CSE 417
Algorithms & Computational Complexity
Assignment #8 (rev. a)
Due: Wednesday, 3/13/19

Turnin Instructions: Gradescope again this week; use your @uw email and gradescope password as before.

Note: For this assignment, *no submissions will be accepted after 11:00PM, Friday, March 15*. I will post solutions early Saturday.

1. [20 points] My lecture slides (and KT section 8.2) show a reduction from SAT to Independent Set. Apply it to the Boolean formula:

   $$(x_1 \vee x_2 \vee x_4) \wedge (x_1 \vee \neg x_2 \vee x_3) \wedge (x_2 \vee \neg x_3 \vee \neg x_4)$$

   (a) Show the resulting graph and integer "k." Although the graph is technically just an ordinary undirected graph, with no vertex- or edge labels of any kind, in the reduction there is a natural correspondence between literals/clauses in the formula and vertices/vertex groups in the graph; show this somehow on your graph drawing. (Please try to draw your graph neatly and clearly; e.g., it's a planar graph—can be drawn in 2D without any edges crossing each other). It's hard to grade a hairball!)

   (b) The formula has many satisfying assignments, and the resulting graph has many independent sets. The correctness proof for the reduction defines a correspondence between the two. In particular, given a satisfying assignment, the proof explains how to choose one (or more) independent sets. For your graph, what set corresponds to the assignment $x_1 = x_2 = x_3 = False, x_4 = True$? (There should only be one.) Given that independent set, what assignments correspond to it? (There should be two.) Are they both satisfying assignments?

2. [20 points] In lecture I showed a reduction from SAT to Subset Sum (SS; aka KNAP). Apply it to the Boolean formula:

   $$(x_1 \vee x_2 \vee x_4) \wedge (x_1 \vee \neg x_2 \vee x_3) \wedge (x_2 \vee \neg x_3 \vee \neg x_4)$$

   (a) Show us the resulting SS instance. Although the SS instance is technically just a list of numbers, with no labels of any kind, in the reduction there is a natural correspondence between literals/clauses in the formula and numbers or parts of numbers in the SS instance; show this somehow.

   (b) The formula has many satisfying assignments, and the SS instance has many subsets summing to the specified target C. The correctness proof for the reduction defines a correspondence between the two. In particular, given a satisfying assignment, it explains how to choose one (or more) SS subsets whose total is C. For your SS instance, what subsets correspond to the assignment $x_2 = x_3 = False, x_1 = x_4 = True$? (There should four.) Given any of those subsets, what assignment(s) correspond to it? Are they satisfying assignments?

3. [20 points] The **MaxCut** problem is the following: given an undirected graph $G = (V, E)$ and an integer k, is there a partition of the vertices into two (nonempty, nonoverlapping) subsets V_1 and V_2 so that k or more edges have one end in V_1 and the other end in V_2?

   (a) Prove that MaxCut is **in** NP. Carefully, separately, describe the components that are central to the definition of membership in NP (slide ≈60; KT §8.3):

   i. What, precisely, is the "hint" / "certificate" / "witness" you envision for this problem?

   ii. How long is it?

   iii. What does the "verifier" do with it? In particular,

   - How does the verifier conclude that a specific "hint" proves that this is a "yes" instance of MaxCut, vs
   - What would cause the verifier to conclude that a specific "hint" *fails* to prove this? Why can't any "hint" for a "no" instance "fool" the verifier into saying "yes" when it shouldn't?

iv. Asymptotically, as a function if the length of the MaxCut instance and the length of the "hint", how much time does it take to run the verifier?

(b) Give an algorithm for MaxCut and analyze its running time. (A non-polynomial-time algorithm shouldn't be a big surprise.)

I want moderately detailed proofs/algorithms here, as in my slides concerning the definition of NP (numbered ≈48-74; see also KT §8.3).

Note: Recall that P is a subset of NP. The same facts can be established in almost the same way for the analogous MinCut problem (defined like MaxCut, except $\leq k$ instead of $\geq k$). In fact, MinCut is in P (using Max Flow/Min Cut tools; see KT chapter 7 and section 13.2, if you're interested), whereas MaxCut is known to be NP-complete (but you don't need to prove it). So, showing that a problem is in NP doesn't necessarily mean that all algorithms for it will be slow, although that becomes increasingly likely if you can't find a polynomial time algorithm after concerted effort, or, even better, if you prove it to be NP-complete.

4. [20 points] For any fixed integer $k \geq 2$, the k-PARTITION PROBLEM (abbreviated 'k-PP') is: given a sequence of positive integers $(w_1, w_2, \ldots, w_n)$, is it possible to partition them into k groups having equal sums. More formally, is there an n-vector h each of whose entries is a integer in the range $1, \ldots, k$ such that, for each $1 \leq j \leq k$:

$$\left(\sum_{i \in h^{-1}[j]} w_i \right) = \frac{1}{k} \sum_{i=1}^{n} w_i,$$

where $h^{-1}[j] = \{i \mid h[i] = j\}$. For example, $(1, 2, 1, 3, 1, 1)$ is a "yes" instance of 3-PP, but a "no" instance of 4-PP; $(3, 2, 1, 3, 3)$ is the reverse.

(a) As in the previous problem, carefully show that k-PP is in NP.

(b) Give an algorithm for k-PP and analyze its running time.

(c) Show that 2-PP $\leq_p$ KNAP (slide 36). (Make sure all the pieces required by my definition of reduction—slide 33—are carefully explained.) polynomial time algorithm that converts 2-PP to KNAP

(d) Suppose I could show that there is no polynomial time algorithm for KNAP. Would this together with 4c imply that there cannot be a polynomial time algorithm for 2-PP? Explain why or why not.

(e) **Extra Credit:** Show KNAP $\leq_p$ k-PP for each $k \geq 2$. Hint: start with $k = 2$.

5. [20 points] Give an algorithm to construct an Euler tour (slide 51) in a connected, undirected graph in which all degrees are even. (Recall, the "degree" of a vertex in a graph is the number of edges touching it.) Assume the graph is represented by edge lists. Your algorithm should run in time $O(|V| + |E|)$.

Revision History:

Rev a: removed "draft" label. — 3/8/19.