ĐỀ CƯƠNG BÀI GIẢNG
BÀI 5: THUẬT TOÁN XỬ LÝ XÂU KÝ TỰ

Xử lý xâu ký tự là một vấn đề cơ bản trong lập trình. Các yêu cầu xử lý xâu ký tự rất phong phú và đa dạng. Trong quá trình học lập trình người học đều gặp những bài toán xử lý xâu ký tự cơ bản như: Đếm ký tự, tìm kiếm ký tự, phân loại ký tự, đếm số từ trong xâu, chuẩn hóa xâu, tìm xâu con, v.v... Trong bài học này chúng ta sẽ tập trung vào việc tìm hiểu và ứng dụng một số thuật toán phát hiện xâu con trong một xâu ký tự, đây là một vấn để xử lý xâu ký tự điển hình.

5.1. Một số bài toán cơ bản

Bài toán 1: Phân loại ký tự

Cho một xâu ký tự s gồm các chữ cái, chữ số và dấu cách. Hãy cho biết số loại ký tự có trong xâu s và số lần xuất hiện của mỗi loại ký tự trong xâu s.

Ví dụ: Xâu ký tự s = “abccabdad1231”, có 7 loại ký tự và số lần xuất hiện của chúng như trong bảng sau:

Ký tự xuất hiện	Số lần xuất hiện
a	3
b	2
c	2
d	2
1	2
2	1
3	1

) Phương pháp vét cạn:

- Sử dụng hai mảng đầu ra là t[] kiểu ký tự và c[] kiểu nguyên để lưu trữ loại ký tự xuất hiện trong xâu s và số lần xuất hiện tương ứng của nó.
- Duyệt xâu s, mỗi ký tự s[i] kiểm tra xem s[i] có trong mảng t hay chưa.
    - Nếu s[i] đã có trong t[], giả sử tại vị trí t[m], tăng c[m] lên 1 đơn vị.
    - Ngược lại, thêm s[i] vào cuối mảng t[]: t[m] = s[i], c[m] = 1; tăng m lên 1 đơn vị.
- Kết quả trả về trong mảng t[] và c[].

) Thuật toán vét cạn:

//Hàm kiểm tra ký tự có trong mảng t hay không?

```
int indexOf(char *t, char key, int m) {
	for (int index = 0; index < m; index++) {
		if (t[index] == key)
			return index;
	}
	return -1;
}
//Hàm phân loại ký tự, kết quả trả về trong t[] và c[]
void solve(char *s, char *t, int *c, int &m) {
	m = 0;
	int n = strlen(s);
	for (int i = 0; i < n; i++) {
		int index = indexOf(t, s[i], m);
		if (index == -1) {
			t[m] = s[i];
			c[m] = 1;
			m++;
		}
		else {
			c[index] ++;
		}
	}
}
```

) Phương pháp sử dụng sàng ký tự

- Ta biết rằng để biểu diễn dữ liệu trong bộ nhớ máy tính người ta sử dụng 256 ký tự, mỗi ký tự tương ứng với một số nguyên từ 0 đến 255, gọi là mã ASCII của ký tự.
- Sàng ký tự là một mảng một chiều sang[], kiểu nguyên, có kích thước là 256 tương ứng với 256 ký tự trong bảng mã ASCII, với ý nghĩa chỉ số của phần tử mảng là mã ASCII của ký tự và giá trị phần tử mảng là số lần xuất hiện của ký tự trong xâu s. Ví dụ sang[97] = 3 với ý nghĩa 97 là ký tự ‘a’ và 3 là số lần xuất hiện của ký tự ‘a’ trong xâu s.
- Thuật toán như sau:

```
void sang_ky_tu(char *s, int *sang) {
```

```
        for (int i = 0; i < strlen(s); i++) {
            sang[s[i]]++;
        }
    }
```

Việc cài đặt chương trình ứng dụng đối với các thuật toán vừa trình bày ở trên xin được dành cho bạn đọc.

Nhận xét: Có nhiều phương án để giải quyết một bài toán, việc thay đổi cấu trúc dữ liệu có thể dẫn đến thay đổi thuật toán với cách viết gọn hơn nhiều.

Bài toán 2: Tìm xâu con

- Cho 2 xâu ký tự T, và P chỉ gồm các ký tự {'0' … '9', 'a' … 'z', 'A' … 'Z'}.
- Kiểm tra xem P có phải là một xâu con của T hay không?

Định nghĩa: Các xâu con chiều dài m của T là các xâu gồm m ký tự liên tiếp trong T bắt đầu từ T[i] ($0 \le i \le strlen(T) - m$).

) Mô tả thuật toán vét cạn để phát hiện xâu P trong xâu T

- Duyệt xâu T bắt đầu từ T[0] đến T[strlen(T) – strlen(P)].
- Với mỗi T[i]:
    - So sánh P với xâu con của T bắt đầu từ T[i], nếu vượt qua được xâu P thì dừng thuật toán và trả về i (trường hợp P là xâu con của T).
    - Nếu không vượt qua được xâu P thì kiểm tra i.
        - Nếu $i \le strlen(T) - strlen(P)$ thì tiếp tục so sánh P với xâu con của xâu T bắt đầu từ T[i + 1].
        - Ngược lại, dừng thuật toán và trả về -1 (trường hợp P không là xâu con của T).

) Cài đặt thuật toán

```
    int indexOf(const char *p, const char *t) {
        int m = strlen(p);
        int n = strlen(t) - m;
        for (int i = 0; i <= n; i++){
            int j = 0;
            while (j < m && t[i + j] == p[j]) {
                j++;
            }
            if (j == m)
                return i;
        }
```

```
        return -1;
    }
```

5.2. Thuật toán Boyer Moore Horspool

Thuật toán này được sử dụng để phát hiện xâu con P trong xâu T với thời gian thực hiện tốt hơn so với thuật toán vét cạn vừa trình bày trong phần trên.

Bài toán 2: Tìm xâu con

- Cho 2 xâu ký tự T, và P chỉ gồm các ký tự {'0' … '9', 'a' … 'z', 'A' … 'Z'}.
- Kiểm tra xem P có phải là một xâu con của T hay không?

) Phương pháp:

- So sánh xâu P lần lượt với các xâu con của T: Các xâu con của T gồm các ký tự liên tiếp trong T, bắt đầu từ T[j] ($0 \le j \le$ strlen(T) – strlen(P)) và có chiều dài bằng chiều dài của xâu P.
- Việc so sánh xâu P với các xâu con của T được thực hiện từ ký tư cuối của P trở về đầu xâu P.
- Giả sử vị trí bắt đầu so sánh trong T là T[i], và vị trí cuối cùng của P là P[k] (k = strlen(P) - 1).
- Ta sẽ so sánh T[i] với P[k] và dịch chuyển về đầu xâu P.
- Nếu việc khớp từng ký tự vượt qua được P[0] thì P có mặt trong T.
- Ngược lại (có T[i] != P[k]) thì kiểm tra xem T[i] có xuất hiện trong P không?
    - Nếu T[i] không có trong P, thì i = i + v (với v = strlen(P)), tức là dịch xâu P qua các xâu con của T mà có chứa ký tự T[i].
    - Ngược lại (T[i] có trong P): gọi x là vị trí xuất hiện đầu tiên của T[i] trong P (nghĩa là T[i] = P[x]), tính i = i + v – x – 1. Tức là dịch xâu P đến vị trí xâu con của T sao cho ký tự T[i] khớp với P[x].

Ví dụ:

- Xâu T = "mothaibabonnamsaubay".
- Xâu P = "namsau".
- v = strlen(P) = 6.
- Trường hợp này ta so sánh xâu con của T với xâu P bắt đầu từ vị trí T[i] với i = 5 trong T (quan sát hình vẽ dưới đây).

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
m	o	t	h	a	i	b	a	b	o	n	n	a	m	s	a	u	b	a	y
n	**a**	**m**	**s**	**a**	**u**														

Ở lượt so sánh này ta gặp T[i] ≠ P[k] (T[5] ≠ P[5]) và T[5] không xuất hiện trong P nên i = i + v, dịch P qua các xâu con của T có chứa T[5] (như hình bên dưới dây).

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
m	o	t	h	a	i	b	a	b	o	n	n	a	m	s	a	u	b	a	y
						n	**a**	**m**	**s**	**a**	**u**								

Ở lượt so sánh này gặp T[i] ≠ P[k] (T[11] ≠ P[5]) và T[11] xuất hiện trong P ở vị trí x = 0 (T[11] = P[0]), nên i = i + v – x – 1, dịch P đến vị trí xâu con của T sao cho T[11] khớp với P[0] (như hình bên dưới đây).

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
m	o	t	h	a	i	b	a	b	o	n	n	a	m	s	a	u	b	a	y
											n	**a**	**m**	**s**	**a**	**u**			

Đến đây ta nhận thấy rằng việc so sánh sẽ vượt qua được P[0], tức khớp được hoàn toàn xâu P với xâu con của T, nghĩa là P là một xâu con của T.

Ta có thể nhận thấy rằng thuật toán này nhanh hơn nhiều so với thuật toán vét cạn mà ta đã thiết kế trong phần trước.

) Cài đặt thuật toán

```
bool Boyer_Moore_Horspool(P, T) {
    int v = strlen(P), i = v - 1;
    while (i < strlen(T)) {
        int k = v – 1;
        while (k > -1 && T[i] == P[k]){
            i--; k--;
        }
        if (k < 0) {
            return true;
        }
        else {
            x = char_in_string(T[i], P);
            if (x < 0) {
                i = i + v;
            }
            else {
                i = i + v – x – 1;
            }
        }
    }
```

```
        return false;
    }
```

5.3. Thuật toán Z

Thuật toán này được sử dụng để đếm số lần xuất hiện cũng như xác định các vị trí xuất hiện của một xâu con P trong một xâu T.

Đầu vào của thuật toán Z là một xâu ký tự và đầu ra là một mảng Z (được định nghĩa bên dưới đây), sử dụng mảng Z ta sẽ xác định được số lần xuất hiện cũng như các vị trí xuất hiện của một xâu P trong một xâu T.

) Mô tả thuật toán Z

- Cho s là một xâu ký tự chỉ gồm các chữ số, chữ cái và dấu cách.

- Định nghĩa: Xâu tiền tố của xâu s là một xâu con dài nhất của xâu s tính từ vị trí s[i] bằng với phần đầu của xâu s ($1 \le i \le strlen(S) - 1$).

Ví dụ: Với xâu s = "**ACB**DAC**ACB**AC", ta có xâu con của s là **"ACB"** được gọi là một xâu tiền tố của xâu s vì xâu con này gồm 3 ký tự (dài nhất) bằng với 3 ký tự đầu tiên của xâu s.

- Thuật toán Z xây dựng một mảng Z[] với ý nghĩa Z[i] là độ dài của xâu tiền tố của xâu S bắt đầu từ vị trí S[i] ($1 \le i \le strlen(S) - 1$).

Với xâu ký tự s trong ví dụ trên theo thuật toán Z ta xây dựng được một mảng Z[] như sau:

index	0	1	2	3	4	5	6	7	8	9	10
s	A	C	B	D	A	B	A	C	B	A	C
Z	-1	0	0	0	1	0	3	0	0	2	0

Xâu con dài nhất giống phần đầu, từ vị trí này có độ dài 1 (A)

Xâu con dài nhất giống phần đầu, từ vị trí này có độ dài 3 (ACB)

Xâu con dài nhất giống phần đầu, từ vị trí này có độ dài 2 (AC)

***) Cài đặt thuật toán Z**

- Duyệt chuỗi từ vị trí i = 1 đến n − 1 (mảng bắt đầu từ 0).
- Mỗi vị trí i ta quản lí một đoạn [left, right] với right lớn nhất có thể sao cho xâu con từ left tới right là xâu tiền tố của xâu s.
- Khởi tạo: left = right = 0.
- Giả sử ở vị trí i ta đã có đoạn [left, right] của vị trí i – 1 và giả sử đã tính được tất cả các giá trị Z[1] đến Z[i – 1] trước đó.
- Chia 2 trường hợp để cập nhật left, right, và Z[i].
    - Trường hợp 1: Nếu i > right, trong trường hợp này không có tiền tố nào bắt đầu trước i và kết thúc sau i.
        - Gán left = i;

- Cho right chạy từ i trở lên để tìm đoạn [left, right] mới;
- Sau đó tính Z[i] = right – left;, right = right – 1.

- Trường hợp 2, ngược lại, i ≤ right:
  - Đặt k = i − left, ta thấy xâu S[k...] và xâu S[i...] giống nhau ở ít nhất right − i + 1 phần tử đầu, vì vậy có thể tận dụng Z[k] để tính Z[i].
  - Ta có: Z[i] ≥ min(Z[k], right − i + 1).
  - Nếu Z[k] < right − i + 1 thì Z[i] = Z[k]
  - Nếu Z[k] ≥ right − i + 1 thì:
    - Gán lại left = i và cho right tiếp tục tăng để tìm đoạn [left, right] mới.
    - Sau đó cũng gán Z[i] = right – left; right = right – 1; như trên.

Thuật toán Z như sau:

```
void z_algo(const char *s, int *z) {
    int n = strlen(s), left = 0, right = 0;
    for (int i = 1; i < n; i++) {
        if (i > right) {
            left = right = i;
            while (right < n && s[right - left]== s[right])
                right++;
            z[i] = right - left;
            right --;
        }
        else if (z[i - left] < right – i + 1) {
            z[i] = z[i - left];
        }
        else {
            left = i;
            while (right < n && s[right - left] == s[right])
                right ++;
            z[i] = right - left;
            right --;
        }
    }
}
```

***) Ứng dụng thuật toán Z**

Bài toán 3: Đếm xâu con

- Cho 2 xâu ký tự T, và P chỉ gồm các ký tự {‘0’ … ’9’, ‘a’ … ’z’, ’A’ … ’Z’}.
- Kiểm tra xem P xuất hiện bao nhiêu lần trong xâu T?

Ứng dụng thuật toán Z để giải bài toán 3 như sau:

- Tạo một xâu mới s = P + '$' + T trong đó ký tự '$' không có trong các xâu P và T.
- Áp dụng thuật toán Z để tìm độ dài các xâu tiền tố của xâu s.
- Độ dài của tiền tố nào (Z[i] nào) bằng với độ dài của xâu P thì tiền tố đó chính là xâu con P.

Ví dụ:

- Xâu T = “Ban Viet o Viet Nam”.
- Xâu P = “Viet”.
- Ta có xâu S = “Viet$Ban Viet o Viet Nam”.

Khi đó ta xây dựng được mảng Z như sau:

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
s	V	i	e	t	$	B	a	n		V	i	e	t		o		V	i	e	t		N	a	m
Z	-1	0	0	0	0	0	0	0	0	4	0	0	0	0	0	0	4	0	0	0	0	0	0	0

Ta thấy rằng ở các vị trí Z[i] có giá trị bằng 4, bằng chiều dài của xâu P thì xâu tiền tố đó chính là xâu con P.

Việc cài đặt chương trình ứng dụng thuật toán Z trong trường hợp này xin được dành cho bạn đọc, hãy xem đó như một bài tập.