# CS3230
# Tutorial 11

1. Show that checking whether an undirected graph is 2-colorable is in $P$.

2. Consider the following modification of the Knapsack approximation algorithm we did in class, where $r > 1$ is a constant given to you:

   Knapsack
   Inputs:
   (1) A set $A$ of $n$ objects (with corresponding weights and values, which are assumed to be positive integers).
   (2) A knapsack weight $L$, which is a positive integer.
   (3) An optimality factor $r$ (a positive integer greater than one).
   Output: A subset $A'$ of $A$, such that $weight(A') \leq L$, and $value(A')$ is at least $(1 - \frac{1}{r})*$ optimal.
   1. Let $S_1, S_2, S_3 \ldots$, be all subsets of $A$ which have at most $r$ members.
   2. For each $S_j$, such that $weight(S_j) \leq L$, let $B_j$ be $B'$ given by algorithm Select, when the inputs to Select are $B = A - S_j$ and $K = L - weight(S_j)$.
      (Think of $B_j$ as choosing the remaining elements, when we have already decided to chose $S_j$).
      Output $A' = S_j \cup B_j$, which maximizes $value(S_j \cup B_j)$.
   End

   Note that the above algorithm runs in time polynomial in the size of the input, assuming $r$ is a constant (the algorithm's time complexity is exponential in $r$).

   Show that the above algorithm gives an output whose value is at least $(1 - \frac{1}{r})*$ optimal.

   Hint: Use the lemma done in class to find how much the difference in value given by the optimal algorithm and the above algorithm could be, when you fix the most valuable $r$ elements which belong to the optimal algorithm.

3. Consider the following variation of the knapsack problem.

   Input:
   (a) $n$ objects, $O_1, O_2, \ldots, O_n$, where the weight and value of object $O_i$ is $w_i$ and $v_i$ respectively. Here $w_i$ and $v_i$ are both posiitive integers. Furthermore, it is given that, for all $i$, $v_i \leq c * n$. where $c$ is a prefixed constant and $n$ is the number of objects.
   (b) A knapsack capacity $K$.

Output: A subset $S \subseteq \{1, 2, \ldots, n\}$ such that, $\sum_{i \in S} v_i$ is maximized subject to the constraint that $\sum_{i \in S} w_i \leq K$.

Show that the above problem can be solved in time polynomial in $n$ (note that $c$ is a constant).

Hint: Use a dynamic programming algorithm.

4. In this question we give an approximation algorithm for the general knapsack problem based on the previous question. Let $r > 1$ be a constant integer. Intuitively, the algorithm below finds a solution for the knapsack problem, whose value is at least $(1 - \frac{1}{r})$ times optimal value.

Approximate-Knapsack
Input:
(a) $n$ objects, $O_1, O_2, \ldots, O_n$, where the weight and value of object $O_i$ are $w_i$ and $v_i$ respectively. Here $w_i$ and $v_i$ are both posiitive integers. (b) A knapsack capacity $K$.
(* Assume without loss of generality that $w_i \leq K$ for each $i$. If not, we can clearly drop such objects from consideration. *)
1. Let $V = \max(\{v_i \mid 1 \leq i \leq n\})$.
   (* $V$ denotes the maximum of the values of all the objects. *)
2. For $1 \leq i \leq n$, let $s_i = \lfloor \frac{v_i * n * r}{V} \rfloor$.
   (* Note that $0 \leq s_i \leq n * r$. *)
3. Using the algorithm done in Q3 of this tutorial, solve the following modified Knapsack problem optimally:
   The objects are $O_1, \ldots, O_n$, knapsack capacity is $K$, where the weight and value of $O_i$ are $w_i$ and $s_i$ respectively (thus everything remains same except value is changed from $v_i$ to $s_i$).
   (* Note that this can be done optimally and in polynomial time, since $r$ is a constant. *)
4. Output the optimal set of objects as obtained for the modified problem in step 3.

End

Show that the above algorithm is a good approximation algorithm by showing the following parts.

(a) Show that $v_i - (s_i * \frac{V}{r*n}) \leq \frac{V}{r*n}$.

(b) Show that the value of the output given by the above algorithm differs from the value of optimal set by at most $\frac{V}{r}$.

(c) Show that the value of the output given by the above algorithm is at least $\frac{r-1}{r}$ times optimal.

5. The multiprocessor scheduling problem is given as follows:

   INPUT:

   (a) There are $m$ processors.

   (b) There are $n$ tasks, $a_0, a_1, \ldots, a_{n-1}$, with task length (time to do the task) $\ell(a_0), \ell(a_1) \ldots$. The tasks cannot be split into different processors.

   (c) a deadline $D$.

   (d) A schedule is a partition of the tasks into $m$ sets $S_0, S_1, \ldots, S_{m-1}$ such that $S_1 \cup S_2 \ldots \cup S_{m-1} = \{a_0, a_1, \ldots, a_{n-1}\}$. Time taken by this schedule is

$$\max(\{\sum_{a \in S_i} \ell(a) : 1 \leq i \leq m\})$$

   That is, time taken by the schedule is the maximum load on any of the processors.

   QUESTION: Is there a way to assign the tasks to the $m$ processors such that the time taken by the schedule is at most $D$?

   The multiprocessor scheduling problem is NP-complete. Thus the corresponding problem of finding an optimal schedule $S$, which minimizes the time taken by the schedule, is NP-hard. Consider the following approximation algorithm for Multiprocessor Scheduling. Intuitively the idea is to schedule the tasks in order of decreasing length (time taken to execute the task), where the tasks are assigned to the processors in rotating order.

   Multi_Schedule $(A, \ell, m)$
   (* $A$ is a set of $n$ tasks. $\ell$ is an array, where $\ell(a)$ denotes the time taken by task $a$. $m$ is the number of processors. *)
   1. Sort all the tasks based on non-increasing order of $\ell(\cdot)$.
   Say the sorted order is $a_0, a_1, \ldots, a_{n-1}$, where $\ell(a_i) \geq \ell(a_{i+1})$, for $i < n-1$.
   2. For $0 \leq i < m$, let $S_i = \{a_j \mid j < n \text{ and } (j \bmod m) = i\}$
   3. $S = (S_0, S_1, \ldots, S_{m-1})$ is the schedule for the $m$ processors, where $S_i$ is the set of tasks assigned to processor $i$.
   End

   Show that the above algorithm is a good approximation algorithm by showing that the completion time for the schedule generated by the above algorithm is no worse than twice the completion time of the optimal schedule.

   Hint: Consider the load due to the longest task and the rest of the tasks.