Anna Pacanowska, 306412

Użyjemy algorytmu zachłannego - wybieramy odcinek o najmniejszym końcu nieprzecinający się z już wybranymi odcinkami. Aby sprawdzić, czy odcinek przecina poprzednie, będziemy porównywać jego początek z maksymalnym dotychczas końcem.
Algorytm wygląda następująco:
1. Posortuj tablicę I niemalejąco według końców w czasie $n \log n$.
2. $S = \{I[1]\}$
3. $k_{max} = I[1]$.k
4. for $i \in 2, 3, ..., n$:
    if $I[i].p > k_{max}$
        S.add($I[i]$)
        k_{max} =max($I[i].k,\ k_{max}$)
5. Wynikiem jest zbiór S.

Aby udowodnić poprawność algorytmu, weźmy dowolne optymalne rozwiązanie R różne od S. Posortujmy oba rozwiązania niemalejąco według końców odcinków. Niech i będzie pierwszą pozycją, na której te rozwiązania się różnią. Mamy $S[i].k <= R[i].k$ - w przeciwnym przypadku odcinek $R[i]$ zostałby wybrany przez nasz algorytm. Mamy również
$S[i].p, R[i].p > max\{R[1].k, ..., R[i-1].k\}$. $S[i]$ nie występuje w dalszej części R, gdyż R jest posortowane według końców. W takim razie możemy zamienić $R[i]$ na $S[i]$ bez straty optymalności ani poprawności. W ten sposób otrzymujemy optymalne rozwiązanie z dłuższym wspólnym prefiksem z S. Powtarzając tą operację, otrzymamy S.

Teraz pokażemy złożoność obliczeniową podanego algorytmu. Krok 1. zostanie wykonany w czasie $O(n \log n)$. Kroki 2., 3., 5. zajmą czas O(1), a krok 4. - czas $O(n)$. Cały algorytm zostanie więc wykonany w czasie $O(n \log n)$.