AKWB 4

Co, jak i po chuj?

Wstęp:

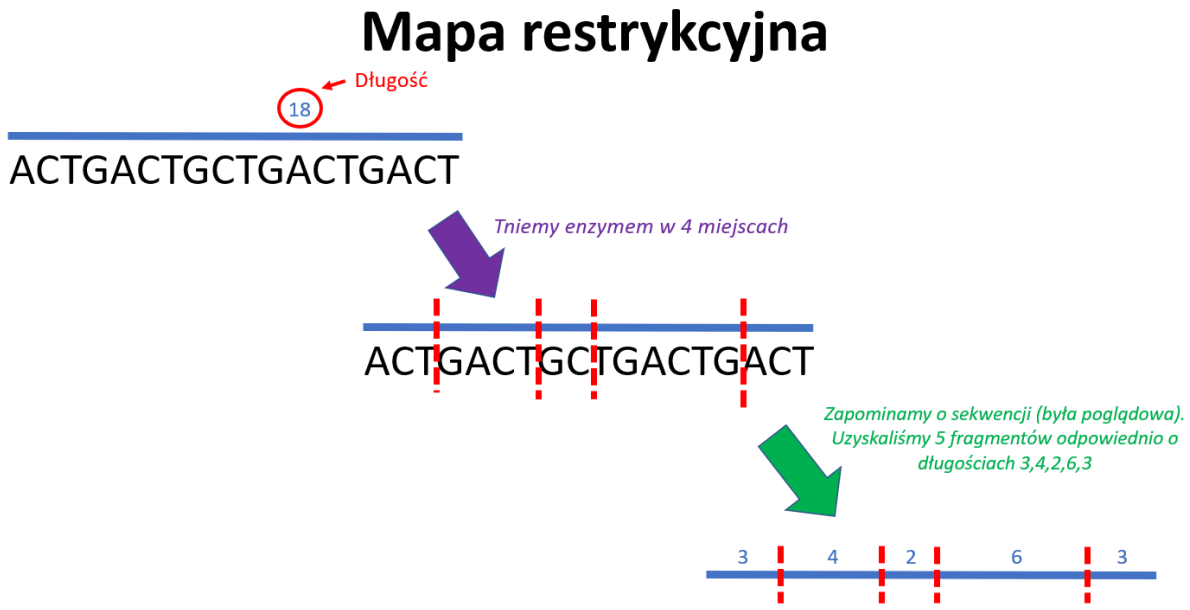
Należy zaimplementować algorytm dokładny (przeszukiwanie z nawrotami, o złożoności wykładniczej) realizujący następujące zadania:

- *Wczytanie z pliku instancji problemu mapowania metodą częściowego trawienia (multizbioru A)*
- *Skonstruowanie mapy restrykcyjnej z podanym multizbiorem (wystarczy jedna z możliwych map)*
- *Wypisanie rezultatu na wyjściu w sposób czytelny dla użytkownika. W razie braku rozwiązania (gdyby instancja zawierała błędy) użytkownik powinien zostać poinformowany odpowiednim komunikatem*

Bla bla bla. Ten plik służy do wyjaśnienia teoretycznego wszystkiego tego co powyżej. Nie jest on instrukcją, a bardziej wstępem do pracy, może jakimiś poradami. To co powyżej zostanie rozbite na czynniki pierwsze i wyjaśnione.

Trochę teorii:

Chcąc pociąć jakieś DNA, używamy enzymów restrykcyjnych. Z reguły takie jedno cięcie w jednym miejscu jest uwarunkowane czasem jaki damy na trawienie. A więc mając jeden enzym ale wiele takich „momentów" czasowych, uzyskamy DNA pocięte w różnych miejscach. A pocięte w różnych miejscach = ileś fragmentów. Poniżej jak to wygląda rysunkowo

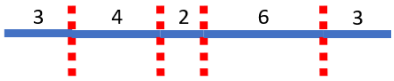


Powyżej można zobaczyć mapę restrykcyjną (*spoiler → Nasz output*). Ale czym jest zatem multizbiór? Ano najprościej, **są to wszystkie „kombinacje" długości, jakie jesteśmy w stanie uzyskać z otrzymanych fragmentów.** Liczy się to od lewej do prawej (wraz z odległością która została do prawej strony). Poniżej przykład który nieco to rozjaśni.

Multizbiór (wstęp)

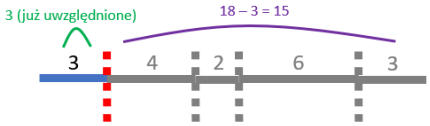
Multizbiór = {3,4,2,6,3,18}

Na start możemy już włożyć wszystkie długości z mapy, oraz całą długość sekwencji (suma elementów mapy)



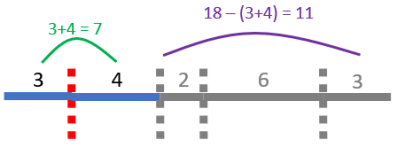
Multizbiór = {3,4,2,6,3,18,15}

Samej „3" już nie uwzględniamy, bo włożyliśmy to razem z innymi fragmentami mapy. Do końca sekwencji zostało nam „15"



Multizbiór = {3,4,2,6,3,18,15,7,11}

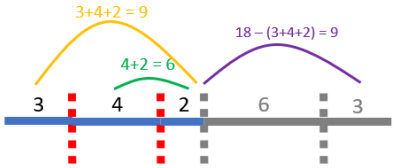
Potem zaczynamy sumowanie. Od lewej możemy zbudować fragment z „3" i „4", oraz do końca sekwencji zostaje nam odległość „11"



Multizbiór = {3,4,2,6,3,18,15,7,11,6,9,9}

Kontynuacja

- *Od lewej do fragmentu tworzymy długość „9"*
- *Sąsiedzi tworzą fragment „6"*
- *Do końca sekwencji zostaje nam „9"*



Okej a dlaczego nie możemy zsumować sobie fragmentu „3" + „2". Ponieważ wtedy Sekwencja nie byłaby ciągła. Widzimy, że jeślibyśmy to zrobili, to powstaje nam dziwna przerwa pośrodku, a to już jest nielegalne. Poniżej jak wygląda taki cały przykładowy workflow.

Multizbiór

Stan multizbioru	Obliczenia	Analiza mapy
{...}	**POCZĄTEK (pusty multizbiór)**	(3,4,2,6,3)
{3,4,2,6,3,18}	Dokładamy wszystkie elementy, i całą długość sekwencji	(3,4,2,6,3)
{3,4,2,6,3,18,**15**}	„3" już uwzględnione (bo był to pierwszy element mapy) 18 – 3 = 15	(**3**,4,2,6,3)
{3,4,2,6,3,18,15,**7,11**}	3+4 = 7 18 – 7 = 11	(**3,4**,2,6,3)
{3,4,2,6,3,18,15,7,11,**6,9,9**}	4+2 = 6 3+4+2 = 9 18 – 9 = 9	(**3,4,2,**6,3)
{3,4,2,6,3,18,15,7,11,6,9,9,**8,12,15**}	2+6 = 8 4+2+6 = 12 3+4+2+6 = 15 18 – 15 = 3 (już uwzględnione, ponieważ był to ostatni element mapy)	(**3,4,2,6,**3)
{2,3,3,4,6,6,7,8,9,9,11,12,15,15,18}	**KONIEC (sortowanie multizbioru)**	(3,4,2,6,3)

Cel projektu

W skrócie, należy zaimplementować algorytm który zajebie operacje „na odwrót". Czyli mając już jakiś multizbiór, musimy zrekonstruować mapę cięć restrykcyjnych. (To właśnie jest problem mapowania metodą częściowego trawienia (*Partial Digest Problem dla zainteresowanych*). Algorytm ma stosować przeszukiwanie z nawrotami (czyli jeśli dane rozwiązanie jest złe, to wraca do momentu gdzie podejmował decyzję), oraz mieć złożoność wykładniczą (najprościej: każda rekurencja).

Czyli składając to do kupy: Mamy stworzyć program, który na podstawie multizbioru odtworzy mapę cięć restrykcyjnych. Algorytm ma stopniowo budować rozwiązanie, a w razie błędu powrócić do momentu, gdzie podjął decyzję i kontynuować rozwiązanie. Dodatkowo, ma być rekurencją.

Budowa algorytmu:

Wczytywanie instancji:

Nasza instancja to multizbiór. Ma być on zaprezentowany jako pojedyncza linia z liczbami oddzielonymi spacją. Prosta sprawa, na tym etapie wczytywanie czy to do wektora, czy do innej struktury było trzaskane milion razy

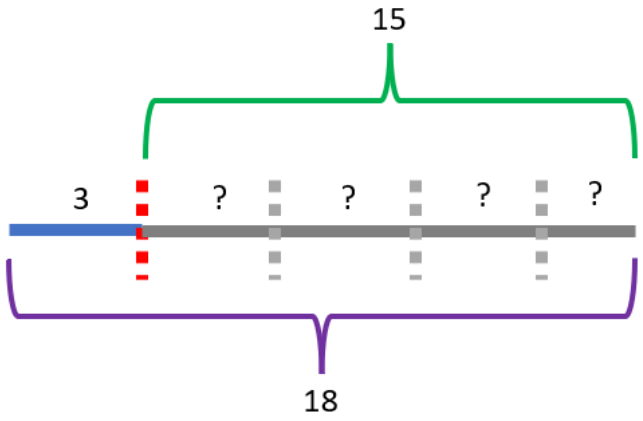
Preprocessing:

Generalnie nasz program musi brać pod uwagę to, że wczytana instancja może być zła. Na tym etapie możemy skupić się tylko na dwóch rzeczach

1. Rozmiar
2. Pierwszy element

Co ro rozmiaru, istnieje wzór który opisuje zależność liczby cięć od wielkości multizbioru. Mianowicie: $|A| = \binom{k+2}{2}, gdzie\ |A| to\ ilość\ elementów\ w\ multizbiorze, a$ "k" $to\ liczba\ cięć$. (Proszę pamiętać, że to jest symbol Newtona a nie ułamek) Najlepszą metodą jest przekształcenie wzoru, tak by wyliczyć „k”, a następnie sprawdzić czy jest to liczba całkowita (czy to konwersją, czy porównywaniem z wektorem). Jeśli jest, to znaczy że rozmiar jest właściwy. Dodatkowo, k+1 to jest liczba elementów w multizbiorze.

Co do pierwszego elementu mapy, możemy od razu go wyznaczyć. Mianowicie, jest to różnica pomiędzy dwoma największymi elementami multizbioru. Jeśli takowego elementu nie ma, to na tym etapie jesteśmy w stanie odrzucić instancję. Dodatkowo, od razu mamy pierwszy element oraz „wykorzystane odległości” (więcej o tym dalej).



Konstruowanie mapy:

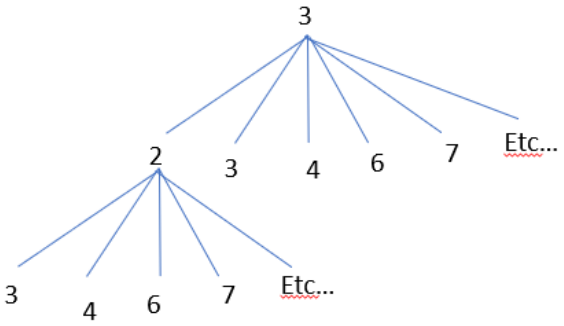
Tak jak ustaliliśmy: algorytm musi być rekurencyjny i stopniowo budować rozwiązanie. Z czego cały core tej procedury sprowadza się do wykonania na odwrót operacji opisanej w Trochę Teorii. Rozwiązanie budujemy stopniowo za pomocą multizbioru, gdzie całość opiera się poniekąd na algorytmie przeszukiwania drzewa. Ilość poziomów to tak naprawdę ilość elementów w mapie, gdzie pierwszy element już jest znany. Dodatkowo, z każdym kolejnym elementem rozwiązania wykorzystujemy odległości z multizbioru (na powyższym przykładzie, wykorzystaliśmy już jedną „3”,

jedną „15”, co do „18” (największego elementu) – pozostawienie jej zależy od przyjętej metody rekurencyjnej).

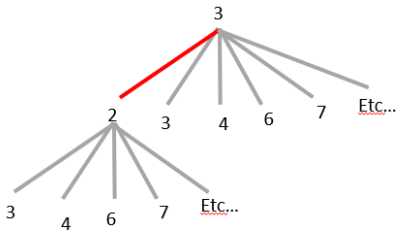
Multizbiór = {2,**3**,3,4,6,6,7,8,9,9,11,12,**15**,15,18} *Elementy wykorzystane: 3,15*

Mapa = (3,) *Pierwszy element wyliczony wcześniej*

Rozmiar mapy = 5



PRZYKŁAD 1: Niewłaściwy element

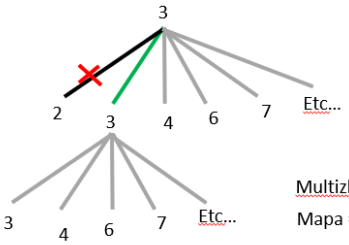


3+2 = 5

Nie ma tego w multizbiorze!

Musimy wziąć inny element

PRZYKŁAD 2: Właściwy element



Multizbiór = {2,**3**,**3**,4,**6**,6,7,8,9,9,11,**12**,**15**,15,18}

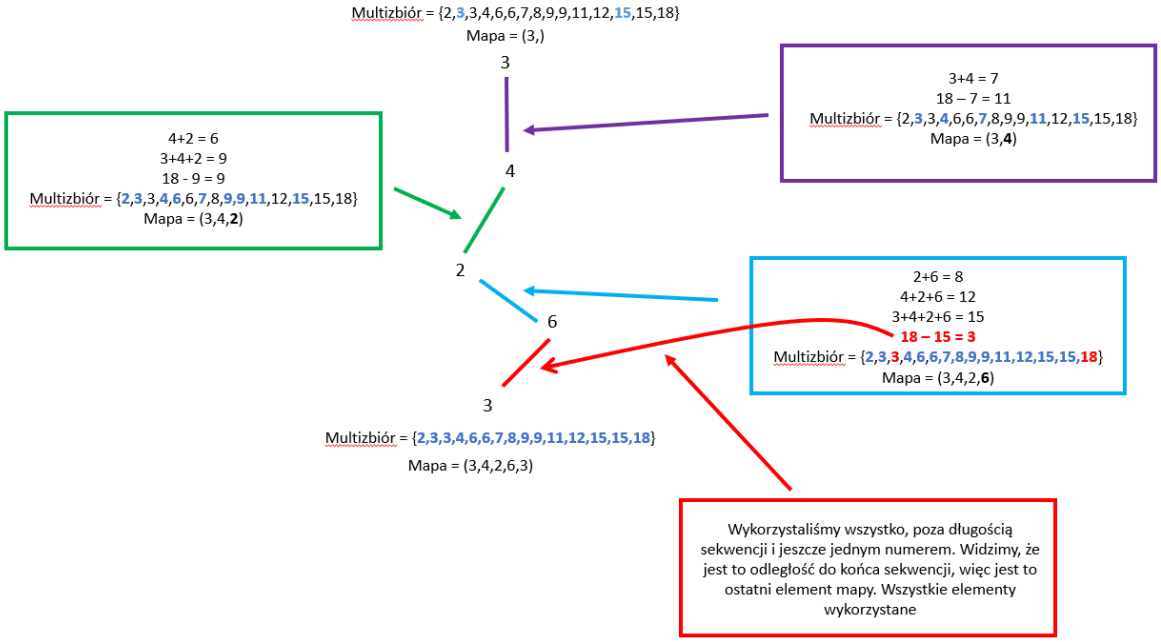
Mapa = (3,**3**)

3+3 = 6

18-6 = 12

Są w multizbiorze!

Schodzimy poziom niżej i aktualizujemy



W przypadku rekurencji można rozróżnić 2 rodzaje przejść:

a) Poziom niżej – kiedy znaleźliśmy pasujący element
b) Poziom wyżej – kiedy wyczerpaliśmy wszystkie elementy w danym poziomie. Wtedy należy powrócić poziom wyżej i podjąć inną decyzję

Dodatkowo trzeba pamiętać, że algorytm w jakiś sposób musi zapamiętywać postęp jaki poczynił. O ile łatwo o tym mówić w przypadku przejścia poziom niżej, tak w przypadku poziom wyżej trzeba przywrócić stan sprzed przejścia (no bo przecież nie wykorzystaliśmy „jakichś" elementów) + Warto zapamiętywać elementy, które nie dały nam rozwiązania na danym poziomie, bo w instancji mogą wystąpić powtórzenia (dlatego multizbiór).

Rekurencję z kolei można przerwać w dwóch wypadkach:

a) Ukończenie mapy
b) Dojście do poziomu od którego zaczynaliśmy

Całość problemu tego zadania opiera się na tym jak dobrze rozumiesz rekurencję. Czy kumasz mechanizmy przejścia dalej i powrotu, oraz jak dobrze zarządzasz pamięcią. Złożoność wykładnicza rządzi się tym, że w miare rozmiaru instancji, kurewsko zwiększa czas obliczeń. Dlatego trzeba korzystać z każdego możliwego triku. Jeśli z kolei kumasz rekurencję – to kod do tego zadania jest najprostszy ze wszystkich 3 projektów.

Tips & Tricks (czyi jak przyspieszyć program):

1. Podejście oprate na usuwaniu. Zauważmy, że wykorzystane elementy możemy równie dobrze usunąć z multizbioru. Ogranicza to mocno ilość przeszukiwania w miarę przechodzenia z poziomu na poziom. Jeśli chodzi o przywracanie danych – możemy do następnego poziomu posłać nowy multizbiór i mapę. Dzięki temu jeśli wrócimy poziom wyżej, to automatycznie przywrócimy kopię

2. Obliczanie przed przejściem na kolejny poziom nowej odległości do końca sekwencji. Oszczędzi to nam zazwyczaj parę setnych + możemy wtedy usunąć na wstępie usunąć długość sekwencji
    a. Dzięki temu możemy potem szybciej przerywać pętle, o ile badany element jest większy niż pozostała odległość
3. Struktura „optional” - Dzięki niej możemy uniknąć używania wskaźników do zakończenia działania rekurencji
4. Im mniej operacji wykonujemy i im mniej struktur mamy – tym lepiej