ASD, wykład 05, 2022.03.24

Sortowanie Szybkie (Quick Sort)

Anthony Hoare – twórca

Pomysł

① Dzielimy elementy tablicy tak, że w lewej części są mniejsze bądź równe od tych w prawej części

② Sortujemy obie części rekurencyjnie.

Jak podzielić tablicę?

① Wybieramy element tablicy nazwijmy ten element pivot.

② Elementy ≤ pivot przesuwamy na lewo.

— " — ≥ pivot — " — na prawo.

Kod sortowania szybkiego

```
//sortuje tablice t[left, ...,right]
void _quicksort(int *t, int left, int right){
    if(left>=right)
        return;
    int i=partition(t,left,right);
    _quicksort(t,left, i-1);
    _quicksort(t,i+1,right);
}


quicksort(int *t, int end){
    _quicksort(t, 0, end-1);
}
```

Własności funkcji partition

Jeśli $i = partition(t, left, right)$

to

$\forall j<i \quad t[j] \leq t[i]$

$\forall j>i \quad t[j] \geq t[i]$

Oczywiście $j \in \{left, \ldots, right\}$

$t[i]$ - pivot

partition działa w czasie liniowym.

Partition

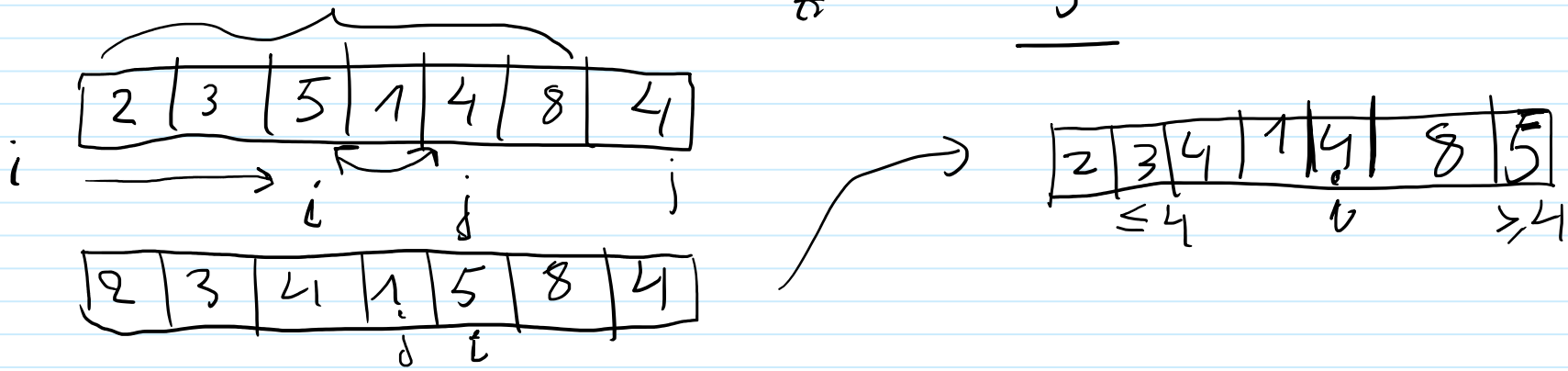
```
int partition(int *t, int left, int right){
      int v=t[right];
      int i=left-1, j=right;
      if(left>=right)
            return right;
      while(i<j){
        do{ i++;}while(t[i]<v && i<j);
        do{ j--;}while(t[j]>v && i<j);
        if(i<j)
          swap(t,i,j);
      }
      swap(t,right,i);
      return i;
}
```

① $v = t[right]$ – wybór pivota

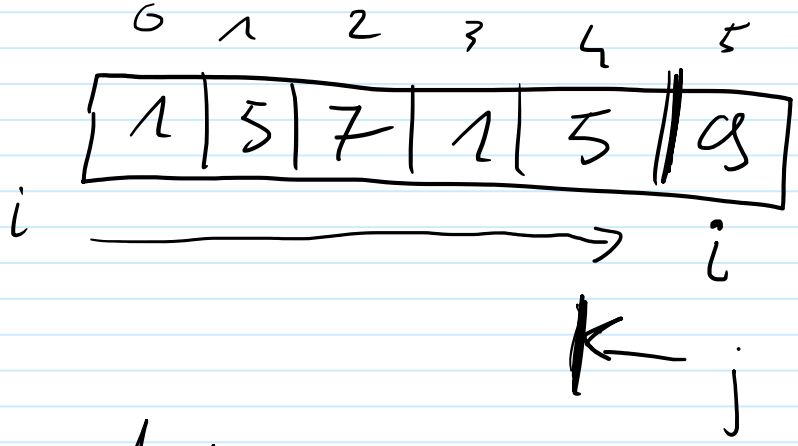
② Indeksami przebiegamy tablicę
i – od lewej, j – od prawej

③
$\forall k \; (left \le k \le i \Rightarrow t[k] \le pivot)$

$\forall k \; (j \le k \le right-1 \Rightarrow t[k] \ge pivot)$



Co jeśli pivot to np. największy element w tablicy?



partition zwróci 5 i wywołujemy rekurencyjnie

qs(0, 4)

qs(6, 5) ← sortujemy pustą tablicę.

Gdyby wejściowa tablica była już uporządkowana, to zawsze pivotem będzie największy element. Wtedy złożoność quick sort jest kwadratowa.

$$T(n) = T(n-1) + T(0) + C \cdot n$$

$$= T(n-2) + T(0) + T(0) + C \cdot n + C \cdot n-1$$

$$\dots$$

$$= n \cdot T(0) + C \cdot (1+2+\dots+n) = O(n^2).$$

Dlaczego wiesz quick sort jest tak szybki?

Q.S. jest szybki dla przypadku średniego.

Niech A - algorytm, $A: \Sigma^* \to \Sigma^*$

gdzie Σ - alfabet

$\Sigma^n = \{a_1 \dots a_n : a_i \in \Sigma\}$

$\Sigma^* = \bigcup_{n \in \mathbb{N}} \Sigma^n$.

$w \in \Sigma^*$

Niech $C_A(w)$ = "koszt" obliczenia A na wejściu w.

Pesymistyczny koszt działania A na wejściach długości n to

$$C_A^P(n) = \max\{C_A(\omega) : \omega \in \Sigma^n\}$$
$$|\omega| = n$$

Średni czas działania A na wejściach dł. n

$$C_A^{\acute{s}}(n) = \frac{\sum_{\omega \in \Sigma^n} C_A(\omega)}{\text{card}(\Sigma^n)},$$

$\text{card}(X)$ = moc zbioru X

Będziemy szacować średni czas działania Q.S. mierzony liczbą przestawień.

Załóżmy, że mamy w tablicy elementy $1, 2, \ldots, n$.

parami różne.

$$QS(w) = QS(w') + QS(w'') + O(n)$$

$|w| = n$ $\quad |w'| = k-1$ $\quad |w''| = n-k$

Q.S. tworzy dwie podtablice, w', w'' i wywołuje się rekurencyjnie

Jeśli każdy element zbioru $\{1, \ldots, n\}$ ma równe szanse zostać pivotem, to dla $k \in \{1, \ldots, n\}$ $|w'| = k-1$, $|w''| = n-k$

Niech $T(n)$ = średni czas działania Q.S.

$$T(n) = \frac{1}{n} \sum_{k=1}^{n} \left(T(k-1) + T(n-k) + \min(k-1, n-k) \right)$$

$$= \frac{2}{n} \sum_{k=0}^{n-1} T(k) + \frac{1}{n} \sum_{k=1}^{n} \min(k-1, n-k)$$

$$\leq \frac{2}{n} \sum_{k=0}^{n-1} T(k) + \frac{2}{n} \cdot \sum_{k=1}^{\frac{n}{2}} k \quad \text{, niech } n \text{ - parzyste}$$

$$= \frac{2}{n} \sum_{k=0}^{n-1} T(k) + \frac{2}{n} \cdot \frac{\frac{n}{2}\left(\frac{n}{2}+1\right)}{2}$$

$$n \cdot T(n) = 2 \cdot \sum_{k=0}^{n-1} T(k) + \frac{n\left(\frac{n}{2}+1\right)}{2}$$

$$n \cdot T(n) = 2 \cdot \sum_{k=0}^{n-1} T(k) + \frac{n(n/2+1)}{2}$$

$$n \cdot T(n) = 2 \cdot \sum_{k=0}^{n-1} T(k) + \frac{n(n+2)}{4}$$

$$(n-1)T(n-1) = 2 \cdot \sum_{k=0}^{n-2} T(k) + \frac{(n-1)(n+1)}{4}$$

$$n \cdot T(n) - (n-1)T(n-1) = 2 \cdot T(n-1) + \frac{n(n+2)}{4} - \frac{(n-1)(n+1)}{4}$$

$$n \cdot T(n) = (n+1)T(n-1) + \frac{n^2+2n-n^2+1}{4}$$

$$\frac{2n+1}{4} \le \frac{2n+2}{4} \le \frac{n+1}{2}$$

$$nT(n) \le (n+1) \cdot T(n-1) + \frac{n+1}{2}$$

$$n \cdot T(n) \leq (n+1) \cdot T(n-1) + \frac{n+1}{2} \quad / : (n(n+1))$$

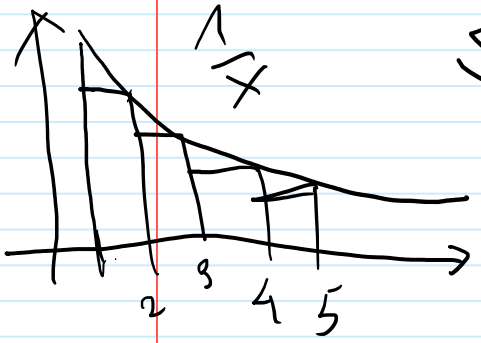
$$\boxed{\frac{T(n)}{n+1} \leq \frac{T(n-1)}{n} + \frac{1}{2} \cdot \frac{1}{n}}$$

$\leq$ dla dowolnego n.

$$\leq \frac{T(n-2)}{n-1} + \frac{1}{2} \cdot \frac{1}{n-1} + \frac{1}{2} \cdot \frac{1}{n}$$

$$\vdots$$

$$\leq \frac{1}{2}\left(\sum_{k=1}^{n} \frac{1}{k}\right) \leq \frac{1}{2} \int_{1}^{n} \frac{1}{x} dx + \frac{1}{2} \cdot 1$$



$$\leq \frac{1}{2} \cdot \ln(n) + \frac{1}{2}$$

Więc $\frac{T(n)}{n+1} \leq \frac{1}{2} \log(n) + \frac{1}{2}$

$$T(n) \leq \frac{1}{2} \cdot (n+1) \cdot \log(n) + \frac{n+1}{2}$$

$$\leq 0{,}7 \cdot n \cdot \log_2(n)$$, dla odpowiednio dużych n

Więc średnio przestawień będzie bardzo mało.

Można też pokazać, że w zdegenerowanej większości przypadków Q.S. będzie działać szybko.

Modyfikacje / warianty sortowania szybkiego

① Żeby Q.S. działał szybko, dobrze jest postarać się o wybór pivota t.z. będzie on, z dużym prawdopodobieństwem, blisko mediany tablicy.

② Można wybrać trzy elementy (np. skrajnie prawy i lewy i ze środka) i jako pivot przyjąć ten o wartości pośredniej spośród skrajnych, np. wybieramy 3, 10, 12 |→ → pivot = 10.

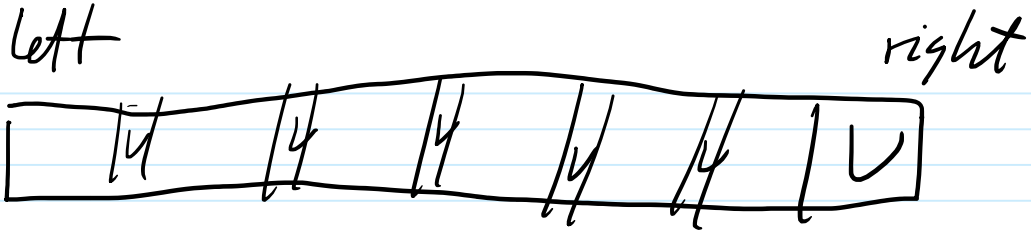
b) Można te elementy losować
ale losowanie jest kosztowne.

c) Można policzyć podłogę średniej
z tych elementów ale tedy
pivot może nie być elementem tablicy.

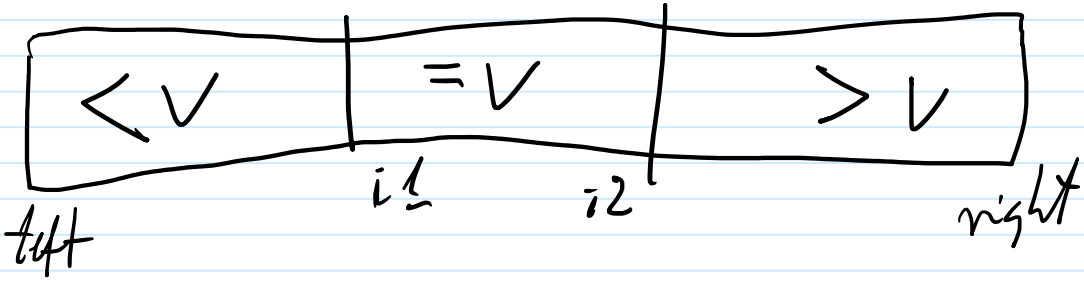
d) Można, w czasie liniowym, znaleźć
medianę tablicy i wziąć ją jako
pivota.
Ale taki wybór pivota jest zbyt kosztowny.

(2) Dla małych tablic warto jest wybrać inne, proste sortowanie (np. selection sort, ...)

(3) „Problem flagi holenderskiej" (Dijkstra).
Jeśli pivot, jego wartości, powtarza się często w tablicy, to warto przenieść te elementy do odcinka zawierającego pivot (w miejsce podziału).



partition = problem flagi holenderskiej



qs(t, left, right)

qs(left, i1-1)

qs(i2+1, right)

Zmniejszamy rozmiary problemów do rekursji.

④ Można pozbyć się jednego wywołania rekurencyjnego i zamienić je na pętlę. (tzw. rekursja ogonowa).

⑤ Możemy zagwarantować sobie, żeby głębokość rekursji była $\leq \log_2(n)$