# 3B – Kampania wyborcza

Autor zadania: Adam Gąsienica-Samek

Autor omówienia: Konrad Paluszek

## Treść

Mamy dany graf o $n$ wierzchołkach i $m$ krawędziach, którego wierzchołki są pokolorowane na $k$ kolorów. Chcemy wiedzieć, czy jest możliwe uzyskanie tego kolorowania, jeśli zaczniemy od grafu, w którym wszystkie wierzchołki są niepokolorowane, a następnie każdym kolorem dokładnie raz pomalujemy pewien spójny podzbiór wierzchołków (kolejność kolorów może być dowolna – nie jest podana).

## Rozwiązanie

Będziemy odtwarzać kolejność kolorowania od końca. Wierzchołki, które zostały pokolorowane ostatnim z użytych kolorów, nie zostały już później przekolorowane, tworzą zatem spójny fragment grafu. Jeśli chodzi o kolor przedostatni, to wierzchołki nim kolorowane mogły zostać później przekolorowane na kolor ostatni. Zatem w w grafie z pozostawionymi wierzchołkami jedynie kolorów przedostatniego i ostatniego wszystkie wierzchołki przedostatniego koloru są w jednej spójnej składowej. Podobnie dla każdego innego koloru $c$: w grafie z pozostawionymi wierzchołkami koloru $c$ oraz kolorów użytych później wszystkie wierzchołki koloru $c$ muszą być w jednej spójnej składowej. Zauważmy ponadto, że jeżeli jakiś kolor $c$ spełnia ten warunek dla jakiegoś zbioru $X$ kolorów wziętych później, to będzie go nadal spełniał jeśli jako kolory późniejsze weźmiemy jakikolwiek nadzbiór $Y$ zbioru $X$. Możemy więc zacząć od grafu z wszystkimi wierzchołkami pokolorowanymi. Następnie, dopóki istnieje kolor $c$ o takiej własności, że w podgrafie zawierającym tylko wierzchołki koloru $c$ oraz wierzchołki niepokolorowane wszystkie wierzchołki koloru $c$ są w jednej spójnej składowej, zmieniamy status wszystkich wierzchołków koloru $c$ na „niepokolorowany". Odpowiedź to `TAK` wtedy i tylko wtedy, gdy udało nam się w ten sposób dostać graf, w którym wszystkie wierzchołki są niepokolorowane.

Bezpośrednia implementacja tego algorytmu, sprawdzająca każdy kolor po kolei korzystając np. z algorytmu DFS, da łączną złożoność $O(n^2m)$, czyli zdecydowanie za wolno.

Żeby poprawić złożoność skorzystamy z algorytmu Find&Union. Będziemy utrzymywać spójne składowe wierzchołków tego samego koloru w grafie, w którym można przechodzić jedynie przez wierzchołki tego koloru oraz przez wierzchołki niepokolorowane, jak również spójne składowe wierzchołków niepokolorowanych w grafie, w którym można przechodzić jedynie przez wierzchołki niepokolorowane. Ponadto będziemy dla każdej składowej wierzchołków niepokolorowanych trzymać mapę z kolorów w numer jednego z wierzchołków danego koloru sąsiadujących z tą składową. Dla każdego koloru będziemy też trzymać liczbę składowych, do których należy, a za każdym razem kiedy ta liczba spada do 1 będziemy go dodawać do kolejki kolorów, które są gotowe do usunięcia.

W momencie, gdy zmieniamy wierzchołek na niepokolorowany, iterujemy się po wszystkich jego sąsiadach. Wszystkich, którzy są pokolorowani wrzucamy do nowej mapy dla danego wierzchołka. Następnie dla wszystkich jego niepokolorowanych sąsiadów chcemy połączyć te wierzchołki w jedną składową oraz połączyć wszystkie wierzchołki pokolorowane sąsiadujące z nimi. W tym celu wybierzemy mniejszą z map przypisanych tym niepokolorowanym wierzchołkom i iterujemy się po wszystkich kolorach w niej zawartych. Jeśli dany kolor znajduje się również w drugiej mapie, to łączymy składowe w Find⋃ jeśli nie, to dodajemy go do mapy drugiego wierzchołka.

Wykonywanie operacji w czasie proporcjonalnym do wielkości mniejszego zbioru gwarantuje nam, że przy każdym przejrzeniu wierzchołka rozmiar jego składowej rośnie co najmniej dwukrotnie, więc każdy wierzchołek zostanie przepisany co najwyżej $O(\log n)$ razy. Daje nam to łączną złożoność $O((n+m)\log^2 n)$ w przypadku użycia mapy opartej o drzewo BST, lub $(O(n+m)\log n)$ w przypadku użycia hash-mapy.