Projeto e Análise de Algoritmos
Trabalho 1

1 Descrição

Este trabalho considera a implementação e análise de um algoritmo de divisão e conquista para o problema do Fecho Convexo (*Convex Hull*).

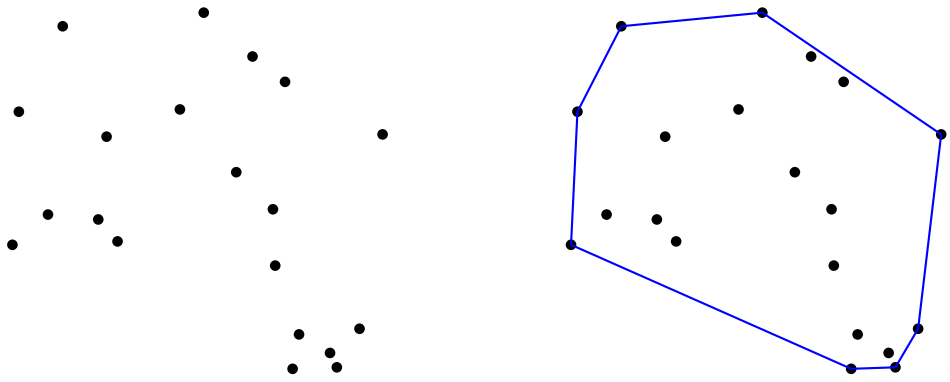
Cada trabalho submetido deverá ser desenvolvido por no máximo três estudantes. Os códigos devem conter cabeçalhos discriminando cada um dos estudantes que desenvolveu o trabalho submetido, com nome completo e registo acadêmico.

A avaliação do trabalho irá considerar os códigos, a execução da implementação e a descrição dos algoritmos. Todo o conteúdo submetido pelos estudantes deve ser de autoria deles.

Na sequência deste documento serão descritos os detalhes sobre o trabalho. Leia com atenção. Trabalhos que não respeitarem esta descrição serão penalizados.

2 Problema do Fecho Convexo

Dado um conjunto de n pontos $S = \{p_1, p_2, \ldots, p_n\}$ no plano ($p_i = (x_i, y_i)$ para $i = 1, 2 \ldots, n$), o fecho convexo de S é o menor polígono convexo P que contêm S. As Figuras 1a e 1b ilustram, respectivamente, um conjunto de pontos no plano e o seu fecho convexo.



(a) (b)

Figura 1: Ilustração de alguns pontos no plano (a) e do fecho convexo (b).

3 Algoritmos para o Problema

O algoritmo para encontrar o fecho convexo que consideraremos neste trabalho é chamado *Quick Hull*. Passos básicos e alguns conceitos auxiliares são descritos na sequência.

3.1 Início

O algoritmo inicia encontrando dois pontos extremos, que certamente pertencem ao fecho convexo do conjunto de pontos. Podemos, por exemplo usar o ponto com menor coordenada X (e maior Y em caso de empate) e o ponto com maior coordenada X (e menor Y em caso de empate). A Figura 2 mostra, em azul, os pontos p e q encontrados para o exemplo.

Considerando o segmento orientado pq, podemos dividir o conjunto de pontos encontrando os pontos à sua esquerda (Figura 3a) e, da mesma forma, podemos encontrar os pontos à esquerda do segmento orientado qp (Figura 3b). Na sequência, computamos recursivamente o fecho convexo para os pontos considerando separadamente cada uma das partições.

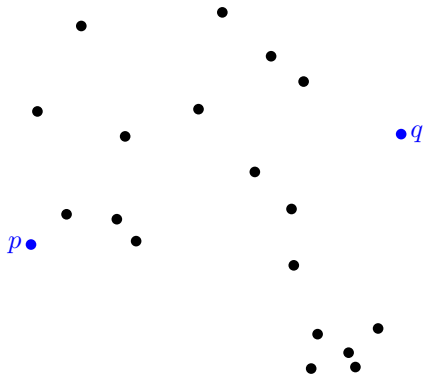


Figura 2: Identificação de pontos extremos.

3.2 Considerando o Subconjunto do Segmento Orientado pq

Considerando o subconjunto de pontos à esquerda de pq, encontramos o ponto r mais distante em relação ao segmento pq (Figura 4a). Na sequência, subdividimos os pontos entre aqueles à esquerda do segmento pr e aqueles à esquerda do segmento rq (Figura 4b). Os pontos localizados à direita dos segmentos, consequentemente dentro do triângulo pqr, são eliminados pois não podem pertencer ao fecho convexo. Recursivamente, resolvemos o problema usando os segmentos pr e rq com os pontos respectivamente à esquerda deles (Figuras 4ca 4f). Os pontos mais distantes encontrados em cada recursão formam o fecho convexo à esquerda de pq, como no exemplo da Figura 4g.

3.3 Considerando o Subconjunto do Segmento Orientado qp

Seguindo os mesmos passos, agora considerando o segmento qp, computamos a segunda parte do fecho convexo (Figura 5), completando o fecho convexo de todos os pontos que o formam (Figura 5g).

3.4 Representação do Fecho Convexo

A saída do algoritmo deve ser escrita como a sequência dos pontos, que compõem o fecho convexo, dada em sentido anti-horário. Isso é fácil de obter usando os passos do algoritmo (talvez com alguma alteração na ordem dos passos). No exemplo da Figura 5g, tomando arbitrariamente p como o ponto inicial, teríamos a sequência p, w, u, v, q, t, r, s. Essa é uma representação computacional comumente usada para polígonos. Note que é uma representação cíclica, o último ponto é conectado com o primeiro.

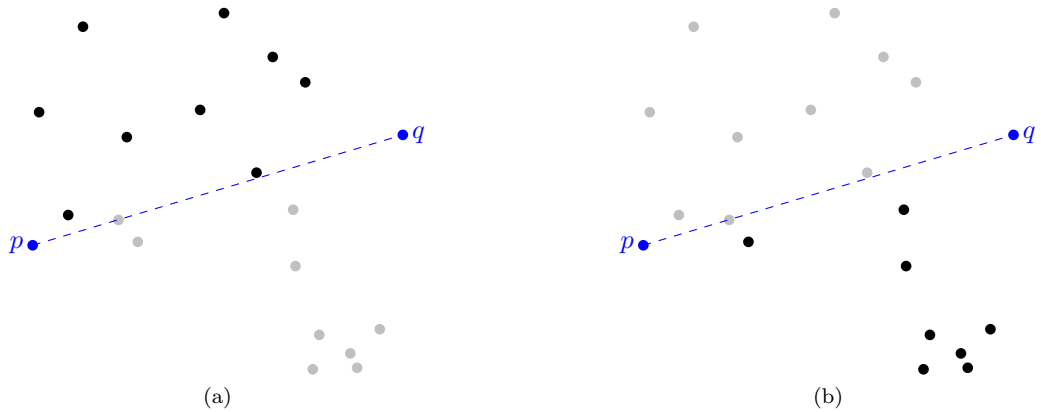


Figura 3: Subdivisão do conjunto de pontos em conjuntos à esquerda do segmento orientado pq (a) e à esquerda do segmento orientado qp (b).

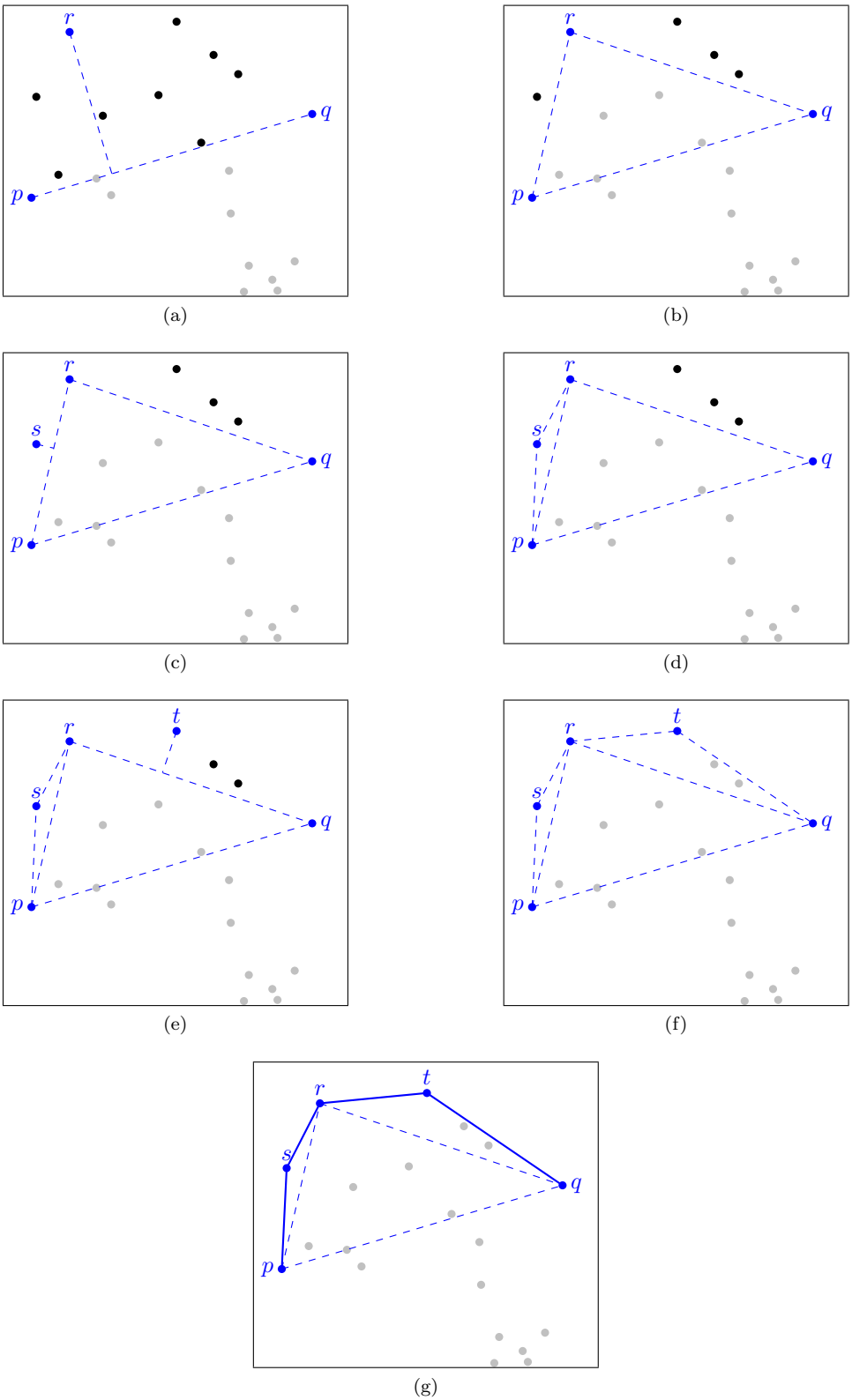


Figura 4: Ilustração dos passos recursivos.

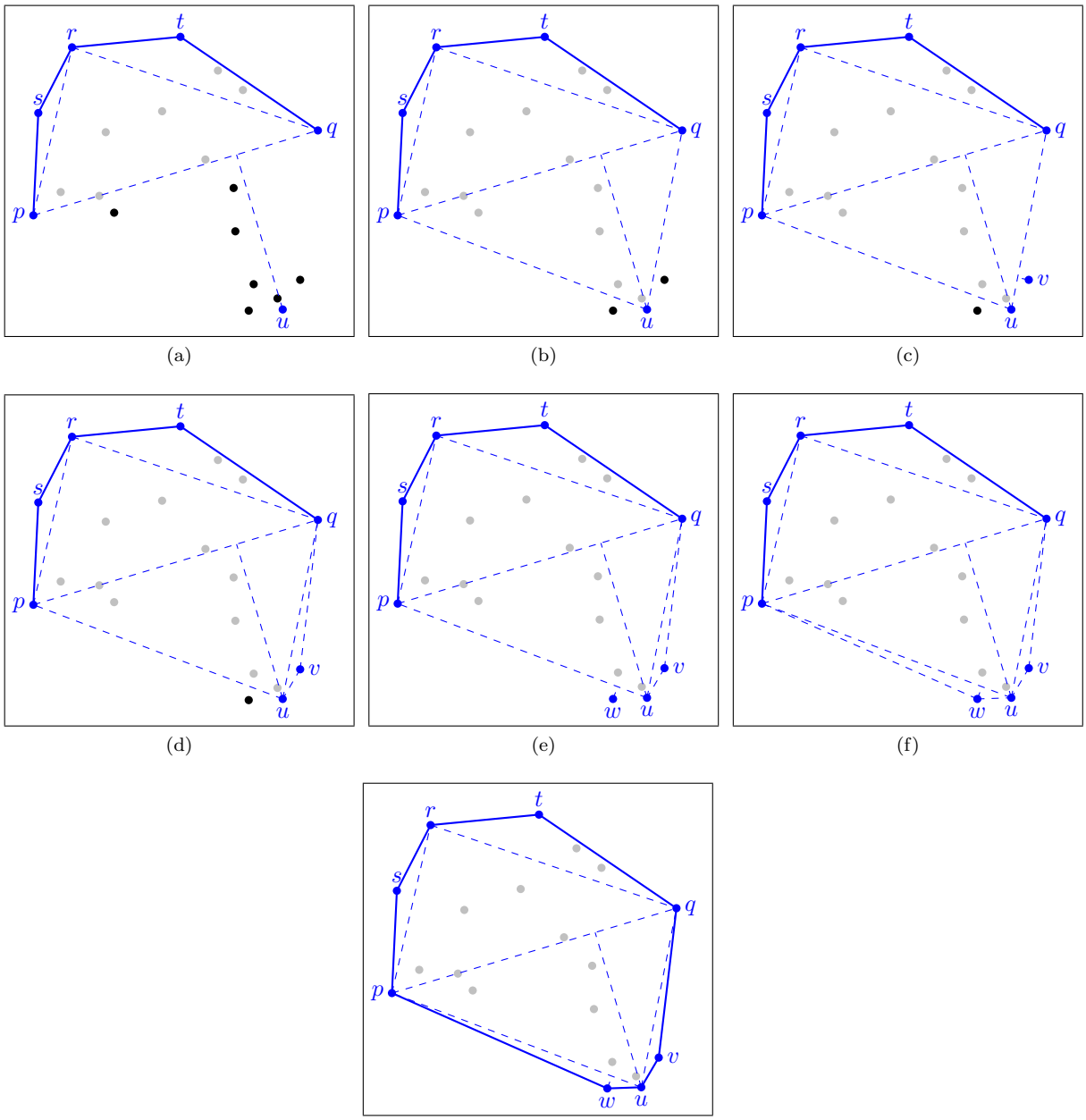


Figura 5: Ilustração dos passos recursivos.

3.5 Posição de um ponto em relação à uma reta

Como descrito no algoritmo, uma operação importante é descobrir se uma ponto $p_3 = (x_3, y_3)$ está à esquerda de um segmento de reta p_1p_2, definido por seus pontos finais $p_1 = (x_1, y_1)$ e $p_2 = (x_2, y_2)$. Essa operação pode ser facilmente realizada com alguns conceitos geométricos.

Considerando $p_1 = (x_1, y_1)$, $p_2 = (x_2, y_2)$, temos um segmento orientado p_1p_2. Como exemplo, veja a Figura 6. O ponto $p_3 = (x_3, y_3)$ pode estar à esquerda do segmento (Figura 6a), à sua direita (Figura 6b) ou ser coincidente (Figura 6c).

Considerando quaisquer três pontos $p_1 = (x_1, y_1)$, $p_2 = (x_2, y_2)$ e $p_3 = (x_3, y_3)$, Figura 7, temos um paralelogramo formado pelos vetores $\overrightarrow{u} = p_2 - p_1 = (x_2 - x_1, y_2 - y_1)$ e $\overrightarrow{v} = p_3 - p_1 = (x_3 - x_1, y_3 - y_1)$.

Considerando que o módulo do produto vetorial corresponde à área do paralelogramo, podemos com-

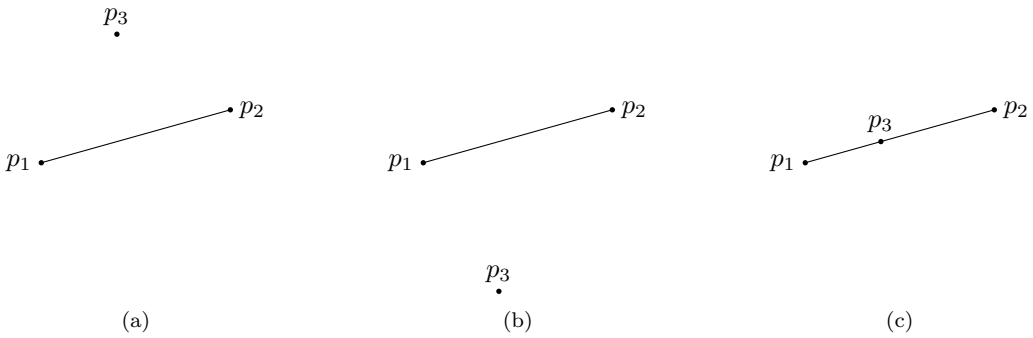


Figura 6: Posição relativa de um ponto dado um segmento.

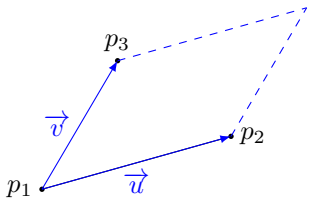


Figura 7: Paralelogramo formado por vetores de três pontos.

putar a área com sinal do triângulo $p_1p_2p_3$ como

$$A(p_1, p_2, p_3) = (x_2 - x_1)(y_3 - y_1) - (y_2 - y_1)(x_3 - x_1),$$

cujo resultado é:

- $A(p_1, p_2, p_3) > 0$, se p_3 está à esquerda de p_1p_2;
- $A(p_1, p_2, p_3) < 0$, se p_3 está à direita de p_1p_2;
- $A(p_1, p_2, p_3) = 0$, se p_3 é coincidente a p_1p_2.

3.6 Distância ponto-reta

A distância de um ponto $p_3 = (x_3, y_3)$ à reta definida pelo segmento p_1p_2 (Figura 8), $p_1 = (x_1, y_1)$ e $p_2 = (x_2, y_2)$, também poder ser calculada geometricamente.

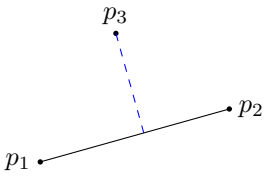


Figura 8: Distância ponto-reta.

Temos que a distância d é a altura do paralelogramo formado pelos vetores $\overrightarrow{u} = p_2 - p_1 = (x_2 - x_1, y_2 - y_1)$ e $\overrightarrow{v} = p_3 - p_1 = (x_3 - x_1, y_3 - y_1)$ (Figura 9).

A área do paralelogramo pode ser então definida como o módulo do produto vetorial $|u \times v|$ ou pelo produto da base (módulo do vetor v) pela altura d. Igualando os dois, temos

$$|u \times v| = |u|d$$

ou

$$d = \frac{|u \times v|}{|u|}.$$

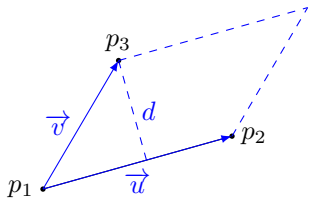


Figura 9: Distância ponto-reta

Portanto, dados os pontos p_1, p_2 e p_3,

$$d(p_1, p_2, p_3) = \frac{|u \times v|}{|u|}$$
$$= \frac{\sqrt{((x_2 - x_1)(y_3 - y_1) - (y_2 - y_1)(x_3 - x_1))\,((x_2 - x_1)(y_3 - y_1) - (y_2 - y_1)(x_3 - x_1))}}{\sqrt{((x_2 - x_1)(x_2 - x_1) + (y_2 - y_1)(y_2 - y_1))}}.$$

4 Implementação e Execução

As implementações devem ser feitas em linguagem C ou C++ e compilar, respectivamente, usando os compiladores gcc e g++, em sistema operacional Linux.

Um arquivo Makefile deve ser disponibilizado para compilar e gerar o arquivo executável. A compilação deve ser feita usando o comando *make* em um terminal de comandos do Linux, conforme o exemplo abaixo. Nos exemplos, *prompt* indica comandos no terminal.

```
prompt: make
```

O programa gerado pelo comando *make* deve ter o nome *hull* e receber como argumento um arquivo TXT com os pontos de entrada (o formato do arquivo será descrito na sequência). A execução deve ser iniciada em um terminal de comandos do Linux, como no exemplo abaixo.

```
prompt: ./hull input.txt
```

O programa *hull* deve computar o fecho convexo usando o algoritmo *Quick Hull* e apresentar as seguintes saídas:

- tempo de execução, na saída do terminal Linux;
- um arquivo contendo a sequência de pontos que formam o fecho convexo.

Os detalhes de formatação das saídas são descritos na sequência.

4.1 Arquivo de Entrada

O arquivo de entrada deve ser formatado da seguinte maneira:

- a primeira linha contém um inteiro n com o número de pontos no arquivo a serem lidos como entrada para o programa;
- as n linhas que seguem contêm as coordenadas dos pontos em valores inteiros. Cada linha representa um ponto e contém o seu valor de x e depois de y, separados por um espaço.

Um exemplo de arquivo de entrada com 20 pontos é mostrado abaixo.

```
20
630 574
896 58
544 996
759 813
```

```
316 391
55 734
878 96
733 327
779 54
171 960
481 740
673 880
1017 674
796 145
38 382
287 668
956 160
727 476
132 462
265 449
```

No Moodle da disciplina foi disponibilizado um código para gerar um arquivo de pontos aleatórios neste formado. O código tem nome "genpoints.c". Para compilar basta executar o make que acompanha o código.

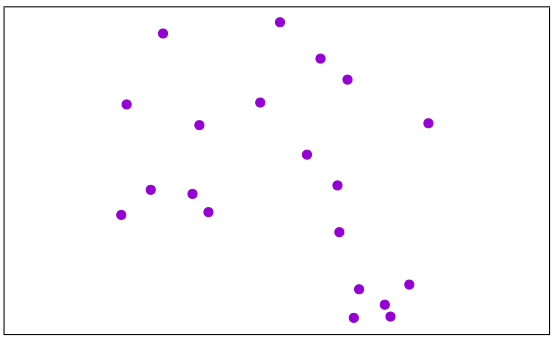
```
prompt: make
```

O programa recebe como argumento o número de pontos desejados, conforme o exemplo abaixo,

```
./genpoints 523
```

e gera um arquivo chamado "input.txt" no formato da entrada.

Caso queira visualizar os pontos do arquivo usando o programa *gnuplot*, está disponível no Moodle o script *pontos.plot*, que gera um arquivo chamado *pontos.pdf* com os pontos. A figura abaixo mostra a saída do *gnuplot* para o arquivo de entrada do exemplo.



Para criar o arquivo basta executar a seguinte linha no terminal[1].

```
prompt: npts=$(head -1 input.txt); tail -$npts input.txt > input2.txt; gnuplot pontos.plot
```

4.2 Formatação da Saída no Terminal

O programa *hull* deve imprimir no terminal uma linha com o tempo de execução do algoritmo em segundos. O valor deve estar em formato de ponto flutuante com seis casas decimais. Veja o exemplo abaixo. Você não deve acrescentar nada mais à saída, apenas o valor, exatamente como no exemplo.

```
prompt: ./hull input.txt
2.000301
prompt:
```

[1] Para execução em Linux e você deve ter o *gnuplot instalado.*

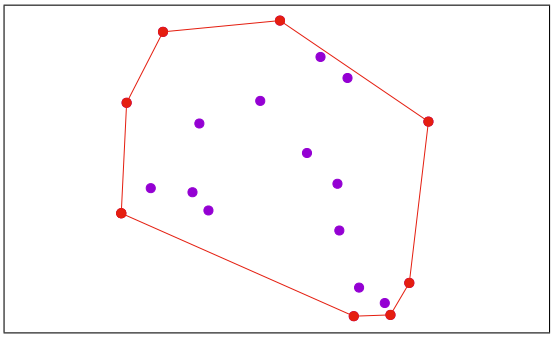
4.3 Formatação do Arquivo de Saída

O arquivo de saída para os pontos do fecho convexo deve ter o nome *fecho.txt*. As coordenadas x e y de um ponto do fecho devem aparecer em uma linha, separadas por espaço. A ordem das linhas no arquivo deve seguir a ordem anti-horária conforme explicado na Seção 3.4.

Um exemplo de arquivo para o nosso exemplo é mostrado abaixo.

```
38 382
779 54
896 58
956 160
1017 674
544 996
171 960
55 734
```

Caso queira visualizar o fecho do arquivo usando o *gnuplot*, está disponível no Moodle o script *fecho.plot* que gera um arquivo chamado *fecho.pdf*. A figura abaixo mostra a saída para o exemplo.



Para criar o arquivo basta executar a seguinte linha no terminal.

```
prompt: cp fecho.txt fecho2.txt; head -1 fecho.txt >> fecho2.txt; gnuplot fecho.plot
```

5 Documentação do código

Seu código deve estar bem comentado, descrevendo de forma clara as principais linhas do código. Também, precedendo os itens abaixo, deverá haver comentários descrevendo os itens de forma clara, completa e bem organizada:

- estruturas de dados criadas: o que armazenam e como, onde são usadas, etc.;
- funções: suas entradas, saídas, descrição dos algoritmos, adaptações importantes, explicação sobre ordem de complexidade, explicação sobre corretude, etc.

6 Entrega

As entregas deverão ser realizadas usando o link de entrega no Moodle e não serão aceitas entregas fora do prazo.

Os autores do trabalho devem reunir os códigos-fonte em um diretório nomeado com os números dos seus registros acadêmicos (RAs) separados por '_'. Por exemplo, se dois estudantes desenvolveram o trabalho e têm RAs 123456 e 789012, então o diretório deve ter o nome 123456_789012.

Dentro do diretório deverão estar os códigos-fonte e o arquivo Makefile. Por exemplo, o diretório poderia conter os arquivos listados abaixo (não coloque acentos ou espaços em nomes de arquivos).

```
123456_789012/hull.c
123456_789012/Makefile
```

O arquivo de entrega deve ser um ZIP do diretório. O arquivo ZIP deve inclusive conter o diretório e ser nomeado da mesma maneira que o diretório. No nosso exemplo: 123456_789012.zip.