CSE 565: Algorithm Design and Analysis **Due 2022-11-04T23:59**

Problem Set 4 Solutions

Instructor: Dr. Antonio Blanca
TA: Jeremy Huang

Release Date: 2022-10-21

Notice: Type your answers using LaTeX and make sure to upload the answer file on Gradescope before the deadline. Recall that for any problem or part of a problem, you can use the "I'll take 20%" option. For more details and the instructions read the syllabus.

Problem 1. That's Bing

Suppose you have a procedure which runs in polynomial time and tells you whether or not a graph has a Hamiltonian path (given a graph $G = (V, E)$, whether there exists any $s, t \in V$ s.t. there exists a Hamiltonian-(s,t)-path in G). Show that you can use it to develop a polynomial-time algorithm for the Hamiltonian Path (which returns the actual path, if it exists).

Solution

We first reduce the problem of deciding a if a graph has a Hamiltonian-(s,t)-PATH (i.e. if the graph has a Hamiltonian path starting at s and ending at t) to the problem of deciding if the graph has *any* Hamiltonian path. We shall then use the procedure for deciding the (s,t)-Hamiltonian path to find the path.

Let G be the given graph. For given (s,t), we create a graph G' by two nodes s' and t' and edges (s,s') and (t,t'). Since s' and t' have only one edge incident on them, they cannot be in the middle of a path and hence G' has a Hamiltonian path if and only if G has an (s,t)-Hamiltonian path.

We now give an algorithm for finding a Hamiltonian path using the decision algorithm for (s,t)-Hamiltonian path. We can find the vertices in the path one by one starting at an endpoint. We first try all pairs of vertices till we find a pair (s,t) such that the graph has an (s,t)-Hamiltonian path - if no such pair exists, then there cannot be any Hamiltonian path in the graph. Since there is an (s,t)-Hamiltonian path, one of the neighbors of s must ne the second vertex in the path. Hence, the graph obtained by removing s from G (denoted as $G \backslash \{s\}$) must have a (u,t)-Hamiltonian path for some u such that $(s,u) \in E$. This gives us basis for recursion.

```
FIND-Hamiltonian(G)
    for all s ∈ V, t ∈ V:
        if HAS-Hamiltonian(G,s,t) ≠ false
            return FIND-(s,t)-Hamiltonian(G,s,t)


FIND-(s,t)-Hamiltonian(G,s,t)
    for all u such that (s,u) ∈ E
        if HAS-Hamiltonian(G\{s},u,t) ≠ false
            return {s} → FIND-(s,t)-Hamiltonian(G\{u},u,t)
```

Here $\{s\} \rightarrow$ FIND-(s,t)-Hamiltonian$(G\backslash\{u\},u,t)$ denotes adding s to beginning of the path returned by the called function. At each call, the size of the graph is reduced by 1. If $T(G,s,t)$ denotes the number of calls to HAS-Hamiltonian made by FIND-(s,t)-Hamiltonian(G,s,t), then

$$T(G,s,t) \leq T(G\backslash\{s\},u,t) + dergree(s)$$

Note that FIND-(s,t)-Hamiltonian(G,s,t) calls itself only once, because after that the function returns and hence exists the loop. Therefore $T(G,s,t) \leq \sum_{v \in V} degree(v) = 2|E|$. The total number of calls to HAS-Hamiltonian is at most $|V|^2$

in the upper loop in FIND-Hamiltonian and $2|E|$ in the recursion. If T is the running time of HAS-Hamiltonian, then the running time of FIND-Hamiltonian is $O((|V|^2+|E|)T)$, which is polynomial if T is.

Problem 2. 324

Assume that you are given a set of clauses where each of which is a disjunction of exactly 4 literals, and such that each variable occurs at most once in each clause. The goal is to find a satisfying assignment (if one exists). Prove that this problem is *NP*-complete.

Hint: Reduce 3SAT to this problem. Think about how to convert a clause from a 3SAT problem to an equivalent clause in this problem.

Solution

This problem is called EXACT 4SAT problem.

First prove that it is in NP. This is easy since given an assignment, one can check if it satisfies the input in linear (which is polynomial) time.

Reduction: We start from 3SAT and reduce it to an instance of EXACT 4SAT. We can assume that 3SAT formula has no clauses with one variable, since those variables can directly be assigned. Let $C_2 = (l_1 \vee l_2)$ and $C_3 = (l_3 \vee l_4 \vee l_5)$ be two clauses having 2 and 3 literals respectively. We can write them as the following equivalent groups of clauses with exactly 4 literals - we need to add one new variable for C_3 and two new ones for C_2.

$$C_3' \equiv (x \vee l_3 \vee l_4 \vee l_5) \wedge (\bar{x} \vee l_3 \vee l_4 \vee l_5)$$

$$C_2' \equiv (y \vee z \vee l_1 \vee l_2) \wedge (\bar{y} \vee z \vee l_1 \vee l_2) \wedge (y \vee \bar{z} \vee l_1 \vee l_2) \wedge (\bar{y} \vee \bar{z} \vee l_1 \vee l_2)$$

We can reduce any instance of 3SAT to EXACT 4SAT by re-writing all the clauses in the 3SAT formula using the above rewrite rules.

Correctness: Any assignment satisfying C_3' must assign either `true` or `false` to x, making one of the clauses true; then the other clause becomes exactly C_3. Any assignment satisfying C_2' must assign some pair of true-false values to (y,z), which makes 3 of the clauses true; then the remaining clause becomes exactly C_2'.

Complexity: The reduction can by done in poly-time since rewriting each clause takes constant time (the clauses have constant size) and by definition we only have polynomially many clauses to rewrite.

Problem 3. Brute Force

Show that for any problem $\Pi \in$ NP, there is an algorithm which solves Π in time $O(2^{p(n)})$, where n is the size of the input instance and $p(n)$ is a polynomial (which may depend on Π).

Solution

Since the problem Π is in NP, there must be an algorithm which verifies that a given solution is correct in polynomial time, say bounded by a polynomial $q(n)$. Since the algorithm reads the given solution, the size of the solution can be at most $q(n)$ bits. Hence, we run the algorithm on *all* $2^{q(n)}$ binary strings of length $q(n)$. If the given instance has any solution, then one of these inputs must be a solution. The running time is $O(q(n)2^{q(n)}) = O(2^n 2^{q(n)}) = O(2^{q(n)+n})$. Taking $p(n) = n + q(n)$, we are done.

Problem 4. Mengsk

In an undirected graph $G = (V,E)$, we say $D \subseteq V$ 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 budget 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.

Hint: This problem is similar to the vertex cover problem.

Solution

We reduce VERTEX-COVER to DOMINATING-SET. Given a graph $G = (V,E)$ and a number k as an instance of VERTEX-COVER, we convert it to an instance of DOMINATING-SET as follows. For each edge $e = (u,v)$ in the graph G, we add a vertex a_{uv} and the edges (u,a_{uv}) and (v,a_{uv}). Thus we create a "triangle" on each edge of G. Call this new graph $G' = (V',E')$.

We now claim that a G' has a dominating set of size at most k if and only if G has a vertex cover of size at most k. It is easy to see that vertex cover for G is also a dominating set for G' and hence one direction is trivial. For the other direction, consider a dominating set $D \subseteq V'$ for G'. For each edge $e = (u,v) \in E$, D must contain at least one of u, v or a_{uv}, since a_{uv} or one of its neighbors needs to be in the dominating set. If D contains only a_{uv}, we can replace it by either u or v and the resulting set is still a dominating set of the same size. If it contains one of u and v, then we can remove a_{uv} if it is present in D.

This gives a set D' which does not contain any of the extra vertices we introduced. Also, for each edge $(u,v) \in E$ at least one of u and v is in D'. Hence, D' is a vertex cover for G. Finally, by construction $|D'| \leq |D|$ (since we only remove or exchange vertices).

Problem 5. Gethen

One experimental procedure for identifying a new DNA sequence repeatedly probes it to determine which k-mers (substrings of length k) it contains. Based on these, the full sequence must then be reconstructed. Let's now formulate this as a combinatorial problem. For any string x (the DNA sequence), let $\Gamma(x)$ denote the multiset of all of its k-mers. In particular, $\Gamma(x)$ contains exactly $|x| - k + 1$ elements. The reconstruction problem is now easy to state: given a multiset of k-length strings, find a string x such that $\Gamma(x)$ is exactly this multiset.

(1) Show that the reconstruction problem reduces to Hamiltonian PATH. (Hint: Construct a directed graph with one node for each k-mer, and with an edge from a to b if the last $k-1$ characters of a match the first $k-1$ characters of b.)

(2) Part 1 doesn't have to be bad news. Show that the same problem also reduces to EULER PATH. (Hint: This time, use one directed edge for each k-mer.)

Solution

(1) Constructing a digraph as in the hint, a Hamiltonian path exactly corresponds to a sequence of k-mers such that the last $k-1$ characters of each element match the first $k-1$ characters of the next element. This is just the reconstructed sequence.

(2) Let S be the *set* (not multiset) of all the strings formed by the first $k-1$ characters and all strings formed by the last $k-1$ characters of all the given k-mers. Construct a directed graph with $V = S$, with an edge $(u,v) \in E$ if there exists a k-mer having u as the $k-1$ characters and v as the last $k-1$ characters. It is easy to see that each k-mer corresponds to exactly one edge. An Euler path in this graph gives a sequence of elements as the Hamiltonian path above.

Problem 6. Tricolor

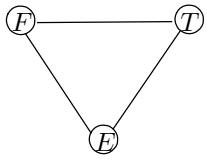
The k-coloring problem is the problem of deciding whether a graph $G = (V,E)$ is k-colorable – whether there exists an assignment c of colors $\{1,\ldots,k\}$ to the vertices V such that $c[u] \neq c[v]\ \forall (u,v) \in E$. That is, in a proper coloring, the endpoints of every edge are assigned different colors. Please follow the steps below to prove that 3COLOR, the 3-coloring problem, is NP-complete by reducing 3SAT to it.

Part (1)

Prove that 3COLOR is in NP.

Part (2)

Consider the following labeled 3-clique, denoted K_3:

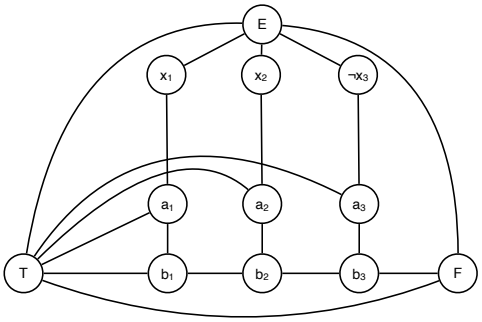


This graph has 3 nodes, and each node must be colored with a different color. We can use it to label the colors $\{1,2,3\}$ in a 3-coloring of K_3 as (T)rue, (F)alse, or (E)xcluded; that is, if vertex F is assigned color 2, then we associate color 2 with False.

Given a set of n boolean variables $x_1 \ldots x_n$, construct a graph G and a bijection between valid 3-colorings of G and all 2^n possible assignments of the boolean variables.

Part (3): Connecting One OR Gadget

Consider the following 12 node graph.



i. Argue that T, F, E all have different colors and that $x_1, x_2, \bar{x}_3$ must be colored with T or F.

ii. Prove that there is no valid 3-coloring of the graph where $\{x_1, x_2, \bar{x}_3\}$ all have the color F.

iii. Prove that for each other coloring of $\{x_1, x_2, \bar{x}_3\}$ (with T or F) there exists a compatible 3-coloring of the graph.

Part (4)

Given a set of n boolean variables $x_1 \ldots x_n$ and a set of m disjunctive clauses $c_1 \ldots c_m$ with three literals, describe a poly-time procedure that produces a graph which is 3-colorable if and only if all m clauses are satisfiable. Hint: connect multiple OR gadgets from Part (3) to the graph you made in Part (2).

Part (5)

Prove the correctness of your reduction and prove that it is a poly-time reduction. You should use the results from Part (3) in your proof.

Solution

(1) Here is a verifier for 3COLOR: Giving a coloring c of G, check that $c[u] \neq c[v]$ for each edge $\{u,v\} \in E$. If so, then accept c as a 3-coloring of G. Otherwise reject. The verifier is poly-time since it does constant work for each edge and the certificate c is poly-size since $|c| = |V|$, so 3COLOR is in NP.

(2) Graph creation procedure: Start with the 3-clique given in the question. For each variable x_i create two nodes labeled x_i and $\bar{x}_i$; add edges so that x_i, $\bar{x}_i$, and E form a 3-clique. Mapping:

$$\begin{aligned} \text{Node } x_i \text{ has same color as } T &\leftrightarrow x_i \text{ is True} \\ \text{Node } x_i \text{ has same color as } F &\leftrightarrow x_i \text{ is False} \end{aligned}$$

Since each $x_i, \bar{x}_i$ are only have edges to E and each other, there are exactly 2 ways to color them: T,F or F,T. So the mapping is bijective for one variable. Since each pair of nodes is only connected to every other pair through E, the choice of coloring for each pair is independent of all the other pairs, so the mapping is bijective for all n variables.

(3)(i) T, F, E are a 3-clique, so they must all have different colors. $x_1, x_2, \bar{x}_3$ are connected to E, so they must be colored with T or F.

(3)(ii) Assume that $x_1, x_2, \bar{x}_3$ are all colored with F.

Then a_1 must be colored with E, since it shares an edge with T and x_1 (F). Ditto for a_2 and a_3.

Then b_1 must be colored with F since it shares an edge with T and a_1 (E).

Then b_2 must be colored with T since it shares an edge with b_1 (F) and a_2 (E).

Then b_3 cannot be colored because it shares edges with b_2 (T), a_3 (E), and F.

Since b_3 cannot be given a valid color if $x_1, x_2, \bar{x}_3$ all have color F, there is no valid coloring of the graph where $x_1, x_2, \bar{x}_3$ all have color F.

(3)(iii) Let $c[v]$ represent the color of node v. We will show that if one variable has color T, then there is a valid coloring no matter the color of the other two. The argument is similar for all 3 cases.

If $c[x_1] = T$, we can set $c[a_2]$ and $c[a_3]$ to E no matter what $c[x_2]$ and $c[\bar{x}_3]$ are. Then $c[b_2] = F, c[b_3] = T$, $c[b_1] = E$ and $c[a_1] = F$ is a valid coloring. So there is a valid coloring for all $c[x_1], c[\bar{x}_3]$.

If $c[x_2] = T$, we can set $c[a_1]$ and $c[a_3]$ to E no matter what $c[x_1]$ and $c[\bar{x}_3]$ are. Then $c[b_1] = F, c[b_3] = T$, $c[b_2] = E$ and $c[a_2] = F$ is a valid coloring. So there is a valid for all $c[x_2], c[\bar{x}_3]$.

If $c[\bar{x}_3] = T$, we can set $c[a_1]$ and $c[a_2]$ to E no matter what $c[x_1]$ and $c[x_2]$ are. Then $c[b_1] = F, c[b_2] = T$, $c[b_3] = E$ and $c[a_3] = F$ is a valid coloring. So there is a valid coloring for all $c[x_1], c[x_2]$.

These three cases cover all possible colorings of $x_1, x_2, \bar{x}_3$ besides F,F,F as requested.

(4) Start with the graph from Part 2. For each clause c_i, create 6 new nodes $c_i a_1, \ldots c_i b_3$. Connect these nodes to T, F, and each other so that $c_i a_1$ is analogous to a_1 in Part 3, etc. Then connect the node corresponding to the 1st variable of c_i to $c_i a_1$, 2nd var of c_i to $c_i a_2$, and 3rd var of c_i (if there is one) to $c_i a_3$. If c_i doesn't have a 3rd var, connect F to $c_i a_3$ instead. Call this graph G.

(5) Correctness: fix an arbitrary clause c_i. If c_i is a 3 variable clause, then by the proof in Part 3 its corresponding OR gadget in G is 3-colorable iff one of the nodes for the variables in c_i can be colored with T. If c_i is a 2 variable clause then $c_i a_3$ must have color E so the proof in Part 3 still applies. Since G is only 3-colorable iff all of the OR gadgets in it are 3 colorable, G is 3-colorable iff at least one variable node from each clause can be colored with T.

So if there is an assignment of all x_i that statisfies all c_i, then there is a coloring of the variable nodes in G such that at

least one variable node from each clause can be colored with T, so G is 3-colorable;

and if G is 3-colorable, at least one variable node from each clause can be colored with T, so there is an assignment of the x_i that satisfies all the c_i.

Complexity: G has three types of nodes: variable nodes, clause nodes, and the nodes T, F, E. G has $O(n)$ many variables nodes, $O(m)$ many clause nodes (since a constant number of nodes are added for each variable/clause), and 3 nodes T, F, E. Since G has polynomially many nodes it is of polynomial size, so it can be constructed in polynomial time, so the reduction is poly-time overall.

Problem 7. Fixed Span

Determine which of the following problems are NP-complete and which are in P and explain why. In each problem you are given an undirected graph $G = (V, E)$, along with:
(a) A set of nodes $L \subseteq V$, and you must find a spanning tree such that its set of leaves includes the set L.
(b) A set of nodes $L \subseteq V$, and you must find a spanning tree such that its set of leaves is precisely the set L.
(c) A set of nodes $L \subseteq V$, and you must find a spanning tree such that its set of leaves is included in the set L.
(d) An integer k, and you must find a spanning tree with k or fewer leaves.

Solution

(a) This can be solved in polynomial time. Delete all the vertices in the set L from the given graph and find a spanning tree of the remaining graph. Now, for each vertex $l \in L$, add it to any of its neighbor present in the tree. It is clear that such a tree, if it is possible to construct one, must have all the vertices in L as leaves. If the graph becomes unconnected after removing L, or some vertex in L has no neighbors in $G \backslash L$, then no spanning tree exists having all vertices in L as leaves.

(b) This generalizes the (undirected) (s,t)-Hamiltonian PATH problem. Given a graph G and two vertices s and t, we set $L = \{s,t\}$. We now claim that the tree must be a path between s and t. It cannot branch out anywhere, because each branch must end at a leaf and there are no other leaves available. Also, since it is a spanning tree, the path must include all the vertices of the graph and hence must be Hamiltonian path. Similarly, every Hamiltonian path is a tree of the type required above.

(c) This is also a generalization of (undirected) (s,t)-Hamiltonian PATH. We use the same reduction as in the previous part. Note that a tree must have at least two leaves. Hence for $L = \{s,t\}$, the set of leaves must be exactly equal to L.

(d) Again, setting $k = 2$, gives exactly the (undirected) Hamiltonian path problem, since a spanning tree with at most two leaves must be a path containing all the vertices. Also, it will have exactly two leaves (since that's the minimum a tree can have).

Problem 8. (I Can Get Some) Satisfaction

In the textbook, it has mentioned that 3SAT remains NP-complete even when restricted to formulas in which each literal appears at most twice.

(a.) Show that if each literal appears at most *once*, then the problem is solvable in polynomial time.

(b) Consider a special case of 3SAT in which all clauses have exactly three literals, and each variable appears at most three times. Show that this problem can be solved in polynomial time.

Solution

(a.) Let the number of variables be n and the number of clauses be m. Note that each variable x can satisfy at most 1 clause (because it appears as x and $\bar{x}$ at most once). We construct a bipartite graph, with the variables on the left side and the clauses on the right. Connect each variable to the clauses it appears in. For the formula to be satisfiable, each clause must pick (at least) one variable it is connected to with each variable being picked by at most one of the clauses it is connected to. If we connect all the variables to a source s, all clauses to a sink t and fix the capacity of all edges as 1, this is equivalent to asking if we can send a flow of m units from s to t. Since the flow problem can be solved in polynomial time, so can the satisfiability problem.

(b.) Consider a bipartite graph with clauses on the left and variables on the right. Consider any subset S of clauses (left vertices). This has exactly $3|S|$ edges going out of it, since each clause has exactly 3 literals. Since each variable on the right has at most 3 edges coming into it, S must be connected to at least $|S|$ variables on the right. Hence, this graph has a matching using Hall's theorem proved in the last discussion section. This means we can match every clause with a unique variable which appears in that clause. For every clause, we set the matched variable appropriately to make the clause true, and set the other variables arbitrarily. This gives a satisfying assignment. Since a matching can be found in polynomial time (using flow), we can construct the assignment in polynomial time.

Problem 9. Stingy Set

(a.) STINGY_SAT is the following problem: given a set of clauses (each a disjunction of literals) and an integer k, find a satisfying assignment in which at most k variables are true, if such an assignment exists. Prove that STINGY SAT is NP-complete

(b.) Show that INDEPENDENT_SET remains NP-complete even in the special case when all nodes in the graph have degree at most 4.

Solution

(a.) STINGY_SAT is a generalization of SAT, so SAT reduces to it. Given a SAT formula φ with n variables, (φ, n) is an instance of STINGY SAT which has a solution if and only if the original SAT formula has a satisfying assignment.

(b.) Consider the reduction from $3SAT$ to $INDEPENDENT\_SET$. If each literal is allowed to appear at most twice, we claim that the graph we create in this reduction has degree at most 4 for each vertex. Let $(x_1 \vee x_2 \vee x_3)$ be a clause in the original formula. Then, in the graph, the vertex corresponding to x_1 in this clause, has one edge each to x_2 and x_3. Also, it has edges to all occurrences of $\bar{x}_1$. But, since $\bar{x}_1$ is allowed to appear at most twice, this can add at most two edges. Similarly, each vertex in the graph has at most 4 edges incident to it.

However, we know that the special case of 3SAT, with each literal occurring at most twice is NP-complete. The above argument shows that this special case reduces to the special case of INDEPENDENT_SET, with each vertex having degree at most 4. Hence, this special case of INDEPENDENT_SET must also be NP-complete.

Problem 10. Frozen Snake

A (2d) self-avoiding walk is a sequence of points $(p_1, \ldots, p_n)$ such that: 0. p_i are grid points – points with integer coordinates in $\mathbb{R}^2$, 1. no two points are equal, 2. p_i and p_{i+1} are exactly distance 1 apart, 3. $p_1 = (0,0)$.

$A(n)$ is defined to be the number of unique self-avoiding walks of length n.

Describe a poly-space algorithm to compute $A(n)$ given n. Show that it works and that it has the correct space complexity. This will prove that the problem of computing $A(n)$ is in PSPACE.

Solution

Idea: we iterate through all possible walk of length n and count how many are self avoiding. We can descibe a self avoiding walk using the steps it takes (up, down, left, right) instead of the exact points it ends up on to help us do the iteration.

Algorithm: Start a counter at zero and iterate through all strings in $\{U,D,L,R\}^{n-1}$ (we can do this by incrementing through all binary numbers of length $2n$ and interpreting each pair of bits as one of four letters). For each string, compute the sequence of points in that walk to see if any points are repeated (which can be done by linear search). If no points are repeated, increment our counter. At the end of the iteration, the value of our counter will be $A(n)$.

Correctness: The algorithm iterates through all walks of length n once, so it also iterates through every self-avoiding walk exactly once. It increments the counter iff the current walk is self-avoiding, so the value of the counter after the iteration is over must be $A(n)$.

Space Complexity: Storing the walk as a string and as a list of points uses $O(n)$ space. Storing the counter also uses $O(n)$ space (since the number of walks is exponential). We only need to store one walk at a time, so the overall space usage is polynomial.