Геометрията в състезателното програмиране
Част II

Автори
Христо Борисов
Иван Тодоров

24 април 2009 г.

Съдържание

1 Алгоритми **2**
1.1 Лице на многоъгълник 2
1.2 Изпъкнала обвивка 2
1.3 Взаимно положение на точка и обекти в равнината 5
1.4 2D дървета 7
1.5 Метод на помитащата(сканираща) линия 9
1.6 Алгоритъм на Bentley-Ottmann 9
1.7 Диаграми на Вороной 10
1.8 Триангулация на Делоне 11

2 Референции **12**

1 Алгоритми

1.1 Лице на многоъгълник

Лицето на изпъкнал многоъгълник може да се намери, като многоъгълникът се раздели на триъгълници и се съберат лицата на отделните триъгълници. Ако ни е даден изпъкналият многоъгълник $A_1A_2...A_n$, то лицето е абсолютната стойност на израза:

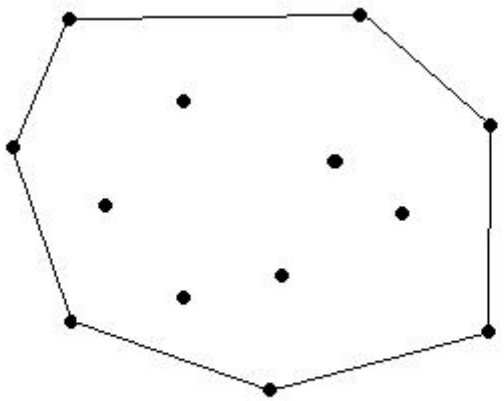
$$\frac{1}{2}\left(\begin{vmatrix} x_1 & y_1 \\ x_2 & y_2 \end{vmatrix} + \begin{vmatrix} x_2 & y_2 \\ x_3 & y_3 \end{vmatrix} + \ ... \ + \begin{vmatrix} x_n & y_n \\ x_1 & y_1 \end{vmatrix} \right)$$

Ако разделим произволен N-ъгълник $A_1A_2..A_N$ на триъгълници $\Delta A_1A_2A_3$, $\Delta A_1A_3A_4$, $\Delta A_1A_4A_5$, ..., $\Delta A_1A_{N-1}A_N$ и точно в този порядък им сметнем ориентираните лица чрез *Cross Product*, ще компенсираме неизпъкналите участъци и ще получим точното лице:

$$S_{A_1A_2..A_N} = \frac{1}{2}.|\vec{A_1A_2}\text{x}\vec{A_1A_3} + \vec{A_1A_3}\text{x}\vec{A_1A_4} + ... + \vec{A_1A_{N-1}}\text{x}\vec{A_1A_N}|$$

1.2 Изпъкнала обвивка

Изпъкналата обвивка за дадено множество от точки P е изпъкнал многоъгълник с минимално лице, който съдържа изцяло множеството P. От дефиницията следва, че върховете на изпъкналия многоъгълник са точки от множеството P.



Съществуват различни алгоритми за намиране на изпъкналата обвивка, които като цяло не се различават много по идея или сложност.

Наивен подход

1. Избираме от множеството точки тази, която има най-малка X координата. Ако има няколко такива, избираме тази с най-малка Y координата. Нека тази точка е S. S със сигурност е от обвивката.

2. Взимаме произволна точка от останалите (нека тази точка е T).

3. Обхождаме останалите точки P_i и ако текущата точка е от дясно на правата ST, P_i става T (T:=P_i).

4. Така намерената права ST е от обвивката, защото от дясно на нея няма други точки.

5. Повтаряме стъпки 1-4, като този път избора на S е различен - избираме за S последно намереното T.

6. Алгоритъмът продължава докато не получим затворен полигон т.е. T съвпадне с първоначално избраната точка S, тъй като намерените точки от обвивката са в порядъка, в който се срещат по изпъкналия полигон.

Сложността на алгоритъма е $O(n^2)$, където n е броят на точките.

Съществуват алгоритми, които намират обвивката доста по-бързо. Един от най-бързите и не особено труден за реализация е тъй наречения ***Graham Scan***. Той се състои в следното:

1. Избираме точката, която има най-малка X координата. Нека тя е S.

2. Сортираме останалите точки спрямо S, така че точка P_{i+1} да е от ляво на правата SP_i . Това се реализира като при сортировката разменяме местата на P_i и P_j ако ориентираното лице на триъгълника SP_iP_j е отрицателно. След размяната това лице ще е положително.

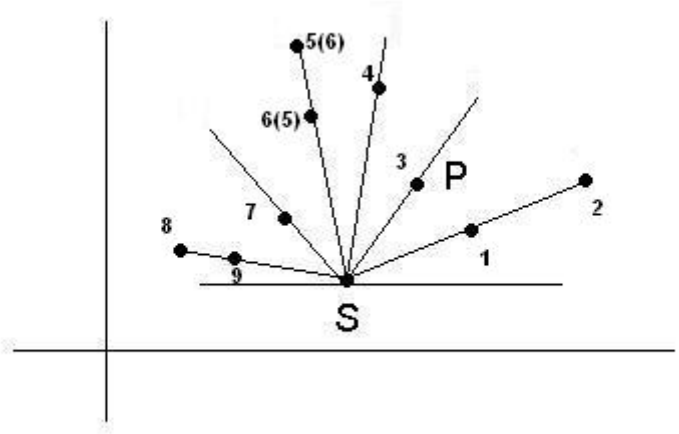
3. Добавяме S и първата точка от сортираната последователност в множеството на точките от обвивката (тези две точки винаги са от обвивката). Най-удобно е това множество да се реализира със стек, така че винаги да знаем точките на върха му. Точките в стека във всеки един момент ще описват изпъкнал контур, който не е задължително да е от обвивката. В края на алгоритъма стека ще съдържа всички точки от обвивката в порядъка, в който се срещат по изпъкналия полигон (обвивката), с начална точка S и посока обратна на часовниковата стрелка.

4. Нека двете точки на върха на стека са са P_{end} и P_{end-1}.

5. Взимаме поредната точка от сортираните (нека тя е A).
   - Ако е от положителна страна на правата $P_{end-1}P_{end}$, то тя запазва изпъкналостта. Добавяме я на върха на стека.
   - Ако е от отрицателната страна на правата $P_{end-1}P_{end}$, то тя нарушава изпъкналостта и за да поправим това, изтриваме точката на върха на стека, докато изпъкналостта продължава да бъде нарушавана. След това добавяме A в стека.

6. След обхождането на всички точки, точките от обвивката ще се намират в стека.

Ако имаме точки, които лежат на една права с т.S, то имаме основание да смятаме, че този алгоритъм няма да работи. Първото нещо, за което се сещаме, след като сме се сблъскали с този проблем, е при сортирането ако срещнем две точки P_i и P_j, такива че S,P_i,P_j са колинеарни, по-близката до S точка да е първа в сортираната последователност. Това обаче освен излишни операции, в някой случаи се оказва и водещо до грешно решение. Ако си дадем сметка за действието на алгоритъма ще ни стане ясно,че няма смисъл да сравняваме всички двойки точки, лежащи на една права с началната т.S, а само за точките в началото и в края на сортираната последователност (точки 1,2 и точки 8,9 на фигурата по долу). Точките, които се намират на една права с началната S и първата точка от сортираната последователност (точки 1,2 на фигурата по долу) , трябва да се сортират в нарастващ ред спрямо разстоянието от тях до S, а точките, лежащи на една права с S и последната точка от сортираната последователност (точки 8,9 от фигурата по долу) - в намаляващ ред. По този начин няма да изпуснем да намерим точки, които са от обвивката.



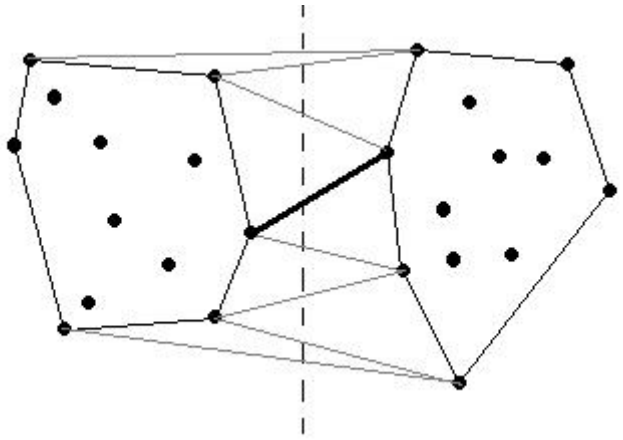
Сложността на алгоритъма всъщност зависи от сложността на сортирането тъй като след него всяка точка е добавена и премахната най-много веднъж от стека, което води до линейна амортизирана сложност. При добра реализация на сортирането алгоритъма има сложност $O(n \log n)$. Съществуват много други алгоритми за намиране на изпъкнала обвивка, но тъй като е доказано, че самият проблем се свързва със сортиране на точките, то сложността на кой да е алгоритъм не би могла да бъде по-малка от $O(n \log n)$.

Подход "разделяй и владей"

Във връзка с развитието на мултипроцесорните системи и мултинишковото програмиране е важно да се познават техники, които могат да се приложат на такива системи. Един съвсем тривиален подход, използващ техниката *"разделяй и владей"*, е да разделим множеството от точки на две подмножества, всяко от които се намира в една полуравнина спрямо произволно избрана разделяща права, да се намери обвивката на всяко от тези множества и след това двете обвивки да се обединят в една. Самото обединяване е сравнително лесно за реализация:

- Свързваме с две отсечки най-лявата точка от дясното подмножество и най-дясната точка от лявото подмножество. Ако означим обвивката на лявото подмножество с H_1, обвивката на дясното с H_2 и отсечките с s_1 и s_2, то получаваме обединението на тези елементи, като ги свържем в следния порядък: $H_1 \cup s_1 \cup H_2 \cup s_2$.

- Очевидно, след обединяването, около краищата на отсечките могат да се получат неизпъкнали участъци. Затова изтриваме всички точки, които нарушават изпъкналостта. За да се реализира бързо операцията изтриване, подходяща за иползване структура е двойно свързаният списък. Също така не е необходимо да се обхожда целия списък, за да се намерят точките, нарушаващи изпъкналостта, а е достатъчно да се прави проверка за краищата на отсечките s_1 и s_2, като при всяко изтриване тези краищата се променят (виж долната фигура).



Разбира се, за намирането на обвивката на всяко от двете подмножества може да се приложи същия метод, като всяко от тях се раздели на още две подмножества. Можем да продължаваме така рекурсивно, като за дъно на рекурсията имаме следните съображения:

- Обвивката на точка е самата точка.

- Обвивката на две точки са два противоположни вектора, определени от точките.

- Обвивката на три точки е триъгълник с върхове тези точки.

1.3 Взаимно положение на точка и обекти в равнината

Основен проблем в изчислителната геометрия, който се намира в много практически приложения, е да се определи дали дадена точка лежи в даден геометричен обект(фигура). Ще разгледаме някой от основните положения:

- Взаимно положение на точка и триъгълник.

  Произволна точка Oлежи във вътрешността на ΔABC тогава и само тогава,

когато въпросната точка се намира от една и съща страна на векторите $\vec{AB}, \vec{BC}, \vec{CA}$ или на векторите $\vec{AC}, \vec{CB}, \vec{BA}$. Доказателството на този факт е очевидно: ако точката се намира от обратната страна на някоя от страните, то тя е в различна полуравнина спрямо триъгълника и няма как да е във вътрешността му. Имплементацията изплозва *Cross Product* за да определи положението на точката спрямо всяка от страните. Друг възможен начин за осъществяване на проверката е да сравним сумата от лицата $\Delta AOB + \Delta BOC + \Delta COA$ с лицето на ΔABC. Ако са равни, то точка O е във вътрешността или по контура на ΔABC. В противен случай точка O е извън ΔABC.

- Взаимно положение на точка и изпъкнал многоъгълник.

  Горното твърдение може да се обобщи за всеки изпъкнал N-ъгълник.

- Взаимно положение на точка и общ многоъгълник.
  Доказателствата на горните твърдения се оповава на факта, че когато многоъгълник е изпъкнал, всяка следваща страна се намира в една полуравнина с многоъгълника, като имаме предходната страна за секуща. Когато става въпрос за общ многоъгълник, твърдението не е вярно.
  Подходът за решаване на този проблем е следният:

  1. Строим отсечка с краиша въпросната точка и точка, за която сме сигурни, че е извън многоъгълника (например точка с координати, по-големи от координатите на всички точки) или един вид лъч с начало въпросната точка и произволна посока (за улеснение често се избира лъч, перпендикулярен на абсцисата или на ординатата).
  2. Ако този лъч пресича четен брой страни на многоъгълника, то въпросната точка е външна.
  3. Ако този лъч пресича нечетен брой страни на многоъгълника, то въпросната точката е вътрешна.
  4. Ако лъча минава през връх на многоъгълника броим това за една пресечена страна, когато предишната и следващата точка на този връх са от различни страни на лъча и за две пресечени страни, когато предишната и следващата точка на върха са от една и съща страна на лъча.
  5. Ако някоя от страните на многоъгълника лежи на лъча, то броим това за една пресечна точка ако предишната и следващата страна са от различни страни на лъча. В противен случай го броим за 2 пресечни точки.

  Сами можете да се уверите в това като разгледате няколко прости примера и забележите, че ако началото на лъча е вътре в многоъгълника, то за да излезе навън трябва да пресече нечетен брой страни.

- Взаимно положение на точка и кръг.

  Окръжност с център O и радиус r е геометрично място на точки на разстояние

r от O. Кръгът, определен от тази окръжност, е геометрично място на точки на растояние по-малко или равно на r от O.
Следователно за дадена точка A:

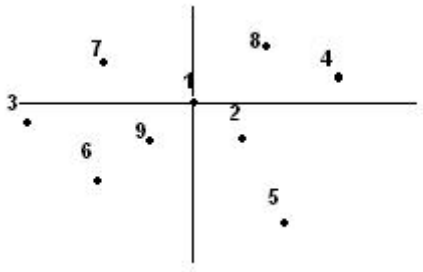
1. A лежи извън кръга тогава и само тогава, когато $OA > r$.

2. A лежи на окръжността тогава и само тогава, когато $OA = r$.

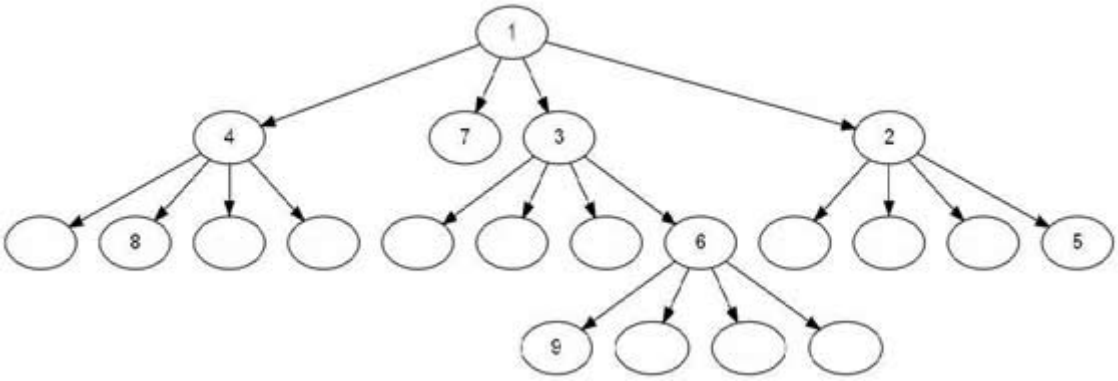
3. A лежи вътре в кръга тогава и само тогава, когато $AO \leq r$.

1.4 2D дървета

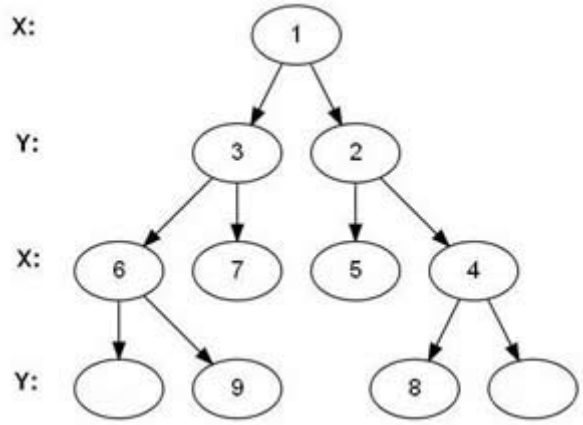
2D дърветата представляват ефективен метод за търсене в равнината. Всеки връх в дървото има 4 връзки към поддървета, като всяко от тях съдържа точките в съответния квадрант спрямо точката в текущия връх. Дефиницията е рекурсивна за поддърветата. Необходимо е също да се дефинира в кой квадрант лежат точките които се намират на границата между два квадранта. Като цяло тези дървета са аналог на двоичните дървета за търсене на числа по числовата ос, но тук става въпрос за точки в равнината (пространтво с едно измерение повече). Като аналог на двоичните дървета, то наследява всички негови хубави и лоши свойства. Операциите за вмъкване и претърсване имат логаритмична сложност, но в много случаи дървото се нуждае от балансиране. Също така при повече празни връзки, ще имаме доста излишно използвана памет. Има възможност това дърво да се представи като двуично по следния начин: ако въведем ниво за всеки връх в дървото така че то е равно на разстоянието от него до корена, то за всеки връх с четно ниво точките с по-голяма x-координата стоят от дясно, а с по-малка от ляво и за всеки връх с нечетно ниво точките с по-голяма y-координата стоят от дясно, а тези с по-малка - отляво. Тоест това е добре познатото ни двуично дърво, просто на четните нива сравняваме по x-координата, а на нечетните нива - по y-координата. Това дърво изглежда по-компактно, представено в паметта.



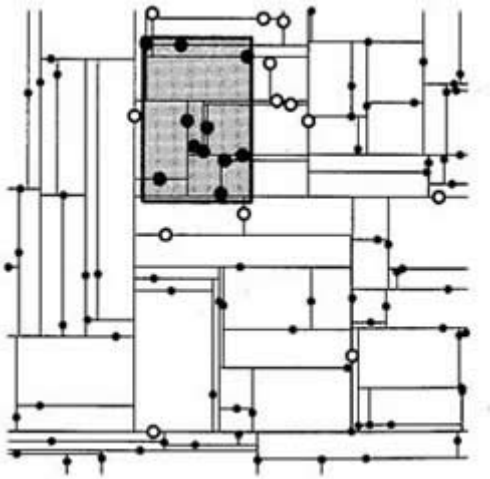
2D дърво за даденото множество точки:



Двоично дърво за даденото множество точки:



Тези структури са подходящи за търсене на точки в даден правоъгълен сегмент от равнината. Търсенето се извършва както при двоичните дървета, като се спазват въведените по-горе критерии при сравнение. Ето една визуализация на точки в равнината, изградени в дърво, чрез разделяне на полуравнини при всяка точка по x или y в зависимост от четността на нивата в дървото, както и заявка за търсене в сегмент:



Дефинициите на тези структури са аналогични и за пространства с повече измерения т.е. 2D-tree можем да обобщим до KD-tree. За големи K обаче, не е подходящо да се

използва дърво, което разделя пространството на квадранти спрямо дадена точка, тъй като всяко K-измерно пространство се разделя от дадена точка на 2^K квадранта.

1.5 Метод на помитащата(сканираща) линия

Това е метод, при който мислена линия (наречена помитаща линия или *sweep line*) се движи през всички обекти в равнината като ги "сканира"и върши определени действия с тях. Известен проблем, за решаването на който се използва този метод, е проблема за намиране на пресечни точки на отсечки в равнината. За да илюстрираме метода ще се спрем на по-простият вариант - когато всяка от отсечките е успоредна на някоя от осите. Въвеждаме помитаща линия, която е успоредна на $Oy^{\rightarrow}$ и се движи от ляво надясно (по $Ox^{\rightarrow}$. Тогава:

1. Ако линията срещне начална точка на отсечка, успоредна на $Ox^{\rightarrow}$, добавяме y-координатата в някакво множество P.

2. Ако линията срещне крайна точка на отсечка, успоредна на $Ox^{\rightarrow}$, премахваме y-координатата от множеството P.

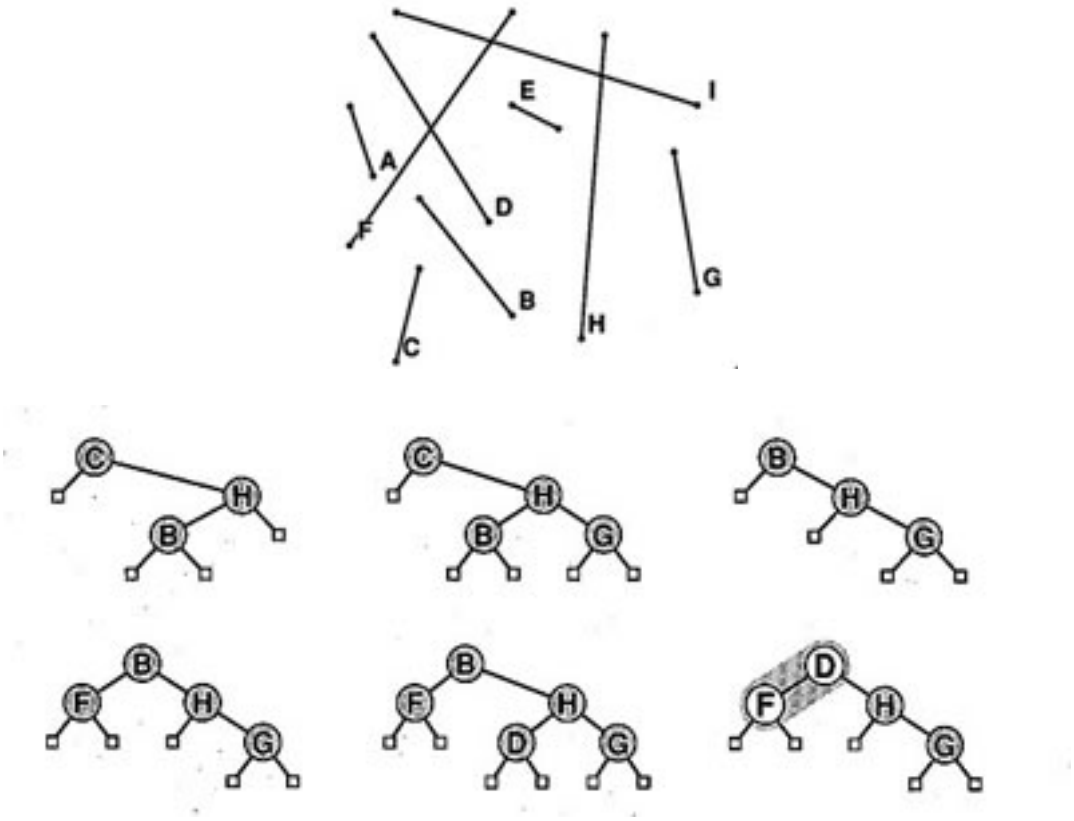
3. Ако линията срещне отсечка, успоредна на $Oy^{\rightarrow}$, то броят на пресечните точки е броя на елемтите в множеството P в интервала $[y_1,y_2]$, където y_1 и y_2 са y-координатите на двете крайни точки на отсечката, която сканираме в момента.

Разбира се, помитащата линия не се движи реално, а условно. Достатъчно е да премине само през крайните точки на отсечките, като за да спестим време, сортираме всички точки в нарастващ ред спрямо тяхната x-координата.

1.6 Алгоритъм на Bentley-Ottmann

За обобщения вариант на алгоритъма за намиране на пресечни точки на отсечки (известен като алгоритъм на Бентли-Отман), когато отсечките не са задължително успоредни на осите, се използва по специална структура от данни (известна като x-tree) - двуично дърво, за което важи правилото, че в лявото поддърво се намират всички отсечки, които се намират от ляво на текущата или текущата се намира от дясно на тях, а в дясното поддърво са всички отсечки, които се намират в дясно на текущата или текущата се намира в ляво от тях. След това по същия начин чрез помитаща линия сканираме всички точки и вмъкваме или изтриваме отсечка от дървото в зависимост от това дали е начална или крайна сканираната точка. Когато за дадени 2 отсечки не може да се определи дали едната трябва да е в лявото или дясното поддърво на другата, то те се пресичат. Тогава трябва вмъкваната отсечка да се раздели на 2 нови отсечки за да може да продължи работата си алгоритъма (т.е. да "счупим"вмъкваната отсечка точно в пресечната й точка с другата отсечка, за да получим две нови отсечки, които вече можем да определим в кое поддърво са). Проблема с "отчупените"отсечки, които след като са били отчупени трябва да изчакат помитащата линия да ги достигне и да се опита да ги вмъкне още веднъж в дървото, се решава с друга структура - приоритетна опашка. В нея краищата на

отсечки с по-малка x-координата са с по-голям приоритет. След като получим нови отсечки и вмъкнем техните краища в опашката, елементите на опашката ще бъдат преподредени достатъчно бързо, за да не се отрази значително на цялостното време на работа на програмата. Важно е да се отбележи, че критереия за сравнение в дървото не е транзитивен, т.е. ако една отсечка L от ляво на T, а R е от дясно на T, то не винаги е валидно че L е от ляво на R. Затова при изтриване от дървото трябва да се прави проверка за пресичане на новите съседни върхове.

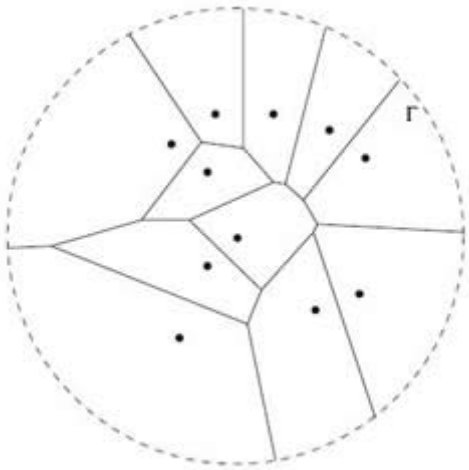


1.7 Диаграми на Вороной

За дадено множество от точки P в равнината диаграмите на Вороной представляват разделяне на равнината на области така че:

1. Във всяка област има точно една точка от дадените.

2. Всяка точка от коя да е област е по-близо до точката от P, намиращата се в същата област, от колкото до всяка друга точка от P.



За да построим диаграмите на Вороной стоим областите една по една. Област, съдържаща дадена точка, се образува като построим симетралите на всички отсечки с краища дадената точка и някоя от останалите точки и вземем сечението на полуравнините съдържащи дадената точка. Това е наивният алгоритъм за построяването на диаграмите. Алгоритъм със сложност $O(N \log N)$ е бил измислен през 1986г. от Стивън Форчън. Състои се от помитаща линия, следвана от "плажна ивица"."Плажната ивица"е представена като множество от параболи, които с прогреса на помитащата линия очертават страните на диаграмите на Вороной. Имплементацията му обаче никак не е тривиална и е отвъд обхвата на тази статия.

Диаграмите на Вороной се използват при локализация и отговаряне за заявки от типа "най-близък съсед при които търсим най-близкия обект до зададена точка (не задължително от P). Ако обектите са ни точките от множеството P, то е достатъчно да определим в коя област се намира зададената точка. Тогава най близкият обект до нея е точката от P която принадлежи на същата област. Като намерим диаграмите на Вороной, можем да намерим най-големия "празен"кръг в равнината (т.е. да намерим точката, за която най-близката точка от P е възможно най-далечна). При планиране на движението на роботи, диаграмите на Вороной се използват за намиране на чист маршрут (например ако точките са препятствия, то най-безопасният маршрут ще бъде този по страните на диаграмите, защото те са максимално отдалечени от заплашващите ни обекти).

1.8 Триангулация на Делоне

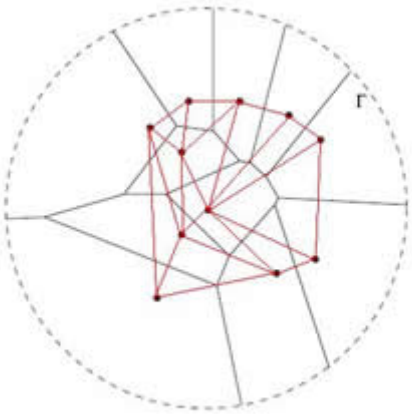
Триангулацията на Делоне за множеството точки P е триангулация DT(P), така че няма точка от P, вътрешна за кой да е от триъгълниците на DT(P) и всички окръжности, описани около триъгълниците на DT(P), са *празни*. Според дефиницията на Делоне, описаната окръжност около триъгълник, съставен от три точки от началното множество точки е *празна* ако не съдържа други точки освен трите,съставящи триъгълника.
За множество колинеарни точки не е дефинирана триангулация на Делоне(всъшност не е дефинирана никаква триангулация).
За 4 точки, лежащи на една окръжност не съществува уникална триангулация

на Делоне - ясно е че и двете триангулации разделящи четириъгълника на два триъгълника удовлетворяват условията на Делоне.
Ако за множеството от точки построим диаграмите на Вороной и свържем всяка точка от P с точките в съседните области, ще получим триангулация на Делоне (под съседна област разбираме област, чийто контур има поне една обща точка с дадената). Разбира се за по-големи множества точки съществува повече от една триангулация DT(P). Триангулацията на Делоне има няколко свойства:

1. Обединението на всички триъгълници ни дава изпъкналата обвивка на точките от P.

2. Триангулацията на Делоне съдържа най много ($n^{d/2}$) триъгълника(n-брой на точките,d-размерността на пространството).

3. Триангулацията на Делоне максимизира минималният ъгъл. Ако имаме произволна триангулация на P, то можем да сме сигурни, че минималният измежду ъглите на триъгълниците ъгъл, при триангулацията на Делоне, е по-голям или равен на минималният в произволна триангулация. Обратното не е вярно: Триангулацията на Делоне не винаги минимизира максималният ъгъл.



2 Референции

Референции

- Algorithms in C - Robert Sedgewick
- http://en.wikipedia.org

Редакция

- Инфоман (главен редактор: Петър Иванов, e-mail: petar.ivelinov@musala.com)

Контакти

- Христо Борисов (автор) - e-mail: hborisoff@gmail.com
- Иван Тодоров (съавтор) - e-mail: ivan.todorov.bs@gmail.com