

Ambedkar College of Arts and Science, Wandoor

Class Test 3 — Solutions

Data Structures and Algorithms — MAT5EJ302(1)

B.Sc. Mathematics Honours (CUFYUGP 2024) — Semester V

Maximum Marks: 30

Time: 30 Minutes

9 questions of 5 marks each; students answer any 6 (ceiling 30 marks).

Question 1. Strongly Connected Components.

[5 Marks]

(a) From P we reach Q, R (via $P \rightarrow Q \rightarrow R$) and back to P via $R \rightarrow P$, so P, Q, R are mutually reachable. Q and R also reach S , but nothing in $\{S, T, U\}$ leads back to P, Q, R , so S is *not* strongly connected to them. $S \rightarrow T \rightarrow S$ makes $\{S, T\}$ mutually reachable. $T \rightarrow U$ but U has no outgoing edge, so $\{U\}$ stands alone.

$$\boxed{\{P, Q, R\}, \{S, T\}, \{U\}.$$

(b) Edges between different components: $Q \rightarrow S$ and $R \rightarrow S$ (from $\{P, Q, R\}$ to $\{S, T\}$), and $T \rightarrow U$ (from $\{S, T\}$ to $\{U\}$).

(c) **No.** Suppose components $C_1 \rightarrow C_2 \rightarrow \dots \rightarrow C_k \rightarrow C_1$ formed a cycle. Then a vertex in C_1 would reach a vertex in C_2 and vice versa, so by definition $C_1 \cup C_2$ would be a set of mutually reachable vertices — contradicting that C_1 and C_2 are maximal and different. Hence the graph of components is always a DAG.

(d) At least one: $\{U\}$ has no outgoing edge, so G is not strongly connected. One edge suffices: add $U \rightarrow P$. Now $P \rightarrow Q \rightarrow S \rightarrow T \rightarrow U \rightarrow P$ contains P, Q, S, T, U , and R lies on $P \rightarrow Q \rightarrow R \rightarrow P$, so every vertex reaches every other.

$$\boxed{\text{one edge, e.g. } U \rightarrow P.}$$

($U \rightarrow S$ would not work: nothing could then return to P, Q, R .)

Question 2. Priority Queues in Dijkstra.

[5 Marks]

(a) n deletemins at $\mathcal{O}(n)$ each and m decreasekeys at $\mathcal{O}(1)$ each: $n \cdot \mathcal{O}(n) + m \cdot \mathcal{O}(1) = \boxed{\mathcal{O}(n^2)}$, since $m \leq n^2$.

(b) All $n + m$ operations cost $\mathcal{O}(\log n)$: $\boxed{\mathcal{O}((n + m) \log n)}$.

(c) If $m = \Theta(n)$ the array costs $\mathcal{O}(n^2)$ and the heap costs $\mathcal{O}(n \log n)$. **The heap is faster.**

(d) If $m = \Theta(n^2)$ the array costs $\mathcal{O}(n^2)$ and the heap costs $\mathcal{O}(n^2 \log n)$. **The array is faster**, because the heap pays $\log n$ on each of the $\Theta(n^2)$ decreasekeys.

(e) The heap bound $m \log n$ matches n^2 when $m \approx n^2 / \log n$. For sparser graphs than this the heap wins; for denser ones the array wins.

Question 3. Breadth-First Search.

[5 Marks]

(a) Queue trace (neighbours scanned in alphabetical order): S discovers A, B ; A discovers C, D (B already seen); B discovers E ; C discovers F ; D, E discover nothing new; F discovers T .

v	S	A	B	C	D	E	F	T
dist	0	1	1	2	2	2	3	4

BFS tree edges: $SA, SB, AC, AD, BE, CF, FT$.

(b) The graph has 11 edges and the tree has 7, so there are 4 non-tree edges: AB (levels 1, 1), BD (1, 2), DF (2, 3), EF (2, 3). A non-tree edge can never join levels differing by 2 or more: if uv is an edge with $\text{level}(u) \leq \text{level}(v)$ then $\text{level}(v) \leq \text{level}(u) + 1$, since v would otherwise have been discovered earlier from u .

(c) Count shortest paths by predecessors: A, B : 1 each; C (via A): 1; D (via A and B): 2; E (via B): 1; F (via C, D, E): $1 + 2 + 1 = 4$; T (via F): 4.

4 shortest S - T paths.

Question 4. Dijkstra's Algorithm.

[5 Marks]

(a) Distances after each removal (new values only):

removed	S	B	A	C	D	T
dist	0	1	3	6	8	9

S sets $A=4$, $B=1$. B (1) lowers A to $1 + 2 = 3$ and sets $C=7$. A (3) lowers C to 6 and sets $D=10$. C (6) lowers D to 8 and sets $T=11$. D (8) lowers T to 9. Following parent pointers back from T : $T \leftarrow D \leftarrow C \leftarrow A \leftarrow B \leftarrow S$.

$S \rightarrow B \rightarrow A \rightarrow C \rightarrow D \rightarrow T$, length 9.

(b) Take vertices s, a, b, c with edges $s \rightarrow a$ (2), $s \rightarrow b$ (3), $b \rightarrow a$ (-2), $a \rightarrow c$ (1). Dijkstra removes a first (dist 2), declares it final and sets $c=3$. It then removes b (dist 3); scanning $b \rightarrow a$ would give $a = 1$, but a has already been finalised and c is never updated. Reported: $a = 2$, $c = 3$. True: $a = 1$ (via b), $c = 2$. The greedy argument “a vertex removed with the smallest distance cannot be improved later” relies on all edge lengths being non-negative.

Question 5. Shortest Paths in a DAG.

[5 Marks]

(a) Initially S is the only vertex with no incoming edge. Delete S (with $S \rightarrow A$, $S \rightarrow B$): now B is the only source (A still has $B \rightarrow A$). Delete B : now A is the only source. Delete A (with $A \rightarrow C$, $A \rightarrow D$): C is the only source (D still has $C \rightarrow D$). Delete C : D . Delete D : T .

S, B, A, C, D, T .

Every edge indeed points left to right.

(b) A vertex's distance depends only on vertices before it in the order, so process left to right:

	S	B	A	C	D	T
dist	0	2	1	-1	0	-3

($B = 0 + 2 = 2$; $A = \min(0 + 3, 2 - 1) = 1$; $C = 1 - 2 = -1$; $D = \min(1 + 4, 2 + 5, -1 + 1) = 0$; $T = \min(-1 + 6, 0 - 3) = -3$.)

(c) Negate every length and repeat; a shortest path in the negated graph is a longest path in the original. The negated run gives $\text{dist}(T) = -7$ via $S \rightarrow A \rightarrow C \rightarrow T$ ($-3 + 2 - 6 = -7$), so

longest path $S \rightarrow A \rightarrow C \rightarrow T$, length $3 - 2 + 6 = 7$.

Question 6. Depth-First Search.

[5 Marks]

(a) From A : $A(1) \rightarrow B(2) \rightarrow C(3)$; $C \rightarrow A$ is a back edge; C done (4). $B \rightarrow E(5) \rightarrow F(6)$, F done (7), E done (8), B done (9). $A \rightarrow D(10)$; $D \rightarrow E, F$ already finished; D done (11). $A \rightarrow F$ already finished; A done (12). Restart at G : pre 13, post 14.

	A	B	C	D	E	F	G
pre	1	2	3	10	5	6	13
post	12	9	4	11	8	7	14

(b) Tree edges: AB, BC, BE, EF, AD .

- $C \rightarrow A$: **back** (A is an ancestor of C).
- $A \rightarrow F$: **forward** (F is a descendant of A reached earlier through B, E).
- $D \rightarrow E, D \rightarrow F, G \rightarrow D, G \rightarrow F$: **cross** (the head finished before the tail started, or is in a different tree branch).

(c) The only cycle is $A \rightarrow B \rightarrow C \rightarrow A$ (the back edge $C \rightarrow A$ is the only edge that returns to an ancestor). Delete $C \rightarrow A$; this does not change any other pre/post number. Sorting by decreasing post number:

 $G, A, D, B, E, F, C.$

Every edge ($G \rightarrow D, G \rightarrow F, A \rightarrow B, A \rightarrow D, A \rightarrow F, B \rightarrow C, B \rightarrow E, D \rightarrow E, D \rightarrow F, E \rightarrow F$) points left to right.

Question 7. Multiplication by Halving and Doubling.

[5 Marks]

(a) Each call halves y (rounding down) until it reaches 0:

y	29	14	7	3	1	0
returns	1073	518	259	111	37	0

Working back up: $y=0 \mapsto 0$; $y=1$ (odd) $\mapsto 37+2 \cdot 0 = 37$; $y=3 \mapsto 37+2 \cdot 37 = 111$; $y=7 \mapsto 37+2 \cdot 111 = 259$; $y=14$ (even) $\mapsto 2 \cdot 259 = 518$; $y=29 \mapsto 37 + 2 \cdot 518 = 1073$. Check: $37 \cdot 29 = 1073$.

(b) The recursion depth equals the number of bits of y , so n recursive calls (plus the base call). Each call does a shift and at most one addition of $\mathcal{O}(n)$ -bit numbers, i.e. $\mathcal{O}(n)$ bit operations. Total: $\boxed{\mathcal{O}(n^2)}$.

Question 8. Merging Sorted Lists.

[5 Marks]

(a) Comparisons (front elements, smaller goes out): $2:3 \rightarrow 2$; $5:3 \rightarrow 3$; $5:4 \rightarrow 4$; $5:10 \rightarrow 5$; $9:10 \rightarrow 9$; $14:10 \rightarrow 10$; $14:11 \rightarrow 11$; $14:25 \rightarrow 14$; $20:25 \rightarrow 20$. The first list is now empty, so 25, 30 are appended with no comparison.

 $[2, 3, 4, 5, 9, 10, 11, 14, 20, 25, 30]$ with 9 comparisons.

(b) Each comparison moves one element to the output and the last element needs none, so at most $m + n - 1$; this is reached when the lists alternate, e.g. $[1, 3]$ and $[2, 4]$ (3 comparisons). The least happens when one list is exhausted first: each of the $\min(m, n)$ elements of the shorter list is compared once, e.g. $[1, 2]$ and $[3, 4, 5]$ (2 comparisons).

 $\min(m, n)$ to $m + n - 1$.

(c) Merging two lists of $n/2$ costs at most $n - 1$, so $W(n) = 2W(n/2) + n - 1$ with $W(1) = 0$: $W(2) = 1$, $W(4) = 2 \cdot 1 + 3 = 5$, $W(8) = 2 \cdot 5 + 7 = \boxed{17}$.

Question 9. Growth Rates.

[5 Marks]

(a) $2^{n+1} = 2 \cdot 2^n$, so $2^{n+1} \leq c \cdot 2^n$ with $c = 2$: **yes**. $2^{2n}/2^n = 2^n \rightarrow \infty$, so no constant c works: **no**.

(b) Recall $f(n) = \Omega(g(n))$ means $f(n) \geq c g(n)$ for all $n \geq n_0$, for some constants $c > 0$, n_0 . For $n \geq 2$, $n! = n(n-1) \cdots 1 \geq n(n-1) \cdots (\lceil n/2 \rceil) \geq (n/2)^{n/2}$, since there are at least $n/2$ factors, each at least $n/2$. Hence

$$\log_2(n!) \geq \frac{n}{2} \log_2 \frac{n}{2} = \frac{n}{2}(\log_2 n - 1) \geq \frac{n}{4} \log_2 n \quad (n \geq 4),$$

so $\log_2(n!) = \Omega(n \log n)$.

(c) $n^{\log_2 n} = 2^{\log_2 n \cdot \log_2 n} = 2^{(\log_2 n)^2}$. The two functions are **identical**: neither grows faster.