

Final Solution

Name:

SID:

Exam Room:

SID of student to your left:

SID of student to your right:

Do not turn this page until your instructor tells you to do so.

1 True or False? (11 points)

Bubble in the right answer. No explanation needed. No points will be subtracted for wrong answers, so guess all you want! This part will be graded automatically. Please mark your answer clearly.

1. For every real $0 < x \leq 1$, $\sum_{i=1}^n x^i$ is $O(1)$ in terms of n asymptotically.

True

False

False

2. Dijkstra's algorithm (without any modification) can be used to compute the shortest path between 2 vertices in a DAG with exactly one edge having a negative weight (all other edges have a positive weight).

True

False

Everyone gets points. The problem can be interpreted in two ways. (1) If we call Dijkstra's algorithm only once at the starting vertex, then it may not give the right answer. (2) On the other hand, if we are allowed to run Dijkstra's multiple times, we can run it once at the starting node and once at the node for which the negative-edge weight originates from, and use the information produced to compute the correct answer.

3. A bipartite graph can contain a simple cycle that consists of an odd number of unique vertices.

True

False

False

4. If a bipartite graph has a perfect matching of size n , then the minimum vertex cover of this graph must be also be of size equal to n . (A perfect matching is a bijective pairing of vertices).

True

False

True

5. It is possible that one-way permutations exist and $P = NP$.

True

False

False

SID:

6. For every odd prime p and every integer $a \in \{1, \dots, p-1\}$, $a^{(p-1)/2}$ is either 1 or -1 modulo p .

True

False

True

7. It is possible that problem A reduces to problem B, problem B reduces to problem C, problem C reduces to problem A, problem A is NP-complete and problem B is in P. (Assuming $P \neq NP$)

True

False

False

8. Doubling the capacities of all edges of a graph G doubles the maximum flow.

True

False

True

9. Factoring is known to be NP-hard.

True

False

False

10. Consider a graph with exactly one edge having rational capacity and all other edges having integral capacities. The max flow in this graph must be integral.

True

False

False

11. The maximum weight edge in a cut of the graph can never be a part of a MST.

True

False

False

2 Fill in the Blanks (20 points)

When asked for a bound, always give the tightest bound possible. Some questions have choices in parentheses after the answer box

- (4 points) The node that receives the highest (pre/post) number in a depth-first search must lie in a (source/sink) strongly connected component.
- (2 points) If an undirected graph with no duplicate edges has a cycle then it must have at least edges.
- (2 points) The space complexity of the Floyd-Warshall algorithm (all-pairs shortest path dynamic programming algorithm) is , where n and m are the number of vertices and edges in the graph, respectively.
- (2 points) Let $x = \text{BERKELEY}$ and $y = \text{BARKELY}$ and $E(i, j)$ be the edit distance between $x[1 \dots i]$ and $y[1 \dots j]$. Then, $E(2, 3) =$.
- (2 points) Continuing from above, $E(8, 7) =$.
- (2 points) A degree d polynomial is uniquely characterized by its values at any points.
- (2 points) A directed graph has a cycle if and only if its depth-first search reveals a (tree/forward/back/cross) edge.
- (2 points) The size of the maximum flow in a network is always ($<$, \leq , $=$, \geq , $>$) the capacity of any (s, t) -cut.
- (2 points) In union/find data structure of n items, if we use union by size without path compression then a any combination of m unions and/or find operations takes at most time.

3 Zero-Sum Games (9 points)

Consider a zero-sum game given by the following matrix (indicating the payoffs to the row player)

	C_1	C_2	C_3
R_1	-2	3	1
R_2	3	-1	2
R_3	-1	4	-1

Suppose the column player has fixed the following probabilistic strategy:

C_1	C_2	C_3
$\frac{1}{3}$	$\frac{1}{3}$	$\frac{1}{3}$

what is the best strategy for the row player?

R_1	R_2	R_3
0.0	1.0	0.0

The value the row player gains from choosing R_1 is $2/3$, from R_2 is $4/3$, and R_3 is $2/3$, so the row player should choose R_2 always.

If row player chooses to play as above, then instead of the above (uniformly random) column player strategy what would have been the best strategy that the column player could have taken?

C_1	C_2	C_3
0.0	1.0	0.0

The value the column player gains from choosing C_1 is -3, from C_2 is 1, and C_3 is -2, so the column player should choose C_2 always.

SID:

4 Mind Reading Strategy (20 points)

Alice has chosen a number x from $\{1, \dots, n\}$ and Bob is trying to guess the number. Bob can ask Alice questions of the form: Is $x \geq i$? for some $i \in \{1, \dots, n\}$. For each such asked question, Alice answers YES or NO. Let p_i denote the probability that Alice's chosen number is i . (Note that $\sum_{i=1}^n p_i = 1$.)

Bob knows these probabilities, and would like to devise a strategy to determine Alice's number correctly while minimizing the expected number of questions asked.

Given the probabilities p_i as input, design a dynamic programming algorithm to compute the expected number of questions needed for the best possible strategy for Bob.

1. Clearly define a subproblem that could be used to efficiently solve this problem.

2. What are the base cases?

3. What is the recurrence relation?

1. Let $Q(a, b)$ be a $n \times n$ matrix that represents the expected number of questions Bob has to ask in order to figure out which of the numbers $\in [a, b]$ is correct.

2.

$$Q(a, a) = 0 \text{ for all } a \text{ from } 1 \text{ to } n$$

It takes 0 guesses for Bob to figure out what Alice's number is if he's narrowed down his search space to a single number.

3. The expected number of guesses that Bob has to make in order to figure out Alice's number from $\{a \dots b\}$ is one plus the expected number of questions he has to ask after asking the best question possible AKA the question which minimizes the expected number of questions he has to ask down the line. For asking the best question, if Bob has narrowed his search space down to $\{a \dots b\}$, the only intelligent questions he can ask are: Is $x \geq k$? for k from $a + 1$ to b . That means the recurrence relation will take a minimum over the values $a + 1$ to b .

$$Q(a, b) = 1 + \min_{k \in \{a+1 \dots b\}} E[\text{remaining number of questions after guessing } k]$$

For computing the expected number of questions down the line, we take an expectation over Alice's responses to Bob's question. That is to say,

$$E[\text{remaining number of questions after guessing } k] = P(\text{Alice says YES}) \cdot Q(k, b) + P(\text{Alice says NO}) \cdot Q(a, k-1)$$

The probability that Alice says YES can be computed by summing up the probabilities that Alice chose a number $\geq k$ and dividing by the sum of the probabilities that Alice chose a number in $[a, b]$. The probability that Alice says NO, then, is one minus this quantity.

$$P(\text{Alice says YES}) = \frac{\sum_{i=k}^b p_i}{\sum_{i=a}^b p_i}$$

$$P(\text{Alice says NO}) = 1 - \frac{\sum_{i=k}^b p_i}{\sum_{i=a}^b p_i}$$

Solution 1 - DP

Start at a coordinate, go to the neighbor with the smallest height value. Traverse until you reach an end coordinate (smallest height among neighbors). Once you have found this coordinate, backtrack on this path and for each coordinate, set their destination to be the end coordinate found above.

Continue and move on to the next vertex. The traversal will only start at a coordinate if the destination coordinate has not been calculated yet. If the destination has already been set, then stop the traversal and for all coordinates on the current path, set their destination coordinates to the destination of your current position.

This ensures that you do not re-compute paths. Each vertex is visited at most three times: reached from another vertex, backtracked to set destination, potentially starting a traversal.

Runtime $\Theta(N^2)$

Solution 2 - Graph Traversal

We can create N^2 vertices to represent the coordinates. Connect a directed edge (v, u) if v is a neighbor of u and has the lowest height. The source nodes of this graph are the end coordinates. Run DFS from these source nodes. For each DFS tree, set the destination coordinate of all nodes in that tree to be the coordinate of the root. Runtime $\Theta(N^2)$ due to graph construction.

Solution 3 - Graph Traversal 2

Create vertices as above. Connect a directed edge (u, v) if v is a neighbor of u and has the lowest height. Run recursive DFS from each vertex if that vertex does not have a destination set yet. Have the recursive call return the coordinate of the sink. In the function $post(u)$ after the recursive call, set u 's destination to be the return value of the recursive call, which would be the coordinate of the sink. This is exactly like the DP solution but wrapped around a data structure.

The idea is to add up max profit of $S[1..i]$ using one transaction and max profit $S[i+1..N]$ using one transaction. However we can do this in $O(N)$ time instead of $\Theta(N^2)$ time.

- **Solution**

Idea is to create 2 arrays. One contains best profit up to day i with 1 transaction and the other contains best profit from day i .

- Fill in $A[]$ as best profit with 1 transaction from day i by traversing in reverse order. Keep and update a running minimum. At $A[i]$ take larger of $A[i+1]$ or profit with running minimum ($S[i] - \mathbf{running-min}$).
- Fill in $B[]$ as best profit up to day i by traversing in order. Keep and update a running maximum. Update $B[i]$ by taking larger of $B[i+1]$ or profit with running maximum.
- Iterate i and compute best profit of $A[i] + B[i]$. This will contain at most two transactions.

We could use only one array by skipping steps 3 and 4 and find best profit directly by traversing $A[]$ in order, keep a running maximum, and update $A[i]$ accordingly.

Runtime $\Theta(N)$

Rubric Explanation

- Minimal partial for $\Theta(N^2)$.
- More partial credit for a solution that is not optimal, but goes in the right direction of linear.
- If you solve part (b) and specified to solve part (a) with $k = 2$, you will be award full credit if you part (b) runtime is $\Theta(NK)$. However, this is achieved in the implementation level, not the recurrence level. Thus to get full credit here, you must explain how to get to linear from part (b). Otherwise it will be graded like above.

SID:

2. **k trades (20 points)**

Now find the largest profit if we can short sell **at most** K times. Again you cannot interleave sell and buy actions. It must be Sell Buy Sell Buy ...

(a) What are the subproblems?

(b) What are the base cases?

(c) What is the recurrence relation?

(d) What is the runtime in Θ notation?

Let $P(t, i)$ be maximum profit using at most t transactions up to and including day i .

$$P(t, i) = \max \begin{cases} P(t, i - 1) & \text{no transaction on day } i \\ \max_{j=1..i} S[j] - S[i] + P(t - 1, j - 1) & \text{best transaction on day } i \end{cases}$$

$$P(0, i) = 0$$

$$P(t, 0) = 0$$

Runtime is $\Theta(kn^2)$. Can optimize by buying shares on i -th day in constant time.

We will also accept $\Theta(kn)$ solution, but it is not needed for full credit for this part.

One way to achieve $\Theta(kn)$ is to create an array of running maxes of $S[j] + P(t - 1, j - 1)$. We do this in the inner loop when we update i by 1. At this update, we can append the next running max of $S[j] + P(t - 1, j - 1), j = i$ to the array. Thus when we no longer need a third loop for the inner max.

Another way to see this is to rewrite the inner max loop.

$$\begin{aligned} \max_{j=1..i} S[j] - S[i] + P(t - 1, j - 1) &= \max_{j=1..i} \{S[j] + P(t - 1, j - 1)\} - S[i] \\ &= \max \{ \max \{P[t - 1][j] + S[j]\}, S[i - 1] + P(t - 1, i - 1)\} - S[i] \quad \forall j \in [1..n - 1] \end{aligned}$$

That inner max can be computed in constant time in the inner loop as it will serve as a running-max.

7 NP-Completeness Reductions (38 points)

Show that the following problems are NP-complete by providing a polynomial-time reduction. You may skip showing that the given problem is in NP. You may assume that the following problems are NP-complete: Rudrata Path or Hamiltonian Path, Hamiltonian Cycle, Vertex Cover, Independent Set, 3-SAT, CircuitSAT, Integer Linear Programming, Clique, 3D Matching, Partition, and Subset Sum.

1. Set Packing. (8 points)

Input: Subsets S_1, \dots, S_m of a set U and a positive integer $k \leq m$.

Question: Are there k subsets among S_1, \dots, S_m that are mutually disjoint (i.e. the intersection of any two subsets are empty)?

Proof: We will reduce the problem **A** ...

to the problem **B** ...

Given an instance Φ of the problem **A** we construct an instance Ψ of the problem **B** as follows ...

The proof that this is a valid reduction is as follows:

Proof: We can reduce from 3D Matching. Given an instance Φ of 3D Matching, we can construct an instance Ψ of Set Packing as follows. Let the set $U = \{p_1, \dots, p_n, b_1, \dots, b_n, g_1, \dots, g_n\}$ be the set of all pets, boys and girls in Φ and let $k = n$. For each tuple (p_i, b_j, g_k) , we create one set $S_\ell = \{p_i, b_j, g_k\}$. The reduction clearly runs in polynomial time and it is obvious that a collection of tuples is a valid matching for Φ if and only if their corresponding sets is a valid (set packing) solution of Ψ .

2. **Bin Packing. (10 points)**

Input: n items with sizes $a_1 \cdots a_n$ respectively, a positive integer B (bin capacity) and a positive integer k (number of bins).

Question: Is there a partition of the set $\{1 \cdots n\}$ into sets S_1, \dots, S_k such that for each $i \in \{1 \cdots k\}$ we have that $\sum_{j \in S_i} a_j \leq B$?

Proof: We will reduce the problem **A** ...

to the problem **B** ...

Given an instance Φ of the problem **A** we construct an instance Ψ of the problem **B**

as follows ...

The proof that this is a valid reduction is as follows:

Proof: We will reduce from Partition to Bin Packing. Given an instance Φ of Partition we construct an instance Ψ of Bin Packing as follows. Given items of size $a_1 \dots a_n$ in Φ , create items of the same sizes in Ψ . Also, let $k = 2$ and let $B = \frac{\sum_i a_i}{2}$ in Ψ .

We can argue the correctness of the reduction as follows.

--

- If there is a partition $(P, S \setminus P)$ of the elements $S = \{a_1 \dots a_n\}$ in Φ such that $\sum_{i \in P} a_i = \sum_{i \in S \setminus P} a_i$, then the same partition satisfies the requirements of Ψ , which are to find a partition of $S = \{a_1 \dots a_n\}$ into $k = 2$ subsets such that the sum of the elements in each is $\leq B = \frac{\sum_i a_i}{2}$. This is because both $\sum_{i \in P} a_i$ and $\sum_{i \in S \setminus P} a_i$ are equal to (and therefore \leq) $\frac{\sum_i a_i}{2}$.
- If there is a partition of the elements $S = \{a_1 \dots a_n\}$ in Ψ into $k = 2$ subsets (S_1, S_2) such that $\sum_{i \in S_1} a_i \leq B$ AND $\sum_{i \in S_2} a_i \leq B$, then this same partition satisfies the requirements of Φ , which are to find a partition $(P, S \setminus P)$ of S such that $\sum_{i \in P} a_i = \sum_{i \in S \setminus P} a_i$, because by partitioning S into two subsets (S_1, S_2) where the sum of the elements in either is $\leq \frac{\sum_i a_i}{2}$, we must have the sum of the elements in each is exactly $B = \frac{\sum_i a_i}{2}$. Otherwise, if one of them was not, the sum of the elements in one of the sets would be greater than B .

3. Quadratic Assignment Problem. (10 points)

Input: Integers n and b , two $(n \times n)$ integer-value matrices $C = \{c_{ij}\}_{1 \leq i, j \leq n}$ and $D = \{d_{kl}\}_{1 \leq k, l \leq n}$.

Question: Is there a permutation $f : \{1, \dots, n\} \rightarrow \{1, \dots, n\}$ such that $\sum_{i=1}^n \sum_{j=1}^n c_{ij} d_{f(i)f(j)} \leq b$?

Hint: Rudrata Path/Cycle.

Proof: We will reduce the problem **A** ...

to the problem **B** ...

Given an instance Φ of the problem **A** we construct an instance Ψ of the problem **B**

as follows ...

The proof that this is a valid reduction is as follows:

Proof: We will reduce from Rudrata Path to Quadratic Assignment Problem (QAP). Given an instance $G = (V, E)$ of Rudrata Path where n is the number of vertices, label the vertices by numbers from 1 to n . We create an instance of QAP where the matrix D is the adjacency matrix of G , i.e., d_{ij} is equal to one if $(i, j) \in E$ and zero otherwise. As for the matrix C , we set $c_{i, i+1} = -1$ for all $i = 1, \dots, n - 1$ and $c_{ij} = 0$ for all $j \neq i + 1$. Finally, let $b = -(n - 1)$. Clearly, the reduction runs in polynomial time. ■

--

Observe that for the instance that we construct, since the only non-zero entries of C are $c_{i,i+1}$'s which are equal to -1, $\sum_{i=1}^n \sum_{j=1}^n c_{ij} d_{f(i),f(j)}$ can be simplified to $-\sum_{i=1}^{n-1} d_{f(i),f(i+1)}$.

Suppose that there is a Rudrata path v_1, \dots, v_n . Then, for the permutation f defined by $f(i) = v_i$, we have $-\sum_{i=1}^{n-1} d_{f(i),f(i+1)} = -(n-1) = b$ as desired.

On the other hand, if there exists a permutation f such that $\sum_{i=1}^{n-1} d_{f(i),f(i+1)} \geq n-1$, then we must have $d_{f(1),f(2)} = \dots = d_{f(n-1),f(n)} = 1$ since each number is either zero or one. From our definition of D , this means that $f(1), \dots, f(n)$ form a Rudrata path in G .

4. **Induced Path. (10 points)**

Input: An undirected graph $G = (V, E)$ and an integer k .

Question: Is there an induced path of length k (i.e. a sequence of k distinct vertices v_1, \dots, v_k such that there exists an edge in G between every pair of consecutive vertices and for every pair of non-consecutive vertices there does not exist an edge in G)?

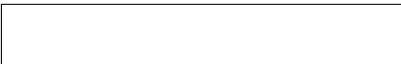
Hint: Try to modify the reduction from 3SAT to Independent Set, in which you create a vertex for every literal in every clause, so that which vertex is chosen in the independent set means which literal is set to true for the clause. Try to follow a similar strategy and think about how to connect edges, and make use of the fact that these are induced paths rather than just paths. There are several constructions that work. You may also try to create two vertices for every literal in every clause instead of just one.

Proof: We will reduce the problem **A** ...

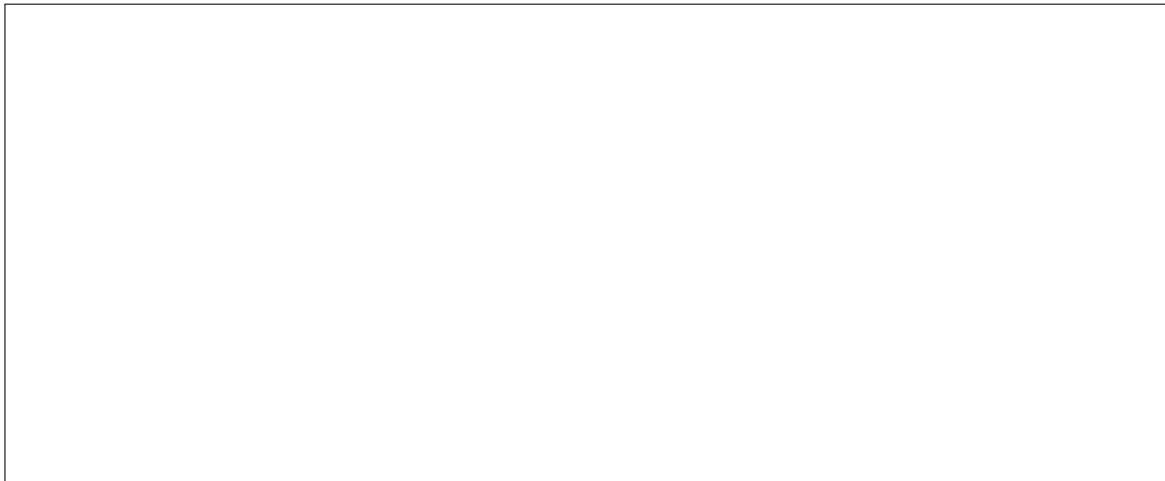
to the problem **B** ...

Given an instance Φ of the problem **A** we construct an instance Ψ of the problem **B**

as follows ...



The proof that this is a valid reduction is as follows:



We will reduce from 3SAT to Induced Path. Given a 3SAT formula Ψ , we create an instance $(G = (V, E), k)$ of Induced Path as follows. Suppose that the clauses of Ψ are C_1, \dots, C_m . Let $k = 2m$. For each clause C_i , create two vertices for each literal in C_i , and create a complete graph between them (i.e. there are edges for every two of the vertices). Now, similar to the construction for Independent Set, we join every pair of vertices that correspond to literals that are negation of each other. Finally, for every $i = 1, \dots, m - 1$, we connect the second copy of each literal in C_i to the first copy of each literal in C_{i+1} .

Clearly, this reduction runs in polynomial time. Moreover, we can argue its correctness as follows.

- Suppose that there exists an assignment that satisfies Ψ . Let b_i be a literal that is set to true in the clause C_i . Then, the following is an induced path of length $k = 2m$ in graph G : the first copy of b_1 , the second copy of b_1 , the first copy of b_2 , the second copy of b_2 , \dots , the first copy of b_m , and the second copy of b_m .
- Suppose that there exists an induced path $v_1 \dots v_{2m}$ in the graph G . First, notice that there can be no three vertices from the same clause; otherwise, they would induce a triangle which cannot appear in an induced path. Hence, exactly two vertices of v_1, \dots, v_{2m} come from each clause C_i . Notice that vertices of the same clauses are linked by an edge, meaning that the two vertices from the same clauses must appear consecutively in the induced path. This implies that v_1, v_2 are from the same clause, v_3, v_4 are from the same clause, \dots , and v_{2m-1}, v_{2m} are from the same clause. Hence, if we consider $v_1, v_3, \dots, v_{2m-1}$, then this is an independent set with exactly one vertex from each clause. Due to a similar argument we used in the reduction from 3SAT to Independent Set, this can be converted in polynomial time to a satisfying assignment for the 3SAT formula Ψ .

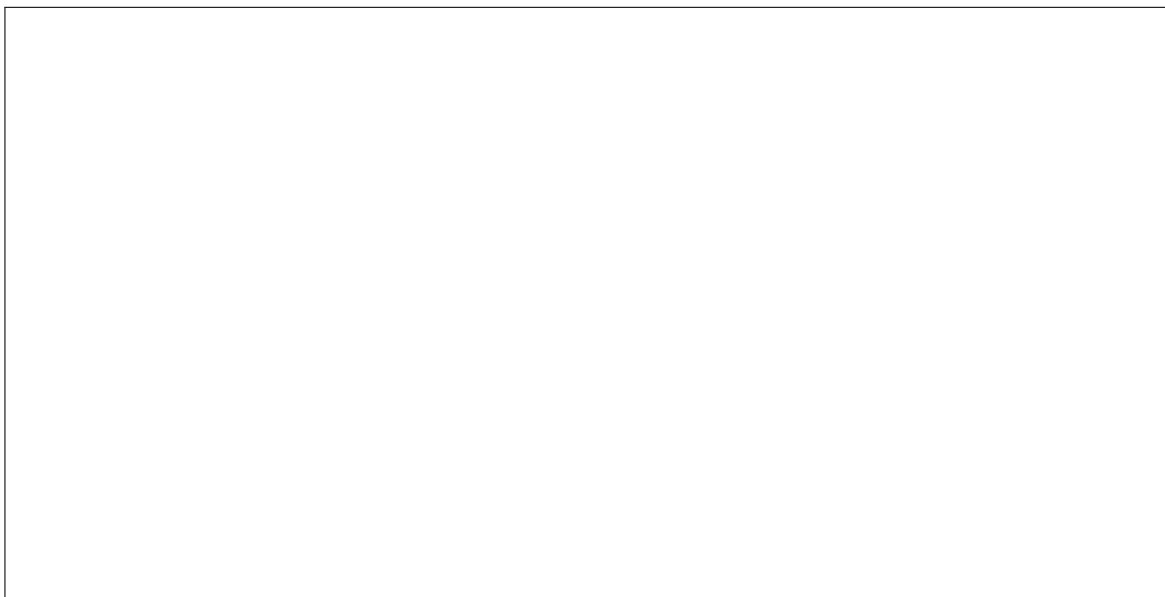
8 Approximation: Maximum Coverage (22 points)

The Maximum Coverage Problem is an optimization problem defined as follows.

Input: Subsets S_1, \dots, S_m of a set \mathcal{U} , and a positive integer $k \leq m$.

Output: Find $T \subseteq \{1, \dots, m\}$ of size k such that the size of $\bigcup_{i \in T} S_i$ is maximized. (I.e., find k subsets whose union has maximum size.) We say that the elements in this union are *covered* by the subsets.

1. (6 points) Briefly argue that, if we can solve the Maximum Coverage problem optimally, then we can also solve the Set Cover problem. (Note: this implies that Maximum Coverage is NP-hard.)



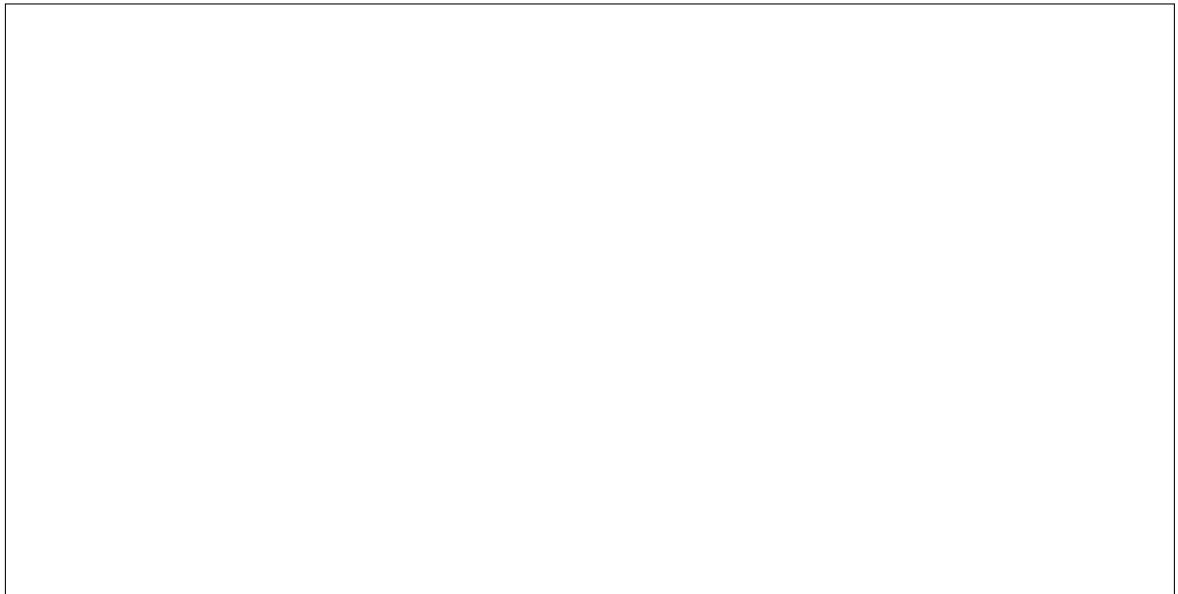
If we can solve Maximum Coverage, then we can run the algorithm (on the same input for the Set Cover problem) and check if the output subsets cover every element. If they do, then these k subsets are a solution for Set Cover. Otherwise, we know that there is no solution of size k for Set Cover.

2. Let us examine the greedy algorithm which proceeds in k steps as follows: In each step, pick the subset that covers the maximum number of (currently) uncovered elements.
 - (a) (4 points) Give a counterexample for which the greedy algorithm does not give an optimal solution.



Consider the case where there are three subsets $S_1 = \{1, 2, 3\}$, $S_2 = \{4, 5, 6\}$, $S_3 = \{2, 3, 4, 5\}$ and $k = 2$. Greedy would pick S_3 in the first iteration, at which point no matter which other set is picked the union is not the whole set $\{1, 2, 3, 4, 5, 6\}$.

- (b) (6 points) Show that, when $k = 2$, this algorithm is a $3/4$ -approximation algorithm for the problem (i.e. the two sets picked by the algorithm covers $3/4$ as many elements as the optimal solution).
Hint: It may be helpful to look at the hint from part (c).



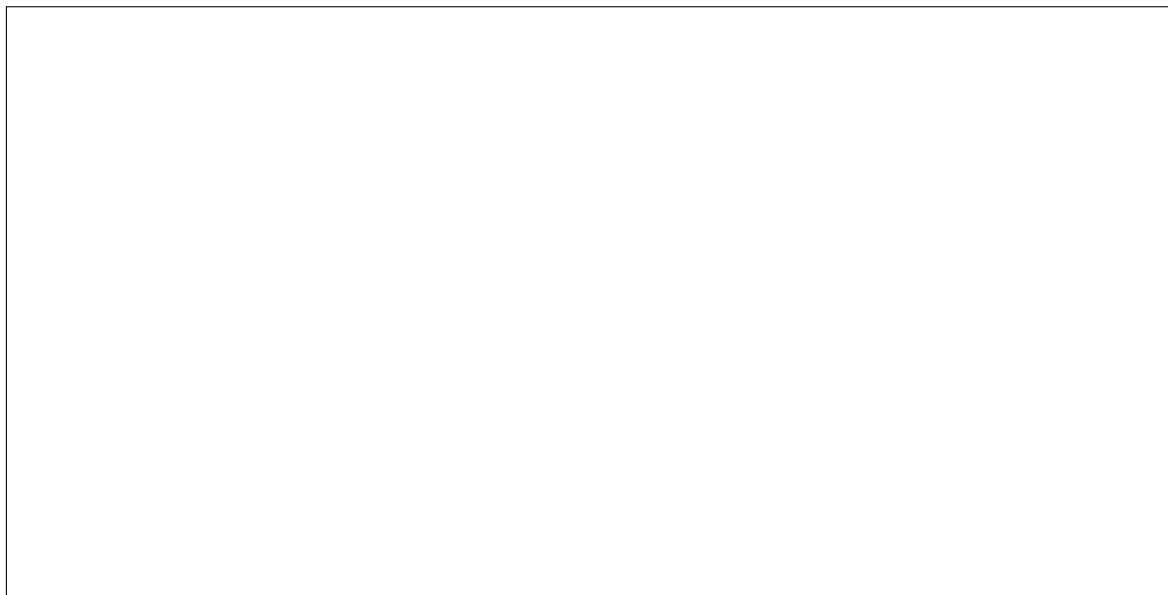
Let OPT denote the total number of elements covered by the optimal solution and let x denote the number of elements covered by the first subset in the greedy solution. Since OPT elements can be covered by two subsets, the first subset picked by the greedy algorithm must cover at least $OPT/2$ elements, i.e., $x \geq OPT/2$.

Let us now examine the second subset picked by the greedy algorithm. Since at least $OPT - x$ elements covered by the optimal solution are uncovered by the first subset picked by the greedy algorithm, the second subset picked must cover at least $(OPT - x)/2$ uncovered elements.

In conclusion, the greedy algorithm covers at least $x + (OPT - x)/2 = OPT/2 + x/2 \geq 3OPT/4$ elements.

- (c) (6 points) Generalize the analysis above to show that, for every k , the greedy algorithm is an $(1 - (1 - 1/k)^k)$ -approximation algorithm for Maximum Coverage.

Hint: Let x_i denote the number of elements covered by the greedy solution after picking i subsets. Try to show that $x_{i+1} - x_i \geq (OPT - x_i)/k$ where OPT is the total number of elements covered by the optimal solution. You might find it useful to consider the quantity $y_i = OPT - x_i$ as well.



Let x_i be as defined as in the hint. After the i -th subset is picked by the greedy algorithm, at least $OPT - x_i$ elements covered by the optimal solution have not been covered by the greedy solution. Since these elements are covered in the optimal solution (which uses k subsets), there must be one subset that covers at least $(OPT - x_i)/k$ such elements. Hence, the greedily picked subset also covers at least $(OPT - x_i)/k$ additional elements. In other words, we have that $x_{i+1} \geq x_i + (OPT - x_i)/k$.

Let $y_i = OPT - x_i$. The above inequality can be written as $y_i(1 - 1/k) \geq y_{i+1}$. This implies that $y_k \leq y_0(1 - 1/k)^k = OPT(1 - 1/k)^k$, which implies that $x_k \geq OPT(1 - (1 - 1/k)^k)$. As a result, the greedy algorithm is an $(1 - (1 - 1/k)^k)$ -approximation algorithm for Maximum Coverage.