АНАЛИЗ НА РЕШЕНИЕТО НА ЗАДАЧА
КРАСИВИ ЧИСЛА

Разглеждаме редицата от цифри разделена на части от повтарящи се последователни цифри в нея, т.е. например редицата 1 1 1 0 0 2 2 2 я преобразуваме до (1 . 3) (0 . 2) (2. 3) – като редица от точкови двойки, като във всяка точкова двойка първото число означава цифрата на поредната част от повтарящи се цифри, а второто число е броят на цифрите в тази част, т.е. дължината на този „сегмент" от повтарящи се цифри. След което започваме да разглеждаме последователно сегментите. Идеята е да си поддържаме една структура, която да съхранява временно образувано „красиво" число и от всички временно образувани числа да вземем най-голямото. Опитваме се да използваме и текущия сегмент за образуване на „красиво" число, което да съчетание от текущо образуваното такова и текущо разглеждания сегмент. Нека означим дължината на текущия сегмент с *k*, а дължината на сегментите в текущо образуваното „красиво" число с *l*. Тук попадаме в три възможни ситуации:

1) *k* = *l*. Дължината на текущия сегмент е равен на дължината на сегментите в текущо образуваното число. Тривиално образуваме ново текущо „красиво" число, включвайки текущия сегмент.
2) *k* > *l*. Дължината на текущия сегмент е по-голяма от дължината на сегментите в текущо образуваното „красиво" число. Тогава можем да вземем първите l на брой цифри от текущо разглеждания сегмент и да ги добавим към текущото „красиво" число и така да получим ново, валидно „красиво" число. След което трябва да проверим дали нямаме нов потенциален отговор (т.е. да сравним това „красиво" число с текущо намереното най-голямо такова). След което образуваме новото временно текущо число за следващата стъпка (т.е. при разглеждане на следващия сегмент) и то се състои само от текущия сегмент.
3) *k* < *l*. Дължината на текущия сегмент е по-малка от дължината на сегментите в текущо образуваното красиво число. Очевидно няма как да образуваме „красиво" число, което да е съчетание от текущото и новия сегмент. Затова, проверяваме дали текущото образувано „красиво" число не е по-голямо от текущо намереното най-голямо. След това образуваме новото текущо „красиво" число от последните k цифри на последния сегмент на текущото „красиво" число и k-те цифри от текущо разглеждания сегмент.

След като разгледаме последователно всички сегменти, отново правим проверка дали текущо пазеното „красиво" число в структурата не е по-голямо от текущо намереното максимално. Сложността на описания алгоритъм е O(N). Изглежда, предвид, че имаме сравняване на текущ отговор с нов предполагаем, сложността може да скочи до O(N^2), но това не е така, тъй като това сравнение се прави само в случаи, в които след сравнението „изчистваме" текущо разглеждането красиво число до един или два сегмента, с което лесно се вижда, че един сегмент участва в сравнение за кандидат-решение най-много два пъти. Въпрос на реализация е каква точно да е структурата в описания алгоритъм – тя може масив в който се пазят цифрите на „красивото число" и променлива за дължина на сегментите в „красивото" число. В примерната реализация, обаче, се използват курсори на начало и край в реалната редица на текущо разглежданото „красиво" число и текущия предполагаем отговор, както и отново променлива за дължина на сегментите в двете „красиви" числа.

Примерна реализация:

```
#include <stdio.h>

#define MAXN 1000000

int n, a[MAXN];
int reps[MAXN], next[MAXN];

int answ_start, answ_end, answ_rep;
int curr_start, curr_end, curr_rep;

struct TLargeNumber {
	int n, rep;
	int dig[MAXN];
};

void readInput() {
	int i;
	scanf("%d", &n);
	for (i = 0; i < n; i++)
		scanf("%d", &a[i]);
}

int getSequenceCount(int ind) {
	int count = 1, i;
	i = ind;

	while ((i+1 < n) && (a[i+1] == a[i])) {
		count++;
		i++;
	}

	return count;
}

void updateAnswer() {
	int i, j, answ_past, curr_past;
	int updateFlag = 1;

	if (curr_end - curr_start < answ_end - answ_start)
		return;

	if (curr_end - curr_start > answ_end - answ_start) {
		answ_start = curr_start;
		answ_end = curr_end;
		answ_rep = curr_rep;
	} else {
		answ_past = 0;
		curr_past = 0;
		i = answ_start;
		j = curr_start;
		while ((i <= answ_end) && (j <= curr_end)) {
			if (a[j] > a[i]) {
				updateFlag = 0;
				break;
			} else if (a[i] > a[j]) {
				break;
			}

			if (answ_past + next[i] - i < curr_past + next[j] - j) {
				answ_past += answ_past + next[i] - i;
				i = next[i];
			} else if (answ_past + next[i] - i > curr_past + next[j] - j) {
				curr_past += curr_past + next[j] - j;
				j = next[j];
			} else {
				answ_past += answ_past + next[i] - i;
				i = next[i];

				curr_past += curr_past + next[j] - j;
				j = next[j];
			}
		}

		if (updateFlag) {
			answ_start = curr_start;
			answ_end = curr_end;
			answ_rep = curr_rep;
```

```
                }
        }
}

void solve() {
        int i = 0, nextSeq;
        int rep;

        while (i < n) {
                reps[i] = getSequenceCount(i);
                nextSeq = i + reps[i];
                next[i] = nextSeq;

                while ((i+1 < n) && (a[i+1] == a[i])) {
                        i++;
                        reps[i] = reps[i-1];
                        next[i] = nextSeq;
                }
                i++;
        }

        answ_start = 0;
        answ_end = -1;
        curr_rep = 0;

        i = 0;
        while (i < n) {
                rep = reps[i];

                if (curr_rep == 0) {
                        if (a[i] != 0) {
                                curr_start = i;
                                curr_end = next[i] - 1;
                                curr_rep = rep;
                        }
                } else {
                        if (rep == curr_rep) {
                                curr_end = next[i] - 1;
                        } else if (rep > curr_rep) {
                                curr_end = i + curr_rep - 1;
                                updateAnswer();
                                if (a[i] == 0) {
                                        curr_rep = 0;
                                } else {
                                        curr_start = i;
                                        curr_end = next[i] - 1;
                                        curr_rep = rep;
                                }
                        } else {
                                updateAnswer();
                                curr_start = i - rep;
                                curr_end = next[i] - 1;
                                curr_rep = rep;
                        }
                }

                i = next[i];
        }

        if (curr_rep > 0)
                updateAnswer();

        for (i = answ_start; i <= answ_end; i++)
                printf("%d", a[i]);
        printf("\n");
}

int main () {

        readInput();
        solve();

        return 0;
}
```

Автор: Момчил Иванов