**ກຸ່ມ VI**

**ລາຍຊື່ສະມາຊິກ**

ທ.ໄຊປັນຍາ

ທ.ອາຫຼຽງ

ນ.ແອນນີ້

ນ.ໄພພານິດ

ທ.ແອ

ນ.ແຫວນຄຳ

ທ.ເກັ່ງເລົ່າ

ທ.ແທນ

ທ.ໂວເວັ້ນ

ທ.ວິໄລ

ບົດທີ10: ໂຄງສ້າງຂໍ້ມູນແບບ **Hash**

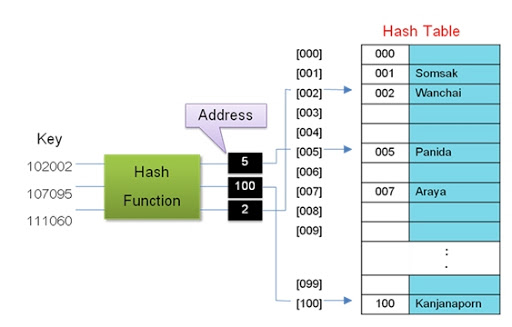
1. **ຄວາມໝາຍຂອງ Hashing**

**Hashing** ເປັນວິທີການກ່ຽວກັບການເກັບມ້ຽນຂໍ້ມູນທີ່ມີຈຳນວນຫຼາຍໄວ້ໃນຕາຕະລາງ ຊື່ງມີການກຳນົດວ່າຂໍ້ມູນແຕ່ລະຕົວຄວນຈະເກັບໄວ້ບ່ອນໃດໃນຕາຕະລາງ ແລະ ວິທີ່ການຄົ້ນຫາຂໍ້ມູນ,ຕາຕະລາງທີ່ໃຊ້ເກັບຂໍ້ມູນເອີ້ນວ່າ Hash Table ສ່ວນວິທີການໃນການຊອກຫາບ່ອນເກັບຂໍ້ມູນເອີ້ນວ່າ Hash Function ຊື່ງເປັນຕຳລາທາງຄະນິດສາດ.

1. **ໂຄງສ້າງຂໍ້ມູນແບບ Hash Table**

**Hash Table** ເປັນຕາຕະລາງທີ່ໃຊ້ໃນການເກັບຂໍ້ມູນຊຶ່ງພາຍໃນຕາຕະລາງຈະມີຊ່ອງທີ່ໃຊ້ສຳຫຼັບເກັບຂໍ້ມູນແຕ່ລະຕົວເອົາໄວ້ ໂດຍແຕ່ລະຊ່ອງເກັບຂໍ້ມູນຈະມີເລກທີ່ຢູ່ Address ກຳກັບຢູ່ ເພື່ອໃຊ້ໃນການອ້າງອີງທີ່ຢູ່ໂຄງສ້າງຂໍ້ມູນ Hash Table ສາມາດສ້າງໄດ້ຈາກ Array ຫຼື Link List.

* ການເຮັດ Hashing ເຂົ້າໃນໂຄ້ງສ້າງຂໍ້ມູນ Hash Table



* ລັກສະນະຈຸດເດັ່ນຂອງໂຄງສ້າງຂໍ້ມູນແບບ Hash Table: ແມ່ນສາມາດ ເພີ່ມ(Insert), ລຶບ(Delete) ແລະ ຄົ້ນຫາ(Find) ຂໍ້ມູນໄດ້ໂດຍໃຊ້ເວລາສະເລ່ຍຄົງທີ່(Constant Average Time) ຊຶ່ງຖືວ່າປະສິດທິພາບດີຫຼາຍເມືອປຽບທຽບກັບໂຄ້ງສ້າງຂໍ້ມູນແບບອື່ນໆ.

Array ທີ່ໃຊ້ສ້າງຕາຕະລາງ Hash Table ສາມາດເປັນປະເພດທີ່ມີຂະໜາດບໍ່ປຽນແປ່ງ(Fix Array) ໄດ້, ແຕ່ບໍ໋ສາມາດຄົ້ນຫາຂໍ້ມູນທີຕຳສຸດ(FindMin), ສູງສຸດ (FindMax) ໄດ້ ແລະ ບໍ່ສາມາດຈັດລຽງລຳດັບຂໍ້ມູນໃນຕາຕະລາງໄດ້.

1. **ຫຼັກການພື້ນຖານຂອງ Hashing**

ການເຮັດ hashing ແມ່ນການປ່ຽນຄ່າ key ຂໍ້ມູນໃຫ້ກາຍເປັນເລກທີ່ຢູ່ເພື່ອທີ່ຈະສາມາດເກັບ ຄາຂອງ key ຂໍ້ມູນນັ້ນເຂົ້າໄປໃນ hash Table ໄດ້ ແລະ ສາມາດທີ່ຈະຄົ້ນຫາ key ຂອງຂໍ້ມູນ ນັ້ນໄດ້ໃນພາຍຫຼັງ ໂດຍທີ່ເທັກນິກວິທີທີ່ເອົາມາໃຊ້ໃນການປ່ຽນຄ່າ key ດັ່ງກ່າວແມ່ນ Hash Function, ສ່ວນຜົນໄດ້ຮັບຊຶ່ງແມ່ນເລກທີ່ຢູ່ທີ່ໄດ້ຈະໃຊ້ສັນຍາລັກ H(x), ໂດຍທີ່ x ແມ່ນ key ຂອງຂໍ້ມູນ ແລະ ຫນ້າທີ່ຫລັກຂອງ Hashing ແມ່ນຄວາມພະຍາຍາມໃນການຄົ້ນຫາທີ່ຢູ່ເພື່ອເກັບມ້ຽນຂໍ້ມູນໃດຫນຶ່ງເຂົ້າໄປໃນ hash Table ເມື່ອມັນຍັງບໍ່ເຕັມ.

Key ຂໍ້ມູນ

Key: x

Hash

Function

ທີ່ຢູ່ໃນ Hash Table

Address: H(x)

Address: H(x)

**ຮູບທີ 10.1** ຫລັກການພື້ນຖານຂອງ hashing

ປະສິດທິພາບຂອງການເຮັດ hashing ແມ່ນຂື້ນຢູ່ກັບ Hash Function ເປັນຫລັກ. ລັກສະນະຂອງ Hash Function ທີ່ດີແມ່ນສາມາດກະຈາຍ key ຂອງຂໍ້ມູນທີ່ປ້ອນເຂົ້າມາໃຫ້ຢູ່ໃນ ບ່ອນທີ່ບໍ່ເຮັດໃຫ້ມີການຕໍາກັນເລີຍໃນ Flash Table ຫຼື ຖ້າຈໍາເປັນຕ້ອງຕໍາກັນ ກໍ່ຕ້ອງໃຫ້ມີການຕໍ ກັນຫນ້ອຍທີ່ສຸດເທົ່າທີ່ເປັນໄປໄດ້ ຊຶ່ງຈະຕ້ອງມີວິທີການທີ່ດີມາແກ້ໄຂບັນຫາດັ່ງກ່າວ.ດັ່ງນັ້ນສາມາສະຫຼຸບໄດ້ຫລັກການສໍາຄັນທີ່ຈໍາເປັນຈະຕ້ອງຄໍານຶງເຖິງໃນການອອກແບບເພື່ອ ທີ່ຈະເອົາມາໃຊ້ກັບໂຄງສ້າງຂໍ້ມູນ Hash Table ໃຫ້ມີປະສິດທິພາບສູງສຸດ 2 ຫລັກການດັ່ງນີ້:

1. ຕ້ອງເລືອກໃຊ້ lash function ທີ່ມີປະສິດທິພາບ ເພື່ອເຮັດໃຫ້ການເກີດບັນຫາການຕໍາກັນ

ຂອງຂໍ້ມູນຫນ້ອຍທີ່ສຸດ.

2. ຕ້ອງມີວິທີການເພື່ອມາແກ້ໄຂບັນຫາໃນກໍລະນີເກີດມີການຕໍາກັນຂື້ນ ຊຶ່ງເມື່ອເກີດມີການຕໍາກັນ

ຂື້ນແລ້ວຈະເອົາ key ຂໍ້ມູນຕົວທີ່ມານໍາຫລັງທີ່ມາຕໍາໄປເກັບໄວ້ບ່ອນໃດໃນ Hash Table.

1. **ການດໍາເນີນການກັບໂຄງສ້າງຂໍ້ມູນ Hash Table**

ການດໍາເນີນການກັບໂຄງສ້າງຂໍ້ມູນ Hash Table ມີທັງຫມົດ 3 ຢ່າງຄື:

1. ການເພີ່ມຂໍ້ມູນ (Insert) ເປັນການເອົາຂໍ້ມູນເຂົ້າໄປເກັບໄວ້ໃນ Hash Table ຊຶ່ງການເພີ່ມຂໍ້ມູນ

ມີຫລາຍວິທີ, ແຕ່ລະວິທີກໍ່ມີເທັກນິກທີ່ແຕກຕ່າງກັນ ຊຶ່ງຈະໄດ້ເວົ້າເຖິງລາຍລະອຽດໃນແຕ່ລະວິທີໃນພາຍຫລັງ.

2. ການຄົ້ນຫາຂໍ້ມູນ (Find) ເປັນການຄົ້ນຫາຂໍ້ມູນທີ່ຕ້ອງການໃນ Hash Table ວ່າມີຂໍ້ມູນທີ່

ຕ້ອງການເກັບໄວ້ໃນນັ້ນຫຼືບໍ່ ແລະ ຖ້າມີມັນເກັບໄວ້ບ່ອນໃດໃນ Hash Table ໃດຫນຶ່ງ, ວິທີການຄົ້ນຫາຂໍ້ມູນສ່ວນຫລາຍແມ່ນໃຊ້ຫລັກການດຽວກັນກັບຕອນເພີ່ມຂໍ້ມູນເຂົ້າໄປ.

3.ການລຶບຂໍ້ມູນ(Delete) ແມ່ນການລຶບຂໍ້ມູນໃດໜຶ່ງອອກຈາກ Hash table ຊຶ່ງການລຶບຂໍ້ມູນແມ່ນຈະຕ້ອງໄດ້ຄົ້ນຫາຂໍ້ມູນທີ່ຕ້ອງການລຶບນັ້ນຂື້ນມາກ່ອນ ຈາກນັ້ນຈຶ່ງລຶບຂໍ້ມູນໄປ.

ດັ່ງນັ້ນເວລາທີໃຊ້ໄປໃນການລຶບຂໍ້ມູນຈຶ່ງໄກ້ຄຽງກັບເວລາທີ່ໃຊ້ໄປໃນການຄົ້ນຫາຂໍ້ມູນ ໂດຍມີຄ່າຫຼາຍກວ່າພຽງເລັກໜ້ອຍ.

ການດຳເນີນການທັງ3 ຢ່າງນີ້ມີຂັ້ນຕອນໃນການເຮັດວຽກທີ່ແຕກຕ່າງກັນອອກໄປ ຂື້ນຢູ່ກັບການ Hashing ືແຕ່ລະວິທີ ຊຶງຈະໄດ້ອະທິບບາຍລາຍລະອຽດຂອງແຕ່ລະວິທີໃນພາຍຫຼັງ.

1. **Hash function**

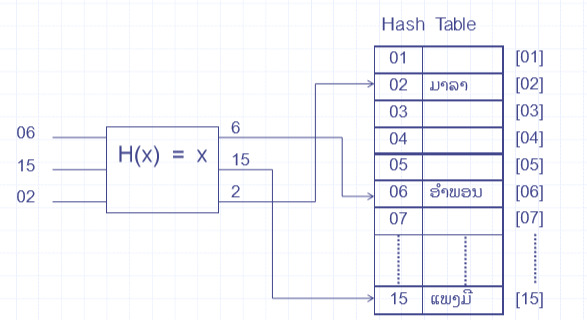
ເປັນສິ່ງທີ່ສຳຄັນໃນການເຮັດວຽກຂອງໂຄງສ້າງຂໍ້ມູນ ດັ່ງນັ້ນ, ຄວນຈະຮຽນຮູ້ເຖິງເທັກນິກພື້ນຖານຕ່າງໆຂອງ Hash function ເພື່ອສາມາດເລືອກເທັກນິກໃຫ້ເໝາະສົມໃນການເຮັດວຽກ

ຮູບແບບເທັກນິກໃນການໃຊ້ Hash function ມີຫຼາຍຮູບແບບເຊັ່ນ:

**1. ຮູບແບບ Direct Hashing**

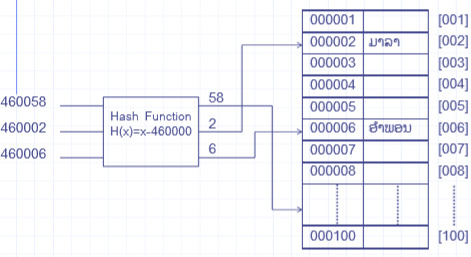
* ເປັນການກໍານົດໃຫ້ key ຂໍ້ມູນທີ່ຮັບເຂົ້າມາມີຄ່າເປັນຕຳແໜ່ງທີ່ຢູ່ ໂດຍບໍ່ຕ້ອງຜ່ານຂັ້ນຕອນວິທີໃດໆ
* ເປັນຮູບແບບທີ່ງ່າຍດາຍ ແລະ ມີປະສິດທິພາບໃນການເຮັດວຽກສູງ ເນື່ອງຈາກສາມາດຮັບປະກັນໄດ້ວ້າຈະບໍ່ເກີດການຕໍາກັນຂອງ key ຂໍ້ມູນແນ່ນອນ
* ແຕ່ຈະມີຂໍ້ຈໍາກັດໃນເລື່ອງ key ຂໍ້ມູນ ຈະຕ້ອງເປັນ key ຂໍ້ມູນທີ່ມີຂອບເຂດທີ່ຊັດເຈນ ແລະ ມີຄວາມຫ່າງຂອງ key ບໍ່ຫຼາຍເກີນໄປ. ນອກຈາກນັ້ນຄວາມແຕກຕ່າງລະຫວ່າງປະລິມານຂໍ້ມູນໃນຕາຕະລາງກັບເນື້ອທີ່ວ່າງໃນຕາຕະລາງບໍ່ຫຼາຍເກີນໄປ

ຕົວຢ່າງ: ກໍາໜົດ key ຂໍ້ມູນມີຄ່າ 01-15 ດັ່ງນັ້ນຕໍາແໜ່ງທີ່ຢູ່ຈະມີຄ່າເປັນ 01-15 ເຊັ່ນກັນ



**2. ຮູບແບບ Subtraction**

* ຄ້າຍຄືກັບ Direct Hashing ພຽງແຕ່ວ່າຮູບແບບນີ້ແມ່ນໃຊ້ສຳລັບກໍລະນີ key ຂໍ້ມູນບໍ່ໄດ້ເລີ່ມຕັ້ງຄ່າ 01 ແຕ່ເລີ່ມຕົ້ນທີ່ຄ່າອື່ນ ສະນັ້ນ ຮູບແບບນີ້ຈື່ງໄດ້ເອົາຄ່າຄົງທີ່ໃກນື່ງມາລົບອອກໄປ
* ຖືໄດ້ວ່າເປັນຮູບແບບທີ່ງ່າຍ ແລະ ມີປະສິດທິພາບສູງອີກວິມີໜື່ງເນື່ອງຈາກສາມາດຮັບປະກັນໄດ້ວ່າຈະບໍ່ມີການຕໍາກັນຂອງຂໍ້ມູນ



**3. ຮູບແບບ Modulo Division**

* ເປັນຮູບແບບທີ່ນິຍົມໃຊ້ກັນຫຼາຍ ເນື່ອງຈາກມີຄວາມຍືດຍຸ່ນສູງ
* ເປັນການຊອກຫາຕໍາແໜ່ງ Address ໂດຍການນຳຄ່າ key ມາຫານແບບ Modulo ດ້ວຍຂະໜາດຂອງຕາຕະລາງ (ການຫານແບບ Modulo ຫຼື MOD ຄືການຫານໂດຍຜົນທ່ີໄດ້ຄື ຄ່າຕົວເສດທີ່ເຫຼືອຈາກການຫານເຊັ່ນ 10 MOD 3 = 1)
* ນິຍົມກໍາໜົດໃຫ້ຂະໜາດຂອງຕາຕະລາງມີຄ່າເປັນຈໍານວນມູນ
* Hash function ຊະນິດນີ້ສາມາດສ້າງໄດ້ 2 ຮູບແບບ
* ຄ່າຂອງຕໍາແໜ່ງທີ່ຢູ່ໄດ້ມາຈາກຄ່າ key ຂໍ້ມູນ Modulo ດ້ວຍຂະໜາດຂອງຕາຕະລາງທີ່ເປັນຈຳນວນມູນ ຄ່າທີ່ໄດ້ຢູ່ລະຫວ່າງ {000 ເຖິງ (ຂະໜາດຕາຕະລາງມາ-1)}

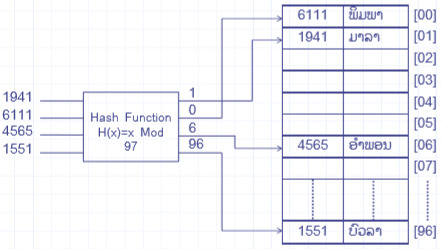
H(x) = x MOD ຂະໜາດຂອງຕາຕະລາງ

* ຄ່າຂອງຕໍາແໜ່ງທີ່ຢູ່ໄດ້ມາຈາກຄ່າ key ຂໍ້ມູນ Modulo ດ້ວຍຂະໜາດຂອງຕາຕະລາງທີ່ເປັນຈຳນວນມູນ ແລ້ວບວກເພີ່ມເຂົ້າໄປອີກ1 ຄ່າທີ່ໄດ້ຢູ່ລະຫວ່າງ {001 ເຖິງຂະໜາດຕາຕະລາງມາ}

H(x) = (x MOD ຂະໜາດຂອງຕາຕະລາງ) +1

ຕົວຢ່າງ: ການຊອກຫາຄ່າຕໍາແໜ່ງທີ່ຢູ່ໃນ Hash Table ເພື່ອຈະສ້າງ Hash Table ເກັບລາຍຊື່ພະນັກງານ ໂດຍຕ້ອງການຕາຕະລາງຂະໜາດ100 ແຕ່ເນື່ອງຈາກຂະໜາດຂອງຕາຕະລາງຄວນເປັນຈຳນວນມູນ ດັ່ງນັ້ນຈື່ງເລືອກຂະໜາດ 97

|  |  |
| --- | --- |
| Key | Address |
| 1941 | H(1941)=1941 MOD 97=1 |
| 6111 | H(6111)=6111 MOD 97=0 |
| 4565 | H(4565)=4565 MOD 97=6 |
| 1551 | H(1551)=1551 MOD 97=96 |



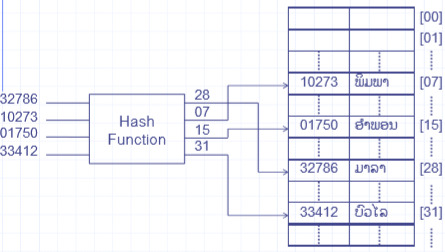
**4. ຮູບແບບ Digit Extraction**

* ເປັນການນໍາເອົາ key ຂໍ້ມູນມາພຽງບາງຕຳແໜ່ງເພື່ອໃຊ້ເປັນຕຳແໜ່ງທີ່ຢູ່ໃນ Hash Table ໂດຍທີ່ຕຳແໜ່ງຕົວເລກທີ່ເລືອກນັ້ນຈະຕ້ອງເປັນຕຳແໜ່ງທີ່ຕົວເລກຊໍ້າກັນໜ້ອຍທີ່ສຸດ

ຕົວຢ່າງ: ກໍາໜົດໃຫ້ key ມີຄ່າ{32768, 10273, 01750, 33412, 28503, 12720}

Keyທີ່ຊໍ້າກັນໜ້ອຍທີ່ສຸດຄືຕຳແໜ່ງທີ່ 2 ແລະ 4. ດັ່ງນັ້ນ, ຈື່ງເອີ້ນເອົາຕໍາແໜ່ງ 2 ແລະ 4 ມາຈັບຄູ່ກັນເພື່ອກໍາໜົດເປັນທີ່ຢູ່ໃນຕາຕະລາງ ເຊີ່ງຈັບຄູ່ອອກມາໄດ້ 28, 07,​ 15, 31, 80, 22

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ຕໍາແໜ່ງທີ່: | 1 | 2 | 3 | 4 | 5 |
| key | 3 | 2 | 7 | 8 | 6 |
|  | 1 | 0 | 2 | 7 | 3 |
| 0 | 1 | 7 | 5 | 0 |
| 3 | 3 | 4 | 1 | 2 |
| 2 | 8 | 5 | 0 | 3 |
| 1 | 2 | 7 | 2 | 0 |



**5. ຮູບແບບ Mid Square**

* ເປັນການນໍາເອົາ key ຂໍ້ມູນມາຍົກກຳລັງສອງຈາກນັ້ນເລືອກເອົາຕຳແໜ່ງກາງມາໃຊ້ເປັນຕຳແໜ່ງທີ່ຢູ່ໃນ Hash Table ເຊັ່ນ: ຖ້າຜົນຂອງການຍົກກຳລັງສອງແມ່ນ 145672 ຈະເລືອກເອົາຕົວເລກຕຳແໜ່ງທີ່ 3,4 ມາໃຊ້ໂດຍແມ່ນ 56

|  |  |  |  |
| --- | --- | --- | --- |
| key | ກໍາລັງສອງ | ຜົນໄດ້ຮັບ | ຕໍາແໜ໋ງທີ່ຢູ່3, 4,5 |
| 312 | 312^2 | 097344 | 734 |
| 124 | 124^2 | 015376 | 537 |
| 542 | 542^2 | 293764 | 376 |
| 056 | 056^2 | 003136 | 313 |

ແຕ່ວິທີ່ນີ້ແມ່ນມີຂໍ້ກຳນົດທີ່ວ່າ key ຂໍ້ມູນຕ້ອງບໍ່ມີຄ່າຫຼາຍເກີນໄປ ເພາະຖ້າຕົວເລກຫຼາຍເວລາຂື້ນກຳລັງສອງຈະເຮັດໃຫ້ເປືອງເນື້ອທີ່ໜ່ວຍຄວາມຈຳ

* ເພື່ອແກ້ໄຂຂໍ້ມູນຈໍາກັດເພີ່ນຈື່ງເພີ່ມທາງເລືອກໃຫ້ ໂດຍກໍລະນີນີ້ມີ key ຂໍ້ມູນມີຂະໜາດໃຫຍ່ ໃຫ້ເລືອກບາງຕຳແໜ່ງຂອງ key ມາຂື້ນກຳລັງສອງ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| key | ຄ່າທີ່ເລືອກ | ກຳລັງສອງ | ຜົນໄດ້ຮັບ | ຕໍາແໜ່ງທີ່ຢູ່ |
| 1356328 | 135 | 135^2 | 018225 | 822 |
| 2652355 | 265 | 265^2 | 070225 | 022 |
| 3645787 | 364 | 364^2 | 132496 | 249 |
| 3253463 | 325 | 325^2 | 105625 | 562 |

**6. ຮູບແບບ Folding Hashing**

ເປັນການກໍານົດຄ່າkey ທີ່ຮັບມາແປງຈຳນວນຂອງຕົວເລກຕາມຈຳນວນ Address ແລ້ວລວມຄ່າຄຳນວນໄໄດ້ຕົວເລກທີ່ຕ້ອງການ ໂດຍມີວິທີຫາ ​Address ຢູ່ 2 ວິທີ:

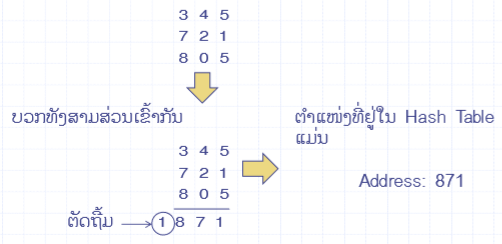
1. ວິທີ Fold Shift

* ແບ່ງ key ໃຫ້ມີຂະໜາດເທົ່າກັບຂະໜາດຂອງຕຳແໜ່ງທີ່ຢູ່ທີ່ຕ້ອງການ ເຊີ່ງປະກອບດ້ວຍ 3 ສ່ວນຄື: ຊ້າຍ, ເຄີ່ງກາງ, ຂວາ.
* ເລື່ອນ key ດ້ານຊ້າຍ ແລະ ດ້ານຂວາໃຫ້ມາເຄີ່ງກັບ key ເຄີ່ງກາງ
* ບວກ key ທັງສາມສ່ວນເຂົ້າກັນ
* ຈະໄດ້ຕໍາແໜ່ງທີ່ຢູ່ໃນ Hash Table, ແຕ່ຖ້າຜົນທີ່ໄດ້ໃຫຍ່ກວ່າຂະໜາດຂອງຕຳແໜ່ງທີ່ຢູ່ທີ່ຕ້ອງການໃຫ້ຕັດຕົວເລກທາງໜ້າສ່ວນທີ່ເກີນອອກໄປ ຕົວເລກທີ່ເຫຼືອແມ່ນຕຳແໜ່ງທີ່ຢູ່ໃນ Hash Table

ຕົວຢ່າງ: ການເຮັດວຽກຂອງ fold shift ໂດຍກໍາໜົດໃຫ້ຕົວເລກຕໍາແໜ່ງຂໍ້ມູນທີ່ຕ້ອງການມີຂະໜາດ 3ຫຼັກ, keyທີ່ຕ້ອງການແມ່ນ {345721805}



ພັບດ້ານຊ້າຍແລະຂວາໃຫ້ມາເຄີ່ງກັບຕົວເຄີ່ງກາງ



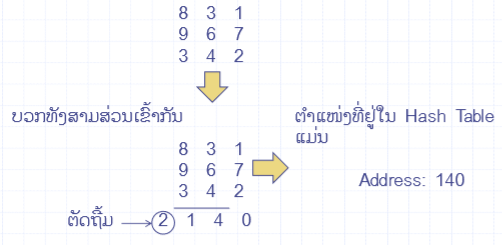
2. ວິທີ Fold Boundary

* ແບ່ງ key ໃຫ້ມີຂະໜາດເທົ່າກັບຂະໜາດຂອງຕຳແໜ່ງທີ່ຢູ່ທີ່ຕ້ອງການ ເຊີ່ງປະກອບດ້ວຍ 3 ສ່ວນຄື: ຊ້າຍ, ເຄີ່ງກາງ, ຂວາ
* ພັບ key ດ້ານຊ້າຍ ແລະ ດ້ານຂວາໃຫ້ມາທັບກັບ key ເຄີ່ງກາງ
* ບວກ key ທັງສາມສ່ວນເຂົ້າກັນ
* ຈະໄດ້ຕໍາແໜ່ງທີ່ຢູ່ໃນ Hash Table, ແຕ່ຖ້າຜົນທີ່ໄດ້ໃຫຍ່ກວ່າຂະໜາດຂອງຕຳແໜ໋ງທີ່ຢູ່ທີ່ຕ້ອງການໃຫ້ຕັດຕົວເລກທາງໜ້າສ່ວນທີ່ເກີນອອກໄປ ຕົວເລກທີ່ເຫຼືອແມ່ນຕຳແໜ່ງທີ່ຢູ່ໃນ Hash Table

ຕົວຢ່າງ: ກໍາໜົດໃຫ້ຕົວເລກຕໍາແໜ່ງຂໍ້ມູນທີ່ຕ້ອງການມີຂະໜາດ 3ຫຼັກ ,keyທີ່ຕ້ອງການແມ່ນ {138967243}



ພັບດ້ານຊ້າຍແລະຂວາໃຫ້ມາເຄີ່ງກັບຕົວເຄີ່ງກາງ



1. ການແກ້ປັນຫາການຕຳກັນຂອງຂໍ້ມູນ

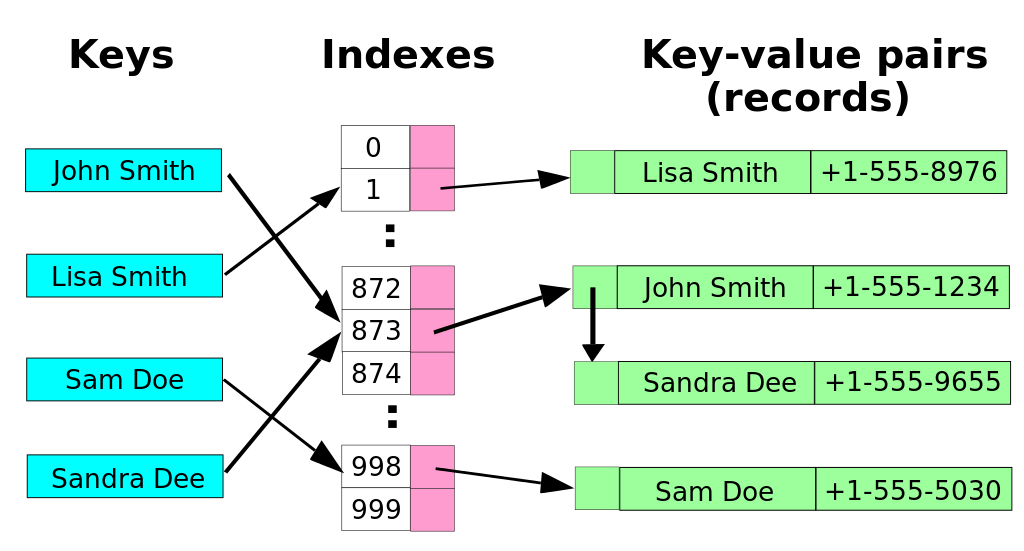
ການຕຳກັນ ເກີດຈາກການທີ່ສະມາຊິກຫຼາຍກວ່າ 2 ຕົວມີຜົນຂອງ ຟັງຊັນ Hash ຄືກັນເຮັດໃຫ້ເກີດການເກັບຂໍ້ມູນໄວ້ບ່ອນດຽວກັນ ຫຼື ເມື່ອຄຳນວນຜົນຈາກ ຟັງຊັນ Hash ແລ້ວ ອາດຈະມີຄ່າຫຼາຍກວ່າຈະໜາດຂອງ Hash table

ວຶທີແກ້ປັນຫາມີ 2 ວິທີຄື:

1, Separate Chaining

Separate Chaining ເປັນການໃຫ້ລາຍການ ຫຼື ແຖວລຳດັບຂະຫຍາຍຂະໜາດໄດ້ແທນການເກັບສະມາຊິກໂດຍກົງ ຕາຕະລາງ Hash ແຕ່ລະຊ່ອງກໍຈະເກັບຂໍ້ມູນໃນລັກສະນະເປັນລາຍການ ເມື່ອຕົວໃດທີ່ຕ້ອງການເກັບໃນຕາຕະລາງ Hash ຕຳແໜ່ງດຽວກັນ ກໍຈະຖືກເກັບໄວ້ໃນລາຍການນີ້ຕໍ່ໄປເລື້ອຍໆ

ຕົວຢ່າງ:



2, Open Addressing

ເມື່ອເກີດການຕຳກັນ ວິທີການຂອງ Open Addressing ຄືການຫາຊ່ອງໃນຕາຕະລາງ Hash

ໃໝ່ ໂດຍການຍ້າຍຈາກທີ່ເດີມ ຈົນກວ່າຈະຫາຊ່ອງວ່າງເຈິ ແລ້ວໃສ່ຄ່າລົງໄປ

ຕົວຢ່າງ:



2.1 Linear Probing

Linear Probing ຈະເຮັດໃຫ້ການຍ້າຍຈາກຈຸດເດີມເປັນໄລຍະຄົງທີ່ ການຍ້າຍແບບນີ້ຈະເຮັດໃຫ້ຂໍ້ມູນຢູ່ຕິດກັນເປັນກຸ່ມຂະໜາດໃຫຍ່ແຕ່ມີຈຳນວນກຸ່ມນ້ອຍໆ ເຮັດໃຫ້ຜົນທີ່ເຂົ້າມາຖ້າມີການຕຳກັນຢູ່ບໍລິເວນນີ້ ຈະຕ້ອງເສຍເວລາຍ້າຍໄປເພື່ອໃຫ້ພົ້ນຈາກຊ່ວງນີ້ ແລະເຮັດໃຫ້ກຸ່ມນີ້ໃຫຍ່ຂຶ້ນໄປອີກເລື້ອຍໆ.

2.2 Quadratic Probing

Quadratic Probing ມັກຈະຄຳນວນວ່າຈະເຮັດໃຫ້ການຍ້າຍໄປຈາກຈຸດເດີມເປັນໄລຍະໜຶ່ງເຊິ່ງເປັນຟັງຊັນ ຕັກກະ (ມັກເປັນຟັງຊັນກຳລັງສອງ) ການຍ້າຍແບບນີ້ຈະເຮັດໃຫ້ເກີດເປັນກຸ່ມຕິດກັນຂະໜາດນ້ອນເປັນຈຳນວນຫຼາຍກຸ່ມ ເພາະການຍ້າຍເປັນການຍ້າຍຂ້າມໄປບໍ່ຕິດກັນ ແຕ່ເຖິງຢ່າງໃດກໍຕາມຖ້າ Hash ຕຳກັນກໍຈະໄປເຈິກຸ່ມນ້ອຍໆກຸ່ມເກົ່າຢູ່ ແລະອະບໍ່ພົບຊ່ອງວ່າງບາງຊ່ອງໄດ້.

2.3 Double Hashing

Double Hashing ຈະໃຊ້ຟັງຊັນການຍ້າຍເປັນຟັງຊັນ Hash ອີກຟັງຊັນໜຶ່ງ(ມັກເອົາຟັງຊັນເກົ່າມາຊ່ວຍຄຳນວນ) ເຮັດໃຫ້ການຍ້າຍຈະຂ້ອນຂ້າງສະໝ່ຳສະເໝີ ແລະ ບໍ່ເກາະກັນເປັນກຸ່ມ.

1. Extendible hashing ແລະ ການ Rehash.

-Extendible hashing

ເໝາະສຳລັບກໍລະນີທີ່ມີຂໍ້ມູນຈຳນວນຫຼາຍໆ ຈົນເກັບໃນໜ່ວຍຄວາມຈຳຫຼັກບໍ່ໝົດ

ເປັນການຂະຫຍາຍຂະໜາດ Hash Table ຈຳນວນໜຶ່ງໄປໄວ້ໃນ Hard Disk ໂດຍໃຊ້ຫຼັກການ Binary ມາຊ່ວຍ

Extendible hashing ໝາຍເຖິງ ແນວຄິດທີ່ຊ່ວຍແບ່ງເບົາພາລະຂອງການ rehashing ໃນສ່ວນໜຶ່ງ ແຕ່ເຫດຜົນຫລັກຂອງ extendible hashing ຄືຂະໜາດຂອງຕາຕະລາງເມື່ອນຳມາປຽບທຽບເຂົ້າກັບໜ່ວຍຄວາມຈຳທີ່ມີ ຖ້າຫາກຂະໜາດຂອງຕາຕະລາງໃຫຍ່ເກີນກວ່າທີ່ຈະບັນຈຸໃນໜ່ວຍຄວາມຈຳປະຖົມພູມ ຫຼືເກີດການຕຳກັນຫຼາຍໆຄັ້ງ ເຮັດໃຫ້ ຕ້ອງaccess disk ຫຼາຍຄັ້ງກວ່າຈະໄດ້ຂໍ້ມູນທີ່ຕ້ອງການ ເຊິ່ງຕ້ອງໃຊ້ເວລາດົນ

ເຖິງແມ່ນວ່າການອະນຸຍາດໃຫ້ຕຳກັນເຮັດໃຫ້ໃຊ້ຕາຕະລາງຂະໜາດນ້ອຍໄດ້ ແຕ່ການເຮັດໃຫ້ເກີດການຕຳກັນຈົນລາຍການຍາວເກີນໄປຫຼືໜາແນ້ນເກີນໄປ ຈະເຮັດໃຫ້ເສຍເວລາໃນການຄົ້ນຫາລາຍການຫຼາຍກວ່າ ຈຶງເຮັດໃຫ້ຜິດຈຸດປະສົງຄວາມເປັນຕາຕະລາງ Hash ທີ່ໄດ້ການເຂົ້າເຖິງຂໍ້ມູນໄດ້ຢ່າງວ່ອງໄວ.

 ເຮົາຈຶ່ງນິຍາມຄ່າສັດສ່ວນບົນຈຸ(load factor:  {\displaystyle \lambda }) ໃຫ້ມີຄ່າເທົ່າກັບຈຳນວນມູນ (N) ຫານດ້ວຍຂະໜາດຂອງຕາຕະລາງ(tablesize)



ເຊິ່ງໃນວິທີ Separate Chaining ຈະສາມາດກຳນົດໄດ້ວ່າເປັນຄວາມຍາວສະເລ່ຍຂອງລາຍການໃນແຕ່ລະຊ່ອງ ສຳລັບວິທີ Open Addressing ນັ້ນຕ້ອງມີຄ່າສັດສ່ວນບົນຈຸນ້ອຍກວ່າໜຶ່ງຢູ່ສະເໝີ ໃນທາງຕົວຈິງສຳລັບ Open Addressing ເຮົາມັກຈະໃຫ້ຄ່າສັດສ່ວນບັນຈຸນ້ອຍກວ່າ 0.5 ເພື່ອກັນການຕຳກັນ.

-Rehash

ເປັນການຂະຫຍາຍຂະໜາດຂອງ Hash Table

ຍ້າຍຂໍ້ມູນຈາກ Hash Table ເດີມໄປ Hash Table ໃໝ່ທັງໝົດ

ເມື່ອມີຂໍ້ມູນຫຼາຍຂຶ້ນ ຈຳນວນຂອງຕາຕະລາງສ່ຳເກົ່າ ຄ່າສັດສ່ວນບັນຈຸກໍຫຼາຍຂຶ້ນເລື້ອຍໆ ວິທີການແກ້ກໍຄືຈະສ້າງຕາຕະລາງ Hash ໃຫ້ໃຫຍ່ກວ່າເກົ່າ ແລະຍ້າຍຂໍ້ມູນທັງໝົດໄປ Hash ໃໝ່ ເຊິ່ງເຮົາມັກຈະເອີ້ນກັນວ່າການ Rehash

ບົດທີ11: ໂຄງສ້າງຂໍ້ມູນແບບ **Graph**

1. ຄວາມໝາຍແລະປະເພດ,ໂຄງສ້າງ

Graph ເປັນຂໍ້ມູນແບບບໍ່ແມ່ນເຊີງເສັ້ນອີກຊະນິດໝື່ງກາຟເປັນໂຄງສ້າງຂໍ້ມູນທີນຳໄປໃຊ້ໃນງານທີກ່ຽວຂ້ອງກັບການແກ້ບັນຫາທີຂ້ອນຂ້າງສັບຊ້ອນກາວາງລະບົບແມ່ຂ່າຍຄອມພີວເຕີ, ການວິເຄາະຫາເສັ້ນທາງວັດຖຸ ແລະ ບັນຫາເສັ້ນທາງທີສັ້ນທີສຸດເປັນຕົ້ນ

ນິຍາມ Graph:

ເປັນໂຄງສ້າງຂໍ້ມູນແບບບໍ່ແມ່ນເຊີງເສັ້ນທີປະກອບດ້ວຍກຸ່ມຂອງສີ່ງຂອງສອງສີ່ງຄື:

1. ໂນດ(Nodes)ຫຼືເວີກເທັກ(Vertexes)

2. ເສັ້ນເຊື່ອມລະຫວ່າງໂນດເອີ້ນເອັດ(Edges)

ປະເພດຂອງ Graph ມີສອງປະເພດຄື:

+ Directed Graph: ແມ່ນ Graph ທີ່ເສັ້ນເຊື່ອມຕໍ່ມີລູກສອນກຳກັບທິດທາງ

+ Undirected Graph: ແມ່ນ Graph ທີ່ເສັ້ນເຊື່ອມຕໍ່ບໍ່ມີລູກສອນກຳກັບທິດທາງ

1. **Graph traversal**

Graph traversal ຫຼື ການແວະຜ່ານກຣາຟ ແມ່ນຂະບວນການໃນການເຂົ້າໄປໃນ Graph ເພື່ອເຂົ້າເຖິງທຸກໆ Vertex ທີ່ຢູ່ພາຍໃນ Graph ເພື່ອສຳຫຼວດ ຫຼື ເຂົ້າເຖິງທຸກ Vertex ແລະ ທຸກເສັ້ນເຊື່ອມໃນ Graph.

Graph traversal ແບ່ງອອກເປັນ 2 ວິທີຄື :

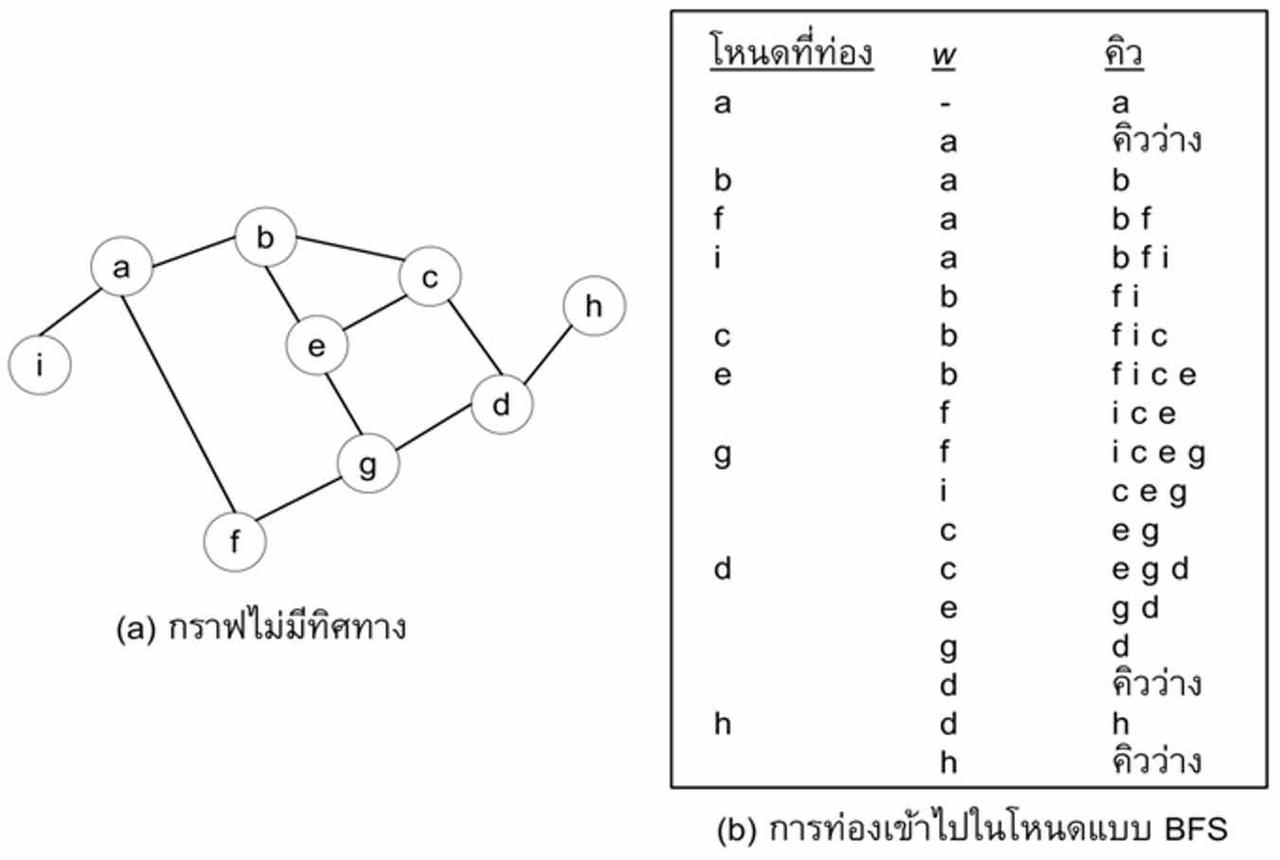
- Breadth-First Search (BFS)

- Depth-First Search (DFS)

1) Breadth-First Search (BFS)

Breadth-First Search ຫຼື ການຄົ້ນຫາແບບແນວກວ້າງ ເປັນເທັກນິກໃນການແວະຜ່ານກຣາຟແບບເຂົ້າເຖິງ Vertex ໜຶ່ງໆ ຈະໄປຫາແຕ່ລະ Vertex ທີ່ຢູ່ໃກ້ຄຽງທັງໝົດຂອງ Vertex ນັ້ນກ່ອນ ຈຶ່ງໄປຍັງ Vertex ລູກຂອງ Vertex ນັ້ນຕໍ່ໄປ ເຊິ່ງຈະໃຊ້ໂຄງສ້າງຂໍ້ມູນແບບ Queue ຊ່ວຍໃນການຈັດເກັບຂໍ້ມູນລະຫວ່າງການແວະຜ່ານກຣາຟ.

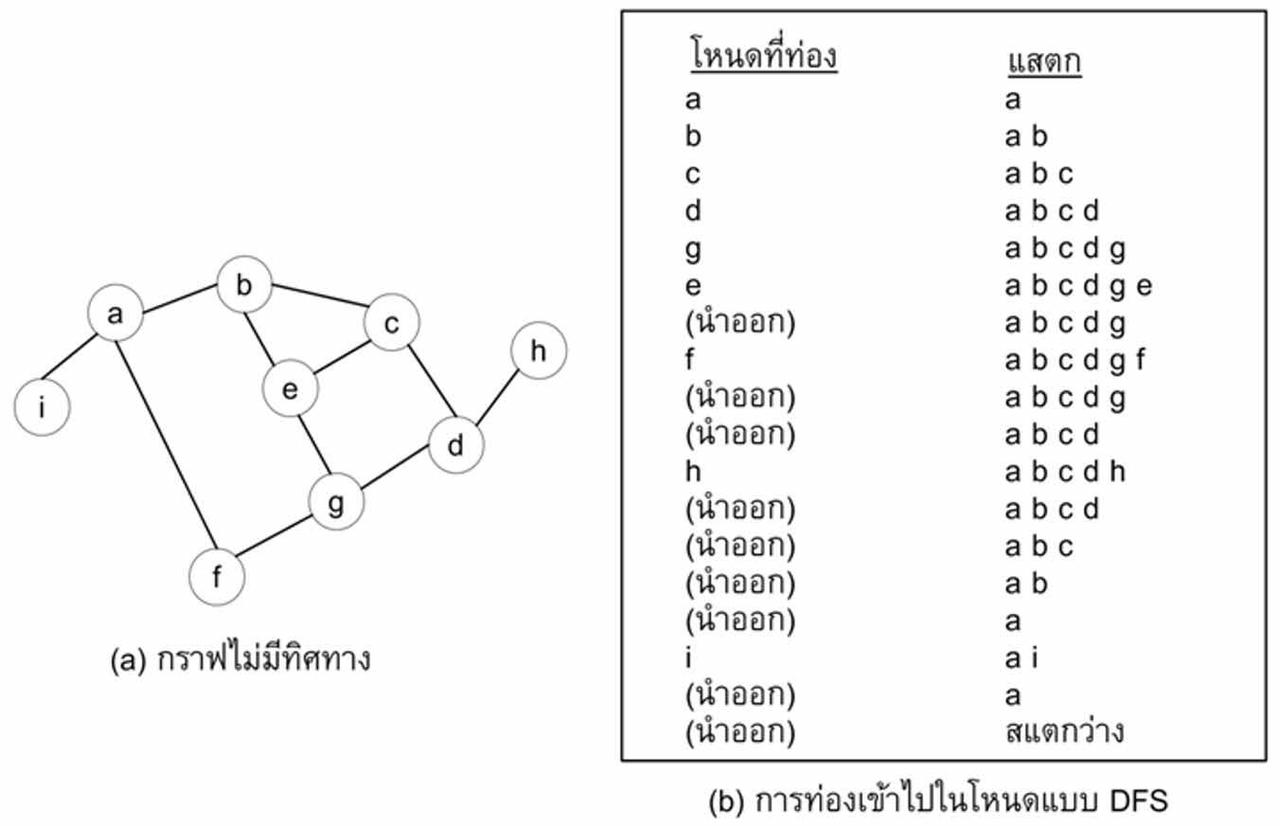
ຕົວຢ່າງ:



2) Depth-First Search (DFS)

Depth-First Search ຫຼື ການຄົ້ນຫາແບບແນວເລິກ ເປັນການເຂົ້າເຖິງ ຫຼື ຄົ້ນຫາຂໍ້ມູນໃນກຣາຟຕາມຄວາມເລິກຂອງເສັ້ນທີ່ເລີ່ມຈາກ Vertex ໜຶ່ງໃນກຣາຟ ໄປຍັງ Vertex ອື່ນໆ ຈົນກວ່າບ່ສາມາດໄປເສັ້ນທາງນັ້ນໄດ້ອີກ ຈຶ່ງເຂົ້າເຖິງ Vertex ໃນກຣາຟເສັ້ນທາງອື່ນຈົນກວ່າຈະເຂົ້າເຖິງທຸກ Vertex ເຊິ່ງຈະໃຊ້ໂຄງສ້າງຂໍ້ມູນແບບ Stack ຊ່ວຍໃນການຈັດເກັບຂໍ້ມູນລະຫວ່າງການແວະຜ່ານກຣາຟ.

ຕົວຢ່າງ:

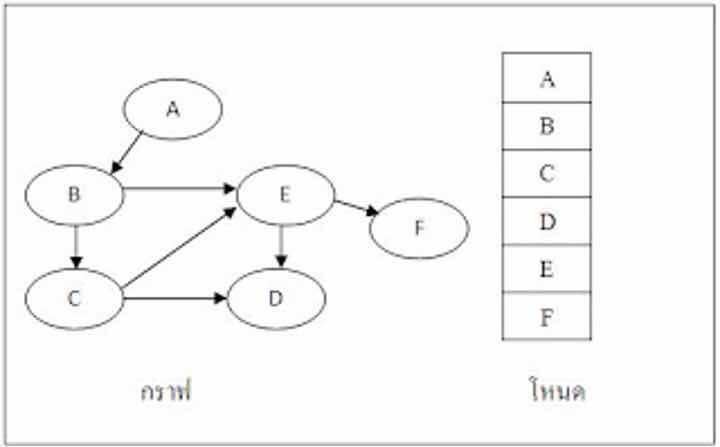


1. **ການຈັດການໂຄງສ້າງຂອງ Graph**

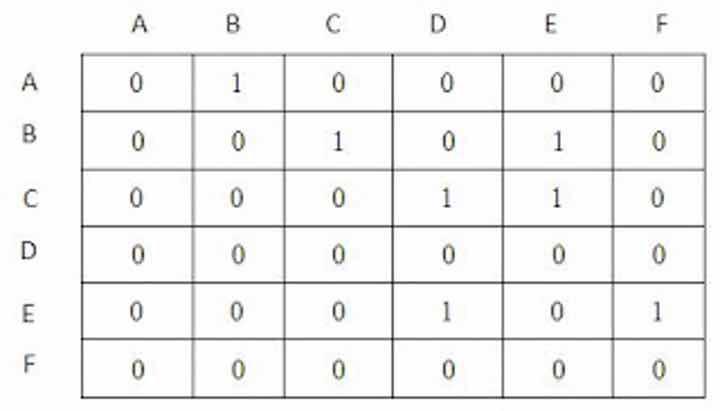
1) ການຈັດການໂຄງສ້າງກຣາຟດ້ວຍ Adjacency Matrix

ໂຄງສ້າງຂອງກຣາຟເປັນໂຄງສ້າງທີ່ປະກອບດ້ວຍໂນດ ແລະ ເສັ້ນເຊື່ອມຕໍ່ທີ່ບອກເຖິງເສັ້ນທ່ງຂອງການເດີນທາງ ຫຼື ຄວາມສຳພັນໃນທິດທາງເຊິ່ງສາມາດນຳມາແທນຄວາມສຳພັນນັ້ນດ້ວຍ Matrix ໄດ້ດ້ວຍການກຳນົດ Matrix n x n

ຕົວຢ່າງ 1: ກຣາຟແບບລະບຸທິດທາງ (Direct Graph)

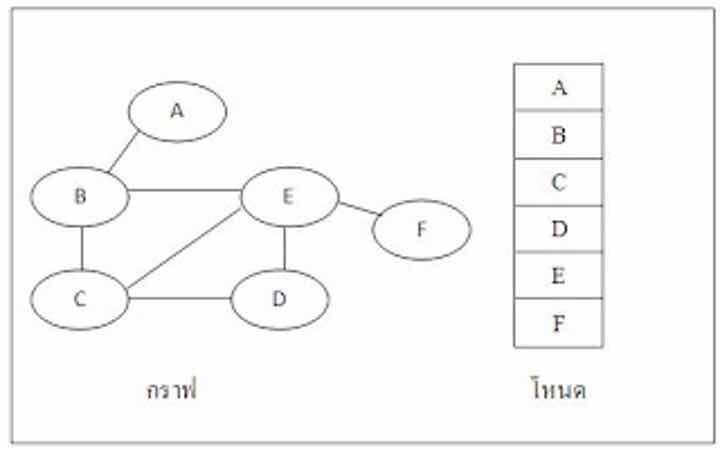


ຈາກຮູບພາບຫາກນຳມາແທນທີ່ດ້ວຍ Matrix ຈະແທນທີ່ໄດ້ດ້ວຍຈຳນວນ n x n ເຊິ່ງ n ກໍ່ຄືຈຳນວນຂອງໂນດ ຈາກຮູບຈະໄດ້ n = 6 ໂນດເຮັດໃຫ້ໄດ້ຄ່າ Matrix ຈຳນວນ 6 x 6 = 36 ຊ່ອງ ແລະ ຫາກໂນດໃດມີເສັ້ນເຊື່ອມຕໍ່ຫາກັນແມ່ນໃຫ້ລະບຸເປັນເລກ 1, ແຕ່ຖ້າໂນດໃດບ່ໄດ້ມີເສັ້ນເຊື່ອມຫາແມ່ນໃຫ້ລະບຸເປັນເລກ 0 ດັ່ງຮູບ

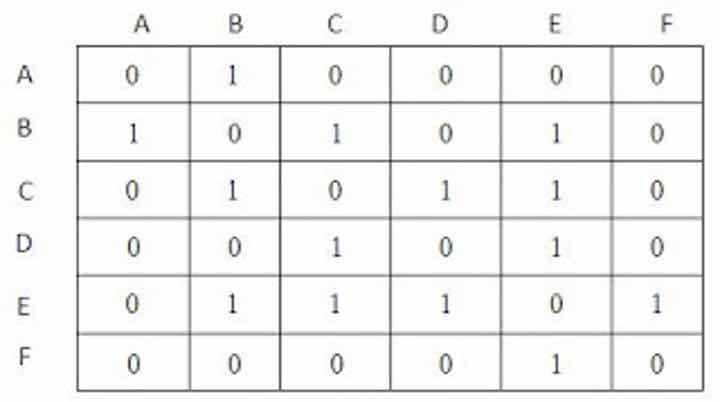


ໂນດທາງແນວນອນຈະເປັນໂນດຕົ້ນທາງ ແລະ ໂນດທາງແນວຕັ້ງຈະເປັນໂນດປາຍທາງ ຄືໂນດທາງແນວຕັ້ງຈະເປັນໂນດທີ່ຫົວລູກສອນຊີ້ໄປຫາ

ຕົວຢ່າງ 2: ກຣາຟແບບບໍ່ລະບຸທິດທາງ (Undirected Graph)



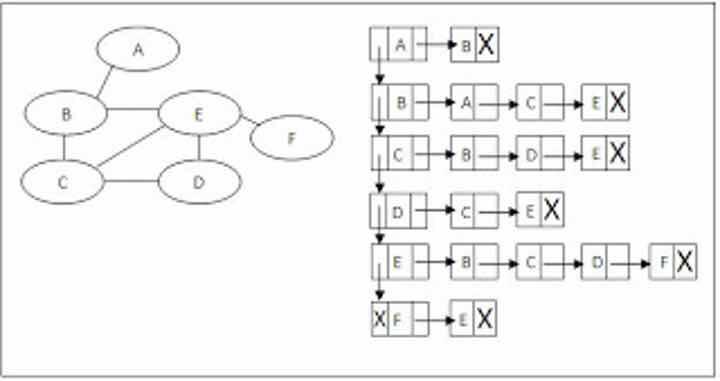
ຈາກຮູບຫາກແທນທີດ້່ວຍ Matrix ສາມາດກຳນົດຈຳນວນຊ່ອງແບບດຽວກັນກັບກຣາຟແບບມີທິດທາງ ແລະ ລະບຸຕົວເລກສຳລັບການເຊື່ອມຕໍ່ຄື 1 ແລະ 0 ເຊິ່ງຈາກຮູບສາມາດລະບຸໄດ້ດັ່ງນີ້



ກຣາຟແບບບໍ່ລະບຸທິດທາງ ຈະລະບຸເລກແຕກຕ່າງຈາກ ກຣາຟແບບລະບຸທິດທາງ ຕົວຢ່າງຄື ໂນດ A ເຊື່ອມຕໍ່ກັບໂນດ B ເນື່ອງຈາກບໍ່ມີລູກສອນລະບຸໂນດຕົ້ນ ແລະ ປາຍ ຈື່ງຕ້ອງລະບຸຄ່າເປັນ 1 ທັງ AB ແລະ BA

2) ການຈັດການໂຄງສ້າງກຣາຟດ້ວຍ Adjacency Lists

ແມ່ນການໃຊ້ List ໃນການຈັດເກັບໂນດ ແລະ Linked List 2 ມິຕິ ໃນການຈັດເກັບການເຊື່ອມຕໍ່



List ທີ່ໃຊ້ໃນການເກັບໂນດສະແດງໃນຮູບແບບຂອງ List ທິດທາງດຽວ ແຕ່ຫາກເປັນການປະຍຸກຈະກຳນົດການໃຊ້ແບບ Linked List ແບບສອງທິດທາງ ຫຼື Linked List ແບບວົງມົນ ໂດຍກຳນົດຊ່ອງທາງຊ້າຍຂອງ List ຊີ້ໄປຍັງໂນດຕໍ່ໄປ ສ່ວນຊ່ອງທາງຂວາເປັນຕົວຊີ້ບອກເຖິງເສັ້ນເຊື່ອມຕໍ່ໄປຍັງໂນດຕ່າງໆ.

1. **ການປະຍຸກໃຊ້ງານ Graph**

ໂຄງສ້າງຂໍ້ມູນ Graph ຖືກນຳໄປໃຊ້ໃນການແກ້ໄຂບັນຫາຕ່າງໆ ໂດຍສະເພາະແມ່ນບັນຫາໃນການຫາເສັ້ງທາງ ເຊັ່ນການຫາເສັ້ນທາງທີ່ສັ້ນທີ່ສຸດ ຕົວຢ່າງແມ່ນ Google map

1. **Algorithm Graph ຫາເສັ້ນທາງທີ່ສັ້ນທີ່ສຸດ**

Dijkstra’s Algorithm ຖືກຄິດຄົ້ນໂດຍນັກຄະນິດສາດຊາວ Dutch ໃນປີ 1959 ເຊິ່ງເປັນວິທີການໃນການຫາເສັ້ນທາງທີ່ສັ້ນທີ່ສຸດລະຫວ່າງ 2 ເມັດໃດໜຶ່ງ ພາຍໃນ Graph, ເງື່ອນໄຂຂອງການໃຊ້ວິທີການນີ້ແມ່ນຄ່ານ້ຳໜັກຂອງ edge ຈະຕ້ອງບໍ່ເປັນຄ່າຕິດລົບ ເຊິ່ງ Graph ຈະເປັນແບບມີທິດທາງ ​ຫຼື ບໍ່ມີກໍໄດ້ ແຕ່ຕ້ອງເປັນ Graph ທີ່ຕໍ່ເຖີງກັນ.

ຂັ້ນຕອນວິທີມີດັ່ງນີ້:

1. ກຳນົດ set v ປະກອບໄປດ້ວຍເມັດຕ່າງໆໃນ Graph
2. ໃຫ້ທຸກເມັດທີ່ບໍ່ແມ່ນເມັດເລີ່ມຕົ້ນມີນ້ຳໜັກ ຫຼື ໄລຍະທາງເປັນອະສົງໄຂ
3. ເລືອກເມັດທີ່ໃກ້ກັບເມັດເລີ່ມຕົ້ນທີ່ສຸດ ຕົວຢ່າງ ເມັດ X
4. ລົບເມັດ x ອອກຈາກ set v
5. ທີ່ເມັດ x ຂຽນໄລຍະທາງຈາກເມັດ x ຫາເມັດເລີິ່ມຕົ້ນ
6. ຫາເມັດທີ່ໃກ້ກັບເມັດ x ຕົວຢ່າງ ເມັດ Y,Z ຂຽນໄລຍະທາງຈາກເມັດເລີ່ມຕົ້ນຫາເມັດ Y,Z
7. ກັບໄປເຮັດຂໍ້ 3 ຊ້ຳຈົນຮອດເມັດທີ່ກຳນົດ
8. Graph ສົມບູນ

ແມ່ນ Graph ທີ່ທຸກເມັດເຊື່ອມຕໍ່ກັນທັງໝົດເຊິ່ງ Graph ສົມບູນມີ 2 ແບບຄື: ແບບມີທິດທາງ ແລະ ບໍ່ມີທິດທາງ.

ສູດຄຳນວນຫາຈຳນວນ edge

* Graph ສົມບູນແບບມີທິດທາງ:

ຈຳນວນ edge = N \* (N -1)

* Graph ສົມບູນແບບບໍ່ມີທິດທາງ:

ຈຳນວນ edge =