Problem 1 (8 + 8 = 16 points).

Given an undirected graph $G = (V,E)$, the maximum independent set problem seeks an independent set $I \subset V$ such that $|I|$ is maximized. We define the degree of a vertex $v \in V$, denoted as $d(v)$ is the number of edges $(u,v) \in E$ incidental to v, i.e., $d(v) = |\{(u,v) \in E \mid v \in V\}|$. The degree $d(G)$ of a graph $G = (V,E)$ is the maximum degree of any of its vertices, i.e., $d(G) := \max_{v \in V} d(v)$.

- Suppose you are give an undirected graph G, a constant d, and a parameter k. Prove that for the class of graphs G whose degree is bounded by the given d, i.e., $d(G) \leq d$, there is an FPT algorithm such that (a) if there is a maximum independent set I with $|I| \leq k$, it computes a I, or (b) otherwise return NO. Design such an algorithm and analyze the correctness and complexity your algorithm.

  Solution: If the maximum independent set (IS) size is at most k, we can show that the total number of vertices is at most $k(d+1)$. This is because each vertex in the IS has at most d neighbors. All vertices in the graph should be either be in the maximum IS or be one neighbor of an IS vertex (otherwise this vertex can be added to the IS to get an IS of larger size, a contradiction). Thus, $|V| \leq k + dk = k(d+1)$.

  Here comes the algorithm: if number of vertices is more than $k(d+1)$, return NO; otherwise, we know that the total number of verticse is bounded by $k(d+1)$ which is a function of the parameter k (recall that d is a constant), so now we can use brute-force: we enumerate all subsets of vertices of size at most k, and for each subset we examine if it is an independent set. Clearly, this brute-force enumeration takes a running time of $f(k)$, which is not a function of n.

  Note: by definition, any solution works with part 2 also works with part 1.

- Now assume that $d(G)$ is not bounded by a constant, but both d and k are parameters. Design an FPT algorithm to (a) compute a maximum independent set I if there is a maximum independent set I with $|I| \leq k$, or (b) otherwise return NO. Your FPT algorithm should use k and d as parameters, i.e., run in $O(poly(n) \cdot f(d,k))$ time. Show the correctness and running time complexity of your algorithm.

  Solution: We can use bounded tree search technique to solve this part. For any vertex v_i in $V = \{v_1, v_2, ..., v_n\}$, either v_i or one of its neightbors should be in the maximum IS (otherwise v_i can be includede to obtain an IS of larger size). Define function bounded-search (G, k) either returns NO which shows that the size of the maximum IS is larger than k, or returns one maximum IS.

```
function bounded-search (G,k)
    If G is empty: return ∅;
    If k = 0: return NO;
    # body of recursion
    Pick any vertex v from V
    Denote all neighbors of v as N(v) := {u ∈ V | (u,v) ∈ E}.
    Use X to store the current maximum IS, initially set as NO.
    Y = bounded-search (G\({N(v)} ∪ {v}), k − 1);
    if Y ≠ NO and |Y| + 1 > |X|: X ← Y ∪ {v}
    for each v′ in {N(v)}:
        Y = bounded-search (G\({N(v′)} ∪ {v′}), k − 1)
        if Y ≠ NO and |Y| + 1 > |X|: X ← Y ∪ {v′}
    end for
    return X
end function;
```

In each iteration, we add either v or one neighbor v' of v into the IS, and then we remove all neighbors of v (or all neighbors of v') from the graph G. If the graph G is empty at the end of the iterations, it means all vertices of G are in the IS or are neighbors of the IS vertices. If $k = 0$ but G is not empty, it means there are still vertices not in the IS nor neighbors of the IS, so we return NO.

In each iteration, we have $(d+1)$ branches (1 branch for putting v in the IS, d branches for putting one of neighbors of v in the IS) and we have at most k iterations. Hence, the time complexity is $O((d+1)^k \cdot n^2)$ where n^2 is the time for graph operation in each branch, *i.e.,* removing a vertex and its neighbors and their adjacent edges to create a new graph.

Problem 2 (10 points).

Given an undirected graph $G = (V,E)$, we seek a cut $(A, B = V \setminus A)$ such that the number of cut-edges, denoted as $|E(A,B)|$ where $E(A,B) := \{(u,v) \in E \mid u \in A, v \in B\}$, is maximized. Consider the following greedy algorithm for this problem: process all vertices in V following an arbitrary order $\{v_1, v_2, \cdots, v_n\}$, and for the current vertex v_k, put it in B if $|N_A(v_k)| \geq |N_B(v_k)|$ and in A otherwise, where $N_A(v_k)$ is defined as $\{v_i \mid i < k, v_i \in A, (v_i, v_k) \in E\}$, and $N_B(v_k) := \{v_i \mid i < k, v_i \in B, (v_i, v_k) \in E\}$. Intuitively, this algorithm assigns a vertex to the different side with the majority of its adjacent vertices being already assigned to. Prove that this algorithm is an approximation algorithm. You will need to guess the approximation ratio, prove it, and design a tight example.

Solution: The approximation is 2. Since this algorithm assigns a vertex to the different side with the majority of its adjacent vertices being already assigned to, whenever a new vertex v_k is processed, at least half of the new edges are cut-edges. In other words, when we process a new vertex v_k, we are seeing $(N_A(v_k) + N_B(v_k))$ new edges, i.e. the number of neighbors of v_k which are already seen. Besides, $max(N_A(v_k), N_B(v_k))$ are cut edges, so $\frac{max(N_A(v_k),N_B(v_k))}{N_A(v_k)+N_B(v_k)} \geq \frac{1}{2}$ of the new edges are cut edges. This variant is guaranteed throughout the algorithm. Therefore, at the end, at least half of the total edges are cut edges, i.e. $Algo(G(V,E)) \geq \frac{1}{2}|E|$, where we use $Algo$ to denote this algorithm. Obviously, $OPT(G(V,E)) \leq |E|$, where OPT denotes the optimal algorithm, so $\frac{OPT(G(V,E))}{Algo(G(V,E))} \leq 2$.

Tight example: Given graph $G(V,E)$ where $V = \{v_1, v_2, v_3, \cdots, v_n\}$ and $E = \{(v_1, v_2)\} \cup \{(v_1, v_i), (v_2, v_i) \mid i = (3, 4, 5, ..., n)\}$. We pick the points in the order of $\{v_1, v_2, v_3, \cdots, v_n\}$. Then, we will have v_1 in B, v_2 in A, v_i in B, $i = (3, 4, 5, \cdots, n)$. Hence, the number of cut edges is $(n-2)+1 = n-1$. The max cut edge is $2 \cdot (n-2) = 2n-4$. The max is reached by putting v_1 and v_2 in A and all the other vertices in B. The limit of the ratio $\frac{OPT}{Algo}$ is $\lim_{n\to\infty} \frac{2n-4}{n-1} = 2$. This example shows above analyis is asymptotically tight.

Problem 3 (8 + 8 = 16 points).

Consider the following *bin packing* problem: given n items of sizes $\{a_1, a_2, \cdots, a_n\}$, find a way to pack them into unit-sized bins so that the number of bins needed is minimized. Consider a greedy algorithm: process all items in an arbitrary order; for the current k-th item, if there exists a partially packed bin that can further fit the k-th item, then put it in that bin; otherwise, open a new bin and put the k-th item in it.

- Show that the approximation ratio of above greedy algorithm is at most 2.

  Solution: After putting all of the n items into m bins, at least $m-1$ bins are more than half full. Because if there are two bins less than half full, the items in one bin must be put into another bin following the greedy algorithm. If m is odd, i.e., $(m-1)$ is even, then the optimal solution must use at least $(m-1)/2+1$ bins: $OPT \geq (m-1)/2+1$; if m is even, i.e., $(m-1)$ is odd, then the optimal solution must use at least $m/2$ bins: $OPT \geq m/2$. In any case, we can show that $m/OPT \geq 2$.

- Design an instance such that above algorithm performs at least as bad as $5/3 \cdot OPT$.

  Solution: Suppose we have 6 items with size 0.15, then 6 items with size 0.34, and finally 6 items with size 0.51. Using the greedy algorithm, there will be 10 bins: 1 bin for the first 6 items, 3 bins for the following 6 items and 6 bins for the final 6 items. The optimal answer is 6 bins with each bin contain one 0.15 item, one 0.34 item and one 0.51 item. The ratio for this instance is $5/3$.

Problem 4 (10 points).

Consider the optimization version of the 3D matching problem: given disjoint sets X, Y, Z, and a set of triples $E \subseteq X \times Y \times Z$ (you may assume that $|X| = |Y| = |Z| = n$), to find a matching $M \subseteq E$ such that $|M|$ is maximized. Design a 3-approximation algorithm for this problem: describe your algorithm, prove that the approximation ratio of your algorithm is 3, and give an example to show that your analysis is tight. *Hint:* consider a greedy strategy.

Solution: We can design a simple greedy algorithm for this problem. We randomly pick one triple (x_0, y_0, z_0) from E, then remove all the triples in E that contain x_0, y_0, or z_0, until all the triples in E are picked or removed. We now prove the approximation ratio of this algorithm is 3. Let M be the matching returned by this greedy algorithms. We call all elements in M as *marked*, i.e., there are $3|M|$ marked elements. For any triple that are not picked by the algorithm (i.e., not in M), it must contain at least one marked element; otherwise this triple will be later on picked by the greedy algorithm. Consider the optimal matching OPT: each triple in M must contain a marked element, and two triples in OPT cannot share any element, in particular, cannot share any marked element. Since the number of marked element is at most $3|M|$, we have that $OPT \leq 3|M|$.

Tight example: There are four triples in this instance: $(x_0, y_0, z_0), (x_0, y_1, z_1), (x_2, y_0, z_2)$ and (x_3, y_3, z_0). The answer of the greedy algorithm will be one if it selects (x_0, y_0, z_0). The optimal answer could be $OPT = \{(x_0, y_1, z_1), (x_2, y_0, z_2)\ (x_3, y_3, z_0)\}$.

Problem 5 (8 + 8 = 16 points).

Consider a minimal-weighted hitting set problem define as follows. We are given a set $U = \{u_1, u_2, \cdots, u_n\}$ and a collection $\{B_1, B_2, \cdots, B_m\}$ of subsets of U, i.e., B_j is a subset of U. Also, each element $u_i \in U$ has a weight $w_i \geq 0$. The problem is to find a *hitting set* $H \subset U$ such that the total weight of the elements in H is as small as possible, i.e., to minimize $\sum_{u_i \in H} w_i$. Note that we say H is a hitting set if $H \cap B_j \neq \emptyset$ for any $1 \leq j \leq m$. Let $b = \max_{1 \leq i \leq m} |B_i|$ be the maximum size of any of the sets $\{B_1, B_2, \cdots, B_m\}$.

- Design a b-approximation algorithm for above problem using the LP + rounding schema.

  Solution: Consider the ILP, using x_i to represent whether element a_i is in hitting set S or not:

$$min \sum_{i=1}^{n} w_i x_i$$

$$s.t.\ \ 0 \leq x_i \leq 1,\ i = 1, 2, ..., n; \quad \sum_{i: a_i \in B_j} x_i \geq 1,\ j = 1, 2, ..., m$$

  Let x be the solution of the LP-relaxation of this problem. Now define the set S to be all those elements where $x_i \geq 1/b$, i.e., $S = \{u_i | x_i \geq 1/b\}$.
  let x^* be the solution of this ILP problem and H be the min-weighted hitting set. (1) S is a hitting set. The set B_j contains at most b elements, and the sum of all x_i where $a_i \in B_j$. Therefore some $x_i > 1/b$ for some $a_i \in B_j$. Hence, S intersects with B_j by at least a_i.

(2) The total weight of all elements in S is at most bw_{LP}: for each $a_i \in S$, we know that $x_i > 1/b$, i.e. $1 < bx_i$, therefore:

$$w(S) = \sum_{a_i \in S} w_i \leq \sum_{a_i \in S} w_i b x_i \leq b \sum_{i=1}^{n} w_i x_i \leq b \sum_{i=1}^{n} w_i x_i^* = bw(H)$$

Thus, we have a hitting set S that is a b-approximation of this problem.

- Design a b-approximation algorithm for above problem using the primal-dual schema.

  Solution: LP:

$$min \sum_{i=1}^{n} w_i x_i$$

$$s.t. \sum_{i:a_i \in B_j} x_i \geq 1, j = 1, ..., m; \;\; x_i \geq 0, i = 1, ..., n$$

  Dual LP:

$$max \sum_{j=1}^{m} y_j$$

$$s.t. \sum_{j:a_i \in B_j} y_j \leq w_i, i = 1, ..., n; \;\; y_j \geq 0, j = 1, ..., m$$

  Let S be the current hitting set, y be the dual and let B_j be a set that has not been hit. Increase y_j until

$$\sum_{j:a_i \in b_j} y_j = w_i$$

  for some $a_i \in B_j$. Include a_i in S. Repeat until all sets are hit.

Problem 6 (10 points).

Recall that in Lecture 22 we designed an d-approximation algorithm using the LP + rounding technique for the weighted set cover problem. In that algorithm, we set $x'_j = 1$ if $x^*_j \geq 1/d$ and set $x'_j = 0$ if $x^*_j < 1/d$. Consider this (bold) rounding: set $x'_j = 1$ if $x^*_j > 0$ and set $x'_j = 0$ if $x^*_j = 0$. Prove that by using this bold rounding the resulting algorithm still guarantees an approximation ratio of d. *Hint:* consider the primal/dual complementary slackness conditions.

Solution: Above algorithm using LP following by bold rounding can be intepreted as an primal-dual algorithm with $\alpha = 1$ and $\beta = d$. Recall in class that, to achieve such a primal-dual approximation algorithm, we will need to find a feasible solution $\{x'_j\}$ of the ILP, a feasible solution $\{y_i\}$ of the dual-LP, such that they satisfy the following relaxed complementary slackness conditions (stated in the bottom of page 4 of lecture notes 23, copied below):

Relaxed primal conditions: for any $1 \leq j \leq m$: either $x'_j = 0$ or $\sum_{e_i \in S_j} y_i = w_j$;
Relaxed dual conditions: for any $1 \leq i \leq n$: either $y_i = 0$ or $\sum_{j:e_i \in S_j} x'_j \leq d$.

Again the relaxed dual conditions are naturally satisfied by the definition of d. The relaxed primal conditions can be proved using the Theorem of complementary slackness conditions for LP and dual-LP (on top of page 2, lecture notes 23). Specifically, in above algorithm, as $\{x^*\}$ and $\{y_i\}$ are optimal solutions of LP and dual-LP, we have

for any $1 \leq j \leq m$: either $x^*_j = 0$ or $\sum_{e_i \in S_j} y_i = w_j$.

According to the bold rounding from x_j^* to x'_j, if x_j^* then $x'_j = 0$; if $x_j^* > 0$, then above condition implies that $\sum_{e_i \in S_j} y_i = w_j$, which proves the above desired relaxed primal conditions (copied below).

for any $1 \le j \le m$: either $x'_j = 0$ or $\sum_{e_i \in S_j} y_i = w_j$.

Problem 7 (0 points).

Recall that in Lecture 20 we designed a greedy algorithm for the weighted set cover problem which guarantees an approximation ratio of $\Theta(\log n)$. Recall too that the vertex cover problem is a special case of the (weighted) set cover problem, and therefore this algorithm also gives an $\Theta(\log n)$-approximation algorithm for the vertex cover problem. Show that, the analysis for the ratio of $\Theta(\log n)$ is tight, i.e., you will need to design an instance of vertex cover problem such that when running this algorithm on that instance the ratio $|V_1|/OPT$ is $\Theta(\log n)$, where V_1 is the vertex cover returned by above algorithm, and n is the number of vertices in your instance.

Solution. The performance of above algorithm could be very poor on certain instance. Figure 1 gives such an instance, on which this algorithm finds a vertex cover with $m \log m$ vertices, while the optimal solution uses m vertices. This instance is a bipartite graph constructed as follows. The upper part consists of m vertices. The lower part consists of m groups of vertices, and the k-th group consists of m/k vertices, $1 \le k \le m$; each vertex in the k-th group connects to k vertices in the upper part, and two vertices in the k-th group won't connect to the same vertex in the upper part. Hence, every vertex in the upper part connects to a vertex in the k-th group once. In the resulting graph, the degree of each vertex in the upper part is m, and the degree of each vertex in the k-th group is k.

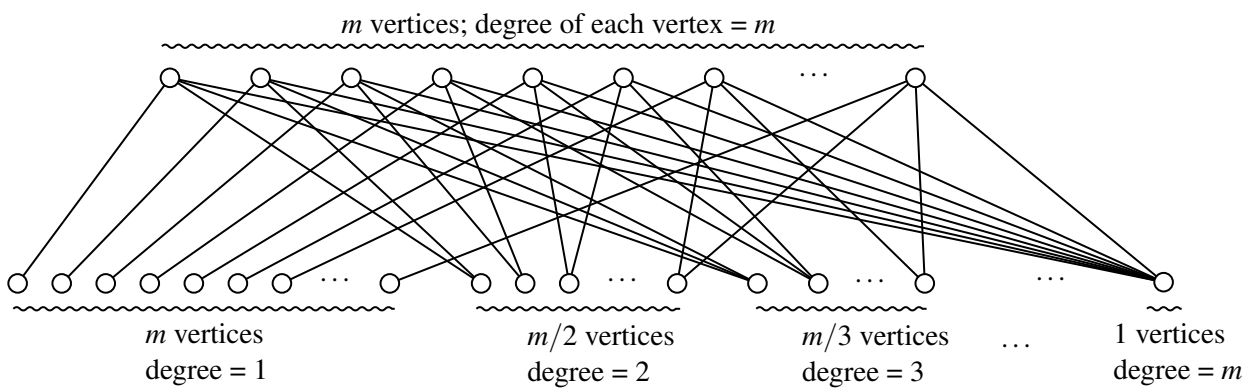

Figure 1: An instance on which above vertex-based approximation algorithm finds a vertex cover consists of all vertices in the lower part, while the optimal solution consists of all vertices in the upper part.

Let's run above greedy algorithm on this complex instance. Recall that the algorithm picks the vertex with largest degree in each iteration. Consider the first iteration, these m vertices in the upper part and the m-th group in the lower part (with 1 vertex) all have the largest degree of m. At a bad luck the algorithm picks the one in the lower part. After removing its adjacent edges, the vertices in the upper part have degree of $(m-1)$. Consider the second iteration, again the m vertices in the upper part and the $(m-1)$-th group in the lower part (with $\lfloor m/(m-1) \rfloor$ vertices) have degree of $(m-1)$. And assume that the algorithm picks the one(s) in the lower part. Clearly, if the algorithm always picks the vertices in the lower part to break tie, the solution returned by this greedy algorithm will include all vertices in the lower part: the total number of vertices is $m + m/2 + m/3 + \cdots + m/m = \Theta(m \log m)$. Notice that the optimal solution consists of all vertices in the upper part with m vertices. Hence, the ratio on this instance is $\Theta(m \log m/m) = \Theta(\log m)$.