Wprowadzenie do Algorytmiki. Sortowanie oraz Bignumy.

Artur Laskowski

18 listopada 2021, Poznań

Wyniki

Zadanie \ Osoba		STATS	jakub_brambor	michal_chjnck	zuza_lawniczak	Kamil_Krzywon	Mikolaj_Nowak	AMIGHTYPANCAKE	mikolajl	kinga_burnejko	karmo104	jakub_klosinski	jan_kukorowski	piotrsuchodolski1	gaja_hinca1	musielinskikarol	JanNowakowski	olek_karminski5	dominika_jarz	gienon
WDA-01-2021	*[PP] The Blind Passenger*	94%	49	12	43	91	25	39	44	8916	55	344	50	7236		70	83	14388	5752	86
	JSZYCER - Szyfr Cezara	94%	12	42	19	9540	37	58	4055	9092	259	68	60	1584		58	10080	14395	85	21
	OSTSIL - Last digit of factorial	94%	39	83	81	10120	74	12704	8828	9045	5449	1583	6075	266		1915	24875	14437	11682	10084
	PP0502B - Tablice	100%	16	33	6	166	54	3174	4090	40	4469	1514	72	1501	37669	47	50	76	3086	25
WDA-02-2021	*[PP] PRIME_T - Liczby Pierwsze*	94%	5	2	6	52	12	10	4083	8749	151	38	29	7		4	55	5824	18	28
	DIVSUM - Divisor Summation	83%	1274	9	35		66	11412	4109	9006	6881	81	106	10183		49	9087		7295	2975
	PP0501A - NWD	94%	1	14	2	5999	6	64	4114	8765	6872	3127	44	16		6	19	36	29	33
	ADDREV - Adding Reversed Numbers	94%	17	18	12	12625	40	78	4123	10010	163	3215	82	86		24	71	69	3085	2982
	AGRESYWNE KROWY	72%	7121	103	228		310	14187	9639	13223	7182	7036	261	10015		46				12934
WDA-03-2021	*PZPI3 - Liczby znaczące*	44%	14360	1723	13262		297		11240	14647			3196			81				
	BFNTEST - Precz z komentarzami!	33%		1894			1714	14391	5753				1195			13231				
	[PP] Equation	50%	316	227	7116		402	14837	4271				293	12907		55				
	TICKETS 4	67%	2753	1555	2751		1463	14299	11670	14438		12691	1292	13239		126				443

Tips

```cpp
int main() {
    std::ios_base::sync_with_stdio(false);
    std::cout << "Hello, World!" << std::endl;
}
```

Sortowanie

- Sortowanie bąbelkowe
- Sortowanie przez wstawianie
- Sortowanie przez scalanie
- Bogosort
- QuickSort
- Sortowanie przez zliczania
- Sortowanie pozycyjne
- Sortowanie kubełkowe

Sortowanie bąbelkowe

Sortowanie Bąbelkowe

```
void bubble_sort(int n, int tab[]) {
    for(int i = 0; i < n; ++i) {
        for(int j = 1; j < n; ++j) {
            if(tab[j-1] > tab[j]) {
                swap(tab[j-1], tab[j]);
            }
        }
    }
}
```

Swap

```
void swap1(int &a, int &b) {
    int c = a;
    a = b;
    b = c;
}

void swap2(int &a, int &b) {
    if (a - b) a ^= b ^= a ^= b;
}

#include <algorithm>
```

Sortowanie przez wstawianie

Sortowanie przez wstawianie

```
void insertion_sort(int n, int tab[]) {
    for(int i = 1; i < n; ++i) {
        int j = i;
        while(j > 0 && tab[j-1] > tab[j]) {
            swap(tab[j-1], tab[j]);
            --j;
        }
    }
}
```

Stabilność sortowania

Sortowanie jest stabilne, jeśli dla dwóch elementów o tej samej wartości. W posortowanym ciągu są podane w tej samej kolejności, co w ciągu wejściowym.

Sortowanie przez scalanie

- Podziel ciąg na dwie części
- Posortuj każdą z nich
- Scal posortowane ciągi

Sortowanie przez scalanie

Sortowanie przez scalanie

```
void merge_sort(int first, int last, int tab[]) {
    if(first == last) {
        return;
    }
    int middle = (first + last) / 2;

    merge_sort(first, middle, tab);
    merge_sort(middle + 1, last, tab);
    merge(first, middle, last, tab);
}
```

Funkcja merge

```
void merge(int first, int middle, int last, int tab[]) {
    int left = first;
    int right = middle + 1;
    int res = first;

    int tmp[N];

    while(left <= middle && right <= last) {
        if(tab[left] <= tab[right]) {
            tmp[res] = tab[left];
            ++res, ++left;
        } else {
            tmp[res] = tmp[right];
            ++res, ++right;
        }
    }

    for(int i = left; i < middle; ++i) {
        tmp[res] = tab[i];
        ++res;
    }

    for(int i = right; i < last; ++i) {
        tmp[res] = tab[i];
        ++res;
    }

    for(int i = first; i < last; ++i) {
        tab[i] = tmp[i];
    }
}
```

Zliczanie inwersji w ciągu

Liczba inwersji to liczba par elementów, które w ciągu nie występują w kolejności posortowanej.

7 2 9 3 1

Powyższy ciąg posiada 7 inwersji.

Sortowanie szybkie

Sortowanie Szybkie

```
void quicksort(int first, int last, int tab[]) {
    int left = first;
    int right = last;
    int pivot = tab[(left+right)/2];

    do {
        while(tab[left] < pivot) {
            ++left;
        }
        while(pivot < tab[right]) {
            --right;
        }
        if(left <= right) {
            swap(tab[left], tab[right]);
            ++left, --right;
        }
    } while(left <= right);
    if(left < last) {
        quicksort(left, last, tab);
    }
    if(first < right) {
        quicksort(first, right, tab);
    }
}
```

Statystyki pozycyjne

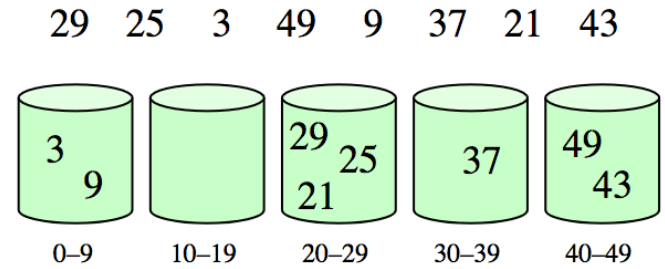
Dany jest nieposortowany ciąg n liczb.
Wyznaczyć k-tą co do wartości liczbę w ciągu.
Stosujemy algorytm QuickSort, ale po podziale sortujemy tylko tą część zbioru, do której należy k-ty element.

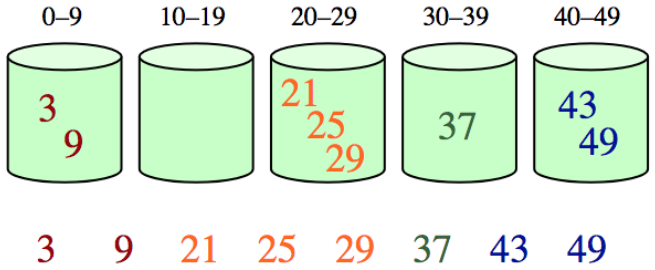
Sortowanie przez zliczanie

```
void counting_sort(int n, int tab[]) {
    int cnt[N];
    for(int i = 0; i < n; ++i) {
        ++cnt[tab[i]];
    }

    int pos = 0, it = 0;
    while(pos < n) {
        if(cnt[it] > 0) {
            tab[pos] = it;
            --cnt[it];
            ++pos;
        } else {
            ++it;
        }
    }
}
```

Sortowanie kubełkowe





Bogosort

- Sprawdź, czy sekwencja jest posortowana
- Jeśli tak to skończ
- Jeśli nie to wylosuj nową kolejność i wróć do punktu pierwszego

Sortowania wizualizacja

https://www.youtube.com/watch?v=kPRA0W1kECg

Arytmetyka dużych liczb

- Maksymalna wartość *long long* to $9'223'372'036'854'775'807 < 10^{19}$
- Dłuższe liczby możemy trzymać jako tablica kolejnych elementów np. *char*

Dodawanie

```
string dodaj(string a, string b) {
    string res = "";
    int carry = 0, i;
    for(i = 0; i < max(a.size(), b.size()); ++i) {
        int tmp = 0;
        if(i < a.size()) {
            tmp += a[i] - '0';
        }
        if(i < b.size()) {
            tmp += b[i] - '0';
        }
        tmp += carry;
        carry = tmp / 10;
        res += (tmp % 10) + '0';
    }
    if(carry > 0) {
        res += carry + '0';
    }
    return res;
}
```

Dodawanie

```
int main() {
    string a = "1834";
    string b = "543";
    std::reverse(a.begin(), a.end());
    std::reverse(b.begin(), b.end());
    string c = dodaj(a, b);
    std::reverse(c.begin(), c.end());
    cout << c << endl;
    return 0;
}
```

Dodawanie

- Liczba cyfr to $O(logn)$, gdzie n to wartość liczby.
- Pomysł na przyspieszenie?

Dodawanie z bazą

```
#define BASE 100
#define SIZE 10

long long c[SIZE];

void dodaj(long long a[], long long b[], int la, int lb) {
    for(int i = 0; i < SIZE; ++i) {
        c[i] = 0;
    }
    long long carry = 0, i;
    for(i = 0; i < max(la, lb); ++i) {
        long long tmp = 0;
        if(i < la) {
            tmp += a[i];
        }
        if(i < lb) {
            tmp += b[i];
        }
        tmp += carry;
        carry = tmp / BASE;
        c[i] +=  tmp % BASE;
    }
    if(carry > 0) {
        c[i] += carry;
    }
}
```

Dodawanie z bazą

```
int main() {
    long long a[] = {1, 6, 6};
    long long b[] = {3, 6};

    dodaj(a, b, 3, 2);

    for(int i = SIZE - 1; i >= 0; --i) {
        cout << c[i] << ", ";
    } cout << endl;
    return 0;
}
```

Dodawanie z bazą

- Jeżeli użyjemy dużej bazy eg. 10^9 to mamy 9-krotne przyspieszenie
- Mnożenie wykonajmy również "w słupku" - n razy wykonaj mnożenie liczby przez cyfrę i dodawaj
- Złożoność obliczeniowa mnożenia $O(n^2)$
- Przyspieszamy 81 razy przez lepszą bazę!

Tip

```cpp
typedef long long ll;

int main() {
    ll a = 123, b = 23456;
    ll c = a + b;
    cout << c << endl;
}
```

Laboratoria

https://www.hackerrank.com/wda-04-2020