Lista 4

Kamil Matuszewski

22 marca 2016

1	2	3	4	5	6	7	8	9	10
	✓	✓	✓	✓	✓				

Zadanie 2

Ułóż algorytm który dla danego n-wierzchołkowego drzewa i liczby k pokoloruje jak najwięcej wierzchołków tak, by na każdej ścieżce prostej było nie więcej niż k wierzchołków pokolorowanych.

Dla każdego wierzchołka ustalimy warstwę. Warstwy naszego drzewa będą wyglądać następująco: w warstwie i będą wszystkie liście powstałe po usunięciu wierzchołków z poprzednich warstw. Teraz, dla danego k, jeśli k jest parzyste, to zaznaczamy wszystkie wierzchołki z pierwszych $\frac{k}{2}$ warstw. Jeśli k jest nieparzyste, to dodatkowo bierzemy dowolny niepomalowany wierzchołek. Wszystko to można zrealizować np trzymając kolejkę wierzchołków o stopniu 1. Usuwając kolejne warstwy aktualizujemy stopnie wierzchołków. Algorytm może wyglądać jakoś tak:

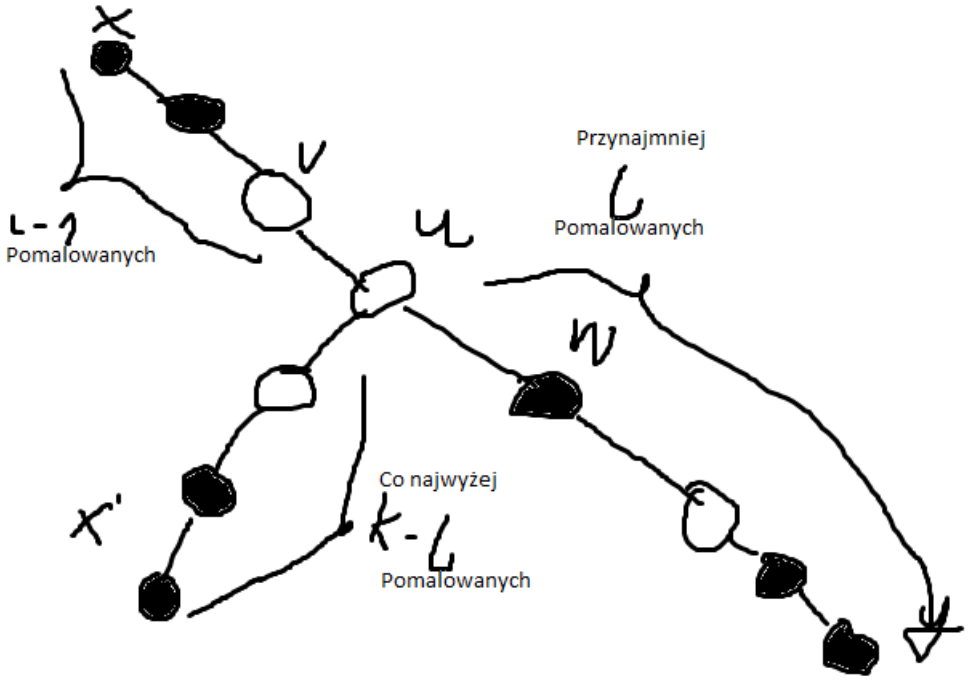
warstwa=1;
Dopóki istnieją jakieś wierzchołki:
  Dla każdego liścia v, włóż v do kolejki.
  Dla każdego v z kolejki, jeśli $warstwa \leqslant \frac{k}{2}$ pomaluj v. Usuń v z grafu (pomocniczego?).
  warstwa++
Jeśli $k \ mod \ 2 == 1$ pomaluj dowolny niepomalowany wierzchołek (o ile istnieje);

Lemat 1 : Algorytm jest optymalny.

Dowód. Pseudodowód: Weźmy rozwiązanie optymalne. Sprowadźmy je do naszego. Więcej wierzchołków w naszym rozwiązaniu pomalować się nie da, bo istniałaby ścieżka, w której jest więcej niż k wierzchołków pomalowanych. Dlatego rozwiązanie optymalne musi nie malować wszystkich wierzchołków z naszych $\frac{k}{2}$ warstw, w zamian malując inne. Weźmy wierzchołek z $\frac{k}{2}$ warstw który jest niepomalowany w rozwiązaniu optymalnym - dokładniej weźmy dowolny wierzchołek z najniższej możliwej warstwy. Tą warstwę nazwiemy l, wierzchołek v. Weźmy pomalowany w rozwiązaniu optymalnym wierzchołek, taki, że leży on powyżej warstwy l i pomiędzy nim a v nie ma żadnych wierzchołków. To, że taki istnieje jest oczywiste, bo $l \leqslant \frac{k}{2}$, oraz, jeśli by takiego nie było, to rozwiązanie optymalne byłoby mniej liczne niż nasze. Ten wierzchołek nazwiemy w. Sprawdźmy, że możemy pomalować v kosztem w nie psując naszego pokolorowania tj na dowolnej ścieżce będzie nie więcej niż k pomalowanych wierzchołków. Weźmy dowolną ścieżkę. Mamy sytuacje:

- Ścieżka nie zawiera v oraz w - zmiana pokolorowania w tym przypadku nic nie zmienia.
- Ścieżka zawiera v oraz w - znów, zmiana pokolorowania nic nam nie zmienia.
- Ścieżka zawiera w, nie zawiera v - "odmalowanie" w i pomalowanie v zmniejszy nam liczbę pomalowanych wierzchołków na tej ścieżce, więc z pewnością nic się nie popsuje.

- Ścieżka zawiera v nie zawiera w.
  Tu sprawa się lekko komplikuje. Oznaczmy ścieżkę $x\ x'$ naszą rozpatrywaną ścieżką, ścieżkę $x\ y$ ścieżką zawierającą zarówno v i w, a u jako wierzchołek, leżący pomiędzy v i w który rozdziela te dwie ścieżki (musi taki istnieć). Rozpatrzmy rozwiązanie optymalne. Na ścieżce u do y poza w było przynajmniej $l-1$ wierzchołków pomalowanych, jako, że l była najmniejszą warstwą z niepomalowanym wierzchołkiem. Do tego dochodzi w, mamy więc przynajmniej l wierzchołków pomalowanych na tej ścieżce. Na ścieżce u do x' nie może być więc więcej niż $k-l$ wierzchołków pomalowanych, bo w rozwiązaniu optymalnym na ścieżce x' do y nie mogło być więcej niż k wierzchołków pomalowanych. Teraz, dla ścieżki x do u mamy dokładnie $l-1$ wierzchołków pomalowanych - ponownie, przynajmniej $l-1$, a do tego pomiędzy v a w a w szczególności v a u nie ma wierzchołków pomalowanych. Stąd, na ścieżce x do x' w rozwiązaniu optymalnym mamy maksymalnie $k-l+(l-1)=k-1$ wierzchołków pomalowanych. Dokładając do tego v otrzymujemy k wierzchołków, więc nic się nie psuje.



Gdy k jest nieparzyste, odmalujmy dowolny wierzchołek. Mamy parzyste, możemy sprowadzić do rozwiązania optymalnego. Nasz algorytm maluje dowolny wierzchołek, więc sprowadziliśmy rozwiązanie do optymalnego.

Teraz, widać, że dowolne rozwiązanie optymalne można przekształcić tak, by w $\frac{k}{2}$ warstw wszystkie wierzchołki były pomalowane. Skoro nie da się pomalować więcej, to nasze rozwiązanie jest tak dobre jak formalne. □

Zadanie 3

Mamy n odcinków $I_j =< p_j, k_j >$, leżących na OX. Ułóż algorytm znajdujący zbiór nieprzecinających się odcinków o największej mocy.

Posortujmy odcinki względem k_j, tj względem końca odcinka. Bierzmy kolejne odcinki z tak posortowanej listy, o ile początek jest większy-równy końcowi poprzedniego.

Dowód. Mamy uporządkowanie A, załóżmy, że istnieje lepsze uporządkowanie B. Weźmy pierwsze odcinki które się różnią w naszych uporządkowaniach, tj $A_i \neq B_i$. Mamy następujące opcje:

- B_i kończy się wcześniej niż A_i. Wtedy nasz algorytm wybrałby ten odcinek. Skoro go nie wybrał, to B_i kończy się albo w tym samym miejscu albo później niż A_i co jest sprzeczne z założeniem.
- B_i kończy się w tym samym momencie co A_i. Wtedy to czy wybierzemy w tym kroku B_i czy A_i nie ma znaczenia, bo i w jednym i w drugim uporządkowaniu możemy wybrać te same elementy. Sprzeczne z tym, że B jest lepsze od A.
- B_i kończy się później niż A_i. Wtedy uporządkowanie B może być albo tak samo dobre (w sensie tak samo liczne) co A, lub gorsze (w sensie mniej liczne), gdyż kończy się później, i w najlepszym wypadku nie zmniejszy nam liczby możliwych do wybrania odcinków w kolejnym kroku, albo zablokuje nam dostęp do jakiegoś odcinka (odcinek skończy się później niż zacznie się jakiś inny). Sprzeczne z tym, że B lepsze od A.

□

Zadanie 4

Dla danych $a \leqslant b$ chcemy przedstawić $\frac{a}{b}$ jako sumę różnych ułamków o licznikach równych 1. Udowodnij, że algorytm zawsze znajdzie rozwiązanie, oraz, że nie będzie to rozwiązanie optymalne.

$$\frac{x}{y} = \frac{1}{\lceil y/x \rceil} + \frac{(-y \mod x)}{y\lceil y/x \rceil} = \frac{y + (-y \mod x)}{y\lceil y/x \rceil} \overset{*}{=} \frac{y + x\lceil y/x \rceil - y}{y\lceil y/x \rceil} = \frac{x\lceil y/x \rceil}{y\lceil y/x \rceil} = \frac{x}{y}$$

$$* \quad (n \mod k) = n - \left\lfloor \frac{n}{k} \right\rfloor k \Rightarrow (-n \mod k) = -n - \left\lfloor \frac{-n}{k} \right\rfloor k = -n - \left(-\left\lceil \frac{n}{k} \right\rceil \right) k = k\left\lceil \frac{n}{k} \right\rceil - n$$

Algorytm polega na rozpisywaniu ułamka na sumę dwóch ułamków: jednego o liczniku 1, a drugiego dowolnego. W następnym kroku rozpisujemy ten drugi ułamek (o ile jest taka potrzeba). Proces powtarzamy aż do uzyskania dobrej sumy (tj sumy ułamków o licznikach równych 1).

Lemat 1 : Algorytm zachłanny zawsze daje rozwiązanie.

Dowód. Zauważmy, że wystarczy sprawdzić, że nasz algorytm się skończy. Otrzymamy dobrą postać, ponieważ zawsze otrzymujemy jeden poprawny ułamek a na drugim ponownie wywołujemy algorytm (jeśli licznik $\geqslant 1$). Jeśli algorytm się skończy, będziemy mieli dobrą postać.
To, że algorytm się skończy, jest proste do pokazania. Zauważmy, że w dowolnym przypadku, $a \mod x < x$. Dodatkowo, $\mod x$ zawsze zwraca liczbę naturalną. Więc licznik naszego ułamka będzie się zmniejszał o przynajmniej 1. Skoro x jest liczbą naturalną, to po skończonej liczbie kroków $a \mod x = 1$, czyli otrzymamy poprawny ułamek.
Ps. Nie wiem co w wypadku jeśli $-y \mod x$ zwróci 0. To chyba trzeba rozpatrzeć osobno, ale to jest proste, bo wtedy otrzymamy ułamek o liczniku 0, czyli równy 0, czyli algorytm i tak się skończy. □

Lemat 2 : Algorytm zachłanny nie daje rozwiązania optymalnego.

Dowód. Kontrprzykład:

$$\frac{9}{20} = \frac{1}{3} + \frac{1}{9} + \frac{1}{180} = \frac{1}{4} + \frac{1}{5}$$

Pierwsza równość - rozwiązanie zachłanne. Druga równość - rozwiązanie optymalne. □

Zadanie 5

Udowodnij algorytm Borówki.

Tu można poczytać o borówce

Dowód. Po pierwsze, pokażmy, że w dowolnym kroku algorytmu nie powstanie nam cykl.
Załóżmy, że w którymś kroku algorytmu powstał nam cykl. Oznaczmy ten cykl jako C. Niech $v_1, v_2, \ldots, v_l$ będą kolejnymi wierzchołkami na tym cyklu, a $e_1, e_2, \ldots, e_l$ kolejnymi krawędziami(e_k łączy v_k i v_i, gdzie $i = k + 1 \ mod \ l$). Wtedy dla v_1 algorytm wybrał e_1, dla v_2 e_2 itd. Oznacza to, że ($C(e)$ to waga e):

$$C(e_1) > C(e_2) > \cdots > C(e_l) > C(e_1)$$

Mamy więc sprzeczność (zauważmy, że wagi są różne, dlatego nierówności są ostre).

Po drugie, dla dowolnego superwierzchołka jest on MST.
Pseudoindukcja?

W pierwszym kroku: załóżmy, że nasz superwierzchołek V nie jest MST. Weźmy MST i nazwijmy je T. Wtedy istnieje e, takie, że $e \notin E(T) \wedge e \in E(V)$. Dołóżmy tą krawędź do naszego T. Powstał nam cykl. Skoro tak, to istnieje krawędź e' incydentna do tego samego wierzchołka co e. Ale nasz algorytm dla tego wierzchołka wybrał e, więc $C(e) < C(e')$, więc jak wybierzemy e zamiast e' to dostaniemy drzewo o mniejszej wadze co daje sprzeczność z tym, że T to MST.

W kolejnych krokach: Weźmy pierwszy krok w którym nie dostaliśmy MST(zakładając że w każdym poprzednim mieliśmy MST dla dowolnego superwierzchołka). Podobnie, nazwijmy superwierzchołek V, a MST T. Nasz V składa się z jakichś spójnych składowych którymi są superwierzchołki z kroku poprzedniego. Teraz, w V istnieje e którego nie ma w T. e musi łączyć superwierzchołki z poprzedniego kroku, bo gdyby była wewnątrz jakiegoś z tych superwierzchołków, to wtedy ten superwierzchołek nie byłby MST co jest sprzeczne z zał. indukcyjnym. Skoro tak, to stosując argumentację analogiczną do tej wyżej, dochodzimy do sprzeczności.

□

Zadanie 6

Lemat(*circle property*) : jeśli krawędź e nie jest maksymalna na żadnym cyklu z G, to e należy do jakiegoś MST.

Dowód. Załóżmy, że e nie jest maksymalna na żadnym cyklu z G, i nie należy do MST. Mamy dwie opcje:

- e nie leży na żadnym cyklu.
  Wtedy w trywialny sposób e należy do MST, więc sprzeczność.
- e leży na jakimś cyklu.
  Weźmy MST i dołóżmy e. Postał nam cykl jako, że e nie należało do MST. Na tym cyklu jest jakaś krawędź maksymalna, nazwijmy ją e'. Jeśli ją zabierzemy dostaniemy MST o mniejszej wadze, jako, że $C(e) < C(e')$

□

Skoro już to wiemy, to nasz algorytm wygląda tak:
Wczytaj krawędź e łączącą wierzchołki v i w. Zapamiętaj jej wagę i usuń ją z grafu. Puść lekko zmodyfikowany DFS/BFS po tym grafie zaczynając od v. Lekko zmodyfikowany tzn: jeśli krawędź po której masz zamiar przejść ma większą wagę niż waga e, nie przechodź po tej krawędzi. Jeśli dotarłeś do w zakończ wypisując nie. Jeśli nie dotarłeś do w a nie masz już gdzie iść(algorytm się zakończył) wypisz tak.

Teraz czemu to działa. Jeśli nie dotarliśmy z v do w po krawędziach o mniejszej wadze niż e, to znaczy, że albo po usunięciu e graf się rozspójnił - wtedy e musi należeć do MST, albo e nie była maksymalna na żadnym cyklu G (bo gdyby była to byśmy przeszli po każdej krawędzi z tego cyklu, czyli w szczególności dotarlibyśmy do w), co z *lematu* oznacza, że e należy do jakiegoś MST.