Tomasz Dakowicz
Przedmiot: **AAL**
Zadanie: **T2 – Mrówki**

Dokumentacja końcowa

1. Treść zadania

T2 – Mrówki
Dana jest sieć mrowisk połączonych między sobą ścieżkami. Z każdego mrowiska mrówka może dotrzeć do każdego innego. Natomiast wyruszając w trasę z danego mrowiska nie ma możliwości do niego wrócić bez zawracania. Należy opracować N tras, które rozpoczynają się i kończą w jednym z mrowisk, tak aby mrówka dotartła do jak największej liczby z nich.
Porównać czas obliczeń i wyniki różnych metod.

2. Analiza przedstawionego problemu.

N – liczba wierzchołków w grafie
P – zadana liczba możliwych do wyznaczenia ścieżek

Z treści zadania wynika, że dane jest drzewo, w którym chcemy uzyskać jak największe pokrycie wierzchołków, mając do dyspozycji wyłącznie P tras (przy czym trasy mogą zaczynać się i kończyć w różnych wierzchołkach).

Wiedząc, że będziemy korzystać wyłącznie z drzew (czyli grafów spójnych acyklicznych), optymalnym jest posługiwanie się wyłącznie trasami, które zaczynają się i kończą w liściach drzewa – całe zadanie polega więc na umiejętnym ich połączeniu.

3. Rozwiązania problemu.

3.1 Rozwiązanie oczywiste (naiwne)

W tym rozwiązaniu zostaną wygenerowane wszystkie możliwe ścieżki między liśćmi, w celu wybrania P takich ścieżek, które w sumie zapewnią największe pokrycie drzewa. Między dwoma dowolnymi mrowiskami istnieje tylko jedna ścieżka (zgodnie z definicją drzewa).
L – liczba liści w drzewie.
$\overline{d}$ - średnia długość wygenerowanej ścieżki

Koszt wygenerowania wszystkich ścieżek dla 1 wierzchołka (jedno przejście BFS dla jednego wierzchołka) - $(|V|+(|E|-1))(L-1)\overline{d}=(2N-1)(L-1)\overline{d}=2\overline{d}NL$
Generujemy ścieżki tylko dla liści, więc sumaryczny koszt:
$L(2N-1)(L-1)\overline{d}=2\overline{d}NL^2$

Powyższy koszt dotyczy wygenerowania wszystkich ścieżek – pozostaje problem wybrania największego sumarycznego pokrycia drzewa.
W tym celu wystarczy wybierać spośród niewybranych jeszcze ścieżek tę, która zawiera najwięcej dotąd niewybranych wierzchołków – czyli dodatkowo trzeba przejrzeć każdy wierzchołek każdej ścieżki, wyliczyć aktualne pokrycie i na podstawie tego wybrać ścieżkę. Powtórzyć P razy.
$L(L-1)P\overline{d}=\overline{d}PL^2$

Ostatecznie (zaniedbując $\bar{d}$):
$O(PL^2+2NL^2)=O(L^2(P+2N))$

3.2 Rozwiązanie zaproponowane w koncepcji wstępnej

Opracowując rozwiązanie optymalne należy posłużyć się wyłącznie takimi trasami, które zawierają możliwie najwięcej nieodwiedzonych dotąd wierzchołków – w tym celu użyjemy wielokrotnie algorytmu na wyznaczanie średnicy grafu.

Niech średnicą grafu będzie trasa, która zawiera maksymalną liczbę nieodwiedzonych wierzchołków w danym grafie. Algorytm wyliczania powyższej średnicy wygląda następująco:

- bierzemy dowolny nieodwiedzony liść grafu (liście to wierzchołki o stopniu 1)
- przechodzimy BFSem począwszy od tego wierzchołka zapisując aktualną liczbę nieodwiedzonych wierzchołków dla danej trasy
- bierzemy wierzchołek w_a, dla którego stwierdzamy maksymalną wartość uzyskaną w poprzednim kroku
- ponownie przechodzimy BFS począwszy od w_a, tym razem pamietając dodatkowo poprzednika w celu odtworzenia trasy
- znajdujemy w_b, dla którego uzyskaliśmy maksymalną liczbę nieodwiedzonych wierzchołków
- odtwarzamy trasę $w_a, w_0, \ldots, w_n, w_b$ i zaznaczamy wierzchołki te jako odwiedzone

Powyższy algorytm powtarzamy dopóki istnieje nieodwiedzony liść lub dopóki mamy do dyspozycji utworzenie kolejnej trasy. Jeśli uzyskaliśmy już całe pokrycie grafu oraz posiadamy dodatkowe niewykorzystane trasy – kończymy działanie programu (każda dodatkowa trasa przecież I tak nic nie zmieni w pokryciu grafu).

Złożoność rozwiązania – jedno przejście BFSem to koszt czasowy $O(|V|+|E|)=O(2N-1)$
Na potrzeby każdej ścieżki wyznaczamy 2 takie przejścia.
Otrzymujemy więc $O(2(2N-1)P)=O(4PN)$

4. Implementacja

Program został stworzony i zaimplementowany w C++, kompiluje się i uruchamia zarówno na Linuksie, jak i Windowsie.

Kompilacja: **g++ *.cpp -std=c++11 -o main**

Uruchomienie (wg trybu wykonania oraz źródła danych wejściowych):

- strumien wejsciowy -> **./main –input**
- dane generowane losowo -> **./main –random**
- generacja danych & pomiar & prezentacja -> **./main --complex**

Struktura danych programu została zrealizowana w klasie Tree, która z kolei działa na obiektach klasy Node, reprezentującej poszczególne wierzchołki grafu.

5. Tryby działania programu

➔ **Wykonanie programu na podstawie danych wprowadzanych ręcznie (--input)**

Podstawowy tryb działania programu – w ramach tego trybu użytkownik wybiera algorytm, wielkość danych wejściowych (liczbę mrowisk oraz możliwych ścieżek) oraz wprowadza (|V| - 1) par liczb, reprezentowanych jako połączenia w grafie.
Nie odbywa się weryfikacja danych wejściowych w celu spradzenia, czy wczytany graf jest drzewem – użytkownik ma pełny wpływ na wprowadzane dane.
Na standardowym wyjściu użytkownik otrzyma: skonstruowane ścieżki oraz sumaryczne pokrycie grafu.

➔ **Wykonianie programu na podstawie danych generowanych losowo (--random)**

W ramach tego trybu użytkownik wybiera, podobnie jak w powyższym trybie, liczbę mrowisk oraz liczbę ścieżek – jednakże tym razem wszystkie połączenia zostaną wygenerowane losowo w celu otrzymania drzewa. Wywołanie programu spowoduje przetestowanie obu algorytmów.
Na standardowym wyjściu użytkownik otrzyma: skonstruowane ścieżki oraz sumaryczne pokrycie grafu dla obu metod.

➔ **Wykonanie z generacją danych, pomiarem czasu oraz prezentacją wyników(--complex)**

W ramach tego trybu użytkownik wybiera liczbę zestawów danych do testów.
Dla każdego testu podaje standardowe dane, do nich generowane są losowe połączenia, zgodnie z zasadą podaną w trybie –random. Dodatkowo, dla każdego testu jest zapamiętywana liczba liści w grafie oraz czasy wykonania dla obu metod.
Po skończeniu przetwarzania wszystkich testów następuje prezentacja wyników w formie tabeli.

6. Pomiary czasów

N – liczba wierzchołków grafu
P – liczba możliwych ścieżek do ułożenia
Liczba liści w generowanych drzewach jest równa średnio połowie **N.**

$$q(N,P)=\frac{t(N,P)}{cT(N,P)}=\frac{t(N,P)T(N_{mediana},P_{mediana})}{T(N,P)t(N_{mediana},P_{mediana})}$$

Algorytm brutalny (czasy uśrednione z 15 pomiarów):
$O(L^2(P+2N))$

N	P	t(N, P)[ms]	q(N, P)
100	**10**	32.208	2.3867178271
200	10	129.373	1.2275986867
300	10	297.220	0.8424852374
400	10	584.689	0.7020637282
500	10	981.601	0.6049649644
100	**20**	34.946	2.4719028939
200	20	139.758	1.2945654421

300	20	358.573	1
400	20	644.111	0.7639826487
500	20	1004.116	0.6127739876
1000	**10**	4165.404	0.8052190469
1000	20	4647.21	0.8939102054
1000	30	5224.48	1
1000	40	5911.03	1.1258640789
1000	50	6563.34	1.2440103924

Algorytm optymalny (czasy uśrednione z 15 pomiarów):
$O(4PN)$

N	P	t(N, P)[ms]	q(N, P)
100	**10**	6.166	0.9721974037
200	10	12.215	0.962973669
300	10	19.396	1.0193934935
400	10	26.757	1.0546985862
500	10	34.738	1.095432806
100	**20**	11.859	0.9349082882
200	20	24.356	0.9600567614
300	20	38.054	1
400	20	51.365	1.0123443002
500	20	64.810	1.0218636674
1000	**10**	71.318	1.0139279198
1000	20	137.819	0.9796862782
1000	30	211.015	1
1000	40	279.646	0.993931711
1000	50	358.78	0.9897220577

7.Wnioski

Oszacowanie złożoności metody optymalnej okazało się zadowalające – pomiary to potwierdzają z dokładnością +- 3%.

Natomiast złożoność metody brutalnej okazała się zbyt skomplikowana (tj. zależna od wielu czynników). Podczas wyliczania q(N, P) pominięto średnią długość ścieżki, która z kolei zależy od gęstości drzewa – a to jest wyznaczane losowo, przy generowaniu połączeń.