື່ອງຈາກເອວີແອວທຣີເປັນໂຄງສ້າງທີ່ມິລັກສະນະໃກ້ຄຽງກັບທຣີແບບສົມບູນຫຼາຍທີ່ສຸດ ທຸກຄັ້ງທີ່ມີການການປະຕິບັດການເພື່ອແທຣກໂນດໃໝ່ໃນທຣີ ຈະມີການກວດສອບວ່າທຣີຂາດຄວາມສົມບູນຫຼືບໍ່ ເຊິ່ງເຫັນວ່າໂນດ 15 ເປັນຕຳແໜ່ງທີ່ເຮັດໃຫ້ທຣີຂາດຄວາມເປັນດອວີແອວທຣີ ເນື່ອງຈາທຣີ່ຍ່ອຍທາງຊ້າຍ (ໂນດ 13) ມີຄວາມສູງເທົ່າກັບ 2 ໃນຂະນະທີ່ທຣີຍ່ອຍທາງຂວາ (ໂນດ 16) ມີຄວາມສູງເທົ່າກັບ 0 ດັ່ງຮູບທີ່ 2.1 (ຂ) ເຊິ່ງຕ້ອງແກ້ໄຂດ້ວຍການໝູນໂນດ 13 ມາແທ່ນທີ່ໂນດ 15 ດັ່ງຜົນລັບໃນຮູບທີ່ 2.1 (ຄ)

**1.3.2 ການໝູນທຣີ**

ເມື່ອໄບນາລີທຣີຂາດຄວາມເປັນເອວີແອວທຣີ ຈະແກ້ໄຂດ້ວຍການໝູນໂນດໃນທຣີເພື່ອປັັບທຣີໃຫ້ມີລັກສະນະເປັນດອວີແອວທຣີເກົ່າ ເຊິ່ງມີ 2 ຮູບແບບ ຄື ການໝູນຄັ້ງດຽວ ( Single Rotation) ແລະ ການໝູນສອງຄັ້ງ (Double Rotation) ມີລາຍລະອຽດດັ່ງລຸ່ມນີ້

1. **ການໝູນຄັ້ງດຽວ**

ການໝູນຄັ້ງດຽວທີ່ສາມາດປັບໂຄງສ້າງທຣີ ໃຫ້ມີຄວາມສົມບູນຕົງກັບຄຸນສົມບັດຂອງເອວີແອວທຣີ ດັ່ງຮູບທີ່ 2.2​ ເຫັນວ່າໂນດ a ຄື ໂນດທີ່ເຮັດໃຫ້ໄບນາລີທຣີຂາດຄຸນສົມບັດຄວາມເປັນເອວີແອວທຣີ

**X**

**b**

**a**

**Y**

**Y**

**(**

ຂ

**)**

**ການໝູນຈາກໂນດທາງຂວາໄປທາງຊ້າຍ**

**Y**

**a**

**b**

**Z**

**X**

**ດຶງໂນດ**

**b**

**ຂື້ນ**

**ໄປແທ່ນທີ່ໂນດ**

**a**

**X**

**Y**

**Z**

**X**

**a**

**b**

**b**

**a**

**Z**

**Z**

**ບໍ່ເປັນ**

**AVL**

**ທີ່ໂນດ**

**a**

**(**

**ກ)**

**ການໝູນຈາກໂນດທາງຊ້າຍໄປທາງຂວາ**

ຮູບທີ່ 2.2 ການໝູນຄັ້ງດຽວເພື່ອປັບຄວາມສົມບູນຂອງເອວີແອວທຣີ

ວິທີ່ການໝູນຈະມີຢູ່ ​2 ລັກສະນະດັ່ງລຸ່ມນີ້

1. ການໝູນທາງຊ້າຍົປຂວາ (Rotation With Left Node) ຈະໃຊ້ໃນກໍ່ລະນີທີ່ຍ່ອຍທາງຊ້າຍຂອງໂນດ a ມີຄວາມສູງຫຼາຍກວ່າທຣີຍ່ອຍທາງຂວາ ຈຶ່ງຕ້ອງຫຼຸດລະດັບຄວາມສູງຂອງທຣີຍ່ອຍທາງຊ້າຍຂອງໂນດ a ດ້ວຍການໝູນໂນດລູກທາງຊ້າຍ (ໂນດ b) ໃຫ້ຂື້ນໄປຢູ່ຕຳແໜ່ງທີ່ໂນດ a ເກົ່າ ນັ້ນຄືໂນດພໍ່ຂອງ a ຈະມີໂນດ b ເປັນໂນດລູກແທ່ນ ດັ່ງຮູບທີ່ 2.2 (ກ) ໂດຍກຳນົດໃຫ້

b ຄື ໂນດລູກທາງຊ້າຍຂອງໂນດ a

y ຄື ທຣີຍ່ອຍທາງຂວາຂອງໂນດ b

z ຄື ທຣີຍ່ອຍທາງຂວາຂອງໂນດ a

ວິທີ່ການໝູນແບບນີ້ຈະເຮັດໃຫ້ມີການປ່ຽນການເຊື່ອມໂຍງຂອງໂນດດັ່ງນີ້

* ຍ້າຍທຣີຍ່ອຍທາງຂວາຂອງໂນດ b (ໝາຍເຖີງ y ) ໄປເປັນທຣີຍ່ອຍທາງຊ້າຍຂອງໂນດ a ເນື່ອງຈາກໂນດຂອງທຣີຍ່ອຍ y ຈະມີຄ່າຂໍ້ມູນຫຼາຍກວ່າໂນດ b (ເພາະຢູ່ທາງຂວາຂອງ b) ແຕ່ມີຄ່ານ້ອຍກວ່າໂນດ a
* ແລະຍ້າຍໂນດ a ໄປເປັນໂນດລູກທາງຂວາຂອງໂນດ b

1. ການໝູນໂນດທາງຂວາໄປຊ້າຍ (Rotation With Right Node) ຈະໃຊ້ໃນກໍ່ລະນີທີ່ທຣີຍ່ອຍທາງຂວາຂອງໂນດ a ມີຄວາມສູງຫລາຍກວ່າທຣີຍ່ອຍທາງຊ້າຍ ຈຶ່ງຕ້ອງຫຼຸດລະດັບຄວາມສູງຂອງທຣີຍ່ອຍທາງຂວາຂອງໂນດ a ດ້ວຍການໝູນໂນດລູກທາງຂວາ (ໂນດ b ) ໄປຢູ່ຕຳແໜ່ງແທ່ນສະແດງດັ່ງຮູບທີ່ 2.2 (ຂ) ໂດຍກຳນົດໃຫ້

b ຄື ໂນດລູກທາງຂວາຂອງໂນດ a

y ຄື ທຣີຍ່ອຍທາງຊ້າຍຂອງໂນດ b

z ຄື ທຣີຍ່ອຍທາງຊ້າຍຂອງໂນດ a

ວິທີການໝູນແບບນີ້ຈະເຮັດໃຫ້ມີການປ່ຽນການເຊື່ອມໂຍງຂອງໂນດດັ່ງນີ້:

* ຍ້າຍທຣີຍ່ອຍທາງຊ້າຍຂອງໂນດ b (ໝາຍເຖີງທຣີຍ່ອຍ y ) ໄປເປັນທຣີຍ່ອຍທາງຂວາຂອງໂນດ a ເນື່ອງຈາກໂນດຮາກຂອງທຣີຍ່ອຍ y ຈະມີຄ່າຂໍ້ມູນນ້ອຍກວ່າໂນດ b (ເພາະຢູ່ທາງຊ້າຍຂອງ b) ແຕ່ຈະມີຄ່າຫຼາຍກວ່າໂນດ a
* ແລະຍ້າຍໂນດ a ໄປເປັນໂນດລູກທາງຊ້າຍຂອງໂນດ b ແທ່ນ

**3**

**2**

**1**

**4**

**5**

**6**

**15**

**16**

**7**

**14**

**13**

**12**

**3**

**2**

**1**

**4**

**5**

**6**

**15**

**16**

**7**

**14**

**13**

**12**

(ກ) ທຣີທີ່ຂາດຄວາມເປັນເອວີແອວທຣີທີ່ໂນດ 14 (ຂ) ຫຼັງການໝູນຈາກໂນດທາງຊ້າຍໄປຂວາ

ຮູບທີ່ 2.3 ຕົວຢ່າງການໝູນຈາກໂນດທາງຊ້າຍໄປຂວາ

ຈາກຮູບທີ່ 2.3 ທີ່ສະແດງຕົວຢ່າງໂຄງສ້າງໄບນາລີທຣີທີ່ເຫນວ່າຕຳແໜ່ງໂນດ 14 ມີຄວາມສູງຂອງທຣີຍ່ອຍທາງຊ້າຍເທົ່າກັບ 1​ໃນຂະນະທີ່ທຣີຍ່ອຍທາງຂວາເປັນທຣີວ່າງ (ມີຄວາມສູງເທົ່າກັບ -1) ຈຶ່ງມີຜົົນເຮັດໃຫ້ໂນດ 14 ເປັນໂນດທີ່ເຮັດໃຫ້ໄບນາລີທຣີຂາດຄຸນສົມບັດຂອງຄວາມເປັນເອວີແອວທຣີ ດັ່ງງຮູບທີ່ 2.3 (ກ) ແລະ ສາມາດແກ້ໄຂໄດ້ດ້ວຍວິທີ່ການໝູນຈາກໂນດທາງຊ້າຍໄປທາງຊ້າຍໄປຂວາເນື່ອງຈາກທຣີຍ່ອຍທາງຊ້າຍມີຄວາມສູງຫຼາຍກວ່າດ້ວຍການໝູນໂນດ 13​ ຂື້ນໄປແທ່ນທີ່ໂນດ 14 ຕາມລຳດັບຂັ້ນຕອນດັ່ງນີ້

1. ຍ້າຍທຮີຍ່ອຍທາງຂວາຂອງໂນດ 13 ໄປເປັນທຣີຍ່ອຍທາງຊ້າຍຂອງໂນດ 14 ໃນທີ່ນີ້ ຄື ທຣີວ່າງດັ່ງນັ້ນໂນດ 14 ຈຶ່ງບໍາມີລູກທາງຊ້າຍ
2. ຍ້າຍໂນດ 14 ໄປເປັນໂນດລູກທາງຂວາຂອງໂນດ 13

ຫຼັງຈາກາານໝູນໂນດ 13 ຂື້ນໄປແທ່ນທີ່ໂນດ 14 ເຮັດໃຫ້ແກ້ບັນຫາລະດັບຄວາມສູງຂອງທຣີຍ່ອຍຂອງໂນດ 14 ໄດ້ ແລະ ເຮັດໃຫ້ທຣີມີຄຸນສົມບັດຄວາມເປັນເອວີແອວທຣີ ດັ່ງຮູບທີ່ 2.3 (ຂ)

ຮູບທີ 10.14 ສະແດງຕົວຢ່າງໂຄງສ້າງໄບນາຣີທຣີທີ່ມີຕຳແໜ່ງໂໝດ 4 ເຮັດໃຫ້ໄບນາຣີທຣີ ຂາດຄຸນສົມບັດຂອງຄວາມເປັນເອວີແອລທຣີ (AVLT) ແລະສາມາດແກ້ໄຂໄດ້ດ້ວຍວິທີການໜູນ ຈາກໂໝດທາງຂວາໄປຊາຍ ເນື່ອງຈາກໂໝດ 4 ມີທຣີຍ່ອຍທາງຂວາມີຄວາມສູງຫຼາຍກວ່າທຣີຍ່ອຍທາງຊ້າຍ ເພື່ອຫຼຸດລະດັບຄວາມສູງຂອງທຣີ ຍ່ອຍທາງຂວາຈື່ງຕ້ອງໝູນໂໝດ 7 ຂື້ນໄປແທນໂໝດ 4 ແລ້ວຍ້າຍໂໝດອື່ນໆ ດັ່ງນີ້:

1. ຍ້າຍທຣີຍ່ອຍທາງຊ້າຍຂອງໂໝດ 7 ໄປເປັນທຣີຍ່ອຍທາງຂວາຂອງໂໝດ 4
2. ຍ້າຍໂໝດ 4 ໄປເປັນໂໝດລູກທາງຊ້າຍຂອງໂໝດ 7

**3**

**2**

**1**

**4**

**5**

**6**

**15**

**16**

**7**

**14**

**13**

**3**

**2**

**1**

**4**

**5**

**6**

**15**

**16**

**7**

**14**

**13**

(ກ) ທຣີທີ່ຂາດຄວາມເປັນເອວີແອລທຣີທີ່ໂໝດ 4 (ຂ) ຫຼັງຈາກໝູນຈາກໂໝດທາງຂວາໄປຊ້າຍ

ຕາຕະລາງທີ 10.1 ຕົວຢ່າງການປັບຄວາມສົມບູນຂອງເອວີແອລທຣີຫຼັງແຊກໂໝດໄໝ່ດ້ວຍການໝູນຄັ້ງດຽວ

|  |  |  |  |
| --- | --- | --- | --- |
| **ເອວີແອລທຣີ** | | ທຣີທີ່ຂາດຄຸນສົມບັດເອວີແອລທຣີຫຼັງເພີ່ມໂໝດໄໝ່ | **ເອວີແອລທຣີຫຼັງການໝູນ** |
| **3** | **5** | **5**    **3**    **1**    ຫຼັງເພີ່ມໂໝດ 1 ຂາດຄຸນສົມບັດເອວີແອລທຣີທີ່ໂໝດ 5 | **5**    **3**    **1**    ຫຼັງໝູນໂໝດ 3 ຂື້ນ |
| **1** | **5**    **3**    **9** | **5**    **3**    **1**    **9**    **12**    ຫຼັງເພີ່ມໂໝດ 12 ຂາດຄຸນສົມບັດເອວີແອລທຣີທີ່ໂໝດ 5 | **5**    **3**    **1**    **9**    **12**    ຫຼັງໝູນໂໝດ 9 ຂື້ນ |
| **1** | **5**    **3**    **9**    **12** | **5**    **3**    **1**    **9**    **12**    **14**    ຫຼັງເພີ່ມໂໝດ 14 ຂາດຄຸນສົມບັດເອວີແອລທຣີທີ່ໂໝດ 3 | **5**    **3**    **1**    **9**    **12**    **14**      ຫຼັງໝູນໂໝດ 9 ຂື້ນ |

ຈາກຕາຕະລາງທີ 10.1 ສະແດງຕົວຢ່າງການໝູນຄັ້ງດຽວເພື່ອແກ້ໄຂບັນຫາການຂາດຄຸນສົມບັດເອວີແອລທຣີຫຼັງການແຊກໂໝດໄໝ່ໃນທຣີມີ ດັ່ງນີ້:

1. ເມື່ອແຊກໂໝດ 1 ໃນທຣີແລ້ວເຮັດໃຫ້ທຣີຂາດຄຸນສົມບັດ ຄວາມເປັນເອວີແອລທຣີທີ່ໂໝດ 5 ເຊີ່ງແກ້ໄຂດ້ວຍການໝູນຈາກໂໝດຊ້າຍໄປຂວາ ໂດຍການໝູນໂໝດ 3 ຂື້ນໄປແທນທ່ີ່ໂໝດ 5 ທີ່ສະແດງໃນແຖວທີ 1 ຂອງຕາຈະລາງທີ 10.1
2. ເມື່ອແຊກໂໝດ 12 ໃນທຣີແລ້ວເຮັດໃຫ້ທຣີຂາດຄຸນສົມບັດຄວາມເປັນເອວີແອລທຣີ ເຊີ່ງແກ້ໄຂດ້ວຍການໝູນຈາກໂໝດຂວາໄປຊ້າຍ ໂດຍການໝູນໂໝດ 9 ຂື້ນໄປແທນທີ່ໂໝດ 5 ທີ່ສະແດງແຖວທີ 2 ຂອງຕາຕະລາງ 10.1
3. ເມື່ອແຊກໂໝດ 14 ໃນທຣີແລ້ວເຮັດໃຫ້ທຣີຂາດຄຸນສົມບັດຄວາມເປັນເອວີແອລທຣີທຮ່ໂໝດ 3 ເຊີ່ງແກ້ໄຂດ້ວຍການໝູນຈາກໂໝດຂວາໄປຊ້າຍ ໂດຍການໝູນໂໝດ 9 ຂື້ນໄປແທນທີ່ໂໝດ 3 ທີ່ສະແດງໃນແຖວທີ 3 ຂອງຕາຕະລາງ 10.1

ໃນບາງກໍລະນີ ວິທີການໝູນທຣີພຽງຄັ້ງດຽວອາດບໍ່ສາມາດແກ້ໄຂບັນຫາທຮ່ທຣີຂາດຄຸນສົມບັດເປັນເອວີແອລທຣີໄດ້ ດັ່ງຮູບທີ 10.15 ໂຄງສ້າງທຣີຂາດຄຸນສົມບັດຄວາມເປັນເອວີແອລທຣີ ຕຳແນ່ງໂໝດທີ 14 ເຊີ່ງທຣີຍ່ອຍທາງຂວາມີຄວາມສູງຫຼາຍກວ່າທຣີຍ່ອຍໄປທາງຊ້າຍ ດັ່ງຮູບທີ 10.15 (ກ) ແລະໄດ້ເຮັດການໝູນຈາກໂໝດທາງຂວາໄປທາງຊ້ສຍ ດັ່ງຮູບທີ 10.15 (ຂ) ດ້ວຍການໝູນໂໝດ 20 ໄປແທນທີ່ໂໝດ 14 ແລ້ວຍ້າຍໄປໂໝດອື່ນໆ ດັ່ງນີ້:

1. ຍ້າຍທຣີຍ່ອຍໄປທາງຊ້າຍຂອງໂໝດ 20 ໄປເປັນທຣີຍ່ອຍທາງຂວາໂໝດ 14
2. ຍ້າຍໂໝດ 14 ໄປເປັນໂໝດລູກທາງຊ້າຍຂອງໂໝດ 20

**5**

**3**

**1**

**9**

**12**

**2**

**0**

**24**

**14**

**16**

**18**

**5**

**3**

**1**

**9**

**12**

**20**

**24**

**14**

**16**

**18**

(ກ) ທຮີທີ່ຂາດຄວາມເປັນເອວີແອລທຣີທີ່ໂໝດ 14 (ຂ) ຫຼັງການໝູນຈາກໂໝດທາງຂວາໄປຊ້າຍ

(2) ການໜູນສອງຄັ້ງ

ຈາກຮູບທີ 10.15 (ຂ) ເຫັນໄດ້ວ່າຫຼັງການໝູນທຣີຄັ້ງດຽວ ບໍ່ສາມາດຫຼຸດລະດັບຄວາມສູງຂອງທຣີໄດ້ ເພາະໂຄງສ້າງທຣີທຮ່ໄດ້ຫຼັງການໝູນຂາດຄຸນສົມບັດຂອງຄວາມເປັນເອວີແອລທຣີທີ່ໂໝດ 20 ແທນເຊື່ງສາມາດແກ້ໄຂບັນຫານີ້ໄດ້ດ້ວຍວິທີການໝູນ 2 ຄັ້ງ ເມື່ອພົບເຫັນເຫດການທີ່ການໝູນຄັ້ງດຽວບໍ່ສາມາດແກ້ໄຂບັນຫາການຂາດຄຸນສົມບັດເອວີແອລທຣີໄດ້ ວິທີການໝູນສອງຄັ້ງອາດໝູນຈາກຊ້າຍໄປຂວາ ແລ້ວຈື່ງໝູນຈາກຂວາໄປຊ້າຍອີກຄັ້ງ ຫຼື ສະລັບກັນລັກສະນະການໝູນສອງຄັ້ງຈະມີ 2 ແບບ ດັ່ງນີ້:

1. ການໝູນສອງຄັ້ງດ້ວຍໂໝດລູກທາງຂວາ

ກຳນົດໃຫ້ໂໝດ A ຄືໂໝດທີ່ເຮັດໃຫ້ໄບນາຣີທີ່ຂາດຄຸນສົມບັດຄວາມເປັນເອວີແອລທຣີ ດັ່ງຮູບທີ 10.16 ເຫັນໄດ້ວ່າທຣີຍ່ອຍທາງຂວາຂອງໂໝດ A ມີຄວາມສູງຫຼາຍກ່ວາທຣີຍ່ອຍທາງຊ້າຍ ຫາກໝູນໂມດລູກທາງຂວາຂອງໂໝດ A ຄືໂໝດ B ຂື້ນໄປແທນທີ່ໂໝດ A ດັ່ງຮູບທີ 10.16 (ກ) ເຫັນວ່າຈະເຮັດໃຫ້ໂໝດ B ມີບັນຫາເລື່ອງລະດັບຄວາມສູງຂອງທຣີຍ່ອຍທາງຊ້າຍຕ່າງຈາກທຣີຍ່ອຍທາງຂວາຫຼາຍກວ່າ 1 ມີຜົນໃຫ້ທຣິຂາດຄຸນສົມບັດຄວາມເປັນເອວີແອລທຣີທີ່ໂໝດ B ແທນ ສະແດງວ່າການໝູນຄັ້ງດຽວ ບໍ່ສາມາດແກ້ໄຂບັນຫາການຂາດຄຸນສົມບັດຄວາມເປັນເອວີແອລທຣີໄດ້ ລັກສະນະນີ້ສາມາດແກ້ໄຂບັນຫາຂອງເຫດການນີ້ໄດ້ໂດຍການແຍກ Y ອອກເປັນ 2 ທຣີຍ່ອຍ Y1 ແລະ Y2 ໂດຍສົມມຸດໃຫ້ໂໝດ C ເປັນໂໝດຮາກຂອງທຣີຍ່ອຍ Y ດັ່ງນັ້ນ Y1 ຈະເປັນທຣີຍ່ອຍທາງຊ້າຍຂອງໂໝດ C ແລະ Y2 ເປັນທຣີຍ່ອຍທາງຂວາຂອງໂໝດ C ດັ່ງຮູບ 10.16 (ຂ) ຈາກນັ້ນເຮັດການໝູນໂໝດ C ຈຳນວນ 2 ເທື່ອເພື່ອຫຼຸດລະດັບຄວາມສູງຂອງທຣີຍ່ອຍ Y ດັງນີ້

* ການໝູນເທື່ອ 1 ດ້ວຍການໝູນໂໝດ C ໄປແທນທີ່ໂໝດ B ໂດຍໃຊ້ຫຼັກການໝູນຄັ້ງດຽວຈາກໂໝດຊ້າຍໄປຂວາ ເຊື່ງຈະມີການຍ້າຍການເຊື່ອມໂຍງຂອງທຣີດັ່ງນີ້:

1.ຍ້າຍທຣີຍ່ອຍໄປທາງຂວາຂອງໂໝດ C ໄປເປັນທຣີຍ່ອຍທາງຊ້າຍຂອງໂໝດ B

2.ຍ້າຍໂໝດ B ໄປເປັນໂໝດລູກທາງຂວາຂອງໂໝດ C

* ຈາກນັ້ນໝູນຄັ້ງທີສອງ ດ້ວຍການໝູນໂໝດ C ໄປແທນທີ່ໂໝດ A ໂດຍໃຊ້ຫຼັກການໝູນຄັ້ງດຽວຈາກໂໝດຂວາໄປຊ້າຍ ເຊີ່ງຈະມີການຍ້າຍການເຊື່ອມໂຍງຂອງທຣີດັ່ງນີ້:

1.ຍ້າຍທຣີຍ່ອຍໄປທາງຊ້າຍຂອງໂໝດ C ໄປເປັນທຣີຍ່ອຍທາງຂວາຂອງໂໝດ A

2.ຍ້າຍໂໝດ A ໄປເປັນໂໝດລູກທາງຊ້າຍຂອງໂໝດ ​C

ຫຼັງຈາກການໝູນໂໝດ C (ໂໝດຮາກຂອງທຣີຍ່ອຍ Y) ຈຳນວນ 2 ຄັ້ງຈະເຮັດໃຫ້ລະດັບຄວາມສູງຂອງທຣີຫຼຸດລົງ 1 ລະດັບ ເຊີ່ງເຮັດໃຫ້ແກ້ບັນຫາຂາດຄຸນສົມບັດຄວາມເປັນເອວີແອລທຣີໄດ້

**ໝູນຂື້ນ**

**Y**

**Z**

**a**

**b**

**X**

**Y**

**b**

**a**

**Z**

**X**

(ກ) ບັນຫາການໝູນຄັ້ງດຽວຈາກໂໝດທາງຂວາໄປຊ້າຍ ລະດັບຄວາມສູງ Y ບໍ່ປ່ຽນ

**Z**

**a**

**b**

**X**

**c**

**Y**

**1**

**Y**

**2**

**Y**

**Z**

**a**

**b**

**X**

**c**

**Y**

**1**

**Y**

**2**

**Y**

**Z**

**a**

**b**

**X**

**c**

**Y**

**1**

**Y**

**2**

(ຂ) ການໝູນສອງຄັ້ງຈາກໂໝດທາງຊ້າຍໄປຂວາ ຕາມດ້ວຍການໝູນໂໝດທາງຂວາໄປຊ້າຍ

ຮູບທີ 10.17 ສະແດງຕົວຢ່າງການໝູນສອງຄັ້ງດ້ວຍໂໝດລູກທາງຂວາ ເນື່ອງຈາກຮູບທີ 10.17 (ກ)

ຕຳແໜ່ງໂໝດ 14 ເຮັດໃຫ້ໂຄ້ງສ້າງໄປນາຣີທຣີຂາດຄຸນສົມບັດຄວາມເປັນເອວີແອລທຣີ ແລະຕ້ອງແກ້ໄຂດ້ວຍການຫຼຸດລະດັບຄວາມສູງຂອງທຣີຍ່ອຍທາງຂວາ (ທີ່ຕຳແໜ່ງໂໝດ 20) ຂອງໂໝດ 14 ເຊີ່ງເຫັນວ່າທຣີຍ່ອຍທາງຂວາ ດັ່ງນັ້ນຈະໃຊ້ວິທີ ການໝູນໂໝດ 16 ຈຳນວນ 2 ຄັ້ງ ເພື່ອຫຼຸດລະດັບຄວາມສູງຂອງທຣີ ມີດັ່ງນີ້:

* ຄັ້ງທີ 1 ໃຫ້ໝູນໂໝດລູກທາງຊ້າຍ (ໂໝດ 16) ຂອງໂໝດ 20 ດ້ວຍການໝູນໂໝດ 16 ໄປແທນທີ່ໂໝດ 20 ດັ່ງຮູບ 10.17 (ຂ) ໂດຍຕ້ອງຍ້າຍໂໝດອື່ນໆ ດັ່ງນີ້

1. ຍ້າຍທີ່ຍ່ອຍໄປທາງຂວາຂອງໂໜດ 16 ໄປເປັນທຣີຍ່ອຍທາງຊ້າຍຂອງໂໜດ 20.
2. ຍ້າຍໂໜດ 20 ໄປຫາລູກທີ່ຖືກຕ້ອງຂອງໂໜດ 16.

* ຄັ້ງທີ 2 ຫມູນໂໜດລູກໄປທາງຂວາ (ໂໜດ 1 6) ຂອງໂໜດ 14 ໂດຍການຫມູນໂໜດ 16 ໄປປ່ຽນໂນດ 14 ຕາມທີ່ສະແດງໃນຮູບ10.17 (ຄ).

1. ຍ້າຍທຣີຍ່ອຍໄປທາງຊ້າຍຂອງໂໜດ 16 ໄປຫາ subtree ໄປທາງຂວາຂອງ node 14.
2. ຍ້າຍ node 14 ໄປຫາລູກໄປທາງຊ້າຍຂອງnode 16.

ຫຼັງຈາກການຫມູນວຽນຂອງໂນດ 16 ຈຳນວນ 2 ຄັ້ງ, ຄວາມສູງຂອງຕົ້ນໄມ້ຈະຫຼຸດລົງ 1 ລະດັບແກ້ໄຂບັນຫາຄຸນສົມບັດຂອງຕົ້ນໄມ້ AVL ຂອງnode 14

**5**

**12**

**20**

**24**

**14**

**16**

**18**

**5**

**3**

**1**

**9**

**12**

**20**

**24**

**14**

**16**

**18**

**5**

**3**

**1**

**9**

**12**

**20**

**24**

**14**

**16**

**18**

(ກ) ການໝູນnode16ໄປແທນທີ່node20. (ຂ) ການໜູນnode16ໄປແທນທີ່node14. (ຄ) ຫຼັງໜູນnode16ຈຳນວນ ຄັ້ງ 2 ເທື່ອ.

1. ຫມູນສອງຄັ້ງດ້ວຍໂນດລູກທາງຊ້າຍ

ຈາກຮູບ 10.18 ສະແດງຕົວຢ່າງໂຄງສ້າງ Tree ກໍລະນີ subtree ຍ່ອຍທາງຊ້າຍຂອງ node A ມີຄວາມສູງຫຼາຍກວ່າ subtree ທາງຂວາແລະເຫັນວ່າເມື່ອໝູນ node ລູກທາງຂວາ (node B) ຂອງ node A ຂື້ນໄປແທນ node A ດັ່ງຮູບ 10.18 (ກ) ກໍບໍ່ສາມາດແກ້ບັນຫາການຂາດຄຸນສົມບັດຄວາມເປັນ AVLT ໄດ້ເນື່ອງຈາກວ່າຄວາມສູງຂອງ subtree ທາງຂວາຂອງ node B (subtree Y) ຍັງຄົງມີລະດັບຄວາມສູງເທົ່າເດີມການໝູນ node B ໄປແທນ node A ຈື່ງເຮັດໃຫ້ Tree ຂາດຄຸນສົມບັດຄວາມເປັນ AVLT ທີ່ node B ແທນ

ໃນກໍລະນີນີ້ສາມາດແກ້ໄຂບັນຫາດ້ວຍການໝູນ 2 ຄັ້ງຈາກ node ລູກໄປຊ້າຍໂດຍຈະແຍກ Y ອອກເປັຮ 2 subtree Y1 ແລະ Y2 ສົມມຸດໃຫ້ node C ເປັນ node ຮາກຂອງ subtree Y ດັ່ງນັ້ນ Y1 ຈະເປັນ subtree ຊ້າຍຂອງ node C,Y2 ເປັນ subtree ຂວາຂອງ node C ດັ່ງຮູບ 10.18 (ກ) ຈາກນັ້ນໝູນ node C ຈຳນວນ 2 ຄັ້ງເພື່ອຫຼຸດລະດັບຄວາມສູງຂອງ subtree Y ດັ່ວນີ້:

* ຫມູນວຽນ 1 ໂດຍການຫມູນວຽນnode c ເພື່ອທົດແທນ node b ໂດຍໃຊ້ຫຼັກການຫມູນຄັ້ງດຽວ.

ຈາກໂນດຂວາຫາຊ້າຍ, ການເຊື່ອມໂຍງຕົ້ນໄມ້ຕໍ່ໄປນີ້ແມ່ນເຄື່ອນຍ້າຍ

1. ຍ້າຍ subtree ໄປທາງຊ້າຍຂອງ node c ໄປຫາ subtree ໄປທາງຂວາຂອງ node b.
2. ຍ້າຍ node b ໄປຫາລູກໄປທາງຊ້າຍຂອງ node c.

* ຈາກນັ້ນຫມູນ 2 ຄັ້ງ ໂດຍການຫມູນວຽນຂອງ node c ເພື່ອປ່ຽນໂນດໂດຍໃຊ້ຫຼັກການຫມູນຄັ້ງດຽວຈາກໂນດຊ້າຍຫາຂວາ, ການເຊື່ອມໂຍງຕົ້ນໄມ້ຈະຖືກເຄື່ອນຍ້າຍ. ດັ່ງນີ້

1. ຍ້າຍ subtree ໄປທາງຂວາຂອງ node c ໄປຫາ subtree ໄປທາງຊ້າຍຂອງ node a.
2. ເຄື່ອນຍ້າຍ node a ໄປເປັນໂໜດລູກທາງຂວາ node c

ຫຼັງຈາກການໜູນ node C (node subtree Y) ຈຳນວນ 2ຄັ້ງ ລະດັບຄວາມສູງຂອງ tree ຫຼຸດລົງ 1 ລະດັບ ແລະສາມາດແກ້ບັນຫາການຂາດຄຸນສົມບັດເປັນ AVLT ດັ່ງຮູບ 10.18 (ຂ)

**หมุนขึ้น**

**Y**

**Z**

**a**

**b**

**X**

**Y**

**b**

**a**

**Z**

**X**

(ກ) ບັນຫາການໝູນຄັ້ງດຽວຈາກ node ຂວາໄປຊ້າຍ ລະດັບຄວາມສູງຂອງ Y ບໍ່ປ່ຽນ

**Z**

**a**

**b**

**X**

**c**

**Y**

**1**

**Y**

**2**

**Y**

**Z**

**a**

**b**

**X**

**c**

**Y**

**1**

**Y**

**2**

**Y**

**Z**

**a**

**b**

**X**

**c**

**Y**

**1**

**Y**

**2**

(ຂ) ການໝູນ 2 ຄັ້ງຈາກ node ຊ້າຍໄປຂວາ ຕາມດ້ວຍການໝູນ node ຂວາໄປຊ້າຍ

**10.3.3 ລຶບໂນດໃນຕົ້ນໄມ້ AVL**

ການລຶບnodes ໃນຕົ້ນໄມ້ AVL ໃຊ້ວິທີການດຽວກັນກັບການລຶບnodes ໃນຕົ້ນbinary ສໍາລັບການຄົ້ນຫາ, ແລະ ຫຼັງຈາກລຶບnode, ໂຄງສ້າງຕົ້ນໄມ້ອາດຈະຖືກຕັດອອກຈາກຕົ້ນໄມ້ AVL ເຊັ່ນກັນ. ດັ່ງທີ່ສະແດງໃນຮູບ 10.19 (ກ) ຫຼັງຈາກຖອດໂນດ 32 ອອກຈາກໂຄງສ້າງຕົ້ນໄມ້ແລ້ວ ຕົ້ນໄມ້ຈະຖືກສ້າງຂຶ້ນຕາມຮູບ32. 10. 19(ຂ) ຊຶ່ງພົບວ່າ ໂຄງສ້າງຂອງຕົ້ນໄມ້ຂາດຄຸນສົມບັດເປັນຕົ້ນໄມ້ AVL ທີ່node 44 ແລະສາມາດໄດ້ຮັບການແກ້ໄຂດ້ວຍຫຼັກການຫມູນວຽນ.ເທື່ອດຽວໂນດຂວາຫາຊ້າຍໂດຍການຫມູນວຽນຂອງໂນດເດັກຂວາ (node 62) ເພື່ອປ່ຽນnode 44. ເພາະsubtree ທີ່ຢູ່ເບື້ອງຂວາສູງກວ່າ subtree ໄປທາງຊ້າຍຂອງnode 44, ດັ່ງທີ່ສະແດງໃຫ້ເຫັນໃນຮູບ10.19 (c)

**5**

**12**

**20**

**24**

**14**

**16**

**18**

**5**

**3**

**1**

**9**

**12**

**20**

**24**

**14**

**16**

**18**

**5**

**3**

**1**

**9**

**12**

**20**

**24**

**14**

**16**

**18**

(ກ) ຕົ້ນໄມ້ກ່ອນລຶບnode 32 (ຂ) ຕົ້ນໄມ້ຫຼັງລຶບnode 32 (ຄ) ຫຼັງຈາກຫມູນnode 62 ເພື່ອປ່ຽນnode 44

**10.4 ຢຸດຕົ້ນໄມ້**

ການຢູດຕົ້ນໄມ້ (Tree Traversal) ຄືການເຂົ້າເຖີງ (Access) ຫຼື ການຢ້ຽມ (Visit) node ໃນ tree ເຊີ່ງການຢຸດຜ່ານ tree ຢ່າງມີປະສິດທິພາບຈະ ເຮັດໃຫ້ການເຂົ້ນເຖີງ node ແບບບໍ່ໍ້ຳກັນໄດ້ຢ່າງວ່ອງໄວແລະເຊື່ອຖືໄດ້ ສາມາດປະມວນຜົນໄດ້ຄົບທຸກ node ໃນ tree

**10.4.1 ການເຂົ້າເຖິງໂນດໃນຕົ້ນໄມ້**

ວິທີເຂົ້າເຖີງ node ຕ່າງໆໃນຕົ້ນໄມ້ມີ 2 ຮູບແບບຄື:

1. ການຢຸດຜ່ານຕົ້ນໄມ້ໃນ ໄບນາຣີແບບແນວກ້ວາງ (Breadth-first) ມີລັກສະນະການເຂົ້າເຖີງແຕ່ລະ node ໃນຕົ້ນໄມ້ທີ່ລະ ລະດັບໂດຍເຂົ້າເຖີງ node ທັງໝົດໃນລະດັກແລກກ່ອນຈື່ງເຂົ້າເຖີງທຸກ node ໃນລະດັບຖັດໄປຈົນຄົບທຸກລະດັບໃນໂຄງສ້າງຕົ້ນໄມ້ ການເຂົ້າເຖີງຂໍ້ມູນແບບນີ້ສາມາດນຳໂຄງສ້າງຄິວມາຊ່ວຍໃນການເກັບຂໍ້ມູນມີຂັ້ນຕອນການເຂົ້າເຖີງຂໍ້ມູນດັ່ງນີ້:

breadth-first(q) {

q.enqueue(p)

while (!q.isEmpty())

q.dequeue(p)

Print item in node p for each child w of p

q.enqueue(w)

}

1. ການຢຸດຜ່ານຕົ້ນໄມ້ໄບນາຣີແບບແນວເລິກ (Depth-first) ແບ່ງເປັນ 2 ລັກສະນະຄື:

2.1) ການເຂົ້າເຖີງແບບ ພຣີອໍເດີ (Preorder-Traversal)

ຂໍ້ມູນຂອງແຕ່ລະໂນດໃນຕົ້ນໄມ້ ຈະຖືກເຂົ້າເຖີງຫຼືຖືກປະມວນຜົນກ່ອນໂນດລູກຫຼານຂອງຕົນເອງ ມີອັນກໍຣິທືມ ໃນການດຳເນີນການດັ່ງນີ້:

preOrder(p){

print item in node p

for each child w of p

preOrder (w)

}

2.2) ການເຂົ້າເຖີງແບບໂພສອໍເດີ (Postorder-Traversal)

ຂໍ້ມູນຂອງແຕ່ລະໂນດໃນຕົ້ນໄມ້ຈະຖືກເຂົ້າເຖີງຫຼືຖືກປະມວນຜົນຫຼັງຈາກການເຂົ້າເຖີງຂໍ້ມູນຂອງໂນດລູກຫຼານຂອງຕົນເອງ ມີອັນກໍຣິທືມໃນການດຳເນີນການດັ່ງນີ້:

postOrder(p){

for each child w of p

postOrder (w)

print item in node p }

**B**

**F**

**D**

**C**

**H**

**I**

**G**

**E**

**A**

**J**

**Child Node**

**ຂອງໂນດ**

ຈາກຕົວຢ່າງຕົ້ນໄມ້ໃນຮູບທີ່ 10.20 ລຳດັບການເຂົ້າເຖີງແຕ່ລະໂນດໃນຕົ້ນໄມ້ແບບແນວກ້ວາງຄື A B E F C D G H I J ສ່ວນລຳດັບການເຂົ້າເຖີງໂນດແບບພຮີອໍເດີຄື A B C D E F G I J ແລະລຳດັບການເຂົ້າເຖີງແຕ່ລະໂນດໃນຕົ້ນໄມ້ແບບ ໂພສອໍເດີຄື C D B E G H J I F A

**10.4.2 ການເຂົ້າເຖີງໂນດໃນໄບນາຣີ**

ການຢຸດຜ່ານຕົ້ນໄມ້ໄບນາຣີ (Traversing Binary Tree) ສາມາດເຮັດໄດ້ 4 ວິທີ ແຕ່ລະວິທີແຕກຕ່າງກັນບ່ອນທີ່ຈຸດເລີ່ມຕົ້ນໃນການຢຸດຜ່ານຕົ້ນໄມ້ວ່າຈະເລີ່ມຕົ້ນທີ່ໂນດຮາກ ຫຼື subtree ຊ້າຍ () ຫຼື subtree ຂວາ () ກ່ອນ ຫຼືຈະເຂົ້າເຖີງໂນດຂໍ້ມູນໃນແຕ່ລະລຳດັບຂອງຕົ້ນໄມ້ ແຕ່ລະວິທີມີ

1. ການຢຸດຜ່ານຕົ້ນໄມ້ໄບນາຣີແບບອີນອໍເດີ (Inorder Traversal) ມີລັກສະນະການເຂົ້າເຖີງແບບ ຄືຈະເຂົ້າເຖີງທຸກໂນດໃນ subtree ຂວາຂອງໂນດໃດໆ ກ່ອນຈະເຂົ້າເຖີງຂໍ້ມູນຂອງໂນດນັ້ນ ແລ້ວຕາມດ້ວຍທຸກໂນດ subtree ຊ້າຍຂອງໂນດນັ້ນ ຕາມລຳດັບມີຂັ້ນຕອນດັ່ງນີ້

inOrder(p) {

if hasLeft (p)

inOrder (leftChild (p))

Print item in node p if hasRight (p)

inOrder (rightChild (p))

}

1. ການຢຸດຜ່ານຕົ້ນໄມ້ໄບນາຣີແບບພຣີອໍເດີ (Preorder Traversal) ມີລັກສະນະການເຂົ້າເຖີງແບບ ຄືຈະເຂົ້າເຖີງຂໍ້ມູນໃດໆກ່ອນຈື່ງໄປຍັງທຸກໂນດໃນ subtree ຂວາແລະຕາມດ້ວຍທຸກໂນດ subtree ຊ້າຍຂອງໂນດນັ້ນ ມີຂັ້ນຕອນການເຮັດວຽກດັ່ງນີ້

preOrder(p) {

Print item in node p

if hasLeft (p)

preOrder (leftChild (p))

if hasRight (p)

preOrder (rightChild (p))

}

1. ການຢຸດຜ່ານຕົ້ນໄມ້ໄບນາຣີແບບ ໂພສອໍເດີ (Postorder Traversal) ມີລັກສະນະການເຂົ້າເຖີງແບບ ຄືຈະເຈົ້າເຖີງທຸກໂນດໃນ subtree ຂວາຂອງໂນດໃດໆກ່ອນຕາມດ້ວຍທຸກ ໂນດ subtree ຊ້າຍຂອງໂນດນັ້ນ ແລ້ວຈື່ງຈະເຂົ້າເຖີງຂໍ້ມູນຂອງໂນດນັ້ນ ມີຂັ້ນຕອນດັ່ງນີ້:

postOrder(p) {

if hasLeft (p)

postOrder (leftChild (p))

if hasRight (p)

postOrder (rightChild (p))

Print item in node p

}

1. ການຢຸດຜ່ານຕົ້ນໄມ້ໄບນາຣີແບບແນວກ້ວາງ (Breadth-first) ມີລັກສະນະການເຂົ້າເຖີງແຕ່ລະໂນດໃນຕົ້ນໄມ້ທີລະລຳດັບເຊັ່ນດຽວກັນກັບຕົ້ນໄມ້ທົ່ວໄປ

ຮູບທີ 10.21 ສະແດງຕົວຢ່າງຕົ້ນໄມ້ໄບນາຣີໃນ ຮູບທີ 10.21 (ກ) ເຊີ່ງແບ່ງເປັນ subtree ຊ້າຍແລະ subtree ຂວາ ຜົນການເຂົ້າເຖີງຂໍ້ມູນໃນຕົ້ນໄມ້ໄບນາຣີ ແບບອີນອໍເດີ ແບບພຣີອໍເດີ ແບບໂພສອໍເດີ ແລະແບບແນວກວ້າງສະແດງດັ່ງຮູບ 10.21 (ຂ)-(ຈ) ຕາມລຳດັບ

**B**

**E**

**C**

**A**

**F**

**A**

**B**

**E**

**C**

**D**

(ກ) ໂຄງສ້ງຂໍ້ມູນຄົ້ນໄມ້ໄບນາຣີ (ຂ) ຜົນການເຂົ້າເຖີງແບບອີນອໍເດີ

|  |  |  |
| --- | --- | --- |
| **A** | | |
|  | **B** |  |
| **C D F** |

**A**

**B**

**E**

**C**

**D**

**F**

**A**

**B**

**E**

**C**

**D**

**F**

**10.4.3 ການປະຍຸກໃຊ້ງານການຢຸດຜ່ານຕົ້ນໄມ້**

ການຢຸດຜ່ານຕົ້ນໄມ້ສາມາດຖືກນຳໄປປະຍຸກໃຊ້ເພື່ອການຄົ້ນຫາແລະສະແດງຜົນຂໍ້ມູນໃນຕົ້ນໄໃ້ ເຊັ່ນການສະແດງໂຄງສ້າງການຈັດເກັບຂໍ້ມູນຂອງຟາຍແລະໂຟນເດີລະບົບການຈັດການຟາຍ (File system) ໃນຄອມພີວເຕີດັ່ງຮູບ 10.22

**1**

**comps**

**hw1.java**

**hw1.doc**

**5**

**4**

**6**

**7**

**hw3.java**

**hw2.doc**

**8**

**2**

**9**

**hw2.java**

**homeworks**

**programs**

**cou**

**rse.txt**

**3**

ຮູບທີ 10.22 ໂຄງສ້າງການຈັດເກັບຂອງລະບົບໄຟລ໌ໃນຄອມພິວເຕີແລະລຳດັບການເຂົ້າເຖີງແບບພຣີອໍເດີ

ເມື່ອເຂົ້າເຖີງຂໍ້ມູນແບບພຣີອໍເດີ ແລະແບບໂພສອໍເດີຈະສາມາດສະແດງໂຄງສ້າງການຈັດເກັບໄຟລ໌ແລະໂຟນເດີໄດ້ດັ່ງນີ້:

ຜົນລັບການເຂົ້າເຖີງແບບພຣິອໍເດີ ຜົນລົບການເຂົ້າເຖີງແບບໂພສອໍເດີ

comps hw1.doc

homeworks hw2.doc

hw1.doc homeworks

hw2.doc hw1.java

programs hw2.java

hw1.java hw3.java

hw2.java programs

hw3.java course.txt

course.txt comps

ສຳລັບຕົວຢ່າງການນຳໂຄງສ້າງຕົ້ນໄມ້ໄບນາຣີໄປປະຍຸກໃຊ້ງານ ເຊັ່ນການສ້າງຕົ້ນໄມ້ສຳລັບການຕັດສີນໃຈ (Decision Tree) ໂດຍໂນດພາຍໃນ (Internal nodes) ປະກອບດ້ວຍປະໂຫຍກຄຳຖາມໃຫ້ຕອບວ່າ Yes or No ໂດຍແຕ່ລະຄຳຖາມຕໍ່ໄປທຮ່ໂນດລູກຊ້າຍ ແລະຫາກສີ້ນສຸດການຕັດສີນໃຈ ດັ່ງຕົວຢ່າງໃນຮູບທີ 10.23

ຮູບທີ 10.32 ຫາກຕອບ Yes ທີ່ຄຳຖາມ “ຕ້ອງການອາຫານແບບດ່ວນ” ຈະໄປຍັງຄຳຖາມ “ຕ້ອງການດື່ມກາເຟ” ຖ້າຫາກຕອບ Yes ທີ່ຄຳຖາມນີ້ ກໍຈະສີ້ນສຸດການຕັດສີນໃຈເລືອກສະຖານທີ່

ຕ້ອງການອາຫານແບບຈານດ່ວນ

**?**

**ຕ້ອງການກາເຟ**

**?**

**ຕ້ອງການເຄັກ**

**?**

**?**

**ຟູດແອນດອບພີ**

ມາລີນຟູນ

**ເຝີ**

**ເສັ້ນແລະໄຂ່**

**Yes**

**No**

**Yes**

**No**

**Yes**

**No**

ຮູບທີ 10.23 ຕົວຢ່າງຕົ້ນໄມ້ໄບນາຣີສຳລັບການຕັດສີນໃຈເລືອກສະຖານທີ່ສຳລັບກີນອາຫານ

ຮູບທີ 10.24 ສະແດງຕົວຢ່າງການປະຍຸກໃຊ້ໂຄງສ້າງຂໍ້ມູນຕົ້ນໄມ້ໄບນາຣີໃນການຈັດເກັບຄຳນວນທາງຄະນິດສາດຂອງການສະແດງອອກ ໂດຍໂນດພາຍໃນ (Internal nodes) ໃຊ້ສຳລັບຈັດເກັບການດຳເນີນການທາງຄະນິດສາດຂອງເຄື່ອງໝາຍ + -\* ແລະ/ ສ່ວນໂນດໃບຈະເປັນຕົວຖືກດຳເນີນການ (Operands) ເຊີ່ງອາດເປັນຕົວແປ ຫຼືຄ່າຄົງທີ່ສຳລັບໃຊ້ໃນການດຳເນີນການ

**+**

**\***

**\***

**-**

**3**

**x**

**1**

**4**

**y**

ຮູບທີ 10.24 ການຈັດເກັບການສະແດງ (3\*(x-1) + (4\*y)) ໃນໂຕງສ້າງຂໍ້ມູນຄົ້ນໄມ້ໄບນາຮີ

ການເຂົ້າເຖີງໂນດຕົ້ນໄມ້ແບບພຣິອໍເດີ ສາມາດສະແດງຄ່າການສະແດງອອກໃນລັກສະນະແບບ prefix ຄື +\*3-x1\*4y ແລະຫາກເຂົ້າເຖີງໂນດໃນຕົ້ນໄມ້ແບບໂພສອໍເດີ ຈະໄດ້ສະແດງອອກແບບ postfix ຄື 3 x1-\*4y\*+ ໂດຍສະແດງທັງແບບ prefix ແລະ postfix ທີ່ຈະສາມາດແປງເປັນການສະແດງ (3\*(x-1) + (4\*y)) ໂດຍໃຂ້ສະແຈັກຊ່ວຍໃນການຈັດເກັບຂໍ້ມູນ ຢ່າງໃດກໍຕາມຫາກເຂົ້າເຖີງໂນດຕົ້ນໄມ້ແບບອີນອໍເດີ ກໍສາມາດສະແດງລັກສະນະ infix ໄດ້ຄື 3\* x-1+4\*y ພຍງແຕ່ໄດ້ຍັງມີລຳດັບໃນການຄຳນວນທີ່ຜິດຈື່ງຕ້ອງໃສ່ວົງເລັບເພີ່ມລະຫວ່າງການເຂົ້າເຖີງໂນດເພີ່ມເຕີມ

ນອກຈາກນີ້ ການຈັດເກັບຂໍ້ມູນການຄຳນວນໃນຕົ້ນໄມ້ໄບນາຣີຍັງສາມາດຖືກນຳໄປປະຍຸກໃຊ້ເພື່ອການດຳເນີນການການຄຳນວນຄ່າໄດ້

ItemType evalExpr (BinaryTree p) {

var a, b : ItemType

if isExternal (p)

return p.getItem()

else

a← evalExpr (leftChild(p))

b← evalExpr (rightChild(p))

x ← p.getItem ( ) //operator stored at node p

return Compute(a, x, b)

}

**10.5 ການນຳຕົ້ນໄມ້ໃຊ້**

ການນຳຕົ້ນໄມ້ໄປໃຊ້ໃນຄອມພິວເຕີດ້ວຍການຂຽນພາສາຄອມພິວເຕີ ສາມາດສ້າງດ້ວຍ ອາເຣ ຫຼື ລິງລິດ

**10.5.1 ການອອກແບບການສ້າງຕົ້ນໄມ້**

ໂຄງສ້າງຂໍ້ມູນຕົ້ນໄມ້ທີ່ສ້າງດ້ວຍລິງລິດ ແຕ່ລະສາມະຊີກຈະຖືກເຊື່ອມໂຍງກັນດ້ວຍໂນດຂໍ້ມູນ ດັ່ງຮູບ 10.25 ຈາກຂໍ້ມູນໃນການສ້າງຕົ້ນໄມ້ປະກອບດ້ວຍຈຳນວນສາມະຊີກທັງໝົດ 5 ຄ່າ ຄື A B C D ແລະ E ສາມາດອອກແບບການຈັດເກັບຂໍ້ມູນດ້ວຍລິງລິດ ແຕ່ລະໂນດຂໍ້ມູນປະກອບດ້ວຍ item ທີ່ເກັບຄ່າສາມະຊີກຂອງຕົ້ນໄມ້ parent ເຮັດໜ້າທີ່ເຊື່ອມໂຍງໄປໂນດພໍ່ ຕົວ firstChild ເຮັດໜ້າທີ່ເຊື່ອມໂຍງໄປຍັງໂນດລູກທີ່ເປັນໂນດແລກຂອງລິດ ທີ່ເກັບລາຍການເຊື່ອມໂຍງຊອງໂນດລູກແລະ sibling ເຮັດໜ້າທີ່ເຊື່ອມໂຍງໄປຍັງໂນດຖັດໄປເປັນໂນດພີ່ນ້ອງກັນ

typedef struct node\_content {

itemType item;

struct node\_content \*parent;

struct node\_content \*firstChild;

struct node\_content \*sibling;

} node\_type; root node\_type \*root;

**root**

**B**

**Æ**

**C**

**Æ**

**E**

|  |  |  |  |
| --- | --- | --- | --- |
| **A** | **Æ** |  | **Æ** |

**A**

**B**

**C**

**D**

**E**

**F**

|  |  |  |  |
| --- | --- | --- | --- |
| **D** |  | **Æ** | **Æ** |

|  |  |  |  |
| --- | --- | --- | --- |
| **F** |  | **Æ** | **Æ** |

(ກ) ຕົວຢ່າງຂອງໂຄງສ້າງຕົ້ນໄມ້. (ຂ) ຕົວຢ່າງຂອງໂຄງສ້າງຕົ້ນໄມ້ທີ່ຖືກສ້າງຂື້ນດ້ວຍ

ລາຍຊື່ລິງ.

(ຮູບ 10.25 ຕົວຢ່າງຂອງການສ້າງໂຄງສ້າງຕົ້ນໄມ້ທີ່ມີລາຍຊື່ linkຈາກຮູບທີ 10.25 (ຂ),

ຕົວຢ່າງຂອງການເກັບຂໍ້ມູນໃນ node ສາມາດອະທິບາຍໄດ້ດັ່ງຕໍ່ໄປນີ້: o ຂໍ້ ທຳ ອິດຂອງຕົ້ນໄມ້ມີຕົວຊີ້ທີ່ມີຊື່ວ່າຮາກ, ເຊິ່ງເຮັດ ໜ້າ ທີ່ເຊື່ອມຕໍ່ກັບຮາກ (node ​​A) ຂອງ ຕົ້ນໄມ້ຂອງຂໍ້ B ແມ່ນດັ່ງຕໍ່ໄປນີ້:

* ຕົວຊີ້ແລະພໍ່ແມ່ອ້າຍເອື້ອຍນ້ອງແມ່ນຕົວເປົ່າເພາະວ່າຂໍ້ A ແມ່ນຂໍ້ຂອງຮາກຂອງຕົ້ນໄມ້.
* ຕົວຊີ້ຕົວ ທຳ ອິດຂອງ node A ແມ່ນພົວພັນກັບ node B, ຂໍ້ ທຳ ອິດຂອງລູກທີ່ຢູ່ໃນຕໍາແ

ໜັງB, C, ແລະ D, ດັ່ງນັ້ນ ທັງສາມຂໍ້ນີ້ຈະມີຕົວຊີ້ຂອງພໍ່ແມ່ດຽວກັນກັບຂໍ້ A.

ຄຸນລັກສະນະຂອງ node D ມີດັ່ງນີ້.

* ຕົວຊີ້ຂອງຜູ້ປົກຄອງເຊື່ອມໂຍງກັບຂໍ້ A ເປັນ node ຂອງລູກ.
* ຕົວຊີ້ຂອງອ້າຍເອື້ອຍນ້ອງແມ່ນກ່ຽວຂ້ອງກັບຄ່າທີ່ບໍ່ມີຄ່າເພາະມັນແມ່ນຂໍ້ສຸດທ້າຍຊີຂອງ nling B, C, ແລະ D.
* ຕົວຊີ້ ທຳ ອິດຂອງຕົວອັກສອນ ທຳ ອິດແມ່ນບໍ່ມີຂໍ້ມູນເພາະວ່າຂໍ້ D ບໍ່ມີລູກ.

**10.5.2 ການອອກແບບການກໍ່ສ້າງຕົ້ນໄມ້ແບບຖານສອງ**

ການສ້າງໂຄງສ້າງຂໍ້ມູນກ່ຽວກັບຕົ້ນໄມ້ຖານສອງກັບລາຍຊື່ການເຊື່ອມໂຍງ ແຕ່ລະສະມາຊິກຂອງຕົ້ນໄມ້ແມ່ນເຊື່ອມຕໍ່ໂດຍ node ຂໍ້ມູນ, ດັ່ງທີ່ສະແດງໃນຮູບທີ 10. 26. ການອອກແບບຕົ້ນໄມ້ທີ່ມີຕົວຢ່າງການເຊື່ອມຕໍ່ແມ່ນສະແດງໃຫ້ເຫັນ. ແຕ່ລະຂໍ້ຂໍ້ມູນປະກອບດ້ວຍລາຍການທີ່ຖືເອົາສະມາຊິກຂອງຕົ້ນໄມ້, ຕົວຊີ້ວັດແມ່ທີ່ເຊື່ອມຕໍ່ກັບ node parent, pointer ຊ້າຍທີ່ເຊື່ອມຕໍ່ກັບ node child left, ແລະສິດທີ່ເຊື່ອມຕໍ່. ເນື່ອງຈາກວ່າມັນເປັນຕົ້ນໄມ້ຖານສອງ, ມີພຽງແຕ່ຂໍ້ລູກທີ່ຢູ່ເບື້ອງຊ້າຍແລະຂໍ້ຂອງລູກທີ່ຖືກຕ້ອງ. ກຳ ນົດຕົວຊີ້ຂອງເດັກນ້ອຍໃຫ້ເປັນ null. ເພື່ອສ້າງຕົ້ນໄມ້, ຕົວຊີ້ທີ່ມີຊື່ວ່າຮາກຈະຖືກມອບ ໝາຍ ໃຫ້ເຮັດເປັນຕົວເຊື່ອມຕໍ່ກັບຂໍ້ (ຮາກ B) ຂອງຕົ້ນໄມ້ເຊິ່ງຂໍ້ມູນຂອງຕົ້ນໄມ້ນັ້ນຈະບໍ່ມີພໍ່ ຂໍ້. ເພາະສະນັ້ນ, ຂໍ້ ກຳ ນົດຮາກແມ່ນມີພໍ່ແມ່ຕົວຊີ້ແບບບໍ່ມີສຽງ, ດັ່ງທີ່ສະແດງຢູ່ໃນຮູບ 10.26 (b). ສ້າງຕົ້ນໄມ້ຖານສອງທີ່ມີບັນຊີລາຍຊື່ການເຊື່ອມຕໍ່ທີ່ປະກອບດ້ວຍ ຈຳ ນວນສະມາຊິກທັງ ໝົດ ແມ່ນ 5 ຄຸນຄ່າ, A, B, C, D ແລະ E ດັ່ງທີ່ສະແດງໃນຮູບ.10.26 (ກ)

**ຄືຕົວຢ່າງການສອອກແບບຕົົ້ນໄມ້ດັ່ງລຸ່ມນີ້:**

**Ty pedef struct node\_content { itemType item;**

**struct node\_content \*parent; struct node\_content \*left, \*right;**

**} node\_type; node\_type \*root;**

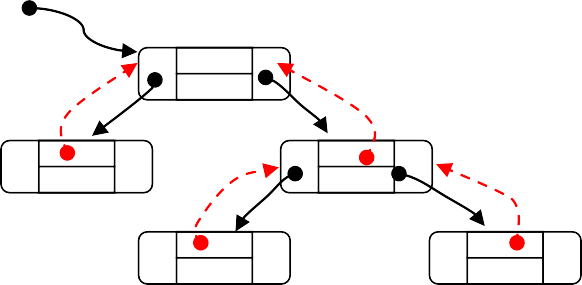
**B**

**A**

**D**

**C**

**E**



**root**

**Æ**

**B**

**Æ**

**A**

**Æ**

**D**

**Æ**

**C**

**Æ**

**Æ**

**E**

**Æ**

(ກ) ຕົວຢ່າງໂຄງສ້າງຂອງຕົ້ນໄມ້ຖານສອງ . (ຂ) ການເກັບຂໍ້ມູນຕົ້ນໄມ້ຖານສອງສ້າງດ້ວຍລາຍຊື່ການເຊື່ອມຕໍ່.

ຮູບພາບ 10.26 ຕົວຢ່າງຂອງການສ້າງໂຄງສ້າງຂອງຕົ້ນໄມ້ຖານສອງທີ່ມີລາຍຊື່ລິງ. ມັນຍັງເປັນໄປໄດ້ທີ່ຈະອອກແບບການສ້າງຕົ້ນໄມ້ດ້ວຍຂະບວນ. ຕົວຢ່າງໃນຮູບທີ 10.27 ສະແດງໃຫ້ເຫັນຕົວຢ່າງຂອງໂຄງສ້າງຂອງຕົ້ນໄມ້ໄບນາລີປະກອບມີ 11 ຂໍ້, ດັ່ງທີ່ສະແດງໃນຮູບທີ 10. 27 (ກ) node AA ເປັນຂໍ້ຂອງຮາກ.

ໃນຮູບທີ 10. 27 (ກ) node AA ເປັນຂໍ້ຂອງຮ(ກ) ຕົວຢ່າງ

**AA**

**BB**

**CC**

**DD**

**EE**

**GG**

**HH**

**FF**

**JJ**

**KK**

**LL**

|  |  |  |  |
| --- | --- | --- | --- |
|  | **item** | **left** | **right** |
| **1** | **AA** | **2** | **3** |
| **2** | **BB** | **4** | **5** |
| **3** | **CC** | **7** | **8** |
| **4** | **DD** | **0** | **0** |
| **5** | **EE** | **6** | **0** |
| **6** | **FF** | **0** | **0** |
| **7** | **GG** | **0** | **0** |
| **8** | **HH** | **9** | **10** |
| **9** | **JJ** | **11** | **0** |
| **10** | **KK** | **0** | **0** |
| **11** | **LL** | **0** | **0** |
| **12** |  | **13** | **0** |
| **13** |  | **14** |  |
| **14** |  | **15** |  |
| **15** |  | **0** |  |

ໂຄງສ້າງຂໍ້ມູນກ່ຽວກັບຕົ້ນໄມ້ຖານສອງຊັ ຂ ຮູບສະແດງ

10.27 ຕົວຢ່າງຂອງການສ້າງໂຄງສ້າງຂອງຕົ້ນໄມ້ຖານສອງທີ່ມີອາເລ.

truct binary\_node { itemType item[10];

int left, right; }

node\_type;

node\_type tree[16];

int root, avail

ການອອກແບບໂຄງສ້າງຕົ້ນໄມ້ ກັບຂບວນຂ້າງເທິງ ຈຳ ນວນທັງ ໝົດ 16 ຂໍ້ສາມາດເກັບຮັກສາໄວ້ໃນຕົ້ນໄມ້ node ຮາກຈະຖືກເກັບໄວ້ໃນ array [1], ດັ່ງນັ້ນຕົວແປຮາກເກັບຄ່າ 1 ເພາະວ່າ node ຮາກ (node ​​AA) ມີລູກທີ່ຢູ່ເບື້ອງຊ້າຍຂອງມັນ BB ແລະ ເດັກທີ່ຖືກຕ້ອງແມ່ນ node CC, ເຊິ່ງເກັບຮັກສາໄວ້ໃນຕົ້ນໄມ້ [2] ແລະຕົ້ນໄມ້ [3] ຈັດລຽງລໍາດັບຕາມ ລຳ ດັບ, ສະນັ້ນ, ເບື້ອງຊ້າຍແລະຂວາໃນຕົ້ນໄມ້ [1] ແມ່ນເທົ່າກັບ 2 ແລະ 3 ຕາມ ລຳ ດັບ, ຊຶ່ງ ໝາຍ ຄວາມວ່າສະຖານທີ່ຂອງເດັກນ້ອຍຢູ່ເບື້ອງຊ້າຍ

ແລະຂວາຂອງ AA ແມ່ນສະແດງຢູ່ໃນຮູບ 10.27 (ຂ). ຕຳ ແໜ່ງ ແຖວທີ່ບໍ່ມີບ່ອນຫວ່າງເພື່ອເກັບຂໍ້ມູນຂໍ້ມູນ ໃໝ່ ເພື່ອເພີ່ມໃສ່ຕົ້ນໄມ້. ທີ່ນີ້ຕົວແປທີ່ມີຢູ່ຈະຖືຄ່າ 12.

**10.5.3 ການອອກແບບກໍ່ສ້າງຕົ້ນໄມ້ AVL**

ພາກນີ້ອະທິບາຍເຖິງຕົວຢ່າງຂອງການອອກແບບໂຄງສ້າງ AVL Tree ທີ່ມີຊື່ link. ການ ນຳ ໃຊ້ຕົວຢ່າງຂອງການສອນພາສາ C ໃນຮູບແບບການຂຽນໂປແກຼມທີ່ເນັ້ນໃສ່ວັດຖຸ. ລາຍລະອຽດມີດັ່ງນີ້.

1. ການອອກແບບໂຄງສ້າງ Node ສຳ ລັບການເກັບຂໍ້ມູນໃນ AVL Tree ດັ່ງຕໍ່ໄປນີ້:

typedef int item Type;

typedef struct AvlNodeData

{

itemType element;

// the data in the node struct AvlNodeData \*left;

struct AvlNodeData \*right;

int height; // Height of node }

AvlNode;

2) ການອອກແບບຫ້ອງຮຽນ ສຳ ລັບການຈັດການໂຄງສ້າງຂໍ້ມູນ AVL Tree ໂດຍການອອກແບບຊັ້ນຮຽນ ສຳ ລັບໂຄງສ້າງຕົ້ນໄມ້ໂດຍການ ກຳ ນົດຕົວປ່ຽນຮາກເພື່ອຊີ້ໃຫ້ເຫັນຮາກຂອງຕົ້ນໄມ້ທີ່ອອກແບບມາຂ້າງເທິງ, ແລະອອກແບບການຈັດຕັ້ງປະຕິບັດ. ເພື່ອປະຕິບັດການ ດຳ ເນີນງານຕ່າງໆ ສຳ ລັບການ ໝູນ ໃຊ້ຂໍ້ມູນໃນໂຄງສ້າງຕົ້ນໄມ້, ເຊັ່ນວິທີການ insert New Node () ສຳ ລັບເພີ່ມຂໍ້ມູນເຂົ້າໃນຕົ້ນໄມ້, ແລະອື່ນໆ, ລາຍລະອຽດມີດັ່ງນີ້:

class AvlTree{

private: AvlNode \*root;

public:

AvlTree( ) { //Constructor

root = NULL;//empty tree

}

void insert New node( item Type x );

private: //private methods

AvlNode \*create New Node (item Type data);

AvlNode \*rotate Right Child(AvlNode \*a );

AvlNode \*rotate Left Child(AvlNode \*a );

AvlNode \*double Rotate Right Child(AvlNode \*a );

AvlNode \*double Rotate LeftChild(AvlNode \*a );

int MAX(int x, int y);

int Height( AvlNode \*p );

AvlNode \*insert (item Type x, AvlNode \*t );}

**ຕໍ່ໄປນີ້ແມ່ນຕົວຢ່າງຂອງການອອກແບບທີ່ເຮັດວຽກ. ຂອງແຕ່ລະການປະຕິບັດງານ ປະຕິບັດພື້ນຖານຂອງ AVL3.**



1) createNewNode () ແມ່ນ ໜ້າ ທີ່ ສຳ ລັບ ສຳ ລັບການເພີ່ມຄ່າຂໍ້ມູນ ໃໝ່ ທີ່ຈະເກັບໄວ້ໃນຕົ້ນໄມ້, ມີ ຄຳ ສັ່ງພາສາ C ດັ່ງຕໍ່ໄປນີ້:

ສ້າງ node ໃຫມ່ ເພື່ອເກັບຂໍ້ມູນ, ກຳ ນົດຈຸດຊີ້ທາງຊ້າຍ (ເບື້ອງຊ້າຍ) ແລະຂວາ (ຂວາ) ໃສ່ NULL

ແລະຕັ້ງຄ່າຄວາມສູງຂອງ node ໃຫ້ 0.

AvlNode \*createNewNode (itemType data){

AvlNode \*new\_node;

new\_node = new AvlNode;

new\_node ->element= data;

new\_node->left=new\_node ->right = NULL;

new\_node ->height = 0;

return(new\_node);

}



2) rotateRightChild () ແມ່ນ ໜ້າ ທີ່ ສຳ ລັບ ສຳ ລັບການ ໝູນ ວຽນແບບດຽວທີ່ ໝຸນ ຂໍ້ມູນເດັກຈາກຂວາຫາຊ້າຍ, ມີ ຄຳ ສັ່ງຄືໃນພາສາ C ດັ່ງຕໍ່ໄປນີ້:

ໝຸນ ເສັ້ນເດັກທີ່ ເໝາະ ສົມ (node ​​b) ເພື່ອທົດແທນ node a

AvlNode \*rotateRightChild(AvlNode \*a ) {

AvlNode \*b = a->right;

a->right = b->left; b->left = a;

a->height = MAX( Height( a->left ),

Height( a->right ) ) + 1;

b->height = MAX( Height( b->right ),

a->height ) + 1; return b;

}



ຮູບສະແດງ 10.28 ສະແດງຕົວຢ່າງທີ່ node ຂໍ້ມູນ 2 ມີຄວາມສູງຂອງ subtree ຂວາ (ລວງກວ້າງຂອງ

Node 4) ແມ່ນແຕກຕ່າງຈາກລະດັບຄວາມສູງຂອງ ດຳ ລັດດ້ານຊ້າຍ. (ລະດັບຄວາມສູງຂອງຂໍ້ 1) ເທົ່າກັບ 2 ຈະມີຜົນບັງຄັໃຊ້, ເຊິ່ງກໍ່ໃຫ້ເກີດ

ໝຸນ ເດັກນ້ອຍທີ່ຖືກຕ້ອງເພື່ອທົດແທນຂໍ້ 1.

ໃຫ້ຈຸດ ໜຶ່ງ ຫາຂໍ້ 1 ແລະປະຕິບັດ ໜ້າ ທີ່. Rotate Right Child (a) ພາຍໃນ ໜ້າ ທີ່ເຮັດວຽກຄືດັ່ງນີ້:

1) ໃຫ້ b ຊີ້ໄປທີ່ຂໍ້ເດັກນ້ອຍທີ່ຖືກຕ້ອງຂອງ a ດ້ວຍ ຄຳ ສັ່ງ b = a-> ສິດດັ່ງທີ່ສະແດງໃນຮູບ 10.28 (a).

2) Rotate node b ເພື່ອທົດແທນ node a ດ້ວຍ ຄຳ ສັ່ງ a-> right = b-> ຊ້າຍແລະ b-> left = a ຜົນໄດ້ຮັບສະແດງຢູ່ໃນຮູບ 10.28 (b).

**b**

**a**

**b**

ຄວາມສູງ

**ເທົ່າກັບ**

**2**

**4**

**6**

**5**

**1**

**3**

**7**

ອເ

**ຄວາມສູງເທົ່າ 2**

**a**

**4**

**6**

**2**

**5**

**7**

**3**

**1**

ຄວາມສູງເທົ່າ1

ຄວາມສູງເທົ່າ2

(ກ) ຕົ້ນໄມ້ທີ່ບໍ່ ເໝາະ ສົມຕົ້ນໄມ້ AVL ທີ່ຕັ້ງຢູ່ a. (ຂ) ຕົ້ນໄມ້ພາຍຫຼັງທີ່ ໝຸນ ຂວາຫາຊ້າຍ

ຮູບທີ 10.28 ການ ໝູນ ວຽນຈາກຂວາຫາ node ຊ້າຍ ຫຼັງຈາກທີ່ເດັກນ້ອຍທີ່ຖືກຕ້ອງຖືກ ໝູນ ວຽນ, ຄືໃນຮູບ 10.28 (b), ລະດັບຄວາມສູງຂອງບົດບັນຍັດ a ແລະ b ໄດ້ຖືກ ກຳ ນົດຄືນ ໃໝ່ ໂດຍພິຈາລະນາມູນຄ່າສູງສຸດຂອງລະດັບຄວາມສູງລະຫວ່າງລັດຖະມົນຕີຂວາແລະຊ້າຍ.

a->height = MAX( Height( a->left ), Height( a->right ) ) + 1;

b->height = MAX( Height( b->right ), a->height ) + 1;

1) ເພື່ອຊອກຫາມູນຄ່າສູງສຸດ, ດຳ ເນີນການ ຄຳ ສັ່ງ MAX () ດັ່ງຕໍ່ໄປນີ້:

int MAX (int x, int y) {ກັບຄືນ ((x)> (y)? (x): (y)); }

2) ຊອກຫາລະດັບຄວາມສູງຂອງບົດບັນຍັດ ໜ້າ ທີ່ Height () ເອີ້ນວ່າດັ່ງຕໍ່ໄປນີ້:

ຄວາມສູງ int (AvlNode \* p) {

ຖ້າ (p! = NULL) ກັບຄືນ (p->

3) rotateLeftChild () ແມ່ນ ໜ້າ ທີ່ ສຳ ລັບ ໃນກໍລະນີຂອງອະນຸສັນຍາຊ້າຍຂອງຂໍ້ a, ລະດັບຄວາມສູງແມ່ນສູງກວ່າລະດັບຂອງຂໍ້ n.

**AvlNode \*rotateLeftChild(AvlNode \*a ){ AvlNode \*b = a->left;**

ມີຕົວແປຂຫາຈຸດຍ່ອຍຂອງຊ້າຍຂອງຂໍ້ກ. a

**a->left = b->right; b->right = a;**

**a->height = MAX( Height( a->left ),**

ປັບລະດັບຄວາມສູງຂອງ a ແລະ b ຍ່ອຍ.

**Height( a->right ) ) + 1;**

**b->height = MAX( Height( b->left ), a->height ) + 1;**

3.5) doubleRotateRightChild() ແມ່ນ ໜ້າ ທີ່ ສຳ ລັບການ ໝູນ ວຽນຂອງເດັກນ້ອຍທີ່ ເໝາະ ສົມເປັນສອງເສັ້ນດັ່ງຕໍ່ໄປນີ້:

**AvlNode \*doubleRotateRightChild(AvlNode \*a ){**

ຕ້ອງ ໝຸນ ຈາກຊ້າຍຫາຂວາດຽວ. ແລະ ໝຸນ ຈາກຂວາຫາຊ້າຍອີກເທື່ອ ໜຶ່ງ

**a->right = rotateLeftChild( a->right ); return rotateRightChild( a);**

**}**

3.6) doubleRotateLeftChild() ແມ່ນ ຄໍາສັ່ງ ສຳ ລັບການ ໝູນ ວຽນຂອງເ ລຸກສອງເບື້ອງດັ່ງຕໍ່ໄປນີ້:

**AvlNode \*doubleRotateLeftChild(AvlNode \*a ) { a->left = rotateRightChild( a->left );**

ຕ້ອງໄດ້ ໝູນ ວຽນຈາກຂວາຫາຊ້າຍຄັ້ງດຽວ ແລະຫັນຈາກຊ້າຍຫາຂວາອີກເທື່ອ ໜຶ່ງ

**return rotateLeftChild( a );**

**}**

3.7) insertNewnode() ແມ່ນຄໍາສັ່ງສຳລັບການເພີ່ມຂໍ້ມູນໃໝ່ໃນຕົ້ນໄມ້ດັ່ງຕໍ່ໄປນີ້:

ຕ້ອງເລີ່ມຕົ້ນຄົ້ນຫາ ບ່ອນທີ່ຈະໃສ່ຂໍ້ມູນ x

ຈາກ node ຮາກທີ່ມີຄໍາສັ່ງinsert (x, root)

**void insertNewnode( itemType x ) { root = insert( x, root );**

**}**



**ຫຼັງຈາກນັ້ນຊອກຫາ ສະຖານທີ່ຂອງໂຫນດເພື່ອເພີ່ມ node ໃໝ່ ໂດຍການໂທຫາ function () ທີ່ເອີ້ນຄືນ.**

**26**

**ເຮັດວຽກຄືນຈົນກ່ວາທ່ານຊອກຫາ node ບ່ອນທີ່ທ່ານຕ້ອງການໃສ່ node ໃຫມ່. ໃນລະຫວ່າງການແຊກໃສ່ node ໃໝ່, ມີການກວດສອບວ່າການຕິດຕັ້ງຂອງ node ໃໝ່ ແມ່ນ ຕົ້ນໄມ້ບໍ່ ເໝາະ ສົມກັບ AVL Tree ບໍຖ້າມີ, ມັນຈະ ໝຸນ ຕົ້ນໄມ້ໂດຍເລືອກທີ່ຈະ ໝຸນ ມັນສອງຄັ້ງຫຼື ໜຶ່ງ ເທື່ອຂື້ນກັບລັກສະນະຂອງມັນ. ສະຖານທີ່ຂອງ node ທີ່ເຮັດໃຫ້ຂາດ AVL Tree ທີ່ມີລາຍລະອຽດຂອງ ຄຳ ສັ່ງດັ່ງຕໍ່ໄປນີ້.AvlNode \*insert (itemType x, AvlNode \*t ) { if ( t == NULL )**

ຖ້າ t ແມ່ນ NULL, ສ້າງ node ຂໍ້ມູນ.

**t = createNewNode (x);**

ຖ້າຂໍ້ມູນ x ນ້ອຍກວ່າ t-> ອົງປະກອບ, ຫຼັງຈາກນັ້ນໃສ່ node ໃໝ່ ໃສ່ລູກເບື້ອງຊ້າຍຂອງ t

**elseif (x< t->element ) {**

**t->left = insert( x, t->left );**

ຫຼັງຈາກການໃສ່ node ໃຫມ່ໄປຫາບົດບັນຍັດເບື້ອງຊ້າຍຂອງ t, ໃຫ້ກວດເບິ່ງຕົ້ນໄມ້ avl ຢູ່ໃນຄວາມສູງ. ຂອງລັດຖະບັນຍັດໄປທາງຊ້າຍແລະຂວາຂອງ t. ຖ້າຄວາມສູງແຕກຕ່າງກັນໄປ 2, ໃຫ້ຫັນຕົ້ນໄມ້.

**if (Height( t->left ) - Height( t->right ) == 2 ) if ( x< t->left->element )**

**t = rotateLeftChild(t );**

**else t = doubleRotateLeftChild (t);**

**}**

**else if (x> t->element ) {**

ຖ້າ x ໃຫຍ່ກວ່າ t-> ອົງປະກອບ, ໃສ່ node ໃໝ່ ໃສ່ລູກສິດຂອງ t.

**t->right = insert( x, t->right);**

**if (Height( t->right ) - Height( t->left ) == 2 ) if (x > t->right->element )**

ໃນເວລາທີ່ node ໃຫມ່ຖືກໃສ່ເຂົ້າໃນປະໂຫຍກທີ່ຖືກຕ້ອງຂອງ t, ມັນຈະກວດເບິ່ງວ່າຕົ້ນໄມ້ຕ້ອງຖືກ ໝູນ ວຽນຫຼືບໍ່.

**t = rotateRightChild (t);**

**else t = doubleRotateRightChild (t);**

**}**

ປັບລະດັບຄວາມສູງຂອງ node t ໃຫ້ເປັນມູນຄ່າສູງສຸດຂອງລັດຖະສາຂາຊ້າຍແລະຂວາ.

**t->height = MAX( Height( t->left ), Height( t->right ) ) + 1; return t;}**

10.6 ຄົ້ນຫາແລະຈັດຮຽງ ລໍາດັບໂດຍໃຊ້ຕົ້ນໄມ້ຄູ່ໃນການຄົ້ນຫາ

ນື່ອງຈາກວ່າຕົ້ນໄມ້ຖານສອງມີຄຸນສົມບັດທີ່ສຳຄັນທີ່ສຸດໜຶ່ງຄ່າທີ່ສຳຄັນແມ່ນຄ່າ n x ໃດກໍ່ຕາມ,ທັງໝົດແມ່ນຄ່າ ສຳຄັນຂອງ subtree. ປະໂຫຍກທີ່ຢູ່ເບື້ອງຊ້າຍແມ່ນນ້ອຍກວ່າຄ່າທີ່ສຳຄັນຂອງ node x, ແລະຄ່າສຳຄັນທັງໝົດຂອງດຳ ລັດທີ່ຖືກຕ້ອງແມ່ນໃຫຍ່ກວ່າມູນຄ່າທີ່ ສຳ ຄັນຂອງ n x. ລຳ ດັບຂອງຂັ້ນຕອນການຄົ້ນຫາໃນຕົ້ນໄມ້ມີດັ່ງນີ້:

**Algorithm find (k, v) if v is null**

(1) ຖ້າ node v ບໍ່ມີຄ່າ, ຂໍ້ມູນ ສຳ ຄັນທີ່ຕ້ອງຄົ້ນຫາກໍ່ບໍ່ພົບ.

(2) ຖ້າຄ່າ ສຳ ຄັນ (k) ນ້ອຍກວ່າຄ່າທີ່ ສຳ ຄັນຂອງ node v, ໃຫ້ໄປທີ່ node ດ້ານຊ້າຍຂອງ v.

(3) ໃນກໍລະນີມູນຄ່າທີ່ ສຳ ຄັນ (k) ເທົ່າກັບຄ່າຫຼັກຂອງ node v

ສົ່ງຄືນ node v (ພົບ node ທີ່ຈະຖືກຄົ້ນຫາ) (4) ຖ້າຄ່າ key (k) ສູງກວ່າຄ່າທີ່ ສຳ ຄັນຂອງ node v.

ເຖິງຂໍ້ ກຳ ນົດເດັກນ້ອຍທີ່ ເໝາະ ສົມຂອງ v

**return null**

**if k < key(v)**

**return find(k,left (v))**

**else if k = key(v)**

**return v**

**else // k > key(v)**

**return find(k,right (v))**

ຈາກຮູບທີ 10.29 ສະແດງຕົວຢ່າງ. ລໍາດັບ node ຈາກ node ຮາກຫາ node ດ້ວຍຄ່າທີ່ສໍາຄັນທີ່ຕ້ອງໄດ້ຄົ້ນຫາ.

ໃນກໍລະນີທີ່ຊອກຫາຫຼັກ 50 ດ້ວຍ ຄຳ ສັ່ງ Find (50, root), k ແມ່ນ 50 ແລະ v ໃຫ້ຈຸດ node ແລະ 50 ແມ່ນໃຫຍ່ກວ່າຄີຂອງ v node (ຄ່າ 44), ສະນັ້ນມັນກໍ່ໄປທາງຂວາ of node v. ຄຳ ສັ່ງຊອກຫາ (k, ຂວາ (v)) ເພື່ອໃຫ້ v ຊີ້ເຖິງ node. 62 ຫຼັງຈາກນັ້ນ, ພົບວ່າ 50 ຕ່ ຳ ກວ່າມູນຄ່າທີ່ ສຳ ຄັນຂອງ v node (ມູນຄ່າ 62), ສະນັ້ນມັນຕ້ອງໄປທາງຊ້າຍຂອງ node v ດ້ວຍ ຄຳ ສັ່ງຊອກຫາ (k, left (v)), ເຊິ່ງພົບວ່າ node ເຊິ່ງ key ມູນຄ່າແມ່ນ 50 ກັບຄໍາສັ່ງຂອງຂໍ້ໃນຕົ້ນໄມ້ ເຂົ້າເຖິງ ສຳ ລັບການຊອກຫາຄ່າທີ່ ສຳ ຄັນ 50 ແມ່ນ 44, 62, ແລະ 50, ຕາມ ລຳ ດັບ, ດັ່ງທີ່ສະແດງໂດຍເສັ້ນລອກໃນຮູບ 10.29.

ຮູບພາບ 10.29 ຕົວຢ່າງຂອງເສັ້ນທາງ (ປະຕິບັດຕາມເສັ້ນທາງຈຸດ) ຊອກຫາຂໍ້ມູນ 50 ຂໍ້ມູນຈາກສະຖານທີ່ຕັ້ງຂອງຮາກ. ຈາກຂັ້ນຕອນໃນ ຊອກຄ່າທີ່ ສຳ ຄັນໃນຕົ້ນໄມ້ຊອກຫາ ຈຳ ນວນເວລາທີ່ຈະສົມທຽບເພື່ອຊອກຫາຄ່າທີ່ ສຳ ຄັນບໍ່ແມ່ນເກີນລະດັບຄວາມສູງຂອງຕົ້ນໄມ້ ເພາະສະນັ້ນ, ຕົ້ນໄມ້ໄບນາລີຄົ້ນຫາຈຶ່ງ ເໝາະ ສົມກັບການ ນຳ ໃຊ້ໃນຮູບແບບດັ່ງກ່າວ

**44**

**17**

**62**

**32**

**50**

**78**

**48**

**54**

**88**

ຄົ້ນຫາຂໍ້ມູນເລື້ອຍໆ

ພ້ອມດ້ວຍຊັບສິນທີ່ຈະເກັບຄ່າທີ່ ສຳ ຄັນຂອງແຕ່ລະ node ໃນຕົ້ນໄມ້ຖານສອງເພື່ອຈຸດປະສົງໃນການຄົ້ນຫາ. ຖ້າພວກເຮົາໃຊ້ວິທີການເຂົ້າເຖິງ ຂໍ້ມູນເພື່ອສະແດງຄຸນຄ່າທີ່ ສຳ ຄັນຂອງທຸກໆ node ໃນຕົ້ນໄມ້ດ້ວຍວິທີການປ່ຽນເສັ້ນທາງຕົ້ນໄມ້ຕາມ ລຳ ດັບດັ່ງຕໍ່ໄປນີ້:

**Algorithm printTree( ) {**

**if (root != NULL)**

**inOrder( root );}**

**ຈາກ ຄຳ ສັ່ງຂ້າງເທິງຈະສະແດງຄ່າຂໍ້ມູນ ສຳ ລັບແຕ່ລະ node ໃນຕົ້ນໄມ້ດ້ວຍ c. ຄໍາສັ່ງດັ່ງຕໍ່ໄປນີ້**

**Algorithm inOrder (p ) {**

**if (left(p) != NULL ) inOrder(left(p));**

**Print an element stored in node p**

**if (right(p) != NULL) inOrder(right(p));**

**ຈາກຮູບທີ 10.29, ໃນເວລາທີ່ເຂົ້າເຖິງທຸກໆ node ໃນຕົ້ນໄມ້ຕາມ ລຳ ດັບ, ຈະສະແດງມູນຄ່າຂໍ້ມູນ 17**

**32 ,44, 48, 50, 54, 62, 78 ແລະ 88 ຕາມ ລຳ ດັບ, ເຊິ່ງຈັດລຽງຕາມ ລຳ ດັບ. ເພາະສະນັ້ນ, ໂຄງສ້າງຕົ້ນໄ**

**28**

**ຖານສອງແມ່ນຖືກ ນຳ ໃຊ້ທົ່ວໄປໃນການ ນຳ ໃຊ້ຮຽກຮ້ອງໃຫ້ເຂົ້າເຖິງຂໍ້ມູນ ລຳ ດັບຕົ້ນຕໍ.**

**ນອກ ເໜືອ ຈາກຫ້ອງຄົ້ນຄ້ວາ, ສຳ ລັບວຽກງານທີ່ສຸມໃສ່ການຄົ້ນພົບຂໍ້ມູນແລະການເຂົ້າເຖິງຂໍ້ມູນເປັນ ລຳ ດັບເລື້ອຍໆ, ການເກັບຂໍ້ມູນສາມາດຖືກອອກແບບໃນຕົ້ນໄມ້ຄົ້ນຫາແບບຫຼາຍຮູບແບບ, ໂຄງສ້າງຕົ້ນໄມ້ຂໍ້ມູນທີ່ເອີ້ນວ່າ ຕົ້ນໄມ້ຄົ້ນຫາທີ່ມີຫຼາຍວິທີແມ່ນຕົ້ນໄມ້ທີ່ບັນຈຸການຄົ້ນຫາທີ່ ສຳ ຄັນເປັນ ລຳ ດັບ. ຄຸນສົມບັດຂອງຂໍ້ໃນຕົ້ນໄມ້ມີດັ່ງນີ້:**

**•ແຕ່ລະຂໍ້ທີ່ບໍ່ແມ່ນໃບມີຢ່າງ ໜ້ອຍ ສອງຂໍ້ຂອງເດັກແລະຖືເອົາຄຸນຄ່າ ສຳ ຄັນຂອງ d-1, ບ່ອນທີ່ d ແມ່ນ ຈຳ ນວນຂໍ້ຂອງເດັກ, ດັ່ງນັ້ນແຕ່ລະຂໍ້ແມ່ນປະກອບດ້ວຍຂໍ້ຂອງເດັກ v1, v2 …, vd ແລະ. k1, k2 …, kd-1 ບ່ອນໃດ**

**- ຄ່າ ສຳ ຄັນທັງ ໝົດ ໃນປະໂຫຍກທີ່ມີ v1 ເນື່ອງຈາກຮາກຂໍ້ມູນມີຄ່າ ສຳ ຄັນນ້ອຍກວ່າ k1.**

**- ຄ່າ ສຳ ຄັນທັງ ໝົດ ໃນປະໂຫຍກຍ່ອຍດ້ວຍ vi ເປັນຂໍ້ຮາກມີຄ່າ ສຳ ຄັນລະຫວ່າງ ki-1 ແລະ ki ເມື່ອ I = 2, 3, …, d-1.**

**- ຄ່າ ສຳ ຄັນທັງ ໝົດ ໃນບົດບັນຍາຍດ້ວຍ vd ເພາະວ່າຂໍ້ ກຳ ນົດຮາກມີຄ່າ ສຳ ຄັນຫຼາຍກ່ວາ kd-1.**

**•ຂໍ້ຂອງໃບບໍ່ເກັບຂໍ້ມູນຕົ້ນໄມ້ເພື່ອໃຊ້.**

**2 6 8**

**11 24**

**15**

**27 32**

**30**

**ຮູບທີ 10.30 ຕົວຢ່າງຂອງໂຄງສ້າງຂໍ້ມູນຕົ້ນໄມ້ທີ່ມີຫລາຍໆຄົ້ນຫາ.**

**node ຮາກທີ່ຖື 2 ຄ່າທີ່ ສຳ ຄັນ, ຫຼັກ ທຳ ອິດແມ່ນ 11 ແລະຄີທີສອງແມ່ນ 24 ແລະປະກອບມີ 3 ຂໍ້ຂອງເດັກດັ່ງຕໍ່ໄປນີ້:**

**•ເດັກນ້ອຍທີ່ຢູ່ເບື້ອງຊ້າຍມີຄຸນຄ່າທີ່ ສຳ ຄັນ 2, 6, ແລະ 8, ເຊິ່ງທັງ ໝົດ ແມ່ນ ໜ້ອຍ ກວ່າຄ່າທີ່ ສຳ ຄັນ ທຳ ອິດ (ຄ່າ 11).**

**•ເດັກນ້ອຍກາງ ບັນຈຸມີຄ່າທີ່ ສຳ ຄັນ 15, ເຊິ່ງແມ່ນຄ່າຫຼັກທີ່ໃຫຍ່ກວ່າມູນຄ່າຫລັກ (ມູນຄ່າ 11) ແລະ ໜ້ອຍ ກວ່າຄ່າທີ່ສອງ (ຄ່າ**

**ຂໍ້ທີ່ຖືກຕ້ອງທີ່ສຸດ ປະກອບມີຄ່າທີ່ ສຳ ຄັນ 27 ແລະ 32, ເຊິ່ງໃຫຍ່ກວ່າມູນຄ່າຫຼັກສອງ (ຄ່າ 24).**

**ທັງ ໝົດ ຂອງຂໍ້ເດັກນ້ອຍເຫຼົ່ານີ້ມີຂໍ້ເດັກນ້ອຍກາງເຊິ່ງປະກອບດ້ວຍຄ່າທີ່ ສຳ ຄັນ 30 ເຊິ່ງແມ່ນຄ່າໃນລະຫວ່າງ.**

**27 ແລະ 32 ຈາກຊັບສິນຕົ້ນໄມ້ຄົ້ນຫາຫລາຍໆບ່ອນ, ມັນພົບວ່າທຸກໆຄຸນຄ່າທີ່ ສຳ ຄັນພາຍໃນຕົ້ນໄມ້ທີ່ມີການສັ່ງຊື້ແມ່ນຖືກເຂົ້າເຖິງ.**

**ມັນແມ່ນການເຂົ້າເຖິງທີ່ ສຳ ຄັນຈັດຮຽງ. ຕົວຢ່າງ: ຖ້າທຸກໆຄ່າທີ່ ສຳ ຄັນຂອງທຸກໆ node ໃນຮູບ 10.30 ແມ່ນເຂົ້າເຖິງ, 2, 6, 8, 11, 15, 24, 27, 30 ແລະ 32 ແມ່ນໄດ້ຮັບຕາມ ລຳ ດັບເຊິ່ງເປັນຄຸນລັກສະນະການເຂົ້າເຖິງ ລຳ ດັບການຄົ້ນຫາຄ່າທີ່ ສຳ ຄັນ. ໃນຕົ້ນໄມ້ຄົ້ນຫາຫລາຍທາງໃຊ້ວິທີທີ່ຄ້າຍຄືກັບການຊອກຫາຄ່ານິຍົມ. ຢູ່ໃນຕົ້ນໄມ້ຖານສອງ ສຳ ລັບການຄົ້ນຫາ ຖ້າແຕ່ລະຂໍ້ທີ່ປະກອບດ້ວຍ node ຂອງເດັກ v1, v2, …, vd ແລະເກັບຮັກສາຄ່າ k1, k2 …, kd-1, ມີ ລຳ ດັບເພື່ອຊອກຫາຄ່າທີ່ ສຳ ຄັນ (k) ໃນຕົ້ນໄມ້ເລີ່ມຕົ້ນຈາກຮາກ. ດັ່ງຕໍ່ໄປນີ້**

**1) ຖ້າມັນບໍ່ແມ່ນໃບໃບ, ປະຕິບັດການຄົ້ນຫາມູນຄ່າທີ່ ສຳ ຄັນໂດຍການປຽບທຽບຄ່າທີ່ ສຳ ຄັນທັງ ໝົດ ໃນຂໍ້.**

**•ຖ້າຄ່າ k ສຳ ຄັນເທົ່າກັບຄ່າ ki ຫຼັກໃດ ໜຶ່ງ ໃນ node (ເມື່ອ i = 1, 2, …, d-1), ຫຼັງຈາກນັ້ນ, ຄ່າທີ່ຖືກຄົ້ນຫາກໍ່ພົບ.**

**•ຖ້າຄ່າ k ສຳ ຄັນ k ນ້ອຍກວ່າຄ່າ k1**

**ຂໍສືບຕໍ່ຊອກຫາຄຸນຄ່າທີ່ ສຳ ຄັນຢູ່ node v1 ຂອງເດັກໂດຍການກັບໄປຫາຂັ້ນຕອນທີ 1.**

**•ຖ້າຄ່າ k ສຳ ຄັນຢູ່ລະຫວ່າງຄ່າ ki-1 ແລະຄ່າ ສຳ ຄັນ ki (ki-1 <k <ki ເມື່ອຂ້ອຍ = 2, …, d-1), ຊອກຫາຂໍ້ມູນຂອງເດັກ. vi ສືບຕໍ່ໂດຍການກັບໄປຂັ້ນຕອນທີ 1.**

**•ຖ້າຄ່າ k ສຳ ຄັນ k ມີຄ່າ kd-1ໄປທີ່ node vd ເດັກຕໍ່ໄປໂດຍການກັບໄປຫາຂັ້ນຕອນທີ 1 2) ຖ້າພົບເຫັນໃບໄມ້, ມັນ**

**29**

**ໝາຍ ຄວາມວ່າກຸນແຈທີ່ທ່ານຕ້ອງການຄົ້ນຫາບໍ່ພົບ.**

**ຈາກຮູບທີ 10.31, ເພື່ອຊອກຫາຄ່າ 30, ຄ່າ 30 ທີ່ ສຳ ຄັນແມ່ນໃຫຍ່ກວ່າ 24 ຂອງຮາກ.**

**ແລະເພື່ອປຽບທຽບ node ເດັກນ້ອຍຢູ່ເບື້ອງຂວາຂອງ node ຮາກ, ເພາະວ່າ 30 ແມ່ນຢູ່ໃນລະຫວ່າງ 27 ແລະ 32, ສະນັ້ນປຽບທຽບຄ່າທີ່ ສຳ ຄັນກັບ node ເດັກຢູ່ເຄິ່ງກາງຂອງ node ນີ້ແລະຊອກຫາຄ່າ 30 ເພື່ອຊອກຫາຮູບ 10.31 ສະແດງຂໍກະແຈ -value ເສັ້ນທາງການປຽບທຽບເພື່ອຊອກຫາຄ່າ 30.**

**2 6 8**

**11 24**

**15**

**27 32**

**30**

## (ຮູບສະແດງ 10.31 ຕົວຢ່າງຂອງໂຄງສ້າງຂໍ້ມູນຕົ້ນໄມ້ທີ່ມີການຊອກຫາແບບຫຼາຍຮູບແບບ)

ຕົ້ນໄມ້ B-ຕົ້ນແມ່ນຕົ້ນໄມ້ທີ່ຄົ້ນຫາເຊິ່ງມີຂໍ້ມູນຂອງເດັກນ້ອຍຫລາຍກ່ວາສອງລູກຖ້າມັນແມ່ນຕົ້ນໄມ້ m order beat, ມັນຖືກເອີ້ນວ່າ m-way Search Tree. (ມ / 2) ຂໍ້ເດັກນ້ອຍ, ແລະໃບຂອງຕົ້ນໄມ້ທັງ ໝົດ ໃນຕົ້ນໄມ້ຕ້ອງຢູ່ໃນລະດັບດຽວກັນ, ເຊິ່ງແມ່ນລັກສະນະຕົ້ນໄມ້ຢ່າງແທ້ຈິງ, ຍົກຕົວຢ່າງ 2-3 ຕົ້ນ. Tree)

ມັນແມ່ນງໍທີ່ສາມາດຄົ້ນຫາ 2 - 3 ເສັ້ນທາງ, ແລະອື່ນໆໃນພາກນີ້, ພວກເຮົາຈະເວົ້າກ່ຽວກັບເຄື່ອງທຸບຕີທີ່ສາມາດຄົ້ນຫາເສັ້ນທາງ 2-4 ເສັ້ນ, ເອີ້ນວ່າຕົ້ນໄມ້ 2-4. (Two-four Tree)

(1) ຄຸນສົມບັດຂອງ 2-4 ຕົ້ນໄມ້

2-4 ຕົ້ນໄມ້ຫຼື 2-3-4 ຕົ້ນໄມ້ຫລື (2,4) ຕົ້ນໄມ້ແມ່ນຕົ້ນໄມ້ເຜິ້ງຊະນິດ ໜຶ່ງ ທີ່ມີຄຸນສົມບັດດັ່ງຕໍ່ໄປນີ້:

ໃນບັນດາ 2-4 ຕົ້ນໄມ້, ເສັ້ນກ່າງໃບທີ່ບໍ່ແມ່ນໃບສາມາດເກັບຮັກສາໄດ້ເຖິງ 3 ຄ່າທີ່ ສຳ ຄັນ.ນີ້ອະນຸຍາດໃຫ້ 3 ຂໍ້ຂອງຂໍ້ໃນ 2 - 4 ຕົ້ນໄມ້ທີ່ຖືກຈັດປະເພດຂຶ້ນກັບຈຳນວນຂອງມັນ.ຈຳນວນຂອງຂໍ້ຕໍ່ເດັກໂດຍການໂທ

node ແມ່ນ node 2-node, 3-node, ຫຼື 4-node ເທົ່ານັ້ນຖ້າ node ເຊື່ອມໂຍງກັບຂໍ້ຂອງເດັກນ້ອຍ ຈຳນວນໜຶ່ງ.

2 ຂໍ້, 3 ຂໍ້, ຫຼື 4 ຂໍ້, ຕາມລຳດັບຮູບ 10.40 ສະແດງຕົວຢ່າງຂອງຕົ້ນໄມ້ 2-4, ບ່ອນທີ່ຮາກຂອງຮາກມີຄຸນຄ່າທີ່ສຳຄັນ 10, 15, ແລະ 24 ເຊິ່ງເຊື່ອມໂຍງກັບ 4 ຂໍ້ຂອງເດັກ, ເຊິ່ງສະແດງວ່ານີ້ node ຮາກມີ 4 ປະເພດ.-node ຂໍ້ ກຳນົດເດັກນ້ອຍທີ່ຢູ່ເບື້ອງຊ້າຍແລະຂໍ້ເດັກທີ່ຢູ່ເບື້ອງຂວາຂອງຂໍ້ຮາກແມ່ນ 3 ຂໍ້ເພາະວ່າທັງສອງຂໍ້ສອງຂໍ້ນີ້ປະກອບມີສອງກຸນແຈແລະສາມາດເຊື່ອມໂຍງກັບ 3 ຂໍ້ຂອງເດັກແລະຂໍ້ທີ່ເດັກຢູ່ໃນສະຖານທີ່ທີສອງແລະທີສາມຂອງຂໍ້ກຳນົດຮາກຈະເປັນຂອງປະເພດ 2-node ເພາະມັນມີກຸນແຈໜຶ່ງ ແລະສາມາດເຊື່ອມໂຍງກັບ 2 ຂໍ້ຂອງເດັກ.

**2**

**8**

**10 15 24**

**12 18**

**27 32**

( ຮູບທີ 10.40 ຕົວຢ່າງຂອງໂຄງສ້າງຂໍ້ມູນຕົ້ນໄມ້ Beat)

(**2)**  ຕົ້ນໄມ້ທີ່ມີ ຈຳ ນວນສະມາຊິກ n ມີຕົ້ນໄມ້ສູງ O (log n) ເພາະວ່າ 2-4 ຕົ້ນໄມ້ມີຄວາມສູງຂອງຕົ້ນໄມ້ສູງທີ່ສຸດ. ໃນເວລາທີ່ທຸກໆຮູໃນຕົ້ນໄມ້ແມ່ນ 2-node, ຫຼັງຈາກນັ້ນຄວາມເລິກຂອງຕົ້ນໄມ້ຢ່າງ ໜ້ອຍ ກໍ່ຈະມີ 2 node ເມື່ອ i = 0, …, h-1 ແລະບໍ່ມີສະມາຊິກໃນຄວາມເລິກ h ດັ່ງທີ່ສະແດງ (10.41

0 1

1 2

h1 h2

h 0

( ຮູບທີ 10.41 ຕົວຢ່າງຂອງໂຄງສ້າງຂໍ້ມູນຕົ້ນໄມ້ Beat )

ຖ້າ h ແມ່ນລະດັບຄວາມສູງຂອງຕົ້ນໄມ້, ຫຼັງຈາກນັ້ນ n> = 1 + 2 + 4 + … + 2h-1 = 2h-1, h <= log (n + 1), ຫຼັງຈາກນັ້ນເວລາທີ່ຈະຊອກຫາຄ່າທີ່ ສຳ ຄັນໃນ 2- 4 ຕົ້ນໄມ້ທີ່ມີສະມາຊິກ n ໃຊ້ເວລາດຽວກັນກັບ O (log (n)).

(3) ເພີ່ມຄ່ານິຍົມໃນຕົ້ນໄມ້ 2-4 ຕົ້ນ.

ເພີ່ມຄຸນຄ່າ ສຳ ຄັນ ໃໝ່ ໃນຕົ້ນໄມ້ ຈະເລີ່ມຕົ້ນດ້ວຍການຄົ້ນຫາ Node ບ່ອນທີ່ທ່ານຕ້ອງການໃສ່ຄ່າ key (k) ໃໝ່, ຈາກນັ້ນຕື່ມຄ່າ ໃໝ່ ໃສ່ node ນັ້ນຮູບ 10.42 ສະແດງຕົວຢ່າງຂອງການເພີ່ມຄ່າ key (30) ໃສ່ຕົ້ນໄມ້ຖ້າ v ແມ່ນ ຕຳ ແໜ່ງ node ທີ່ຕ້ອງການ ເພື່ອໃສ່ຄ່າ ສຳ ຄັນ ໃໝ່ ດັ່ງທີ່ສະແດງໃນຮູບທີ 10.42 (ກ) ເຖິງຢ່າງໃດກໍ່ຕາມ, ຫຼັງຈາກໃສ່ປຸ່ມ ໃໝ່.

ໃນໂນດ V ຫາກໂນດ V ມີຈຳນວນຄີຫຼາຍກວ່າ 3 ສະແດງວ່າເກີດບັນຫາການລົ້ນຂອງໂນດ ດັ່ງຮູບ 10.42 (ຂ) ເຊີ່ງເຮັດວ່າໂນດ V ມີຈຳນວນຄີເທົ່າກັບ 4 ຈື່ງເປັນໂນດແບບ 5-node

ວິທີຈັດການການເກີດໂນດລົ້ນຂອງຄ່າຄີຄືການແບ່ງໂນດອອກເປັນ 2 ໂນດດັ່ງນີ້

1) ຫາກ v1…v5 ຄືໂນດຂອງລູກໂນດv ແລະມີຄີ k1…k4 ໃຫ້ແບ່ງໂນດ v ເປັນ 2 ໂນດ ເອີ້ນ v’ ແລະ v” ດັ່ງນີ້

* 1. V’ ເປັນຕົ້ນໄມ້ບີແບບ 3-ໂນດທີ່ເກັບຄ່າຄີ k1ແລະ k2 ແລະໂນດລູກ v1 v2 v3
  2. V” ເປັນຊະນິດແບບ 2-ໂນດທີ່ເກັບຄ່າຄີ k4 ແລະໂນດລູກ v4 v5

2) ຫາກໂນດ V ມີໂນດພໍ່ (node U) ໃຫ້ນຳ k3 ຂື້ນໄປເກັບທີ່ໂນດ U ກໍລະນີ V ບໍ່ມີໂນດພໍ່ (ເມື່ອVເປັນໂນດຮາກຂອງຕົ້ນໄມ້) ໃຫ້ມີການສ້າງໂນດໄໝ່ເພື່ອເກັບຄ່າ k3

3) ຫາກໂນດ U ເກີດເຫດການລົ້ນຂອງຄ່າຄີໃຫ້ຈັດການກ່ນລົ້ນດ້ວຍການແບ່ງໂນດເຊົ່ນດຽວກັນກັບໂນດ V

**24**

**15**

**1**

2

**18**

**27**

**30 32**

**35**

V

**15**

**24**

**1**

2

**18**

**27**

**32**

**35**

V

(ກ) ຕຳແໜ່ງໂນດ V ສຳລັບເພີ່ມຄີ 30 (ຂ) ຫຼັງແຊກຄີ 30 ໃນໂນດ

**32**

**24**

**15**

**35**

**1**

2

**18**

V

**27**

**30**

V’

U

(ຄ) ການແບ່ງໂນດ V ເປັນ V’ ແລະ V”

ຈາກຮູບ 10.42 (ຂ) ໂນດ V ປະກອບດ້ວຍ 4 ຄີ ຄື 27 30 32 ແລະ 35 ເມື່ອແບ່ງເປັນ 2 ໂນດ ຄື V’ ປະກອບດ້ວຍຄີ 27 ແລະ 30 ສ່ວນ V” ປະກອບດ້ວຍຄີ 35 ສ່ວນຄີ 32 ຖືກນຳໄປຈັດເກັບໃນໂນດພໍ່

ອັນກໍຣິທືມໃນການເພີ່ມຄ່າຄີໄໝ່ ມີດັ່ງນີ້:

Algorithm insertItem(k, o)

1. ຄົ້ນຫາຕຳແໜ່ງທີ່ຕ້ອງການເພີ່ມໂນດ V ໃນຕົ້ນໄມ້
2. ເພີ່ມຄ່າຄີ (k) ໄໝ່ໃນໂນດ V
3. ໃນຂະນະທີ່ໂນດ V ເກີດເຫດການການລົ້ນຂອງໂນດ (Overflow)
4. ຖ້າ V ເປັນໂນດຮາກ ໃຫ້ສ້າງໂນດໄໝ່ແລ້ວກຳນົດຮາກແທນໂນດ V
5. ແບ່ງໂນດ V ອອກເປັນ 2ໂນດດ້ວຍການໃຊ້ V <- split(V)

**(4) ການລົບຄ່າຄີໃນ 2-4 ຕົ້ນໄມ້**

ການລົບຄ່າຂໍ້ມູນໃນ 2-4 ຕົ້ນໄມ້ ຫາກເປັນໂນດທີ່ຕິດກັບໂນດໃບໃຫ້ລົບຂໍ້ມູນໃນໂນດນັ້ນຫາກບໍ່ແມ່ນໂນດທີ່ຕິດກັບໂນດໃບໃຫ້ນຳຂໍ້ມູນທີ່ເປັນ Inorder successor ມາແທນທີ່ຂໍ້ມູນທີ່ຕ້ອງການລົບ ແລ້ວຍ້າຍໄປລົບຂໍ້ມູນທີ່ຖືກນຳມາແທນທີ່ ດັ່ງຕົວຢ່າງໃນຮູບ 10.43 ເມື່ອຕ້ອງການລົບຄ່າຄີ 24 ຈະແທນຄ່າຄີ 24 ດ້ວຍຄ່າຄີ 27 (ເປັນ Inorder successor ຂອງ 24) ຂອງໂນດ V ແລ້ວໄປດຳເນີນການລົບຄ່າຄີ 27 ໃນໂນດ V ແທນ

**15**

**10**

**24**

**2**

**8**

**3**

**27**

**35**

**2**

**1**

**2**

**18**

**V**

**10**

**15 27**

**8**

**2**

**32**

**35**

**1**

**2**

**18**

**V**

(ກ) ຕຳແໜ່ງໂນດ V ທີ່ເກັບຄ່າຄີ 27 (ຂ) ຍ້າຍຄີ 27 ໄປແທນທີ່ຄີ 24 ທີ່ຖືກລົບອອກຈາກຕົ້ນໄມ້

ຫຼັງການລົບຄ່າຄີໃນຕົ້ນໄມ້ ອາດເຮັດໃຫ້ເກີດເຫດການ underflow ຄືເຮັດໃຫ້ໂນດ V ເຫຼືອໂນດລູກພຽງ 1 ໂນດ ແລະບໍ່ມີຄ່າຄີ ເຊີ່ງມີວິທີການຈັດການບັນຫາກໍລະນີໂນດ V ເກີດ underflow ເປັນ 2 ກໍລະນີ ດັ່ງນີ

**(4.1) ກໍລະນີທີ 1 ຫາກໂນດພີ່ນ້ອງຂອງ V ມີຊະນິດເປັນ 2 ໂນດ ຄືມີຄີດຽວ**

ກຳນົດໃຫ້ W ເປັນໂນດພີ່ນ້ອງຂອງ V ທີ່ມີລັກສະນະເປັນແບບ 2 ໂນດ ແລະ U ເປັນໂນດພໍ່ຂອງ V ແລະW ໃຫ້ຮ່ວມໂນດ V ເກັບໂນດພີ່ນ້ອງ W ເອີ້ນວ່າໂນດ V’ ພ້ອມທັງດືງຄ່າຄີຈາກໂນດ U ລົງມາຮ່ວມດ້ວຍ ເຊີ່ງຈະໄດ້ໂນດ V’ ທີ່ເກັບຄ່າຄີຮຽງຈາກນ້ອຍໄປຫຼາຍ ແລະມີລັກສະນະເປັນ 3-node ດັ່ງຕົວຢ່າງໃນຮູບທີ 10.44 ແລະເນື່ອງຈາກຈຳນວນຄີຂອງໂນດ U ຫຼຸດລົງ 1 ຄ່າຈື່ງອາດມີຜົນເນັດໃຫ້ໂນດ U ເກີດເຫດການ underflow ໄດ້ ຈື່ງອາດມີການວົນເຮັດຊ່ຳໆເພື່ອຈັດການໂນດ U ເຊັ່ນດຽວກັບໂນດ V

V’

U

**9**

**2**

**5**

**7**

**10**

**14**

**14**

**5**

**7**

**101**

**0**

**W**

**U**

(ກ) ຕຳແນ່ງໂນດ V ທີ່ເກີດ underflow (ຂ) ການສ້າງໂນດ V’ ທີ່ຮ່ວມຄີຈາກໂນດ W ແລະ U

ຈາກຮູບທີ 10.44 ຕົວຢ່າງການຮ່ວມໂນດເພື່ອແກ້ບັນຫາ underflow ທີ່ໂນດ V ແລະໂນດພີ່ນ້ອງ W ມີຄີດຽວ ຈື່ງຕ້ອງຮ່ວມໂນດ V ແລະ W ໂດຍການນຳຄີ 14 ຈາກໂນດ U ມາຮ່ວມກັນເປັນໂນດ V’ ທີ່ປະກອບດ້ວຍຄ່າຄີ 10 ແລະ 14

**(4.2) ກໍລະນີ 2 ຫາກໂນດພີ່ນ້ອງມີຊະນິດເປັນ 3-ໂນດ ຫຼືແບບ 4-ໂນດ**

ກຳນົດໃຫ້ W ເປັນໂນດພີ່ນ້ອງຂອງ V ທີ່ມີລັກສະນະເປັນແບບ 3 ໂນດ ຫຼື 4 ໂນດທີ່ເກັບ 2 ຫຼື 3ຄີ ຕາມລຳດັບແລະ U ເປັນໂນດພໍ່ຂອງ V ແລະ W ມີຂັ້ນຕອນໃນການເຮັດວຽກ ແກ້ບັນຫາ underflow ຂອງໂນດ V ເນັດໃຫ້ການໂອນຍ້າຍຄ່າຄີຈາກໂນດ W ແລະໂນດ U ດັ່ງນີ້:

* ຍ້າຍຄ່າຄີຈາກໂນດ U ໄປຍັງໂນດ V
* ແລະຍ້າຍຄ່າຈາກໂນດ W ໄປຍັງໂນດ U

ໃນກໍລະນີທີ 2 ຈະບໍ່ເກີດເຫດການ underflow ທີ່ໂນດໃດໆຂື້ນອີກຢ່າງແນ່ນອນ ເນື່ອງຈາກໂນດພີ່ນ້ອງມີຄ່າຄີຕັ້ງແຕ່ 2 ຄ່າຂື້ນໄປໃຫ້ໃຊ້ ແລະບໍ່ໄດ້ຫຼຸດຈຳນວນຄ່າຄີຂອງໂນດພໍ່ ຕ່າງຈາກກໍລະນີທີ 1

ຮູບທີ10.45 ເປັນຕົວຢ່າງກໍລະນີເກີດ underflow ທີ່ໂນດ V ແລະໂນດພີ່ນ້ອງ W ມີຄີຫຼາຍກວ່າ 1ຄີ ປະກອບດ້ວຍ 9 ແລະ 10 ດັ່ງຮູບທີ 10.45 ຈື່ງສາມາດແກ້ໄຂບັນຫາ underflow ຂອງໂນດ V ໄດ້ແລະຍ້າຍຄີ 10 ຂຈາກໂນດ W ໄປຍັງໂນດ U

**14**

**7**

**5**

**2**

**9**

**V**

**W**

**109**

**7**

**52**

**2**

**14**

**VV**

**W**

**14**

(ກ) ຕຳແໜ່ງໂນດ V ທີ່ເກີດ underflow (ຂ) ໂອນຍ້າຍຄີລະຫວ່າງໂນດ W U V

**(5) ປະສິດທີພາບໃນການດຳເນີນການສຳລັບ 2-4 ຕົ້ນໄມ້**

ຫາກຕົ້ນໄມ້ມີຈຳນວນໂນດເທົ່າກັບ N ໂນດ ແລະມີຄວາມສູງເທົ່າກັບ Log(N) ຈະເຫັນວ່າການດຳເນີນການໃນການເພີ່ມຄ່າຄີໃນຕົ້ນໄມ້ ຈະໃຊ້ເວລາໃນການຄົ້ນຫາຕຳແໜ່ງໂນດໃນການແຊກຄ່າຄີໄໝ່ບໍ່ເກີນຄວາມສູງຂອງຕົ້ນໄມ້ ແລະຫາກເກີດບັນຫາການລົ້ນຂອງໂນດກໍສາມາດແກ້ໄຂດ້ວຍການແບ່ງໂນດເຊີ່ງອາດເກີດການວົນຊ້ຳທີ່ໂນດພໍ່ໄດ້ ແຕ່ກໍຈະໃຊ້ເວລາບໍ່ເກີນຄວາມສູງຂອງຕົ້ນໄມ້ເຊັ່ນກັນ ດັ່ງນັ້ນການດຳເນີນການເພີ່ມຄີໄໝ່ໃນການເພີ່ມຄີໄໝ່ໃນຕົ້ນໄມ້ບີ ຈື່ງໃຊ້ເວລາບໍ່ເກີນ O(Log N)

ສຳລັບການດຳເນີນການໃນການລົບຄ່າຄີໃນຕົ້ນໄມ້ ຈະໃຊ້ເວລາໃນການຫາຕຳແໜ່ງໂນດທີ່ຕ້ອງການລົບແລະການແກ້ໄຂບັນຫາ underflow ຂອງໂນດດ້ວຍການຮ່ວມຄີຫຼືການໂອນຍ້າຍຄີຈາກໂນດພີ່ນ້ອງ ໃຊ້ເວລາບໍ່ເກີນຄວາມສູງຂອງຕົ້ນໄມ້ ຫຼື ເທົ່າກັບ O(Log N) ເຊັ່ນກັນເຖີງວ່າຈະເກີດເຫດການ underflow ຊ້ຳທີ່ໂນດພໍ່ກໍຕາມ

**}**