Zadania przed kolokwium III - rozwiązania

Zadanie 1
Najpierw zliczamy wystąpienia każdej wartości za pomocą słownika w czasie O(n). Potem korzystamy z faktu, że wybór par wartości (P, Q) to po prostu wybór k po 2 elementów, gdzie k to rozmiar słownika.

Zadanie 2
Budujemy drzewo Trie, przy wyszukiwaniu zamieniamy liczbę na postać binarną i przechodzimy po drzewie.
W Pythonie można uzyskać postać binarną przez `str(bin(my_integer))[2:]`. Wycinamy 2 pierwsze znaki, bo to "0b", sygnalizujące, że string koduje liczbę binarną, nam tu jest to niepotrzebne.

Zadanie 3
Przechodzimy po hashtable'u, dla każdego elementu różnego od -1 (czyli coś tam jest) obliczamy jego indeks na podstawie hasza. Dzięki temu jesteśmy w stanie policzyć w czasie O(1) odległość między jego "idealnym" położeniem a tym "faktycznym" (powstałym przez kolizje). Jest to zasadniczo to samo, co przehaszowanie każdego elementu po kolei i przejście aż do faktycznego położenia, ale to byłoby $O(n^2)$, a przejście liniowo po hashtable'u w ten sposób to O(n).

Zadanie 4
Budujemy tablicę 2D intów i uzupełniamy w $O(n^2)$, ale to nas nie interesuje - ważne jest tylko, że sprawdzenie potem to O(1).

Zadanie 5
Z tablicy sentences tworzymy drzewo Trie w taki sposób, że w każdym węźle jest lista indeksów w tablicy sentences, pod którymi są słowa mające jak prefix tę ścieżkę w Trie. Wyszukiwanie działa tak, że idziemy wzdłuż ścieżki w Trie i wrzucamy do kopca pary (priorytet słowa, słowo) i teraz wyciągamy z kopca k pierwszych elementów mających jako prefix aktualny bufor znaków od użytkownika. Po przyjęciu znaku #, czyścimy bufor.

Zadanie 6
Dla każdego słowa generujemy jego wszystkie prefixy, sufixy i łączymy je w pary za pomocą konkatenacji wsadzając między nie jakiś znak np. #.
Przykładowo, dla słowa app: app#, app#app, app#pp, app#p itp. Takie stringi wsadzamy do drzewa Trie, i pamiętamy jaki był największy priorytet dla takiego słowa. Teraz jak wyszukujemy to po prostu łączymy prefix#suffix i szukamy w Trie.

Zadanie 7
W tablicy nodes[] trzymanej w każdym węźle SkipListy pamiętamy w każdym nodes[i] ile wartości przeskoczymy, idąc dalej o 1 node na i-tym poziomie.
https://stackoverflow.com/questions/50122444/find-k-th-element-in-skiplist-explanation-needed