

Geometria Computacional

Envoltório Convexo

Prof. Edson Alves

2019

Faculdade UnB Gama

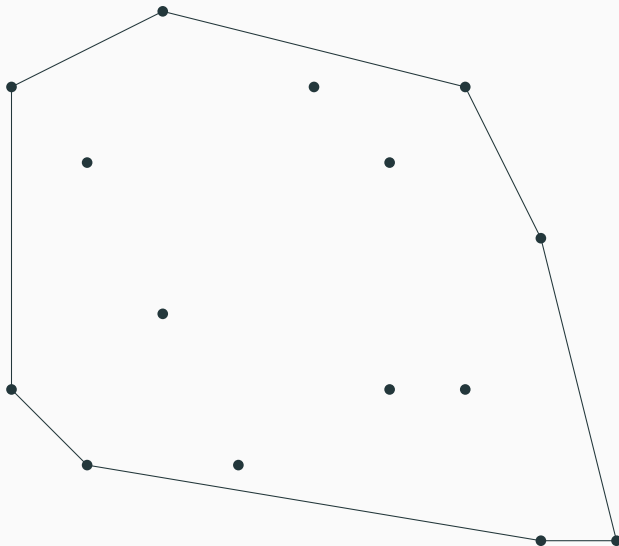
1. Definição
2. Algoritmo de Graham

Definição

Envoltório convexo

- Dado um conjunto de N pontos P , o envoltório convexo $C_H(P)$ de P (*convex hull*) é o menor polígono convexo tal que cada ponto de P ou pertence ao interior de $C_H(P)$ ou é um de seus vértices
- O termo menor na definição acima se refere à menor área
- O envoltório convexo não é único, pois não impõem restrição na orientação do polígono
- Existem vários algoritmos para se determinar o envoltório convexo
- O mais conhecido é o algoritmo de Graham
- Além deles, outros dois algoritmos importantes são a cadeia monótona de Andrew e a marcha de Jarvis
- Como os vértices de $C_H(P)$ são pontos de P , a essência dos algoritmos é determinar, para cada ponto de P , se ele pertence ou não ao $C_H(P)$

Exemplo de envoltório convexo



Algoritmo de Graham

Algoritmo de Graham

- O algoritmo de Graham (*Graham Scan*, no original), foi proposto por Ronald Graham em 1972
- Ele inicialmente ordena todos os N pontos de P de acordo com o ângulo que eles foram com um ponto pivô fixado previamente
- A escolha padrão para o pivô é o ponto de menor coordenada y
- Caso exista mais de um ponto com coordenada y mínima, escolhe-se o de maior coordenada x dentre eles
- Se P é armazenado em um vetor, o algoritmo pode ser simplificado movendo-se o pivô para a primeira posição

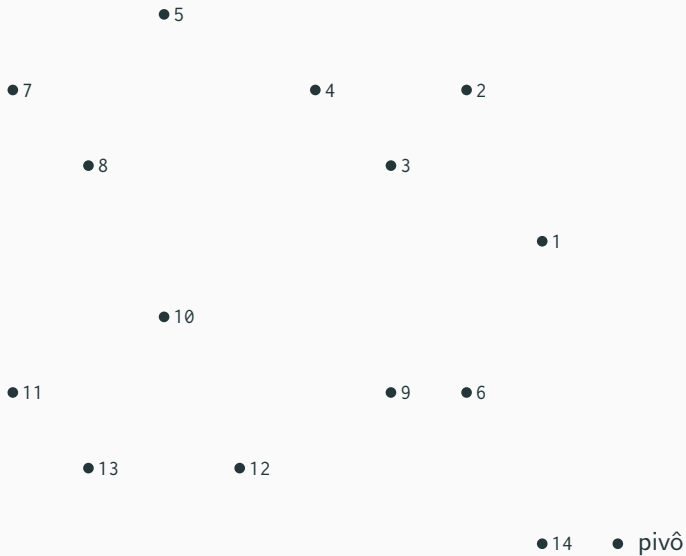
Implementação da escolha do pivô

```
34 template<typename T>
35 class GrahamScan
36 {
37 private:
38     static Point<T> pivot(vector<Point<T>>& P)
39     {
40         size_t idx = 0;
41
42         for (size_t i = 1; i < P.size(); ++i)
43             if (P[i].y < P[idx].y or
44                 (equals(P[i].y, P[idx].y) and P[i].x > P[idx].x))
45                 idx = i;
46
47         swap(P[0], P[idx]);
48
49         return P[0];
50     }
51 }
```


Ordenação dos pontos de acordo com o ângulo

- Para realizar a ordenação dos pontos é preciso definir um operador booleano que receba dois pontos P e Q e retorne verdadeiro se P antecede Q de acordo com a ordenação proposta
- Como é necessário o conhecimento do pivô para tal ordenação, há três possibilidades para a implementação deste operador:
 1. implementar o operador $<$ da classe `Point`, tornando o pivô um membro da classe para que o operador tenha acesso a ele;
 2. tornar o pivô uma variável global;
 3. usar uma função lambda no terceiro parâmetro da função `sort()`, capturando o pivô por referência ou cópia
- O ângulo que o vetor diferença entre o vetor-posição do pivô e o vetor posição de um ponto do conjunto P faz com o eixo- x positivo pode ser obtido através da função `atan2()` da biblioteca `math.h` da linguagem C/C++

Exemplo de ordenação por ângulo



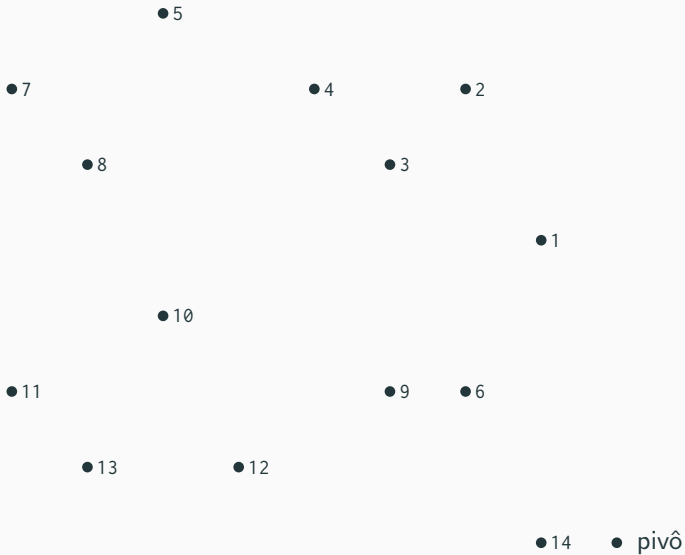
Implementação da rotina de ordenação dos pontos

```
52  static void sort_by_angle(vector<Point<T>>& P)
53  {
54      auto P0 = pivot(P);
55
56      sort(P.begin() + 1, P.end(),
57          [&](const Point<T>& A, const Point<T>& B) {
58              // pontos colineares: escolhe-se o mais próximo do pivô
59              if (equals(D(P0, A, B), 0))
60                  return A.distance(P0) < B.distance(P0);
61
62              auto alfa = atan2(A.y - P0.y, A.x - P0.x);
63              auto beta = atan2(B.y - P0.y, B.x - P0.x);
64
65              return alfa < beta;
66          })
67      );
68  }
```

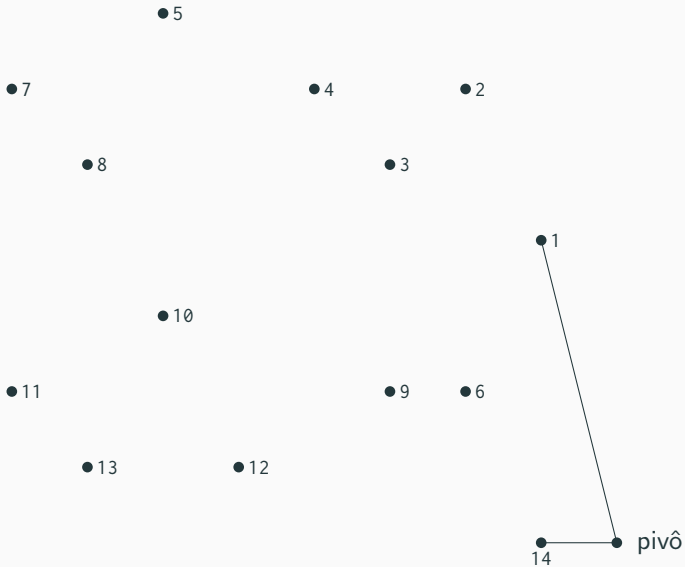
Identificação do envoltório convexo

- Após a ordenação dos pontos, o algoritmo procede empilhando três pontos de P : inicialmente os pontos cujos índices são $n - 1, 0$ e 1
- O invariante a ser mantido é que os três elementos do topo de pilha estão em sentido anti-horário ($D > 0$)
- Para cada um dos demais pontos Q_i de P , com $i = 2, 3, \dots, n - 1$, verifica-se se este ponto mantém o sentido anti-horário com os dois elementos do topo da pilha
- Em caso afirmativo, o ponto é inserido na pilha
- Caso contrário, remove-se o topo da pilha e se verifica o invariante para Q_i novamente
- Como cada ponto é ou inserido ou removido uma única vez, este processo tem complexidade $O(N)$, e o algoritmo como um todo tem complexidade $O(N \log N)$, devido à ordenação

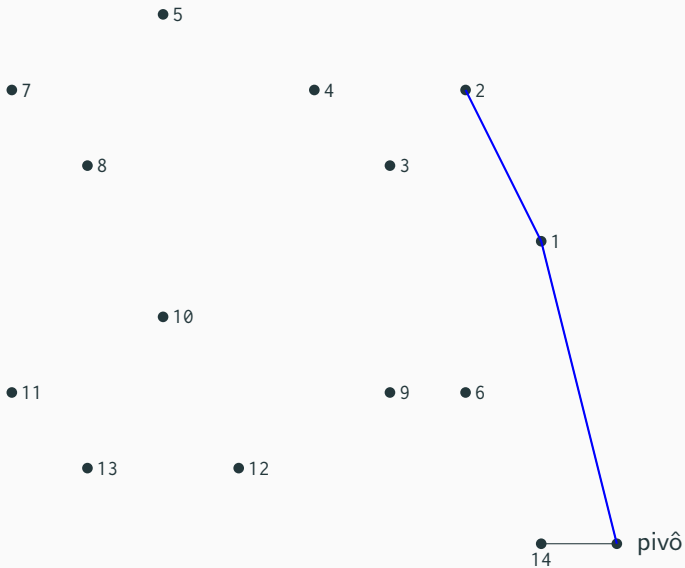
Visualização do algoritmo de Graham



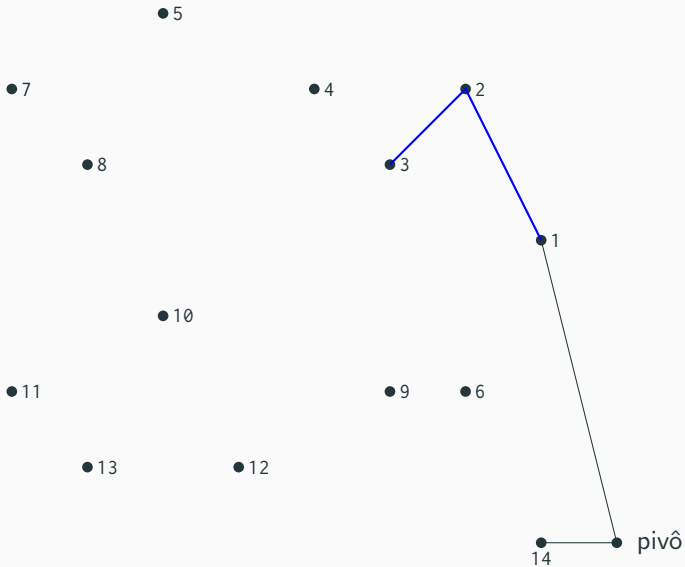
Visualização do algoritmo de Graham



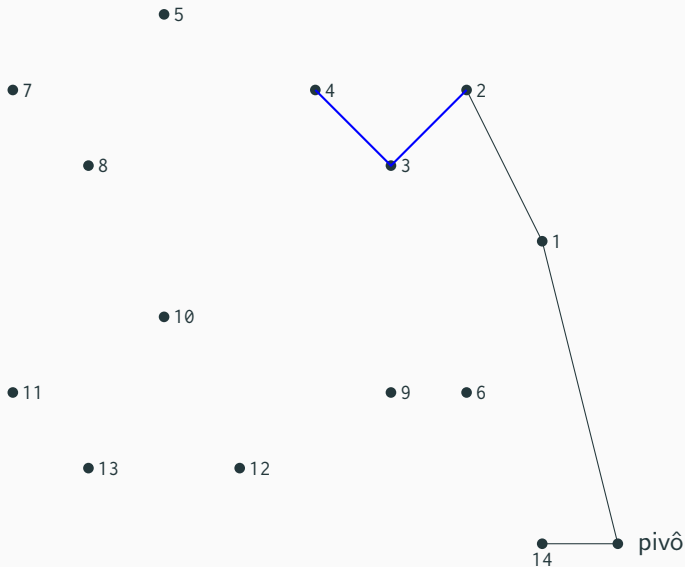
Visualização do algoritmo de Graham



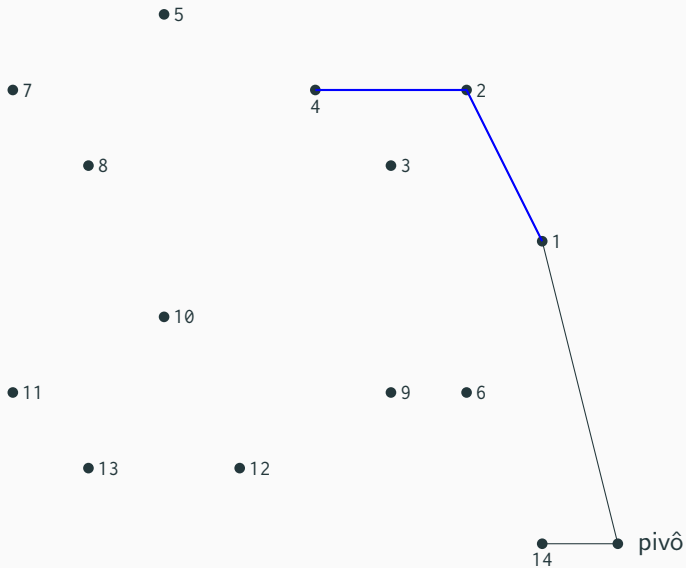
Visualização do algoritmo de Graham



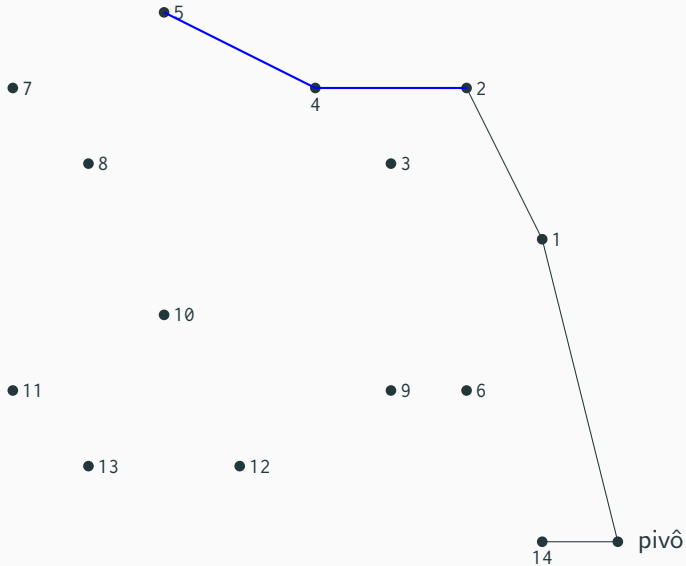
Visualização do algoritmo de Graham



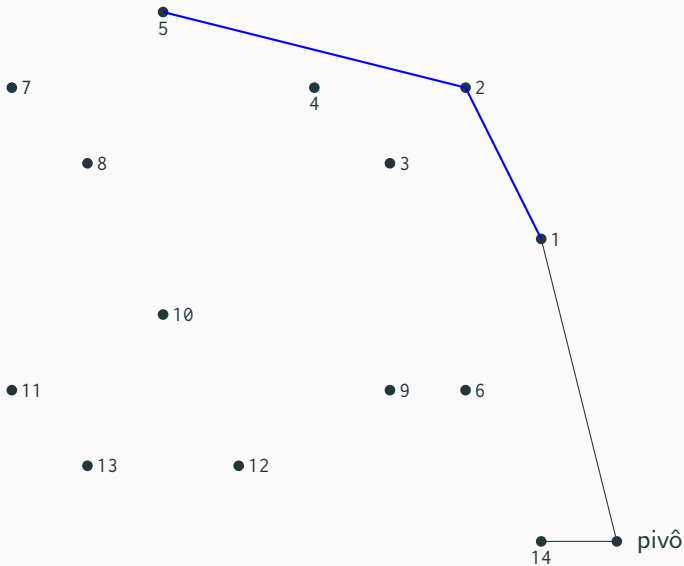
Visualização do algoritmo de Graham



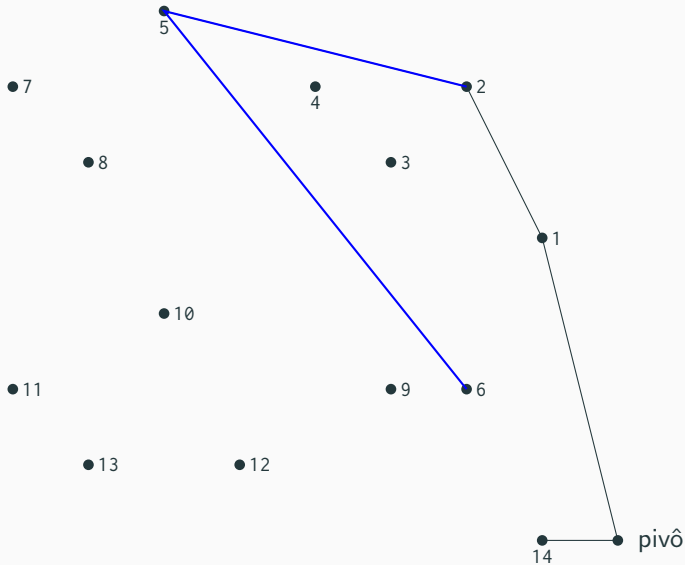
Visualização do algoritmo de Graham



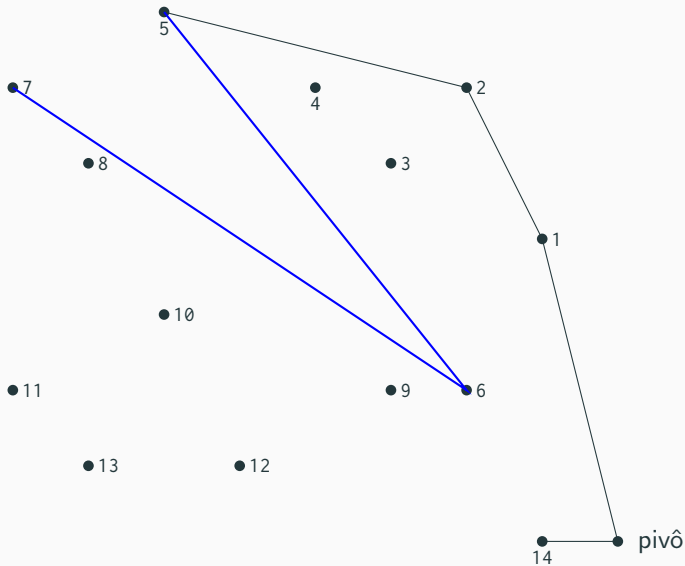
Visualização do algoritmo de Graham



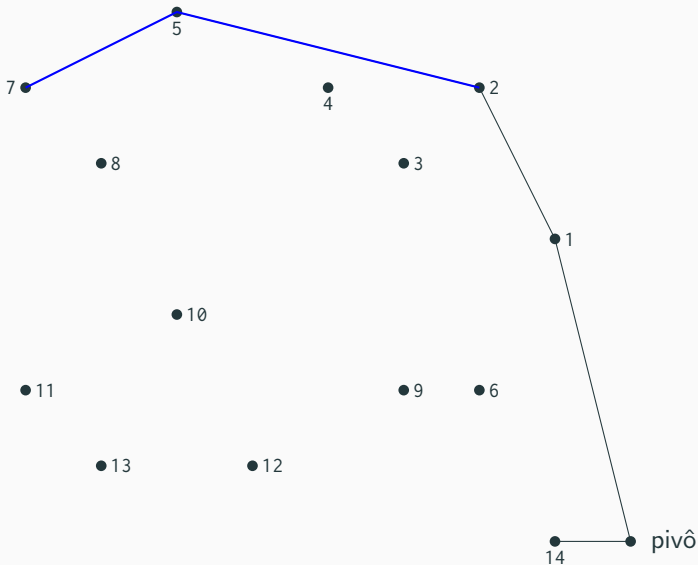
Visualização do algoritmo de Graham



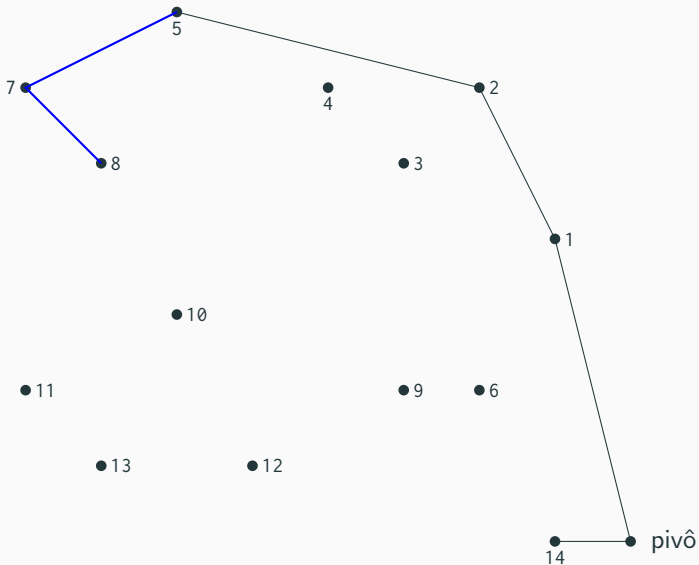
Visualização do algoritmo de Graham



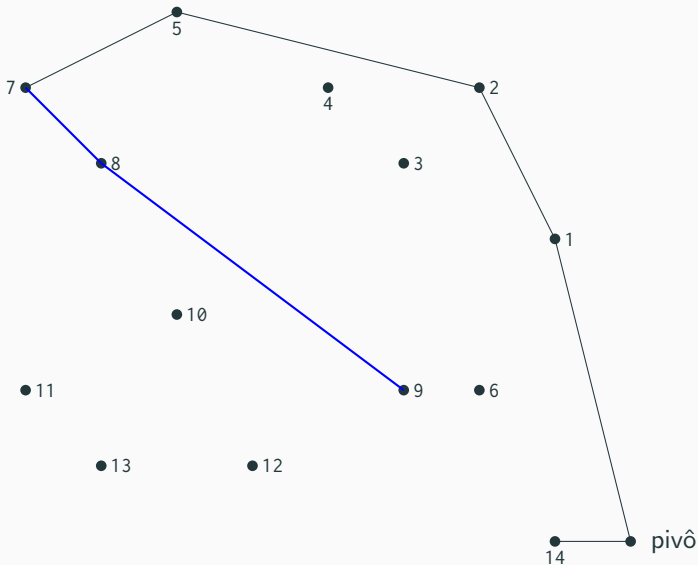
Visualização do algoritmo de Graham



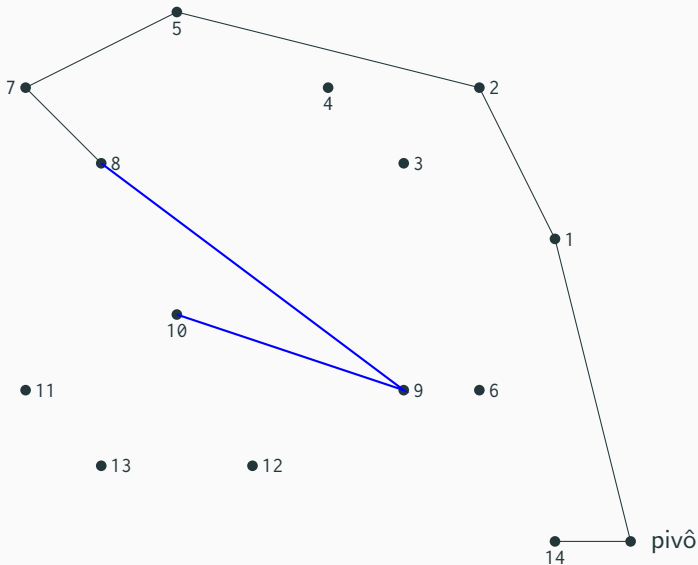
Visualização do algoritmo de Graham



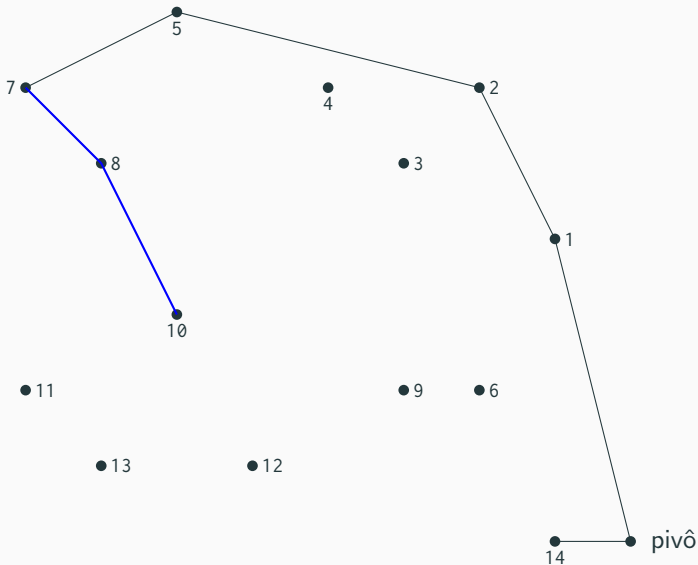
Visualização do algoritmo de Graham



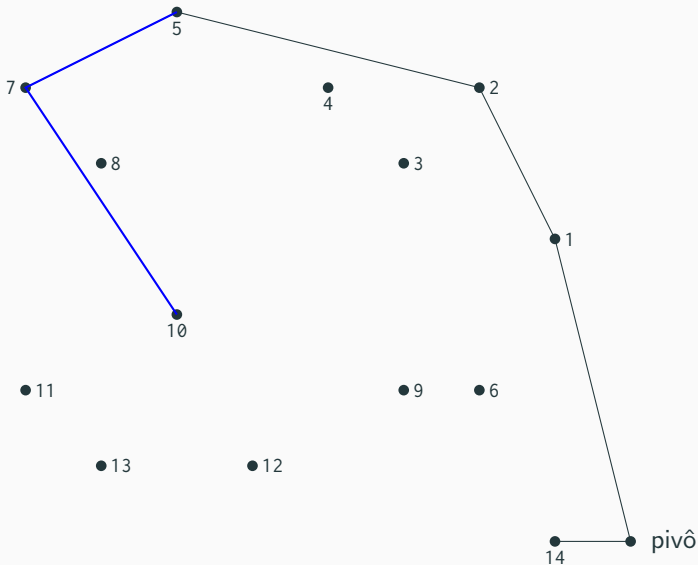
Visualização do algoritmo de Graham



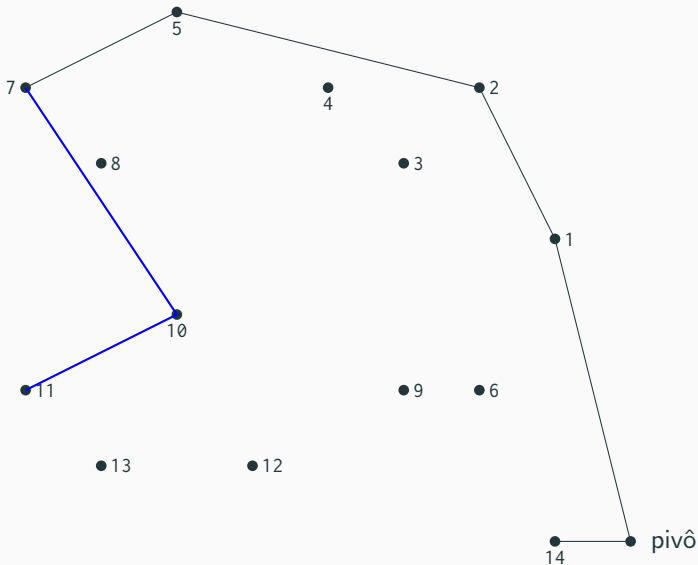
Visualização do algoritmo de Graham



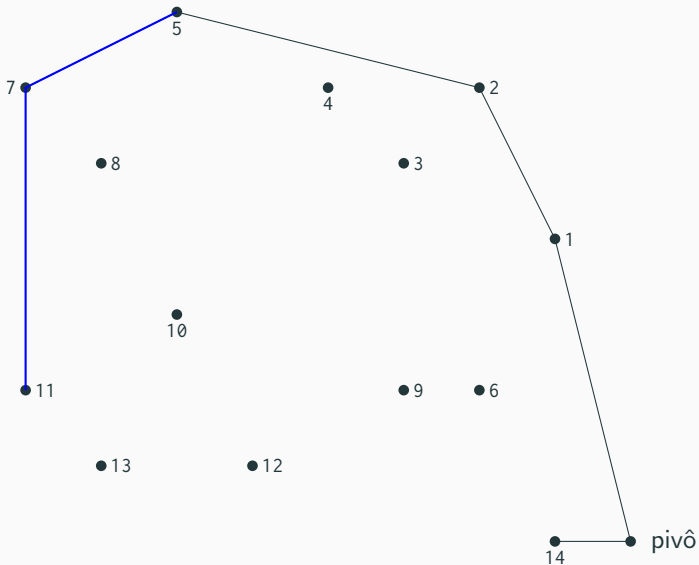
Visualização do algoritmo de Graham



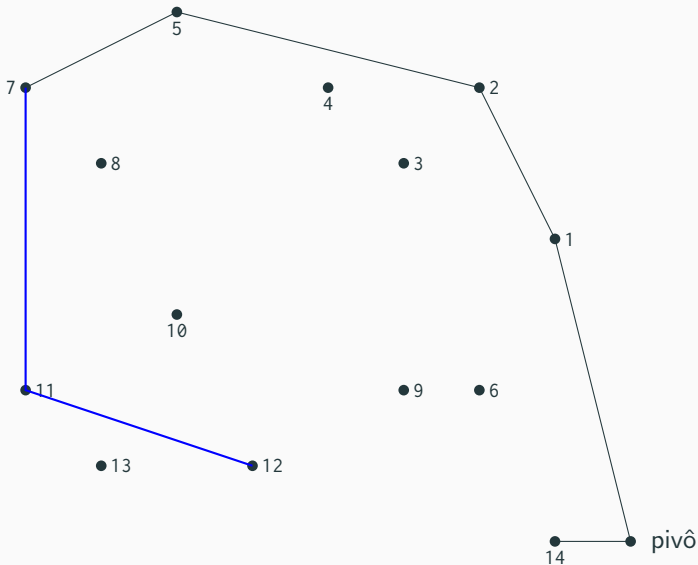
Visualização do algoritmo de Graham



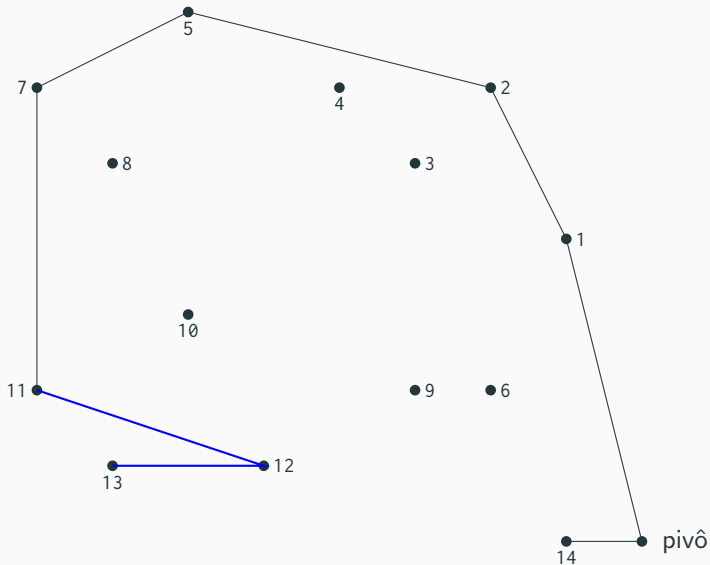
Visualização do algoritmo de Graham



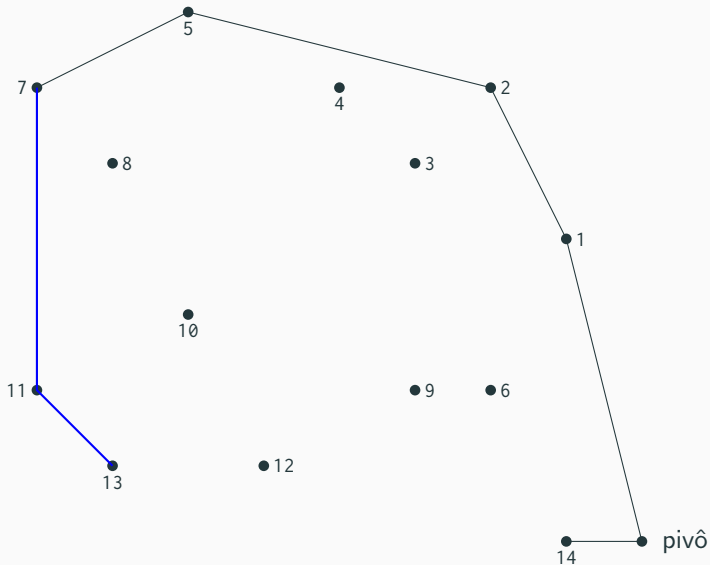
Visualização do algoritmo de Graham



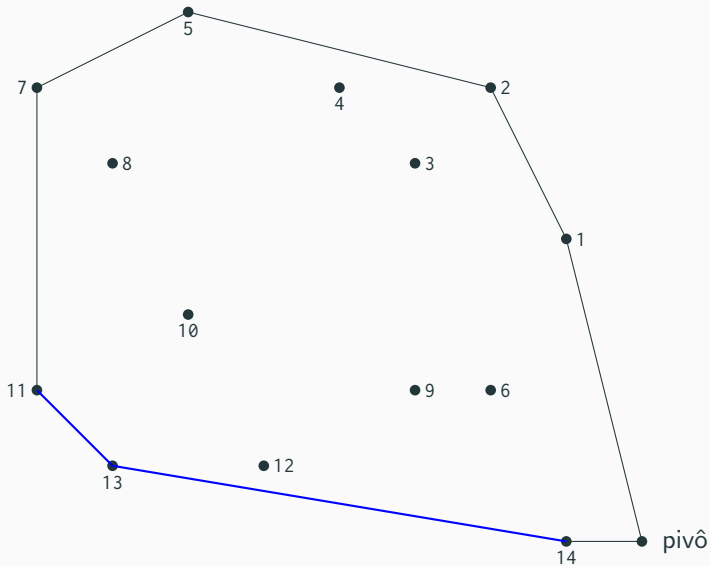
Visualização do algoritmo de Graham



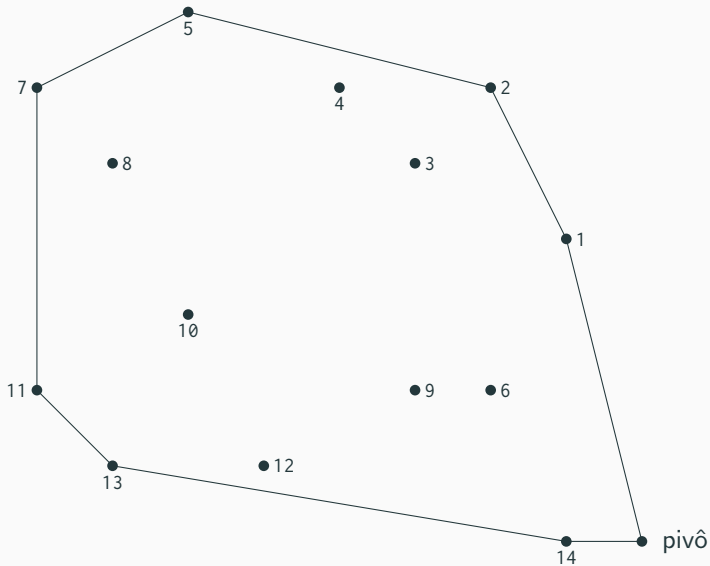
Visualização do algoritmo de Graham



Visualização do algoritmo de Graham



Visualização do algoritmo de Graham



Implementação da rotina de envoltório convexo

```
70 public:
71     static vector<Point<T>> convex_hull(const vector<Point<T>>& points)
72     {
73         vector<Point<T>> P(points);
74         auto N = P.size();
75
76         // Corner case: com 3 vértices ou menos, P é o próprio convex hull
77         if (N <= 3)
78             return P;
79
80         sort_by_angle(P);
81
82         vector<Point<T>> ch;
83         ch.push_back(P[N - 1]);
84         ch.push_back(P[0]);
85         ch.push_back(P[1]);
86
87         size_t i = 2;
88
```

Implementação da rotina de envoltório convexo

```
89     while (i < N)
90     {
91         auto j = ch.size() - 1;
92
93         if (D(ch[j - 1], ch[j], P[i]) > 0)
94             ch.push_back(P[i++]);
95         else
96             ch.pop_back();
97     }
98
99     // O envoltório é um caminho fechado: o primeiro ponto é igual
100    // ao último
101    return ch;
102 }
103 };
```

1. **GRAHAM**, R. L. *An Efficient Algorithm for Determining the Convex Hull of a Finite Planar Set*. Information Processing Letters vol. 1 (4), pg. 132-133, 1972.
2. **HALIM**, Felix; **HALIM**, Steve. *Competitive Programming 3*, 2010.
3. **O'ROURKE**, Joseph. *Computational Geometry in C*, Cambridge University Press, 2nd edition, 1998.
4. **DE BERG**, Mark. *Computational Geometry: Algorithms and Applications*, Springer, 3rd edition, 2008.
5. Wikipedia. [Graham scan](#), acesso em 09/06/2019.