COMP261 – 2014 Paper

(1)

a) Three tasks:
- Pseudocode for Dijkstra's single source shortest path algorithm
- Generalisation for all shortest paths
- Cutting execution time by a half

```
Dijkstra ( graph, weights, start)
    D[]      // path lengths from start vertex
    paths[]
    queue = graph.getAllVertices()

    D[start] = 0
    for each vertex apart from start
        D[vertex] = +∞

    while (queue is not empty)
        currentV = queue.extract_min()
*       for each neighbour V of currentV s.t. neighbour V is in queue
            if ( D[currentV] + weights(currentV, neighbourV)
                 < D[neighbourV] )
            { D[neighbourV] = D[currentV] + weights(currentV, neighbourV)
              paths[neighbourV].add(paths[currentV])
              updateQueue() }
        end-for
    end-while
    for each vertex
        paths[vertex].add(vertex)
    end-for
    return D[] and paths[]
```

"relaxation"

(1)

a)
(continued)

Generalisation

```
for ( startVertex = 0 ; startVertex < graph.size ; startVertex ++ )
    Dijkstra ( graph, weight, startVertex );
```

Cutting execution time by half

~~for (startVertex = 0~~

alter this line from preceeding page *
to be

[for each neighbourV of currentV such that
neighbourV is in queue AND neighbourV > currentV]

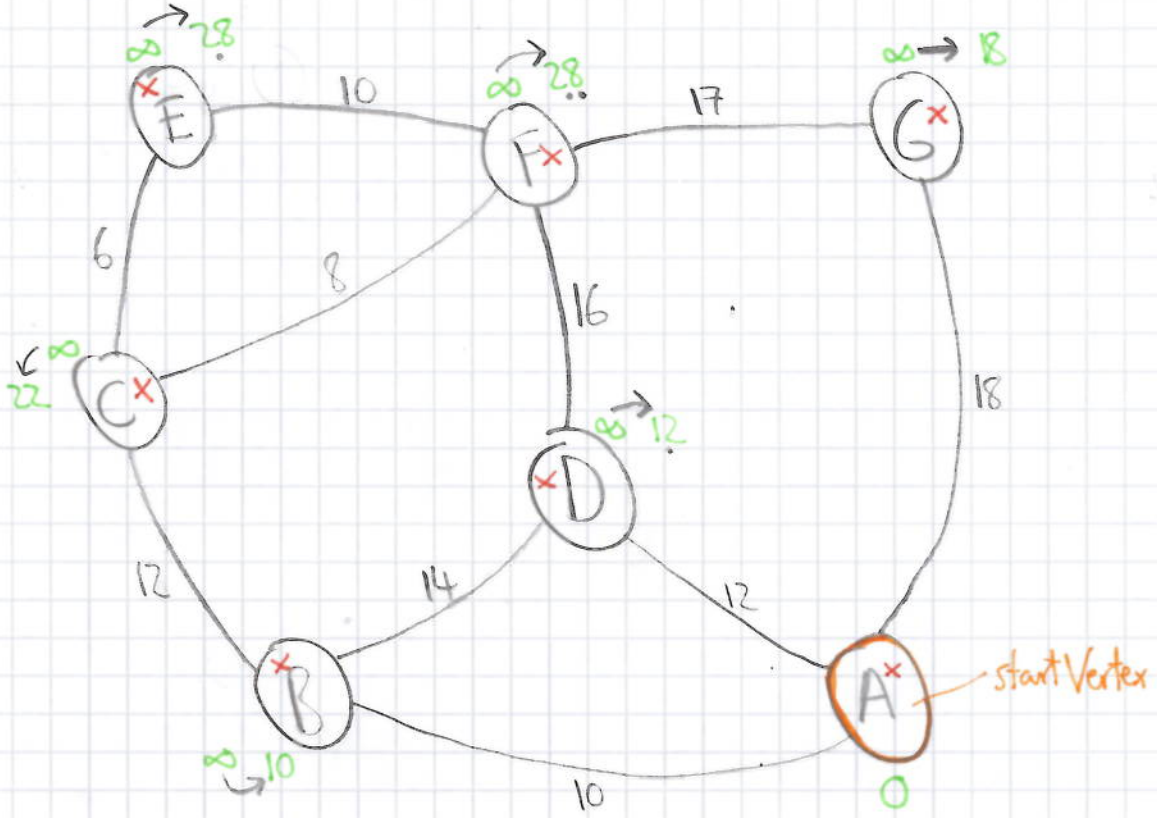
...

⇒ We don't calculate shortest path from
x to y and then again from y to x,
but only once (from x to y).

1

b) Task: Find shortest path between A and E

i) using Dijkstra's algorithm:



Queue

	step0		step1		step2		step3		step4		step5		step6	
	name	D	name	D	name	D	name	D	name	D	name	D	name	D
Head	A	0	B	10	D	12	G	18	C	22	F	28	E	28
	B	∞	D	12	G	18	C	22	F	28	E	28		
	C	∞	G	18	C	22	F	28	E	∞				
	D	∞	C	∞	F	∞	E	∞						
	E	∞	E	∞	F	∞								
	F	∞	F	∞										
	G	∞												

→ The shortest path from A to E has length 28
and it's this one: A —10→ B —12→ C —6→ E

4

a)
- We have a knapsack of a given capacity W, that is the maximum weight it can carry.
- We have a set of items $S = \{0, 1, \ldots, i, \ldots, N\}$
- Each item i has a value V_i and a weight w_i
- Then the 0/1 knapsack is this:

How do we maximise $\sum_{i \in S} V_i$

subject to $\sum_{i \in S} w_i \leq W$?

b)
- Preprocessing step: Compute the value-to-weight ratio for each item and sort the items in descending order according to this ratio.
- Pick the items in the order mentioned (i.e. biggest value-to-weight ratio to smallest) until ~~or~~ the next item would exceed W. Don't pick the last item that would exceed W.
  → you will get a good approximation, but not the optimal solution.

c)
i) 10^{****} represents the solution where we take item 1 and don't take item 2.

$FUB_{10****} = 400 + 0 + \frac{15}{22} \cdot 220 = 400 + 150 = 550$

(400 from item 1, 0 from item 2, $\frac{15}{22} \cdot 220$ from item 3; fractional upper bound)

ii) 10^{****}, 1^{****}, 100^{**} have the same FUB.

possibly more ...

2

d)

"real i" from given instance

"helper i" for table/algorithm

real i	i \ w	0	1	2	3	4	5	6	7	8	9	10	11
1	0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	0	0	0	12	12	12	12	12	12	12	12	12
1	2	0	0	0	12	12	12	24	24	24	24	24	24
1	3	0	0	0	12	12	12	24	24	24	36	36	36
1	4	0	0	0	12	12	12	24	24	24	36	36	36
2	5	0	0	0	12	16	16	24	28	28	36	36	36
2	6	0	0	0	12	16	16	24	28	32	36	40	44

⇒ optimal solution: value = 44
achieved by taking item (2) twice and
item (1) once. total_value = 44
total_weight = 11 ≤ C

Algorithm used: see summary (Pseudocode (DP))

3

a) $V = \{2, 17, 24, 8, 11, 4, 6, 13\}$, V is our set of values

Hash function:

index $= (5x + 9) \% 13$

Hash Table, storing values from V

step \ index	0	1	2	3	4	5	6	7	8	9	10	11	12
1							2						
2				17									
3													24
4											8		
5	11												~~24~~
6				~~17~~	4								
7	~~11~~	6											
8	~~11~~	~~6~~	19										

~~X~~ means we've had a clash