Projekt 1. Wyszukiwanie wzorców w sekwencji

Niech będzie dany ciąg zmiennych, np: “abba” nazywany wzorcem oraz ciąg wartości nazywany sekwencją. Mówimy, że wzorzec występuje w sekwencji jeżeli istnieje takie podstawienie wartości do zmiennych (bijekcja), że po podstawieniu wzorzec jest (nieciągłą) podsekwencją sekwencji. Przykładowo, dla wzorca “abba” oraz sekwencji “124353621” można wykonać następujące podstawienia:

- a=1 b=2: 124353621
- a=1 b=3: 124353621
- a=2 b=3: 124353621

Ponieważ przypisanie wartości musi być bijekcją, to nie jest możliwe przypisanie tej samej wartości do różnych zmiennych. Napisz program, który (w zależności od argumentu) znajdzie wszystkie możliwe przypisania dla których wzorzec jest podsekwencją sekwencji, bądź tylko jedno wystąpienie (sprawdzi czy wzorzec w ogóle występuje w sekwencji).

Projekt 2. Najdłuższa wspólna podsekwencja

Dane są dwa łańcuchy X i Y. Znajdź długość największej wspólnej (nieciągłej) podsekwencji oraz jej wystąpienia. Przykładowo, dla łańcuchów:

X=1232412

Y=243121

Najdłuższa wspólna podsekwencja ma długość 4. Przykładowymi wspólnymi sekwencjami są tutaj: 2412, 2312 i 2321.

Algorytm sekwencyjny rozwiązujący to zadanie to algorytm programowania dynamicznego. Należy przygotować tablicę o wymiarach (|X|+1)x(|Y|+1) w poniższy sposób:

	i		0	1	2	3	4	5	6	7
j	LCS		X	0	1	2	3	4	5	6
				1	2	3	2	4	1	2
0	Y									
1	0	2								
2	1	4								
3	2	3								
4	3	1								
5	4	2								
6	5	1								

Następnie, tabelkę należy wypełnić zgodnie z funkcją:

$$LCS[i][j] = \begin{cases} 0 & i = 0 \text{ lub } j = 0 \\ LCS[i-1][j-1] + 1 & X[i] = Y[j] \\ \max(LCS[i-1][j], LCS[i][j-1]) & X[i] \neq Y[j] \end{cases}$$

W powyższym przykładzie wyglądałoby to następująco:

	i		0	1	2	3	4	5	6	7
j	LCS		X	0	1	2	3	4	5	6
				1	2	3	2	4	1	2
0	Y		0	0	0	0	0	0	0	0
1	0	2	0	0	1	1	1	1	1	1
2	1	4	0	0	1	1	1	2	2	2
3	2	3	0	0	1	2	2	2	2	2
4	3	1	0	1	1	2	2	2	3	3
5	4	2	0	1	2	2	3	3	3	4
6	5	1	0	1	2	2	3	3	4	**4**

Komórka zaznaczona na czerwono oznacza długość najdłuższej wspólnej podsekwencji. Aby odnaleźć teraz wszystkie podsekwencje należy rozpocząć analizę wygenerowanej tablicy od tej komórki (początkowe wartości i,j). Algorytm wygląda następująco:

```
funkcja rek(i,j):
  if i==0 lub j==0 then
    Zwróć tablicę z jednym pustym łańcuchem.
  if X[i-1]!=Y[j-1] then
    tab1=tab2=pusta tablica
    if LCS[i-1][j]>=LCS[i[j-1] then
      tab1=rek(i-1,j)
    if LCS[i-1][j]<=LCS[i[j-1] then
      tab2=rek(i,j-1)
    Sklej tab1 i tab2 i zwróć wynik
  else if X[i-1]==Y[j-1] then
    tab=rek(i-1,j-1)
    Dodaj X[i-1] na końcu każdego łańcucha w tab i zwróć wynik.
```

Przykładowy przebieg rekursji wyglądałby następująco (jak widać algorytm rekurencyjny może generować duplikaty):

	i		0	1	2	3	4	5	6	7
j	LCS		X	0	1	2	3	4	5	6
				1	2	3	2	4	1	2
0	Y		0	0	0	0	0	0	0	0
1	0	2	0	0	1	1	1	1	1	1
2	1	4	0	0	1	1	1	2	2	2
3	2	3	0	0	1	2	2	2	2	2
4	3	1	0	1	1	2	2	2	3	3
5	4	2	0	1	2	2	3	3	3	4
6	5	1	0	1	2	2	3	3	4	4

Wskazówki:

1. Etap budowy zawartości tabelki nie jest taki trywialny jak na pierwszy rzut oka (nie da się jej całej wypełnić równolegle). Zastanów się, które wartości tabelki można wypełniać niezależnie, a które nie.
2. Drugi etap – zastanów się jak pozbyć się rekursji jeżeli masz dużo wątków z których część można „zmarnować”. Można to zrealizować wykonując podobny sposób przeglądania tablicy jak w przypadku pierwszego etapu.

Projekt 3. Znajdowanie i zliczanie wysp oraz obliczanie kursu statku

Dana jest dwuwymiarowa tablica wypełniona zerami i jedynkami. Zera oznaczają wodę, a jedynki ziemię. Grupy jedynek tworzą wyspy. Napisz program, który zliczy liczbę wysp.

Przykładowo, tablica mogłaby wyglądać następująco (jedynki kolor zielony, zera kolor niebieski):

	0	1	2	3	4	5	6	7	8	9	10	11
0	1	0	0	0	1	0	0	0	0	0	0	0
1	1	1	1	0	1	0	0	0	0	0	0	0
2	1	1	1	0	1	0	0	0	0	0	0	0
3	0	0	1	0	1	0	0	0	0	1	0	1
4	0	0	0	0	1	0	0	0	0	1	0	1
5	0	0	0	0	0	0	0	0	0	1	1	0
6	0	0	0	0	0	0	0	0	0	1	1	0
7	1	0	0	0	0	0	0	0	1	1	1	0
8	1	0	0	0	0	0	0	1	1	1	1	1

W powyższym przykładzie znajdują się 4 wyspy. Poniżej zaznaczono je różnymi kolorami:

	0	1	2	3	4	5	6	7	8	9	10	11
0	1	0	0	0	1	0	0	0	0	0	0	0
1	1	1	1	0	1	0	0	0	0	0	0	0
2	1	1	1	0	1	0	0	0	0	0	0	0
3	0	0	1	0	1	0	0	0	0	1	0	1
4	0	0	0	0	1	0	0	0	0	1	0	1
5	0	0	0	0	0	0	0	0	0	1	1	0
6	0	0	0	0	0	0	0	0	0	1	1	0
7	1	0	0	0	0	0	0	0	1	1	1	0
8	1	0	0	0	0	0	0	1	1	1	1	1

Program powinien zwrócić zarówno liczbę wysp, jak i listę współrzędnych komórek dla każdej z wysp.

Dodatkowo, statek chce przepłynąć z kolumny zerowej do ostatniej i trzeba mu zaplanować podróż. Algorytm powinien podać sekwencję kolejnych pól stanowiących jego trasę bądź zwrócić informację, że nie da się dopłynąć. Uwaga! Statek nie może płynąć na ukos (każdy kolejny krok to tylko zmiana jednej współrzędnej w tablicy) oraz zbliżyć się do brzegu (pomiędzy brzegiem a statkiem musi być co najmniej jedno pole 0). Przykładową trasę pokazano na poniższym rysunku.

	0	1	2	3	4	5	6	7	8	9	10	11
0	1	0	0	0	1	0	0	0	0	0	0	0
1	1	1	1	0	1	0	0	0	0	0	0	0
2	1	1	1	0	1	0	0	0	0	0	0	0
3	0	0	1	0	1	0	0	0	0	1	0	1
4	0	0	0	0	1	0	0	0	0	1	0	1
5	0	0	0	0	0	0	0	0	0	1	1	0
6	0	0	0	0	0	0	0	0	0	1	1	0
7	1	0	0	0	0	0	0	0	1	1	1	0
8	1	0	0	0	0	0	0	1	1	1	1	1

Projekt 4. Operacja łączenia zbiorów z algorytmu apriori

Niech będzie dana tablica T jednakowo licznych zbiorów elementów stanowiących podzbiory ustalonej i znanej dziedziny. Zakładamy, że na elementach dziedziny zdefiniowano całkowity porządek. Dla celów niniejszego zadania można założyć, że dziedzina ta składa się z liczb całkowitych od 0 do 31. Niech będą dane dowolne dwa zbiory A i B z tej tablicy takie, że:

1. $|A \cap B| = |A| - 1 = |B| - 1$
2. $\min\big((A \backslash B) \cup (B \backslash A)\big) > \max(A \cap B)$ (elementy stanowiące część wspólną są mniejsze od elementów, którymi zbiory się różnią)

Niech $C = A \cup B$. Jeżeli każdy podzbiór zbioru C o rozmiarze $|C| - 1$ znajduje się w tablicy T, to jest to zbiór, który powinien znaleźć się w wyniku. Napisz program, który wygeneruje wszystkie takie zbiory.

Przykładowo, niech będzie dana tablica T:

T
$\{1,2,3,4\}$
$\{1,2,3,5\}$
$\{1,2,3,6\}$
$\{1,2,4,5\}$
$\{1,3,4,5\}$
$\{1,3,4,6\}$
$\{2,3,4,5\}$
$\{2,3,4,6\}$

Pary zbiorów A i B spełniających omówione wyżej warunki, jak również ich sumę (zbiór C) i podzbiory zbioru C, przedstawiono w tabeli poniżej. Na czerwono zaznaczono podzbiory, których nie ma w tablicy T.

A	B	$C = A \cup B$	Podzbiory C
$\{1,2,3,4\}$	$\{1,2,3,5\}$	$\{1,2,3,4,5\}$	$\{1,2,3,4\}$, $\{1,2,3,5\}$, $\{1,2,4,5\}$, $\{1,3,4,5\}$, $\{2,3,4,5\}$
$\{1,2,3,4\}$	$\{1,2,3,6\}$	$\{1,2,3,4,6\}$	$\{1,2,3,4\}$, $\{1,2,3,6\}$, $\{1,2,4,6\}$, $\{1,3,4,6\}$, $\{2,3,4,6\}$
$\{1,2,3,5\}$	$\{1,2,3,6\}$	$\{1,2,3,5,6\}$	$\{1,2,3,5\}$, $\{1,2,3,6\}$, $\{1,2,5,6\}$, $\{1,3,5,6\}$, $\{2,3,5,6\}$
$\{1,3,4,5\}$	$\{1,3,4,6\}$	$\{1,3,4,5,6\}$	$\{1,3,4,5\}$, $\{1,3,4,6\}$, $\{1,3,5,6\}$, $\{1,4,5,6\}$, $\{3,4,5,6\}$
$\{2,3,4,5\}$	$\{2,3,4,6\}$	$\{2,3,4,5,6\}$	$\{2,3,4,5\}$, $\{2,3,4,6\}$, $\{2,3,5,6\}$, $\{2,4,5,6\}$, $\{3,4,5,6\}$

Z powyższej tabelki wynika, że do tablicy wynikowej powinien trafić jedynie zbiór $\{1,2,3,4,5\}$.

Wskazówki:

1. Jako reprezentacji zbioru użyj bitmapy przechowywanej w pojedynczym int-cie.
2. Do testowania, czy podzbiór jest w tablicy możesz użyć tablicy haszowej.

Projekt 5. Grupowanie k-medoids za pomocą algorytmu PAM

Grupowanie to nienadzorowana metoda eksploracji danych, w której podobne obiekty przydzielane są do różnych grup. W zależności od algorytmu grupowania liczba grup może być parametrem algorytmu, bądź też wyznaczana przez algorytm. Poszczególne algorytmy różnią się również reprezentacją grup.

W algorytmie PAM grupy reprezentowane są przez tzw. medoidy, tj. obiekty, które są najbardziej „typowe" dla danej grupy. Jeżeli atrybuty obiektów reprezentować jako współrzędne punktów w n-wymiarowej przestrzeni, to medoidy stanowiłyby punkty centralne grup. Liczba grup jest parametrem algorytmu.

Podobieństwo obiektów można wyrazić w różny sposób. W niniejszym projekcie załóż, że obiekty są opisane atrybutami liczbowymi (wymiarowość jest parametrem) a podobieństwo (a w zasadzie niepodobieństwo) pomiędzy punktami wyznacza się za pomocą metryki Euklidesowej.

Algorytm PAM działa w następujący sposób:

1. Niech k oznacza wybraną liczbę grup.
2. Wylosuj k obiektów i utwórz z nich medoidy
3. Oblicz dla każdego obiektu o odległość do najbliższego medoidu D_o i drugiego najbliższego medoidu E_o.
4. Przydziel obiekt do najbliższego medoidu i oblicz całkowity koszt grupowania jako sumę odległości obiektów od najbliższych medoidów.
5. Dla każdego obiektu o_h nie będącego medoidem i każdego obiektu o_i będącego medoidem oblicz jaka będzie zmiana całkowitego kosztu grupowania jeżeli zostaną one zamienione rolami. Weryfikację tą można wykonać w następujący sposób. Dla każdego obiektu o_j oblicz jego cząstkowy wpływ na tą zmianę rozważając 4 poniższe przypadki
   a. Jeżeli o_j jest dalej od o_i i o_h niż od innego medoidu (tego do którego należy w tej chwili), to cząstkowy wpływ tego obiektu wynosi 0, bo obiekt ten nie zmieni przydziału do medoidu po zamianie.
   b. Jeżeli o_j jest bliżej do o_i niż do jakiegokolwiek innego medoidu (przed zmianą) oraz $d(o_j, o_h) < E_{o_j}$, to o_j zostanie przydzielony do nowego medoidu. Wpływ obiektu na koszt grupowania wyniesie $d(o_j, o_h) - d(o_j, o_i)$.
   c. Jeżeli o_j jest bliżej do o_i niż do jakiegokolwiek innego medoidu (przed zmianą) oraz $d(o_j, o_h) \geq E_{o_j}$, to wtedy wpływ obiektu na koszt grupowania wyniesie $E_{o_j} - D_{o_j}$ (obiekt zostanie przydzielony do drugiego najbliższego medoidu przed zamianą).
   d. Jeżeli o_j jest dalej od o_i niż od jakiegoś innego medoidu, ale bliżej do o_h niż jakiegokolwiek innego, to zostanie przydzielony do nowego medoidu. W takiej sytuacji wpływ o_j na koszt grupowania wyniesie $d(o_j, o_h) - D_{o_j}$.

   Zsumuj wszystkie cząstkowe wpływy. Jest to „delta", którą należałoby dodać do kosztu grupowania. Oznacza to, że zamiana jest opłacalna jedynie wtedy, jeżeli delta jest ujemna. Znajdź zamianę dającą najmniejszą deltę.
6. Wykonaj zamianę najbardziej zmniejszającą koszt grupowania. Wróć do punktu 3. Algorytm kończy działanie, jeżeli nie będzie ujemnej delty.

Opracuj wersję algorytmu PAM zrównoleglającą obliczenia na GPU. Zwróć uwagę na reprezentację danych w pamięci, która powinna minimalizować liczbę transakcji z pamięcią globalną.