Complejidad Computacional
Reposición de Tarea 1

Karla Adriana Esquivel Guzmán
Andrea Itzel González Vargas
Luis Pablo Mayo Vega
Carlos Gerardo Acosta Hernández

Entrega: 23/03/17
Facultad de Ciencias UNAM

Ejercicios

1. En clase se presentó una descripción general de cómo codificar una máquina de Turing como una cadena binaria. Completa la descripción de tal forma que una máquina universal pueda utilizarla para imitar la máquina codificada.

Comencemos con la codificación de la máquina de Turing como una cadena binaria.
Sea $M = (Q, \Sigma, \Gamma, \delta, s, r, t, \vdash, \sqcup)$ una máquina de Turing, con

- $Q = i$
- $\Sigma = j$
- $\Gamma = k$
- s el m-ésimo estado
- t el n-ésimo estado
- r el p-ésimo estado
- $\sqcup$ el q-ésimo caracter de Σ
- $\vdash$ el r-ésimo caracter de Σ

Dicha especificación será codificada por la siguiente cadena binaria:

$$0^i 1 0^j 1 0^k 1 0^m 1 0^n 1 0^p 1 0^q 1 0^r 1. \tag{1}$$

Continuamos con la codificación de una transición en delta[1] de la forma $\delta(s_u, a_v) = (s_w, a_x, \rightarrow)$ como (2), mientras que una transición de la forma $\delta(s_u, a_v) = (s_w, a_x, \leftarrow)$ como (3):

$$0^u 1 0^v 1 0^w 1 0^x 1 0 \tag{2}$$

$$0^u 1 0^v 1 0^w 1 0^x 1 0 0 \tag{3}$$

De esta manera, podemos codificar completamente la tabla original de M.
Finalmente, una cadena de entrada en Γ^* de la forma $a_{n_1} \dots a_{n_z}$, quedará codificada por la cadena binaria:

$$0^{n_1} 1 0^{n_2} 1 \dots 0^{n_z} \tag{4}$$

Sólo resta decir para completar la codificación que podemos designar con $\langle M \rangle$ a la concatenación de (1) junto con las codificaciones de la forma (2) y (3), mientras que a la entrada codificada como en (4), la designaremos $\langle \alpha \rangle$. Combinadas en una sola cadena, podemos usar un símbolo de separación predefinido como # para obtener: $\langle M \rangle \# \langle \alpha \rangle$.

Recordemos que la Máquina Universal de Turing U que simulará a M, consta de tres cintas, además de la de entrada y salida. La primera de ellas será utilizada para la descripción de M, es decir, la tabla que quedó codificada para cada transición δ como en (3); la segunda, contendrá lo que corresponde con la cinta original de M; y la tercera el estado en el que la M se encontraría.
Al iniciar U, en la primera cinta ya debe tener la tabla de M codificada, mientras que en la tercera cinta aparecerá un 0, indicando que se encuentra en el estado inicial de la máquina original.
Separados por unos, podemos considerar una cadena binaria proveniente de una codificación de transición en δ, como una quinteta, por los 5 elementos en total que componen una transición. Para emular a M, la Máquina Universal de Turing U, deberá hacer lo siguiente por cada transición de la TM original:

1. Localiza en la primera cinta aquella quinteta en la que el primer componente sea igual al que se encuentra en la tercera cinta y el segundo componente sea igual al que se encuentra en la segunda cinta.

2. Copia a la tercera cinta el tercer componente de la primera cinta (el estado al que se transfiere).

3. Sustituye la segunda cinta con lo que tiene el cuarto componente.

4. Interpetra el movimiento de la cabeza que se encuentra en el quinto componente, moviendo acorde la cabeza de la segunda cinta.

[1] Con las cadenas finales, después del último 1 separador $0 := \rightarrow$ y $00 := \leftarrow$.

2. Demuestra que los siguientes problemas sobre gráficas (es decir, los lenguajes respectivos) están en P (elige la representación que prefieras para las gráficas):

- CONNECTED: el conjunto de todas las gráficas conexas.

  Para saber si el problema está en P tratamos de encontrar un algoritmo que corra en tiempo polinomial con cualquier entrada:
  Sea G(V, E) una gráfica, para saber si es conexa podemos utilizar el algoritmo BFS (Breadth-First Search), con el cual se recorren los vértices en V, llegando a cada uno a través de sus vecinos a partir de un vértice inicial v_i. Si hay algún vértice v que no pueda ser alcanzado desde v_i (i. e. no existe un camino entre v y v_i) entonces no podrá ser recorrido en el algoritmo y significará que la gráfica no es conexa. Para darnos cuenta de si esto sucede, basta con aumentar un contador por cada vértice que recorremos con BFS, si el contador es igual a |V|, entonces habremos recorrido exitosamente todos los vértices de G, y por lo tanto en tal caso G sería conexa, por otro lado si el contador es menor a |V| no se habrá alcanzado todos los vértices de la gráfica y por lo tanto no es conexa. BFS funciona de la siguiente manera:

  Empieza en v_i, lo mete en una pila y lo marca como visitado, procede a moverse a los vértices vecinos N_{v_i} de v_i, los marca como visitados y los mete también a la pila. A continuación se repite el procedimiento con los vértices en N_{v_i}, se recorren sus vecinos, se marcan como visitados y se meten a la pila. Si se llega a un vértice v que después de ser visitado, marcado y metido en la pila, nos damos cuenta de que todos sus vecinos ya han sido marcados como visitados, sacamos a v del tope de la pila y regresamos al vértice que ahora esté en el tope de la pila y seguimos recursando (nótese que en la implementación del algoritmo no es necesario regresarse físicamente arista por arista a éste vértice, ya que sólo con sacarlo de la pila podemos saber quiénes son sus vecinos si se tiene una buena implementación de la gráfica). Cuando se vacíe por completo la pila, habrá terminado el algoritmo.

  Notamos que cada vértice fue puesto en la pila una sóla vez y que se recorre cada arista máximo dos veces (ya que se vuelven a recorrer al sacar cada vértice de la pila), y por lo tanto la complejidad del algoritmo es $O(|V| + 2|E|) = O(|V| + |E|)$. Analizamos ahora este resultado. Sea $n = |V|$, el máximo número de aristas que puede tener una gráfica es $n(n-1)$ (si es dirigida) o $n(n-1)/2$ si no es dirigida. Por lo tanto la complejidad de BFS es a lo más $O(n + n(n+1)) = O(n^2)$, que es tiempo polinomial y por lo tanto $CONNECTED \in P$.

- BIPARTITE: el conjunto de todas las gráficas bipartitas, es decir, aquellas cuyos vértices puedan ser divididos en dos conjuntos A y B tales que todas las aristas en la gráfica tengan en un extremo un vértice de A y en el otro uno de B.

  Sea G(V, E) una gráfica. En este caso también podemos utilizar el algoritmo BFS, pero ahora marcamos a cada vértice que visitemos con uno de dos colores de la siguiente manera: coloreamos al vértice inicial con un color C1, a sus vecinos con otro color C2, a los vecinos de los vecinos de nuevo con C1, y así sucesivamente intercalando el color C1 con C2. De esa manera podemos poner a los vértices marcados con C1 en el conjunto A y a los marcados

con C2 en el conjunto B. Si ningún vértice termina con algún vecino coloreado con su mismo color, entonces la gráfica es bipartita, de otro modo no lo es. Si la gráfica no es conexa podemos hacer éste procedimiento sobre cada componente de la gráfica.

En el caso anterior habíamos visto que BFS corre en tiempo polinomial, y por lo tanto $BIPARTITE \in P$.