Tutorial 11, Design and Analysis of Algorithms, 2019

1. Consider the complete graph of n vertices K_n in which there is an edge between any two distinct vertices (K_6 is shown in Figure 1). Each vertex of K_n has weight 1. Yor objective is to find the minimum vertex cover.

   (a) Give an *Integer Linear Programming* formulation for the given problem.

   (b) Convert your formulation in (a) into a *Linear Programming* formulation.

   (c) Solve the Linear Program in (b), giving a proof that your solution is indeed optimal.

   (d) Convert your solution in (c) into a vertex cover, giving the rule that you followed for the conversion.

   (e) Find the minimum vertex cover for K_n, and also prove your result.

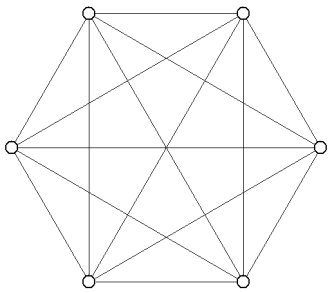

Figure 1: K_6: An example of a complete graph with 6 vertices.

2. Consider the following greedy algorithm for the knapsack problem. We initially sort all the items in order of non-increasing ratio of value to size so that $v_1/s_1 \geq v_2/s_2 \geq ... \geq v_n/s_n$. Let i^* be the index of an item of maximum value so that $v_i^* = \max_{i \in [1...n]} v_i$. The greedy algorithm puts items in the knapsack in index order until the next item no longer fits; that is, it finds k such that $\Sigma_{i=1}^{k} s_i \leq B$ but $\Sigma_{i=1}^{k+1} s_i > B$. The algorithm returns either $\{1, 2, ..., k\}$ or $\{i^*\}$, whichever has greater value. Prove that this algorithm is a 2-approximation algorithm for the knapsack problem. You are required to give complete proof without making any assumptions. You can assume that $s_i \leq B$ for each $i \in [1...n]$. Here B is the knapsack capacity, v_i is the profit of item i, and s_i is the size of item i for $i \in [1...n]$.

3. Suppose you are given a set of positive integers $A = \{a_1, a_2, ..., a_n\}$ and a positive integer B. A subset $S \subseteq A$ is called feasible if the sum of the numbers in S does not exceed B:

$$\sum_{a_i \in S} a_i \leq B$$

   The sum of the numbers in S will be called the total sum of S. You would like to select a feasible subset S of A whose total sum is as large as possible. For example, if $A = \{8, 2, 4\}$ and $B = 11$, then the optimal solution is the subset $S = \{8, 2\}$. Give a polynomial-time approximation algorithm for this problem with the following guarantee: it returns a feasible set $S \subseteq A$ whose total sum is at least half as large as the maximum total sum of any feasible set $S \subseteq A$. Your algorithm should have a running time of at most $O(n \log n)$.

4. Suppose you are given an $n \times n$ *grid graph* G, as in Figure 2.
Associated with each node v is a weight $w(v)$, which is a nonnegative integer. You may assume that the weights of all nodes are distinct. Your goal is to choose an independent set S of nodes of the grid, so that the sum of weights of the nodes in S is as large as possible. (The sum of the weights of the nodes in S will be called its *total weight.*)
Consider the following greedy algorithm for this problem.

```
The ‘‘heaviest-first’’ greedy algorithm:
  Start with S equal to the empty set
  While some node remains in G
    Pick a node v_i of maximum weight
    Add v_i to S
    Delete v_i and its neighbors from G
  Endwhile
  Return S
```

(a) Let S be the independent set returned by the "heaviest-first" greedy algorithm, and let T be any other independent set in G. Show that, for each node $v \in T$, either $v \in S$, or there is a node $v' \in S$ so that $w(v) \leq w(v')$ and (v, v') is an edge of G.

(b) Show that the "heaviest-first" greedy algorithm returns an independent set of total weight at least $\frac{1}{4}$ times the maximum total weight of any independent set in the grid graph G.

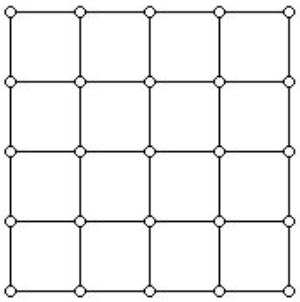

Figure 2: An example of a 5×5 grid graph.

5. We formulate the *Load Balancing Problem* as follows. We are given a set of m machines $M_1, ..., M_m$ and a set of n jobs; each job j has a processing time t_j. We seek to assign each job to one of the machines so that the loads placed on all machines are as "balanced" as possible. More concretely, in any assignment of jobs to machines, we can let $A(i)$ denote the set of jobs assigned to machine M_i; under this assignment, machine M_i needs to work for a total time of $T_i = \sum_{j \in A(i)} t_j$, and we declare this to be the load on machine M_i. We seek to minimize a quantity known as the makespan; it is simply the maximum load on any machine, $T = \max_i T_i$. Prove that the following algorithm is a $2-$approximation algorithm for the *Load Balancing Problem*:

```
Greedy-Balance:
Start with no jobs assigned
Set T_i = 0 and A(i) = φ for all machines M_i
For j = 1, ..., n
  Let M_i be a machine that achieves the minimum min_k T_k
  Assign job j to machine M_i
  Set A(i) ← A(i) ∪ {j}
  Set T_i ← T_i + t_j
EndFor
```

6. Let `MAX-3SAT` be the problem of finding, given a 3CNF Boolean formula Φ as input, an assignment that maximizes the number of satisfied clauses. Give a $2-$approximation algorithm for `MAX-3SAT`.

7. Consider the following instance of the *Weighted Vertex Cover Problem* with vertices $\{v_1, v_2, v_3, v_4, v_5, v_6\}$, having weight of vertices defined as $w(v_1) = 1, w(v_2) = 3, w(v_3) = 5, w(v_4) = 2, w(v_5) = 4, w(v_6) = 4$, for an undirected graph having the following adjacency matrix: (edge between v_i and v_j is lebelled as e_{ij})

$$\begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \end{bmatrix}$$

   (a) Formulate the above problem as a *Set Cover Problem*.

   (b) Using the *Primal Dual Approximation Algorithm* for the *Set Cover Problem*, solve the above instance of the *Set Cover Problem*.

   (c) Now, from the above solution, get back your original solution of the *Weighted Vertex Cover Problem*.