Linguaggi di Programmazione

AA 2014-2015
Progetto luglio 2015
Consegna 26 luglio 2015, h. 23:59 GMT+1

Chiglie Convesse (Convex Hulls)

Marco Antoniotti, Gabriella Pasi e Giuseppe Vizzari

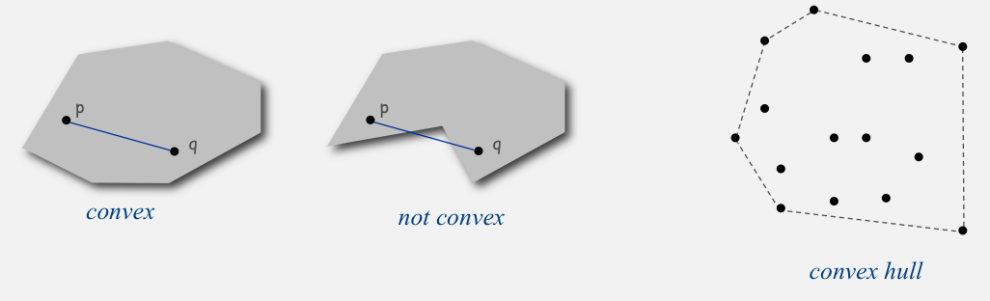
Introduzione

Il calcolo delle *chiglie convesse* (*convex hulls*) è uno degli algoritmi fondamentali della *geometria computazionale*. L'algoritmo ha numerosissime applicazioni e, una volta implementate correttamente alcune primitive geometriche, è di una semplicità sorprendente.

Il vostro compito è di implementare due librerie, una in *Common Lisp* e una in *Prolog* che forniscano un'implementazione dell'algoritmo per il calcolo delle chiglie convesse.

Definizioni

Dato un insieme di punti **P**, l'insieme è convesso se, presi due qualunque punti p e $q \in \mathbf{P}$, il segmento pq è completamente all'interno di **P**. La convex hull di **P** è il più piccolo insieme convesso che contiene tutti i punti di **P**. Nel seguito useremo la notazione CH(**P**) per indicare la convex hull di **P**[1].



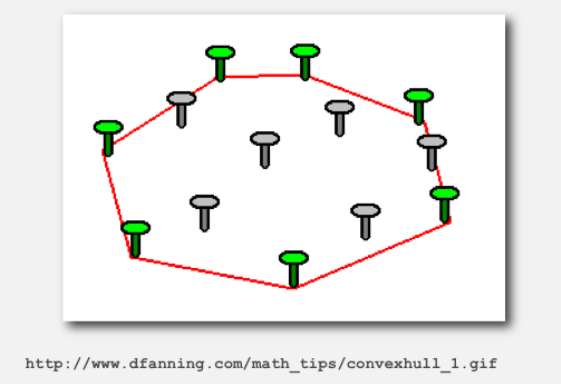
La CH(**P**) ha le seguenti proprietà.

- È la forma più semplice che "approssima" un insieme di punti.
- È il perimetro più corto che delimita **P**.
- È il poligono di area minore che copre tutti i punti di **P**.

Intuizione

Il dispositivo "meccanico" per calcolare CH(P) è costituito da una tavola di legno, chiodi, martello ed un elastico.

[1] Tutte le figure sono tratte dalle slides dei corsi di Sedgewick su Algoritmi e Strutture Dati di Princeton.



http://www.dfanning.com/math_tips/convexhull_1.gif

Soluzioni

Vi sono molte soluzioni al problema del calcolo di CH(**P**). Se consideriamo un insieme di punti **P** come input con |**P**| = *N*, allora il miglior algoritmo ha complessità temporale O(*N*lg(*N*)). Una sua implementazione usa il cosiddetto "*Graham's Scan*" (la *scansione di Graham*).

Si sceglie il punto $s \in \mathbf{P}$ con l'ordinata minore (in caso di pareggio si sceglie quello con l'ascissa minore).

Si ordinano i rimanenti punti secondo l'angolo polare che formano con s, in modo da ottenere un poligono semplice (anche se non convesso).

Si considerano i punti in ordine e si scartano quelli che imporrebbero una "*svolta a destra*".

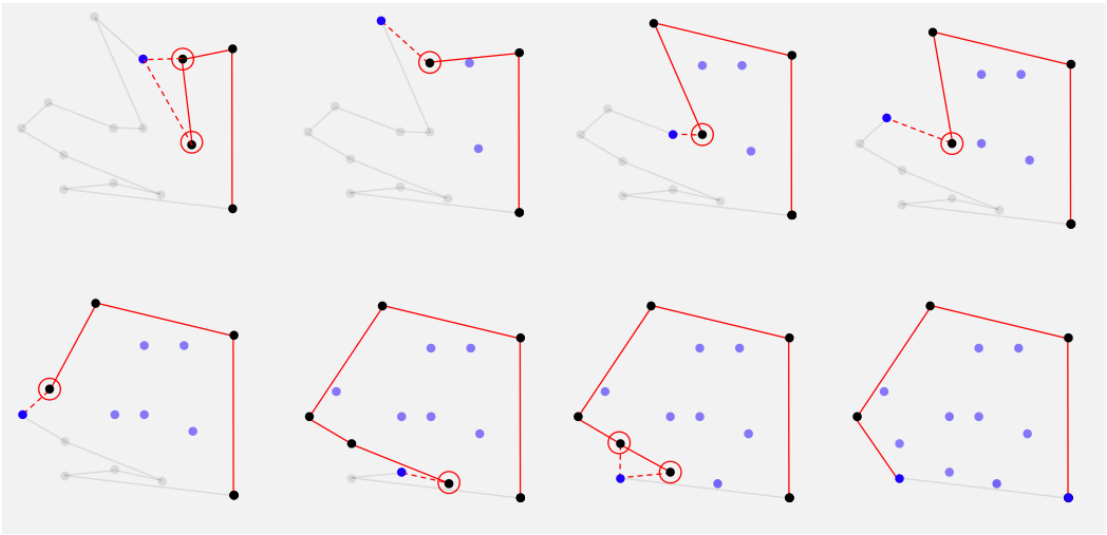


Figura 1: I punti da cui bisognerebbe svoltare a destra sono scartati.

L'algoritmo è molto semplice, ma richiede l'implementazione molto attenta di una serie di primitive geometriche. Scopo del progetto è di implementare tutte queste primitive e l'algoritmo per il calcolo della CH(**P**).

Preliminari

Le primitive geometriche sono molto complesse da realizzarsi correttamente. Per il momento ci si limiterà a punti nel piano a 2D con coordinate **intere** (questo vincolo è fondamentale).

L'interfaccia CL richiesta è la seguente

`make-point` *x y* ⇒ *point-2D*

`x` *point-2D* ⇒ <*integer coordinate*>

`y` *point-2D* ⇒ *<integer coordinate>*

La rappresentazione interna di un punto è lasciata alla vostra immaginazione. Comunque devono valere le seguenti uguaglianze:

```
cl-prompt> (x (make-point 0 42))
0

cl-prompt> (y (make-point 42 123))
123
```

Per il Prolog le cose sono relativamente più semplici. Potete usare direttamente il meccanismo di unificazione per gestire i punti ed estrarre le loro coordinate. Ovvero, potete usare il termine **`pt(X, Y)`** per rappresentare un punto.

A questo punto avete bisogno di primitive geometriche più raffinate. In particolare dovrete implementare il test "svolta a sinistra". Avrete bisogno delle funzioni/predicati seguenti descritti di seguito.

Area di un triangolo

Implementate una funzione `area2` che calcoli l'area di un triangolo date le coordinate dei vertici. Dati tre punti, *a*, *b*, e *c* (vertici del triangolo) si suggerisce di usare la formula:

$$2 \times A(a,b,c) = \Big(\big(x(b) - x(a)\big) \times \big(y(c) - y(a)\big)\Big) - \Big(\big(y(b) - y(a)\big) \times \big(x(c) - x(a)\big)\Big)$$

Notate che il valore può essere negativo (quando i 3 punti sono ordinati in senso orario), il che è in realtà molto utile: se il valore è positivo, allora $\angle abc$ è un angolo a sinistra, se è negativo allora $\angle abc$ è un angolo a destra. Il caso 0 è degenere e capita quando *a*, *b* e *c* sono *collineari*. Il capitolo 1 di [O98] spiega bene tutti i dettagli. Dato che il nostro interesse non è calcolare l'area del triangolo ma capire se un certo angolo è legato a una "svolta a destra" non è necessario fare la divisione per 2 finale.

`area2` *a b c* ⇒ *integer*

In Prolog dovrete implementare il predicato

`area2(A, B, C, Area)`

`left`, `left-on`, `is-collinear`, `left/3`, `left_on/3`, `collinear/3`

Data la funzione `area2`, si possono scrivere i predicati:

`left` *a b c* ⇒ *boolean*

`left-on` *a b c* ⇒ *boolean*

`is-collinear` *a b c* ⇒ *boolean*

che ci dicono se i tre punti rappresentano una svolta a sinistra oppure no.

In Prolog avrete i predicati:

`left(A, B, C)`

`left_on(A, B, C)`

`collinear(A, B, C)`

Angolo tra due punti

Infine avrete bisogno di una funzione `angle2d` che ritorni l'angolo (in radianti) tra due punti. Avete bisogno di una funzione standard delle librerie Common Lisp e Prolog.

`angle2d` *a b* ⇒ *real* ∈ $[-\pi,\pi]$.

In Prolog il predicato diventa:

`angle2d(A, B, R).`

Chiglia convessa

Date le funzioni e i predicati di cui sopra potete implementare tranquillamente l'algoritmo per la costruzione di CH(**P**).

`ch` *points* ⇒ *result*

La variabile *points* (ovvero **P**) contiene una lista di punti, il risultato *result* (ovvero CH(**P**)) è anch'esso una lista di punti.

In Prolog:

`ch(`*`Points`*`, `*`Result`*`)`

Altri suggerimenti

Avrete bisogno della funzione standard Common Lisp `sort`. Secondo il tipo d'implementazione che sceglierete di fare potreste avere bisogno di `push`, `pop` e `loop`.

In Prolog potete usare il predicato `sort` (attenzione a com'è definito!)

Sia in Common Lisp che in Prolog avete bisogno della funzione `atan` (o simili).

Input (ed Output)

Il vostro progetto deve prevedere anche una funzione (un predicato) che legga da file una serie di punti.

`read-points` *filename* ⇒ *points*

filename è un nome di file e *points* è una lista di punti.

Il contenuto di filename è costituito da un numero **pari** d'interi, due per riga, separati da un carattere di *tabulazione* (il carattere "Tab": in Common Lisp `#\Tab`, in Prolog `0'\t`, in C e derivati `'\t'`). Ad esempio questo è un file di test corretto:

```
0      6
3      7
4      6
4      5
3      4
2      4
5      0
42     123
-1     0
1024   -42
0      -2
1      2
```

Attenzione ai punti duplicati.

In Prolog:

`read_points(`*`Filename`*`, `*`Points`*`).`

In SWI Prolog potete utilizzare la libreria standard `csv` per leggere il file di punti.

Da consegnare

Dovrete consegnare un file `.zip` (i files `.tar`, `.7z`, `.tgz`, `.tar.gz`, `.rar` etc. **non sono accettabili**!!!) dal nome

`Cognome_Nome_Matricola_ch_LP_201507.zip`

Cognomi e nomi multipli dovranno essere scritti sempre con il carattere "underscore" ('_'). Ad esempio, "Gian Giacomo Pier Carl Luca De Mascetti Vien Dal Mare" che ha matricola 424242 diventerà:

`De_Mascetti_Vien_Dal_Mare_Gian_Giacomo_Pier_Carl_Luca_424242_ch_LP_201507`

Questo file deve contenere una sola directory con lo stesso nome. Al suo interno ci devono essere due sottodirectory chiamate '`Lisp`' e '`Prolog`'. Al loro interno queste cartelle devono contenere i files caricabili e interpretabili, più tutte le istruzioni che riterrete necessarie. Il file Lisp si deve chiamare '`compgeo.lisp`'. Il file Prolog si deve chiamare '`compgeo.pl`'. La cartella deve contenere un file chiamato `README.txt`. In altre parole questa è la struttura della directory (folder, cartella) una volta spacchettata.

```
Cognome_Nome_Matricola_ch_LP_201507
       Lisp
              compgeo.lisp
              README.txt
       Prolog
              compgeo.pl
              README.txt
```

Potete anche aggiungere altri file, ma il loro caricamento dovrà essere effettuato automaticamente al momento del caricamento ("loading") dei files sopracitati.

Ogni studente deve consegnare, anche se lavora in gruppo. Le prime linee dei files devono contenere la composizione del gruppo di lavoro.

```
prompt$ unzip -l Antoniotti_Marco_424242_ch_LP_201507.zip
Archive:  Antoniotti_Marco_424242_ch_LP_201507.zip
  Length      Date    Time    Name
 --------    ----    ----    ----
        0  12-02-14 09:59   Antoniotti_Marco_424242_ch_LP_201507/
        0  12-04-14 09:55   Antoniotti_Marco_424242_ch_LP_201507/Lisp/
     4623  12-04-14 09:51   Antoniotti_Marco_424242_ch_LP_201507/Lisp/compgeo.lisp
    10598  12-04-14 09:53   Antoniotti_Marco_424242_ch_LP_201507/Lisp/README.txt
        0  12-04-14 09:55   Antoniotti_Marco_424242_ch_LP_201507/Prolog/
     4623  12-04-14 09:51   Antoniotti_Marco_424242_ch_LP_201507/Prolog/compgeo.pl
    10598  12-04-14 09:53   Antoniotti_Marco_424242_ch_LP_201507/Prolog/README.txt
 --------                   -------
    30442                   7 files
```

Valutazione

Il vostro programma sarà controllato con set di files assolutamente arbitrari (di cui almeno uno totalmente casuale), quindi, dovrete essere pronti a ogni evenienza.

I tests, dato un *filename*, sono costituiti da espressioni siffatte:

Lisp:

```
cl-prompt> (equalp expected-result (ch (read-points filename)))
```

Prolog:

```
?- read_points(Filename, Points),
   ch(Points, CH),
   CH = Expected_result.
```

La CH(**P**) è deterministica e quindi la sua struttura non cambia a seconda dell'implementazione sulla base delle primitive proposte. Un'implementazione in conformità a primitive diverse non sarà considerata. In particolare, il vincolo sul tipo dei numeri da usare (interi) fa si che non ci siano problemi sul test di uguaglianza tra numeri.

Notate che vi sono diversi algoritmi per il calcolo di CH(**P**). L'algoritmo semplice e brutale ha complessità $O(N^3)$. I correttori non hanno intenzione di aspettare troppo per valutare ogni elaborato… quindi è meglio che implementiate degli algoritmi più efficienti.

Abbiamo a disposizione una serie di esempi standard che saranno usati per una valutazione oggettiva dei programmi. In particolare, i files di test che useremo hanno un numero di punti dell'ordine di 10^2 e sono generati in modo casuale. Farete bene a testare i vostri programmi in modo similare.

Se i files sorgente non potranno essere letti/caricati negli ambienti Lisp e Prolog (NB: Lispworks e SWI-Prolog), il progetto non sarà ritenuto sufficiente.

Il mancato rispetto dei nomi indicati per funzioni e predicati, o anche delle strutture proposte e della semantica esemplificata nel testo del progetto, oltre a comportare ritardi e possibili fraintendimenti nella correzione, può comportare una riduzione nel voto ottenuto o la non correzione dell'elaborato.

Riferimenti

[O98] J. O'Rourke, *Computational Geometry in C*, Cambridge University Press, 1998.

[CLR+01] T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein, *Introduction to Algorithms*, 2nd Edition, MIT Press, 2001