Opis teoretyczny projektu na przedmiot Algorytmy i Struktury Danych II

Skład grupy

- Adam Nastula
- Michalina Lasek
- Igor Jeziorski

Projekt

https://github.com/AdamNastula/algorytmy_II_projekt

Problem 1

Treść problemu

Ustalić, w jaki sposób są transportowane odcinki z fabryki do miejsca budowy płotu. Możliwie szybko i możliwie małym kosztem zbudować płot.

Podproblem a)

Dobrać płaszczaki w pary zgodnie z założeniami z treści projektu, tzn.:

- dobrać płaszczaki w pary w taki sposób, aby każda para składała się z jednego płaszczyka z rękami z przodu (prawego) i jednego z rękami z tyłu (lewego).
- dobrać płaszczaki w pary w taki sposób, aby w każdej parze płaszczaki lubiły się nawzajem

W naszym projekcie rozumiemy relację lubienia jako relację symetryczną (jeżeli A lubi B, to B lubi A).

Rozwiązanie

Aby rozwiązać ten problem, skorzystamy z algorytmu wyznaczania maksymalnego skojarzenia w grafie dwudzielnym prezentowanego na wykładzie. Graf dwudzielny tworzymy w następujący sposób:

- zbiorem wierzchołków grafu jest zbiór płaszczaków. Dzieli się on na dwa rozłączne podzbiory - płaszczaki lewe oraz prawe.
- krawędzie w naszym grafie będą tylko pomiędzy parami płaszczaków lewych i prawych, które się lubią

Znalezienie maksymalnego skojarzenia w tak skonstruowanym grafie wyznaczy nam pary płaszczaków, które będą mogły razem transportować odcinki płotu z fabryki na miejsce budowy. W rozwiązaniu implementujemy algorytm Edmondsa-Karpa.

Złożoność rozwiązania

Jako że korzystamy z algorytmu Edmondsa-Karpa, który wykorzystuje algorytm BFS do przeszukiwania sieci rezydualnej, nasze rozwiązanie tego podproblemu ma złożoność czasową $O(V * E^2)$. V jest to moc zbioru wierzchołków, natomiast E to moc zbioru krawędzi.

Pod problem b)

Możliwie małym kosztem zbudować płot.

Rozwiązanie

Do rozwiązania tego problemu znajdziemy otoczkę wypukłą zbioru punktów orientacyjnych rozmieszczonych w krainie płaszczaków. W tym celu wykorzystujemy algorytm Chana

Źródła:

- https://en.wikipedia.org/wiki/Chan%27s_algorithm
- https://jeffe.cs.illinois.edu/teaching/compgeom/notes/14-convexhull.pdf

Złożoność

Algorytm Chana wykorzystuje algorytm Grahama oraz Jarvisa, aby uzyskać świetną złożoność czasową $O(n * log(h))$. Parametr n to moc zbioru punktów, którego otoczkę wyznaczamy, natomiast h to moc zbioru punktów otoczki.

Implementacja problemu 1

Aby uzyskać rozwiązanie pierwszego problemu, należy uruchomić plik problem1.py, który jest częścią naszego repozytorium.

Wejście

W pierwszej linii wejścia powinna znaleźć się liczba naturalna n oznaczająca ilość par płaszczaków, które się lubią.
W kolejnych n liniach oczekujemy par imion płaszczaków oddzielonych pojedyncza spacją. Linia “Artur Bartek” oznacza, że Artur i Bartek lubią się nawzajem.
W następnej linii oczekujemy liczby naturalnej m, która oznacza moc zbioru punktów orientacyjnych w krainie płaszczaków.
Następne m linii powinno zawierać pary liczb rzeczywistych, oddzielonych pojedynczą spacją, oznaczających współrzędne kolejnych punktów orientacyjnych.

Przykładowe wejście:

```
10
Ludie Richard
Ludie Joseph
Laura Frank
Leslie Albert
Kevin Joseph
Donna Mary
Ludie Albert
Jill Mary
Leslie Frank
Laura Delia
10
65 177
120 56
40 73
76 0
126 101
74 165
```

108 18
79 84
188 56
133 73

Wyjście

W pierwszej linii wyjścia drukowana jest liczba naturalna n informująca o tym, ile par robotników udało się znaleźć (moc maksymalnego skojarzenia w naszym grafie dwudzielnym).
W kolejnych n liniach drukowane są nasze pary w postaci “Adam Michał”. Oznacza to, że Adam powinien współpracować z Michałem.
W następnej linii pojawi się liczba naturalna m. będąca mocą zbioru punktów tworzących otoczkę wypukłą zbioru punktów orientacyjnych.
W kolejnych m liniach pojawiać się będą punkty należące do otoczki w kolejności zgodnej z przechodzenia ich zgodnie z ruchem wskazówek zegara.

Przykładowe wyjście (dla danych wejściowych pokazanych wyżej):

5
Leslie Albert
Laura Delia
Kevin Joseph
Donna Mary
Ludie Richard
4
65.0 177.0
188.0 56.0
76.0 0.0
40.0 73.0

Problem 2

Treść problemu

Zapisać opowieść-melodię w maszynie Informatyka, zamieniając wcześniej „poli” na „boli” oraz próbując oszczędzić wykorzystane miejsce. Znaleźć rozwiązanie problemu ewentualnej zamiany innych fragmentów opowieści-melodii, który niepokoi Heretyka oraz Informatyka.

Podproblem a)

Znaleźć wszystkie wystąpienia „poli” i zamienić je na „boli”.

Rozwiązanie

Jest to klasyczny problem znalezienia wzorca w tekście. Do jego rozwiązania zaimplementowaliśmy algorytm Boyera-Moora. Algorytm ten zwróci wszystkie miejsca, w których rozpoczyna się „poli” w naszej melodii, co pozwoli na szybką ich zamianę na dobry dźwięk.

Złożoność

Złożoność algorytmu Boyera-Moore'a różni się w zależności, czy szukany wzorzec występuje w tekście, czy nie. W naszym przypadku zakładamy, że „poli” zawsze wystąpi w melodii (po przeczytaniu treści projektu ma to jak najbardziej sens). Przyjmujemy zatem złożoność $O(n * m)$.
Parametr n to długość melodii, natomiast m to długość wzorca.

Podproblem b)

Zapisać melodię w maszynie informatyka, zajmując jak najmniej miejsca.

Rozwiązanie

Kompresji melodii dokonamy, stosując algorytm Huffmana. Algorytm Huffmana sprawdzi się idealnie, ponieważ wyznaczy nam kody zero-jedynkowe dla każdej „nuty” w melodii, a więc będziemy w stanie wpisać ją do maszyny informatyka.

Złożoność

Algorytm wyznaczania kodów zaimplementowaliśmy w sposób opisany przez Huffmana. Ma on złożoność $O(n * log(n))$. Parametr n to długość tekstu. Wyznaczanie częstotliwości występowania poszczególnych liter zaimplementowana jest w czasie liniowym. Kompresja danych również działa w czasie liniowym.

Podproblem c)

Znaleźć rozwiązanie potencjalnej zmiany innych fragmentów melodii.

Rozwiązanie

Jako że skompresowana melodia zapisana jest w postaci ciągu binarnego, wykorzystamy kodowanie Hamminga. Kodowanie Hamminga pozwala na wykrycie i skorygowanie pojedynczego przekłamania bitu w słowie Hamminga.

Złożoność

Wyznaczenie kodu korekcyjnego działa w czasie liniowym. Sprawdzenie, czy w wiadomości wystąpił błąd oraz naprawienie go, również działa w czasie liniowym.

Implementacja problemu 2

Aby uzyskać rozwiązanie pierwszego problemu, należy uruchomić plik problem2.py, który jest częścią naszego repozytorium.

Wejście

W pierwszej linii wejścia powinna znaleźć się melodia zapisana jako jedna linia tekstu. Następnie w kolejnej linii powinna znaleźć się liczba naturalna n, będąca mocą alfabetu użytego do zapisania melodii. W kolejnych n liniach powinny znaleźć się kolejne litery naszego alfabetu. **UWAGA** litera ‚A’ i ‚a’ to dwie różne litery i należy podać je osobno.

Przykładowe wejście:

```
polipoliAAAApoliBBBBXXXXCCCCWWWWpolipoli
9
p
o
l
i
A
B
X
C
W
```

Wyjście

Pierwsze n linii wejścia jest postaci **[litera z alfabetu][spacja][kod Huffmana dla tej litery].**
Kolejna linia zawiera liczbę naturalną mówiącą ile wystąpień “poli” zostało znalezionych
W kolejnej linii wypisana zostaje melodia z zamienionymi wystąpieniami dźwięku „poli” na „boli”.
Następna linia to ciąg binarny odpowiadający melodii skompresowanej przy pomocy kodów Huffmana.
Ostatnia linia to skompresowana melodia zakodowana kodami Hamminga.

Przykładowe wyjście (dla danych wejściowych pokazanych wyżej):

W 000
l 001
b 010
o 011
i 100
B 101
p 1100
X 1101
A 1110
C 1111
5
boliboliAAAAboliBBBBXXXXCCCCWWWWboliboliBpppB
0100110011000100110011001110110111011100100110011001011011011101110111011101111111111111111110000
0000000001001100110001001100110010111001100110 0101
1100110001100110101010100011001101001011110111010111110111100100111111000110010101111011101101111 00
0011110111011001100111111111110111011101111000111010001000000100101011000110001001100111001100001100
1111001100100111000101000 0

Problem 3

Treść problemu

Ustalić jak najszybciej grafik pracy strażników i jak najmniejszą liczbę odsłuchań melodii dla każdego strażnika.

Rozwiązanie

Do rozwiązania tego problemu wykorzystamy algorytm zachłanny działający następująco:
Zaczynamy w pierwszej wieży. Przeszukujemy następne wieże w poszukiwaniu wieży o największej jasności, ale mniejszej od jasności wieży, w której aktualnie się znajdujemy. Jeśli ją znajdziemy, to do niej idziemy i powtarzamy wyżej opisane czynności. Jeżeli nie znajdziemy takiej wieży, to idziemy do wieży o największej jasności i słuchamy melodii. Procedurę kończymy, gdy znajdziemy się znowu w początkowej wieży. Wyznaczy nam to trasę dla każdego strażnika, jako że jego energia nie ma wpływu na ten proces (brak informacji o tym w treści projektu). Ze zbioru wszystkich płaszczaków wyznaczymy siedmiu strażników, wybierając tych, którzy mają najwięcej energii. Jako że energię strażnik odzyskuje po tygodniu bez pracy, siedmiu strażników wystarczy nam, aby każdego dnia ktoś patrolował płot.

Złożoność

Zauważmy, że nasz algorytm działa najgorzej wówczas, gdy dla każdej wieży, wieża do której należy się udać w następnej kolejności, jest pierwszą wieżą zaraz po naszej. Prezentuje to ciąg jasności postaci:
9 -> 8 -> 7 -> 6 -> 5 - > 4 itd…
Wówczas dla każdej wieży (nie licząc m ostatnich) będziemy przeglądać jasności m następnych wież, a i tak przesuniemy się tylko o jedną do przodu. W tym pesymistycznym przypadku złożoność naszego algorytmu wynosić będzie $O(n * m)$. Parametr n to ilość wież na trasie strażnika, natomiast m to maksymalne oddalenie kolejnych wież, w których strażnik musi wykonać postój.

Implementacja problemu 3

Aby uzyskać rozwiązanie pierwszego problemu należy uruchomić plik problem3.py, który jest częścią naszego repozytorium.

Wejście

W pierwszej linii wejścia powinna być liczba naturalna n informująca o ilości wież na trasie.
W kolejnych n liniach powinny być parametry opisujące wieże w postaci
X Y Z
X - liczba rzeczywista opisującą położenie wieży w krainie płaszczko wzdłuż osi X
X - liczba rzeczywista opisującą położenie wieży w krainie płaszczko wzdłuż osi Y
Z - liczba naturalna opisująca jasność wieży
W kolejnej linii powinna pojawić się liczba naturalna q >= 7 informująca o ilości płaszczaków chętnych do patrolowania muru, a następnie q linii postaci
A B
A - imię płaszczaka
B - liczba naturalna opisująca ilość jego energii
W ostatniej linii powinna się znaleźć liczba naturalna m, która informuje o maksymalnym oddaleniu kolejnych wież w których trzeba dokonać postoju.

Przykładowe wejście:

```
20
5 12 76
0 18 99
7 15 43
20 20 54
14 3 62
9 7 85
1 19 45
6 11 90
13 17 70
2 8 95
15 10 80
4 14 66
18 5 59
11 16 88
3 6 72
19 4 41
10 9 56
17 2 91
12 13 99
8 0 48
10
Adam 10
Maksio 15
Janek 6
Stas 20
Marek 11
Franek 23
Bartek 9
Olek 21
Igor 7
Nikodem 11
3
```

Wyjście

W pierwszej linii pojawi się liczba naturalna opisująca ilość odsłuchań melodii na trasie. Następnie w kolejnej linii pokaże się wydrukowana pythonowa lista prezentująca trasę strażnika. Liczby w apostrofach opisują indeks wieży, w

której strażnik się zatrzyma. * są oznaczone wieże, w których trzeba posłuchać melodii. Dalej, w kolejnych siedmiu liniach, wydrukowane są imiona i energia każdego strażnika, który został wybrany do patrolowania płotu.

Przykładowe wyjście (dla danych wejściowych pokazanych wyżej):

```
2
['3', '6', '9*', '10', '11', '12', '15', '18*', '20']
Franek  23
Olek  21
Stas  20
Maksio  15
Nikodem  11
Marek  11
Adam  10
```

Poprawność rozwiązań

Do rozwiązania problemu 1. oraz 2. wykorzystujemy powszechnie znane techniki i algorytmy, których poprawności działania nie będziemy dowodzić. W przypadku problemu 3. nie jesteśmy w stanie udowodnić poprawności zaprezentowanego rozwiązania zachłannego. W celu zbadania jego poprawności, wykonaliśmy szereg testów z algorytmem naiwnym, który sprawdza wszystkie możliwe kombinacje trasy strażnika i wyznacza najmniejsza ilość odsłuchu melodii.