1. [25 points] TRUE/FALSE OR PICK ONE . *No need for justification.*

(a) TRUE/FALSE

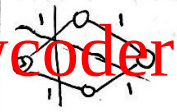
F

Given a connected, undirected graph $G = (V, E)$, one can determine in $O(|V|)$ time whether there is an edge in E that can be removed from G while still keeping G connected.

(b) TRUE/FALSE

T

If (S_1, T_1) and (S_2, T_2) be both minimum (s,t)-cuts in some flow network G, then, $(S_1 \cap S_2, T_1 \cup T_2)$ is also a minimum (s,t)-cut in G.



$S_1 \cap S_2 = S_1 = S$
$T_1 \cup T_2 = T_1$

(c) TRUE/FALSE

F

For an arbitrary edge $u \to v$ in a flow network G, if there exists a minimum (s,t)-cut (S, T) such that $u \in S$ and $v \in T$, then there exists no minimum cut (S', T') such that $u \in T'$ and $v \in S'$.

(d) TRUE/FALSE

F

If the minimum spanning tree of a graph is unique then the edge weights must be distinct.

(e) PICK ONE

Let $G = (V, E)$ be a directed graph with edge-weight function $w : E \to R$. Consider an adjacency matrix $A = (a_{ij})$ where $a_{ij} = w(i,j)$ or ∞. Let $d_{ij}^{(m)}$ denote the weight of a shortest path from i to j that uses at most m edges. Which of the following recurrences correctly formulate a dynamic programming solution for the all-pairs shortest path problem?

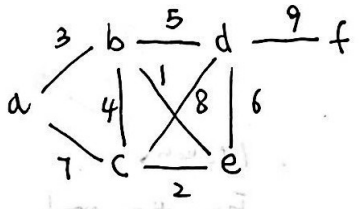
✓ (i) $d_{ij}^{(m)} = \min_{1 \le k \le n}\{d_{ik}^{(m-1)} + a_{kj}\}$ for $m = 1, 2, \ldots, n-1$

(ii) $d_{ij}^{(m)} = \min\{d_{ij}^{(m-1)} + a_{ij}\}$ for $m = 1, 2, \ldots, n$

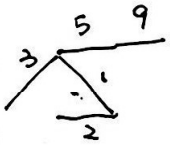
(iii) $d_{ij}^{(m)} = \min_{m}\{d_{ik}^{(m-1)} + a_{kj}\}$ for $k = 1, 2, \ldots, n$

(iv) $d_{ij}^{(m)} = \min_{i \le k \le j}\{d_{ik}^{(m-1)}\} + a_{kj}$ for $m = 1, 2, \ldots, n-1$

(v) none of the above



2. [30 points] SAFEST PATHS AND SPANNING TREES



For an undirected graph $G = (V, E)$, let V represent the campsites in the Everglades Park, and E represent the hiking trails between them. Each edge is weighted with its *danger level*. The *danger level of a path* between two campsites is determined by the weight of the most dangerous edge on it. For any two campsites, a path with the smallest danger level is called *the safest path* between them.

(a) Prove that a safest path between two campsites is always on a minimum spanning tree.

(b) Design and analyze an algorithm that computes the safest path for every pair of campsites in total $O(E \log V)$ time.

(c) Illustrate how your algorithm works on input $G = (V = \{a, b, c, d, e, f\}, E = \{((a, b), 3), ((a, c), 7), ((b, c), 4), ((b, d), 5), ((b, e), 1), ((c, e), 2), ((c, d), 8), ((d, e), 6), ((d, f), 9)\})$.

(a) Prove by contradiction:

Let's assume there is one safest path $p=(u,v)$ not on the minimum spanning tree, then we can know the path $p=(u,v)$ should be the shortest path between vertice u and v.

At the same time, we can find path p' between (u,v) on the MST. Based on the Algorithm we build MST, we use Greedy Algorithms to pick shortest edge to add into MST. Thus, path p' is the shortest path between (u,v) in our graph.

But we assume p is the shortest path between (u,v) and not in MST. Contradiction! Thus the shortest path between two campsites, ie. the safest path is always on a MST.

Another way to proof: (if not by contradiction)

For pair (u,v), we can know the shortest edge on the path should be on MST (based on MST's property). If we remove that shortest edge, we can't find one path containing new shortest edge which is smaller than previous shortest edge. Thus the path should on MST.

(b) Algorithm:

① Use Kruskal's method to build MST.

② For pair (u,v), travelsal starting from u, along the path in MST, stop at v.

③ connection of these edges along the path is the safest path.

Analysis:

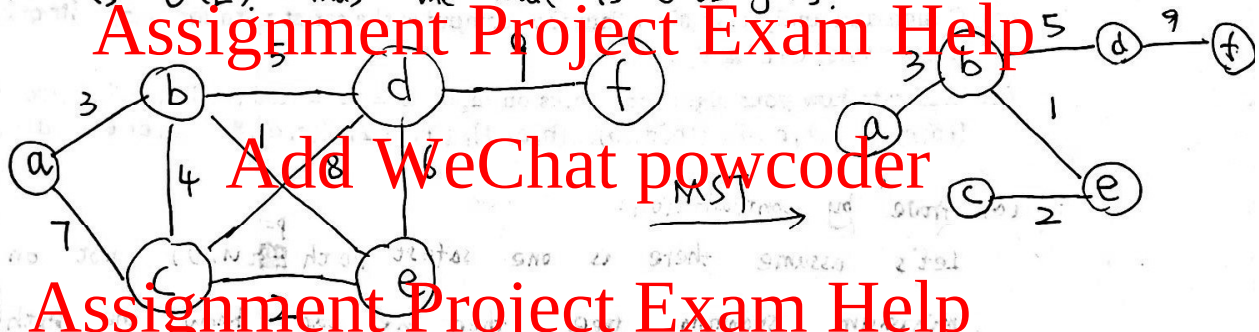
For the proof in (1), we know the shortest path must on the MST. Thus we connected the edges for (u, v) can get a path containing the shortest edge, which corresponding to the safest path.

Complexity:

Build MST will take $O(E \log V)$. The worse case to traversal path is $O(E)$. Thus the total is $O(E \log V)$.

(10)

(3)



In the example, the results of my designed algorithm should be:

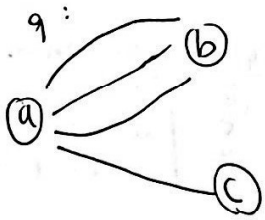
(Notation: $S(u,v)$ records for the safest path for pair (u,v))

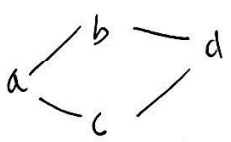
Safest path	smallest danger level
$S(a,b) = a-b$	3
$S(a,c) = a-b-e-c$	1
$S(a,d) = a-b-d$	3
$S(a,e) = a-b-e$	1
$S(a,f) = a-b-d-f$	3
$S(b,c) = b-e-c$	1
$S(b,d) = b-d$	5
$S(b,e) = b-e$	1
$S(b,f) = b-d-f$	5
$S(c,d) = c-e-b-d$	1
$S(c,e) = c-e$	2
$S(c,f) = c-e-b-d-f$	1
$S(d,e) = d-b-e$	1
$S(d,f) = d-f$	9

(10)

we can easily see all these danger level in the path contain smallest danger level edge.

$S(e,f) = e-b-d-f$ 1





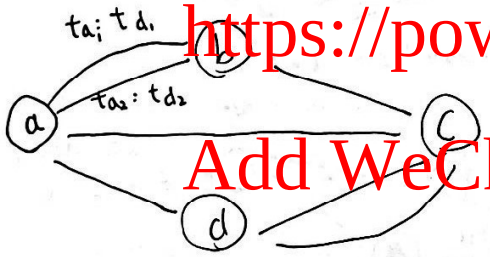
15/25

3. [25 points] SHORTEST TRAVEL TIME BETWEEN TWO AIRPORTS
Let $G = (V, E)$ be a directed graph, where the vertices represent airports and the edges represent flights between them, given together with the departure and arrival times. There could be multiple flights between two cities with different times. Design and analyze an algorithm, given $u, v \in V$, finds a sequence of flights from u to v that minimizes the total transportation time, i.e., the length of time from the initial departure to the final arrival, including the connection times at intermediate airports. The connection time between any two flights must be at least 30 minutes to allow the transfer. Obviously, non stop travels have no connection time.

-10 Algorithm description is lacking.

For this shortest time problem, we can deal like the shortest path problem but need some modification for the judgement of whether we can add that edge to the set of edges which we have already determined the shortest travel time.

Notation: Let t_d be departure times and t_a be arrival times.



For this problem, we can use Bellon-Form method, i.e. Dynamic programming to solve, since greedy method will cut off so many leaves, which will occur to miss the minima.

Algorithm: (u, v)

① Initial list T_a for storage of departure time, T for travel time of elements have checked, S for all the place vertices.

② Loop for start from u, find the time by $(t_d - t_a)$, add to T, add t_a to T_a. Then check the next edge whether $t_a' > t_{d_1} + 0.5h$, if so, add t_2 to t, store in T.

Recursively. $O(|V||E|)$ complexity.

Initial Flow	Augmentation p	$c_f(p)$	Final Flow
0	s - a - d - t	5	5
5	s - b - d - t	2	7
7	s - c - d - t	4	11

4. [10+5+5+10+10 = 40 points] FLOW NETWORKS
Consider the flow network $G = (V, E)$, where $V = \{s, a, b, c, d, t\}$, s is the source, t is the sink (target), and E is given together with the capacity of each edge: $\{((s,a),5), ((s,b),3), ((s,c),4), ((a,d),5), ((a,t),7), ((b,d),2), ((c,d),4), ((c,t),3), ((d,t),11)\}$.

(a) Draw G. Find a max-flow f and a min-cut on G. Draw the residual graph G_f.

(b) An edge of a network is called a *bottleneck edge* if increasing its capacity alone results in an increase in the maximum flow. List all bottleneck edges in G.

(c) Can there be a flow network which has no bottleneck edges? Justify your answer.

(d) Design and analyze an efficient algorithm to identify all bottleneck edges in a network.

(e) Show on G that, counter-intuitive to its better time complexity bound, the Edmonds-Karp algorithm might use more flow augmentations than the Ford-Fulkerson algorithm.

35

(a)

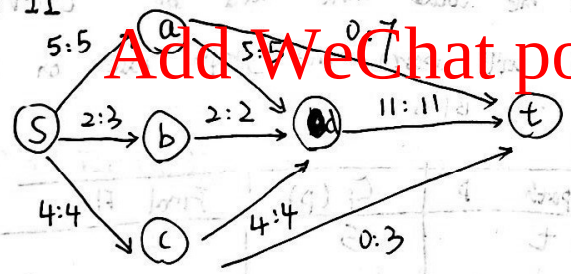
min cut: 5 + 2 + 4 = 11

$S = \{s, b\}$

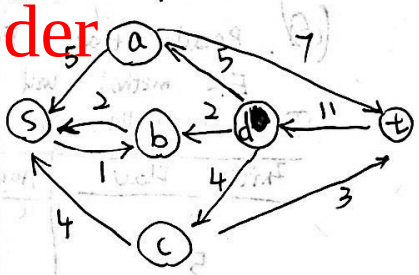
$T = \{a, c, d, t\}$

we can find the max-flow by Ford-Fulkerson method, which matches our min-cut.

G Max-Flow: 11



Residual graph G_f:

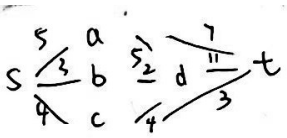


(b) bottle neck edges in this graph:

(a, d), (b, d), (c, d)

(c) Yes. There can be a flow network without bottle neck edges.

5 The simplest example can be



(d) we can find the leftside of bottle neck ~~edge should~~ edge should be reachable to source point s, and the rightside of bottleneck edge should be reachable to sink point t. we can have an algorithm as following:

① Find the max-flow of the graph by Edmonds-Karp Algorithm.

② Build residual network G_f.

③ Do a BFS to obtain set A which contains vertices which are reachable to s.

④ Build reverse residual network G_f^R.

⑤ Do a BFS to obtain set B which contains vertices which are reachable to t.

⑥ output combination (u, v) for which $u \in A$ and $v \in B$.

Analysis:

Find the max flow will take $O(VE^2)$, for the BFS step will take $O(|V|+|E|)$ so the totab time will be $O(VE^2)$

(e) Recall that FF method will search the graph based on residual network, EK method will search by BFS.

① Ford-Fullkerson Algorithm

Initial Flow	Augmentation path p	$C_f(p)$	Final Flow
0	s - a - t	5	5
5	s - b - d - t	2	7
7	s - c - d - t	4	11

② Edmonds-Karp Algorithm.

Initial Flow	Augmentation path p	$C_f(p)$	Final Flow
0	s - a - t	5	5
5	s - c - t	3	8
8	s - b - d - t	2	10
10	s - c - d - t	1	11

Thus the augmentations of Edmonds-Karp is more than Ford-Fullkerson.