MONASH University
Information Technology

6th and 8th May 2015
Monash University – Sunway Campus, Malaysia

FIT 2004
Algorithms and Data Structures

Tutorial/ Practical 09

Ian Lim
lim.wern.han@monash.edu

Tutorial09 Task01

- Pattern Matching with BWT

Tutorial09 Task01

- Pattern Matching with BWT
- Burrows–Wheeler transform (BWT)
  - All of you should be familiar by now?
  - How to convert a String to its BWT(String)
  - How to convert a BWT(String) back to String with inverseBWT(BWT(String))

Tutorial09 Task01

- So now, pattern matching with BWT
- Why?

Tutorial09 Task01

- So now, pattern matching with BWT
- Application of pattern matching

Tutorial09 Task01

- So now, pattern matching with BWT
- Application of pattern matching
  - Searches
  - Spell checks
  - Any many more

Tutorial09 Task01

- So now, pattern matching with BWT
- Application of pattern matching
- Why use BWT for pattern matching?

Tutorial09 Task01

- So now, pattern matching with BWT
- Application of pattern matching
- Why use BWT for pattern matching?
  - Very very good complexity

Tutorial09 Task01

- Convert String in question to BWT

Tutorial09 Task01

- Convert String in question to BWT
  - Write all cyclic string
  - Sort the cyclic string
  - Get the last column

Tutorial09 Task01

- Convert String in question to BWT

A: sort permutations

	Suffix Index
$dbcdbcadbcdbcdbca	18
a$dbcdbcadbcdbcdbc	17
adbcdbcdbca$dbcdbc	07
bca$dbcdbcadbcdbcd	15
bcadbcdbcdbca$dbcd	05
bcdbca$dbcdbcadbcd	12
bcdbcadbcdbcdbca$d	02
bcdbcdbca$dbcdbcad	09
ca$dbcdbcadbcdbcdb	16
cadbcdbcdbca$dbcdb	06
cdbca$dbcdbcadbcdb	13
cdbcadbcdbcdbca$db	03
cdbcdbca$dbcdbcadb	10
dbca$dbcdbcadbcdbc	14
dbcadbcdbcdbca$dbc	04
dbcdbca$dbcdbcadbc	11
dbcdbcadbcdbcdbca$	01
dbcdbcdbca$dbcdbca	08

Tutorial09 Task01

- Now perform the pattern matching
  - It is in your lecture notes

Tutorial09 Task01

- Now perform the pattern matching
  - What do you need?
    - SP aka starting point
    - EP aka ending point

Tutorial09 Task01

- Now perform the pattern matching
  - What do you need?
    - SP aka starting point
    - EP aka ending point
  - Match from the back of the pattern
    - P[1, 2, …, m]
    - Starts from m then go to 1
    - Note you can start your index at 0 instead of 1 as well

Tutorial09 Task01

- Now perform the pattern matching
  - What do you need?
    - SP aka starting point
    - EP aka ending point
  - Match from the back of the pattern
    - P[1, 2, …, m]
    - Starts from m then go to 1
    - Note you can start your index at 0 instead of 1 as well
  - Update SP and EP as you loop from m to 1 (or 0) as i

```
sp = rank(pat[i]) + nOccurrences(pat[i], L[1...sp))
ep = rank(pat[i]) + nOccurrences(pat[i], L[1...ep]) - 1
```

Tutorial09 Task01

- Now perform the pattern matching

```
i = m = 3
sp = rank('a') + nOccurences('a', {}) = 1 + 0 = 1
ep = rank('a') + nOccurences('a', {accddddddbbbbbccc$a}) - 1 = 1 + 2 - 1 = 2
```

Tutorial09 Task01

- Now perform the pattern matching

```
i = m = 3
sp = rank('a') + nOccurences('a', {}) = 1 + 0 = 1
ep = rank('a') + nOccurences('a', {accdddddbbbbccc$a}) - 1 = 1 + 2 - 1 = 2
```

```
i = 1
sp = rank('b') + nOccurences('b', {accddddd}) = 3 + 0 = 3
ep = rank('b') + nOccurences('b', {accdddddbb}) - 1 = 3 + 2 - 1 = 4
```

Tutorial09 Task01

- Now perform the pattern matching

```
i = 0
sp = rank('d') + nOccurences('d', {acc}) = 13 + 0 = 13
ep = rank('d') + nOccurences('d', {accdd}) - 1 = 13 + 2 - 1 = 14

13: dbca$dbcdbcadbcdbc   | 14
14: dbcadbcdbcdbca$dbc   | 04

Multiplicity = ep - sp + 1 = 2
```

Tutorial09 Task02

- Graph
- Graph traversals

Tutorial09 Task02

- Dijkstra
  - A name you would need to remember as he contributed a lot of algorithms

Tutorial09 Task02

- Dijkstra
  - A name you would need to remember as he contributed a lot of algorithms
  - Need me to go through the algorithm?

Tutorial09 Task02

- Dijkstra
  - A name you would need to remember as he contributed a lot of algorithms
  - Need me to go through the algorithm?
    - Let us jump straight to Task03 first

Tutorial09 Task03

- Dijkstra’s shortest distance

Tutorial09 Task03

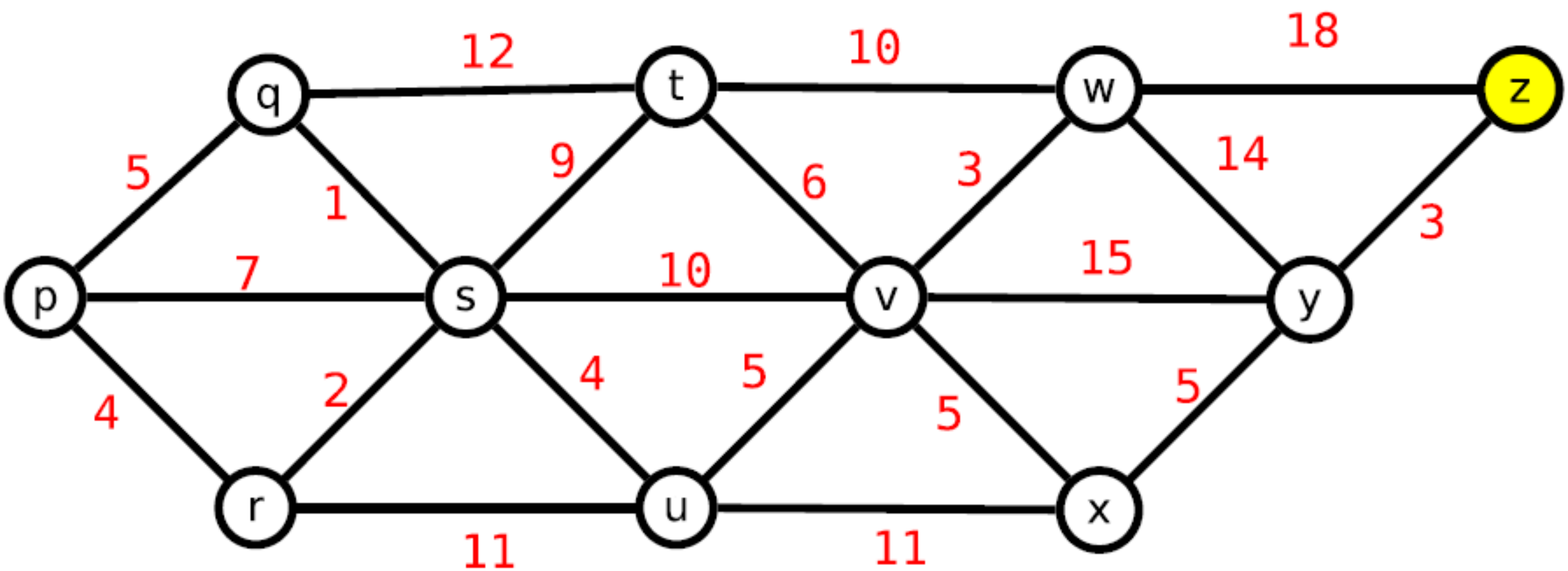
- Dijkstra’s shortest distance
  - Dynamic programming?
  - Greedy algorithm?

Tutorial09 Task03

- Dijkstra's shortest distance
  - Dynamic programming?
    - > Can be split into sub problems
    - > Solution of the sub problems make up the main solution. There are formal proofs to this.
  - Greedy algorithm?
    - > Look for the local optimal (in the sub problems)
    - > Limited to none-negative edges

Tutorial09 Task03

- Dijkstra’s shortest distance
  - Can anyone help me run through this?



Tutorial09 Task03

- Dijkstra's shortest distance
  - Can anyone help me run through this?
  - Traditionally implemented with a queue
    - Sorted according to minimum distance

Tutorial09 Task03

- Dijkstra's shortest distance
  - Can anyone help me run through this?
  - Traditionally implemented with a queue
    - Sorted according to minimum distance
  - Every vertex is initialized to a distance of infinity except the starting point of 0

Tutorial09 Task03

- Dijkstra's shortest distance
  - Can anyone help me run through this?
  - Traditionally implemented with a queue
    - Sorted according to minimum distance
  - Every vertex is initialized to a distance of infinity except the starting point of 0
  - Serve the shortest vertex
    - > Update the distance of the adjacent vertices by adding the distance value of the served vertex with the edge weight to the adjacent vertex

Tutorial09 Task03

- Dijkstra’s shortest distance
  - Can anyone help me run through this?
  - Traditionally implemented with a queue
    - Sorted according to minimum distance
  - Every vertex is initialized to a distance of infinity except the starting point of 0
  - Serve the shortest vertex
    - > Update the distance of the adjacent vertices by adding the distance value of the served vertex with the edge weight to the adjacent vertex
  - Sort the queue

Tutorial09 Task03

- Dijkstra's shortest distance
  - Can anyone help me run through this?
  - Traditionally implemented with a queue
    - Sorted according to minimum distance
  - Every vertex is initialized to a distance of infinity except the starting point of 0
  - Serve the shortest vertex
    - > Update the distance of the adjacent vertices by adding the distance value of the served vertex with the edge weight to the adjacent vertex
  - Sort the queue

Tutorial09 Task03

- Dijkstra's shortest distance
  - No smart board for me to do this directly unlike last year =(so try it on your own in paper

Tutorial09 Task02

- Let us come back to Task02 now
  - So now we have a priority queue via min-heap
  - What is the time complexity now then?

Tutorial09 Task02

- Let us come back to Task02 now
  - So now we have a priority queue via min-heap
  - What is the time complexity now then?

```
while remaining set not empty:
     x = closest vertex in remaining set;
     remove x from remaining set;
     if dist(x) is infinity: break;
     else for every y in remaining adjacent to x:
          est = dist(x) + w(<x,y>)
          if est < dist(y):
               dist(y) = est;
return dist;
```

Tutorial09 Task02

- Let us come back to Task02 now

O(V)

```
while remaining set not empty:
     x = closest vertex in remaining set;
     remove x from remaining set;
     if dist(x) is infinity: break;
     else for every y in remaining adjacent to x:
          est = dist(x) + w(<x,y>)
          if est < dist(y):
               dist(y) = est;
return dist;
```

Tutorial09 Task02

- Let us come back to Task02 now

O(V)

Serve: O(log V)

```
while remaining set not empty:
    x = closest vertex in remaining set;
    remove x from remaining set;
    if dist(x) is infinity: break;
    else for every y in remaining adjacent to x:
        est = dist(x) + w(<x,y>)
        if est < dist(y):
            dist(y) = est;
return dist;
```

Tutorial09 Task02

- Let us come back to Task02 now

O(V)

Serve: O(log V)

O(E)

```
while remaining set not empty:
     x = closest vertex in remaining set;
     remove x from remaining set;
     if dist(x) is infinity: break;
     else for every y in remaining adjacent to x:
          est = dist(x) + w(<x,y>)
          if est < dist(y):
               dist(y) = est;
return dist;
```

Tutorial09 Task02

- Let us come back to Task02 now

O(V)

Serve: O(log V)

```
while remaining set not empty:
     x = closest vertex in remaining set;
     remove x from remaining set;
     if dist(x) is infinity: break;
     else for every y in remaining adjacent to x:
          est = dist(x) + w(<x,y>)
          if est < dist(y):
               dist(y) = est;
return dist;
```

O(E)

Update: O(log V)

Tutorial09 Task02

- Let us come back to Task02 now

Serve: O(log V)

O(V)

```
while remaining set not empty:
      x = closest vertex in remaining set;
      remove x from remaining set;
      if dist(x) is infinity: break;
      else for every y in remaining adjacent to x:
            est = dist(x) + w(<x,y>)
            if est < dist(y):
                  dist(y) = est;
return dist;
```

O(E)

Update: O(log V)

Tutorial09 Task04

- Just building graphs
- Note have it as an adjacency list
  - Good practice for your practical 9 and also practical 10 (evaluated!)
  - Graph would need to be ADT with all key operations and well documented

Practical09 Task01

- Pattern matching!
- Implementation of what you have learnt so far

Practical09 Task01

- Rabin-Karp
- With BWT

Practical09 Task01

- Rabin-Karp
  - Rolling Hash for pattern matching (remember this concept)
  - Once again, making use of Hashing (been 4 practical now).

Practical09 Task01

- So what is a rolling hash?
- Let's use the example here:
  - A, C, G, T (proteins for DNA) that can be represented as a base-4 of 0, 1, 2, 3

Practical09 Task01

- So what is a rolling hash?
- Let's use the example here:
  - A, C, G, T (proteins for DNA) that can be represented as a base-4 of 0, 1, 2, 3
  - Example pattern {gact} would be 2013
  - Example of a long string {gatcaagacta} would be 20310020130

Practical09 Task01

- So what is a rolling hash?
- Let's use the example here:
  - A, C, G, T (proteins for DNA) that can be represented as a base-4 of 0, 1, 2, 3
  - Example pattern {gact} would be 2013
    - Simple hash = 2*10^3 + 0*10^2 + 1*10^1 + 3*10^0
  - Example of a long string {gatcaagacta} would be 20310020130

Practical09 Task01

- So what is a rolling hash?
- Let's use the example here:
  - A, C, G, T (proteins for DNA) that can be represented as a base-4 of 0, 1, 2, 3
  - Example pattern {gact} would be 2013
    - Simple hash = 2*10^3 + 0*10^2 + 1*10^1 + 3*10^0
  - Example of a long string {gatcaagacta} would be 20310020130
    - gatc = 2031
    - atca = 0310
    - Notice how the last 3 of the first and the first 2 of the second is the same?

Practical09 Task01

Practical09 Task01

- So what is a rolling hash?
- Let's use the example here:
  - A, C, G, T (proteins for DNA) that can be represented as a base-4 of 0, 1, 2, 3
  - Example of a long string {gatcaagacta} would be 20310020130
    - gatc = 2031
    - atca = 0310
    - Notice how the last 3 of the first and the first 2 of the second is the same? This is the rolling

Practical09 Task01

- So what is a rolling hash?
- Let's use the example here:
  - A, C, G, T (proteins for DNA) that can be represented as a base-4 of 0, 1, 2, 3
  - Example of a long string {gatcaagacta} would be 20310020130
    - gatc = 2031
    - atca = 0310
    - Notice how the last 3 of the first and the first 2 of the second is the same? This is the rolling
    - 0310 = (2031 – 2*10^3) * 10 + 0*10^0

Practical09 Task01

- So what is a rolling hash?
- Let's use the example here:
  - A, C, G, T (proteins for DNA) that can be represented as a base-4 of 0, 1, 2, 3
  - Example of a long string {gatcaagacta} would be 20310020130
    - 0310 = (2031 – 2*10^3) * 10 + 0*10^0

$$h_j = \left(\left(h_{j-1} - \mathtt{T[j-1]}z^{m-1}\right) z + \mathtt{T[j+m-1]}\right) \ \mathtt{modulo} \ q$$

Practical09 Task01

- So what happen when there is a match now?
  - Compare the characters now.
  - That is all.

Practical09 Task01

- Good read with more complex application (substrings)
  - http://people.csail.mit.edu/alinush/6.006-spring-2014/rec06-rabin-karp-spring2011.pdf

Practical09 Task01

- BWT pattern matching
  - Went through just now.
  - Just implement it now.

Practical09 Task01

- Rabin-Karp vs BWT pattern matching:
  - Which is faster?
  - Complexity?

Practical09 Task02

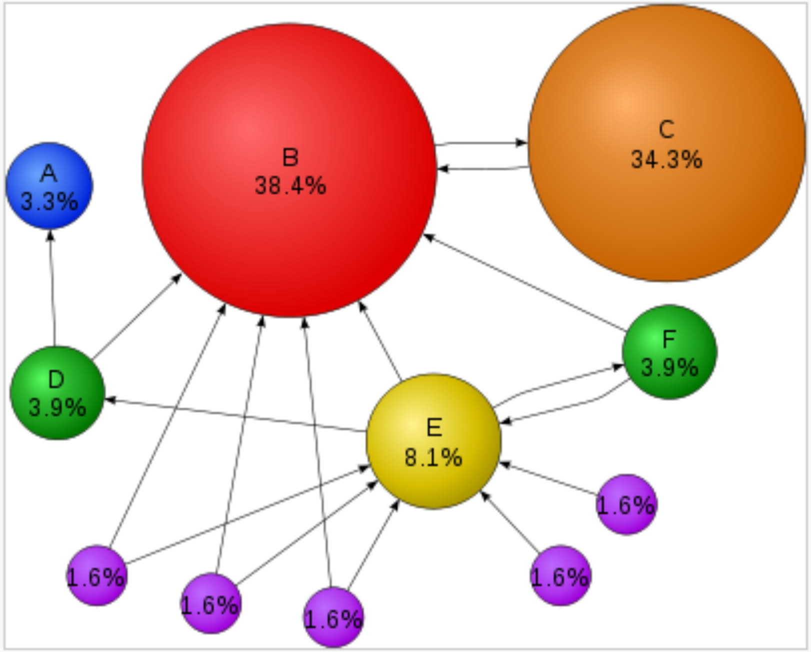
- Graphs!

Practical09 Task02

- Graphs!
  - If you think HashTable is important, graph is even more important.
  - The algorithm that made Google rich? It is a graph algorithm known as the PageRank based upon the Hyperlink-Induced Topic Search (HITS) algorithm.

Practical09 Task02

- Graphs!



Mathematical **PageRanks** for a simple network, expressed as percentages. (Google uses a logarithmic scale.) Page C has a higher PageRank than Page E, even though there are fewer links to C; the one link to C comes from an important page and hence is of high value. If web surfers who start on a random page have an 85% likelihood of choosing a random link from the page they are currently visiting, and a 15% likelihood of jumping to a page chosen at random from the entire web, they will reach Page E 8.1% of the time. (The 15% likelihood of jumping to an arbitrary page corresponds to a damping factor of 85%.) Without damping, all web surfers would eventually end up on Pages A, B, or C, and all other pages would have PageRank zero. In the presence of damping, Page A effectively links to all pages in the web, even though it has no outgoing links of its own.

Practical09 Task02

- Graphs Class for ADT
  - Vertex, V
  - Edge, E

Practical09 Task02

- Graphs Class
  - Vertex, V
  - Edge, E
    - $|E| <= |V|^2$ if directed
    - See lecture notes for undirected

Practical09 Task02

- Graphs Class
- Vertex Class
- Edge Class? Needed here?

Practical09 Task02

- Graphs Class
  - Contains vertices
- Vertex Class
- Edge Class? Needed here?

Practical09 Task02

- Graphs Class
  - Contains vertices
- Vertex Class
  - Contain edges
  - Have IDs
- Edge Class
  - Points to vertex
  - Can be weighted
  - Can be directed or undirected

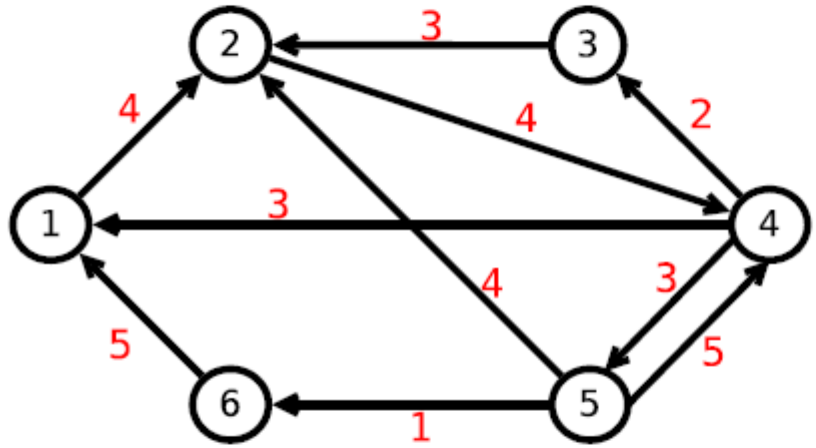
Practical09 Task02

- Why as OO with classes ADT etc?
  - Save space (when graph is sparse)
  - Encapsulate information
- Other approach?
  - Matrix (2D-array)
  - LinkedList
    - Suitable when graph is sparse

Practical09 Task02

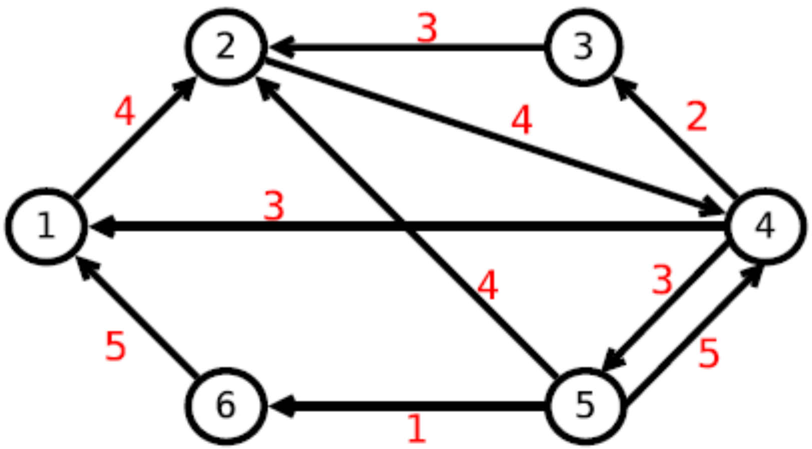
- Why as OO with classes ADT etc?
  - Save space (when graph is sparse)
  - Encapsulate information
- Other approach?
  - Matrix (2D-array)
  - LinkedList
    - Suitable when graph is sparse

Practical09 Task02



	1	2	3	4	5	6
1		4				
2				4		
3		3				
4	3		2		3	
5		4		5		1
6	5					

Practical09 Task02



Adjacency (linked) list

$[1] : \langle 1, 2, 4 \rangle \rightarrow \textbf{nil}$

$[2] : \langle 2, 4, 4 \rangle \rightarrow \textbf{nil}$

$[3] : \langle 3, 2, 3 \rangle \rightarrow \textbf{nil}$

$[4] : \langle 4, 1, 3 \rangle \rightarrow \langle 4, 3, 2 \rangle \rightarrow \langle 4, 5, 3 \rangle \rightarrow \textbf{nil}$

$[5] : \langle 5, 2, 4 \rangle \rightarrow \langle 5, 4, 5 \rangle \rightarrow \langle 5, 6, 1 \rangle \rightarrow \textbf{nil}$

$[6] : \langle 6, 1, 5 \rangle \rightarrow \textbf{nil}$

Practical09 Task02

- Implement the graph

Practical09 Task02

- Breadth-First Search (BFS)
- Depth-First Search (DFS)

Practical09 Task02

- Graph traversal algorithm
  - Breadth-First Search (BFS)
  - Depth-First Search (DFS)

Practical09 Task02

- Breadth-First Search (BFS)
  - How to implement?
  - Simple

Practical09 Task02

- Breadth-First Search (BFS)
  - Have a queue
  - For each vertex (starting from one), add its adjacent vertices into the queue
  - Serve the next vertex
  - Stop when item if found or no more item in the queue

Practical09 Task02

- Breadth-First Search (BFS)
  - Have a queue
  - For each vertex (starting from one), add its adjacent vertices into the queue
  - Serve the next vertex
  - Stop when item if found or no more item in the queue
  - Alternative? Simple recursion if you have the Graph implemented as ADTs

Practical09 Task02

- Depth-First Search (DFS)
  - Almost the same concept
  - Instead of a queue, why not a stack?
  - If ADT, traverse like a tree

Practical10

- Building a D-graph
- Traversing based on the given rule for the E-path

Practical10

- Building a D-graph
  - How to build?
  - What are the vertices?
  - What are the edges?
  - How to represent them?
- Traversing based on the given rule for the E-path

Practical10

- Building a D-graph
- Traversing based on the given rule for the E-path
  - Rules given in your practical sheet already

Practical10

- Task 01
  - Graph building is simple
    - Graph representation?
  - E-path traversal would have a lot of marks here
    - Good complexity during traversal (you would have learnt some traversal before)
- Task 02
  - Graph construction
    - More focus here than in Task01
  - E-path traversal
    - Start at the right point
    - Produce the right output

Practical10

- Once again, whatever you code need to have
  - Proper documentation
  - Good complexity
  - Modular

Practical10

- Haven’t got my marking scheme yet so I will fill that in an email

6th and 8th May 2015
Monash University – Sunway Campus, Malaysia

Thank You