Setřídění posloupnosti metodou Select Sort

Zadání

Úkolem tohoto programu je seřadit posloupnost reálných čísel pomocí metody Select Sort. Vstupní data jsou zadána v textovém souboru. Výstup bude nahrán do nového textového souboru. Další požadavek na program je, aby uživatel zadal vstupní parametr pro seřazení řady vzestupně či sestupně.

Metoda Select Sort

Jedná se o řadící algoritmus s časovou náročností $O(N^2)$. Algoritmus pracuje s tím, že v nahrané posloupnosti najde minimum, které pak prohodí s prvním prvkem. V dalším kroku program opět hledá minimum, ale nyní již vynechává první prvek. Po nalezení prohodí opět pozici minima a tentokrát 2. prvku. Algoritmus takto postupně projde celou posloupností až je seřazena. Pro řazení sestupně postupuje stejně, jen hledá maximum a ne minimum.

Pro větší datové soubory se tato metoda nevyužívá. Pro takové případy je spíše využita metoda Řazení haldou nebo Řazení slučováním, které mají náročnost O(NlogN).

Program

Program je rozdělen do 5 částí. První část je zpracování vstupního souboru ve formátu *.txt* a nahrání jeho obsahu do datové struktury *list.* Pomocí funkce *open* je nahrán textový soubor, dále jsou všechny jeho řádky nahrány do listu názvem *alist*. Každý řádek je reprezentován jako jedna položka listu ve formátu *float*. Program zde počítá také s některými problémy, které mohou nastat a va takovém případě vrátí chybovou hlášku a ukončí se. Takové případy jsou že soubor vstupních dat nebyl nalezen, nebo program není oprávněn tento soubor otevřít.

Další funkce programu *vzestup(alist)* má jeden vstupní parametr, a to list nesoucí posloupnost reálných čísel. Funkce využívá dvou do sebe vnořených cyklů, kde zjišťuje, na jaké pozici se nachází minimum aktuální neseřazené posloupnosti. Když jej nalezne, tak se prvky na pozici minima a aktuální iterované pozici prohodí. Výstupem této funkce je vzestupně seřazená posloupnost *alist* ve formátu *list.*

Funkce *sestup(alist)* pracuje na stejném principu, ale vrací *alist* seřazený sestupně. Jaká z těchto dvou funkcí se má pro seřazení posloupnosti použít zajišťuje funkce *metoda(alist)*, která se uživatele zeptá jak si přeje posloupnost seřadit. Uživatel zadá „VZ“ (pro řazení vzestupně), nebo „SE“ (pro řazení sestupně). Dále spustí příslušnou funkci.

Pro zajištění výstupu seřazené posloupnosti ve formátu *.txt* slouží poslední funkce programu *vystup(alist),* která má vstupní parametr již seřazenou posloupnost a zapíše ji do textového souboru, že každý prvek posloupnosti je na jednom řádku.

Alternativní řešení

Vzhledem k tomu, že se jedná o vcelku jednoduchou metodu, tak o jiných metodách nebylo moc uvažováno.

Vstupní data

Neseřazená posloupnost je v textovém dokumentu s názvem *vstup_posloupnost.txt.* Data jsou zde tak, že na každém řádku je jedno reálné číslo posloupnosti. Desetinné místo čísel je odděleno tečkou (př. 10.02).

Výstupní data

Výstup je v textovém souboru *vystup_posloupnost.txt* a data jsou zde uložena, že na každém řádku je jeden prvek posloupnosti a desetinná místa čísel jsou oddělena tečkou.

Závěr

Takovýto typ úlohy mi připadal za 2 body adekvátní. Procvičí se zde práce s cykly i nahrávání dat ze souboru a jejich opětovné uložení. Dále je zajímavé se podívat kolika způsoby lze řazení posloupnosti řešit.

Ověření, zda bod leží uvnitř konvexního mnohoúhelníku

Zadání

Druhým úkolem bylo sestrojit program, který nahraje body, které definuj vrcholy konvexního mnohoúhelníku, a zkoumaný bod z textového souboru a pomocí metody Ray Crossing, nebo Winding Nuber zjistí, zda bod leží uvnitř polygonu. Výsledek pak vrátí v textovém dokumentu.

Winding number

Pro určení polohy body byl zvolen algoritmus Winding number. Metoda je založena na předpokladu, že když máme polygon P s vrcholy p_i a pozorovatel stojí v bodě q a chce vidět všechny p_i, tak když je uvnitř polygonu musí se otočit o 2π, když stojí vně, tak se otočí o úhel menší než 2π. To znamená, že suma všech úhlů p_i, q, p_{i+1} musí být rovna 2π pro q uvnitř polygonu. Tento výpočet lze aplikovat jak pro konvexní, tak nekonvexní polygony. Jedná se však o časově náročnou metodu, protože je zde pro zjištění kvadrantů úhlů zapotřebí vypočítat inverzní trigonometrickou funkci tan2.

Pro konvexní polygon lze tento algoritmus zjednodušit a pro zjištění polohy bodu vůči polygonu nám stačí vypočítat vektorový součin $w = u \times v$, kde $u = p_{i+1} - p_i$, $v = q - p_i$. Když je bod uvnitř, tak všechny součiny vyjdou se stejným znaménkem. Když počítáme protisměru hodinových ručiček, tak pro bod ležící vpravo od přímky definující hranu polygonu vyjde součin větší než 0, pro bod nalevo méně než 0 a pro bod na přímce 0.

Časová náročnost této metody je O(N).

Výběr algoritmu

V zadání je uvedeno, že lze pro výpočet použít také algoritmus Ray crossing. Tento algoritmus pracuje na principu, že ze zkoumaného bodu vyšleme paprsek jedním směrem (většinou doprava) a pozorujeme, kolikrát protne hranu polygonu. U konvexního polygonu ji protne 1, když je uvnitř a 2 nebo 0, když je vně. Tento algoritmus je sice časově méně

náročný, ale je zde těžší ověřit singulární případy jako třeba bod ležící ve vrcholu polygonu nebo paprsek je kolineární se stranou polygonu. Kvůli tomuto byl vybrán algoritmus winding number, který s těmito singularitami nemá problém.

Program

Program obsahuje 2 třídy. První, která se jmenuje *Points,* zajišťuje ukládání a práci s body. Ukládá do sebe souřadnice x a y bodu. Dále zajišťuje volání jednotlivých souřadnic bodů a jejich reprezentaci a případnou editaci pomocí setterů. Druhá třída s názvem *Polygon* nahraje list bodů uložených ve formátu třídy *Point,* který pak definuje polygon. Body jsou v listu seřazeny za sebou pravotočivě.

Třída *Polygon* obsahuje 3 další funkce. První funkce má za úkol vytvořit pomocí cyklu vytvořit dvojce bodů, které definují jednotlivé hrany polygonu a uložit je do listu. Funkce se nazývá *edges().* Výstupem této funkce je list dvojic bodů definující hrany polygonu.

Další funkce s názvem *konvex()* zajišťuje zda je zadaný polygon konvexní a lze na něj tedy uplatnit zjednodušenou formu algoritmu winding number. Funkce načte vždy trojici sousedních bodů a spočte úhle, který uloží do listu *uhly*. Takto projede postupně úhly u všech vrcholů polygonu. Nakonec se funkce podívá do listu *uhly*, a když obsahuje úhle větší než 180 °, tak polygon není konvexní a funkce zastaví program a vrátí chybovou hlášku.

Poslední funkce třídy *Polygon* se jmenuje *contains(point)*. Do této funkce vstupuje ještě jedna vnější proměnná ve formátu třídy *Point*, která definuje zkoumaný bod. Do funkce vstupuje jako *point.* Dále pracuje s listem hran vytvořených pomocí funkce *edges().* Pomocí těchto proměnných se vypočte vektorový součin a jeho výsledek je uložen do listu *list_results*. Když v listu jsou všechny výsledky menší nebo rovny nule, tak se bod nachází uvnitř a funkce vrátí hodnotu *True*. Je-li nějaká hodnota větší než nula, znamená to, že bod je mimo polygon a funkce vrací hodnotu *False*.

Vstupní data jsou načítána z textového souboru, ve kterém je na každém řádku jeden bod. Na prvním řádku je zkoumaný bod a na dalších body definující polygon. První bod je uložen jako *bod* ve formátu *Point*. Zbytek bodů je nahrán do listu *alist*, kde každá položka obsahuje vrcholový bod polygonu ve formátu *Point*. Když vstupní soubor není nalezen, nebo program nemá právo jej číst, je program ukončen a vrací chybovou hlášku. Při vstupu jsou jednotlivé souřadnice rozděleny a z formátu *string* uloženy do *formátu int.*

Nakonec je vytvořen výstupní soubor ve formátu *.txt,* který obsahuje informaci o zjišťovaném bodu. Když funkce *contains(point)* vrátí hodnotu *True*, zapíše do textového souboru, že bod je uvnitř, když hodnotu *False,* zapíše, že je mimo.

Alternativní řešení

Původně jsem uvažoval o řešení pomocí algoritmu Ray crossing, ale narazil jsem na problém se singulárními případy a připadalo mi jednodušší využít algoritmus winding number než je ošetřovat. Dále jsem původně přemýšlel o zpracovávání vstupních jen jako listů čísel, ale nakonec jsem se uchýlil k využití prcku objektově orientovaného programování a data jsem nahrával do tříd.

Vstupní data

Data bodů, jak definující polygon, tak kritický bod, jsou uloženy v textovém souboru s názvem *vstup_polygon.txt*. Data jsou zde zapsána, že na prvním řádku jsou souřadnice kritického bodu a na dalších řádcích body definující polygon. Každý bod je na novém řádku a body vrcholů polygonu jsou seřazeny postupně po směru hodinových ručiček. Souřadnice jednotlivých bodů jsou ve formátu souřadnice x „mezera“ souřadnice y. Souřadnice x a y jsou celá čísla (př. 5 -4).

Výstupní data

Výstup je uložen v textovém souboru *vystup_polygon.txt* a obsahuje textový řetězec s výsledkem.

Závěr

Ze začátku jsem měl problém s prací se třídami, a hlavně kontrolou jejich výstupů, ale pomocí reprezentace, jsem si to nakonec mohl vždy ověřit, že funkce dělají to, co mají. Ze začátku mi připadala úloha za tři body docela náročná, ale po zjištění, že algoritmy se dají pro konvexní polygon zjednodušit už to nebyl takový problém.