**1. Struktury danych**

* Struktury dla uporządkowanego multizbioru
  + Zbior z elementami niemalejącymi, oraz duplikatami
  + W python można stosować Counter() z collections.
  + Normalnie stosuje się chyba drzewa bst
  + Dodawanie, usuwanie, liczenie, wypisanie O(n)
* Drzewa licznikowe, potęgowe (Fenwicka)
  + Do liczenia sumy prefixowej, choć można przedzialy
  + Tworzona tablica size+1 (pierwszy element tablicy tree ma wartość 0 bądź żadna)
  + Głównie wszystko opiera się wokół indeksów i najmłodszego bitu (index & -index)
  + Dodawanie do drzewa: dopóki index <= size to zwiększamy elementy o indexie index o podaną wartość i zwiększamy potem index o index & -index
  + Sprawdzanie sumy: dopóki index > 0 to dodajemy do result wartość tree[index] a następnie zmniejszamy index o index & -index
* Drzewa przedziałowe (punkt-przedział, przedział-punkt, przedział-przedział)
  + Do liczenia sum przedziałów, bądź min/max
  + Tworzenie tablicy o wielkości n\*2
  + Budowanie na zasadzie takiej za najpierw po prawej stronie drzewa dodajemy elementy z tablicy, a po lewej następne elementy idąc do 0 maja wartość sumy elementow o idenksach i\*2 i i\*2+1
  + Update: Najpierw podmieniamy wartość w drzewie po prawej a następnie w petli gdzie index>0 zmniejszamy index dwukrotnie i sumujemy elementy o ind i\*2 i i\*2+1
  + Query: na start zmienne left i right wskazujące na elementy po prawej stronie. Następnie dopki left<right, to:
    - Jeżeli left%2==1 to dodajemy do result tree[left] i zwiększamy left o 1
    - Jeżeli right%2==1 to zmniejszamy right o 1 i dodajemy do result tree[right]
    - Zmniejszamy left i right o ich dwokrotnosc //2
* Kolejka minimów i dla dowolnej operacji łącznej
  + Stosowana przy wielkich dynamicznych danych do znajdywania min/max o zlozonosci o(1) zamiast o(n)
  + Stosuje się 2 kolejki: do danych i do minimów/maximow
  + Dodanie: dodajemy do kolejki danych element a następnie usuwamy z kolejki minimów ostatni element dopóki ostatni element kolejki będzie mniejsza od podanego elementu albo kolejka minimów będzie pusta (min\_q and min\_q[-1] > value) a na końcu dodajemy element
  + Usuniecie: najpierw sprwadzamy czy pierwszy element kolejki z danymi jest taki sam jak pierwszy element min kolejki. Jak tak to usuwamy z lewej strony z min kolejki. Następnie usuwamy z lewej strony z kolejki z danymi.
  + Get: otrzymujemy min element o zlozonosci O(1)

**2. Przepływy w sieciach**

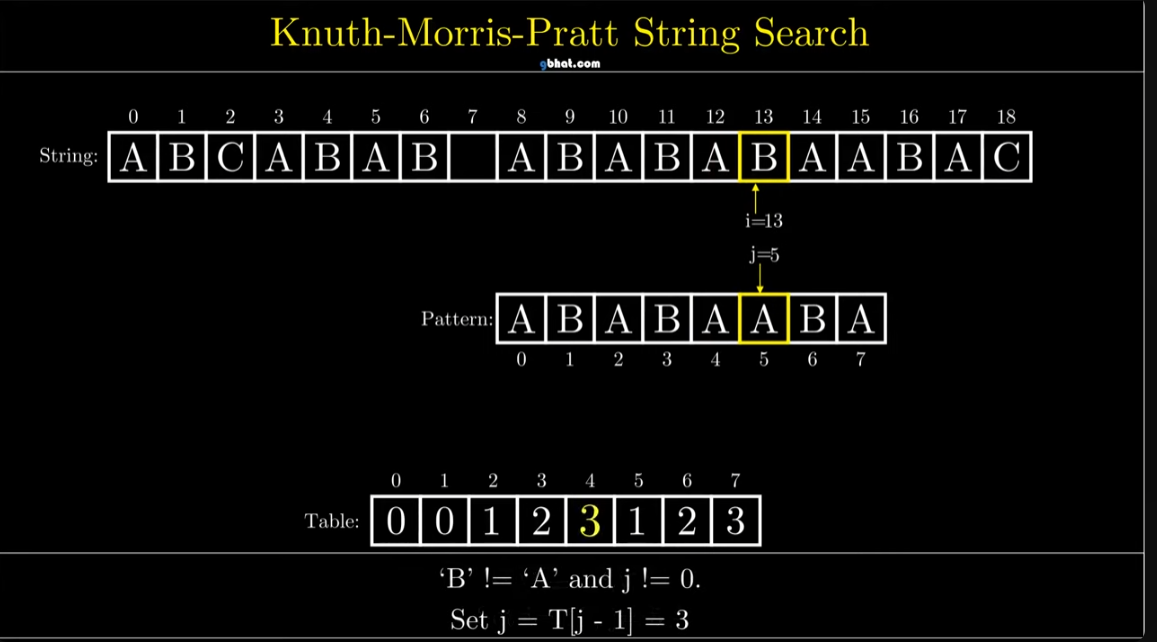
* Algorytm Forda-Fulkersona
* Algorytm Edmondsa-Karpa

**3. Algorytmy geometryczne**

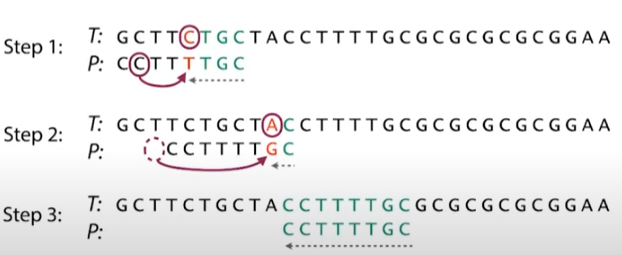
* Położenie punktu względem odcinka
  + Wzór na orientacje pkt względem odcinka: (x2-x1)(y3-y1) – (y2-y1)(x3-x1)
* Przynależność punktu do wielokąta wypukłego, dowolnego
  + Sprawdzamy to poprzez wystrzelenie od punktu w prawo linii a następnie sprawdzenie ile razy ta linia przetnie boki wielokątu. W przypadku ilości nieparzystej ten punkt znajduje się w wielokącie, w przeciwnym przypadku nie.
  + Zalozenie to posiadanie tablicy punktów tworzących boki w odpowiedniej kolejności.
  + Dla każdego boku sprawdzamy, czy Py jest pomiędzy Ay i By oraz czy Px <= x. Jak tak to sprawdzamy jeszcze czy Px <= xinters (x przecięcia P z AB)
* Otoczka wypukła - algorytm Jarvisa
  + Algorytm do wyznaczania otoczki wypuklej na podstawie podanych punktów
  + Jego zlozonosc to O(n\*h) gdzie n to ilość pkt a h to ilość pkt na otoczce
  + Wykorzystujemy wzor na orientacje pkt względem odcinka.
  + Na początku wyznaczamy pkt skrajnie po lewej stronie. W przypadku takich samych to na samym dole.
  + Nastepnie mamy petle która z zalozenia przechodzi przez każdy punkt na otoczce. W tej petli mamy również kolejna petle, która szuka **kandydata** na kolejny punkt na otoczce, przechodząc przez wszystkie punkty. Sprawdzamy to poprzez znalezienie punktu o największym kacie.
  + Jeżeli znaleziony przez nas kolejny punkt na otoczke jest rowny skrajnie lewemu punktowi to kończymy petle.
* Otoczka wypukła - sortowanie biegunowe i algorytm Grahama
  + Tez do otoczki
  + Zlozonosc n\*log(n)
  + Glowna idea to wykorzystanie kątów polarnych punktów względem skrajnie na dole pkt po lewej stronie.
  + Na początku sortujemy tablice punktów względem ich kata polarnego, jeżeli takie same pkt to uwzględniamy odleglosc.
  + Nastepnie tworzymy stos do którego dodajemy na początku 3 pierwsze elementy tablicy posortowanej.
  + W petli przechodzącej przez wszystkie pkt za wyjątkiem 3 pierwszych, w petli WHILE dopóki rozpatrywany punkt względem odcinka tworzonego przez 2 ostatnie pkt na stosie jest po prawej stronie lub na odcinku, to usuwamy ostatni element na stosie. Nastepnie dodajemy ten punkt do otoczki.
* Metoda zamiatania - przecinanie odcinków
  + Algorytm do sprawdzania przecięcia odcinków na danej plaszczyznie.
  + Ideą jest „zamiatanie” w taki sposób, ze przechodzimy przez płaszczyznę od lewej do prawej strony, wirtualną pionową linią, która to przy każdym rozpoczęciu bądź zakończeniu odcinka sprawdza aktualne zdarzenia takie jak dodanie nowego odcinka, usunięcię, bądź szuka przecięć odcinków wykorzystując funkcje matematyczne.

**4. Algorytmy tekstowe**

* Problem wyszukiwania wzorca, algorytm naiwny
* Słownik podsłów bazowych
* Haszowanie i algorytm Rabina-Karpa
* Prefiksy, sufiksy i algorytm KMP



* Algorytm Boyera-Moore'a



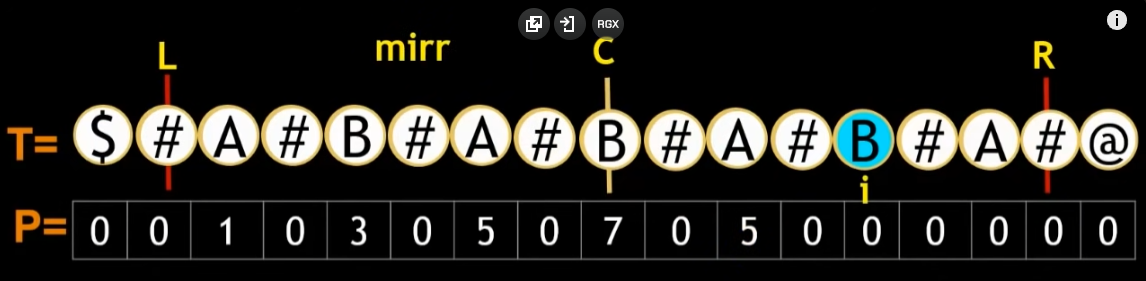
* Algorytm Aho-Corasick wyszukiwania wzorca w tekście

SKIP

* Wyszukiwanie wzorca z wykorzystaniem automatów skończonych

SKIP

* Algorytm Huffmana
* Drzewa palindromów, podpalindromy, algorytm Manachera



**5. Algorytmy aproksymacyjne**

* Pokrycie wierzchołkowe
  + Problem: Dla G(V,E) znaleźć minimalny zbiór C⊂V, taki że każda krawędź e w E ma co najmniej jedno polaczenie w C
  + Najpierw musimy utworzyć tablice krawędzi, np. jako krotek. Wazne aby nie było takich samych krawędzi czyli np. (2,3), (3,2). Następnie w pętli dopóki ta tablica nie jest pusta, bierzemy dowolną krawędź z tego zbioru, przypisujemy pkt tej krawędzi do zmiennych u i v ,usuwamy z tego zbioru ta krawedz, a następnie ze zbioru usuwamy wszystkie krawędzie, których co najmniej jeden wierzchołek zawiera u bądź v.
  + Te rozwiązanie ma wspolczynnik aproksymacyjny 2
* Pokrycie zbioru:
  + Dane: U, S zawierający podzbiory zbioru U
  + Problem: Znaleźć minimalną ilość podzbiorów ze zbioru S których suma będzie rowna U
  + W petli do momentu, kiedy U nie jest puste to szukamy podzbioru ze zbioru S, który pokrywa najwieksza ilość elementow ze zbioru U. kiedy znajdziemy, to ze zbioru U usuwamy elementy z podzbioru a następnie usuwamy ten podzbior ze zbioru S.
  + Wspolczynnik to ln(n)
* Problem sumy podzbioru:
  + Problem: Znaleźć podzbiór liczb ze zbioru S, których suma będzie równa T.
  + W tym algorytmie nie będziemy w stanie zawsze okreslic tej sumy, ale będzie to zblizona wartość do T. Najpierw sortujemy nasz zbiór malejąco. A następnie do current\_sum=0 dodajemy następne elementy z tablicy do momentu kiedy current\_sum<=T
  + Zlozonosc to O(nlogn)
  + Wspolczynnik aproksymacyjny wynosi 2