# Algorytmy i Struktury Danych, 6. ćwiczenia, rozwiązania z serii 5

2024-11-13

## Zadanie 5.1

Mamy do wykonania $n > 0$ zleceń. Dla każdego zlecenia znamy termin t, do którego ma zostać ono wykonane oraz zysk z za jego wykonanie, jeśli zlecenie zostanie zrealizowane w terminie. Za zlecenie niezrealizowane w terminie zysk jest zerowy. Termin jest dodatnią liczbą całkowitą określającą numer dnia, do którego zlecenie ma być wykonane. Zyski są dodatnimi liczbami całkowitymi. Pierwszy dzień wykonywania zleceń ma numer 1. Jednego dnia można wykonać co najwyżej jedno zlecenie.

a) Zaproponuj wydajny algorytm, który obliczy kolejność wykonywania zleceń dającą największy sumaryczny zysk.

b) Wykonaj to samo co w a), przy założeniu że zysk z wykonania każdego zlecenia wynosi 1.

**Rozwiązanie:**

**a) Pierwsze rozwiązanie:**

- Posortuj zadania nierosnąco wg zysku.
- dla każdego zadania o terminie wykonania $t$, znajdź maksymalne $1 \leq i \leq t$, takie, że $i$-ty slot jest jeszcze wolny
- jeśli takie $i$ istnieje to przydziel zadanie do tego slotu
- w przeciwnym przypadku zignoruj zadanie

Implementacja, np. drzewo przedziałowe.

**Alternatywne rozwiązanie:** Czas (pesymistyczny) $O(n \log n)$ używając leniwych kolejek dwumianowych. Szkic algorytmu:

- Dla każdego dnia, utrzymuj (leniwą) kolejkę dwumianową typu Max na zyski zawierającą wszystkie zadania kończące się tego dnia.
- Zaczynając od ostatniego dnia, zrób ExtractMax i zaplanuj maxa na ten dzień. Dodaj kolejkę tego dnia (z której maxa wyjęliśmy) do kolejki poprzedzającego dnia używając operacji union.

Używamy obserwację że jeżeli jakieś zadanie zostanie zaplanowany na jakiś dzień numer $i+1$, to zadania które jeszcze nie zostały zaplanowane i których czas wykonania jest $i+1$ zmieniają swój czas wykonania na $i$ (zakładając że $i$ jest jeszcze wolny). Dokładny opis algorytmu:

- Posortuj zadania wg niemalejących czasów wykonania. ($O(n \log n)$)
- Zainicjuj stos. Przejdź przez posortowaną listę. Dla każdego czasu wykonania, zbuduj leniwą kolejkę dwumianową zawierającą wszystkie elementy kończące się tego dnia. Kolejka jest typu Max na zyski i dodatkowo zapamiętuje dla którego dnia jest odpowiedzialna. Dodaj korzeń kolejki do stosu. ($O(n)$)
- Dopóki stos nie jest pusty i kolejka na górze stosu nie jest odpowiedzialna za dzień numer 0, wybierz następną kolejkę dwumianową $Q$ (więc kolejkę zadań o największym czasie wykonania). Niech $i$ będzie dniem dla którego $Q$ jest odpowiedzialna. Wyjmij z niej maksymalny element i zaplanuj zadanie na dzień $i$. Jeśli kolejka $Q$ jest teraz pusta, przejdź do następnej iteracji pętli. W innym przypadku, zmień odpowiedzialność kolejki na dzień $i-1$. Potem sprawdź czy następna kolejka $R$ w stosie też jest odpowiedzialna za dzień $i-1$. Jeśli tak, to wyjmij tamtą kolejkę i zrób union obydwu kolejek i dodaj sumę kolejek do stosu. Jeśli $R$ jest odpowiedzialną za mniejszy dzień niż $i-1$, to dodaj $Q$ do stosu. (Jedno przejście pętli: zamortyzowane $O(\log n)$. Pętla najwyżej $n$ razy przechodzona, koszt zamortyzowany robi się pesymistyczny: $O(n \log n)$.)

**b)** Jeśli wszystkie zlecania mają zysk równy 1, to można prościej:

- Posortuj zadania wg niemalejących czasów wykonania,
- dla każdego zadania o terminie wykonania $t$, niech $i$ to minimalny wolny slot,
- jeśli takie $i$ istnieje to przydziel zadanie do tego slotu
- w przeciwnym przypadku zignoruj zadanie

## Zadanie 5.2

Na początkowo pustej, zwykłej kolejce dwumianowej wykonujemy n operacji Insert. Dokonaj analizy kosztu zamortyzowanego Insert w tym przypadku

a) metodą kosztu sumarycznego,

b) metodą księgowania,

c) metodą potencjału.

Zadanie sprowadza się do analizy liczby operacji przy zwiększaniu o $k$ licznika binarnego.

### Analiza kosztu zamortyzowanego metodą potencjału

(Cormen, rozdział 17.3, strona 419)
Niech $\Phi_i$ oznacza potencjał po wykonaniu $i$-tej operacji. Zakładamy, że $\Phi_0 = 0$, oraz $\Phi_i \geq 0$.
Koszt zamortyzowany $i$-tej operacji oznaczamy przez:

$$\hat{c}_i = c_i + \Phi_i - \Phi_{i-1}$$

Koszt wykonania $n$ kolejnych operacji:

$$\sum_{i=1}^{n} \hat{c}_i = \sum_{i=1}^{n} (c_i + \Phi_i - \Phi_{i-1}) = \sum_{i=1}^{n} c_i + \Phi_n - \Phi_0$$

### Analiza kosztu zamortyzowanego dla licznika binarnego metodą potencjałów

(Cormen, 17.1, strona 414-416 i 421–422)
Jako funkcję potencjału wybieramy liczbę jedynek w liczniku. **Rozwiązanie:**
(Alternatywne)

**b)** 2 zł przy insercie: jeden dla zrobienie kopca, drugi na przyszły join.

**c)** Potentiał = liczba drzew. Jeśli w jakimś kroku robimy $k$ joiny ($c = O(k)$), to kasujemy $k - 1$ drzewa i dodajemy jedno nowe, więc $\hat{c} = O(1)$.

## Zadanie 5.3

Dokonaj analizy kosztu pesymistycznego operacji DeleteMin, Insert oraz DecreaseKey na kopcach Fibonacciego.

**Rozwiązanie:**

1. DeleteMin $\Theta(n)$ (wykonaj DeleteMin po $n$ operacjach Insert)
2. Insert: zawsze $O(1)$
3. DecreaseKey: $\Theta(logn)$ (czas wynosi $\Theta(k)$ gdzie $k$ to liczba wycinanych poddrzew w tym kroku Ponieważ poddrzewa są wycinane wg ścieżki korzeń-liść której wysokość jest $O(\log n)$, stąd $k = O(\log n)$.)

## Zadanie 5.4

Zaproponuj algorytm, który do n-elementowej kolejki priorytetowej dodaje k zadanych kluczy, w czasie

a) $O(\log^2 n + k)$ dla kopca zupełnego,

b) $O(\log n + k)$ dla kopca lewicowego,

c) $O(\log n + k)$ dla zwykłej kolejki dwumianowej.

**Rozwiązanie:** Kopiec lewicowy i kolejka dwumianowa: utwórz odpowiednią strukturę w czasie $O(k)$ a następnie wykonaj operację Union.
Dla zwykłych kopców:

- jeśli $k = O(n)$ to prostu budujemy kopiec od zera
- jeśli $k = o(n)$ to dodajemy nowe kluczy na koniec kopca po czym robimy down-heap (idąc od dołu do góry), ale tylko dla węzłów które mają w poddrzewie nowe klucze. Takich węzłów jest $O(k)$ (węzły w których w obu poddrzewach są nowe klucze)+$O(\log n)$ (węzły w których tylko w jednym poddrzewie są nowe klucze). Wszystkie operacje down-heap w $O(k)$ węzłach kosztują nas $O(k)$ (ten sam argument co przy tworzeniu kopca). Wszystkie operacje down-heap w $O(\log n)$ pozostałych węzłach kosztują $O(\log^2 n)$ (bo takie jest ograniczenie na długość ścieżki).

## Zadanie 5.5

Dla kolejki dwumianowej Q typu MIN zaproponuj implementację operacji IncreaseKey(Q,v,new_key), która polega na zmianie klucza w węźle v na większy. Twój algorytm powinien działać w czasie O(log n).

**Rozwiązanie:** W czasie $O(\log n)$ możemy usunąć węzeł $v$ z kolejki dwumianowej a następnie ponownie go dodać już z nową wartością

## Zadanie 5.6

Opracuj sposób implementacji kolejki typu FIFO (First In First Out) z pomocą dwóch stosów w taki sposób, żeby zamortyzowany koszt poszczególnych operacji na stosach był stały. Dokonaj analizy kosztu metodami księgowania i potencjału.

**Rozwiązanie:**

- Enqueue(x): S1.push(x)
- Dequeue(): If S2.empty() { move S1 to S2 (reverse order) } S2.pop()

## Zadanie 5.7

Na początkowo pustym, dynamicznym ciąg elementów dozwolone jest dodawanie i usuwanie elementów na początku i na końcu ciągu. Zaproponuj algorytm symulujący te operacje z pomocą trzech stosów. Dokonaj analizy zamortyzowanego czasu wykonywanych operacji.

**Rozwiązanie:** Chcemy za pomocą dwóch stosów (+ jeden pomocniczy) symulować kolejkę na której możemy wykonywać następujące operacje:

- dodaj element na początek kolejki,
- dodaj element na koniec kolejki,
- usuń element z początku kolejki,

- usuń element z końca kolejki.

Utrzymujemy dwa stosy, na czubku pierwszego jest koniec kolejki, na czubku drugiego początek kolejki. Dodawanie wykonujemy przez dodanie elementu na odpowiednim stosie. Przy usuwaniu, jeśli odpowiedni stos nie jest pusty, to po prostu zdejmujemy odpowiedni element, wpp. mamy sytuacje, gdzie wszystkie elementy są na jednym stosie. W takim przypadku przy pomocy pomocniczego stosu przenosimy połowę elementów na drugi stos.
Funkcja potencjału:

$$\Phi(S_1, S_2, S_{pom}) = ||S_1| - |S_2||$$

## Zadanie 5.8 – Tablice dynamiczne

W tym zadaniu rozważamy implementację stosu w tablicy $a[1..N]$, gdzie $N$ jest potęgą dwójki. Możesz przyjąć, że $a$ jest równoważne wskaźnikowi do tablicy. Elementy stosu znajdują się w tablicy $a$ na pozycjach od 1 do $n$.

**Algorytm 1:** Top

```
if n = 0 then
    return pusty stos
else
    return a[n]
```

**Algorytm 2:** Push($e$)

```
if n = N then
    N:=2N
    new(b[1..N])
    b[1..n]:=a[1..n]
    release(a[1..n])
    a:=b
n:=n+1
a[n]:=e
```

**Algorytm 3:** Pop

```
if n = 0 then
    return pusty stos
else
    n:=n-1
    if N ≥ and n = N/4 then
        N := N/2
        new(b[1..N])
        b[1..n]:=a[1..n]
        release(a[1..2N])
        a:=b
    return a[n]
```

Przy założeniu, że koszty alokacji i zwalniania pamięci są jednostkowe, a wykonanie przypisania b[1..n] := a[1..n] na tablicach wymaga wykonania n

jednostkowych przepisań elementów z a do b, dokonaj analizy kosztu zamortyzowanego poszczególnych operacji stosowych.

**Rozwiązanie:** Funkcja potencjału:

$$\Phi(n, N) = 2|n - N/2|$$

**Push** jeśli $n < N$ to koszt rzeczywisty i zmiana potencjału jest równa $O(1)$, jeśli $n = N$ to wykonujemy $n$ przypisań a zmiana potencjału jest równa $\Delta\Phi = n - 2$ ($\Phi_1 = n$, $\Phi_2 = 2$).