Zadania domowe. Zestaw 2.1

Maciej Poleski

20 listopada 2012

1

Wystarczy posortować. Załóżmy że `A[1..n]` i `B[1..n]` są posortowane nie rosnąco. Istnieje optymalne rozwiązanie takie że A[i] jest w tym samym filmie co B[i]. Weźmy 4 indeksy $0 < a < b \leqslant n$ i $0 < c < d \leqslant n$. Jeżeli a nie było by w tym samym filmie co c, to istnieje b i d takie że a jest z d i b jest z c (ponieważ wszystkie mniejsze indeksy są już zajęte). Skoro tablice są posortowane, to `A[a]`$=$`A[b]`$+x$ gdzie $x \geqslant 0$ oraz `B[c]`$=$`B[d]`$+y$ gdzie $y \geqslant 0$. Ale ponieważ

$$(A[b] + x)(B[d] + y) + A[b]B[d] = 2A[b]B[d] + A[b]y + B[d]x + xy \geqslant$$

$$\geqslant 2A[b]B[d] + A[b]y + B[d]x = A[b](B[d] + y) + B[d](A[b] + x)$$

więc nasze sparowanie jest nie gorsze od dowolnego innego.

Złożoność obliczeniowa to koszt sortowania, czyli $O(n \log(n))$. Pseudokodu nie będzie. Całe zadanie polega na przesortowaniu tablicy wskaźników do kolejnych elementów tablicy wejściowej. Wystarczająco efektywne algorytmy zostały omówione na MP.

2

Istnieje rozwiązanie optymalne, w którym każdy punkt jest możliwie blisko 0. Weźmy optymalne rozwiązanie o minimalnej odległości. Zaczynając od lewej strony (od 0) przesuwamy każdy punkt maksymalnie na lewo (tak aby nie zmniejszyć minimalnej odległości z jego lewej strony). W efekcie na pewno nie zmniejszymy minimalnej odległości z jego prawej strony. Wynika z tego, że skrajnie lewy punkt zawsze należy do jakiegoś rozwiązania optymalnego. Możemy poszukać minimalnej odległości binarnie.

```
A[0..n-1] <- tablica ze współrzędnymi punktów na osi liczbowej
if k = 0:
   return ∅
else if k = 1:
   return {A[0]}
sort(A)
l <- 0
r <- max(A) - min(A) + 1
while l < r:
   s <- (l+r)/2
   v <- 0
   a <- −∞
```

```
    for i <- 0 to n):
        if a + s <= A[i]:
            v <- v + 1
            a <- A[i]
    if v < k:
        r <- s
    else
        l <- s+1
s <- pusty zbiór
a <- −∞
for i <- 0 to n):
    if a + r-1 <= A[i]:
        s <- s ∪ {A[i]}
        a <- A[i]
        if |s|=k:
            return s
```

Na uwagę zasługuje funkcja `sort`. Aby uzyskać złożoność wymaganą w treści zadania można użyć algorytmu quicksort jako pivot wybierając punkt na środku przedziału (niekoniecznie istniejący!). Czyli algorytm przyjmowałby jako parametry przedział w tablicy (do posortowania) i odpowiadający mu przedział na osi liczbowej. Po wykonaniu $\log_2(\max_i |x_i|)$ razy operacji wybierania pivota uzyskamy przedział jednostkowy na osi liczbowej kończąc tym samym sortowanie danego przedziału. Jednak dla rozsądnych danych ($n > \max_i |x_i|$) efektywniej będzie zastosować standardowy algorytm działający w $n \log(n)$. Reszta algorytmu to wyszukiwanie binarne największej możliwej minimalnej odległości między dowolnie wybranymi zbiorem co najmniej k punktów. Na koniec konstruowany jest sam zbiór w oparciu o informacje o minimalnej odległości.

3