Cykl życia testów

1. Plan testów (ID: 1)

Wstęp

Testujemy dwie wersje algorytmu znajdowania najbliższego sąsiada – metodę siłową (BruteForce) oraz metodę opartą na KD-Drzewie (KD-tree) oraz struktury danych, na których oparte są oba podejścia.

Algorytm został opisany w książce "Algorytmy. Almanach" G.T. Heineman'a

Kod został pobrany z serwera FTP Helionu: ftp://ftp.helion.pl/przyklady/algalm.zip

Metody testowania

Testujemy metodami:

- białoskrzynkową, przy użyciu biblioteki JUnit, z miarą pokrycia kodu
- metodą czarnoskrzynkową za pomocą specjalnie przygotowanego graficznego interfejsu użytkownika oraz przygotowanych zbiorów danych

Korzystamy również z białoskrzynkowych testów dostarczonych przez autorów.

Zapewnienie jakości testów

Jakość testów zapewniamy poprzez wzajemne przeglądanie i weryfikowanie poprawności skonstruowania przypadków testowych oraz wyników testów. Przynajmniej dwóch członków zespołu musi zaakceptować każdy przypadek testowy oraz jego wynik.

Ograniczenia danych testowych

Testujemy działanie algorytmów dla zbiorów danych o wielkości do 100000 punktów. Jest to wielkość mieszcząca się w pamięci RAM nawet słabych maszyn.

Ograniczamy się również do testowania przy użyciu danych dwuwymiarowych.

Zbiory danych

Testujemy matematyczną poprawność znajdowania najbliższego sąsiada, zgodność algorytmów oraz zachowanie algorytmów dla szczególnych zbiorów danych:

- kilka punktów w tym samym miejscu
- brak punktów
- jeden punkt
- kilka równoodległych punktów

oraz dla typowych zbiorów punktów: ułożone w linii, ułożonych w kwadrat, okrąg w różnych skalach wielkości (odległości bardzo małe, średnie oraz bardzo duże)

Procedura przerwania/wznowienia testów

Napisane przez nas testy dotyczą gotowego i działającego algorytmu. Testy mogą być uruchamiane automatycznie i powinny za każdym razem zwracać te same wyniki. Ewentualne przerwanie oraz wznowienie procesu testowania nie będzie miało wpływu na wyniki testu.

Rezultaty testu

Rezultatem testu jest binarny wynik: znaleziono błędy lub nie znaleziono błędów oraz spis odnalezionych błędów

Środowisko testowe

Dowolny komputer z zainstalowaną maszyną wirtualną Java w wersji 1.6+, środowiskiem Netbeans 7.3+ oraz biblioteką JUnit.

Dla sprawdzenia miary pokrycia testów wykorzystano środowisko Eclipse oraz wtyczkę EclEmma Java Code Coverage 2.2.1

Odpowiedzialność w zespole
Zarządzanie zespołem testerów – GP

Projektowanie testów czarnoskrzynkowych – GP

Testy czarnoskrzynkowe – GP

Przeprowadzanie testów oraz raportowanie testów czarnoskrzynkowych – GP

Testy białoskrzynkowe wraz z miarą pokrycia kodu – DT

Testy białoskrzynkowe z wykorzystaniem obiektów pozornych – PW

Raportowanie testów białoskrzynkowych – DT i PW

Ryzyko w procesie testowania

Ryzyko w procesie testowania jest wysokie: posiadamy mały zespół, w związku z tym wzajemnie sprawdzamy swoje testy, brakuje kolejnego stopnia weryfikacji poprawności przeprowadzonych testów.

2. Projekty testów

Przypadek testowy (ID:C001):

Typ testu	Czarnoskrzynkowy
Przedmiot testu	Elementarna poprawność działania oraz zgodność algorytmów dla zbioru punktów zawierającego jeden punkt
Zakres testu	Algorytm znajdowania najbliższego sąsiada BruteForce i kdTree
Dane wejściowe	Plik 1_p, zawierający 1 punkt (0,0), punkt odniesienia (20,20)
Sposób wykonania	Plik z danymi wczytujemy do programu przeznaczonego do testów czarnoskrzynkowych, ustawiamy punkt odniesienia, test przeprowadzamy kolejno dla algorytmów BF i KD
Przewidywany wynik	KD: (0,0), BF: (0,0), wyniki zgodne

Przypadek testowy (ID:C002):

Typ testu	Czarnoskrzynkowy
Przedmiot testu	Poprawność działania oraz zgodność algorytmów dla punktów przekrywających się
Zakres testu	Algorytm znajdowania najbliższego sąsiada BruteForce i kdTree
Dane wejściowe	Plik two_p, zawierający 2 punkty (1,1) leżące na sobie, punkt odniesienia (1,1)
Sposób wykonania	Analogicznie do testu C1
Przewidywany wynik	KD: (1,1), BF: (1,1), wyniki zgodne

Przypadek testowy (ID:C003):

Typ testu	Czarnoskrzynkowy
Przedmiot testu	Poprawność działania oraz zgodność algorytmów dla pustego zbioru wejściowego
Zakres testu	Algorytm znajdowania najbliższego sąsiada BruteForce i kdTree
Dane wejściowe	Plik zero_p, nie zawierający żadnych punktów, punkt odniesienia (0,0)
Sposób wykonania	Analogicznie do testu C1
Przewidywany wynik	Wyjątek

Przypadek testowy (ID:C004):

Typ testu	Czarnoskrzynkowy
Przedmiot testu	Poprawność działania oraz zgodność algorytmów dla zbioru punktów zawierającego 4 punkty równoodległe od punktu odniesienia.
Zakres testu	Algorytm znajdowania najbliższego sąsiada BruteForce i kdTree
Dane wejściowe	Plik quad_p, zawierający punkty ułożone w kwadrat o wierzchołkach (0,0), (1,0), (1,1), (0,1), punkt odniesienia (0.5, 0.5)
Sposób wykonania	Analogicznie do testu C1
Przewidywany wynik	KD, BF: którykolwiek z boków kwadratu, wyniki zgodne

Przypadek testowy (ID:C005):

Typ testu	Czarnoskrzynkowy
Przedmiot testu	Poprawność działania oraz zgodność algorytmów dla zbioru punktów zawierającego 100 punktów, a punkt odniesienia ma współrzędne ujemne.
Zakres testu	Algorytm znajdowania najbliższego sąsiada BruteForce i kdTree
Dane wejściowe	Plik circle – 100 punktów ułożone w okrąg o promieniu 1.0 i środku w punkcie (0,0), punkt odniesienia (0.0, -0.9) – wewnątrz okręgu
Sposób wykonania	Analogicznie do testu C1
Przewidywany wynik	KD, BF: (0.0, -1.0), wyniki zgodne

Przypadek testowy (ID:C006):

Typ testu	Czarnoskrzynkowy
Przedmiot testu	Poprawność działania oraz zgodność algorytmów dla zbioru punktów zawierającego 100000 punktów
Zakres testu	Algorytm znajdowania najbliższego sąsiada BruteForce i kdTree
Dane wejściowe	Plik circle – 100000 punktów ułożone w okrąg o promieniu 1.0 i środku w punkcie (0,0), punkt odniesienia (0.0, 0.0) – w środku okręgu
Sposób wykonania	Analogicznie do testu C1
Przewidywany wynik	KD, BF: dowolny punkt na okręgu; wyniki zgodne

Kolejne testy czarnoskrzynkowe zostały zamieszczone w dokumencie dotyczącym testów czarnoskrzynkowych.

Przypadek testowy (ID:B001):

Typ testu	Białoskrzynkowy
Przedmiot testu	Testowanie metody equals klasy Hyperpoint na TwoDPoint
Zakres testu	Struktury pomocnicze Hyperpoint i TwoDPoint używane jako wierzchołki kd-Drzewa
Dane wejściowe	Hyperpoint hp o współrzędnych (2,8) oraz TwoDPoint hp2 o współrzędnych (2,8)
Przewidywany wynik	hp.equals(hp2) == True

Przypadek testowy (ID:B002):

Typ testu	Białoskrzynkowy
Przedmiot testu	Test metody nearest realizowanej na kd-Drzewie w przypadkach gdy korzeń jest pusty.
Zakres testu	Algorytm znajdowania najbliższego sąsiada oparty na kdTree
Dane wejściowe	Root=null
Przewidywany wynik	Metoda nearest zwraca null

Przypadek testowy (ID:B003):

Typ testu	Białoskrzynkowy
Przedmiot testu	Test metody nearest realizowanej na kd-Drzewie w przypadkach gdy target jest pusty.
Zakres testu	Algorytm znajdowania najbliższego sąsiada oparty na kdTree
Dane wejściowe	Target=null
Przewidywany wynik	Metoda nearest zwraca null

Przypadek testowy (ID:B004):

Typ testu	Białoskrzynkowy
Przedmiot testu	Działanie metody nearest w drzewie, które zawiera 100 punktów. Badamy tu działanie algorytmu dla kilku punktów i sprawdzamy, czy wynik jest zgodny z oczekiwanym.
Zakres testu	Algorytm znajdowania najbliższego sąsiada oparty na kdTree
Dane wejściowe	kdTree o wierzchołkach (10,0), (10,1) … (10,99), punkty odniesienia kolejno: (100, 10), (-100, 33), (100, -999)
Przewidywany wynik	Wyniki kolejno: (10,10), (10,33), (10,0)

Przypadek testowy (ID:B005):

Typ testu	Białoskrzynkowy
Przedmiot testu	Działanie metody nearest, gdy w drzewie znajdują się punkty o różnej wymiarowości
Zakres testu	Algorytm znajdowania najbliższego sąsiada oparty na kdTree
Dane wejściowe	kdTree o wierzchołkach (1,2,3), (1,2,3,4), punkt odniesienia (0,0)
Przewidywany wynik	Metoda nearest zwraca wyjątek mówiący o nieprawidłowej wymiarowości

Przypadek testowy (ID:B006):

Typ testu	Białoskrzynkowy
Przedmiot testu	Działanie metody nearest, gdy w drzewie znajdują się punkty o różnej wymiarowości
Zakres testu	Algorytm znajdowania najbliższego sąsiada oparty na kdTree
Dane wejściowe	kdTree o wierzchołkach (1,2,3), (1,2,3,4), punkt odniesienia (1,2,3,4)
Przewidywany wynik	Metoda nearest zwraca wyjątek mówiący o nieprawidłowej wymiarowości

Kolejne testy białoskrzynkowe zostały zamieszczone w dokumencie dotyczącym testów białoskrzynkowych.

3. Realizacja testów

Program umożliwiający wykonywanie testów czarnoskrzynkowych oraz zbiory danych zostały zaprojektowane i wykonane przez GP.

Testy białoskrzynkowe z miarą pokrycia kodu zostały zaprojektowane i napisane przez DT.

Dokładniejsze informacje na temat realizacji poszczególnych rodzajów testów znajdują się w dokumentach dotyczących tych testów.

4. Wykonanie testów

Przypadki testowe czarnoskrzynkowe Cxxx zostały wykonane przez GP, a następnie zweryfikowane przez DT poprzez ponowne ich wykonanie.

Przypadki testowe białoskrzynkowe Bxxx zostały wykonane przez DT, a następnie zweryfikowane przez GP poprzez ponowne ich wykonanie.

Testy były zatem przeprowadzane co najmniej dwukrotnie oraz były wykonywane na różnych maszynach i w różnych środowiskach. GP przeprowadzał testy pod systemem Ubuntu 12.10, DT przeprowadzała testy pod systemem Windows.

Dokładniejsze informacje na temat wykonania poszczególnych rodzajów testów znajdują się w dokumentach dotyczących tych testów.

5. Ocena rezultatu testów

Testy czarnoskrzynkowe

Przypadek testowy (ID:C001):

Wyniki zgodne z oczekiwaniami

Przypadek testowy (ID:C002):

Wyniki zgodne z oczekiwaniami

Przypadek testowy (ID:C003):

KD: Wyjątek, BF: null, wyniki niezgodne między wersjami algorytmu

Przypadek testowy (ID:C004):

Wyniki uzyskane metodą BF i KD prawdziwe, ale niezgodne między wersjami algorytmu

Przypadek testowy (ID:C005):

Wyniki zgodne z oczekiwaniami

Przypadek testowy (ID:C006):

Przepełnienie stosu dla algorytmu opartego na KD-drzewie – wynik niezgodny z oczekiwaniami.

Testy białoskrzynkowe

Przypadek testowy (ID:B001):

Wyniki zgodne z oczekiwaniami

Przypadek testowy (ID:B002):

Wyniki zgodne z oczekiwaniami

Przypadek testowy (ID:B003):

Wyniki zgodne z oczekiwaniami

Przypadek testowy (ID:B004):

Wyniki zgodne z oczekiwaniami

Przypadek testowy (ID:B005):

Wyniki zgodne z oczekiwaniami

Przypadek testowy (ID:B006):

Wynik (1,2,3,4) niezgodny z oczekiwaniami. Dla drzewa zawierającego punkty o różnej wymiarowości nadal jest przeprowadzane przeszukiwanie.

Pozostałe wyniki oraz dokładniejsze opisy przeprowadzanych testów znajdują się w załącznikach dotyczących testów białoskrzynkowych i czarnoskrzynkowych.

Zaistniałe incydenty:

1. W ogólności nie można się spodziewać, że algorytm znajdowania najbliższego sąsiada przy użyciu kd-Drzewa zwróci ten sam wynik co metoda Brute Force. W przypadkach szczególnych, kiedy istnieje więcej niż jeden najbliższy sąsiad, algorytmy mogą zwracać różne (ale poprawne) wyniki.
2. Algorytm oparty na kd-Drzewie zachowuje się w sposób nieoczekiwany dla punktów o różnej wymiarowości. Ta własność nie została jednak dokładnie przetestowana, ponieważ mieliśmy się skupić na przypadkach dwuwymiarowych.
3. Algorytm oparty na KD-drzewie nie radzi sobie z dużymi zbiorami danych. Dla zbioru danych liczącego 100000 punktów, występuje przepełnienie stosu.

6. Wnioski

Proces testowania pozwolił nam lepiej zrozumieć zachowanie algorytmu w standardowych i niestandardowych sytuacjach.

Pomimo blisko 100% pokrycia kodu testami białoskrzynkowymi oraz licznych testów czarnoskrzynkowych, nie możemy jednak stwierdzić z całą pewnością, że algorytm zadziała prawidłowo dla wszystkich przypadków. Wydzielamy jednak pewien podzbiór sytuacji, dla których algorytm działa poprawnie. W razie awarii znacznie zawęża to obszar poszukiwań oraz czas potrzebny na znalezienie i naprawę błędu.