Algorytmy Zachłanne.

PROBLEM WYDAWANIA RESZTY + DRZEWA ROZPINAJĄCE

Przykład
Strategia zachłanna dla problemu wydawania reszty może polegać na tym, by w kolejnych krokach do konstruowanego rozwiązania wstawiać monetę o największym nominale nie przekraczającym kwoty pozostałej do wydania.
Można łatwo wykazać, że dla zbioru nominałów $\{1, 2, 5, 10, 20, 50, 100\}$ taka strategia prowadzi zawsze do rozwiązania optymalnego. Natomiast dla zbioru $\{1, 2, 5, 9, 10\}$ czasami daje rozwiązania nieoptymalne: np. dla $R = 14$ znajdzie rozwiązanie $10, 2, 2$, chociaż wystarczają dwie monety.

1.1 Konstrukcja minimalnego drzewa rozpinającego

Rozważamy grafy nieskierowane $G = (V, E; c)$, gdzie V oznacza zbiór wierzchołków, E - zbiór krawędzi, a $c : E \to R_+$ jest funkcją wagową. Wagą podgrafu $G' = (V', E')$ grafu G nazywamy sumę wag krawędzi z E'.

Definicja 1 Drzewem rozpinającym *grafu* $G = (V, E; c)$ *nazywamy dowolne drzewo* $T = (V, E')$, *takie, że* $E' \subseteq E$. *Drzewo rozpinające* T *nazywamy* minimalnym, *jeśli ma minimalną wagę spośród wszystkich drzew rozpinających grafu* G.

Strategia 1: Algorytm Kruskala
Rozpoczynamy od pustego E'. Zbiór C jest początkowo równy E. W kolejnym kroku rozpatrujemy krawędź z C o minimalnej wadze. Dodajemy ją do E', o ile nie powoduje to powstania cyklu.

Strategia 2: Algorytm Prima
Inicjujemy E' wstawiając do niego minimalną krawędź spośród krawędzi incydentnych z wierzchołkiem v (v - wybierany jest arbitrarnie). Podobnie jak poprzednio zbiór C jest początkowo równy E. W kolejnym kroku rozpatrujemy minimalną krawędź z C incydentną z jakąś krawędzią z E' . Dodajemy ją do E', o ile drugi z jej końców nie jest incydentny z E'.

Strategia 3: Algorytm Boruvki
Strategia ta nieco odbiega od ogólnego schematu. Zbiór E' budujemy fazami. W każdej fazie wykonujemy dwa kroki:

- Dla każdego wierzchołka z G znajdujemy najkrótszą incydentną z nim krawędź; krawędzie te dołączamy do zbioru E'.
- Tworzymy nowy graf G'. Wierzchołki w G' (nazwijmy je superwierzchołkami) odpowiadają spójnym składowym w E'. Dwa superwierzchołki S_1 i S_2 łączymy krawędzią wtedy i tylko wtedy, 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.

Uwagi:

- algorytm Boruvki jest szczególnie przystosowany do implementacji na maszynach równoległych;
- algorytm ten działa poprawnie, gdy wszystkie krawędzie mają różne wagi (to jednak zawsze potrafimy zagwarantować).

Dowody poprawności tych algorytmów są podobne. Łatwo zauważyć, że wszystkie algorytmy znajdują drzewa rozpinające. Strategie Kruskala i Boruvki gwarantują bowiem, że w każdym momencie krawędzie z E' tworzą las drzew, a strategia Prima gwarantuje, że krawędzie z E' tworzą drzewo. O ile graf G jest spójny, to po zakończeniu działania algorytmu graf (V, E') także jest spójny (a więc jest drzewem rozpinającym). W przeciwnym razie minimalna krawędź spośród krawędzi o końcach w różnych składowych nie byłaby dołączona przez algorytmy Kruskala i Boruvki do E'; a w przypadku algorytmu Prima taką niedołączoną krawędzią byłaby minimalna krawędź indydentna jednym końcem z E'. W obydwu przypadkach otrzymujemy sprzeczność.

Minimalność wyznaczonych drzew wynika z faktu, że w każdym momencie konstrukcji zbioru E' jest on rozszerzalny do minimalnego drzewa rozpinającego.

Szeregowanie z terminami

PROBLEM:

Dane: ciąg $d_1, \ldots, d_n$ liczb naturalnych oraz
ciąg $g_1, \ldots, g_n$ dodatnich liczb rzeczywistych;
(Interpretacja: d_i-termin dla i-tego zadania; g_i-zysk za i-te zadanie).

Zadanie: znaleźć wykonalny podzbiór (patrz definicja poniżej) zadań $S \subseteq \{1, \ldots, n\}$ maksymalizujący sumę $\sum_{i \in S} g_i$.

Strategia zachłanna: Startując od zbioru pustego konstruujemy wykonalny zbiór zadań, na każdym kroku dodając do niego zadanie o największym g_i spośród zadań jeszcze nie rozważonych (pod warunkiem, że zbiór pozostaje wykonalny).

DOWÓD POPRAWNOŚCI (SZKIC): Załóżmy, że nasz algorytm wybrał zbiór zadań I, podczas gdy istnieje zbiór optymalny $J \neq I$. Pokażemy, że dla obydwu zbiorów zysk za wykonanie zadań jest taki sam. Niech π_I, π_J-wykonalne ciągi zadań z I i J. Dowód przebiega w dwóch etapach:

1. Wykonując przestawienia otrzymujemy ciągi π'_I oraz π'_J takie, że wszystkie wspólne zadania (tj. zadania z $I \cap J$) wykonują się w tym samym czasie.

2. Pokazujemy, że w pozostałych jednostkach czasowych π'_I oraz π'_J mają zaplanowane wykonanie zadań o tym samym zysku.

Ad.1. Niech $a \in C$ będzie zadaniem umieszczonym na różnych pozycjach w π_I oraz π_J. Niech będą to pozycje odpowiednio i oraz j. Bez zmniejszenia ogólności możemy założyć, że $i < j$. Zadanie a w ciągu π_I możemy zamienić miejscami z zadaniem znajdującym się na pozycji j, nazwijmy to zadanie b. Otrzymamy ciąg wykonalny, gdyż:

- $d_a \geq j$, ponieważ a znajduje się na pozycji j w π_J,
- $d_b > i$, ponieważ b znajduje się na pozycji j w π_I.

Liczba zadań z $I \cap J$ rozmieszczonych na różnych pozycjach w π_J i nowym ciągu π_I zmniejszyła się o co najmniej 1. Iterując postępowanie otrzymujemy tezę.

Ad.2. Rozważmy dowolną pozycję i, na której różnią się π'_I oraz π'_J.
Zauważamy, że zarówno π'_I jak i π'_J mają na tej pozycji umieszczone jakieś zadanie, nazwijmy je a i b odpowiednio. Gdyby bowiem π'_J miało tę pozycję wolną, to moglibyśmy umieścić na niej a otrzymując ciąg wykonalny dla $J \cup \{a\}$, co przeczy optymalności J. Z drugiej strony, gdyby π'_I miało tę pozycję wolną to algorytm zachłanny umieściłby b w rozwiązaniu.

Wystarczy teraz pokazać, że $g_a = g_b$:

- gdyby $g_a < g_b$, to algorytm zachłanny rozpatrywałby wcześniej b niż a; ponieważ zbiór $I \setminus \{a\} \cup \{b\}$ jest wykonalny (a więc także i ten jego podzbiór, który był skonstruowany w momencie rozpatrywania b), więc b zostałoby dołączone do rozwiązania.
- gdyby $g_a > g_b$, to $J \setminus \{b\} \cup \{a\}$ dawałby większy zysk niż J?!

Szeregowanie zadań

PROBLEM:

Dane: ciąg $t_1, \ldots, t_n$ dodatnich liczb rzeczywistych;
(interpretacja: t_j - czas obsługi j-tego zadania w systemie).

Zadanie: ustawić zadania w kolejności minimalizującej wartość:
$T = \sum_{i=1}^{n}$ (*czas przebywania i-tego zadania w systemie*)

Strategia zachłanna: Zadania ustawiamy w kolejności rosnących czasów obsługi.

DOWÓD POPRAWNOŚCI: Zauważamy , że jeśli zadania realizowane są w kolejności zadanej permutacją $\pi = (i_1, i_2, \ldots, i_n)$ liczb $(1, 2, \ldots, n)$, to związany z tym koszt wynosi:

$$T(\pi) = t_{i_1} + (t_{i_1} + t_{i_2}) + \ldots + (t_{i_1} + \ldots + t_{i_n}) = \sum_{k=1}^{n} (n - k + 1) t_{i_k}$$

Załóżmy, że π jest optymalnym porządkiem oraz że istnieją x, y takie, że $x < y$ oraz $t_{i_x} > t_{i_y}$. Zamieniając i_x oraz i_y miejscami otrzymujemy nowy porządek π' o koszcie:

$$T(\pi') = (n - x + 1) t_{i_y} + (n - y + 1) t_{i_x} + \sum_{k=1, k \neq x, y}^{n} (n - k + 1) t_{i_k}.$$

Nowy porządek jest lepszy, ponieważ

$$T(\pi) - T(\pi') = (n - x + 1)(t_{i_x} - t_{i_y}) + (n - y + 1)(t_{i_y} - t_{i_x}) = (y - x)(t_{i_x} - t_{i_y}) > 0$$

co jest sprzeczne z założeniem optymalności π □

Pozostaje problem: jak można ustalać, czy dany zbiór J złożony z k zadań jest wykonalny (oczywiście sprawdzanie wszystkich $k!$ ciągów nie jest najlepszym pomysłem). Poniższy prosty lemat mówi, że wystarczy sprawdzać wykonalność tylko jednego ciągu.

Lemat 1 *Niech J będzie zbiorem k zadań i niech $\sigma = (s_1, s_2 \ldots s_k)$ będzie permutacją tych zadań taką, że $d_{s_1} \leq d_{s_2} \leq \ldots \leq d_{s_k}$. Wówczas J jest wykonalny iff σ jest wykonalny.*

Zakładamy, że zadania ułożone są według malejących zysków, tj. $g_1 \geq g_2 \geq \ldots \geq g_n$.

Prosta implementacja polega na pamiętaniu skonstruowanego fragmentu wykonywalnego ciągu zadań w początkowym fragmencie tablicy. Zadania umieszczane są tam według rosnących wartości terminów w sposób podobny jak w procedurze sortowania przez wstawianie.

Fakt 1 *Taka implementacja działa w czasie $\Omega(n^2)$.*

Lemat 2 *Zbiór zadań J jest wykonalny iff następująca procedura ustawia wszystkie zadania z J w ciąg wykonalny:*

$\forall_{i \in J}$ ustaw i-te zadanie na pozycji t, gdzie t jest największą liczbą całkowitą, taką że $0 < t \leq \min(n, d_i)$ i na pozycji t nie ustawiono jeszcze żadnego zadania.

Efektywna realizacja tej procedury używa struktur do pamiętania zbiorów rozłącznych. Poznamy je, gdy będziemy omawiać problem UNION-FIND. Uzyskany czas działania algorytmu będzie bliski liniowego.

Pokrycie zbioru problem należy do klasy problemów $\mathcal{NP}$-trudnych

PROBLEM:

Dane: rodzina $S = \{S_1, S_2, \ldots, S_k\}$ podzbiorów n-elementowego uniwersum U
funkcja kosztu $c : S \to \mathcal{R}_+$

Zadanie: Znaleźć najtańszą podrodzinę S pokrywającą U, tj.
znaleźć $S' \subseteq S$ taką, że $\bigcup_{X \in S'} X = U$ i żadna inna podrodzina nie ma kosztu mniejszego od $Koszt(S')$, gdzie koszt podrodziny Z jest zdefiniowany jako

$$Koszt(Z) = \sum_{X \in Z} c(X).$$

- Strategia 3: wybierz podzbiór, który najtaniej pokrywa elementy.

(b) do rozwiązania wybieramy ten z podzbiorów, dla którego wartość *cne* jest minimalna.

Twierdzenie 1 *Opisana powyżej strategia zachłanna znajduje rozwiązanie o koszcie nie większym niż* $(\log n) \cdot OPT$, *gdzie* OPT *jest kosztem rozwiązania optymalnego.*

Niech $e_1, e_2, \ldots, e_n$ będzie ciągiem elementów uniwersum wypisanych w kolejności pokrywania ich przez algorytm. Jeśli pokrywanym elementom przypiszemy cenę, równą wartości *cne* podzbioru, który je pokrywa, to koszt pokrycia możemy wyrazić jako sumę cen elementów. Zauważamy następujący fakt, z którego wynika dowód twierdzenia.

Fakt 2 *Dla każdego* $i = 1, \ldots, n$

$$cena(e_i) \leq \frac{OPT}{n-i+1} \qquad \sum_{j=1}^{n} \frac{\mathrm{OPT}}{n-j+1} = \mathrm{OPT} \cdot \sum_{j=1}^{n} \frac{1}{j} = \mathrm{OPT} \cdot O(\log n),$$

Wynika on z następujących spostrzeżeń:

- W momencie pokrywania elementu e_i niepokrytych było co najmniej $n - i + 1$ elementów uniwersum.
- Te niepokryte elementy można by pokryć podzbiorami, które należą do rozwiązania optymalnego. Sumaryczny koszt tych podzbiorów jest nie większy od OPT. Koszt ten zostałby rozłożony na ceny tych elementów, więc istnieje element, którego cena jest nie większa od średniej, tj. od $\frac{OPT}{n-i+1}$.

Pokażemy jednak, że trzecia strategia gwarantuje znalezienie rozwiązania, które jest niezbyt odleg
od rozwiązania optymalnego.

Najpierw sprecyzujemy tę strategię. W kolejnych krokach:

(a) dla każdego podzbioru S_i określamy cenę za pokrywany element (w skrócie *cne*):

$$cne(S_i) = \frac{c(S_i)}{|S_i \setminus C|}$$

gdzie C oznacza zbiór dotychczas pokrytych elementów.

ZADANIA Z EGZAMINÓW

Przedstaw strategię zachłanną algorytmu aproksymacyjnego dla problemu Set Cover o współczynniku aproksymacji H_n.

Problem: Wybrać najmniejszy taki podzbiór z rodziny zbiorów, aby wszystkie pojedyncze elementy z całej rodziny znalazły sie w tym podzbiorze. Algorytm aproksymacyjny: W każdym kroku odrzucamy zbiór, który nie pokrywa jak nawiekszej części elementów.

```
U – uniwersum do pokrycia
R – wejsciowa rodzina zbiorow
W – wynikowa rodzina zbiorow na poczatu pusta

while U jest niepusty
    wybierz takie Z z R ze Z przekroj U jest maksymalne
    dodaj Z do rodziny wynikowej W
    U \= Z
    usun Z z R
```

Przedstaw ideę algorytmu Boruvki (Sollina).

▼ Rozwiązanie

`G` - nasz graf

`G'` - nasza odpowiedź, początkowo tylko wierzchołki z G i żadnych krawędzi

Wykonujemy kroki:

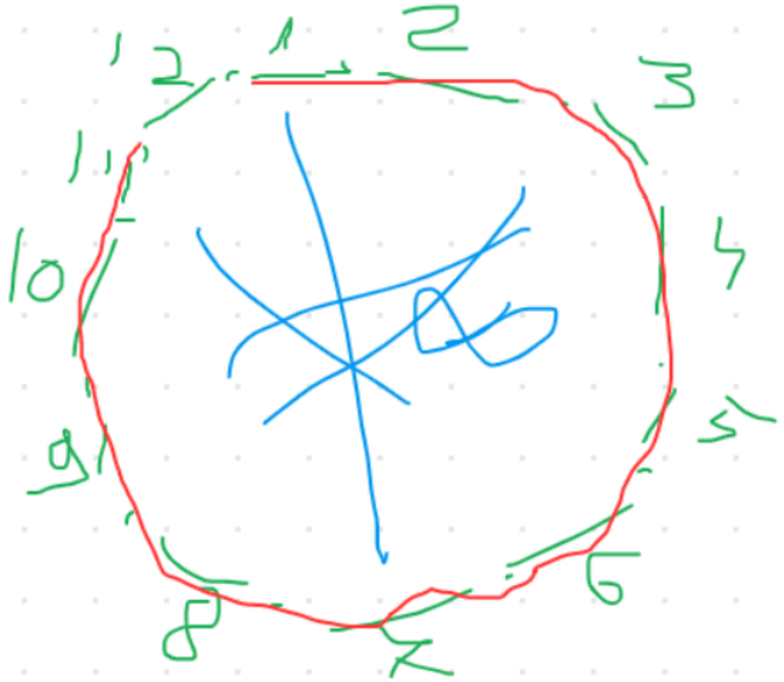
1. Dla każdego wierzchołka w `G` znajdź najkrótszą incydentną krawędź i dodaj ją do `G'`
2. Utwórz nowy graf `G'`, w którym wierzchołkami są spójne składowe w starym `G'`

Iterujemy do póki nie zostanie jednen wierzchołek. Wszystkie dodane krawędzie do `G'` utworzą odpowiedź.

Podaj przykład grafu pełnego o n wierzchołkach, dla którego Borovka znajduje MST w jednej fazie.

▼ Rozwiązanie

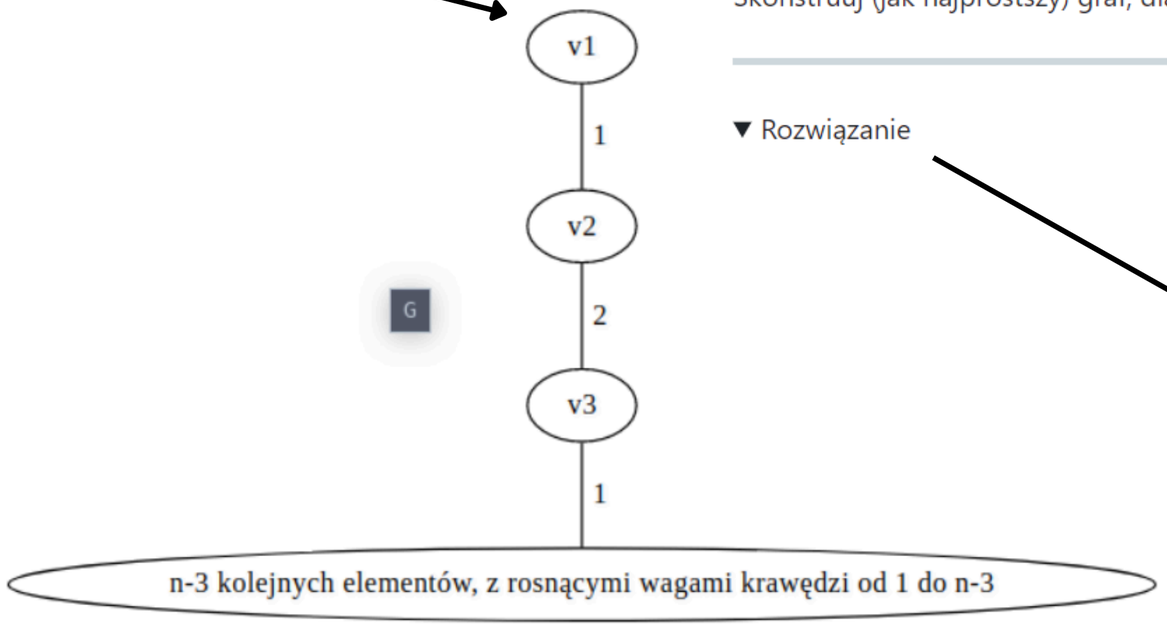
Graf w którym istnieje cykl rosnących wag krawędzi oraz krawędzie poza tym cyklem mają wagi $+\infty$



ZADANIA Z EGZAMINÓW CD

Pokaż, że istnieją n wierzchołkowe grafy(n jest dowolnie duże), dla których algorytm Boruvki znajduje minimalne drzewo spinające wykonując dokładnie dwie fazy.

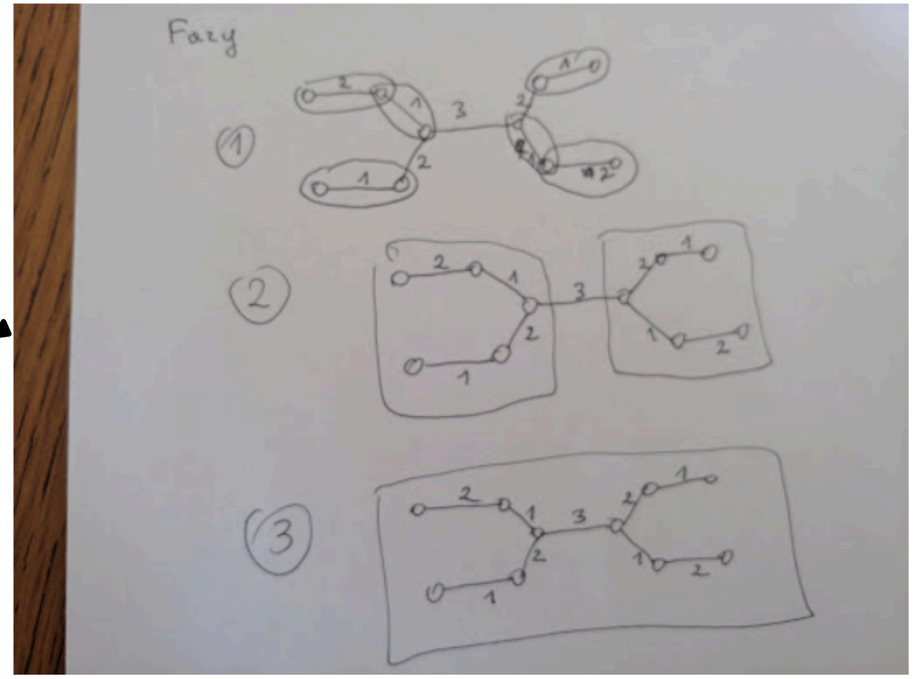
▼ Rozwiązanie



Algorytm Boruvki-Sollina wykonuje pewną liczbę faz. W każdej fazie dorzuca do konstruowanego drzewa pewne krawędzie i tworzy nowy graf, który będzie rozważany w kolejnej fazie.

Skonstruuj (jak najprostszy) graf, dla którego ten algorytm wykona dokładnie 3 fazy lub pokaż,że taki graf nie istnieje.

▼ Rozwiązanie



ZADANIA Z ĆWICZEŃ

1. (0pkt) Przeczytaj notatkę do wykładu o algorytmach zachłannych.

2. (1pkt) Danych jest n odcinków $I_j = \langle p_j, k_j \rangle$, leżących na osi OX, $j = 1, \ldots, n$. Ułóż algorytm znajdujący zbiór $S \subseteq \{I_1, \ldots, I_n\}$, nieprzecinających się odcinków, o największej mocy.

3. (1pkt) Rozważ następującą wersję problemu wydawania reszty: dla danych liczb naturalnych a, b ($a \leq b$) chcemy przedstawić ułamek $\frac{a}{b}$ 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)?

4. (1,5pkt) Rozważ wersję problemu wydawania reszty, w którym i-ty nominał ma wartość równą i-tej liczbie Fibonacciego ($i = 1, 2, \ldots$). Ułóż algorytm, który, wydając resztę, używa co najwyżej jednej monety każdego nominału. Czy taki algorytm jest w stanie wydać każdą kwotę? Czy liczba monet wydana przez Twój algorytm jest zawsze optymalna (tj. minimalna)?

5. (2pkt) Masz początkowo do dyspozycji m monet o nominale 1 i nieskończoną liczbę monet o nominale 100. W kolejnych n dniach masz zrobić zakupy w ulubionym sklepie (w i-tym dniu zakup o wartości c_i). Jeśli w i-tym dniu nie odliczysz dokładnej kwoty (tj. c_i), kasa wyda Ci dokładną wartość reszty, używając minimalnej ilości monet (kasa także używa jedynie monet o nominałach 1 i 100), a Twoja "atrakcyjność kliencka" zostanie zmniejszona o wartość $x \cdot w_i$, gdzie x jest liczbą monet wydanych przez kasę a w_i jest współczynnikiem używanych przez sklep w i-tym dniu.

   Ułóż algorytm obliczający o ile co najmniej zmniejszy się Twoja atrakcyjność kliencka po dokonaniu wszystkich zakupów. Możesz przyjąć, że c_i oraz w_i są liczbami naturalnymi nie większymi od n.

6. (1,5pkt) 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 pokolorowanych wierzchołków.

7. (2pkt) Ułóż algorytm, który dla danego spójnego grafu G oraz krawędzi e sprawdza w czasie $O(n+m)$, czy krawędź e należy do jakiegoś minimalnego drzewa spinającego grafu G. Możesz założyć, że wszystkie wagi krawędzi są różne.

8. (2pkt) Niech $T = (V, E)$ będzie drzewem a $P(u, v)$ niech oznacza ścieżkę w T (rozumianą jako zbiór krawędzi) łączącą wierzchołki u i v.

   Ułóż algorytm, który dla drzewa T znajduje trzy wierzchołki a, b, c, dla których zbiór $\{e \in E : e \in P(a,b) \cup P(a,c) \cup P(b,c)\}$ jest maksymalnie duży.

9. (2pkt) Operacja *swap*(i, j) na permutacji powoduje przestawienie elementów znajdujących się na pozycjach i oraz j. Koszt takiej operacji określamy na $|i - j|$. Kosztem ciągu operacji *swap* jest suma kosztów poszczególnych operacji.

   Ułóż algorytm, który dla danych π oraz σ - permutacji liczb $\{1, 2, \ldots, n\}$, znajdzie ciąg operacji *swap* o najmniejszym koszcie, który przekształca permutację π w permutację σ.

10. (1pkt) Na wykładzie przedstawiono zachłanny algorytm dla problemu *Pokrycia zbioru*, znajdujący rozwiązania, które są co najwyżej $\log n$ razy gorsze od rozwiązania optymalnego.

    Pokaż, że istnieją dane, dla których rozwiązania znajdowane przez ten algorytm są blisko $\log n$ gorsze od rozwiązań optymalnych.