**Attempts at solutions to any past exam questions will be much appreciated, so we can build a document of model answers. :)**

Past papers (more recent = more relevant):

[2009-10](http://studentnet.cs.manchester.ac.uk/assessment/exam_papers/UG_sem2_2009/COMP20512-jdk_jls_der_mdm.pdf)

[2010-11](http://studentnet.cs.manchester.ac.uk/assessment/exam_papers/UG_sem2_2010/COMP26120-sem2.pdf)

[2011-12](http://studentnet.cs.manchester.ac.uk/assessment/exam_papers/UG_sem2_2011/COMP26120-sem2.pdf)

[2012-13](http://studentnet.cs.manchester.ac.uk/assessment/exam_papers/UG_sem2_2012/COMP26120sem2.pdf)

[2013-14](http://studentnet.cs.manchester.ac.uk/assessment/exam_papers/UG_sem2_2013/COMP26120sem2.pdf)

**2013-14**

1. Graph Algorithms

**a)** Give a pseudocode description of Dijkstra’s algorithm for finding the shortest path between a given node and all other nodes in a weighted undirected graph. Suggest how Dijkstra’s algorithm can be generalised to find the shortest paths between all pairs of nodes. Find a modification of this approach which cuts the execution time by a half. In your answer assume that the priority queue and the functions operating on it are defined. (6 marks).

**ans)**

method dijkstra(graph, source):

for each vertex v in graph:

distance[v] = infinity

prev = null

visited[v] = false.

end for

distance[source] = 0

prev = null

add source to Pqueue

while Pqueue is not empty:

u = extractMin() // remove and retrieve the node with

// the min distance value from Pqueue

visited[u] = true

for each neighbour w of u:

d = distance[u] + length(u,w)

if d < distance[w] and visited[w] == false

distance[w] = d

prev[w] = u //store path for convenience

add w to Pqueue

end if

end for

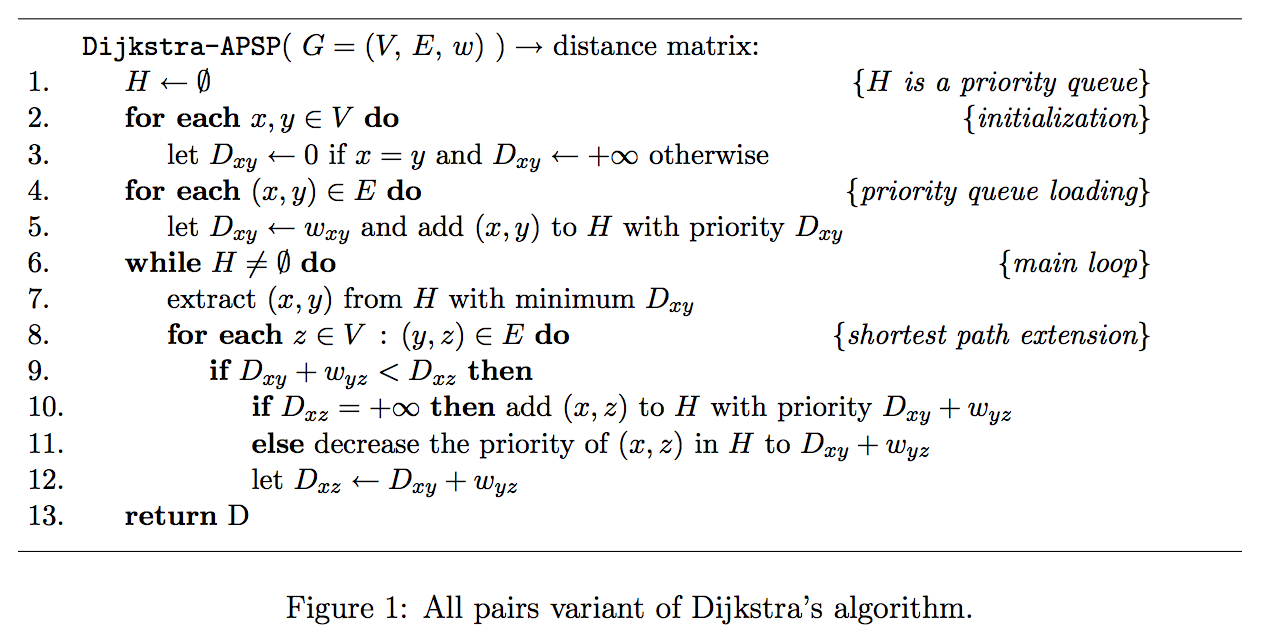
end while

return distance // return the distances array

end method

Simply run the algorithm on all nodes in the graph, keeping track of the the smallest value obtained in the distance array each time to evaluate the overall shortest paths.

Since the graph is undirected, the graph’s adjacency matrix will be symmetrical. Therefore it is only necessary to visit half of the nodes throughout the entire run, so for each source node *i* in a graph of *N* nodes, visit up to (*N*-*i*) nodes.



**b) i)**

Mins: A0, B10, C22, D12, E28, F28, G18

queue: A(0)

pop: A(0)

push: B(10), D(12), G(18)

queue: B(10), D(12), G(18)

pop: B(10)

push: C(22)  
  
queue: D(12), G(18), C(22)

pop: D(12)  
push: F(28)  
  
queue: G(18), C(22), F(28)  
  
pop: G(18)  
push: *nil*

queue: C(22), F(28)

pop: C(22)

push: E(28)

queue: F(28), E(28)

pop: F(28)

push: *nil*

queue: E(28)

pop: E(28)

push: *nil*

queue: empty

Shortest A -> B -> C -> E (28)

**ii)**

**2.** Knapsack problems.

**a)** Describe the 0/1 Knapsack Problem, concerning a set of items and a container of limited capacity. (4 marks)

**ans)** Given a set of items with weights and value, attempt to find the combination of items with the maximum total value whose weight is less than or equal to the weight of the knapsack. Items must be packed in their entirety / cannot be divided.

**b)** Briefly explain the standard Greedy method for solving the 0/1 Knapsack Problem, including any preprocessing step needed. (3 marks)

**ans)** Preprocessing - Sort items by descending weight / value ratio.  
  
Iterate through list of items, if the item will fit in the knapsack, pack it (We already know it has the best value / weight). Once the knapsack is full, or the end of the list has been reached break.

**c i)**

55 - 40 = 15

15/22 \* 220 = 150  
  
400 + 150 = 550 (upper bound)

**Q3**

a)Consider the following sequence of the keys: 2,17,24,8,11,4,6,19 We wish to insert these numbers in the above order (from left to right) into a hashtable of size 13 (with indices denoted from 0 to 12), using the hash function h(x) = (5x + 9) mod 13. Show the insertion procedure, step by step, assuming in your answer that collisions are handled by linear probing. (8 marks)

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
| 11 | 6 | 19 | 17 | 4 |  | 2 |  |  |  | 8 |  | 24 |

**2:** 19 % 13 = **6**

**17:** 94 % 13 = **3**

**24:** 129 % 13 = **12**

**8:** 49 % 13 = **10**

**11:** 64 % 13 = 12 → (12 + 1)% = **0**

**4:** 29 % 13 = 3 → **4**

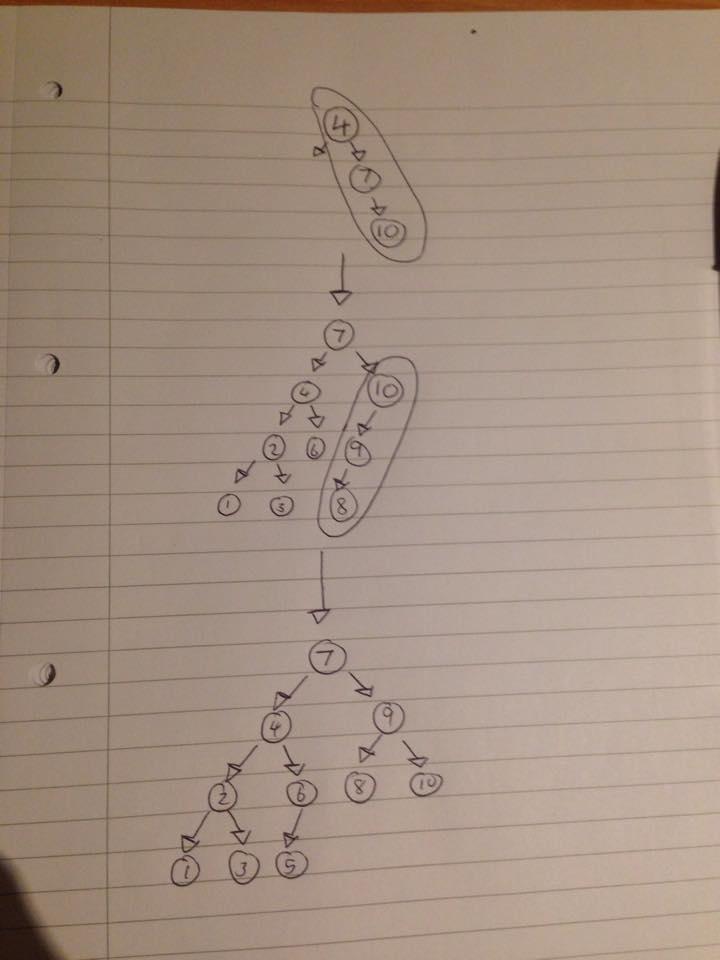
**6:** 39 % 13 = 0 → **1**

**19:** 104 % 13 = 0 → 1 → **2**

b) Consider the following set of keys: 4,7,10,2,6,9,1,3,8,5

[AVL schtuff](https://www.cs.usfca.edu/~galles/visualization/AVLtree.html)

i. Show the process, step by step, of inserting the above keys in the given order (from left to right) into an AVL tree. (7 marks)



*Here I used:*

*z*

*\*

*y → y*

*\ / \*

*x z x*

*Here I used:*

*z*

*/*

*y → y*

*/ / \*

*x x z*

**4)**

**a)** Explain clearly what is meant by the following search strategies for finite rooted trees:

i. Depth-first search (DFS)

We visit all descendants of a node before visiting sibling nodes

ii. Breadth-first search (BFS)

We visit all the children of a node, then all its grandchildren etc. (so visually would look as though we are scanning row by row)

Illustrate your explanations with suitable examples.(4 marks)

[insert illustrations here]

**b)** Now suppose we consider trees whose nodes are assigned numerical priorities. Explain clearly what is meant by a priority search over such a tree. Illustrate your answer with an example. How many priority searches be used to perform heuristic searching? (3 marks)

**ans)** For a given node *N*, a priority search will choose the child of *N* with the highest assigned priority first. Backtracking occurs if a leaf node is reached and the node we are searching hasn’t been found.

[insert illustration here]

An example of a priority search that uses heuristics to find a node *X*, would be to prioritise nodes in decreasing order of number of child nodes. For example, let the root node have child nodes *A*, *B* and *C*. if node *A* has 3 children, node *B* has 0 children and node *C* has 1 child, we first choose node A in the search. if node *X* isn’t found within this subtree, we choose node C as the next child node and repeat.

**c) i)**

A priority queue is a container of elements with keys associated with them, such that during insertion into the queue, an element’s key represents its priority in the queue. The *top()* function r

**ii)** [answer for the first part here]

A heap data structure is a way of storing a tree as a vector-based structure that has what is known as the ‘heap property’, that is, the order-relation of the tree must maintain for the heap to be valid. Available operations: