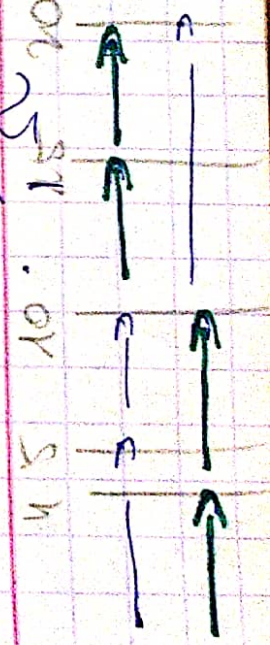
ZAD. 2 DANYCH JEST N ODCINKÓW $I_j = \langle p_j, k_j \rangle$, LEŻĄCYCH NA OSI OX, $j = 1, \ldots n$. UTWÓRZ ALGORYTM ZNAJDUJĄCY ZBIÓR $S \subseteq \{I_1, \ldots I_m\}$, NIEPRZECINAJĄCYCH SIĘ ODCINKÓW, O NAJWIĘKSZEJ MOCY.

$O(n \cdot \log n)$ { Sortujemy wzgl. k_j niemalejąco = X
ostatni_koniec = $-\infty$;

$O(n)$ { dla każdego x w X:
jeżeli x.p > ostatni_koniec
dodaj do S x
ostatni_koniec = x.k

return S



ZAD. 3 ROZWAŻ NASTĘPUJĄCĄ WERSJĘ PROBLEMU WYDAWANIA RESZTY: DLA DANYCH LICZB NATURALNYCH a, b ($a < b$) CHCEMY PRZEDSTAWIĆ UŁAMEK a/b JAKO SUMĘ RÓŻNYCH UŁAMKÓW O LICZNIKACH RÓWNYCH 1. UDOWODNIJ, ŻE ALGORYTM ZACHŁANNY ZAWSZE DAJE ROZWIĄZANIE. CZY ZAWSZE JEST TO ROZWIĄZANIE OPTYMALNE (TJ. O NAJMNIEJSZEJ LICZBIE SKŁADNIKÓW)?

PRZYKŁAD:

$$\left\lceil \frac{18}{5} \right\rceil = 4 \qquad \frac{5}{18} - \frac{1}{4} = \frac{10}{36} - \frac{9}{36} = \frac{1}{36}$$

$$\frac{7}{9} = \frac{1}{2} + \frac{1}{4} + \frac{1}{36}$$

$$\left\lceil \frac{9}{7} \right\rceil = 2 \to \frac{1}{2} \qquad \frac{7}{9} - \frac{1}{2} = \frac{5}{18}$$

ALGORYTM

```
S = { }
ułamek <- podany, który mamy rozbić na sumę ułamków egipskich
dopóki ułamek nie ma w liczniku 1:
    pomocnicza = ceil(ułamek^-1)
    do S dodaj pomocnicza^-1
    ułamek = ułamek - pomocnicza^-1
do S dodaj ułamek
return S
```

Musimy udowodnić, że algorytm się kończy:

Zauważmy co robimy:

$$\frac{a}{b} = \frac{1}{\lceil \frac{b}{a} \rceil} + \left(\frac{a}{b} - \frac{1}{\lceil \frac{b}{a} \rceil} \right)$$

zapisujemy to jako nowy ułamek

LEMAT.
Musimy pokazać, że $a > a' > 0$

$$\frac{a'}{b'} = \left(\frac{a}{b} - \frac{1}{\lceil \frac{b}{a} \rceil} \right) = \frac{a\lceil \frac{b}{a} \rceil - b}{b\lceil \frac{b}{a} \rceil}$$

$a > a'$

$a > a\lceil \frac{b}{a} \rceil - b$ — jakie b wstawiamy?

$b \in \langle ka-(a-1); ka \rangle$

czyli $b \in ka-(a-i)$, $i \in \langle 1; a \rangle$

$$a > a\left\lceil \frac{ka-(a-i)}{a} \right\rceil - (ka-(a-i))$$

$$a > a\left\lceil \frac{ka}{a} - \frac{a}{a} + \frac{i}{a} \right\rceil - ka + a - i$$

$$a > a\left\lceil k - 1 + \frac{i}{a} \right\rceil - ka + a - i$$

$$a > a(k-1+1) - ka + a - i$$

Mogłoby się wydawać, że skoro dzielimy i odejmujemy to zawsze znajdziemy rozwiązanie dla każdego ułamka. Jeśli nie odejmiemy od licznika przynajmniej jej połowy to się zapętlimy (nigdy nie skończymy)

$a > \cancel{ok - ko} + a - i$

$a > a - i \geq 0$

NIE JEST OPTYMALNY

$\frac{5}{21} = \frac{1}{5} + \frac{1}{27} + \frac{1}{945} + \ldots \leq \frac{1}{6} + \frac{1}{14}$

Prawda ponieważ $i \in \langle 1; a \rangle$.

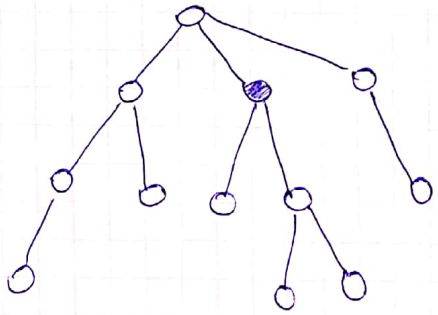
Liczniki wtanków, które wyliczamy w naszym programie kiedyś się skończą (na 1) co dowodzi temu, że algorytm jest skończony.

Czemu dojdą do 1 i się skończą? Ponieważ wykonujemy operacje odejmowania oraz operujemy na liczbach $\mathbb{N}$ czyli na zbiorze liczb, który jest dobrze uporządkowany.

Jeśli w dowolnym podzbiorze, zawsze istnieje element najmniejszy.

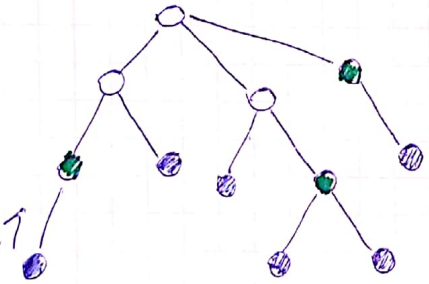
ZAD. 4 Utwórz ALGORYTM, KTÓRY DLA DANEGO N-WIERZCHOŁKOWEGO DRZEWA I LICZBY K, POKOLORUJE JAK NAJWIĘCEJ WIERZCHOŁKÓW TAK, BY NA KAŻDEJ ŚCIEŻCE PROSTEJ BYŁO NIE WIĘCEJ NIŻ K POKOLOROWANYCH WIERZCHOŁKÓW.

$k = 1$



Możemy pokolorować tylko jeden wierzchołek. Więcej nie możemy ponieważ wtedy będziemy mieli ścieżkę prostą łączącą te dwa (lub więcej) wierzchołków.

$k = 2$



Możemy pokolorować wszystkie liście, ponieważ dowolna ścieżka prosta przechodzi przez co najwyżej 2 liście (każdy liść ma 1 krawędź)

Dla $k=3$ jeden wierzchołek możemy dodatkowo.

Nie musi to być liść (cale musi być taki wierzchołek, że da się go zepchnąć do liścia i nie jest przodkiem innego pokolorowanego wierzchołka.)

Dla $k=4$ wychodzimy sobie, że pokolorowane liście z poprzedniego kolorowania (gdy k jest o dwa mniejsze) nie istnieją i kolorujemy liście drzewa dotąd nie pomalowanego.

Zatem ↓

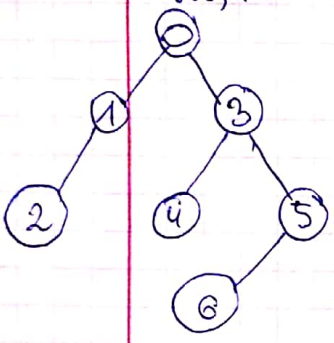
Jeżeli k jest PARZYSTE to:

- kolorujemy liście drzewa nie pokolorowanego

Jeżeli k jest NIEPARZYSTE to:

- wykonujemy krok dla PARZYSTEGO ~~tak~~ (k-1) tak długo jak się da, a potem dodajemy jeden wierzchołek

DANE: L - lista sąsiedztwa
n, k



L: 0 -> 1,3
1 -> 0,2
2 -> 1
3 -> 0,4,5
4 -> 3
5 -> 3,6
6 -> 5

Pokolorowane[n] = [0,0,...,0]
Dla i=0 dopóki i<k/2; i++
  Dla każdego wierz. v w L:
    Jeśli L[v].length == 1
      Pokolorowane[v] = 1
      u = L[v][0]
      usuń L[v][0]
      usuń v z L[u]

Jeśli k%2 == 1
  Pokoloruj dowolny wierzchołek.

ZAD. 5 UDOWODNIJ POPRAWNOŚĆ BORUVKI (SOLLINA)

CYCLE PROPERTY: Istnieje taki cykl C, że e jest najcięższą krawędzią na C
- wtedy e nie jest w MST

CUT PROPERTY: Istnieje A podzbiór V i B = V - A takie, że e jest najlżejszą krawędzią łączącą A z B.
Wtedy e jest w MST.

Zakładamy, że w grafie nie ma takich samych wag.

ALGORYTM BORUVKI wyznacza minimalne drzewo rozpinające dla grafu nieskierowanego.

Z wykładu:

- Dla każdego wierzchołka z G znajdujemy najkrótszą incydentną z nim krawędź, krawędzie te dołączamy do zbioru E'.
- Tworzymy nowy graf G'. Wierzchołki w G' (superwierzchołki) odpowiadają spójnym składowym w E'. Dwa superwierzchołki S_1 i S_2 łączymy krawędzią wtw., gdy jakiś wierzchołek z S_1 był połączony w G krawędzią z jakimś wierzchołkiem z S_2. Jako wagę tej krawędzi przyjmujemy minimalną wagę krawędzi w G

pomiędzy wierzchołkami z S_1 i S_2.
Za G przyjmujemy G' i przechodzimy do nowej fazy.

Zatem należy udowodnić, że na żadnym etapie algorytmu w MST nie pojawi się cykl (1). oraz, że w każdym super wierzchołku jest MST dla tego podzbioru grafu (2)

(2) Niemożliwe jest aby superwierzchołek nie był MST na podzbiorze wierzchołków oraz łączących ich krawędzi na danym przedziale, ponieważ algorytm działa tak, że dla każdego wierzchołka wybieramy mu najlżejszą incydentną krawędź.

Załóżmy, że istnieje wierzchołek, którego najlżejszą krawędzią incydentną jest krawędź u. Zatem wiemy, że (v,u) należy do rozwiązania, ale założyliśmy, że krawędź u nie występuje w MST. Zauważmy, że jeśli do MST dodamy krawędź u to powstanie cykl. Cykl ten będzie zawierał inną krawędź incydentną wierzchołka v, która będzie cięższa. Sprzeczność.

(1) Załóżmy, że w jednej ze spójnych składowych pojawił się cykl. Wiemy jednak, że nasz algorytm w każdym kroku dodaje najlżejszą spośród incydentnych do danego superwierzchołka krawędzi.
Zatem musiałoby zachodzić

$$waga(u_1) < waga(u_2) < \ldots < waga(u_n) < waga(u_1)$$

Sprzeczność.

* Dodatkowo istnieje tw., które mówi:
Jeżeli zbiór A jest podzbiorem V i B jest podzbiorem V takim, że $B = V/A$ to wiemy, że jeśli e to najmniejsza wagowo krawędź łącząca A i B to dodając ją do zbioru składającego się z MST na A i MST na B stworzymy MST na V.
Korzysta z tego twierdzenia Algorytm Boruvki, który za każdym razem gdy łączy kolejne superwierzchołki wybiera krawędź o minimalnej krawędzi.

ZAD.6 UTWÓRZ ALGORYTM, KTÓRY DLA DANEGO GRAFU G ORAZ KRAWĘDZI e SPRAWDZA W CZASIE O(n+m), CZY KRAWĘDŹ e NALEŻY DO JAKIEGOŚ MST GRAFU G. MOŻESZ ZAŁOŻYĆ, ŻE WSZYSTKIE WAGI KRAWĘDZI SĄ RÓŻNE.

Krawędź e łączy dwa wierzchołki v_1 i v_2.
Znając wagę krawędzi e usuwamy z grafu wszystkie wierzchołki, których waga $\geq e$.
Jeśli znajdziemy ścieżkę z v_1 do v_2 to

to znaczy że krawędź e nie jest w MST. Wpw należy.

odwiedzone = [0,0 ... 0]

odwiedź(wierzchołek v, krawędź e)
    odwiedzone[v] = 1
    dla każdego wierzchołka v2 w L:
        jeśli odwiedzone[v2] == 0 ORAZ
        waga(e) > waga(<v, v2>):
            odwiedź(v2)

czy_należy(u1, u2, e)
    odwiedź(u1, e)
    jeśli odwiedzone[u2] == 0
        ZWRÓĆ PRAWDA
    ZWRÓĆ FAŁSZ

Rozwiązanie to opiera się na twierdzeniu:

Dla dowolnego cyklu C w grafie i krawędzi e należącej do tego cyklu, jeśli waga e jest większa niż waga wszystkich innych krawędzi w C, to e nie należy do żadnego MST w tym grafie.

Chcemy sprawdzić czy e należy do MST. Niech u1, u2 będą wierzchołkami połączonymi krawędzią e. Sprawdzimy DFS'em czy z wierzchołka u1 istnieje ścieżka do wierzchołka u2 składająca się z krawędzi o wagach mniejszych od wagi krawędzi e. Jeśli tak, to oznacza, że krawędź e znajduje się w cyklu, w którym jest niezależną krawędzią, korzystając z powyższego twierdzenia, mówimy, że e nie należy do żadnego MST wpp. e nie leży na cyklu lub nie jest najcięższą krawędzią na jakimś cyklu, więc należy do MST.

ZAD. 8 UTÓŻ ALGORYTM, KTÓRY DLA DANYCH LICZB NATURALNYCH a i b, SPRAWDZA, CZY ZACHŁANNA STRATEGIA DLA PROBLEMU WYDAWANIA RESZTY JEST POPRAWNA, GDY ZBIÓR NOMINAŁÓW JEST RÓWNY X = {1, a, b}

Kiedy algorytm zachłanny nie działa poprawnie?

Gdy zwróci monety, których ilość jest większa od innego możliwego rozwiązania.

Zauważmy, że algorytm jest poprawny zawsze gdy a jest wielokrotnością b. (no a chcemy zużyć jak najmniej monet).

Zatem kwota, której szukamy jako kontrprzykład to najmniejsze wielokrotność a,

która jest większa od b. $\longrightarrow k = \lceil \frac{b}{a} \rceil \cdot a$

Najmniejsza wielokrotność $a > b$

$$\sum_{i=0}^{n} a > b \quad \wedge \quad \sum_{i=0}^{n-1} a < b$$

Aby sprawdzić, czy algorytm jest poprawny musimy porównać sumę wydanych monet w dwóch przypadkach:

sum 1 = 1 + k - b

sum 2 = k/a ; liczba monet nominału a

Zatem aby sprawdzić czy algorytm zachłanny wydaje resztę poprawnie należy porównać te dwie sumy:

```
k = ceil(b/a) * a
if( b%a == 0)
    return true

sum1 = 1+k-b
sum2 = k/a

if( sum1 > sum2)
    return false
return true
```

$k < a+b$