Algoritmos e Modelação Computacional

Paulo Mateus

Departamento de Matemática, Instituto Superior Técnico, UTL, Portugal

SQIG, Instituto de Telecomunicações, Portugal

Abril 2018

2

Aula 15

Maior subsequência comum

A sequência Z diz-se uma subsequência de X se existir uma sequência crescente $i_1 \dots i_k$ onde $k = |Z|$ e $X_{i_j} = Z_j$ para todo o $j = 1, ..., k$.

Por exemplo, a sequência $Z = BCDB$ é subsequência de $X = ABCBDAB$ em que a sequência de índices i é $2, 3, 5, 7$.

Problema: Dadas duas subsequências X e Y encontrar a maior subsequência comum a ambas, dita LCS(X,Y).

Tente resolver este problema em Java de forma naive. Neste caso a complexidade seria $O(2^n m)$ onde m é o tamanho X e n é o tamanho de Y.

Este problema tem aplicações muito relevantes para determinar se duas sequências de DNA são semelhantes.

Teorema: Seja $X = x_1 \ldots x_m$ e $Y = y_1 \ldots y_n$ sequências e $Z = LCS(X, Y) = z_1 \ldots z_k$. Então:

- Se $x_m = y_n$ então $z_k = x_n$ e $Z_{k-1} = LCS(X_{m-1}, Y_{n-1})$.
- Se $x_m \neq y_n$ e $z_K \neq x_m$ então $Z = LCS(X_{m-1}, Y)$.
- Se $x_m \neq y_n$ e $z_K \neq y_n$ então $Z = LCS(X, Y_{n-1})$.

Como resolver o problema? O teorema anterior indica como o fazer de forma recursiva com $X = x_1 \dots x_m$ e $Y = y_1 \dots y_n$ sequências:

$$LCS(X,Y) = \begin{cases} \varepsilon & \text{se } m = 0 \text{ ou } n = 0 \\ LCS(X_{m-1}, Y_{n-1}).x_m & \text{se } x_m = y_n \\ \text{longest}(LCS(X_{m-1}, Y_n), LCS(X_m, Y_{n-1})) & \text{se } x_m = y_n \end{cases}$$

A recursão pode ser facilmente adaptada para um algoritmo iterativo, por intermédio da construção de uma tabela.

```
public class LCS {
    public static void main(String[] args) {
        int[] X,Y;
        int m = X.length, n = Y.length;

        // opt[i][j] = length of LCS of X[1..i] and Y[1..j]
        int[][] opt = new int[m+1][n+1];

        // Calcula o comprimento do LCS e todos os seus subproblemas
        for (int i = 1; i <= m; i++) {
            for (int j = 1; j <= n; j++) {
                if (x[i-1] == y[j-1]) opt[i][j] = opt[i-1][j-1] + 1;
                else opt[i][j] = Math.max(opt[i-1][j], opt[i][j-1]);
            }
        }
```

```
        // reconstroi o LCS e guarda-o em Z
        int i = m-1, j = n-1;
        int l=opt[m][n];
        int [] Z=new int[l];

        while(i > 0 && j > 0) {
            if (x[i] == y[j]) {
                l--;
                Z[l]=x[i];
                i--;
                j--;
            }
            else if (opt[i-1][j] >= opt[i][j-1]) i--;
            else j--;
        }
    }
}
```

Problemas NP completos

O problemas da maior subsequência comum para um número arbitrário de sequências não tem solução eficiente (polinomial).

Na realidade pertence a uma classe de problemas para os quais não se conhece nenhuma solução polinomial - classe dos problemas NP completos.

O Clay institute oferece 1 000 000 USD se alguém encontrar um algoritmo polinomial para este problema ou provar que não existe.

A grande maioria dos investigadores em algoritmia/complexidade não acredita exista algoritmo polinomial para este problema.

Definição: Um programa M com entradas $x_1, \ldots x_k$, diz-se de *tempo polinomial* se existe um polinómio p e natural k tal que, para todo o $n \geq k$

$$\max_{\{x_1,\ldots x_k : \sum_{i=1}^{k} |x_i| \leq n\}} Time(M(x_1, \ldots, x_k)) \leq p(n).$$

Se existe um programa de tempo polinomial que induz a função característica de um conjunto $A \subseteq \mathbb{N}^k$ então diz-se que o problema da pertença a A é de *tempo polinomial*.

A classe $\mathcal{P}$ contém todos os conjuntos cujo problema da pertença é de tempo polinomial.

Por exemplo, o subconjunto dos pares está em P. Em 2004, foi demonstrado que o conjuntos dos números primos está em P.

Para definir a classe NP vai ser necessário mais um conceito.

Definição: Um programa M com entradas $x_1, \ldots x_k$, diz-se de *tempo polinomial para os primeiros $t < k$ argumentos* se existe um polinómio p e k suficientemente grande tal que para todo o $n \geq k$

$$\max_{\{x_1,\ldots x_k : \sum_{i=1}^{t} |x_i| \leq n\}} \mathrm{Time}(\xi(x_1, \ldots, x_k), M) \leq p(n).$$

Definição: Um conjunto $A \subseteq \mathbb{N}^k$ diz-se de *tempo polinomial não-determinístico* se existe um programa M com entradas $x_1, \ldots, x_k, x_{k+1}$, polinomial para os primeiros k argumentos, tal que:

- se $(x_1, \ldots, x_k) \in A$ então existe w tal que $f_M(x_1, \ldots x_k, w) = 1$;
- se $(x_1, \ldots, x_k) \notin A$ então para qualquer w tem-se que $f_M(x_1, \ldots x_k, w) = 0$.

A classe $\mathcal{NP}$ contém todos os conjuntos tempo polinomial não-determinístico.

Proposição: O conjunto

$$F = \{(x, y) \in \mathbb{N}^2 : x \text{ tem factores primos menores que } y\}$$

está em $\mathcal{NP}$

As definições de problemas em $\mathcal{P}$ e em $\mathcal{NP}$ estendem-se naturalmente para todos os tipos de dados discretos como grafos, fórmulas (circuitos booleanos), pois estes podem ser codificados em naturais.

Proposição: O problema do isomorfismo de grafos está em $\mathcal{NP}$

Proposição: O conjunto

$$LCM_o = \{(X_1, \ldots, X_n, k) : LCM(X_1, \ldots, X_n) > k\}$$

está em $\mathcal{NP}$

Exercício: Mostre que se LCM_o estiver em $\mathcal{P}$ então encontrar o comprimentos do LCM de $X_1, \ldots, X_n$ está em $\mathcal{P}$.

Redução polinomial

Definição: Uma redução polinomial entre conjuntos $A \subseteq \mathbb{N}^n$ e $B \subseteq \mathbb{N}^m$ é uma função computável $f : \mathbb{N}^n \to \mathbb{N}^m$ em tempo polinomial (isto é um para o qual existe um programa Java que dado x retorna $f(x)$ em tempo polinomial) tal que $x \in A$ sse $f(x) \in B$

Exemplos: Se $A \in \mathcal{P}$, então existe sempre uma redução polinomial de A para $\{1\}$. Se $A \in \mathcal{P}$, então existe sempre uma redução polinomial de A^c para A.

$$RSA = \{(x, y) \in \mathbb{N}^2 : x = pq \text{ com } p, q \text{ primos e } p < y\}$$

RSA reduz-se a F, mas está em aberto se F se reduz ao RSA.

Definição: Um conjunto B diz-se NP completo se é NP e qualquer conjunto NP A reduz-se polinomialmente a B.

Recorde o que um circuito Booleano é um grafo acíclico com dois nós distinguidos (entrada e saída) e tal que um nó n é indexados por uma função Booleana $f_n : 2^{i(n)} \to 2^{o(n)}$ onde $i(n)$ é o grau de entrada do nó e $o(n)$ é o grau de saída do nó n. Dado um circuito B define-se $f_B : 2^i \to 2^o$ a função induzida por B, em que i é o grau de saída do nó entrada e o é o grau de entrada do nó de saída.

Definição: Uma família de circuitos Booleanos $\{B_n : 2^{p(n)} \to 2^{q(n)}\}_{n\in\mathbb{N}}$ diz-se uniforme se:

- Existe um algoritmo que dado n retorna B_n;
- $p(n)$ e $q(n)$ são polinómios crescentes em sentido lato;
- $f_{B_k}(x) = f_{B_m}(x)$ qualquer que seja o $m > k$

A noção de circuito estende-se facilmente para quaisquer aridades. A complexidade de uma família de circuitos corresponde ao tempo de complexidade do algoritmo que constrói B_n ao que se adiciona o número de conectivos básicos do circuito (fan-out's, nand's).

Teorema [Postulado Church-Markov-Turing]: Uma família uniforme circuitos Booleanos têm o mesmo poder computacional que o Java (ou outro modelo computacional) do ponto de vista de algoritmos e a redução é polinomial.

Teorema Cook: A satisfação de circuitos é um problema NP-completo.