Z2 • $T = (p_i, k_i)$ • $R = \emptyset$

• posortuj rosnąco po drugiej współrzędnej T

• dopóki $T \neq \emptyset$

  • weź pierwszy odcinek z T

  • dodaj go do R

  • usuń wszystkie pokrywające się z nim

• return R

Dowód poprawności:

Niech $S = \{1, 2, \dots, n\}$ to zbiór odcinków posortowanych rosnąco po drugiej współrzędnej. Załóżmy, że $A \subseteq S$ to optymalne rozwiązanie, którego pierwszy element (też posortowany) k_1 1. Zatem $B = (A \setminus \{k\}) \cup \{1\}$ to inne optymalne rozwiązanie, bo druga współrzędna elementu k_i 1-ego $\leq$ elementu k-tego oraz $|A| = |B|$.

Dodatkowo zauważmy, że jeśli A jest optymalnym rozwiązaniem, to $A' = A \setminus \{1\}$ jest optymalnym rozwiązaniem dla podproblemu $S = \{i \in S : k_i \geq k_1\}$.

Załóżmy nie wprost, że tak nie jest. Wtedy istnieje zbiór $B' \subseteq S'$ t. że $|B'| > |A'|$, czyli $B = B' \cup \{1\}$ daje nam $|B| > |A|$ co przeczy optymalności A.

Korzystając z tych dwóch stwierdzeń mamy, że w każdym kroku wybieramy analogiczny podproblem i wzięcie zachłannie najmniejszego odcinka daje poprawny wynik.

Z3

Algorytm:

- Dopóki $\frac{a}{b} > 0$:
  - bierzemy $\min(k)$ t.że $\frac{1}{k} \leq \frac{a}{b}$
  - $\frac{a}{b} \leftarrow \frac{a}{b} - \frac{1}{k}$

D-d różności składników sumy:

Załóżmy nie wprost, że nasz algorytm weźmie dwa razy $\frac{1}{a}$. Pokażę, że $\frac{1}{a} \cdot 2 \geq \frac{1}{a-1}$, czyli powinniśmy byli wziąć $\frac{1}{a-1}$, czyli sprzeczność.

$$\frac{2}{a} \geq \frac{1}{a-1}$$
$$\Updownarrow$$
$$2 \geq \frac{a}{a-1}$$
$$2 \geq \frac{a-1+1}{a-1}$$
$$2 \geq 1 + \frac{1}{a-1}$$
$$1 \geq \frac{1}{a-1} \quad a \geq 1, \text{ więc } \square$$

D-d skończoności algorytmu

W danym kroku mamy $\frac{a}{b} - \frac{1}{k} = \frac{ak-b}{bk}$.

Wiemy, że $\lim_{k \to \infty} \frac{1}{k} = 0$ i $\frac{a}{b} > 0$, czyli k zawsze istnieje.

Skoro $\frac{1}{k}$ jest największym ułamkiem jednostkowym t.że $\frac{1}{k} \leq \frac{a}{b}$, to $\frac{1}{k-1} > \frac{a}{b}$, czyli $a > ak - b$ co oznacza, że nowy licznik jest mniejszy od poprzedniego. Skoro a jest naturalne, to proces w końcu zakończy się na 1. $\square$

Z4

Algorytm:

result = 0

while $k > 1$ and G not empty:

- pokoloruj wszystkie liście (result += liczba liści)
- $G \leftarrow G$ - liście
- $k \leftarrow k - 2$

if $k == 1$:

result += 1

Dowód poprawności

Lemat 1

Każde optymalne kolorowanie k grafu G można sprowadzić do optymalnego kolorowania, gdzie każdy liść jest pokolorowany.

D-d:

Weźmy optymalne kolorowanie C i załóżmy, że istnieje liść v, który nie jest pokolorowany. Weźmy wierzchołek w, który jest najbliżej leżącym wierzchołkiem pokolorowanym (nie liść). Pokolorujmy v i odkolorujmy w. Załóżmy nie wprost, że teraz kolorowanie nie jest poprawne. Czyli istnieje ścieżka S t.że S ma więcej niż C kolorowych wierzchołków.

Dodaliśmy tylko jeden wierzchołek więc S musi zawierać v i nie może zawierać w. (bo inaczej przed zmianą byłoby już niepoprawne)

Ale skoro v jest najbliższym pokolorowanym wierzchołkiem, to istnieje ścieżka z v zawierająca wszystkie pokolorowane wierzchołki S czyli ma conajmniej $C+1$ kolorów. Sprzeczność.

Czyli możemy postępować analogicznie dla wszystkich niepokolorowanych liści i uzyskamy interesujący nas efekt.

Lemat 2

Gdy do optymalnie k pokolorowanego drzewa dodamy pokolorowane liście to otrzymamy optymalne kolorowanie $k' = k+2$ dla $G' = G + \text{liście}$.

D-d.

- Długość każdej ścieżki w wzrośnie max o 2 pokolorowane liście więc $k' = k+2$ będzie poprawne.

- Optymalność:

Załóżmy nie wprost, że istnieje lepsze kolorowanie w G' C. Skoro C jest lepsze od k', to w C musi być pokolorowany conajmniej jeden wierzchołek z $G = G' - \text{liście}$. (z lematu 1)

Czyli po odjęciu liści otrzymamy kolorowanie C' t.że

$$C' \geq C - 2 > k' - 2 = k \quad \text{, czyli } C' > k \quad ↯ \text{ bo } k \text{ optymalne}$$

Zatem możemy zacząć kolorowanie od $k=0$ i $k=1$ (zależnie, parzystości większego k)

Możemy teraz dodawać warstwami pokolorowane liście grafu, aż dojdziemy do docelowego k lub skończymy graf. Zauważmy, że to czy te warstwy liści policzymy w preprocessingu i będziemy szli od 0 lub 1 czy zaczniemy od pełnego grafu i będziemy usuwać liście nie ma znaczenia. Zatem to dowodzi poprawności algorytmu.

Złożoność: $O(n+m) = O(|E|+|V|)$

Z5

Lemat 1

W dowolnym momencie działania algorytmu nie powstanie cykl. (zakładamy że $c(e_i) \neq c(e_j)$, gdy $i \neq j$)

Dowód Załóżmy nie wprost, że podczas działania algorytmu w jakiejś spójnej składowej powstał cykl C.
Oznacza to, że powstał on wskutek połączenia w superwierzchołek $v_0, v_1, \ldots, v_n$. Niech $e_0, e_1, \ldots, e_n$ będą kolejnymi krawędziami łączącymi te wierzchołki. Z działania algorytmu dostajemy że

$c(e_0) < c(e_1) < c(e_2) < \ldots < c(e_n) < c(e_0)$ czyli sprzeczność.

Lemat 2

W każdym etapie działania algorytmu otrzymujemy dla każdego superwierzchołka MST.

D-d:

Sytuacja po ~~wykonaniu~~ dodaniu krawędzi.

1) Załóżmy, że istnieje taki superwierzchołek V_i t.że $V_i \neq MST(V(V_i))$.

Niech to MST to T. Istnieje zatem e_i t.że $e_i \in E(V_i) \wedge e_i \notin E(T)$.

Dodajmy to e_i do T. Uzyskujemy w ten sposób cykl.

E_i jest incydentna do wierzchołka v w tym cyklu i istnieje inna krawędź e_i' również incydentna do v. Ale skoro $e_i \in E(V_i)$ to $C(e_i) < C(e_i')$. Zatem po usunięciu e_i' uzyskalibyśmy drzewo rozpinające mniejsze niż T. Sprzeczność.

Wiemy zatem, że zachodzi sytuacja, w której każdy superwierzchołek jest MST. Rozważmy zatem k-tą iterację algorytmu, która jest pierwszą taką, że po scaleniu otrzymujemy drzewo spinające, które nie jest MST.

Rozważmy dwa zbiory krawędzi E_1 i E_2, odpowiednio przed i po wykonaniu k-tej iteracji. Niech V_i to ten superwierzchołek t.że $E(V_i) \neq E(T)$, gdzie T to MST na V_i. Zatem istnieje $e_i \in E(V_i) \wedge e_i \notin E(T)$.

Zauważmy, że ta e_i musiała zostać dodana w k-tym kroku (bo jest to pierwszy moment, gdy nie mamy MST oraz inaczej byłoby to sprzeczne z 1))

Zatem, gdy dodamy e_i do $E(T)$ otrzymamy cykl. A skoro e_i jest najmniejszą krawędzią incydentną do pewnego superwierzchołka to ta inna krawędź incydentna e_i' musi być większa, zatem po zastąpieniu jej e_i uzyskalibyśmy mniejsze drzewo co przeczy temu, że T to MST.

Z6)

Lemat pomocniczy

Dla dowolnego cyklu C w grafie i krawędzi $e \in C$, jeśli $c(e) > c(e_j)$ dla $e_j \in C$ t.że $j \neq i$, to e nie należy do żadnego MST.

D-d:

Załóżmy nie wprost, że $e \in C$ i $c(e)$ jest max w C oraz e należy do jakiegoś MST_1. Po usunięciu e z MST_1 utworzą się dwa poddrzewa T_1 i T_2. Skoro $e \in C$ to istnieje $e' \in C$ t.że końce e' należą do T_1 i T_2.

Wiemy, że $c(e)$ jest max, więc jeśli połączymy T_1 i T_2 przy pomocy e' uzyskamy MST' t.że $MST' < MST_1$, czyli sprzeczność.

Idea algorytmu:

Usuwamy z grafu krawędź e. Niech wierzchołki, które ona łączyła to u i v a nowy graf to G'.
Teraz możemy sprawdzić bfsem dochodzącym po krawędziach o wadze $< c(e)$ czy istnieje droga z u do v.

- Jeśli istnieje, to e leży na cyklu i jest najcięższą krawędzią na nim. Zatem z lematu wiemy, że e nie należy do żadnego MST.

- Jeśli nie istnieje droga między u i v. To e nie leży na cyklu lub nie jest maksymalną krawędzią na żadnym cyklu. Zatem e należy do jakiegoś MST.