

# ບົດທີ 2

ການແກ້ປັນຫາດ້ວຍເທິງກົມືກ  
ປັນຍາປະດິດ

# បំយាការណីខេរ

- ▶ ບັນຫາ ມີ ຢູ່ ວ່າ ໂນ ບີ ຕະ ທຶກ ແມ່ ໃຊ້ ໃຫ້ ດີ ຊື່ ເຄືອງໃນ  
ຕະ ຫລາດ ຂະນະ ນັ້ນ ມີ ເງິນ 100 ກີບ ແລະ ມີ ພາ ຂະນະ ບັນ  
ຈຸ ເຄືອງຂະໜາດ 3 ກີ ໂລ ກຽມ ໂດຍ ທີ່ ຕະ ຫລາດ ມີ ສິນ ຄ້າ  
ດັງ ນີ້

- ໝາກຮູງ 1 ຫ່ວຍ, ຫ່ວຍລະ 20 ກີບ ນໍ້າ ຫັນກ 2 ກີ ໂລ
  - ສິມໂອ 7 ຫ່ວຍ, ກີ ໂລ ລະ 35 ກີບ ນໍ້າ ຫັນກ 1 ກີ ໂລ
  - ໝາກນາວ 10 ຫ່ວຍ ກີ ໂລ ລະ 25 ກີບ ນໍ້າ ຫັນກ 1 ກີ ໂລ
  - ຊື້ນໜຸ ກີ ໂລ ແຜ້ກ ລະ 40 ກີບ ນໍ້າ ຫັນກ 0.5

ເນື້ອງ ຈາກ ແມ່ ເປັນ ຄົນ ມີ ເງິນ ຈຶ່ງ ລະ ບຸ ວ່າ ຕ້ອງ ຫີ້  
ເຄື່ອງໃຫ້ ເທລືອ ເງິນ ນ້ອຍ ທີ່ສຸດ ແລະ ຕ້ອງ ບັນ ຈຸ ເຄື່ອງໃຫ້  
ມີ ນຳ ບັນກ ບາລ ຖື່ອສຸດ ດ້ວຍ ນັກ ສຶກ ສາ ຄິດວ່າ ໂນ ບີ ຕະ

## ប័ណ្ណភាពីជាជួយនៃ Domain Formal Task គឺមិនមែនសម្រាប់បង្កើត

- ▶ ប័ណ្ណភាព មិនមែនតាមរយៈតុលាឌ ទៅលើកដែលត្រូវបានផ្តល់ទៅអ្នកប្រើប្រាស់ ដោយចំណាំថា តុលាឌនេះ ត្រូវបានផ្តល់ទៅអ្នកប្រើប្រាស់។
- ▶ ប័ណ្ណភាព មិនមែនតុលាឌ ទៅលើកដែលត្រូវបានផ្តល់ទៅអ្នកប្រើប្រាស់ ដោយចំណាំថា តុលាឌនេះ ត្រូវបានផ្តល់ទៅអ្នកប្រើប្រាស់។
- ▶ ប័ណ្ណភាព មិនមែនតុលាឌ ទៅលើកដែលត្រូវបានផ្តល់ទៅអ្នកប្រើប្រាស់ ដោយចំណាំថា តុលាឌនេះ ត្រូវបានផ្តល់ទៅអ្នកប្រើប្រាស់។
- ▶ ប័ណ្ណភាព មិនមែនតុលាឌ ទៅលើកដែលត្រូវបានផ្តល់ទៅអ្នកប្រើប្រាស់ ដោយចំណាំថា តុលាឌនេះ ត្រូវបានផ្តល់ទៅអ្នកប្រើប្រាស់។

# វិທីការណ៍រៀបចំបែងទាំងអស់ (State space search)

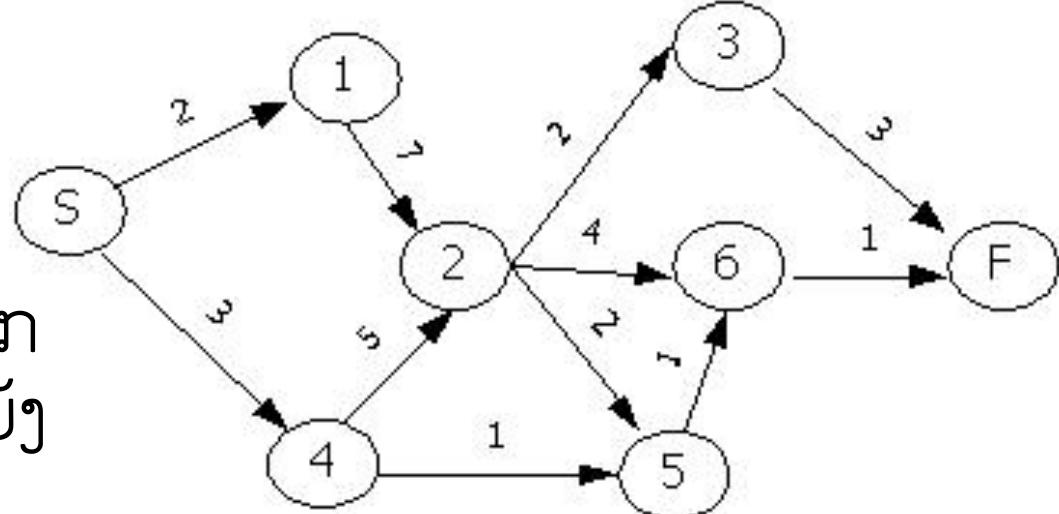
- ▶ เป็น วิธี หนึ่ง ที่ ใช้ แก้ บันชา Formal Task
  - ▶ ถือ งาน กำหนด ให้ บันชา มีสถานะ นิ่ง เลื่อน ตื้น (Initial State) และ มี จุดสิ้นสุด (Final State)
  - ▶ มี ขั้น กำหนด ของ งาน ปัจุบัน สถานะ
  - ▶ งาน แก้ บันชา ถือ งาน หา ทาง ปัจุบัน สถานะ จาก initial state ไป ยัง final state
  - ▶ เหตุ นิ ก ที่ หา ทาง ปัจุบัน สถานะ นิ่ง อาจ เร็ด ให้ ได้ ผิด รับ ที่ แตก ต่าง กัน

# ຂໍ້ຕອນການແກ້ບັນຫາແບບນີ້

- ▶ ນີ້ ຍາມ ບັນຫາ ຢ່າງ ຊັດເຈນ
  - Initial situation
  - Final situation
- ▶ ຜິຈາລະນາ ກົດ
- ▶ ເລືອກ ໃຊ້ ໂຄງ ສ້າງ ເພື່ອ ການ ແກ້ ບັນຫາ
- ▶ ເລືອກ ເທັກ ນິກ ແກ້ ບັນຫາ ທີ່ ໜໍາມາ ສົມ

# ຕົວຢາງບັນຫາ

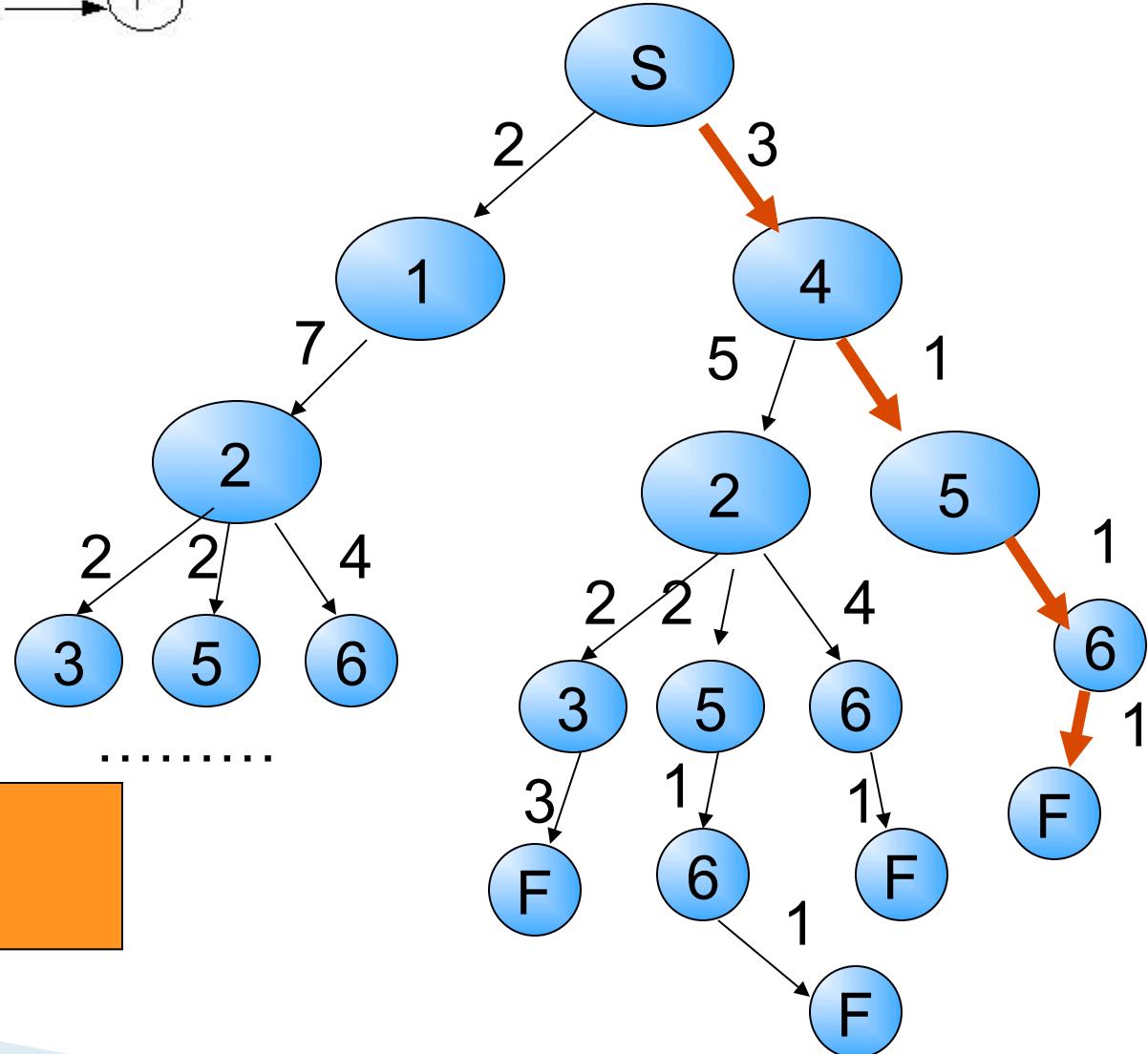
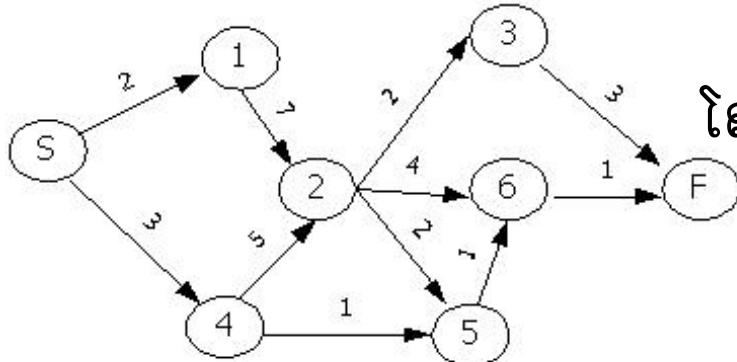
ເສັ້ນ ທາງ ການເດີນທາງ ຈາກ  
ຂົດ ຫົ່ງ ຂອງ ເມືອງ ໄປ ຍັງ  
ຂົດ ຕ່າງ ຖໍ່ ເປັນ ການເດີນ  
ທາງ ເປັນ ແບບ ທາງ ດຽວ  
ເປົ້າ ໝາຍ ຄື ທາ ເສັ້ນ ທາງ  
ທີ່ ເດີນທາງ ຈາກ S ໄປ ຍັງ  
F ໃຫ້ ໄດ້ ລະ ຍະ ທາງ ທີ່  
ສັ້ນ ທີ່ສຸດ ມີ ໃຫ້ ເລືອກ ຈັກ  
ເສັ້ນ ທາງ? ແລະ ທາງ ໄດ້  
ສັ້ນ ທີ່ສຸດ?



1.  $S \rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow F = 14$
2.  $S \rightarrow 4 \rightarrow 5 \rightarrow 3 \rightarrow F = 13$
3.  $S \rightarrow 4 \rightarrow 2 \rightarrow 5 \rightarrow 6 \rightarrow F = 12$
4.  $S \rightarrow 4 \rightarrow 5 \rightarrow 6 \rightarrow F = 6$

ເສັ້ນ ທາງ ທີ່ ດີ ທີ່ສຸດ ຄື  
ເສັ້ນ ທາງ ທີ່ 4

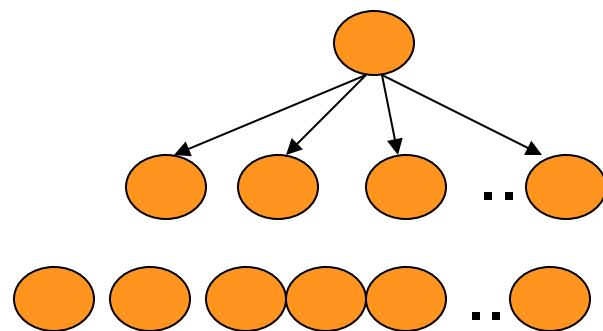
ໂຄງ ສ້າງ ຕົ້ນ ໄມ້ ສະແດງ ລໍາ ດັບ ການ ເລືອກ



ທາງເດີນ ທີ່ ດີ ທີ່ສຸດ  
 $S \ 4 \ 5 \ 6 \ F = 6$

ຂໍສົງເກດ

- ▶ ບັນຫາ ທີ່ ເຜົ່ານຸມາ ມີ ລົ້າ ດັບ ວິທີ ແກ້ ບັນຫາ ທີ່ ແຕກ ຕ່າງ ກັນ
  - ▶ ໃນ ບາງ ບັນຫາ ການ ຄົ່ນ ຫາ ຊດ ລົ້າ ດັບ ການ ແກ້ ບັນຫາ ເປັນ ໄປ ໄດ້ ຍາກ ເຊັນ ການ ຄົ່ນ ຫາ ລົ້າ ດັບ ການ ຢ່າງ ຫມາກ, ລູກ ເຜືອ ເວົ້າ ຊະນະ ຄູ ຕໍສູ ມີ ວິທີ ການ ຢ່າງ ຫລາຍື ແກ້ ໄຂ ໂດຍ ໃຊ້ ການ ຄົ່ນ ຫາ ແບບ ຮົວ ລົ ສະຕິ ກ (Heuristic Search)

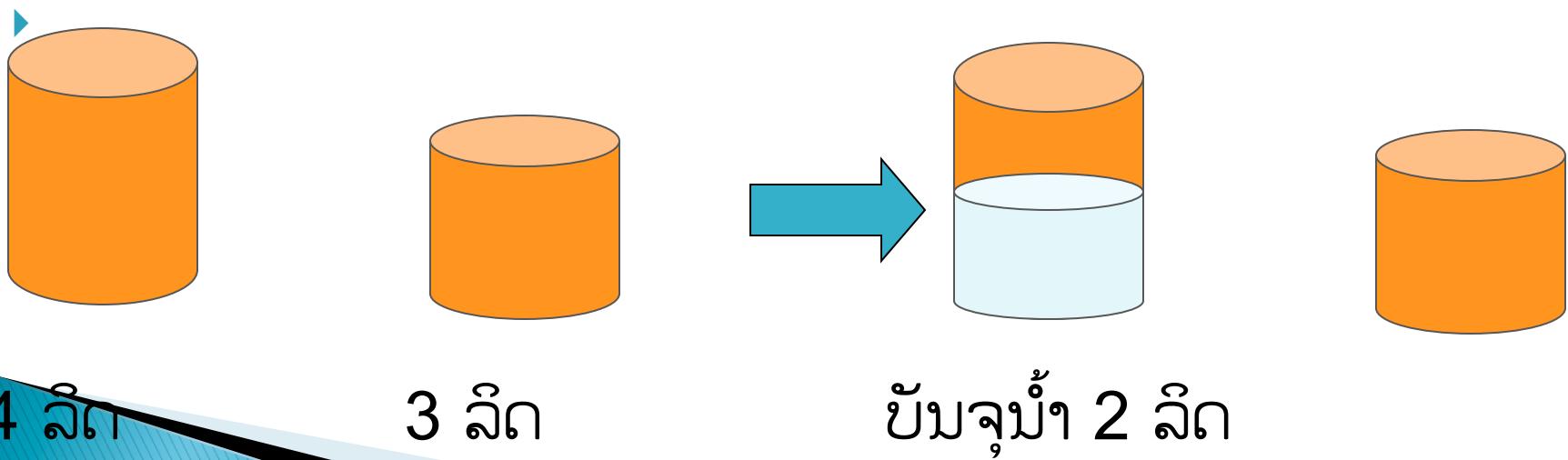


## ຫົວຂໍ້ທີ່ຈະສຶກສາໃນບິດນີ້

- 1. ການ ກຳນົດ ມີ ຍາມ ໃຫ້ ກັບ ບັນຫາ
- 2. ອົງປະກອບ ຂອງ ການ ແກ້ ບັນຫາ ດ້ວຍ ວິທີ State space search
- 3. ການ ແກ້ ບັນຫາ ດ້ວຍ ການ ຄົ້ນ ຫາ (Problem Solving by Searching)
- 4. ການ ແກ້ ບັນຫາ ດ້ວຍ ການ ຄົ້ນ ຫາ ແບບ ຮົວ ລະຕິ ກ (Problem Solving by Heuristic Search)
- 5. ການ ແກ້ ບັນຫາ ການ ຫລືນ ແກ (Game Playing)

# 1. ການກຳນົດມີຢາມໃຫ້ກັບບັນຫາ

- ▶ ຕີ່ ການ ອະທິບາຍ ລັກສະນະ ບັນຫາ ເຜື່ອ ທີ່ ຈະ ຫາ ວິທີ ການ ແກ້ໄຂ
- ▶ ຕົວ ຢ່າງ ທີ່ 1: ບັນຫາ ໂທ ນັ້ນ ກຳນົດ ໃຫ້ ມີ ໂທ ນັ້ນ 2 ອັນ ອັນ 1 ມີ ຄວາມ ຈຸ່ 4 ລົດ ແລະ ອັນ ທີ່ ສອງ ມີ ຄວາມ ຈຸ່ 3 ລົດ ຈະ ເຮັດ ຢ່າງໃດ ໃຫ້ ໂທ ນັ້ນ ອັນທຳອິດ ນີ້ ມີ ຄວາມ ຈຸ່ 2 ລົດ ດີ ໂດຍ ບໍ່ ມີ ເຄື່ອງ ວັດ ໃດ ຖ່າ



# ການກຳນົດນິຍາມຂອງບັນຫາໂທນີ້

- ▶ 1. ຄ່າ ຂອງ ໂອ ກາດ ຕ່າງໆ ທີ່ ຈະ ເກີດ ຂຶ້ນ ໃນ ແຕ່ ລະ ສະຖານ ຂະ ກຳນົດ ໃຫ້ ຕົວ ປ່ຽນ  $x$  ແຫນ ປະລິມານ ນີ້ ໃນ ໂທ ນີ້ ທີ່ 1 ແລະ  $y$  ແຫນ ປະລິມານ ນີ້ ໃນ ໂທ ທີ່ ສອງ
- ▶ ຈະ ໄດ້ ວ່າ  $x = 0, 1, 2, 3, 4$  (ກຳອິດຄວາມ ຈຸ່ 4 ລົດ)
- ▶  $y = 0, 1, 2, 3$  (ກຳອິດຄວາມ ຈຸ່ 3 ລົດ)
- ▶ 2. ກຳນົດ ຈຸດ ເລີ່ມ ຕົ້ນ ແລະ ສິ້ນສຸດ ການ ແກ້ ບັນຫາ
- ▶ - ສະ ຖາ ນະ ເລີ່ມ ຕົ້ນ ກຳນົດ ໃຫ້ ສະ ຖາ ນະ ເລີ່ມ ຕົ້ນ ຂອງ ບັນຫາ ນີ້ ຄື ບໍ່ ມີ ນີ້ ໃນ ໂທ ນີ້ ຫັງ ສອງ ດັ່ງ ນັ້ນ ເຮົາ ຈຶ່ງ ໃຊ້ ອຸ່ນ ດັບ  $(x, y)$   $x =$  ປະລິມານ ນີ້ ໃນ ໂທ ນີ້ ກຳອິດ ແລະ  $y =$  ປະລິມານ ນີ້ ໃນ ໂທ ທີ່ ສອງ
  - ສະ ຖາ ນະ ສິ້ນສຸດ ຕ້ອງ ການ ໃຫ້ ເຫລືອ ນີ້ ໃນ ໂທ ນີ້ ກຳອິດຜຽງ 2 ລົດ ສ່ວນ ໂທ ນີ້ ໃບ ທີ່ ສອງ ຈະ ເຫລືອ ຄວາມ ຈຸ່ ເທົ່າ ໄດ້ ກຳ ໄດ້

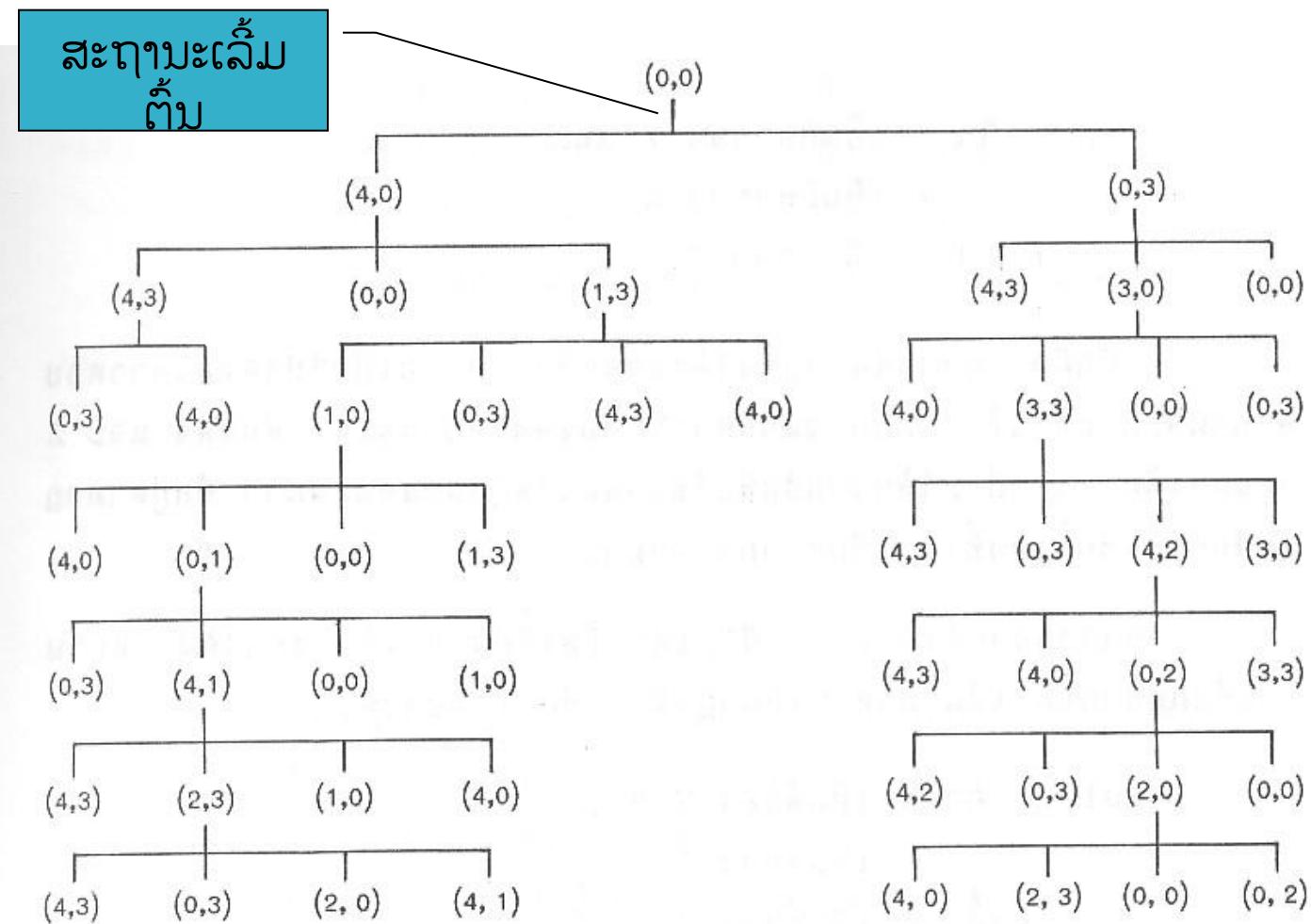
# ການກຳນົດນິຍາມຂອງບັນຫາໂກນ້ຳ

- ▶ 3. ກົດ ທີ່ ອະທິບາຍ ບັນຫາ ນັ້ນ ຖ  
  - ▶ - ໄສ່ ນໍ້າ ໃນ ໂທ 1 ເຕັມ
  - ▶ - ໄສ່ ນໍ້າ ໃນ ໂທ ໃບ ທີ່ ສອງ ຈົນ ເຕັມ
  - ▶ - ເທ ນໍ້າ ບາງ ສ່ວນ ອອກຈາກ ໂທ 1
  - ▶ - ເທ ນໍ້າ ບາງ ສ່ວນ ອອກຈາກ ໂທ ໃບ ທີ່ ສອງ
- ▶ ປຽນ ກົດ ໃຫ້ເປັນ ເງື່ອນ ໄຂ ທີ່ ງ່າຍ ຕໍ່ ການ ນໍາ ໄປ ໃຊ້  
    ຈະ ໄດ້ ດັ່ງ ດັ່ງ ນີ້

# ການກຳນົດນິຍາມຂອງບັນຫາໂກນ້ຳ

ລຳດັບກົດ	ກົດ	ຄໍາອະທິບາຍ
1.	$(X, Y : X < 4) \rightarrow (4, Y)$	ຖອກນ້ຳໃສ່ໂຖທີ່ ອິດຈິນເຕັມ
2.	$(X, Y : Y < 3) \rightarrow (X, 3)$	ຖອກນ້ຳໃສ່ໂຖທີ່ 2 ຈິນເຕັມ
3.	$(X, Y : X > D) \rightarrow (X-D, Y)$	ຖອກນ້ຳບາງສ່ວນອອກຈາກໂຖທີ່ ອິດ
4.	$(X, Y : Y > D) \rightarrow (X, Y-D)$	ຖອກນ້ຳບາງສ່ວນອອກຈາກໂຖ 2
5.	$(X, Y : X > 0) \rightarrow (0, Y)$	ຖອກນ້ຳອອກຈາກໂຖທີ່ ອິດຈິນໜີດ
6.	$(X, Y : Y > 0) \rightarrow (X, 0)$	ຖອກນ້ຳອອກຈາກໂຖທີ່ 2 ຈິນໜີດ
7.	$(X, Y : X+Y \geq 4 \wedge Y > 0) \rightarrow (4, Y-(4-X))$	ຖອກນ້ຳອອກຈາກໂຖທີ່ 2 ໃສ່ໂຖທີ່ ອິດຈິນເຕັມ ໂດຍໂຖທີ່ 1 ມີນ້ຳລວມກັນຫຼາຍກວ່າ 4 ລົດ
8.	$(X, Y : X+Y \geq 3 \wedge X > 0) \rightarrow (X-(3-Y), 3)$	ຖອກນ້ຳອອກຈາກໂຖທີ່ ອິດ ໃສ່ໂຖທີ່ 2 ຈິນເຕັມ ໂດຍໂຖທີ່ 2 ມີນ້ຳລວມກັນຫຼາຍກວ່າ 3 ລົດ
9.	$(X, Y : X+Y \leq 4 \wedge Y > 0) \rightarrow (X+Y, 0)$	ຖອກນ້ຳອອກຈາກໂຖທີ່ 2 ໃສ່ໂຖທີ່ 1
10.	$(X, Y : X+Y \leq 3 \wedge X > 0) \rightarrow (0, X+Y)$	ຖອກນ້ຳອອກຈາກໂຖທີ່ ອິດ ໃສ່ໂຖທີ່ 2

ຂັ້ນ ຕອນ ການ ເລືອກ ກິດ ສະແດງ ດ້ວຍ ໂຄງ ສ້າງ ຕົ້ນ ໄມ



# ติวปางงานแก้บันชา

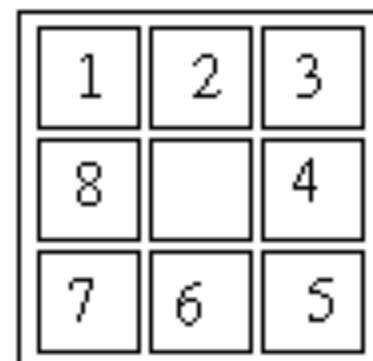
ลำดับ	โจที่ 1	โจที่ 2	กิต	สถานะ
1.	0	0	-	สถานะล้มตื้น
2.	0	3	2	
3.	3	0	9	
4.	3	3	2	
5.	4	2	7	
6.	0	2	5	
7.	2	0	9	สถานะสิ้นสุด

# ຕົວຢາງ: ບັນຫາ 8-Puzzle

- ▶ ບັນຫາ 8-Puzzle ປະກອບ ດ້ວຍ ຖາດ ຂະໜາດ  $3 \times 3$  ຂຶ່ງ ພາຍໃນ ບັນ ຈຸ ແຜ່ນ ຂະໜາດ  $1 \times 1$  ຈຳນວນ 8 ແຜ່ນ ໃນ ແຕ່ ລະແຜ່ນ ຈະ ມີ ຫມາຍ ເລກ ສະແດງ ແລະ ມີ ຊ່ອງ ວ່າງ ຢູ່ ຫົ່ງ ຊ່ອງ ທີ່ ແຜ່ນ ສາມາດ ເລື່ອນ ແຂ້າ ມາ ແທນ ທີ່ ດັ່ງ ບັນຫາ ຄື ໃຫ້ ເລື່ອນ ແຜ່ນ ເຫລື່າ ນີ້ ຈາກ ຕຳແໜ່ງ ເລີ່ມ ຕົ້ນ ໃຫ້ເປັນ ຕຳແໜ່ງ ສຸດ ທ້າຍ ຂຶ່າ ເປັນ ຄຳ ຕອບ



ຕຳແໜ່ງ ເລີ່ມ ຕົ້ນ



ຕຳແໜ່ງ ສິ້ນສຸດ

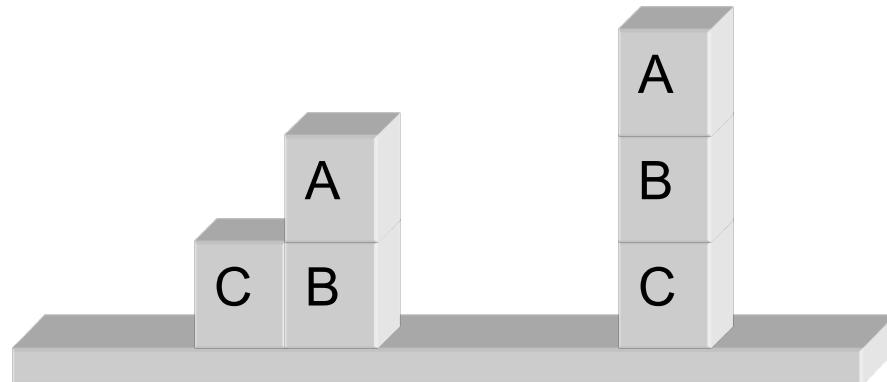
# ການກຳນົດນິຍາມຂອງບັນຫາ 8-Puzzle

1. ໃຫ້  $[x, y]$  ແກນ ຄ່າ ຂອງ ໂອ ກາດ ຕ່າງໆ ທີ່ ຈະ ເກີດ ແລ້ວ ໃນ ແຕ່ ລະ ສະຖານະ
2. ກຳນົດ ຈຸດ ເລີ່ມ ຕົ້ນ ແລະ ສິ້ນສຸດ ການ ແກ້ ບັນຫາ
  - ສະຖານະ ເລີ່ມ ຕົ້ນ  $[2, 4, 1; 8, 5, 6; 3,, 7]$
  - ສະຖານະ ສິ້ນສຸດ  $[1, 2, 3; 8,, 4; 7, 6, 5]$
3. ກົດ ທີ່ ອະທິບາຍ ບັນຫາ ນັ້ນ ເງື່ອ
  - ເຄື່ອນ ແຜ່ນ ທີ່ ບໍ່ ກຳ ກັບ ຫມາຍ ເລກ ແລ້ວ ດ້ວນ ເທິງ
  - ເຄື່ອນ ແຜ່ນ ທີ່ ບໍ່ ກຳ ກັບ ຫມາຍ ເລີ່ງ ດ້ວນ ລຸມ
  - ເຄື່ອນ ແຜ່ນ ທີ່ ບໍ່ ກຳ ກັບ ຫມາຍ ເລກ ໄປ ທາງ ດ້ວນ ຊັ້ນ
  - ເຄື່ອນ ແຜ່ນ ທີ່ ບໍ່ ກຳ ກັບ ຫມາຍ ເລກ ໄປ ທາງ ດ້ວນ ຂວາ

# ប៉ាន្ហាតី ៣: Block World Problem

Block World Problem តិចប៉ាន្ហាតី ដែលត្រូវការងារចំណាំរវាងការបញ្ចប់របស់ការបញ្ចប់ទាំងអស់ ដើម្បីធ្វើការដែលត្រូវបានផ្តល់ជាផ្លូវការ។

- ការបញ្ចប់ X ត្រូវបានធ្វើឡើងដែលមានការបញ្ចប់ទាំងអស់ទាំងពីរ។
- ការបញ្ចប់ X និង Y ត្រូវបានធ្វើឡើងដែលមានការបញ្ចប់ទាំងពីរ។



Initial state  
state

final

# ການກຳນົດນິຍາມບັນຫາໃຫ້ກັບ Block World Problem (ໃຫ້ຂຽນ)

1. ຄ່າ ຂອງ ໂອ ກາດ ຕ່າງ ຖໍ່ທີ່ ຈະ ເກີດ ຂຶ້ນ ໃນ ແຕ່ ລະສະຖານະ?  
- ?
2. ຈຸດ ເລີ່ມ ຕົ້ນ ແລະ ສິ້ນສຸດ ການ ແກ້ ບັນຫາ?  
ສະຖານະ ນະ ເລີ່ມ ຕົ້ນ?  
ສະຖານະ ນະ ສິ້ນສຸດ?
3. ກົດ ທີ່ ອະທິບາຍ ບັນຫາ ນັ້ນ ຖໍ່  
-?

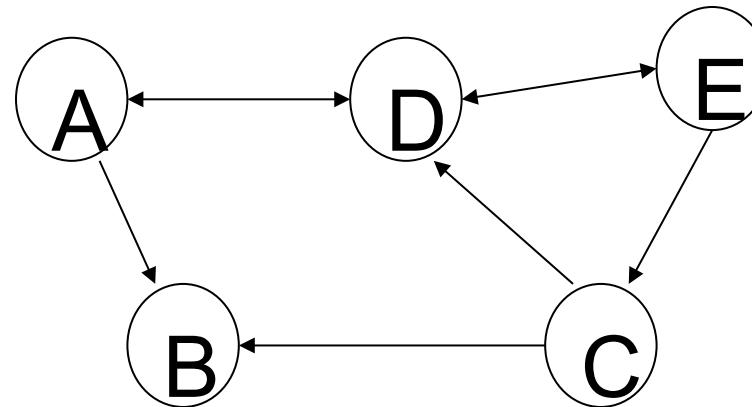
## 2. ອີງປະກອບຂອງການແກ້ບັນຫາ ດ້ວຍວິທີການຄືນຫາແບບບໍ່ມີຂອບເຂດສະຖານະ

- ຮູບແບບໂຄງສ້າງຂຶ້ນທີໃຊ້ໃນການຄືນຫາ
- ການກຳນົດທີດຫາງສໍາລັບການຄືນຫາ
- ການສະແດງຄວາມຮູ້
- ກະບວນການໃນການເລືອກກິດ
- ການຄືນຫາ (Search)

## 1. ຮູບແບບໂຄງສ້າງຂຶ້ມູນທີ່ໃຊ້ໃນການຄົ້ນຫາ

- ▶ ໂຄງ ສ້າງ ຕົ້ນ ໄມ້ ແລະ ໂຄງ ສ້າງ ກາ ດ ເປັນ ທີ່ ນິຍິມ ໃຊ້ ເປັນ ໂຄງ ສ້າງ ສໍາລັບ  
ການ ຄົ້ນ ຫາ ເຊັ່ນ ຕົວ ຢ່າງ ການ ຫາ ໄລ ຍະ ຫາງ ທີ່ ສັ້ນ ທີ່ສຸດ ໃນ ທີ່ ດັ່ງ ສະແດງ ຂ້າງ  
ຕົ້ນ

ໂຄງສ້າງກາຟົກີເຊັດຂອງໂຫມດ(Nodes)ແລະອາລົກທີ່ໃຊ້ເຊື່ອມລະຫວ່າງໂຫມດເຂົ້ານຳກັນແຕ່ລະໂຫມດຈະມີຊື່ປະຈຳໂຫມດຜົ່ອແຍກຄວາມແຕກຕ່າງລະຫວ່າງໂຫມດອອກຈາກກັນໃນອາລົກກະອາດມີຄຳອະທິບາຍຫລືຄ່າຜົ່ອປຶງບອກເຖິງຄຸນສົມບັດຂອງອາລົກກະໜັນງາມເຊັ່ນຄ່າcostຂອງເສັ້ນທາງທີ່ເຊື່ອມລະຫວ່າງໂຫມດອາລົກອາດຈະເປັນທິດທາງໄປດຽວເຮົາຮຽກວ່າdirected

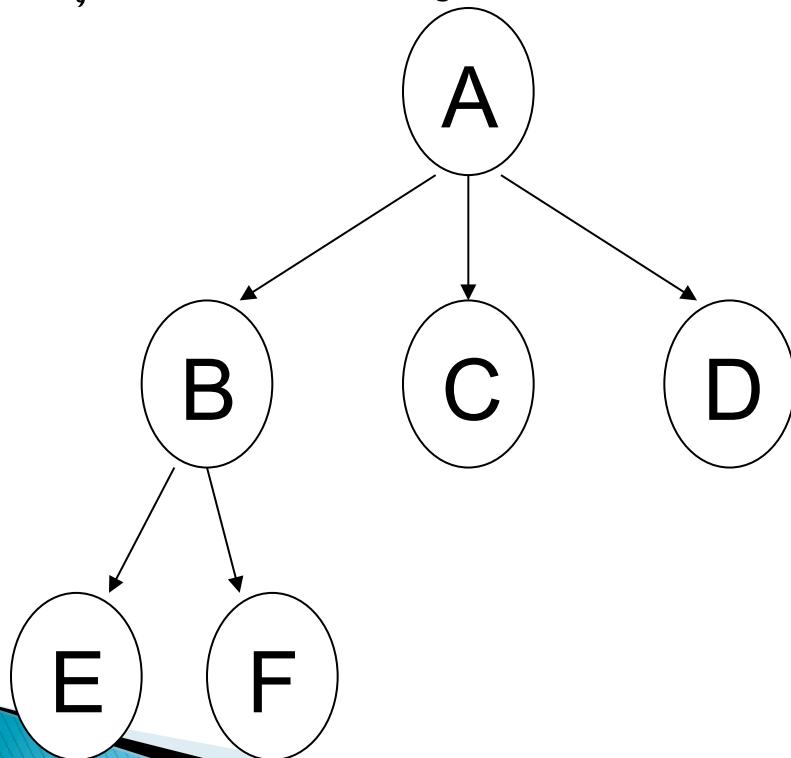


Nodes = {A,B,C,D,E}

$$\text{Arcs} = \{(A,B), (A,D), (B,C), (C,D), (D,A), (D,E), (C,E), (E,D)\}$$

## ຮູບແບບໂຄງສ້າງຂຶ້ມູນທີ່ໃຊ້ໃນການຄົ້ນຫາ (ຕຳ)

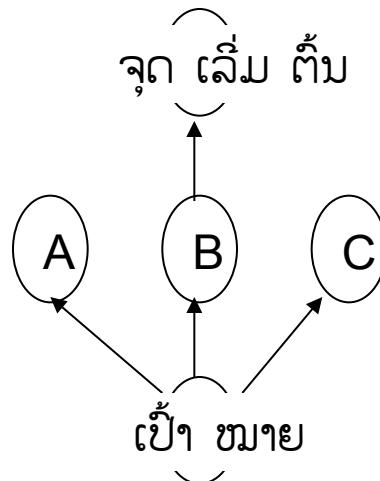
ໂຄງສ້າງຕົ້ນໄມ້ຄື ໂຄງສ້າງກາຟຊະນິດໜຶ່ງທີ່ສອງໂທນັດ ຈະມີເສັ້ນເຊື້ອມລະຫວ່າງກັນບໍ່ເກີນໜຶ່ງເສັ້ນແລະຈະມີrootສະເໜີເຜື່ອເປັນຈຸດເລີ່ມຕົ້ນຂອງຕົ້ນໄມ້ຕົ້ນນັ້ນ



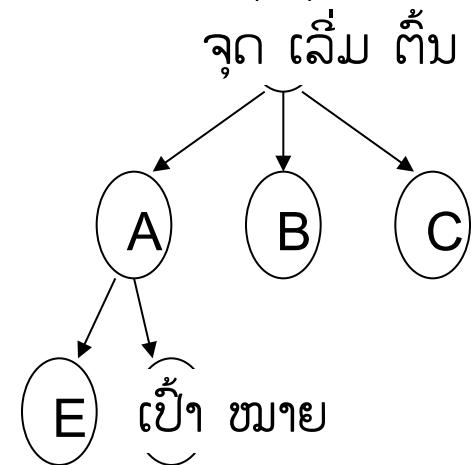
## 2. ການກຳນົດທິດທາງສໍາລັບການຄື່ນຫາ

ໃນ ການ ແກ້ ໄຂ ບັນຫາ ຫົ່ງ ໃນ ລະບົບ ຂອບເຂດ ສະຖານ ຂະໜົນ ເຊິ່ງ ຈະ ມີສະ ຖາ  
ນະ ເລີ່ມ ຕົ້ນ ຂອງ ບັນຫາ ຈາກ ນັ້ນ ຈຶ່ງ ໃຊ້ ວິທີ ການ ຕ່າງໆ ເພີ່ມປ່ຽນສະ ຖາ ນະ  
ໃຫ້ ນຳ ໄປ ສູ່ ເປົ້າ ຫມາຍ ຂອງ ຄວາມ ສໍາ ເລັດ ທິດ ທາງ ການ ປ່ຽນ ສະ ຖາ ນະ ເພື່ອ  
ແກ້ ບັນຫາ ເຮັດ ໄດ້ 2 ທິດ ທາງ

1. Forward reasoning ຈຸດ  
ເລີ່ມ ຕົ້ນ + ການ ໃຊ້ ກົດ-> ເປົ້າ  
ຫມາຍ



2. Backward reasoning -- ເປົ້າ  
ຫມາຍ + ການ ໃຊ້ ກົດ-> ຈຸດ ເລີ່ມ  
ຕົ້ນ

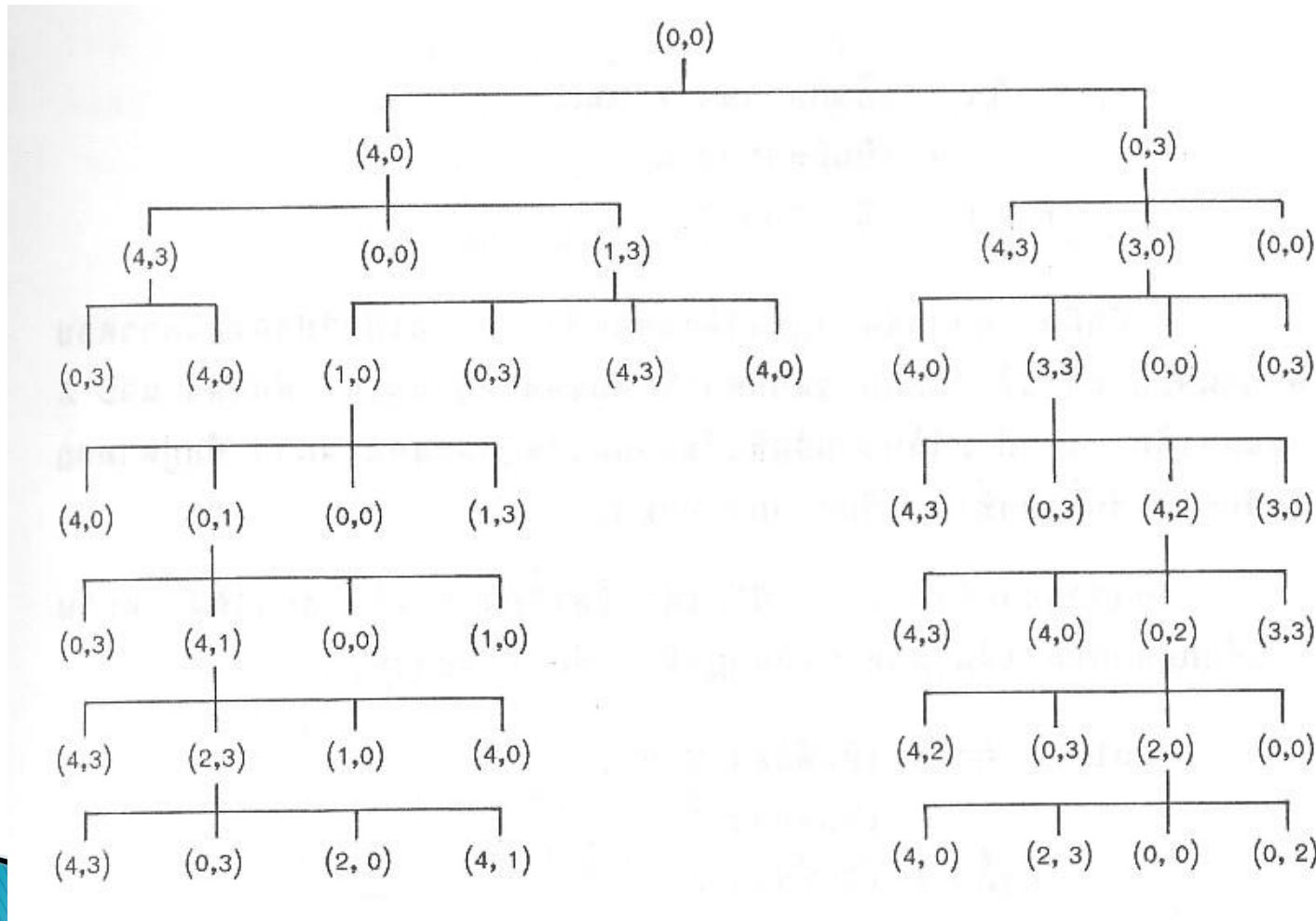


### 3. ການສະແດງຄວາມຮູ້

## 4. ກະບວນການໃນການເລືອກກົດ

4.1 ການຮັດ ດັດ ສະນີ (Indexing) ຈະ ເໝາະ ສົມ ສໍາລັບ ການ ສະແດງ ຄວາມ ຮູ່ ໃນ ລະບົບ ການ ຜະລິດ ທີ່ ວິທີ ການ ປ່ຽນ ສະຖາ ນະ ຕ່າງໆ ໄດ້ ຖືກ ກຳນົດ ໄວ໊ ເປັນ ເງື່ອນ ໄຂ ໄວ໊ ກ່ອນ ຕົວ ຢ່າງ ເຊັນ ການ ແກ້ ໄຂ ບັນຫາ ໂທ ນຳ ເງື່ອນ ໄຂ ຕ່າງໆ ຖືກ ສ້າງ ໄວ໊ ກ່ອນ ແລະ ຖືກ ນຳ ມາ ໃຊ້ ໃນ ຂັ້ນ ຕອນ ການ ຄົ້ນ ຫາ ຄຳ ຕອບ ເຜືອ ການ ອະທິບາຍ ວິທີ ການຮັດ ດັດສະ ນີ້ ໃຫ້ ຂູ້ດາວໂຫຼນ ຂຶ້ນ ຈຶ່ງ ຍົກ ຕົວ ຢ່າງ ດັ່ງ ນີ້ ໃນ ການ ແກ້ ບັນຫາ ໂທ ນຳ ນົ້ນ ສະ ຖາ ນະ ເລີ່ມ ຕົ້ນ ຂອງ ນຳ ໃນ ໂທ໌ຫາ ສອງ ເປັນ  $(0, 0)$  ເຜືອ ທີ່ ຈະ ນຳ ໄປ ສູ່ ສະຖານະ  $(2, \theta)$  ໃນ ແຕ່ ລະ ສະ ຖາ ນະ ຈະ ມີ ກົດ ບາງ ຈຳນວນ ຖືກ ໃຊ້ ເຜືອ ປ່ຽນ ສະຖານະ ເຊັນ ໃນ ສະ ຖາ ນະ ເລີ່ມ ຕົ້ນ  $(0, 0)$  ຈະ ມີ ກົດ ພຽງ ສອງ ກົດ ທີ່ ນຳ ມາ ໃຊ້ ໃນ ສະຖານະ ນີ້ ໄດ້ ຄື ກົດ ທີ່ 1 ແລະ ກົດ ທີ່ 2

## 4. ກະບວນການໃນການເລືອກກົດ (ຕໍ່)



## 4. ກະບວນການໃນການເລືອກກົດ (ຕໍ່)

### 4.2 ການ ປຽບທຽບ ຕົວ ປ່ຽນ (Matching variable)

ຈາກ ວິທີ ກຳນົດ ດັດ ນີ້ ຈະ ໃຊ້ ໄດ້ ກັບ ການ ສະແດງ  
ຄວາມ ຮູ່ ໃນ ລະບົບ ການ ຜະລິດ ສໍາລັບ ການ ປຽບທຽບ ຕົວ  
ແປ ຈະ ໃຊ້ ໄດ້ ກັບ ວິທີ ການ ສະແດງ ຄວາມ ຮູ່ ແບບ  
Predicate Logic ຊຶ່ງ ລາຍ ລະອຽດ ຂອງ ການ ສະແດງ  
ຄວາມ ຮູ່ ແບບ ນີ້ ມີ ລາຍ ລະອຽດ ຢູ່ໃນ ບົດ ທີ່ 3 ຊຶ່ງ ໄດ້  
ກ່າວເຖິງ ວິທີ ການ ສະແດງ ຄວາມ ຮູ່ ແລະ ວິທີ ກາ ນອະນຸ  
ມານ ດ້ວຍ

## 4. ກະບວນການໃນການເລືອກກົດ (ຕຳ)

ກາ ນອະນຸ ມານ ຄວາມ ຮູ່ ໃນ ແບບ Predicate logic ສາມາດ ຍືກ ຕົວ  
ຢ່າງ ເປື້ອງ ຕົ້ນ ໄດ້ ດັ່ງ ນີ້

ກຳນົດ ໃຫ້ ປະ ໂຍກ ໜ້າ ນີ້ ເປັນ ຈຶ່ງ

1. ແດ້ ໃຫຍ່ ເປັນ ຜໍ່ ຂອງ ແດ້

2. ແດ້ ເປັນ ຜໍ່ ຂອງ ແດ້ ນ້ອຍ

ທາກ ຕ້ອງ ການ ຮູ່ວ່າ ແດ້ ໃຫຍ່ ເປັນ ຫຍັງ ກັບ ແດ້ ນ້ອຍ ຄອມຜູ້ວ  
ເຕີ ຍັງ ຕອບ ບໍ່ ໄດ້ ແຕ່ ຖ້າ ເຮົາ ໃຫ້ ຄວາມ ຮູ່ ຫລື ກົດ ບາງຢ່າງ ທີ່  
ອະທິບາຍ ຄວາມ ສຳຜັນ ລົງ ໄປ ເຊັ່ນ

ກົດ: ຖ້າ X ເປັນ ຜໍ່ ຂອງ Y ແລະ

Y ເປັນ ຜໍ່ ຂອງ Z ແລ້ວ X ເປັນ ປຸ່ ຂອງ Z

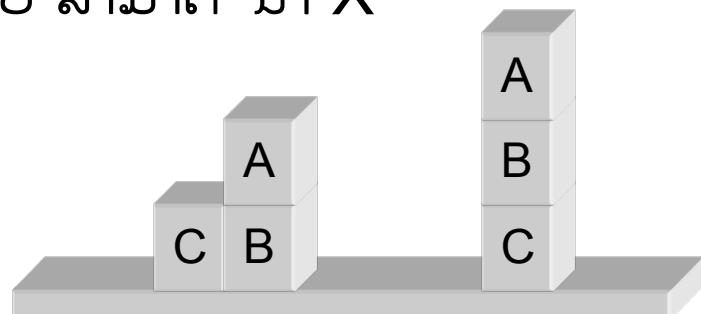
ດັ່ງ ນັ້ນ ໃນ ກະ ບວນ ກາ ນອະນຸ ມານ ຄວາມ ຮູ່ ນີ້ ໃຫ້ X ເປັນ ແດ້  
ໃຫຍ່, Y ເປັນ ແດ້ ແລະ Z ເປັນ ແດ້ ນ້ອຍ ກໍ ຈະ ສະຫາລຸບ ໄດ້ ວ່າ  
ແດ້ ໃຫຍ່ ເປັນ ປຸ່ ຂອງ Z ກະ ບວນ ການ ນີ້ ເຮົາ ອຽກວ່າ ການ  
ປຽບທຽບ ຕົວ ແປ້ ນັ້ນ ໂອງ

## 5. ການຄົ້ນຫາ (Search)

- ▶ ຄື ເທກ ນິກ ທີ່ ໃຊ້ ໃນ ການ ຄົ້ນ ຫາ ວິທີ ການ ປັບປຸງ ສະຖານະ
- ▶ ຫລື ຄົ້ນ ຫາ ຄຳ ຕອບ ເຜື່ອ ໃຫ້ ຜົບ ກັບ ຄຳ ຕອບ ທີ່ ຕ້ອງ ການ
- ▶ ແບ່ງ ອອກ ເປັນ 2 ແບບ
  - ການ ແກ້ ບັນຫາ ດ້ວຍ ການ ຄົ້ນ ຫາ (Problem Solving by Searching)
  - ການ ແກ້ ບັນຫາ ດ້ວຍ ການ ຄົ້ນ ຫາ ແບບ ຮິວ ລີ ສະຕິ ກ (Problem Solving by Heuristic Search)
- ▶ ຊຶ່ງ ຈະ ຍົກ ໄປ ກ່າວ ໃນ ຫົວຂໍ້ ທີ່ 3 ແລະ 4 ແກນ

# 3. Problem Solving by Searching

- ▶ **ຮຽກ Uninformed Search ຫລື Blind Search** ເປັນ ວິທີການຄົ້ນຫາ ທີ່ບໍ່ໃຊ້ຄວາມຮູ້ຮອບຂ້າງ (ຄວາມຮູ້ທີ່ມີປະໂຫຍດຕໍ່ການແກ້ບັນຫາ) ເຊົ້າມາຊ່ວຍໃນການແກ້ບັນຫາແຕ່ຢ່າງໃດເປັນວິທີທີ່ງ່າຍ
  - ▶ **Block World Problem** ຄືບັນຫາທີ່ຈະຕ້ອງການຈັດວາງກ່ອງໃຫ້ລວງລຳດັບກັນຢູ່ເທິງໂຕະດັ່ງຕົວຢ່າງຂ້າງລຸມນີ້ສໍາລັບເງື່ອນໄຂກໍຄື
    1. ກ່ອງ X ທີ່ບໍ່ມີກ່ອງອື່ນທັບຢູ່ສາມາດນຳມາວາງເທິງໂຕະໄດ້
    2. ກ່ອງ X ແລະ Y ບໍ່ມີກ່ອງອື່ນທັບສາມາດນຳ X ມາທັບເທິງ Y

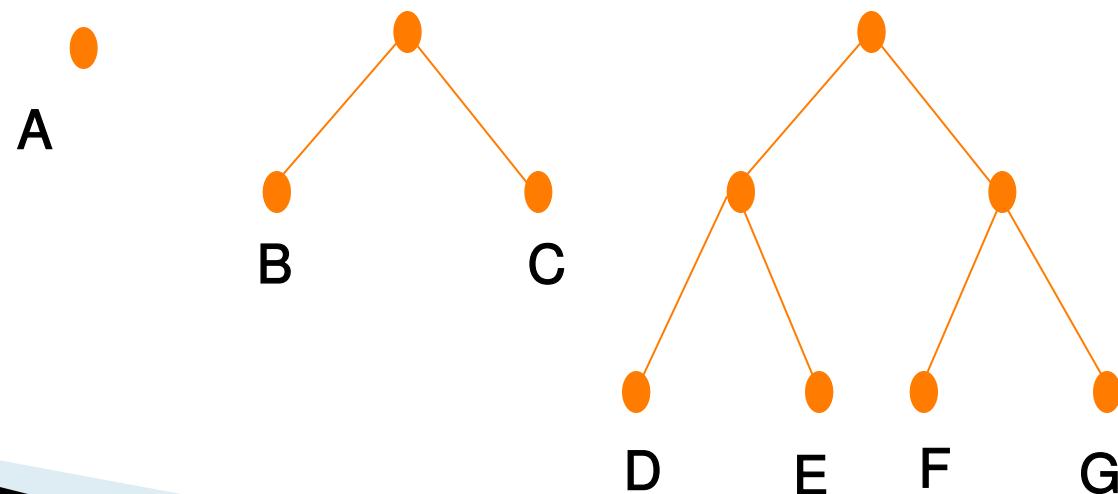


ສະ ຖາ ນະ ເລີ່ມ ຕິນ  
ເປົ້າ ຫາຍ

ສະ ຖາ ນະ

# Breadth-First Search

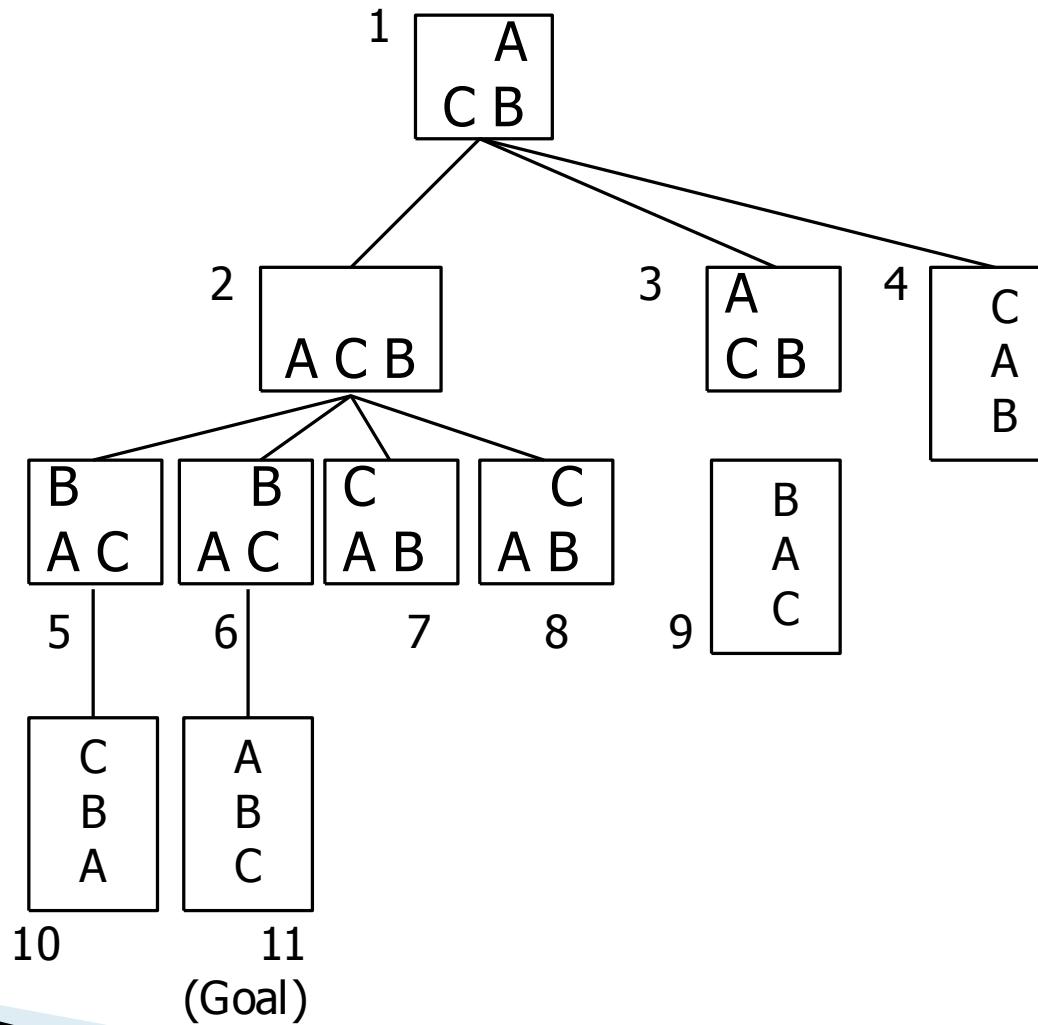
- ຄື ວິທີ ການ ທີ່ ແຕ່ ລະ ສະ ຖາ ນະ ຈະ ກໍ່ການ ວິ ເຄາະ ແລະ ໂໝນດ ລູກ ຂັ້ນ ມາ ຈາກ ນັ້ນ ໂໝນດ ລູກ ແຕ່ ລະ ຕົວ ຈະ ຖືກ ທິດສອບ ວ່າ ເປັນ ສະ ຖາ ນະ ເປົ້າ ຫມາຍ ຫລື ບໍ່ ຫາກ ເປັນ ກຳ ຈະ ຢຸດ ເຮັດວຽກ ຫັນ ທີ່ ຫາກ ຍັງ ບໍ່ ເປັນ ສະ ຖາ ນະ ເປົ້າ ຫມາຍ ໂໝນດ ນັ້ນ ກຳ ຈະ ຖືກ ນຳ ໄປ ສ້າງ ໂໝນດ ລູກ ໃນ ລະ ດັບ ຖັດ ໄປ ແລ້ວ ຈຶ່ງ ເກັບ ໄວ້ ໃນ ໂຄງ ສ້າງ ຄົວ (Queue) ກ່ອນ ແຕ່ ລະ ດັບ ຈະ ຖືກ ຜິຈາລະນາ ໃຫ້ ຮຽບຮ້ອຍ ກ່ອນ ກ່ອນທີ່ຈະ ໄປ ຜິຈາລະນາ ລະ ດັບ ຖັດ ໄປ (ສະ ຖາ ນະ ທີ່ ຖືກ ວິ ເຄາະ ຈະ ມີ ການ ກວດ ສະ ຖາ ນະ ຊົ້າ ກ່ອນທີ່ຈະ ສ້າງ ໂໝນດ ລູກ ສະເໜີ)



# Algorithm: Breadth-First Search

```
1. NODE-LIST: = {initial state}          /* โถง ส້າງ  
queue/  
2. Until (តើម ិន Goal State ហលើ NODE-LIST វា ០) Do  
    1.1 កើង សະមាច្នឹង ពីរ ហំអិត (ឱ្យ ខ្លឹម វា E) ទទួល មា ជាង  
NODE-LIST  
    1.2 FOR EACH operator ហើយ match រាប់ E  
DO  
    2.2.1 ឱ្យ Operator នីមួយៗ ស៉ាង state ឲ្យ  
    2.2.2 IF state ឲ្យ នីមួយៗ បែង goal state THEN  
quit និង តើម តារា state នីមួយៗ  
    ELSE ដើរ state ឲ្យ នីមួយៗ ឱ្យ ហើយ ខ្លួច NODE-  
LIST
```

# Breadth-First Search (sample)



# ເຄື່ອງຊີວັດຄຸນນະພາບຂອງວິທີການຄົ້ນຫາ

- ▶ **Completeness:** ຮັບຮອງ ຄວາມ ສາມາດ ໃນ ຄົ້ນ ຫາ ວິທີ ການ ແກ້ ບັນຫາ ໄດ້ ສະເໜີ ເມື່ອ ມີ ວິທີ ການ ແກ້ ບັນຫາ ນັ້ນ ຢູ່
- ▶ **Optimality:** ສາມາດ ຫາ ວິທີ ການ ແກ້ ບັນຫາ ທີ່ ດີ ທີ່ສຸດ ໄດ້
- ▶ **Time Complexity:** ໃຊ້ ເວລາ ດົນ ສໍາໃດ ໃນ ການເຮັດວຽກ ໂດຍ ປຶກກະຕິ ຈະ ໃຊ້ ຄ່າ Big O ຊື່ ເປັນ ຜັງກຳ ຊັນ ທີ່ ບອກ ຄວາມ ຊັບ ຊ້ອນ ຂອງ ການເຮັດວຽກ ຂອງ ອັນ ກໍ ລື ທຶນ ຕ່າງໆ ເຊັ່ນ  $O(n)$ ,  $O(\log(n))$ ,  $O(n^2)$  ເປັນ ຕົ້ນ

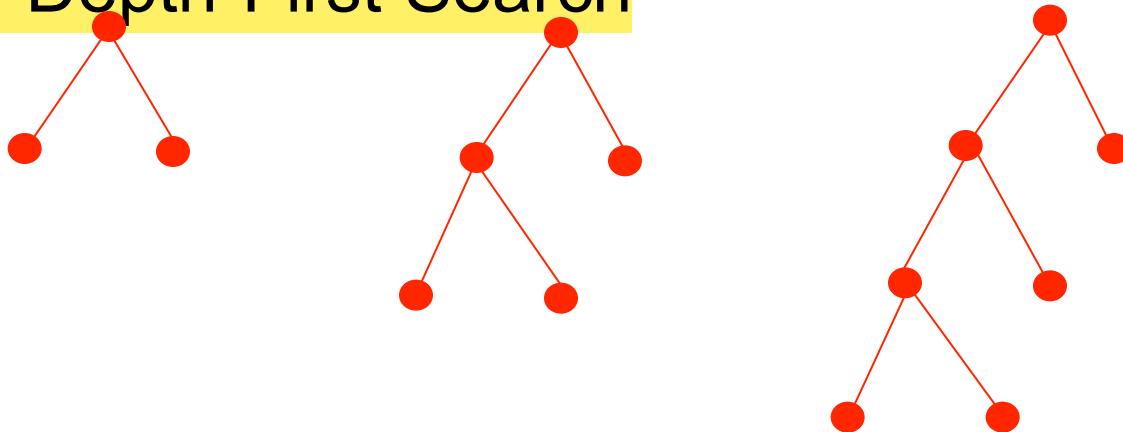
**Space Complexity:** ໃຊ້ ຂະໜາດ ຂອງ ຫນ່ວຍ ຄວາມ  
ຈຳ ເທົ່າ ໄດ້

# ປະເມີນຄຸນນະພາບຂອງ Breadth-First Search

- ▶ Completeness: ແມ່ນ
- ▶ Optimality: ໂດຍ ຫົວ ໄປ ບໍ່ ແຕ່ ໃນ ກໍລະນີ ທີ່ ລະ ດັບຊັ້ນ ມີ ຄ່າ cost ເກົ່າ ກັນ ໂຮມດ ທາງ ດ້ວຍ ຊ້າຍ ແລະ ຂວາ ຈະ ມີ cost ເກົ່າ ກັນ ຈຶ່ງ ຈະ ເຮັດໃຫ້ ການ ເລືອກ ໂຮມດ ທາງ ຊ້າຍ Optimal ເຊັ່ນ ກັນ
- ▶ Time Complexity:  $O(bd)$      $b$ =ຂະໜາດ ຂອງ ໂຮມດ ໃນ ແຕ່ ລະ level;  $d$ =ຄວາມ ສູງ ຂອງ tree
- ▶ Space Complexity  $O(b^d)$

# Depth-First Search

ຄື ວິທີ ການ ທີ່ ວິ ເຕະ ແລະ ສ້າງ ໂໝດ ລູກ ທາງ ດ້ານ ຂໍາຍ  
ຂຶ້ນ ມາ ກ່ອນ ຈາກ ນັ້ນ ຈຶ່ງ ກວດ ສອບ ວ່າ ໂໝດ ທີ່ ສ້າງ  
ຂຶ້ນ ມາ ນັ້ນ ສາມາດ ກະ ຈາຍ ສ້າງ ໂໝດ ລູກ ໄດ້ ອີກ ຫລື  
ບໍ່ ຈະ ສັງເກດ ເຫັນ ເປັນ ການ ກະ ຈາຍ ໃນ ແນວ ເລີກ ເຮົາ  
ຈຶ່ງ ຮຽກວ່າ Depth-First Search



# Algorithm: Depth-First Search

Algorithm Depth-First Search

1. IF initial state = goal state THEN quit และ ถ้า success  
ELSE UNTIL success หรือ failure DO

Recursive

1.1 สังหา successor (ให้ ฉี ว่า E) ของ initial state โดย Operator

IF บໍ່ มີ successor THEN ถິນ ຄ່າ failure

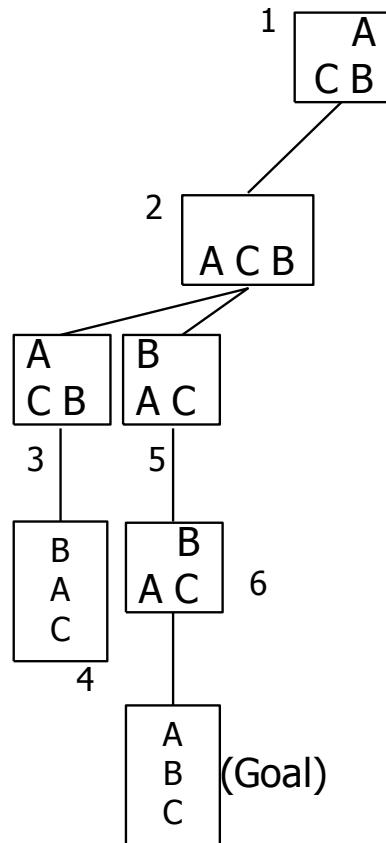
1.2 เอັ້ນ Depth-First Search โดย ໃຫ້ E ເປັນ initial state

1.3 IF ມີ ການ ຄືນ ຄ່າ ຂອງ success THEN ຄືນ ຄ່າ success

ELSE ແຮດ ຊື້ ລູບ ນີ້

Recur  
sive

# Depth-First Search (sample)



ປະເມີນ Depth-First Search

Completeness: \_\_\_\_\_

Optimality: \_\_\_\_\_

Time Complexity:  $O(bm)$        $b = 2$  ຂະໜາດ

ຂອງ ໂ້ານດ ໃນ ແຕ່ ລະ level;  $m = \text{ຄວາມ} \ \text{ສູງ} \ \text{ຂອງ} \ \text{tree}$

Space Complexity  $O(bd)$

# ปรับหูบละหว่าง Breadth-First Search และ Depth-First Search

## ຂໍ້ ດີ ຂອງ Breadth-First Search

- ຈະ ບໍ່ ຕິດ path ທີ່ ເລີກ ຫລາຍ ແລ້ວ ບໍ່ ຜົບ ຄໍາ ຕອບ
- ຖ້າ ມີ ຄໍາ ຕອບ ຮັບຮອງ ວ່າ ຕ້ອງ ຜົບ ຢ່າງ ແມ່ນອນ

## ຂໍ້ ດີ ຂອງ Depth-First Search

- ໃຊ້ ຫນ່ວຍ ຄວາມ ຈຳ ນ້ອຍ ກວ່າ Breadth-First Search ແພະ ໃນ current path ເກົ່າ ນັ້ນ ທີ່ ຖືກ ເກັບ
- ຖ້າ ໂຊ ກຳ Depth-First Search ຈະ ຜົບ ຄໍາ ຕອບ ໂດຍ ບໍ່ ຕ້ອງ ຄົ້ນ ຫາ space ຫລາຍ ເກີນ ຄວາມ ຈຳ ເປັນ ແຕ່ ຖ້າ ເປັນ Breadth-First Search states ທີ່ ຢູ່ໃນ ລະ ດັບ  $n$  ຈະ ຕ້ອງ ຄົ້ນ ຫາ ບືກ ກະ ຈາຍ ອອກ ມາ ໃຫ້ ຫົມືດ ກ່ອນທີ່ຈະ ໄປ ກະ ຈາຍ ໂຮມດ ໃນ ລະ ດັບ  $n+1$

# 4. Heuristic Search / Informed Search

- ▶ Hill-Climbing
- ▶ Best-First Search
- ▶ A\*

Heuristic Search เป็น เทคนิค ที่ ใช้ เพิ่ม ประสิทธิภาพ ของ กะ บวน การ Search โดย ย้อม ให้ ขาด ความ สิมบูรณ์ (Completeness) ถ้า คาด บ์ ผิด คำ ตอบ ภ ได้ หาก ความ รู้ ที่ ใช้ ใน การ แก้ บันຫາ บ์ สิมบูรณ์

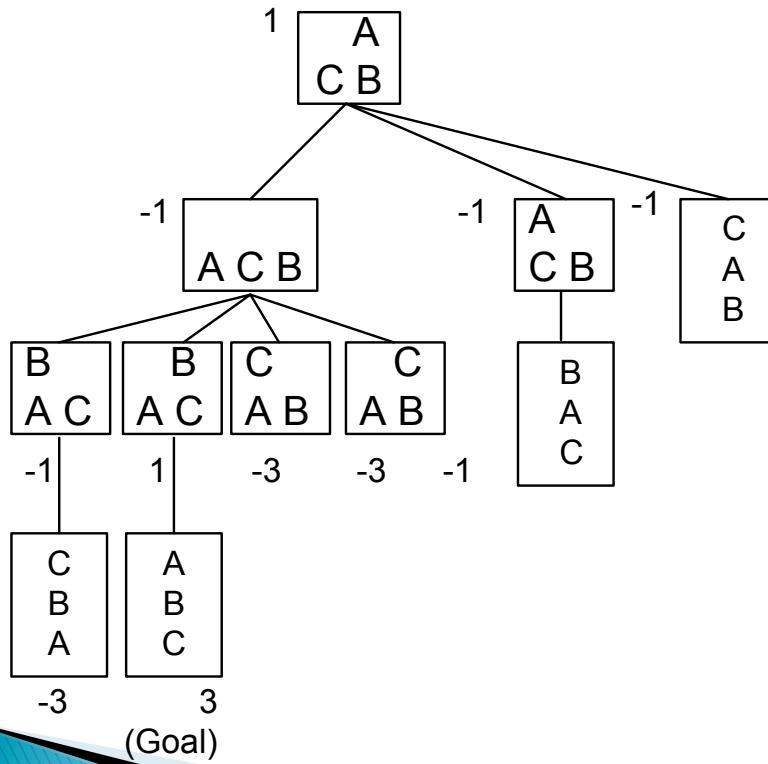
# ອີງປະກອບຂອງ Heuristic Search

ຖ້າ Heuristic Function ບໍ່ ດີ ກຳ ອາດ ໃຫ້ ການ ຄົ້ນ ຫາ ຜິດ ທຶດ ຫາງ  
ໄດ້

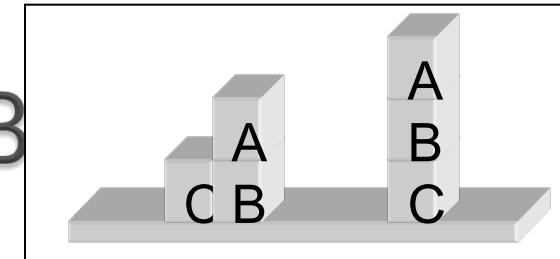
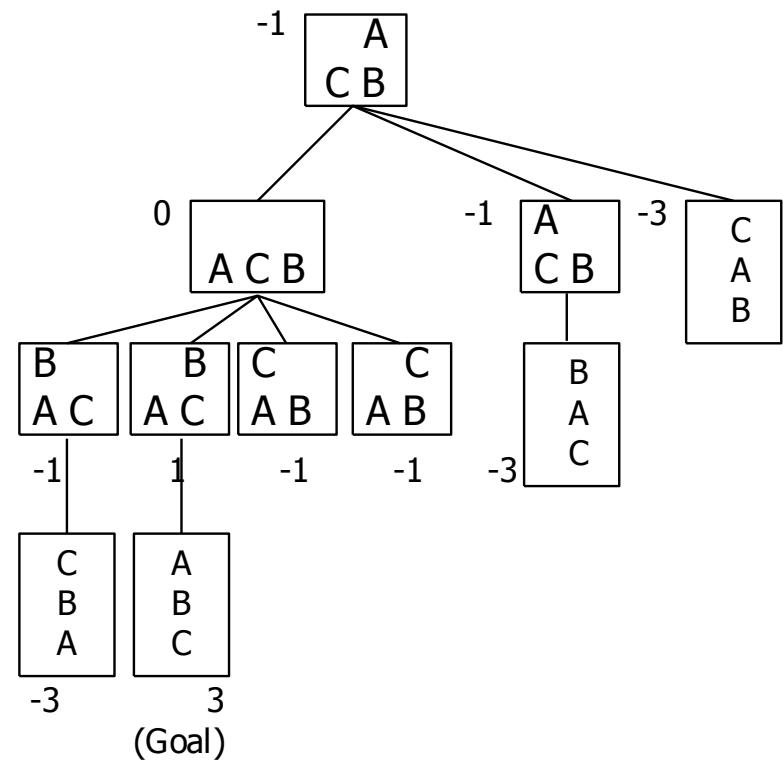
Heuristic Function ໃຊ້ ຄວາມ ຮູ່ ໃນ ໃນ ບັນຫາ ນັ້ນ ເພື່ອ ເປັນ ແລວທາງ ໃນ ການ ໃຫ້ ຄ່າ ຄວາມ ດີ ຂອງ ໂບນດ ຕ່າງໆ (ແຕ່ ເປັນຜຽງ ການ ຄາດ ເດືອນ ເທົ່າ ນັ້ນ)

# Sample: Heuristic Function: B World Problem

- ▶ h1: Heuristic Function ถ้า บวก 1 แต้ม ให้ กับ ทุก Block ที่ วาง ปู เហิ่ง สีง ที่ มัน ถอน ปู และ ลิบ หนึ่ง แต้ม ถ้า ขั้ม่น

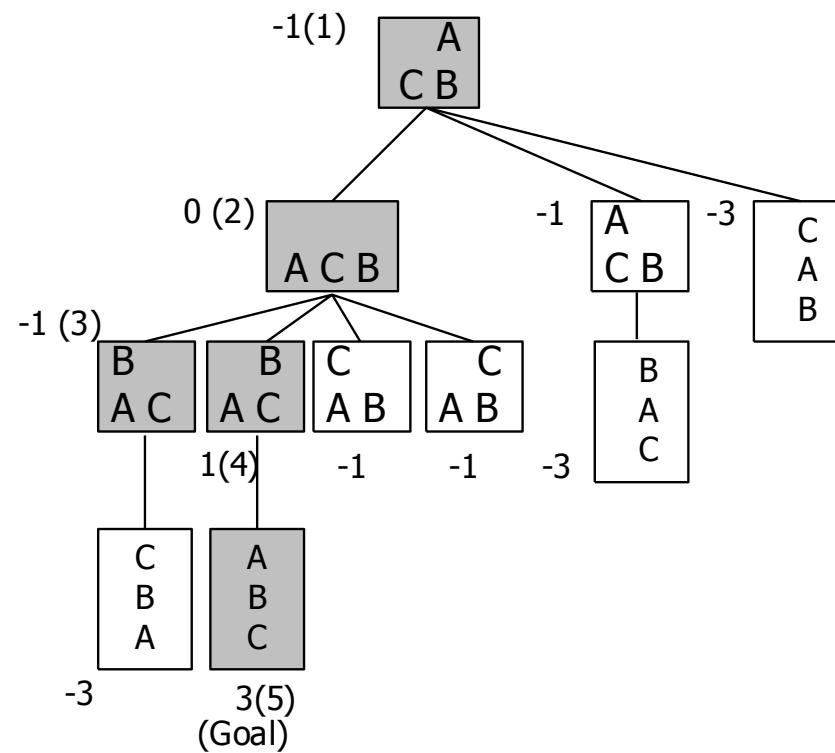
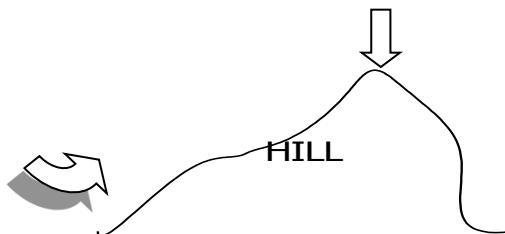


- ▶ h2: สำลับ ทุก blocks ที่ ปู เหิง โถง ส้าง ที่ ถึก บวก1 แต้ม ให้ กับ ทุก blocks ที่ ปู เหิง โถง ส้าง นั้น และ ลิบ 1 แต้ม สำลับ ทุก blocks ที่ ปูใน โถง ส้าง ที่ ผิด



# Hill Climbing

ອັນກໍລົງທຶນນີ້ຈະປຽບທຽບການປິນຜູ້ເຂົາໂດຍມີແນວຄືດວ່າໂຮນດໃໝ່ທີ່ຈະຖືກ generate ຈະຕ້ອງມີຄ່າ Heuristic ທີ່ດີກວ່າໂຮນດປະຈຸບັນ



# ALGORITHM: Hill Climbing

1. Evaluate initial state

IF initial state = goal state THEN តិច តា initial state និង quit

ELSE current state := initial state

2. UNTIL ធន់ goal state ទាំង ប៉ុណ្ណោះ មី operator ហើយ ប៉ុន current state  
ដែល DO

    2.1 ជួយ operator ហើយ ប៉ុណ្ណោះ ដែល ឱ្យ ការពិភាក្សានៃ current state ផ្តើម  
    ដែល new state

    2.2 Evaluate new state

        IF new state = goal state THEN តិច តា new state និង quit

        ELSE IF តា heuristic ខ្ពស់ new state ជូនវា

            THEN current state := new state

            ELSE IF តា heuristic ខ្ពស់ new state ប៉ុណ្ណោះ THEN នៅក្នុង ព័ត៌មាន  
(ប៉ុណ្ណោះ នៅក្នុង 2.1 ព័ត៌មាន សាច់ ធ្លាប់ ឱ្យ និង និង តាម និង និង និង)

# ລັກສະນະ ຂອງ Hill-Climbing

ອາດ ບໍ່ ຜົບ ຄໍາ ຕອບ ເນື້ອງ ຈາກ Heuristic Function ບໍ່ ດີ

❖ **Local Maximum:** ອາດ ໄດ້ path ທີ່ ດີກວ່າ ເສັ້ນ ທາງ ຮອບ ຂ້າງ ແຕ່ ບໍ່ແມ່ນ path ທີ່ ດີ ທີ່ສຸດ

❖ **Plateau:** ອາດ ຜົບ ກໍລະນີ Search space ເປັນ ຜຶ້ນ ທີ່ງຮາບ ສູງ ເຊັ່ນ ໃນ ກໍລະນີ ທີ່ ໂໝາດ ລູກ ທັງ ຫົດ ທີ່ ກໍາລັງ ຖືກ evaluate ມີ ຄ່າ ເທົ່າ ກັບ ໂໝາດ ຂອງ ຕົວ ມັນ ເອງ

❖ **Ridge:** ອາດ ຜົບ ກໍລະນີ Ridge (ສັນ ພູເຂົາ) ຄື ການ ເລືອກ Generate ໂໝາດ ທີ່ ໃຫ້ ຄ່າ Heuristic ທີ່ ດີ ຄ່າ heuristic ນີ້ ຖືກ ຄາດ ຫວັງ ວ່າ ຈະ ນຳ ພາ ໄປ ສູ່ ຄໍາ ຕອບ ຂອງ ການ ແກ້ ບັນຫາ ເມື່ອ ສ້າງ ໂໝາດ ຕໍ່ໄປ ເລືອຍໆ ເຖິງ ຈຸດ ຫົ່ງ ປາກິດ ວ່າ ບໍ່ ມີ ຄ່າ ທີ່ ດີກວ່າ ອີກ ຕໍ່ໄປ ແລະ ຂະນະ ນັ້ນ ກໍ ຢັ້ງ ບໍ່ ຜົບ ຄໍາ ຕອບ ດ້ວຍ ດັ່ງ ນັ້ນ ຈຶ່ງ ເປັນຜຽງ ແລະ ການ ຢ່າງ ໄປ ໃນ ທາງ ທີ່ ຄາດ ວ່າ ຈະ ດີ ແຕ່ ບໍ່ ຜົບ ເປົ້າ ພາຍ ອາດ ຕັ້ງ ນັ້ນ ໄດ້ ໂດຍ ການ ສມ ເລືອກ

# Best-First Search

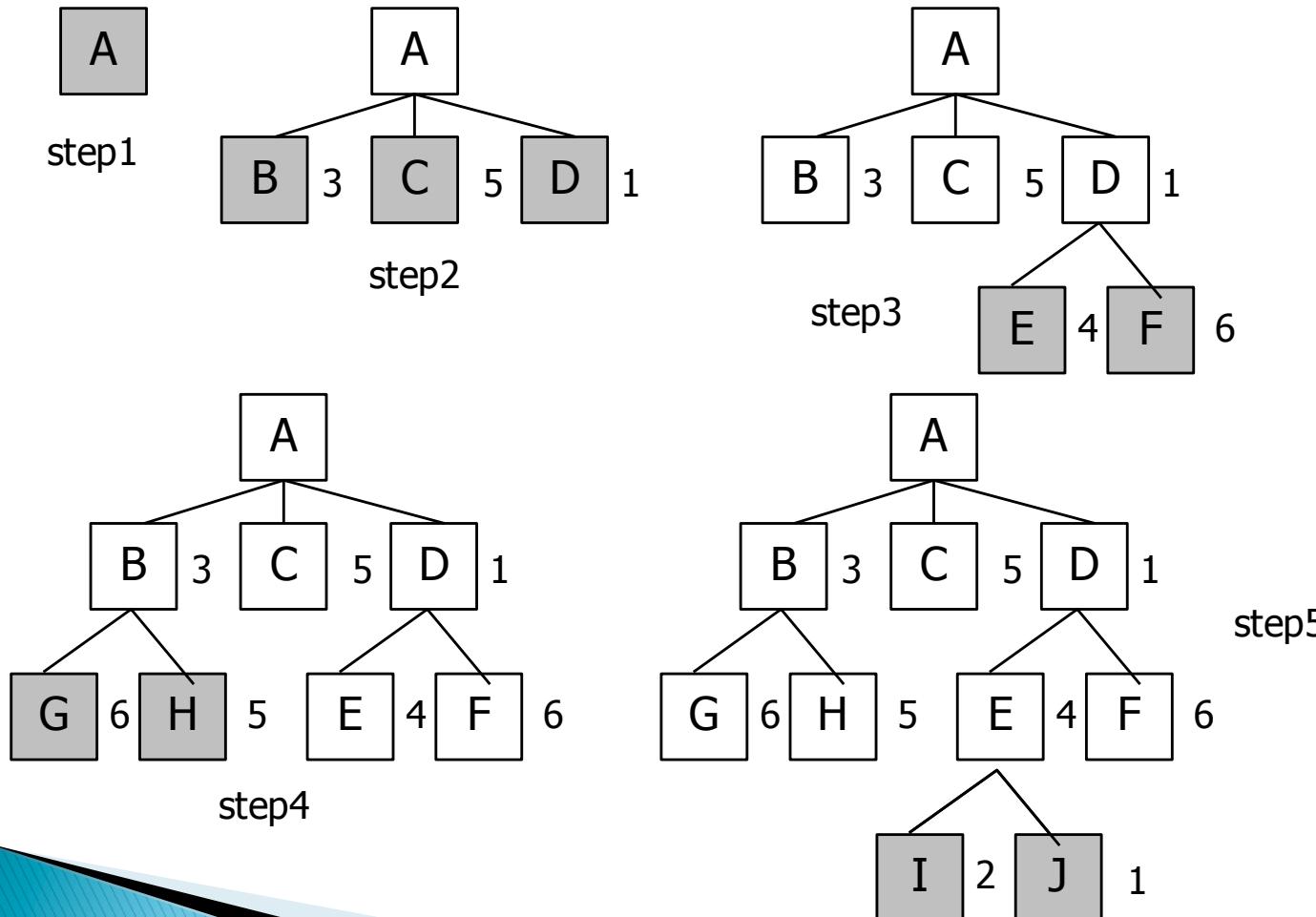
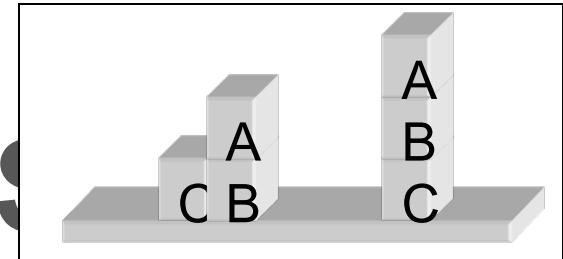
- ການເຮັດວຽກ ຄ້າຍ ຄືກັບ Hill-climbing ໂດຍ ມີ ຂໍ ແຕກ ຕ່າງ ຄື  
ໃນ ກໍລະນີ ຂອງ Hill-climbing ຈະ ເລືອກ state ທີ່ ດີ ຫີ້ສຸດ ແລະ ເດີນທາງ  
ໄປ ທາງ ນັ້ນ ໂດຍ ທີ່ states ອື່ນ ຖໍ່ ທຶກ ສ້າງ ຈະ ຈາກ parent state  
ດຽວ ກັນ ຈະ ທຶກ ຖົມ ໄປ ແຕ່ ໃນ ກໍລະນີ ຂອງ Best-First Search ຈະ  
ເກັບ ຄ່າ states ໜ້າ ນີ້ ໄວ໌ ເຜື່ອ ໃຊ້ ໃນ ອະນາຄົດ ເມື່ອ path ທີ່ ຢ່າງ  
ໄປ ບໍ່ ດີ ເທົ່າ ກັບ states ໜ້າ ນີ້
- Hill-climbing ອາດຈະ ຢຸດ ເມື່ອ ບໍ່ ຜົບ ຄ່າ ທີ່ ດີ ຂຶ້ນ (ກໍລະນີ ທີ່ ບໍ່  
ເລືອກ ສຸມ ໂຫມດ ອື່ນ ອີກ) ແຕ່ Best-First Search, state ທີ່ ມີ ຄ່າ  
Heuristic ທີ່ ດີ ຫີ້ສຸດ ໃນ ປະຈຸບັນ ຈະ ທຶກ ເລືອກ ເຖິງວ່າ ຈະ ບໍ່ ມີ ຄ່າ  
ບໍ່ ດີ ເທົ່າ states ທີ່ ເຄີຍ ເລືອກ ມາ ກໍ ຕາມ

# ALGORITHM: Best-First Search

# Best-First Search Algorithm

1. OPEN = {initial state}
  2. UNTIL (ដិប goal state មាន ឬ មិនមែន OPEN) DO
    - 2.1 ជួយ state ទាំងអស់ទាំងអស់ដែលមានតម្លៃការងារតូចប៉ុណ្ណោះ
    - 2.2 សៅរ៍ ឯកសារ តាមតម្លៃការងារដែលត្រូវបានដាក់
    - 2.3 សំឡើរ ឯកសារ ដោយ តើត្រូវ  
    2.3.1 IF ឯកសារ មានតម្លៃការងារ ឬ មិនមែន  
        ពីរ ឯកសារ ដែលមានតម្លៃការងារតូចប៉ុណ្ណោះ នឹងត្រូវបានដាក់  
        ពីរ ឯកសារ ដែលមានតម្លៃការងារតូចប៉ុណ្ណោះ នឹងត្រូវបានដាក់  
    2.3.2 IF ឯកសារ មានតម្លៃការងារតូចប៉ុណ្ណោះ នឹងត្រូវបានដាក់  
        ពីរ ឯកសារ ដែលមានតម្លៃការងារតូចប៉ុណ្ណោះ នឹងត្រូវបានដាក់

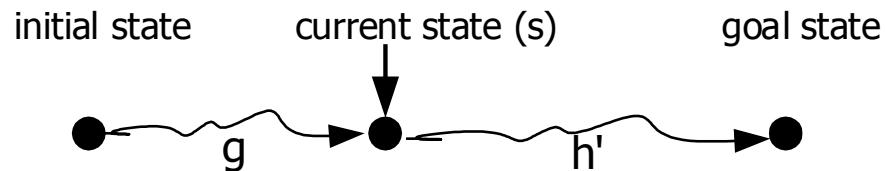
# Sample: Best-First Search



# A\* Algorithm (A-Star Algorithm)

- ▶ - Best-First Search เป็น algorithm ຢ່າງ ງ່າຍ ຂອງ A\*
- ▶ - Heuristic Function ຂອງ BFS ດີ F (s) = h (s) ດີ ຄວາມ ດີ ຂອງ state ປະຈຸບັນ ໄປ ຍັງ goal state
- ▶ -Heuristic Function ຂອງ A\*: ປະສິມ ເຊິ້າ ຄ່າ ຄວາມ ດີ ຂອງ ເສັ້ນ ທາງ ຈາກ initial state ມາຍັງ current state ແລະ ຈາກ current state ໄປ ຍັງ goal state ດ້ວຍ  $F' (s) = g (s) + h' (s)$
- ▶

A\* ແຕກ ຕ່າງ ຈາກ Best-First Search ການ ໃຊ້ ຜັງກໍ ຊັນ  $g$  ທີ່ ບໍ່ ພຽງ ແຕ່ ປະມານ ຄ່າ ຈາກ ຈຸດ ປະຈຸບັນ ໄປ ຍັງ ເປົ້າ ຫມາຍ ທາກ ຍັງ ເພີ່ມ ການ ປະມານ ຄ່າ ຈາກ ຈຸດ ເລີ່ມ ຕົ້ນ ມາຍັງ ຈຸດ ປະຈຸບັນ ດ້ວຍ ໂດຍ ການ ໃຊ້  $g ()$  state ທີ່ ປະມານ ວ່າ ໃກ້ goal state ຫລາຍ ທີ່ສຸດ ອາດຈະ ບໍ່ ຖືກ ກະ ຈາຍ ກໍ ໄດ້

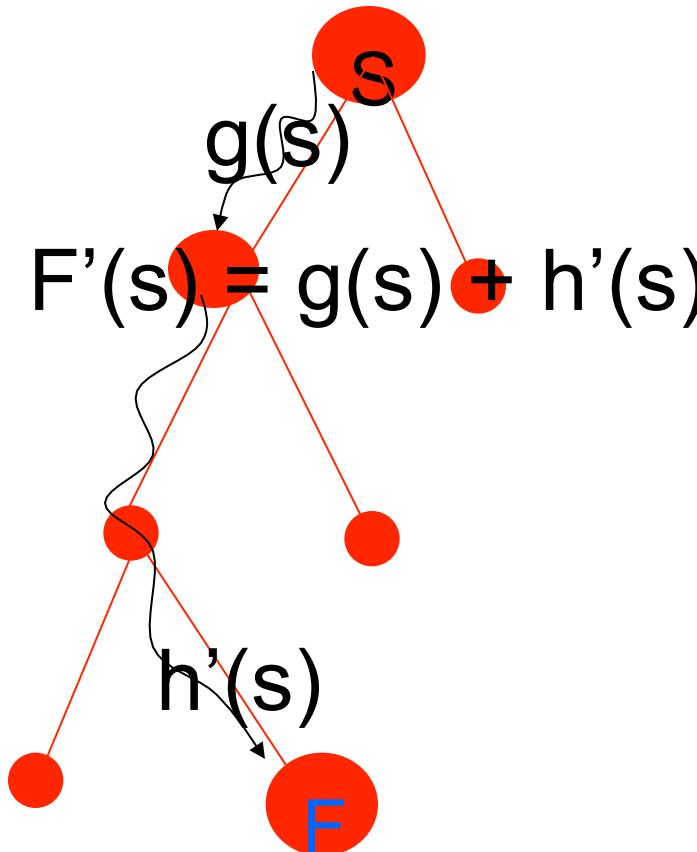


$$f'(s) = g(s) + h'(s)$$

ໂດຍ ທີ່  $g$  ຄື ຜັງກໍ ຊັນ ທີ່ ຄືດໄລ່ ຄ່າ cost ຈາກ initial state ເຖິງ current state

$h'$  ຄື ຜັງກໍ ຊັນ ທີ່ ປະ ມານ  
(estimate) ຄ່າcost ຈາກ current state ເຖິງ goal state

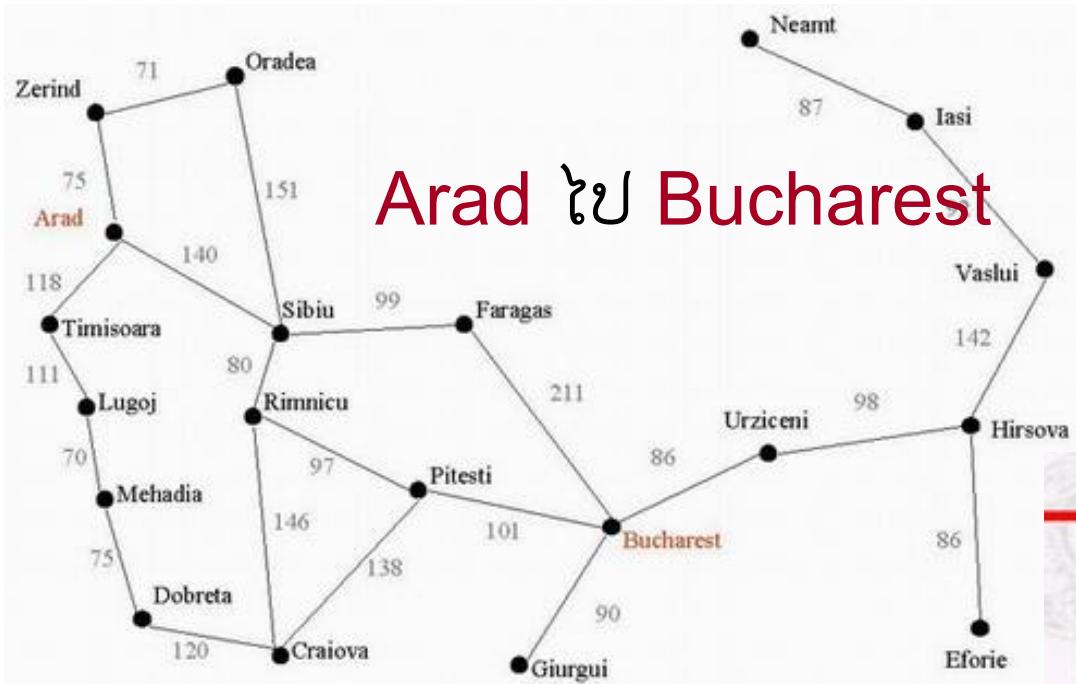
$f'$  ຄື ຜັງກໍ ຊັນ ທີ່ ປະ ມານ ຄ່າ cost ຈາກ initial state ເຖິງ goal state  
(state ທີ່ ດີ ຈະ ມີ ຄ່າ  $f'$  ນ້ອຍ)



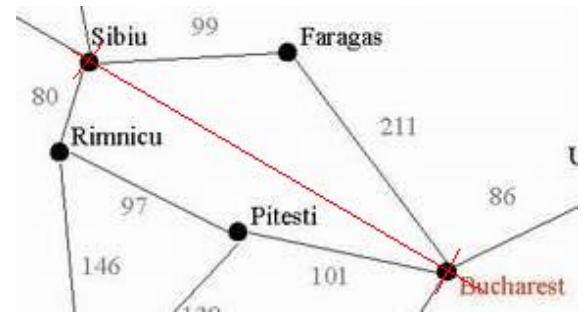
## A\* มีถุนสิมบัณฑ์

1.  $h'(s) = 0$  เมื่อ  $s$  คือ goal state
2. ถ้า  $h'$  เป็น ฝังก์ ดั้ง ที่ ถ้า cost ที่ บวก เกิน ถ้า cost ของ ฝังก์ ดั้ง  $h$  ( $h$  คือ ฝังก์ ดั้ง ที่ ถ้า cost จาก current state เก็บ goal state โดย ที่ ถ้า cost ที่ ถ้า มา มั่น เป็น ถ้า ที่ มั่อย ที่สุด) เริ่ จะ รับรอง ได้ ว่า path ที่ ได้ จาก ฝังก์ ดั้ง  $f'$  () มั่น จะ เป็น Optimal path ละเหรอ

ຕិវាយការងារដែលបានធ្វើឡើងនៅមីនុយធមិត្តបណ្តុះបណ្តុះ



**Arad នៃ Bucharest**



**Straight-line  
Distance**

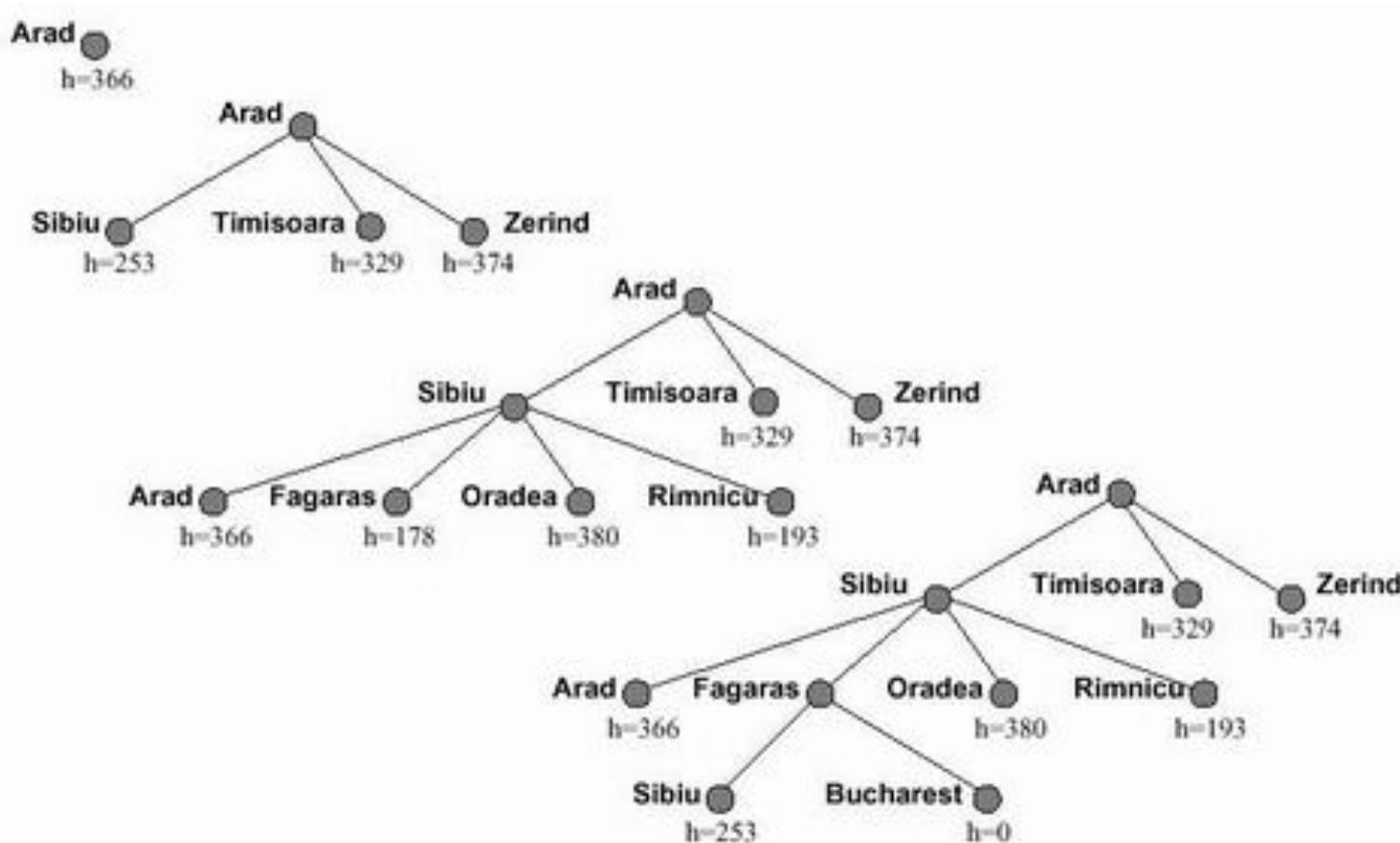
**Straight Line Distances to Bucharest**

Town	SLD
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244

Town	SLD
Mehadi	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

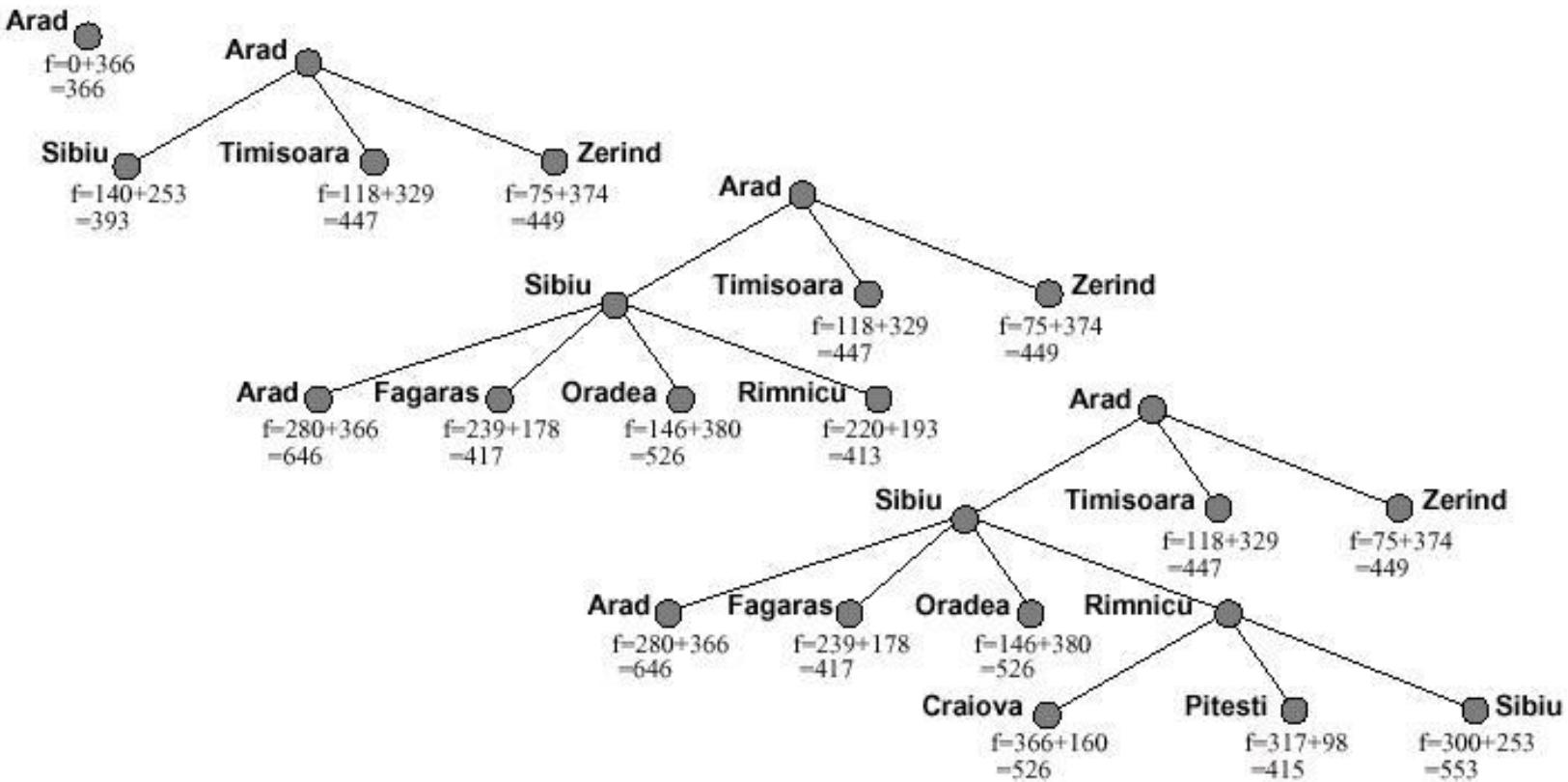
ការ រៀបទាំង វា  $h'$  ( $n$ ) និង បំ ពេញ  
តាតា  $h$  ( $n$ ) និង ពី នី និង ខ្លួន  
**Straight-line Distance** តិច បាន  
ក្នុង ពី បាន ខ្លួន

# ຂັ້ນຕອນການ-ເຮັດ-ວຽກງານຂອງ A\* ເລີມຈາກຈຸດເລີມຕົນຂອງເມືອງຄີ Arad



ຝາບ ສະແດງ ການ ກະ ຈາຍ ໂທມດ ຜ້ອມ ລະ ຍະ ທາງ ກົງ ດ້ວຍ A\* Algorithm

ການ ກະ ຈາຍ ໂ້ານດ ເຟື່ອ ເລື່ອກ ການເດີນທາງ ຈາກ ເມື່ງ Arad ໃປ Bucharest  
 $A^*$  ຕີ່  $f(s) = g(s) + h'(s)$



ຕໍ່ ຈາກ ນີ້ Pitesti ແລ້ວ Fagaras ແລ້ວ ກໍ Bucharest

Heuristic function ຂອງ A\* ຕີ່  $f(s) = g(s) + h'(s)$  ເມື່ອ  $g$  ຕີ່ ລະ ຍາງ ທີ່ ຢ່າງ  
ຜ່ານແນວ ແລະ  $h'(n)$  ຕີ່ ລະ ຍາງ ກົງ ທີ່ ປະ ມານ ໄດ້ ເຮັດໃຫ້ ໄດ້ ລັກສະນະ ການ ກະ ຈາຍ  
ໂຂມດ ດັ່ງ ຂ່າງ ຕົ້ນ

## 5. ภาระผู้เรียนทักษะการทดลองเล่นเกม (Game Playing)

- ▶ Minimax Algorithm
- ▶ Alpha-Beta Cutoffs Algorithm