Problem Solving Homework (Week 2)

161180162 Xu Zhiming

July 4, 2018

TC Chapter 34

34.1-2

The formal definition:
Input: Instance of some undirected graph
Output: A simple cycle
Relation between them: The instance graph contains the cycle, and such cycle is of the largest length.
The decision problem is: LONGEST-CYCLE=$\{\langle G, u, k\rangle : G = (V, E)$ is an undirected graph, $u \in V$, $k \leq 0$ is an integer, and G contains a simple cycle starting from u, which has at most k edges$\}$.
The language is the same as LONGEST-CYCLE

34.1-3

Suppose the alphabet is $\{0, 1, \#, *\}$, where # represents another column and $*$ denotes another row. Then the adjacent matrix representation of a directed graph is as follows ($BiStr[i] \in \{0,1\}^*$ is the distance between $\lfloor i/n+1 \rfloor$ and $\lceil i\%n \rceil$):

$$\begin{gathered} BiStr[0]\#BiStr[1]\#\cdots BiStr[n-1]* \\ BiStr[n]\#BiStr[n+1]\#\cdots BiStr[2n-1]* \\ \cdots\cdots \\ BiStr[n^2-n]\#BiStr[n^2-n+1]\#\cdots BiStr[n^2-1]* \end{gathered}$$

The adjacent table is:

$$\begin{gathered} (s_1[0], s_2[1])(s_1[2], s_2[3])\cdots(s_1[m], s_2[m])* \\ \cdots\cdots \\ (s_1[l], s_2[l])(s_1[l+1], s_2[l+1])\cdots(s_1[l+m], s_2[l+m])* \end{gathered}$$

s_1 denotes the to which vertex, and s_2 denotes the length of such edge.
Obviously, we can think of the adjacent table as a partially-filled adjacent matrix, with empty entries implying no existence of edges. Since the matrix is of $O(n)$ size, we can fill the blanks in $O(n)$ time, yielding the correspondent matrix. Vice versa, remove the $BiStr$ that equals 0 produces the adjacent matrix. Therefore, the two forms are polynomially related.

34.1-5

Proof. Suppose the running time of such polynomial algorithm is $O(n^{c_1})$, where n is the input size and c_1 is some constant. The subroutine's running time is $O(n^{c_2})$ (Symbols' meanings are alike). Consider the following situations.

- Call the subroutine for constant times c_3: The running time of such subroutine is bound in $c_3 \cdot O(n^{c_2})$. Literally, it is $O(n^{c_2})$ as $c_3 = O(1)$, thus making almost no difference to the overall time complexity $O(n^{c_1})$. Therefore, it remains a polynomial-time algorithm.
- Call the subroutine for polynomial times $O(n^{c_4})$: If the polynomial's degree is relevant to the input size n, for example, square the input x. Then subroutine of doing product runs for

  $$x^{2^{\log_2 x}} = x^x = 2^{x\log_2 x} = 2^{\log_2 x \log_2 x} = O(2^x).$$

  It is of exponential time complexity.

□

34.2-3

Proof. Since HAM-CYCLE $\in$ P, the following procedure can help us determine the vertexes in order, and it is still in P. Suppose the algorithm is named HAM-SOLVER(G), and it returns *true* if G has a Hamilton cycle, *false* if not.

```
    VERTEX-IN-HAM-CYCLE(G)
 1  Initialize an empty edge set V_H
 2  if HAM-SOLVER(G) == FALSE
 3      return NULL
 4  else
 5      for edges (u, v) in G.E
 6          Remove the edge from G.V
 7          if HAM-SOLVER(G) == TRUE
 8              V_H = V_H ∪ (u, v)
 9          Add (u, v) to G.V and retrieve original G
10  Sort out all the edges in V_H to form the cycle
11  Print the vertexes on them subsequently
```

The for-loop runs for $|G.E|$ time. Each loop calls HAM-SOLVER for one time. The total running time is still polynomial. Therefore, the statement is proved. □

34.2-4

Proof.

- Union: L_1 and L_1 are two languages. A_1 and A_2 are verifiers of them, respectively. We define the verifier for $L_1 \bigcup L_2$ as follows:

$$\{A_3((x,x'),(y,y')) : A_1(x,y) \vee A_2(x',y')\}.$$

  Then the union of two languages in NP is closed.

- Empty set: The conclusion is trivial.

- Conjunction: Alike above, We define the verifier for $L_1 \bigcap L_2$ as follows:

$$\{A_3((x,x'),(y,y')) : A_1(x,y) \wedge A_2(x',y')\}.$$

  Then the conjunction of two languages in NP is closed.

- Kleen star: We shall define a verifier A_3 as follows:

```
A3(x, y)
1  for i = 1 to n
2      if A1(x[i], y[i]) and
       y[i + 1] is a certificate of x[i + 1] in L*
3          CONTINUE
4      else
5          return FALSE
6  return TRUE
```

  Let $T(n)$ denote the running time for input of size n, and let cn^k be an upper bound on the time to verify that y is a certificate for x. Then we have:

$$T(n) \le \sum_{i=1}^{n} cn^k T(n-1),\ T(1) \le c$$

  It yields that:

$$\begin{aligned} T(n) &\le cn^{k+1} + \sum_{i=0}^{n-1} c' i^{k'} \\ &\le cn^{k+1} + c' n^{k'+1} \\ &= O\left(n^{\max(k,k')+1}\right) \end{aligned}$$

  Thus, by induction, the running time of A_3 is polynomial. Besides, it is assumed $|y| = O(|x|^c)$ for some constant c. Therefore, NP is closed under Kleene star.

- Complement: The property is yet unknown. For a language L, If A returns false for some certificate y, it doesn't necessarily suggest that y is a certificate of $\overline{L}$.

□

34.2-6

Proof. The graph as well as the potential Hamilton cycle can be coded by binary string, which has been discussed before. The certificate y is a series of vertex, thus $|y| = O(|x|^c)$, where c is a constant. The algorithm A traverses along the potential Ham-cycle from u to v. Dye each vertex black once encountering it. Besides check if it is actually a cycle. In the end, verify if all the vertexes dyed black. If all the answers are yes, $A(x,y) = 1$, else $A(x,y) = 0$. Therefore, language HAM PATH belongs to NP. □

32.2-11

Proof. Assumption: G^3 has a Hamiltonian path from u to v where u is a root node of every SPANNING-TREE(G) and $d(u,v) = 1$.

(1) $n \le 4$, trivial

(2) Suppose $n = k$, the assumption holds, then

(3) When $n = k+1$, for each spanning tree, we can adjust it to this tree: If delete a, the graph with k vertices has a Hamiltonian cycle $ua_1a_2\cdots a_kv$. The Hamiltonian cycle of G^3 is $uava_ka_{k-1}\cdots a_1u$.

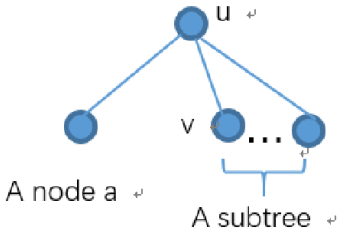

Figure 1: 34.2-11

□

32.3-2

Proof. Suppose that A_3 is a polynomial-time algorithm for decision problem L_3, F is a polynomial-time algorithm that calculates the reduction function f. Alike Figure 34-5 in the textbook, we can easily construct a polynomial-time algorithm A_2 for L_2. In the same way, we can also construct a polynomial-time algorithm A_1 from A_2 with the help of another reduction algorithm F' and the function f'. The overall time complexity is still polynomial. Therefore,

$$(L_1 \le_P L_2) \wedge (L_2 \le_P L_3) \implies (L_1 \le_P L_3),$$

i.e., the $\le_P$ relation is transitive. □

34.4-3

Proof. The formula could have $O(n)$ free variables, therefore, the truth table corresponding to it would a number of rows that is $O(2^n)$ since it needs to consider every possible assignment to all the variables. This means that the reduction as described is going to increase the size of such problem exponentially. Consequently, it can not yield a polynomial-time reduction.

□

34.4-5

Proof. Denote the disjunctive formula by $\vee_i \phi_i$, where each ϕ_i represents $v_1 \wedge v_2 \wedge \cdots v_k$ (v_i is either a variable or its negation). For each ϕ_i, we examine whether it has some variable and its negation, if so, then the formula is not satisfiable. Otherwise, it can be satisfied. The total running time of this process is in $O(n)$, where n is the size of all variables and/or its negations in the formula. Therefore, we prove that this problem is solvable in polynomial-time. □

34.4-7

Proof. Suppose that the original formula was $\wedge_i(x_i \vee y_i)$, and the set of variables were v_i. Then, consider the directed graph which has a vertex corresponding both to each variable, and its negation. Then, for each of the clauses $x \vee y$, place an edge from $\neg x$ to y, and an edge from $\neg y$ to x. Anytime an edge in the directed graph means that the vertex it originates is true, the vertex it points to should be true. Therefore, what we need to check is whether there exists a path from a vertex to its negation, or vice versa. If so, the formula is unsatisfiable. If not, then it is. In order to do this in a graph, we can run STRONGLY-CONNECTED-COMPONENT (The time complexity is $O(n)$) on the constructed graph. If any strongly-connected component contains both a variable v_i and $\neg v_i$. Then return false, otherwise, return yes. In this way, we solve this problem in polynomial time. Therefore, 2-CNF-SAT $\in$ P □

34.5-6

Proof. In Exercise 34.2-6, it has been proved that HAM PATH belongs to NP. Denote language HAM PATH by L, we shall prove that

$$\forall L' \in \text{NP}, L' \leq_P L.$$

To do this, we can prove that HAM CYCLE$\leq_P$HAM PATH, since the former problem shown to be NP-complete.
Let $G = (V, E)$ be any graph. We will construct a graph G' as follows. For each edge $e_i \in G.E$, let G_{e_i} denote the graph with vertex set $G.V$ and edge set $G.E - e_i$. Let e_i have the form $\{u_i, v_i\}$. Now, G' will contain one copy of G_{e_i} for each $e_i \in G.E$. Additionally, G' will contain a vertex x connected to u_1, an edge from $v_i to u_{i+1}$ for $1 \leq i \leq |G.E|-1$, and a vertex y and edge from $v_{|G.E|}$ to y. It is clear that we can construct G' from G in polynomial time with regard to the size of G. Therefore, HAM CYCLE $\leq_P$ HAM PATH, HAM PATH is NP-complete. □

Exercise 2.3.3.8 (Revised)

(i) A: Input: $(x, c) \in \{0, 1, \sharp\}^\star \times \Sigma^\star_{bool}$

(1) x is a representation of a graph and we represent the set of vertices as $\{x_1, x_2, \ldots, x_n\}$. c codes a sequence of vertices: $I = \{x_{i1}, x_{i2}, \ldots, x_{in}, x_{i1}\}$.

(2) If each pair of adjacent vertices have an edge, then A accepts (x, c) otherwise A rejects it.

(ii) A: Input: $(x, c) \in \{0, 1, \sharp\}^+ \times \Sigma^\star_{bool}$

(1) $x = u\sharp w \in \{0, 1, \sharp\}^+$ where $u \in \{0, 1\}^+$ and w represents a graph with n vertices. Let $k = Number(u)$. $c \in \Sigma^n_{bool}$ codes k vertices using k symbols of 1.

(2) If each edge of the graph is incident to at least one vertex in c, then A accepts (x, c), otherwise, A rejects it.

(iii) A: Input: $(x, c) \in \{0, 1, \sharp\} \times \Sigma^\star_{bool}$

(1) $x = u\sharp w \in \{0, 1, \sharp\}^+$ where $u \in \{0, 1\}^\star$ and w represents a graph. Let $k = Number(u)$. c codes k vertices.

(2) If each pair of the k vertices has an edge, then A accepts (x, c), otherwise, A rejects.