CS1010S Tutorial 7

Wang Zexin

National University of Singapore
Year 2 QF

wang.zexin@u.nus.edu

March 19, 2017

Today's Agenda

1. Recap
2. Question One
   - Discussion
3. Question 2
   - Standard solutions
4. Question 3
5. Question 4
6. Question 5
7. Extra stuff: One final question

Recap - Sorting

- Go VisualAlgo when you forget how they operate!
- Pay attention to the first four below!
  - Selection sort
  - Bubble sort
  - Insertion sort
  - Merge sort
  - Quick sort
  - Shell sort
  - Bucket sort
  - Heap sort

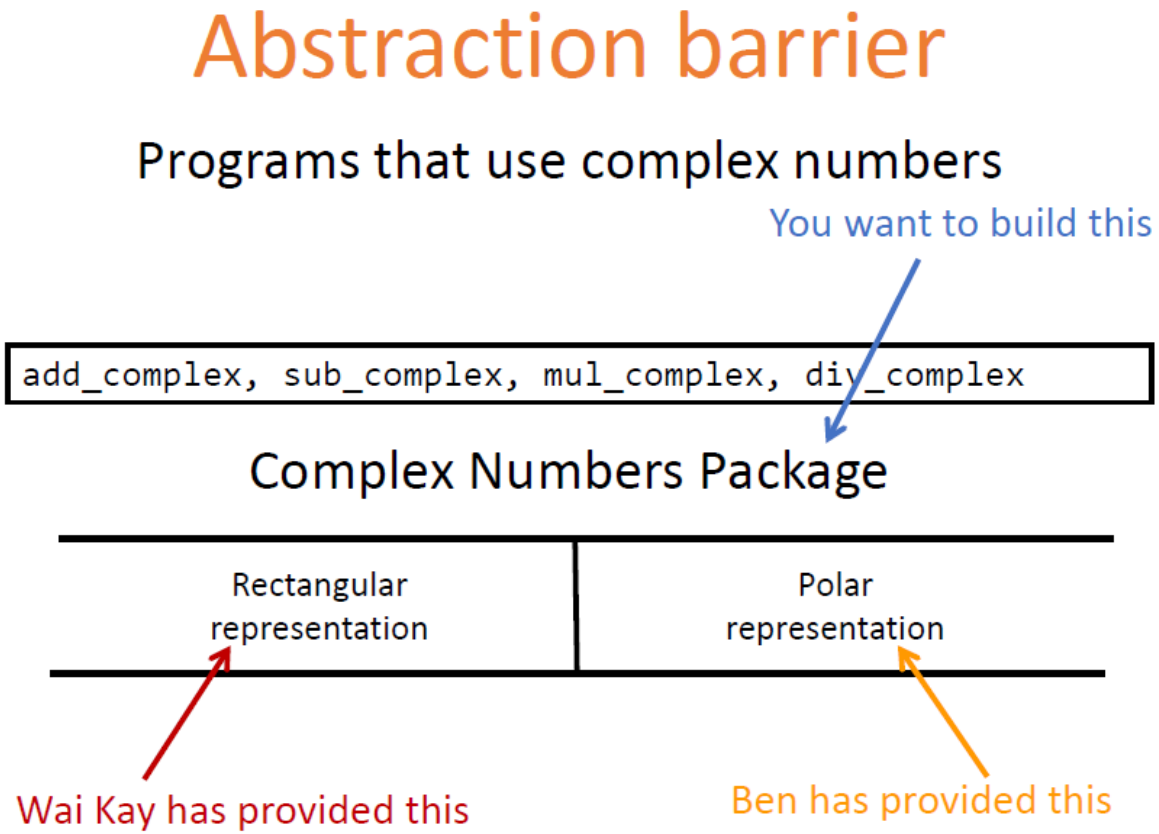
Recap - Implementation of data structures

What is a data structure?

- A collection of some **objectised** things!
- Have similar functionality and constitution!
- Same functions to operate on them!
- For example: GameState, Stack, ComplexNumber, Sets, Queue(later)

Recap - Abstraction Barrier

- May not be so important as you will not be fully tested on this
- Important for your mission 11
- The essence is to accept different implementation for the same objects



Question 1

Your task is to make use of *accumulate* to implement a function *accumulate_n* which $\forall i \in [1..n]$, accumulate all the ith element of each sequence together and form a new sequence.

```
def accumulate(op, init, seq):
    if not seq:
        return init
    else:
        return op(seq[0], accumulate(op, init, seq[1:]))
```

Question 1 Discussion

Your task as per tutorial sheet is to fill in the missing expressions $< T1 >$ and $< T2 >$ in the definition of *accumulate_n*, but aim higher!

- First of all, you are supposed to understand how *accumulate* works.
- It simply takes the elements in *sequences* one by one.
- And apply *op* on them!
- Also note the order of execution of *op*

```
def accumulate(op, init, seq):
    if not seq:
        return init
    else:
        return op(seq[0], accumulate(op, init, seq[1:]))
```

Question 1 Discussion

How to apply *accumulate* onto all the *i*th elements of each sequence?

- Remember what we did for the matrix question last tutorial?
- Try *map* or list comprehension!

```
def accumulate(op, init, seq):
    if not seq:
        return init
    else:
        return op(seq[0], accumulate(op, init, seq[1:]))
```

Question 1 Zexin's solutions

```
def accumulate_n(op, init, sequences):
    if (not sequences) or (not sequences[0]):
        return type(sequences)()
    else:
        return ( [accumulate(op, init, [x[0] for x in sequences])]
            + accumulate_n(op, init, [x[1:] for x in sequences]))
```

Try this on your matrices!

Question 2 Part A

Re-define the function *col_sum* using *accumulate_n*. The function should still take in a matrix and return a list, where the *i*-th element in the list is the sum of the elements in the *i*-th column of the matrix.

Question 2 Part A Discussion

Your task now is to:

- Loop from the left most column to the right most one.
- Extract all the elements from the matrix in each column.
- Sum these elements up and add it to the list
- Note that you should be using *accumulate_n* to do that now.

Question 2 Part B

Re-define the function *row_sum* using *accumulate_n*. The function should still take in a matrix and return a list, where the i-th element is the sum of the elements in the i-th row of the matrix.

Question 2 Part B Discussion

Your task now is to:

- Loop from the upmost row to the bottom most one.
- Extract all the elements from the matrix in each column.
- Sum these elements up and add it to the list.
- Note that you should be using *accumulate_n* and *transpose* to do so

Question 2 Zexin's solution

```
def col_sum(matrix):
    return accumulate_n(lambda x,y:x+y, 0, matrix)

def row_sum(matrix):
    return accumulate_n(lambda x,y:x+y, 0, transpose(matrix))
```

Question 3 Part A

Write a function *count_sentence* that takes a sentence representation (as described above) and returns a series list with two elements: the number of words in the sentence, and the number of letters in the sentence.

Assume that spaces count as 1 letter per space, and that there is exactly 1 space between each word (but none at the start or end of the sentence).

What is the order of growth in time and space (in terms of the number of letters in the sentence) of the function that you wrote?

Question 3 Part A Discussion

Your task now is to:

- Check the number of words
- Check the number of letters in each word
- Calculate the number of letters in each sentence
- Do you need to loop through all the letters?
- Then what is the time complexity?
- Do you need to build any other objects other than the lists?
- Then what is the space complexity?

Question 3 Part A Zexin's Solution

```
def count_sentence(sentence):
    noWords = len(sentence)
    noLetters = accumulate(lambda x, y: len(x) + y, 0, sentence)
    return (noWords, noLetters + noWords - 1)
```

Question 3 Part B

Write a function *letter_count* that takes a sentence and returns a list of lists, where you have one list for each distinct letter in the sentence and each list has two elements.

The first element of the list pair is the letter and the second element is the count for the letter. The order of the list pairs does not matter.

What is the order of growth in time and space (in terms of the number of letters in the sentence) of the function that you wrote?

Question 3 Part B Discussion

Your task now is to:

- Create a counter list for the letters
- Check the letters in each word
- Add the letters into counter list
- Do you need to loop through all the letters?
- Then what is the time complexity?
- Do you need to build any other objects other than the lists?
- Then what is the space complexity?

Question 3 Part B Zexin's Solution

```
def letter_count(sentence):
    counter_list = []
    letters = accumulate(lambda x,y: x+y, [], sentence)
    for letter in letters:
        flag = True
        for counter in counter_list:
            if letter == counter[0]:
                counter[1] += 1
                flag = False
        if flag:
            counter_list.append([letter, 1])
    return counter_list
```

Question 3 Part C

Write a function *most_frequent_letters* that takes a sentence representation and returns a list of letters that occur most frequently in the given sentence.

The order of the letters does not matter. If only one such letter exists, then return a list with one element. If the sentence is empty, return an empty list.

What is the order of growth in time and space (in terms of the number of letters in the sentence) of the function that you wrote?

Question 3 Part C Discussion

Your task now is to:

- Create a counter list for the letters
- Check the letters in each word
- Add the letters into counter list
- Check the maximum frequency of letter
- Extract those with maximum frequency out
- Do you need to loop through all the letters?
- Then what is the time complexity?
- Do you need to build any other objects other than the lists?
- Then what is the space complexity?

Question 3 Part C Zexin's Solution

```
def most_frequent_letters(sentence):
    letterCounter = letter_count(sentence)
    maxFrequency = max([x[1] for x in letterCounter])
    return list(map(lambda x:x[0], filter(lambda x:x[1] == maxFrequ
```

Make use of *filter* and *map* to save the trouble of writing loops.
Make use of previously defined function to practice abstraction!

Question 4

Implement a **queue** data structure. (FIFO)

Question 4 Discussion

How to tackle this?

- The structure must be **mutable**.
- Tuple is not an option.
- Only list is left.

Question 4 Zexin's solution

```
def make_queue():
    return []

def enqueue(q, item):
    q.append(item)

def dequeue(q):
    return q.pop(0)

def size(q):
    return len(q)
```

Question 5

Write a function *who_wins* that will take an integer m and a list of players and return the last $m - 1$ players in the game.

Question 5 Discussion

How to tackle this?

- Use a queue as per the hint
- Note that you will have to convert the list of names into queue.
- How do you do that?
- Also you will have to rotate the queue.
- How do you do that? - dequeue and enqueue

Question 5 Zexin's solution

```
def who_wins(m, players):
    q = make_queue()
    for player in players:
        enqueue(q, player)
    while size(q) >= m:
        for i in range(m):
            player = dequeue(q)
            enqueue(q, player)
        dequeue(q)
    result = []
    while size(q):
        result.append(dequeue(q))
    return result
```

Extra stuff: One final question

If time permits, we will go through this.

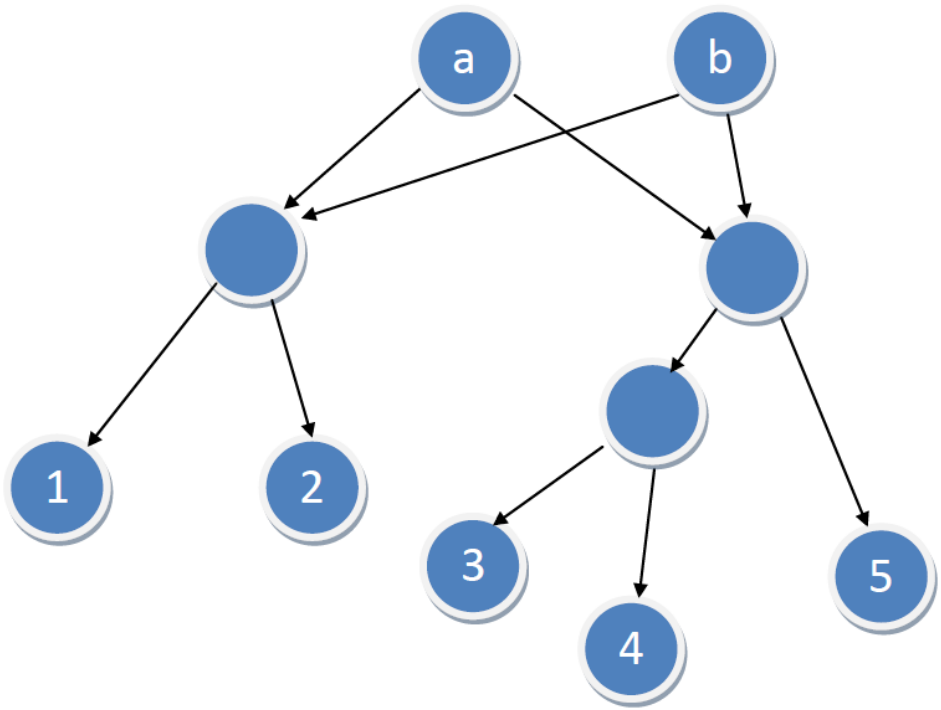
```
a = [[1, 2], [[3, 4], 5]]
b = a.copy()
a[0][1], b[1][0][0] = b[1][0], a[1][1]
print(a)
print(b)
```

- This is a problem in which you need to use box-and-pointer diagram.

Extra stuff: One final question

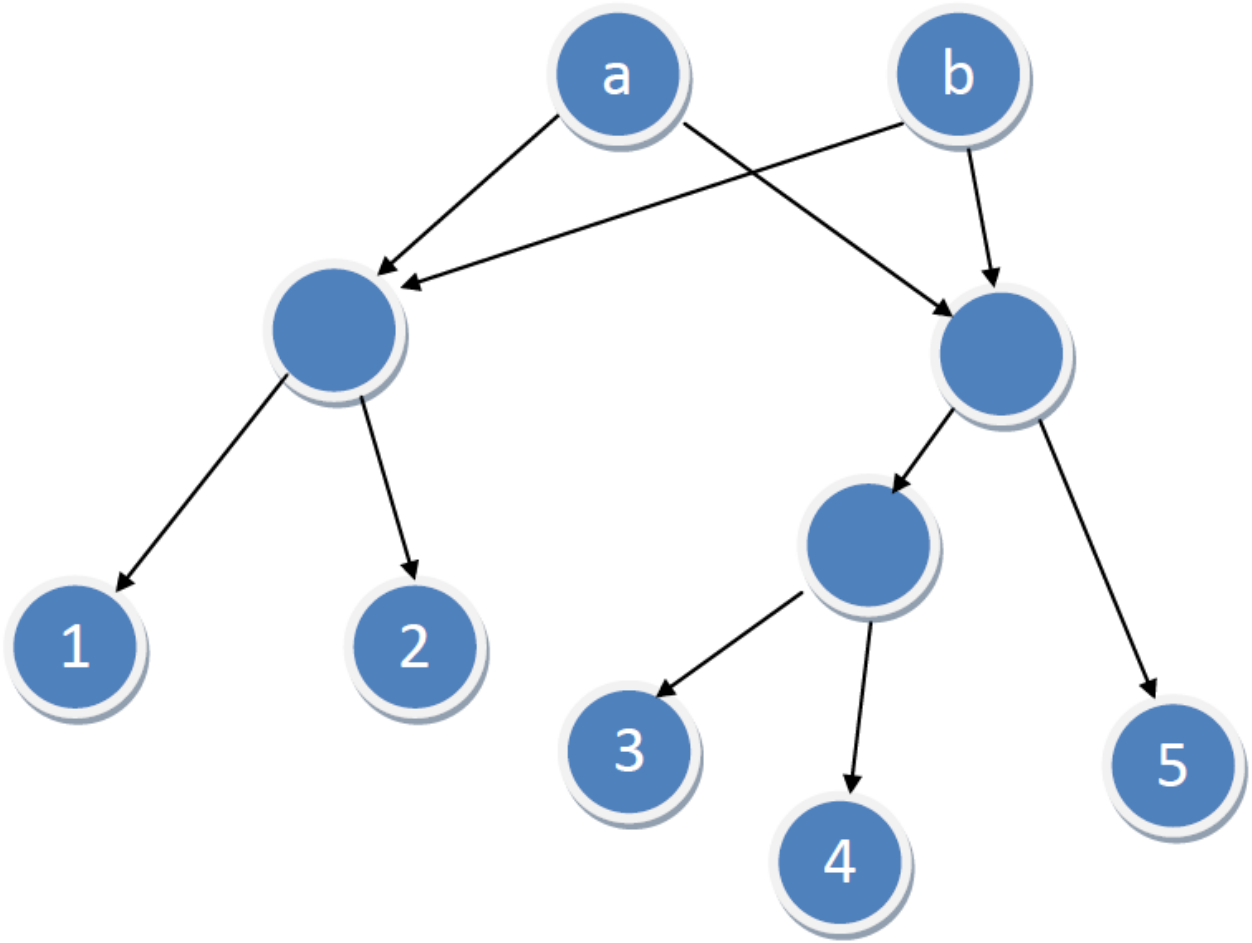
```
a = [[1, 2], [[3, 4], 5]]
b = a.copy()
a[0][1], b[1][0][0] = b[1][0], a[1][1]
print(a)
print(b)
```

Craft out the box and pointer diagram



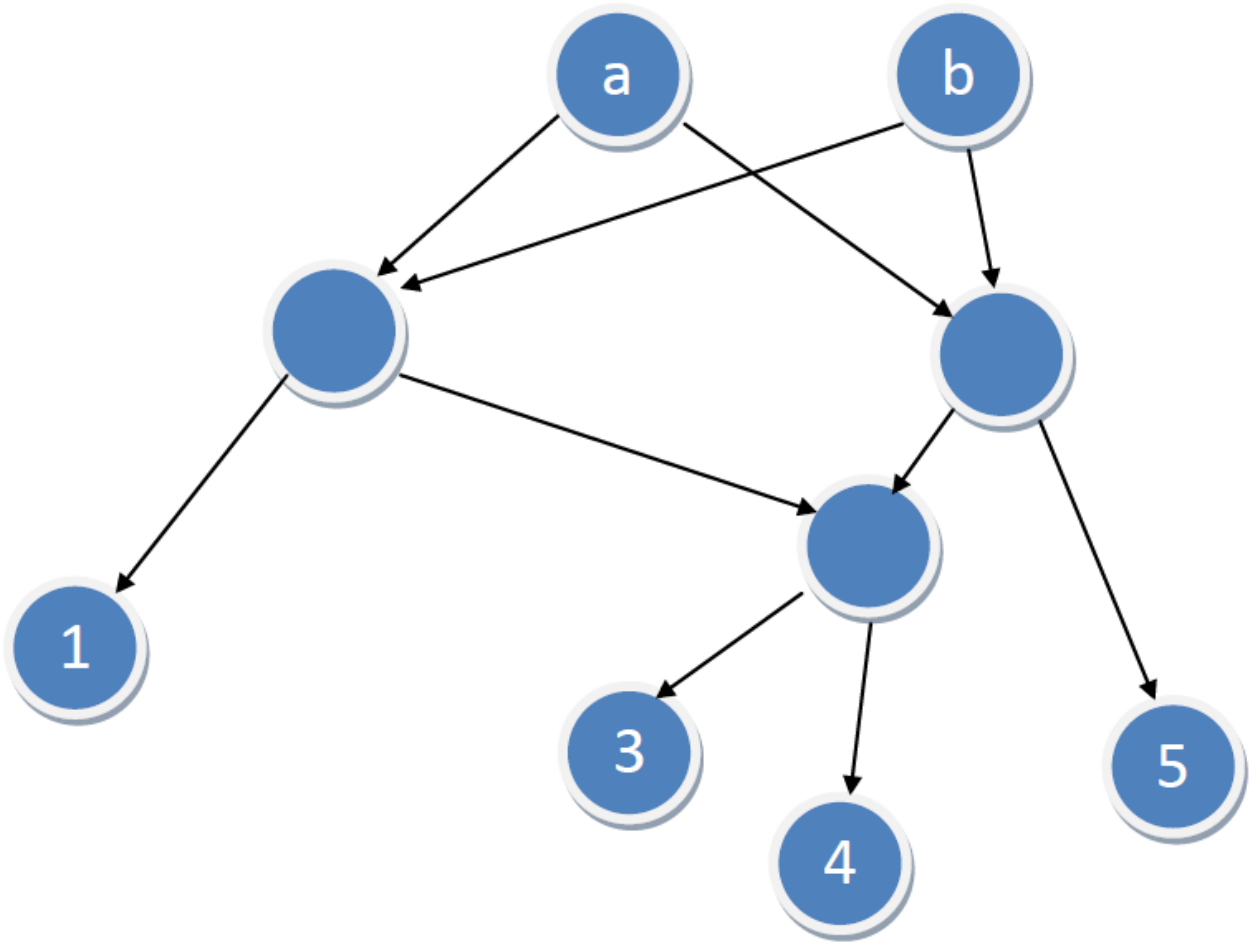
Extra stuff: One final question

Start



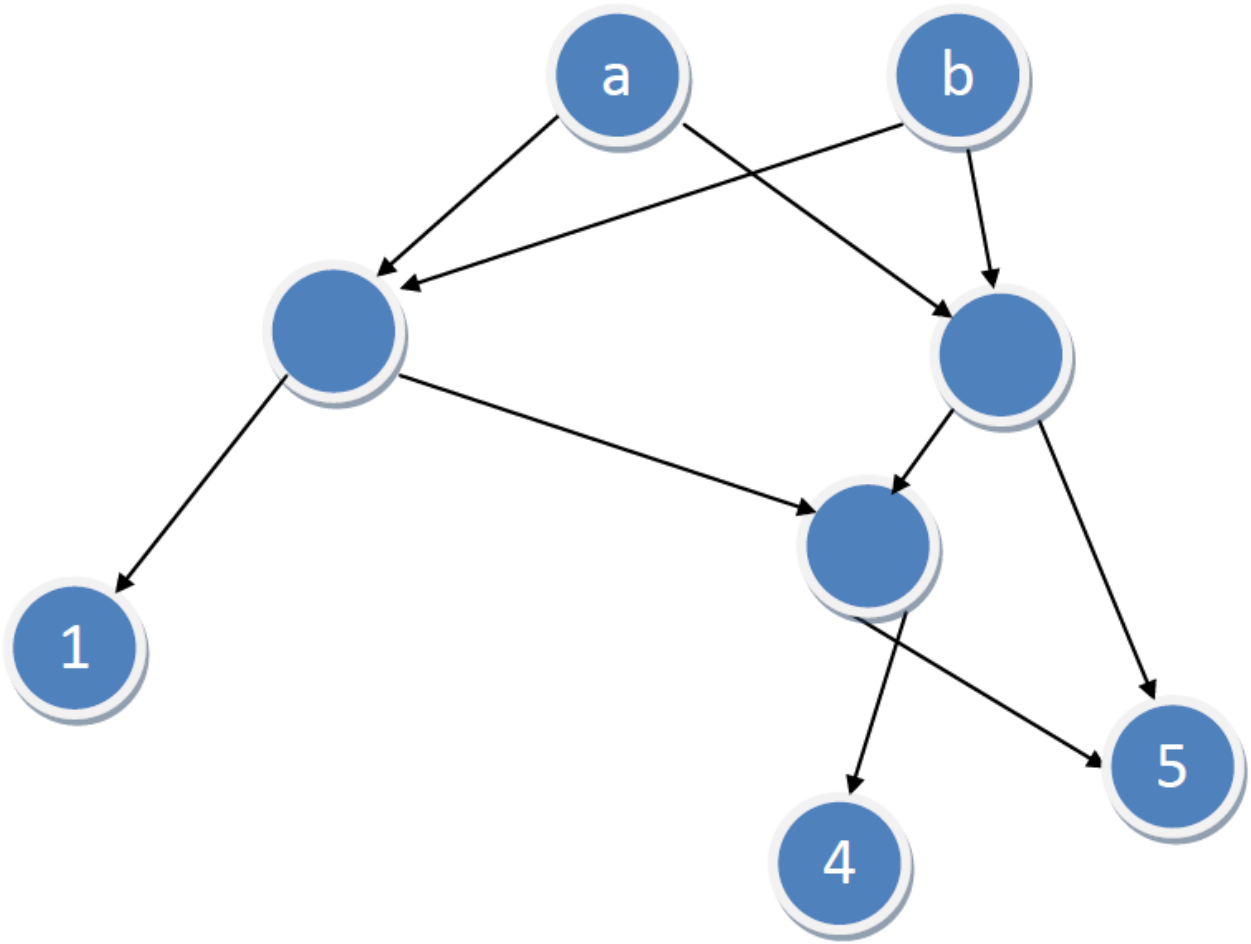
Extra stuff: One final question

$$a[0][1] = b[1][0]$$

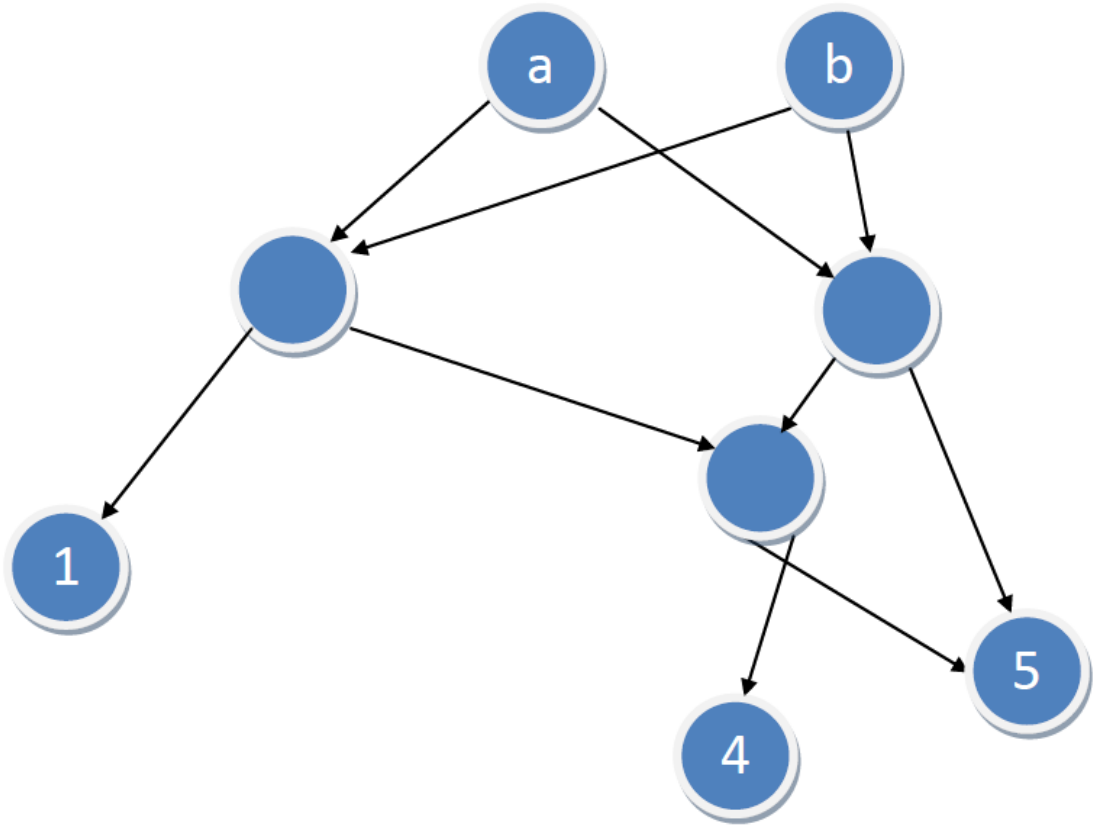


Extra stuff: One final question

$$b[1][0][0] = a[1][1]$$



Extra stuff: One final question



Hence the solution is:

$$a = [[1, [5, 4]], [[5, 4], 5]]$$
$$b = [[1, [5, 4]], [[5, 4], 5]]$$

Feedback & more

- Slides + relevant material available at:

  `https://github.com/wangzexin/Teaching`

- After the tutorial, if you have further questions:

  `wang.zexin@u.nus.edu`

Thank You

wang.zexin@u.nus.edu