HW 5

Saturday, December 2, 2017 4:22 PM

1. Since you should be very comfortable and expert at programming, its easier to write a program for problems 2, 3 and 4 and you are strongly encouraged to do so and submit that. But for practice and better understanding, also *do go through first 5-10 steps* by yourself (i.e., hand-trace the algorithm in question). Writing program is optional but strongly encouraged (its for your own good). Turning in your well-designed code using good programming practices would be worth an extra credit of 20 pts total. Please avoid the temptation to simply copy code from the web! (otherwise the learning objective is defeated, what else?)

2. (20pts) Given a chain of six matrices M1, M2, ..., M6, where matrix Mi is of size r_{i-1} X r_i, with r0=3, r1=35, r2=15, r3=5, r4=10, r5=2 and r6=25. Show all the steps of exceuting the dynamic programming algorithm **as discussed in class** to find the best way to compute the product of the chain. Show the parenthesization as well as the number of operations used ito compute the product.

See code in ZIP file. Open the Matrix.java class and run it.

$r_0 = 3$ $A1 = 3, 35$ $m[1,2] = 3 \times 35 \times 15 = 1575$

$r_1 = 35$ $A2 = 35, 15$ $m[2,3] = 35 \times 15 \times 5 = 2625$

$r_2 = 15$ $A3 = 15, 5$ $m[3,4] = 15 \times 5 \times 10 = 750$

$r_3 = 5$ $A4 = 5, 10$ $m[4,5] = 5 \times 10 \times 2 = 100$

$r_4 = 10$ $A5 = 10, 2$ $m[5,6] = 10 \times 2 \times 25 = 500$

$r_5 = 2$ $A6 = 2, 25$

$r_6 = 25$

M	1	2	3	4	5	6
1	0	1575	1800	1950	1510	1600
2		0	2625	4375	1300	3050
3			0	750	250	1000
4				0	100	350
5					0	500
6						0

$m[1,3] = \min\{m[1,1] + m[2,3] + r_0 r_1 r_3\} = 0 + 2625 + 525 = 3150$

$m[1,3] = \min\{m[1,2] + m[3,3] + r_0 r_2 r_3\} = 1575 + 0 + 225 = 1800$

$m[1,3] = 1800$

$m(2,4) = \min\{m[2,2] + m[3,4] + r_1 r_2 r_4\} = 0 + 750 + 5250 = 6000$

$m(2,4) = 4375$ min

$m(3,5) = 250$

$m(4,6) = 350$

$m(1,4) = 1950$

$m(2,5) = 1300$

$m(3,6) = 1000$

$m(1,5) = 1510$

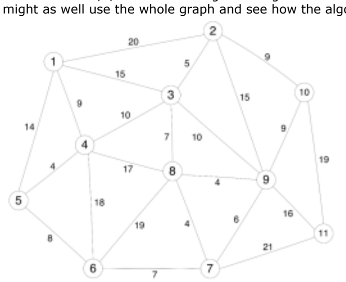
$m(2,6) = 3050$

$m(1,6) = 1660$

$(A_1 A_2)(A_3 A_4)(A_5 A_6)$

Time $O(n^3)$

3. (20pts) Show all the steps of the Floyd-Warshall's dynamic programming algorithm to compute all-pairs shortest paths in the graph in section 2.1 of the linked pdf note. Since the graph is rather large, for hand tracing, prune the graph to first four vertices (i.e., consider the graph with vertices 1,2, 3 and 4 and edges among these vertices with weights as shown). If you are writing a program, then you might as well use the whole graph and see how the algorithm works.



$d_0 =$

	1	2	3	4
1	0	20	15	9
2	20	0	5	∞
3	15	20	0	10
4	9	∞	10	0

1 ⟷ 2 | 20
1 ⟷ 3 | 15
1 ⟷ 4 | 9
2 ⟷ 3 | 5
2 ⟷ 4 | ∞
3 ⟷ 4 | 10

$d_1 =$

	1	2	3	4
1	0	20	15	9
2	20	0	35	29
3	15	35	0	24
4	9	29	24	0

2 ⟷ 1 ⟷ 3 | 35
2 ⟷ 1 ⟷ 4 | 29
3 ⟷ 1 ⟷ 4 | 24

$d_2 =$

	1	2	3	4
1	0	20	25	∞
2	20	0	5	∞
3	25	5	0	∞
4	∞	∞	∞	0

3 ⟷ 2 ⟷ 1 | 25
4 ⟷ 1 | ∞
4 ⟷ 2 | ∞
4 ⟷ 3 | ∞

$d_3 =$

	1	2	3	4
1	0	20	15	25
2	20	0	5	15
3	15	5	0	10
4	25	15	10	0

2 ⟷ 3 ⟷ 1 | 20
1 ⟷ 3 ⟷ 4 | 25
2 ⟷ 3 ⟷ 4 | 15

$d_4 =$

	1	2	3	4
1	0	∞	19	9
2	∞	0	∞	∞
3	19	∞	0	10
4	9	∞	10	0

2 ⟷ 3 | ∞
2 ⟷ 4 | ∞
1 ⟷ 4 ⟷ 3 | 19

```
for each source vertex
    for all vertices as destination
      if the distance[source][nextV] + distance[nextV][destination]
         < distance[source][destination] {
            dist[source][destination] = distance[source][nextV] + distance[nextV][destination]
```

4. (30pts) Consider the following instance of the 0/1 knapsack problem, with knapasack capacity 100

Weights	10	20	30	40	50
Profits	40	20	90	60	30

In class we discussed a DP formulation of solving the 0/1 knapsack problem, show all the steps using this formulation on the above instance.

See code in ZIP file. Open the knapsack.java class and run it.

weight = (10 + 20 + 30 + 40 + 50) > 100
construct temp Array[][] in bottom-up

Profits	W	0	1	2	3	4	5	6	10	20	...	30	...	40	...	60	...	90	100
40	10	0	0	0	0	0	0	0	40	40	40								
20	20	0	0	0	0	0	0	0	40	40	40	60	60	60	60	60	60	60	60
90	30	0	0	0	0	0	0	0	0	0	0	90	90	130	130	150	150	150	150
60	40	0																	
30	50	0																	210

max(20 + Array[10][10], 40) = 60
max(90 + Array[20][0], 60) = 90
max(90 + Array[20][10], 60] = 130
max(90 + Array[20][30], 10) = 150

210

$O(n\,Capacity)$

5. (30pts) Recall the RDS problems as discussed in class:

Given n jobs, each associated with a release time, a deadline, and a processing time on a machine M, is there a non-preemptive feasible schedule on M such that all jobs meet their deadlines?
For the following three instances of the RDS problem, determine whether there is a feasible schedule. If there is a feasible schedule, clearly indicate the schedule. The lists specify the attributes of jobs in order.

- n=5, ProcessingTimes={3, 4, 1, 2, 3}, ReleaseTimes={4, 2, 7, 5, 0} and Deadlines={13, 8, 13, 9, 9}.
- n=5, ProcessingTimes={3, 4, 1, 2, 3}, ReleaseTimes={4, 2, 7, 5, 0} and Deadlines={13, 5, 13, 9, 9}.
- n=5, ProcessingTimes={3, 4, 1, 2, 5}, ReleaseTimes={9, 2, 7, 5, 0} and Deadlines={12, 8, 13, 9, 9}.

Deadline − PT = Latest Start

	1	2	3	4	5
ES	4	2	7	5	0
End	7	6	8	7	3
LS	10	4	12	7	6

Sort by RT

5	2	1	4	3

sort by RT

	5	2	1	4	3
ES	0	2	4	5	7
end	3	6	7	7	8
LS	6	4	10	7	12

add J5

5

t = 3

- J2 will overlap, so have it start at 3

5, 2

T = 7

- J 4 has a latest start of 7, so start that

5, 2, 4

T = 9

- J 3 has a PT of 1, and J1 must start by 10

add J3

5, 2, 4, 3

T = 10

add J1

5, 2, 4, 3, 1

T = 13

	1	2	3	4	5
ES	4	2	7	5	0
End	7	6	8	7	3
LS	10	1	12	7	6

sort by earliest start time

	5	2	1	4	3
ES	0	2	4	5	7
End	3	6	7	7	8
LS	6	1	10	7	12

- start with J2 because it has to be started by T1

2

T = 4

- add J5 since it can start before T6

2, 5

T = 7

- add J4 since it starts by T7

2, 5, 4

T = 9

- add J1 since it must start by T10

2, 5, 4, 1

T = 12

- add J3

2, 5, 4, 1, 3

T = 13

	1	2	3	4	5
ES	9	2	7	5	0
End	12	6	8	7	3
LS	9	4	12	7	4

sort by earliest start time

	5	2	4	3	1
ES	0	2	5	7	9
End	3	6	7	8	12
LS	4	4	7	12	9

- start with J5

  5

  T = 3
- add J2 because it must start before T4

  5, 2

  T = 7
- add J4 because it must start by T7

  5, 2, 4

  T = 9
- add J1

  5, 2, 4, 1

  T = 12

add J 3

5, 2, 4, 1, 3