Dokumentacja

Sobolewski Konrad
Indeks:270820
AAL - 10 - LS - *HOTEL*

Spis Treści:

1.Problem

Dana jest grupa n osób. Niektóre osoby znają się nawzajem. Relacja znajomości jest symetryczna. Tę grupę należy jak najtaniej zakwaterować w hotelu , w którym znajdują się jednoosobowe pokoje trzech rodzajów: z łazienką , ale bez telewizora (50zł), z telewizorem , ale bez łazienki (50zł) oraz bez łazienki i bez telewizora (30zł). Każda z osób lub jej bezpośredni znajomy musi posiadać dostęp zarówno do telewizora jak i do łazienki. W hotelu znajduje się nieograniczona liczba pokojów każdego rodzaju.

2. Obsługa programu

Interfejs:

1. **Pokaz informacje o projekcie.**

   Zawiera podstawowe informacje o projekcie.

2. **Generuj Osoby**

Wybór pozwala na wygenerowanie osób które w późniejszym etapie zakwaterujemy. Aby poprawnie wprowadzić dane należy podać liczbę osób większą od 1 . Wprowadzenie mniejszej ilości osób jest sprzeczne z założeniami zadania - nie ma pokoi z telewizorem i łazienką. W celu tej operacji posługuje się funkcją *quantity_of_people*.
Po wybraniu ponownie wszystkie poprzednie dane są kasowane .

3. **Generuj Zwiazki**

Wybór służy do generacji losowej lub ręcznej związków. Liczba relacji m musi być m>=n-1 oraz m <=n*(n-1)/2.

a) **Ręczny:**

W przypadku generacji ręcznej możemy przerwać wprowadzania danych poprzez 0. Wprowadzania działa na zasadzie podania pierwszej osoby i po zatwierdzeniu kolejnej. Tak wprowadzona relacja tworzy krawędź grafu. Wykonywana jest funkcja *make_branches*, która na bieżąco monitoruje wprowadzane dane . Jeśli dane się powtórzyły lub wierzchołki grafu są identyczne operację należy powtórzyć.

b) **Losowy:**

W przypadku generacji losowej należy podać liczbę krawędzi do wygenerowania. Tworzony zbiór par cyfr losowych niepowtarzających się przy użyciu konstrukcji *set* i *pair* . Następnie generowane są krawędzie odczytując ich indeks ze wspomnianych konstrukcji. Jesli krawędź się powtórzy *set* nie wstawia jej do zbioru.

W czasie tworzenia krawędzi także zapisywane są wskaźniki do znajomych, do kontenera *vector_friends*.
Po wybraniu opcji ponownie poprzednie dane związków są kasowane .

4. **Uzyj algorymu**

Wybór powoduje użycie algorytmu do rozwiązania problemu z treści zadania jeśli wszystkie dane zostały wprowadzone. Szczegółowy opis działania w dalszej części sprawozdania.

Podczas wykonywania się algorytmu liczony jest czas wykonania. Poprzednie dane są kasowane i wyliczane 30 razy po to aby uzyskać średni czas trwania operacji algorytmicznych.
Po wykonaniu się algorytmu spełnione są wszystkie wymagania do zrealizowania następnych punktów.

5. **Pokaz wyniki**

Wyświetla informacje o :

a) Ilosc osob z telewizorem,
b) Ilosc osob z lazienka ,
c) Ilosc osob z pustym pokojem,
d) Cena zakwaterowania wszystkich osób,
e) Sredni czas algorytmu.

6. **Testuj (dla małych danych)**

Funkcja służy do stestowania poprawność poszczególnych etapów algorytmu przydzielania pokoi oraz do wyświetlenia szczegółowych informacji o grafie takich jak:

a) Test poprawności sortowania.

Sprawdza poprawność sortowania malejąco krawędzi grafu względem sumy znajomych w grafie. Dane są wypisane.

b) Test znajomych

Sprawdza poprawność tworzenia znajomości. Wypisuje wszystkich znajomych oraz sumę znajomych każdej osoby w hotelu.

c) Test przydzielenia pokoi poszczególnym osobom

Sprawdza poprawność algorytmu . Wypisuje wszystkie osoby oraz jaki pokój dostały.

7. **Zamknij**

Powoduje wyłączenie programu.

3. Struktury danych

Do rozwiązania zadania występują 4 struktury danych:
Person,Edge,Menu,AboutAlgorithm

Person: przechowuje informacje o ilości swoich znajomych - posiada vector wskaźników *vector_friends* na obiekty klasy Person o ID innym niż swoim , własne ID, pokój jaki został jej przydzielony , oraz czy wystąpiła w czasie wykonywania algorytmu.

Edge: przechowuje informacje o krawędzi grafu - dwa wskaźniki na obiekty klasy Person o różnym ID, oraz wspólną sumę znajomych w krawędzi.

Menu : służy do interfejsu , zawiera dwa vectory : *vectorEdge , vectorPerson* , oraz służy do generacji danych:

- *showMenu* - wyświetla menu wyboru ,
- *typeChoice* - decyduje o wyborze oraz o tym czy może sie wykonać,
- *showProjectInfo* - wyświetla informacje podstawowe o projekcie,
- *quantity_of_people* - tworzy zadaną ilość obiektów klasy Person i umieszcza je w wektorze *vectorPerson*,
- *generateData* - decyduje czy krawędzie mają być tworzone losowo czy ręcznie,
- *make_relations* - słuzy do kontroli podanych danych,tworzy krawedzie w generacji losowej,
- *make_branches* - tworzy krawędzie i kontroluje czy nie wystąpił duplikat w czasie wprowadzania ręcznego,
- checkSolution - sprawdza po utworzeniu krawędzi czy każda osoba posiada znajomych. W przypadku ich brak nie da się przeprowadzić wykonania algorytmu,
- *showData* - wyświetla podstawowe informacje o wynikach,

- *useAlgorithm* - rozpoczyna procedure algorytmiczną,
- *tests* - wyświetla szczegółowe informacje o grafie , testując poprawność danych.

AboutAlgorithm : Zawiera następujące funkcje :

- *Search_vector* - przeszukuje vector znajomych klasy Person w celu odnalezienia osoby z telewizorem lub łazienką.
- *QuickSort* - sortuje vectorEdge w zależności od sumy znajomych w krawędzi malejąco
- *Algorithm* - główny algorytm programu opisany w dalszej części sprawozdania

4.Analiza pokrycia przypadków

W zadaniu można spotkać się z kilkoma przypadkami danych wejściowych :

1. Jeżeli liczba osób jest mniejsza od dwóch , należy wprowadzić dane ponownie. Zadanie w tym wypadku jest niemożliwe do wykonania ponieważ nie występuje pokój z telewizorem i łazienką.
2. Jeżeli liczba osób n>1 :
    a. Relacja między osobami wprowadzana ręcznie
    b. Relacja między osobami generowana losowo
3. Liczba relacji generowana ręcznie/losowo m musi być m>=n-1 oraz m <=n*(n-1)/2. Dla danych wprowadzonych poza danym przedziałem jest niemożliwe rozwiązania zadania. Dane należy wprowadzić ponownie.
4. W przypadku ręcznego wprowadzania związków, wprowadzania można przerwać przez cyfrę 0.
5. Jeśli w czasie losowej generacji lub przy ręcznym wprowadzaniu danych relacji, dane się powtórzą - są pomijane. W przypadku ręcznym występuje stosowny komunikat.
6. Jeśli w czasie losowej generacji lub przy ręcznym wprowadzaniu danych relacji , początek i koniec relacji jest tą samą osobą - są pomijane. W przypadku ręcznym występuje stosowny komunikat.

7. Jeśli wystąpi osoba bez żadnego związku z innymi lokatorami , nie możliwe jest spełnienie wymogu zadania z dostępem do telewizora i łazienki, dane należy wygenerować/wprowadzić ponownie.
8. W razie chęci powtórzenia operacji, po ponownym wyborze stare dane są kasowane. Umożliwia to przeprowadzenie testów na wielu rodzajach grafów bez zamykania programu.

5.Algorytm problemu

Algorytm polega na przeszukiwaniu krawędzi w grafie.
Etapy:

a) Pierwszym etapem jest posortowanie utworzonych krawędzi malejąco względem ilości sumy znajomych w krawędzi używając funkcji *QuickSort*. Umożliwia to rozpoczęcie przydzielanie pokoi od krawędzi, która ma największy wpływ w grafie. Zwiększa to prawdopodobieństwo posiadania pustych pokoi dla innych osób co ostatecznie zmniejsza koszt pobytu w hotelu.
   Algorytm bez sortowania , przydzielał by pokoje w kolejności losowej. Zwiększało by to możliwość nieoptymalnego przydzielenia pokoi. Wykonanie sortowania według innych kryteriów (np. Rosnąco) okazałoby się mniej optymalne , gdyż wybór krawędzi zaczynałby się od mniej znaczących w grafie co powodowałoby zmniejszenie ilości pustych pokoi.
b) Wybór końca krawędzi (osoby) do przydzielenia pokoju na podstawie ilości znajomych . Jeśli koniec A ma więcej lub tyle samo znajomych co B , wybieram A. W innym przypadku B.
   Wykonanie podpunktu a) i b) pozwala na wybranie osoby w grafie która ma największe znaczenie w zbiorze danych. Przydzielenie jej pokoju zapewnia dostęp wielu osobom do jednego udogodnienia, co zwiększa prawdopodobieństwo przydzielenie im tańszego pokoju.
c) Główna procedura. Używana jest funkcja *Algorithm,* która przydziela pokój osobie w zależności od jej znajomych oraz przeciwnego końca krawędzi.

Zasada działania:

- Jeśli osoba A nie ma przydzielonego pokoju , przystępujemy do następnego etapu. W innym przypadku wychodzimy z funkcji.
- Sprawdzamy czy osoba B(przeciwległy koniec krawędzi) ma przydzielony pokój. Jeśli nie ma to:
    - sprawdzamy czy flaga *done* osoby A jest ustawione. Ma ona na celu informowanie nas czy brak pokoju z telewizorem lub łazienką jest celowym zabiegiem . Ustawienie jej powoduje pomijanie danej osoby w następnej iteracji algorytmu.
    - Jeśli flaga ta jest nieustawiona przeszukujemy wszystkich znajomych osoby A funkcją *search_vector* . Zwraca ona pierwszy typ pokoju z telewizorem lub łazienką jaki odnajdzie w momencie przeszukiwania. Jeśli odnajdzie telewizor lub nic nie znajdzie, przydzielamy osobie A pokój z łazienką i ustawiamy flagę *done.* Jeśli odnajdzie pokój z łazienką, przydzielamy pokój z telewizorem. Funkcja kończy swe działanie.
- Jeśli B ma przydzielony pokój:
    - przeszukujemy wszystkich znajomych osoby A funkcją *search_vector .*
    - Jeśli B ma pokój z łazienką ,a typ pokoju odnalezionego to pokój z telewizorem lub jeśli B ma pokój z telewizorem ,a typ pokoju odnalezionego to pokój z łazienką ,oznacza to że A ma dostęp do obu typów pokojów. Dla A Ustawiamy flagę *done* i wychodzimy z funkcji.
    - Jeśli B ma pokój z łazienką ,a typ pokoju odnalezionego to łazienka lub brak pokoju to przydzielamy A pokój z telewizorem. Wychodzimy z funkcji
    - Jeśli B ma pokój z telewizorem ,a typ pokoju odnalezionego to telewizor lub brak pokoju to

przydzielamy A pokój z łazienką. Wychodzimy z funkcji

Szacowana złożoność asymptotyczna algorytmu wynosi $O(n^2)$. Występują dwie pętle for zagnieżdżające się. W optymistycznym aspekcie wykonania algorytmu złożoność szacowana *od do u* wynosi O(n) . Istnieje możliwość probablistyczna , że funkcja *search_vector* wykona się za każdym razem w pierwszej iteracji programu. Ma to ogromny wpływ na czas wykonywania się algorytmu. Sam algorytm znajduje jedynie przybliżony wynik(często optymalny). Uzyskanie rozwiązania optymalnego jest zależne od danych wejściowych i relacji łączących osoby. Przydzielanie pokoi w jednej części grafu wpływa na przebieg przydzielania w pozostałej części. Znalezienie rozwiązania dla przypadku globalnego jest niezwykle trudne , dlatego dalsze szukanie rozwiązania bardziej optymalnego zostało przeze mnie porzucone.

6.Pomiar czasu wykonania

Dla ilości krawędzi mniejszej od 50000 czas wykonania algorytmu był niezauważalny bądź ponowne testy posiadały dużą rozbieżność .W wyniku czego postanowiłem rejestrację wyników od 50000 elementów wzwyż.
Do generacji współczynnika q(n) posłużył mi skrypt Matlaba zawarty w folderze z plikami.

Algorytm z asymptotą O(T(n))=O(n)			
Ilość krawędzi	**t(n) ms**	**q(n)**	**$q(n^2)$**
50000	**56**	**0.8175**	**3.2701**
60000	**67**	**0.8151**	**2.7170**
80000	**92**	**0.8394**	**2.0985**

100000	**120**	**0.8759**	**1.7518**
120000	**151**	**0.9185**	**1.5308**
150000	**197**	**0.9586**	**1.2782**
200000(med)	**274**	**1.0000**	**1.0000**
250000	**380**	**1.1095**	**0.8876**
300000	**474**	**1.1533**	**0.7689**
350000	**597**	**1.2450**	**0.7115**
400000	**702**	**1.2810**	**0.6405**
450000	**798**	**1.2944**	**0.5753**
500000	**907**	**1.3241**	**0.5296**

Szacowana złożoność czasowa to $O(T(n)) = O(n^2)$.
Rzeczywista złożoność czasowa okazała się inna . Dla q(n) wyszło niedoszacowanie, dla $q(n^2)$ spore przeszacowanie. Wynika to z występującej funkcji *search_vector ,* która kończy swoje przeszukiwanie znajomych na pierwszej odnalezionej osobie z łazienką lub telewizorem. Zgodnie z powyższą tabelą może oszacować , że dzieje się to dość szybko ,dlatego rzeczywistej złożoności czasowej jest bliżej do O(n). Jego zachowanie jest zależne od danych wejściowych.