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 – w celu stwierdzenia **poprawności**.
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 metodę rozwiązania.
Generowane jest **10 testów**, od zadanego N i P począwszy, zwiększając dla każdego kolejnego N o wczytaną różnicę. Dodatkowo, dla każdego testu jest zapamiętywana liczba liści w grafie oraz czasy wykonania. 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):
We wnioskach poniżej wyjaśnienie pomiarów

$O(L^2(P+2N))$

N	P	t(N, P)[ms]	q(N, P)
100	20	71	5.68993
200	20	168.978	2.24438
300	20	369.867	1.32747
400	20	676.061	1.17415
500	20	1280	1

600	20	1606.72	0.794096
700	20	2607.94	0.770782
800	20	3891.82	0.816202
900	20	4963.28	0.704834
1000	20	5940.54	0.607201

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

N	P	t(N, P)[ms]	q(N, P)
1000	100	749.127	1.026
1200	100	836.178	0.954351
1400	100	985.385	0.963981
1600	100	1143.17	0.978551
1800	100	1314.26	1
2000	100	1464.56	1.00292
2200	100	1635.37	1.01809
2400	100	1792.52	1.02292
2600	100	1945.69	1.02492
2800	100	2115.53	1.03479

N	P	t(N, P)[ms]	q(N, P)
1000	200	1466.58	0.985837
1200	200	1694.22	0.94905
1400	200	2019.36	0.969584
1600	200	2366.84	0.994373
1800	200	2677.76	1
2000	200	2988.11	1.00431
2200	200	3363.16	1.0276
2400	200	3669.1	1.02766
2600	200	4000.54	1.0343
2800	200	4311.35	1.03504

7.Wnioski

Oszacowanie złożoności metody optymalnej okazało się zadowalające – pomiary to potwierdzają z dokładnością +- 3% dla q(N,P).

Natomiast złożoność metody brutalnej okazała się zbyt skomplikowana (tj. zależna od wielu czynników). Podczas wyliczania q(N, P) pominąłem średnią długość ścieżki, która z kolei zależy od gęstości drzewa – a to jest wyznaczane losowo, przy generowaniu połączeń. Dla tych samych N i P, powtórzonych wiele razy, rozrzut czasowy wynosi nawet +-20% - co pokazuje jak ciężko wyznaczyć jednoznacznie złożoność i q(N,P) dla tego algorytmu.

Tabela dla oszacowania $O(N^2)$ rozwiązania brutalnego

N	P	t(N, P)[ms]	q(N, P)
1000	20	5747.54	0.895428
1200	20	9549.37	1.03315
1400	20	12649.5	1.00547
1600	20	15107.5	0.919392
1800	20	20796.8	1
2000	20	24099.6	0.93864
2200	20	34649.6	1.11533
2400	20	40761.4	1.10249
2600	20	47380.2	1.09194
2800	20	56723.6	1.12719