COMP 6651: Assignment 2

Submission through Moodle is **due by Feb 16th at 23:59**

Instructions: refer to Assignment 1 for instructions.

PROBLEMS.

1. [You are supposed to have some elementary knowledge of graph theory for this question, review it or ask PoD for help if you need.]

   In lecture 3 we discussed the jobs (where jobs are represented by intervals) scheduling problem and its solution via a greedy algorithm. The job scheduling problem can also be represented by a graph: each job (interval) is represented by a vertex, and two vertices are connected by an edge whenever the corresponding two jobs have a conflict (i.e., intervals overlap). Such graphs are called interval graphs. See Figure 1 and Figure 2-(a) for an illustration. Convince yourself that the graph in Figure 2-(b) is not an interval graph, i.e., there are no 4 interval combinations whose graph representation gives you the graph in Figure 2-(b).

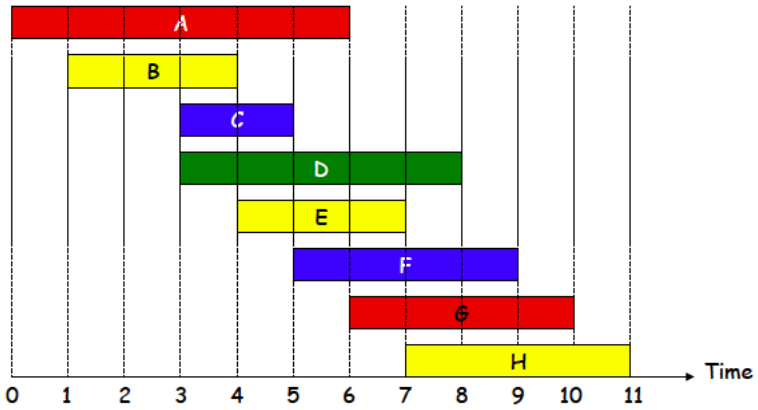


Figure 1: job scheduling problem

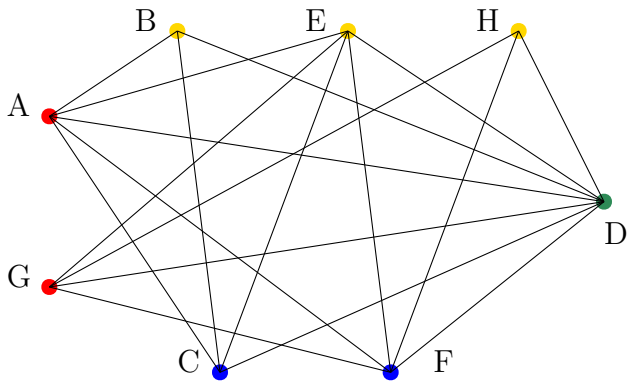
Observe that a subset of compatible jobs correspond to a subset of vertices in the corresponding interval graph satisfying the following condition:

the Condition: no edges between any two vertices in this subset.

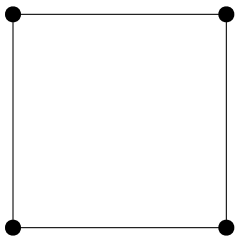
In graph theory, a subset of vertices satisfying **the Condition** is called an *independent set*.

In Figure 2, $\{B, E, H\}$, $\{A, G\}$, $\{C, F\}$, $\{D\}$, they are all independent sets of the graph, among which $\{B, E, H\}$ is the maximum independent set of size 3. Clearly, finding a maximum subset of compatible jobs is equivalent to finding a maximum independent set in the corresponding interval graph representation.

Below we discuss the independent set problem in arbitrary graphs: given a graph $G = (V, E)$, find one maximum independent set. Since the goal is to find vertices



(a)



(b)

Figure 2: (a) Interval graph representation of job conflicts, (b) NOT an interval graph.

where there are no edges between them, it seems natural to start with a vertex of lowest degree. So, consider the following natural lowest-degree greedy algorithm $\mathcal{A}$: find a vertex $v \in V$ of lowest degree, add v into a set S, remove v and all its neighbors, and then repeat until there are no more vertices

(1) Write pseudocode for the above greedy algorithm $\mathcal{A}$. What is its time complexity?

(2) Prove that the subset S output by $\mathcal{A}$ is always an independent set of G.

(3) Consider the following graphs: the complete graph K_n, the complete bipartite graph $K_{n,n}$. If you run $\mathcal{A}$ on these graphs, do you get one maximum independent set? Give your reason.

(4) Is it true that for any graph G, $\mathcal{A}$ will output one maximum independent set of G? If yes, prove it, if no, give a counter example.

2. You are given a set of m constraints over n Boolean variables $x_1, \ldots, x_n$. The constraints are of two types:

- **equality constraints**: $x_i = x_j$ for some $i \neq j$;
- **inequality constraints**: $x_i \neq x_j$ for some $i \neq j$.

Design an efficient **greedy** algorithm that given the set of equality and inequality constraints determines if it is possible or not to satisfy all the constraints simultaneously. If it is possible to satisfy all the constraints, your algorithm should output an assignment to the variables that satisfies all the constraints.

(1) Choose a representation for the input to this problem and state the problem formally using the notation **Input: ...**, **Output: ...**.

(2) Describe your greedy algorithm in plain English. In what sense is your algorithm "greedy"?

(3) Describe your greedy algorithm in pseudocode.

(4) Briefly justify correctness of your algorithm.

(5) State and justify the running time of your algorithm. The more efficient algorithm the better. Inefficient solutions may lose points.

3. Consider the Longest Increasing Subsequence (LIS) problem (see slides for problem definition).

   (1) Give a greedy algorithm for LIS. You should: firstly describe your algorithm in English, then write down the pseudocode, and finally point out its time complexity.

   (2) Your algorithm most likely is not optimal. Give one example on which your greedy algorithm outputs an optimal solution. Given another example on which your greedy algorithm outputs a suboptimal solution.

4. Consider the maximum-subarray problem. We've seen a divide and conquer algorithm for it using $\Theta(n \log n)$ time. Use dynamic programming to design a *linear-time*, i.e., $\Theta(n)$, algorithm for it. You can refer exercise 4.1-5 on page 75 of our text book (CLRS, Introduction to algorithms, 3rd ed.) for ideas.

   (1) Firstly describe your algorithm in English, then write down the pseudocode, then point out why its time complexity is $\Theta(n)$.

   (2) Run your algorithm on the following example:

   $$13, -25, 18, 20, -7, 12, -5, 4.$$

   Specifically, this means to write down the array used in your dynamic programming algorithm, and fill in the values as your algorithm will do. [Double check that your algorithm indeed outputs the optimal solution: this is not a part of the work to be submitted, just verify yourself.]