MID-SEMESTER EXAMINATION, April-2026
Algorithm Analysis and Design-2(CSE2632)

Program: B.Tech(CSE/CSIT/CDS/CIOT/CAIML/CCS) **Semester: 4th**

Full Marks: 30 **Time: 2 Hours**

Subject/Course Learning Outcome	Ques. Nos.	Marks
CO1: understand the network flow problem and apply it to real-world problems.	1(a), 1(b), 1(c), 2(a), 2(b)	2+2+2 +2+2
CO2: -distinguish between computationally tractable and intractable problems. - define and relate class-P, class-NP and class NP-complete, PSPACE, PSPACE-complete. - given a problem in NP, define an appropriate certificate and the verification algorithm.	2(c), 3(a), 3(b), 3(c), 4(a), 4(b), 4(c),	2+2+2 +2+2+ 2+2
CO3: understand approximation algorithms and apply this concept to solve problems.	5(a), 5(b), 5(c)	2+2+2
CO4: understand local search techniques and apply this concept to solve problems.		
CO5: understand randomization and apply this concept to solve problems.		
CO6: identify and apply an appropriate algorithmic approach to solve a problem and explain the challenges to solve it.		

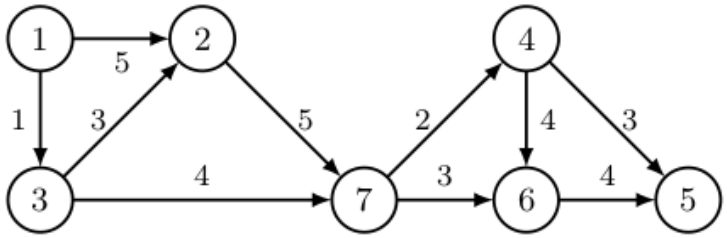
*Bloom's taxonomy levels: Remembering (L1), Understanding (L2), Application (L3), Analysis (L4), Evaluation (L5), Creation (L6)

Answer all questions. Each question bit carries equal mark(i.e., two)

1. (a) A weighted digraph G = ((V, E), c) has the sets V = {1, 2, 3, 4, 5, 6, 7} of nodes and E = {(1, 2), (1, 3), (2, 7), (3, 2), (3, 7), (4, 5), (4, 6), (6, 5), (7, 4), (7, 6)} of arcs and the following weights, or costs c(e); e ∈ E, of the arcs: L5

e	(1,2)	(1,3)	(3,2)	(2,7)	(3,7)	(4,5)	(4,6)	(7,4)	(6,5)	(7,6)
c(e)	5	1	3	5	4	3	4	2	4	3

Determine the minimum-cut value and the corresponding cut-edges.



Source: 1 (since the only vertex with positive out-degree and zero in-degree)Sink: 5 (since the only vertex with positive in-degree and zero out-degree)

Minimum cut-value = 5
Minimum cut: S = {1,2,3,7} and T = {4,5,6} --- [may differ depending on the selection of augmenting paths]
Cut edges = {(7,4), (7,6)} ----[based on the minimum cut mentioned above]

(b) ***Explain*** with proper justifications regarding why the maximum flow value equals to the minimum cut value in a flow network. L4

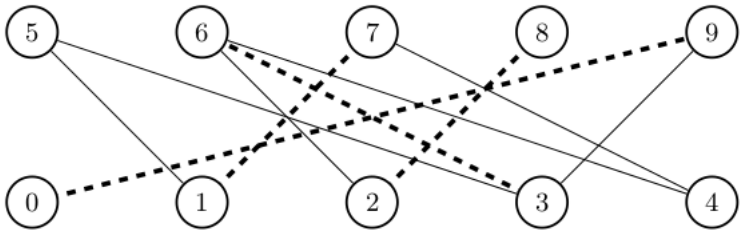
If f is a flow in a flow network G = (V, E) with source s and sink t, then the
following conditions are equivalent:
1. f is a maximum flow in G.
2. The residual network G_f contains no augmenting paths.
3. |f | = c(S, T) for some cut (S, T) of G.

(c) Suppose that a maximum flow for G has been computed using Ford-Fulkerson, and a new edge with unit capacity is added to E. ***Describe how*** the maximum flow can be efficiently updated. (Note: It is not the value of the flow that must be updated, but the flow itself.) ***Analyze*** your algorithm. L5 L4

Simply run one more BFS to find one augmenting path in the residual flownetwork. This costs O(V+E) time. One path suffices because assuming all edges have capacity 1 so any augmenting path will have capacity 1 as well.
We could also precompute in O(E) time for each vertex in the graph an edge on a path either to sink or from source in the residual flow network. (Can't have both if the flow is maximum!) Then, given the new edge we can update the flow in O(V) time.

2. (a) A bipartite graph G = (V, E) has the sets V = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9} of vertices and E = {(0, 9), (1, 5), (1, 7), (2, 6), (2, 8), (3, 5), (3, 6), (3, 9), (4, 6), (4, 7)} of edges. Given a maximal matching M° = {(0, 9), (1, 7), (2, 8), (3, 6)} of the four pairs of vertices, ***find*** two maximum matchings of all five pairs of vertices. ***Select*** two possible augmenting paths P_1 and P_2 for M° and ***justify*** your selection. L3 L5

{(0, 9), (1, 5), (2, 8), (3, 6), (4,7)}
{(0, 9), (1, 7), (2, 8), (3, 5), (4,6)}



Possibility-1: Instead of (1,7) , add (1,5) and (4,7)
Possibility-2: Instead of (3,6), add (3,5) and (4,6)

(b) "For each node v in flow network G, the sum of the flow value f(e) over all edges entering node v equals to the sum of flow values over all egdes leaving node v." --- ***Justify*** that truthness or falsity of the given statement. L2
False. It is true for all vertices except source and sink vertices.

(c) ***Show*** that INDEPENDENT-SET $\equiv_P$ VERTEX-COVER. L5

Pf. We show S is an independent set iff V – S is a vertex cover.

⇒

Let S be any independent set.

Consider an arbitrary edge (u, v).

S independent ⇒ u ∉ S or v ∉ S ⇒ u ∈ V – S or v ∈ V – S.

Thus, V – S covers (u, v).

⇐

Let V – S be any vertex cover.

Consider two nodes u ∈ S and v ∈ S.

Observe that (u, v) ∉ E since V – S is a vertex cover.

Thus, no two nodes in S are joined by an edge ⇒ S independent set. ■

3. (a) ***Design*** a polynomial-time reduction algorithm for Breadth-First-Search $\leq_P$ Dijkstra-Shortest-Path. L6

BFS(G=(V,E), s)

{

Transform G to G_w by taking each edge-weight equals to one;

Dijkstra-Shortest-Path(G_w=(V,E_w),s);

}

(b) ***Show*** that "Graph G = (V, E) contains a vertex cover of size k iff (U, S, k) contains a set cover of size k." [Symbols have their usual meraning] L5

Claim. VERTEX-COVER ≤ P SET-COVER .

Pf. Given a VERTEX-COVER instance G = (V, E), k, we construct a set

cover instance whose size equals the size of the vertex cover instance.

Construction.

Create SET-COVER instance:

– k = k, U = E, S v = {e ∈ E : e incident to v }

Set-cover of size ≤ k iff vertex cover of size ≤ k. ■

(c) ***Justify*** that truthness or falsity of the following statement: L5

P ⊆ NP ⊆ PSPACE

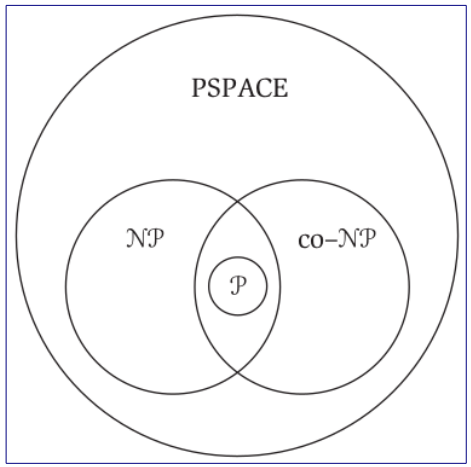
Theorem. NP ⊆ PSPACE.

Pf. Consider arbitrary problem Y in NP.

Since Y $\leq_P$ 3-SAT, there exists algorithm that solves Y in poly-time plus polynomial number of calls to 3-SAT black box.

Can implement black box in poly-space. ■

4. (a) Using a suitable Venn-diagram, ***analyze*** the present belief on how class P, NP, co-NP and PSPACE are related. L5



OR

Formulate the 8-puzzle problem as PLANNING problem and ***justify*** why it is considered under class-PSPACE.

Refere: Lecture Slide 10 of Ch.9 Tardos

Planning example. Can we solve the 8-puzzle?

Conditions. C_{ij} , 1 ≤ i, j ≤ 9. ←- C_{ij} means tile i is in square j

Initial state. c_0 = {C_{11} , C_{22} , …, C_{66} , C_{78} , C_{87} , C_{99} }.

Goal state. c* = {C_{11}, C_{22}, …, C_{66}, C_{77}, C_{88}, C_{99}}.

Operators.

Precondition to apply O_i = {C_{11}, C_{22}, …, C_{66}, C_{78}, C_{87}, C_{99} }.

After invoking O_i , conditions C_{79} and C_{97} become true.

After invoking O_i , conditions C_{78} and C_{99} become false.

(b) ***Prove*** that: Q-SAT ∈ PSPACE. L5

Pf. Recursively try all possibilities.
Only need one bit of information from each subproblem.
Amount of space is proportional to depth of function call stack.

(c) "Given a graph G = (V, E) and an integer k, is there a subset of vertices $S \subseteq V$ such that $|S| \leq k$, and for each edge (u, v) either $u \in S$ or $v \in S$ or both?" -- But what if k is small in this problem statement. ***Explain*** with proper reasons. L4

Brute force. $O(k\, n^{k+1})$.
Try all $C(n, k) = O(n^k)$ subsets of size k.
Takes O(kn) time to check whether a subset is a vertex cover.
Goal: Limit exponential dependency on k, e.g., to $O(2^k kn)$.
Remark. If k is a constant, algorithm is poly-time; if k is a small constant, then it's also practical.

5. Suppose we have m machines and n = m(m-1) + 1 jobs. The first m(m-1) jobs each require processing time $t_j = 1$. The last job is much larger and it requires a processing time $t_n = m$.

(a) What does our greedy algorithm (i.e., current job j is to be assigned to the machine whose load is smallest so far) for load-balancing-problem do with this sequence of jobs? What will be the resulting makespan (i.e., $T = \max_i T_i$) in terms of m? L3

It evenly balances the first n – 1 jobs, and then has to add the giant job n to one of them; the resulting makespan is T = 2m – 1.

(b) What does the optimal solution look like in this example? L3

It assigns the large job to one of the machines, say, M_1, and evenly spreads the remaining jobs over the other m–1 machines. This results in a makespan of m.

(c) Briefly ***explain*** "lower-bound on optimum". ***Devise*** at least two such lower bounds on optimum for the load-balancing-problem, i.e., to minimize the maximum load on any machine. L4 L6

* Lower-bound on optimum –- a quantity with the guarantee that no matter how good the optimum is, it cannot be less than this bound.
* One of the m machines must do at least a 1/m fraction of the total work.
* The optimal makespan is at least $T^* \geq \max_j t_j$.

End of Questions