Lista 02 zadanie 02

Wiktor Hamberger
308982

5 kwietnia 2020

Niech tablica $I[i]$ przetrzymuje nasze odcinki jako pary $< p_i, k_i >$ posortowane rosnąco względem współrzędnej k_i. Aby obliczyć podzbiór S o największej mocy wykorzystamy następujący algorytm (zakładam, że jeżeli koniec jednego i początek drugiego odcinka się mają tą samą współrzędną, to one się przecinają):

```
Data:
n - ilość odcinków;
I[]-tablica odcinków (I[i].first = p_i, I[i].second = k_i);
Result:
S[] - największa możliwa tablica odcinków z I[], które się ze sobą nie
przecinają;
Function fun(int n, int I[]) is
    sort(I[]);
    last := -1;
    for i:=0 to n do
        if I[i].first>last then
            S.pushback(I[i]);
            last=I[i].second;
        end
    end
    return S[];
end
```

Rozwiązanie to wybiera zachłannie odcinki kończące się najwcześniej, ale pilnuje, żeby nie wziąć odcinka zaczynającego się wcześniej, niż koniec poprzedniego (zmienna last). Poprawność algorytmu można udowodnić nie wprost. Weźmy pewien podzbiór nieprzecinających się ze sobą odcinków I i nazwijmy go Z. Załóżmy, że $|Z| > |S|$. Z tego wynika, że dla pewnego $0 \leq x \leq n$ $\forall i < x \, (Z_i = S_i \wedge Z_x \neq S_x)$. Mamy teraz trzy opcje:

1. Koniec Z_x jest przed końcem S_x. Nie jest to możliwe, ponieważ algorytm rozpatrzyłby Z_x przed S_x i, skoro $Z_{x-1} = S_{x-1}$, dodałby Z_x do S

2. Koniec Z_x i S_x są w tym samym miejscu. Wtedy to inna para odcinków sprawia, że moce tych zbiorów są różne i należy przeglądać oba zbiory dalej w poszukiwaniu różnic.

3. Koniec Z_x jest za końcem S_x. Skoro $Z_{x-1} = S_{x-1}$ to wzięcie Z_x sprawia, że albo pomijamy jakiś odcinek wcześniej, albo zaczynamy szukać kolejnego później, co w obu przypadkach nie jest optymalne.

Z tego wynika, że nie istnieje taki podzbiór Z, więc S jest największym podzbiorem I w którym nie ma przecinających się ze sobą odcinków.

Pozostaje jeszcze kwestia złożoności obliczeniowej algorytmu. Posortowanie I może zostać wykonane przy użyciu sortowania przez scalanie ($O(nlog(n))$), a następnie algorytm wykonuje jedno przejście po każdym z odcinków ($O(n)$), więc sumryczna złożoność będzie wynosiła $O(n + nlog(n))$.