Drzewa i słowniki e02 - rozwiązania

Zadanie 1.
Żeby zrobić to wzorcowo, musimy dla każdego elementu w tablicy znaleźć, gdzie powinien być jego hash (O(1) wywołanie hash(x)) i sprawdzić, czy między nim a jego hashem jest -1. Tworzymy sobie dodatkową tablicę P, gdzie P[i] oznacza indeks ostatniej napotkanej -1, na lewo od i. Tablicę P uzupełniamy tak, że iterujemy po T i jak nie ma -1 pod T[i], to wpisujemy indeks ostatniej zapamiętanej -1, a jak T[i] jest -1, to aktualizujemy indeks ostatniej napotkanej -1. Dla liczb w T, poprzedzających pierwszą -1, możemy w P[i] wpisać jakieś -1 albo None. I teraz jeżeli i < hash(T[i]%N) to sprawdzamy, czy P[hash(i)%N] > i jak tak to jest źle, bo jest -1 między pozycją i hashem. No i przypadek symetryczny dla i > hash(T[i]%N).

Zadanie 2.
Dla każdej pary punktów obliczamy parametry prostej na której leżą. Wstawiamy taką parę (a, b) do słownika i inkrementujemy tam licznik.

Zadanie 3.
Tworzymy tablicę hashującą, gdzie kluczami są referencje na stare węzły, a wartościami są referencje na nowe węzły. Teraz iterujemy po listach sąsiadów starych nodów i do listy nowego noda pod kluczem starego wstawiamy referencje na nody pod kluczami sąsiadów starego noda.

Zadanie 4.
Robimy słownik z tablicy, dla każdego elementu o wartości X szukamy w tablicy, czy jest tam S - X; jeżeli tak, to mamy nasz poszukiwany drugi element.

Zadanie 5.
Wrzucamy wszystkie słowa z W do hashmapy i mapujemy te słowa na ich częstotliwości. Oznaczamy przez m długość tablicy W, a przez M długość jednego słowa z W. Dla każdego indeksu i ze słowa S, m razy wycinamy substring od długości M, i wrzucamy do tymczasowej hashmapy ten substring i sprawdzamy, czy ani razu częstotliwość w tymczasowej hashmapie nie przekroczyła częstotliwości z tej poprzedniej; jeżeli nie, to dodajemy i do listy wynikowej.

Zadanie 6.
Przepisujemy obie tablice do tablic haszujących o tym samym rozmiarze, gdzie stosujemy łańcuchowe rozwiązywanie konfliktów. Sortujemy każdą listę i porównujemy listy pod tymi samymi indeksami. Mało kolizji -> dużo krótkich list -> tak O(n) jak O(n) jest bucket sort. Dużo kolizji -> mało długich list -> zwykłe sortowanie listy w O(nlog(n)).

Zadanie 7.
Dla każdego słowa, dla każdego prefixu tego słowa sprawdzamy, czy doklejenie rewersu tego prefixu spowoduje utworzenie palindromu. Jeżeli tak, to szukamy tego rewersu w hashmapie, do której wcześniej wrzuciliśmy każde słowo z tablicy. Wartościami pod kluczami są indeksy tych słów w tablicy słów.

Zadanie 8.
d - wymiarowa taka struktura, to drzewo, gdzie kluczami są pierwsze elementy wektora, a wartościami d - 1 wymiarowe struktury.