COL 351

TUTORIAL SHEET 11

1. [**KT-Chapter8**] You have a set of friends F whom you're considering to invite, and you're aware of a set of k project groups, $S_1, \ldots, S_k$, among these friends (these sets need not be disjoint). The problem is to decide if there is a set of n of your friends whom you could invite so that not all members of any one group are invited. Prove that this problem is NP-complete.

   Solution: It is in NP because a solution can just exhibit such a set of friends, and the verifying algorithm has to just check that for each group this set does not contain all the elements from that set – a task which can be easily done in polynomial time.

   To prove NP-completeness, we reduce from independent set. Consider a graph G and a number k. We map it to an instance of this problem as follows: set of friends F correspond to the set of vertices in G. For every edge $e = (u, v)$ in G, we have a set S_e consisting of $\{u, v\}$ only. It is easy to check that G has an independent set of size k iff the instance of this problem has a set of friends of size k with the desired property.

2. [**KT-Chapter8**] Given an undirected graph $G = (V, E)$, a feedback set is a set $X \subseteq V$ with the property that $G - X$ has no cycles. The undirected feedback set problem asks: given G and k, does G contain a feedback set of size at most k? Prove that the undirected feedback set problem is NP-complete.

   Solution: Easy to check that the problem is in NP: a solution needs to show a set of vertices S. The verifying algorithm first removes S, and then checks that the resulting graph does not have a cycle (can be done by any graph traversal algorithm).

   We reduce from vertex cover. Let G, k be an instance of vertex cover. We produce a graph $G' = (V', E')$ from $G = (V, E)$ as follows: for every vertex v in G, we have a vertex v in G' as well (G' will have some more vertices). For an edge $e = (u, v)$ in G, we create a new vertex v_e in G', and add edges $(u, v_e), (v_e, v), (u, v)$ in G'. Now, suppose G have a vertex cover of size k. Let these vertices be S. We claim that S will be a feedback set in G'. Indeed, any cycle in G' must contain a pair u, v, where (u, v) is an edge in G. Since S contains at least one of u and v, S must intersect this cycle. Thus after removing S from G', we will not have any cycle.

   Conversely, consider a feedback set S of size k in G'. First we claim that S need not contain any of the vertices v_e – indeed, if $e = (x, y)$, then any cycle containing v_e must also contain x. Thus we can replace v_e by x. Finally, any feedback set in S' must contain a vertex from the cycle x, y, v_e in G', where $e = (x, y)$ is an edge in E. We just argued that S must contain either x or y. Thus, S is a vertex cover in G.

3. [**KT-Chapter8**] We have seen the Interval Scheduling problem in class; here we consider a computationally much harder version of it that we will call Multiple Interval

Scheduling. As before, you have a processor that is available to run jobs over some period of time. (E.g. 9 AM to 5 PM.)

People submit jobs to run on the processor; the processor can only work on one job at any single point in time. Jobs in this model, however, are more complicated than we've seen in the past: each job requires a set of intervals of time during which it needs to use the processor. Thus, for example, a single job could require the processor from 10 AM to 11 AM, and again from 2 PM to 3 PM. If you accept this job, it ties up your processor during those two hours, but you could still accept jobs that need any other time periods (including the hours from 11 to 2). Now, you're given a set of n jobs, each specified by a set of time intervals, and you want to answer the following question: For a given number k, is it possible to accept at least k of the jobs so that no two of the accepted jobs have any overlap in time? Show that Multiple Interval Scheduling is NP-complete.

Solution: Easy to check that it is in NP. We reduce from Independent Set. Take an instance of independent set: graph G and number k. We map it to an instance of the multiple interval scheduling (MIS) as follows: the time interval stretches from 0 to m, where m is the number of edges. Let the edges be numbered $e_1, \ldots, e_m$. Then the interval $[i, i+1]$ will correspond to the edge e_i. For a vertex v, we define a job: this job will consist of intervals $[i, i+1]$ for every edge e_i which has v as an end-point. It is easy to check that G has an independent set of size k iff the corresponding k jobs in the MIS instance do not overlap (prove this yourself).

4. **[KT-Chapter8]** The following is a version of the Independent Set problem. You are given a graph $G = (V, E)$ and an integer k. For this problem, we will call a set $I \subseteq V$ strongly independent if for any two nodes $v, u \in I$, the edge (v, u) does not belong to E, and there is also no path of 2 edges from u to v, i.e., there is no node w such that both $(u, w) \in E$ and $(w, v) \in E$. The Strongly Independent Set problem is to decide whether G has a strongly independent set of size k. Prove that the Strongly Independent Set problem is NP-complete.

   Solution: Checking membership in NP is again straightforward. We reduce from independent set. Let G, k be an input to the independent set problem. We map it to an input to the Strongly Independent Set problem. For every edge e in G, we subdivide it by adding a new vertex v_e, i.e., if $e = (u, v)$ is an edge, then we replace it by two edges : $(u, v_e), (v_e, v)$. Further, we form a clique over all the new vertices v_e. Call this new graph G'. Now check that if S is an independent set in G, then the same set of vertices is a strongly independent set in G'. Conversely, let S be a strongly independent set in G'. First observe that $k > 1$ (otherwise, the reduction is trivially correct). Now S cannot contain any of the new vertices v_e (because any other vertex in G' is reachable from v_e by a path of length 2). Therefore, S contains vertices which correspond to those in G. Now check that these vertices form an independent set in S.

5. **[KT-Chapter8]** Consider the following problem. You are given positive integers $x_1, \ldots, x_n$, and numbers k and B. You want to know whether it is possible to partition the numbers $\{x_i\}$ into k sets $S_1, \ldots, S_k$ so that the squared sums of the sets add up to

at most B:

$$\sum_{i=1}^{k}\left(\sum_{x_j\in S_i} x_j\right)^2 \le B.$$

Show that this problem is NP-complete.

Solution: A solution just needs to exhibit the partition. Since addition and multiplication take polynomial time, we can easily verify a solution in polynomial time. We reduce PARTITION to this problem. Recall that an input to partition consists of a set of positive numbers $x_1, \ldots, x_n$, and we need to check if they can be divided into two parts, each of which has the same sum. We map this to our problem as follows: the set of numbers is $x_1, \ldots, x_n$ again. The parameter $k = 2$, and $B = \Sigma^2/2$, where Σ is the sum of these n numbers. Now suppose there were a partition of $x_1, \ldots, x_n$ into two sets A and B of equal sum. Then, the same partition in our problem will have the sum $\Sigma^2/2$.

Now we show the converse. Suppose it is possible to partition the numbers into parts A and B such that $\Sigma_A^2 + \Sigma_B^2 \le \Sigma^2/2$, where Σ_A denotes the total sum of numbers in A (and similarly for Σ_B). But note that $\Sigma_A + \Sigma_B = \Sigma$. It is an easy exercise to show that $\Sigma_A^2 + \Sigma_B^2 \ge \Sigma^2/2$ with equality if and only if $\Sigma_A = \Sigma_B$. Thus, $\Sigma_A = \Sigma_B$.

6. [**KT-Chapter8**] Suppose you're consulting for a group that manages a high-performance real-time system in which asynchronous process make use of shared resources. Thus, the system has a set of n processes and a set of m resources. At any given point in time, each process specifies a set of resources that it requests to use. Each resource might be requested by many processes at once; but it can only be used by a single process at a time. Your job is to allocate resources to processes that request them. if a process is allocated all the resources it requests, then it is active; otherwise it is blocked. You want to perform the allocation so that as many processes as possible are active. Thus, we phrase the Resource Reservation problem as follows: given a set of process and resources, the set of requested resources for each process, and a number k, is it possible to allocate resources to processes so that at least k processes will be active? Show that Resource Reservation is NP-complete.

   Solution: The problem is in NP because a solution needs to exhibit the resources allocated to each process. We reduce from independent set. Let G, k be an instance of the independent set problem. For every edge e, we have a resource r_e. For every vertex v, we have a process P_v which demands the resources $\{r_e : v \text{ is an end-point of } e\}$. It is easy to check that G has an independent set of size k if and only we can have k active processes.

8. In an undirected graph $G = (V, E)$, we say a subset $D \subseteq V$ is a dominating set if every $v \in V$ is either in D or adjacent to at least one member of D. In the DOMINATING SET problem, the input is a graph and a number b, and the aim is to find a dominating set in the graph of size at most b, if one exists. Prove that this problem is NP-complete.

   Solution: A solution just needs to exhibit the vertices in the dominating set – one can easily verify the solution in polynomial time. So the problem is in NP. We can reduce from the vertex cover problem. Let $G = (V, E)$ be an instance of the vertex cover problem. We map it to an instance of the dominating set problem as follows: we define a graph $G' = (V', E')$. V' has one vertex for every vertex in V, and one vertex (call it w_e) for every edge e in G. Now we describe E'. For an edge $e = (u, v)$ in G, we add edges $(u, w_e), (v, w_e)$ in G'. Further, we add edges between every pair of vertices from V in G'. Now suppose S is a vertex cover of size k in G. Then S is also a dominating set in G'. Conversely, let S be a dominating set of size k in G'. First of all, if a vertex of type w_e belongs to S, we can get another dominating set by replacing w_e by v in S, where v is one of the end-points of e in the graph G. Thus, we can assume S contains vertices which form a subset of V. Now it is easy to check that S should be a vertex cover in G.