Computational Complexity 2023 **Week 2**

Proposed Solutions to Exercises #2

Instructor: Bruno Loff *TA:* Jorge Pereira

Exercise 2.2)

2COL) Given a graph G it's easy to see that the representation of a two color assignment to it can be done such that its length is polynomial in $|<G>|$, in fact the length of that representation doesn't need to exceed the length of the representation of G (particularly in this case where we have only two colors, for instance in the case where the representation of G comes from an incidence matrix the coloration can be represented by a diagonal matrix such that each color is represented by either a 0 or a 1; if we adopt a representation of G by a list, first listing the vertices and then listing tuples stating pairs of connected vertices then the coloring can come as a list of tuples stating what color each vertex gets, thus as long as we have less colors then the number of vertices we know that the length of the representation of the coloring must be at most the length of the representation of G, and if we have more colors than vertices then the coloring problem becomes trivial).

As such the question now is if we can construct a TM that verifies the coloring in polynomial time in the length of the inputted graph (if the certificate has representation of polynomial length in the length of the graph then the TM runs in polynomial time in the length of the graph and we can omit the fact that it also has as input the coloring as far as polynomial running time goes).

The TM must verify that indeed the certificate is valid, namely the formatting is valid, which we can omit here because it can be done by reading its input only once (here we can include the verification that the coloring has indeed only two colors), and that the coloring is valid, that is for a vertex with color 0 all adjacent vertices have color 1 and vice versa.

We'll call the certificate u (the coloring) and the TM in question M. The TM M will have 4 tapes, tape 1 for the input $<G>$, tape 2 for the input $<u>$, tape 3 as the work tape and tape 4 can be the output tape (we note here that the output tape is in fact superfluous, we add it so as to be consistent with the definitions in the book, the TM M could simply use the work tape to write the output in the end of the computation, in fact we don't even need output we just want to know if M enters an *accept state* in the end of the computation or a *reject state*, because the output is simply a 0 or a 1 it won't affect the performance of the TM; in case we do need output then we must assure that its representation has length polynomial in the length of the input otherwise the TM can't possibly run in polynomial time)

The TM M starts by reading u and finding a "first" vertex on it and its corresponding coloring and proceeds to write the vertex and coloring on the third tape for future reference, then it can read the first tape to find the adjacent vertices and each time it find an adjacent vertex it reads its coloring on the second tape and compares it to the color on the third tape, if it's different it proceed to the next adjacent vertex until no more adjacent vertices are found (that of course must happen eventually as the graph is finite). If an adjacent vertex with the same color is found M can enter the *reject state*, if not it has examined all adjacent vertices and can clear the contents of tape 3 and proceeds to the next vertex on the second tape and repeats the whole process. Eventually it will have examined all vertices on tape 2 (and corresponding adjacent vertices) and if no contradiction was found it can enter the *accept state*.

Let k be the number of vertices in the graph G. The whole computation of M must terminate after at most k steps, each step costing $|<G>| + (2k+1)|<u>|$ because at each step we have to read the contest of tape 1 only once but read the contest of tape 2 and 3 the same number of times as the number of adjacent vertices found (trivially at most k) and the content of tape 3 are of size at most $<u>$, ergo the factor 2 on the $k|<u>|$, plus one because we start each step by reading u. This is clearly polynomial in $|<G>|$.

Therefore we have a certificate with polynomial length in $|<G>|$ and a TM that runs in polynomial time in the length of the input and verifies if a coloration is indeed valid, that is verifies the certificate, thus 2COL$\in$NP.

Note that the TM M operates in a very crude way, we could use *memoization* or any other optimization technique to make the algorithm faster, or just count the running time better, yet we don't have to, we don't care about polynomial differences in running time, for our purposes the algorithm could run in time n^2 or n^{200000} and the conclusion would be the same.

3COL) Analogous to the previous solution (if you can't do this one by using the previous solution then you didn't understand the previous solution and need to spend more time with it!).

CONNECTED) Here we can start from an arbitrary vertex and check if it's connected to every other vertex using breadth first search[1]. If we have k vertices we have $k-1$ pairs of vertices to check if each pair is connected in G, thus we run breadth first search $k-1$ times on G. As is knows breadth first search runs in polynomial time on its input, say the polynomial is $p(x)$, then the running time of running it $k-1$ times if less than $|<G>|p(|<G>|)$ which is still a polynomial in $|<G>|$. As such we can conclude that CONNECTED$\in$P and so is in NP.

If we just want to prove that CONNECTED is in NP then we could use the $k-1$ paths between a particular vertex and any other vertex as the certificate, it clearly has representation of polynomial length in $|<G>|$ as we can just list each path as a (sub) list of vertices and so in total the size of the certificate must be $O(|<G>|^2)$ (I say big O $|<G>|^2$ because as we concatenate each list (representing a path) we must have a proper way to delimit each list or else the TM won't have a way to know where each list start and the next begins and so that will cost some bits, refer to the section on "Representing pairs and tuples" of the book for an example).

With that, all we have to do is build a TM that verifies that each path given is valid and that all vertices are reached, we can as before use a 3 tape TM M (plus an output tape if desired). To do that all we have to do is check that we have $k-1$ sub-lists and the vertices at the end of each sub-list plus the initial vertex in a sub list (any sub-list they all start with the same vertex) give all the vertices of G, with that done we proceed to verify if all paths are valid. That is only a matter of going over a path (sub-list) and checking if each successive vertex is adjacent to the previous by reading that information on $<G>$.

Let M's first tape contain $<G>$ and the second contain the mentioned certificate, call it u. Then (assuming the process of verifying the format of u was already done, which can include checking the number of sub-lists (we can use tape 3 to store the counter)) we first read u to extract the first vertex of the first sub-list and all the last vertices of each list and store it in tape 3, then we go over tape 1 to check if tape 3 contains all vertices, if this passes we proceed to check the paths (sub-lists). To to this all M has to do is start reading u from the start and uppon reading a vertex it stores it in tape 3, goes to the next vertex on the list and then reads the first tape to check if the vertex on tape 3 and the vertex it read on tape 2 are adjacent, if so it proceed to write the vertex it read on tape 2 on tape 3 and repeats the process until the end of the sub-list, and then it goes to the next sub-list and so on and so forth.

Note that instead of using tape 3 to write both vertices we are checking for adjacency we only store the first on tape 3 and use tape 2 as the one that stores the second vertex, after all the head on tape 2 doesn't need to leave the region where that vertex is stored and that region must be properly delimited or else M wouldn't know where a vertex begins and the other ends, this is a nice time to ponder on the necessity of the third tape, I mean we do know that we don't even need three tapes to have a polynomial running time but we don't need to go that far yet. Note that the reason we use here and in the TM used for the 2COL language the third tape is because we want to make it easy for the TM to know where to search for the information it needs, but this can be achieved by extending the alphabet of the tapes to included a marker that marks the position of interest for future reference, and as we know extending the alphabet of the tapes can be reversed with just a polynomial slowdown. So this is a nice example of the use of the robustness of the TM model to assist us in the construction of the machines we need. So in short, if we are just comparing information we don't need an additional work tape beyond the input tape(s).

To conclude all is left to do is analyze the running time of the machine. So we first read $<u>$ and write a list of vertices in the third tape to see if all vertices are reached, requiring us to read $<G>$ just once, so $2|<u>|+|<G>|$ operations in the first step. In the second step for each vertex we read in $<u>$ we read $<G>$ to check for adjacency having a total cost of at most $|<u>||<G>|$ plus the cost of writing each time a vertex in the third tape that in total can't cost more that $|<u>|$, thus is total the cost of running M must be less than $3|<u>|+|<G>|+|<u>||<G>|$ which is polynomial in the length of $<G>$. So we have a certificate whose representation has polynomial length in $|<G>|$ and a TM that runs in polynomial

[1]To see a description of the breadth first search algorithm refer to "Introduction to Algorithms, Third Edition by Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein" Sec. 22.2. The algorithms there are described in a RAM computation model and as was stated last week the Turing machine model is capable of simulation a RAM model with only a polynomial slowdown (this can also be seen in "Introduction to Automata Theory, Languages, and Computation by John E. Hopcroft, Jeffrey D. Ullman" page 167). If you need any of the books I proposed as reading materials please email me and I'll provide them to you.

time in $| < G > |$, thus the conclusion follows.

Who's in P?

Now we answer the final question, which of these languages are also in P. As we've seen CONNECTED is in P; 3COL is in fact NP-complete (refer to exercise 2.21) and so is only in P if P=NP.

Finally 2COL is in P, to see that note that G can only have two possible colorings because a given vertex can have only two colors assigned to it and each possibility result in a full coloration of the graph by inductively coloring its adjacent vertices. It's easy to see that we can produce those colorings in polynomial time, and with that produce two certificates that we can feed to the verifier M described for 2COL (that also runs in polynomial time in $| < G > |$), thus in polynomial time we can determine if a given graph is 2-colorable. (if you don't fell completely comfortable with this answer do try to fill in the details)

Exercise 2.4)

Given the system $Ax \leq b$ (where the $\leq$ is applied component wise), if it has a rational (component wise) solution a then $Aa = b_a$ where b_a is rational (considering a and A are) and $b_a \leq b$. By 2.3) the system $Ax = b_a$ has a solution x' with representation whose length is polynomial in $| < A, b_a > |$ and analyzing the solution to 2.3) one can see that that solution x' is (or can be) rational. If $| < x' > |$ is not polynomial in $| < A, b > |$ then $| < b > |$ is not polynomial in $| < b_a > |$, but for x' both b and b_a are constant, contradicting the previous statement (the rationale here is that given two numbers p, q one can always find a number c such that $p = cq$), thus $| < A, b > |$ is polynomial in $| < A, b_a > |$ and so x' is a solution to $Ax \leq b$ whose representation has polynomial length in $| < A, b > |$. So we have a certificate $(< x' >)$ in the required conditions of the definition of NP (Def. 2.1 in the book), now it's straightforward to see that a TM that runs in polynomial time in $| < A, b, x' > |$ exists for we need only for it to compute the functions SUM, PROD and $\leq$ (lexicographic comparison) all of which run in polynomial time (that is we need only to compute Ax' and then verify that $Ax' \leq b$). Now if A is a $m \times n$ matrix then we compute the PROD function mn times, the SUM function $m(n-1)$ times and the $\leq$ function m times and $m = O(| < A, b, x' > |)$, $n = O(| < A, b, x' > |)$ thus in total we're computing these function a polynomial number of times in $| < A, b, x' > |$ (to be exact $2mn = O(| < A, b, x' > |)$ executions of these functions, combined)[2] therefore M runs in polynomial time in $| < A, b, x' > |$.

Exercise 2.10)

Given languages L_1, $L_2 \in$ NP there exist TM verifiers M_1 and M_2 respectively associated to the languages and we can assume these are single tape TM's.

Let's deal with the case of the union of the languages and then their intersection. We can construct a TM M that simulates M_1 and then M_2 (simulates both sequentially) and if M_1 enters an accept state then M enters an accept state, otherwise it proceed to simulate M_2 and if it enter an accept state M enter and accept state, otherwise if both M_1 and M_2 reject (enter a reject state) then M enters a reject state. This sequential simulation of M_1 and M_2 can be done by modifying the transition function of M_1 such that if M_1 enter the reject state then M enters the initial state of M_2 and thus proceed to work as per the transition function of M_2 (effectively extending the transition function of M_1), of course more states have to be added to allow M itself to have and be capable of entering an accept and reject state and also manage the tape (for more on this refer to "Introduction to Automata Theory, Languages, and Computation by John E. Hopcroft, Jeffrey D. Ullman" Sec. 7.4 and also Example 8.2). This being done we note that given $x \in L_1 \cup L_2$ then there exists a certificate u for x (pertaining to L_1 or L_2 depending on where x is) and if we feed $< x, u >$ to M then either the simulation of M_1 or the simulation of M_2 will accept and thus M will accept, more over M runs in polynomial time for M_1 and M_2 do so and M simply runs both TM's in sequence (a sum of polynomials is a polynomial), to be more precise if $p_1(n)$ and $p_2(n)$ are the polynomials associated with the running time of M_1 and M_2 respectively then M runs in $O(p_1(n) + p_2(n))$. Therefore for each $x \in L_1 \cup L_2$ we have a certificate with representation whose length is polynomial in $| < x > |$ and a TM M that runs in polynomial time of the length of its input, ergo $L_1 \cup L_2 \in$ NP.

We now proceed to the case where we consider the intersection of the languages. Here we construct a TM M with two tapes that simulates M_1 and M_2 simultaneously (in parallel, one on each tape). The TM M enters an

[2] If asymptotic notation (big O) is still daunting to you I strongly suggest going over it in the book "Introduction to Algorithms, Third Edition by Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein" Sec. 3.1.

accept state if both simulations accept (enter an accept state) otherwise it rejects (again you can refer to "Introduction to Automata Theory, Languages, and Computation by John E. Hopcroft, Jeffrey D. Ullman", Sec. 7.4 and 7.5 or even "Introduction to the Theory of Computation by Michael Sipser" Sec. 3). It should be clear that this TM runs in polynomial time, with its running time upper bounded by the maximum running time between M_1 and M_2 (using the polynomials p_1 and p_2 defined as before the running time of M is $O(max(p_1(n), p_2(n)))$). Given $x \in L_1 \cap L_2$ then there exist certificates u_1 and u_2 pertaining to M_1 and M_2 respectively and thus their concatenation $< u_1, u_2 >$ has length polynomial in $| < x > |$, we feed $< x, < u_1, u_2 >>$ to M and then M simulates M_1 on $< x, u_1 >$ and M_2 on $< x, u_2 >$ resulting in M entering an accept state for both simulations will accept by their definition and the definition of the certificates u_1 and u_2. Therefore for each $x \in L_1 \cap L_2$ we have a certificate whose representation has length polynomial in $| < x > |$ and a TM M that runs in polynomial time in the length of (the representation of) its input (even after we modify it to run on a single tape), and so $L_1 \cap L_2 \in$ NP.

Note that the TM M we defined for the language $L_1 \cup L_2$ could, with small modifications, be used for the language $L_1 \cap L_2$ and vice versa.