Geometria Computacional

Cadeia Monótona de Andrew

Prof. Edson Alves

Faculdade UnB Gama

Sumário

Cadeia monótona de Andrew

Algoritmo de Andrew

- O algoritmo conhecido como cadeia monótona de Andrew (*Andrew's Monotone Chain Algorithm*, no original) é uma alternativa ao algoritmo de Graham para a geração do envoltório convexo

Algoritmo de Andrew

- O algoritmo conhecido como cadeia monótona de Andrew (*Andrew's Monotone Chain Algorithm*, no original) é uma alternativa ao algoritmo de Graham para a geração do envoltório convexo
- Este algoritmo foi proposto por Andrew em 1979

Algoritmo de Andrew

- O algoritmo conhecido como cadeia monótona de Andrew (*Andrew's Monotone Chain Algorithm*, no original) é uma alternativa ao algoritmo de Graham para a geração do envoltório convexo
- Este algoritmo foi proposto por Andrew em 1979
- A complexidade é a mesma do algoritmo de Graham: $O(N \log N)$

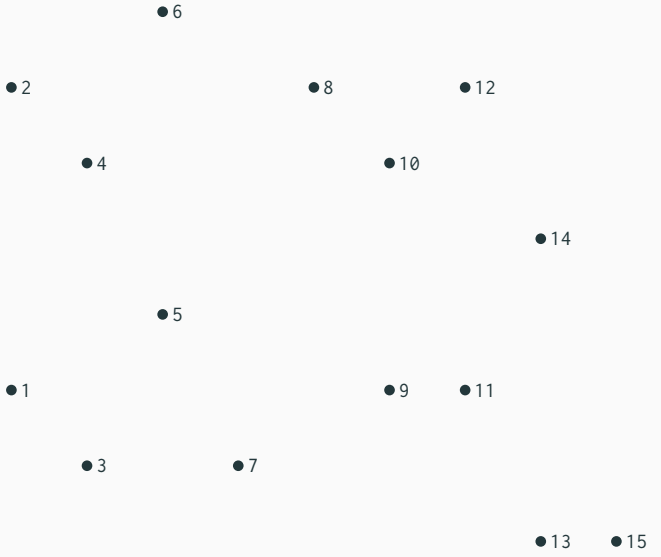
Algoritmo de Andrew

- O algoritmo conhecido como cadeia monótona de Andrew (*Andrew's Monotone Chain Algorithm*, no original) é uma alternativa ao algoritmo de Graham para a geração do envoltório convexo
- Este algoritmo foi proposto por Andrew em 1979
- A complexidade é a mesma do algoritmo de Graham: $O(N \log N)$
- Ele constrói o envoltório em duas partes: a parte superior (*upper hull*) e a parte inferior (*lower hull*)

Algoritmo de Andrew

- O algoritmo conhecido como cadeia monótona de Andrew (*Andrew's Monotone Chain Algorithm*, no original) é uma alternativa ao algoritmo de Graham para a geração do envoltório convexo
- Este algoritmo foi proposto por Andrew em 1979
- A complexidade é a mesma do algoritmo de Graham: $O(N \log N)$
- Ele constrói o envoltório em duas partes: a parte superior (*upper hull*) e a parte inferior (*lower hull*)
- Os pontos são ordenados por coordenada x e, em caso de empate, por coordenada y

Exemplo de ordenação por coordenadas



Implementação da rotina de comparação de pontos

```cpp
#include <bits/stdc++.h>

using namespace std;

template<typename T>
struct Point
{
    T x, y;

    bool operator<(const Point& P) const
    {
        return x == P.x ? y < P.y : x < P.x;
    }
};

template<typename T>
T D(const Point<T>& P, const Point<T>& Q, const Point<T>& R)
{
    return (P.x * Q.y + P.y * R.x + Q.x * R.y) - (R.x * Q.y + R.y * P.x + Q.x * P.y);
}
```

Geração do envoltório convexo

- O envoltório convexo é gerado de forma semelhante ao procedimento usado no algoritmo de Graham

Geração do envoltório convexo

- O envoltório convexo é gerado de forma semelhante ao procedimento usado no algoritmo de Graham
- O ponto de partida é o ponto mais à esquerda, com menor coordenada y

Geração do envoltório convexo

- O envoltório convexo é gerado de forma semelhante ao procedimento usado no algoritmo de Graham
- O ponto de partida é o ponto mais à esquerda, com menor coordenada y
- O *lower hull* é gerado empilhando os pontos de acordo com a ordenação, desde que o novo ponto e os dois últimos elementos da pilha mantenham a orientação anti-horária, ou que a pilha tenha menos do que dois elementos

Geração do envoltório convexo

- O envoltório convexo é gerado de forma semelhante ao procedimento usado no algoritmo de Graham
- O ponto de partida é o ponto mais à esquerda, com menor coordenada y
- O *lower hull* é gerado empilhando os pontos de acordo com a ordenação, desde que o novo ponto e os dois últimos elementos da pilha mantenham a orientação anti-horária, ou que a pilha tenha menos do que dois elementos
- Para gerar o *upper hull*, é preciso começar do ponto mais à direita, com maior coordenada y

Geração do envoltório convexo

- O envoltório convexo é gerado de forma semelhante ao procedimento usado no algoritmo de Graham
- O ponto de partida é o ponto mais à esquerda, com menor coordenada y
- O *lower hull* é gerado empilhando os pontos de acordo com a ordenação, desde que o novo ponto e os dois últimos elementos da pilha mantenham a orientação anti-horária, ou que a pilha tenha menos do que dois elementos
- Para gerar o *upper hull*, é preciso começar do ponto mais à direita, com maior coordenada y
- A rotina é idêntica à usado no *lower hull*: basta processar os pontos do maior para o menor, de acordo com a ordenação

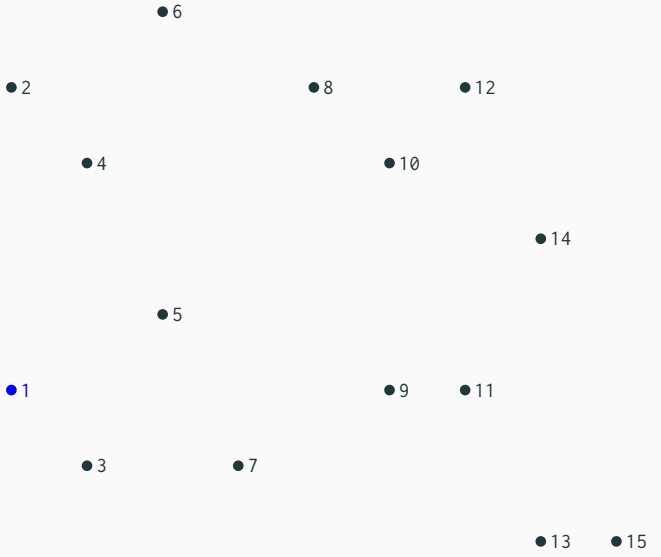
Geração do envoltório convexo

- O envoltório convexo é gerado de forma semelhante ao procedimento usado no algoritmo de Graham
- O ponto de partida é o ponto mais à esquerda, com menor coordenada y
- O *lower hull* é gerado empilhando os pontos de acordo com a ordenação, desde que o novo ponto e os dois últimos elementos da pilha mantenham a orientação anti-horária, ou que a pilha tenha menos do que dois elementos
- Para gerar o *upper hull*, é preciso começar do ponto mais à direita, com maior coordenada y
- A rotina é idêntica à usado no *lower hull*: basta processar os pontos do maior para o menor, de acordo com a ordenação
- Ao final as duas partes devem ser unidas

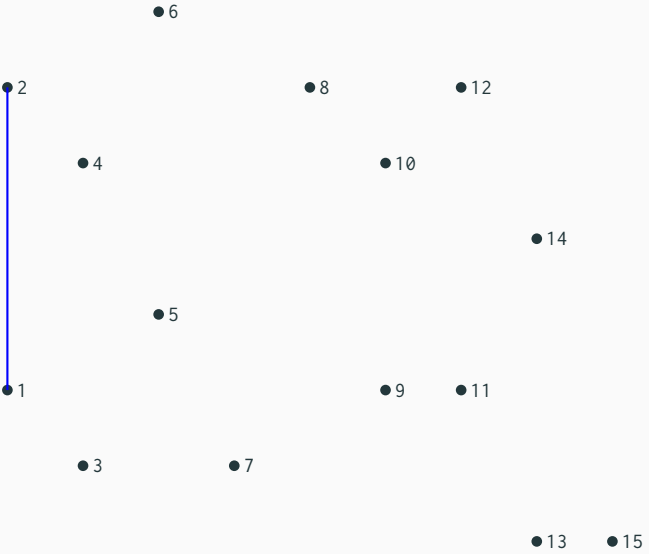
Geração do envoltório convexo

- O envoltório convexo é gerado de forma semelhante ao procedimento usado no algoritmo de Graham
- O ponto de partida é o ponto mais à esquerda, com menor coordenada y
- O *lower hull* é gerado empilhando os pontos de acordo com a ordenação, desde que o novo ponto e os dois últimos elementos da pilha mantenham a orientação anti-horária, ou que a pilha tenha menos do que dois elementos
- Para gerar o *upper hull*, é preciso começar do ponto mais à direita, com maior coordenada y
- A rotina é idêntica à usado no *lower hull*: basta processar os pontos do maior para o menor, de acordo com a ordenação
- Ao final as duas partes devem ser unidas
- O ponto final do *lower hull* deve ser descartado, uma vez que é idêntico ao ponto inicial do *upper hull*

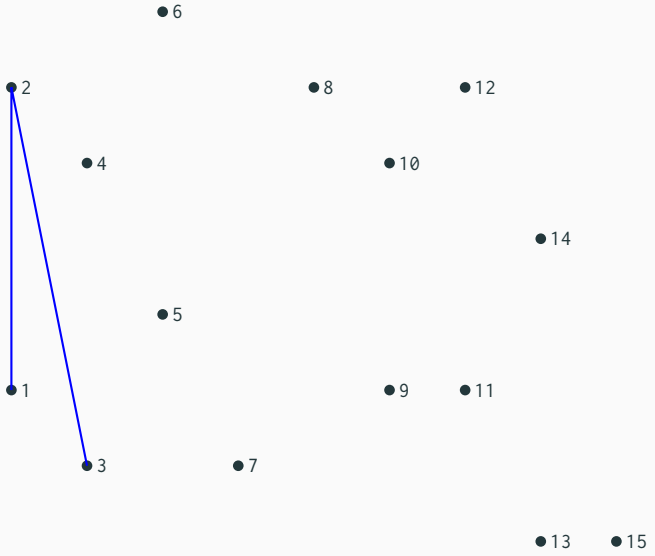
Geração do lower hull



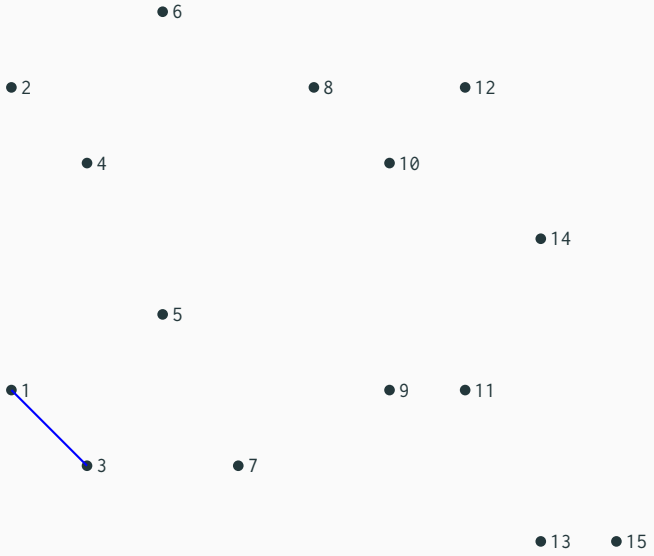
Geração do lower hull



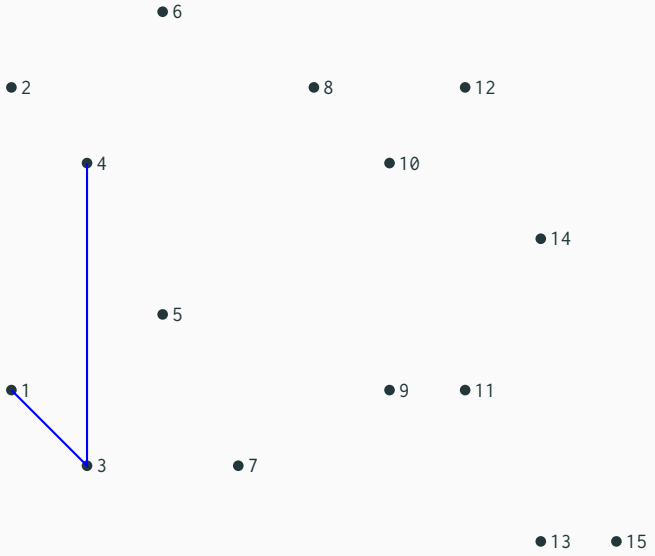
Geração do lower hull



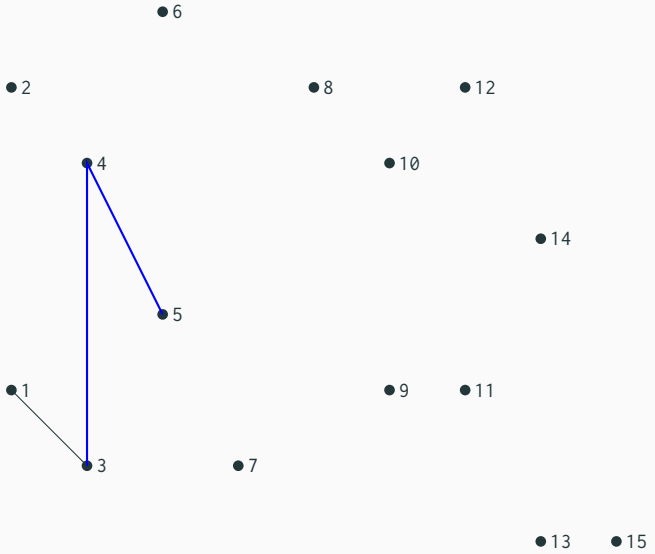
Geração do lower hull



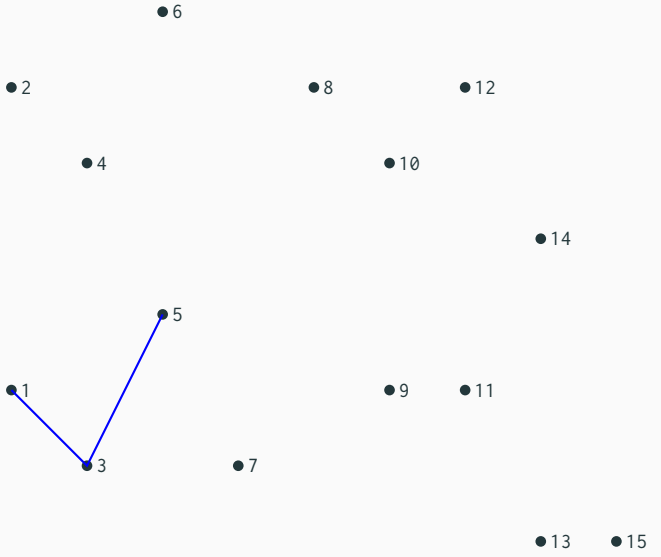
Geração do lower hull



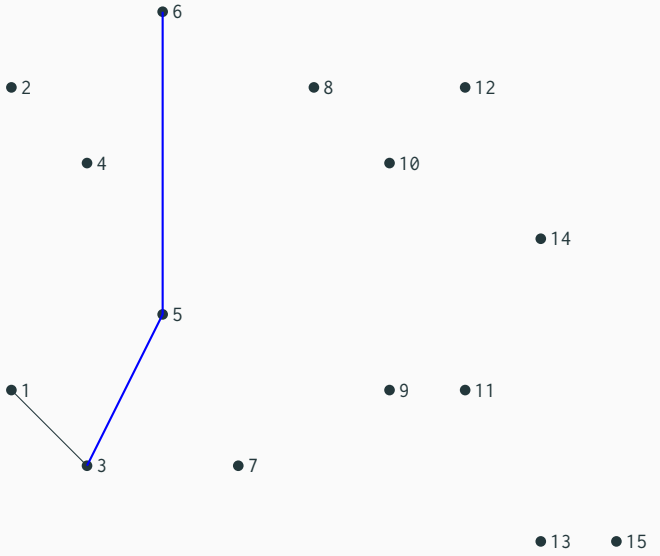
Geração do lower hull



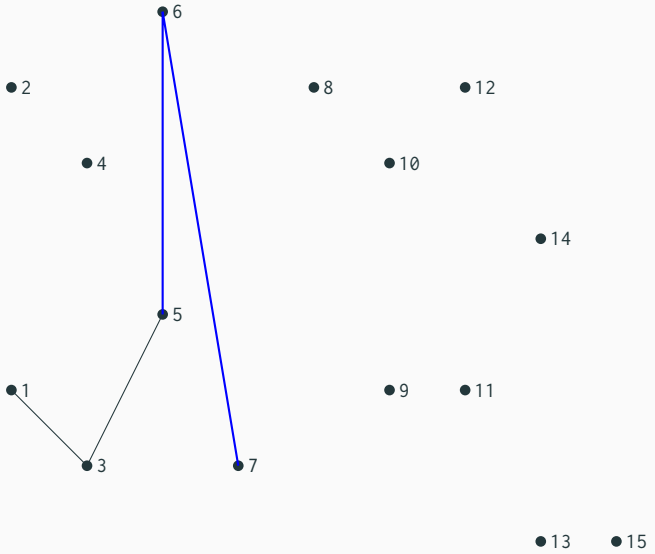
Geração do lower hull



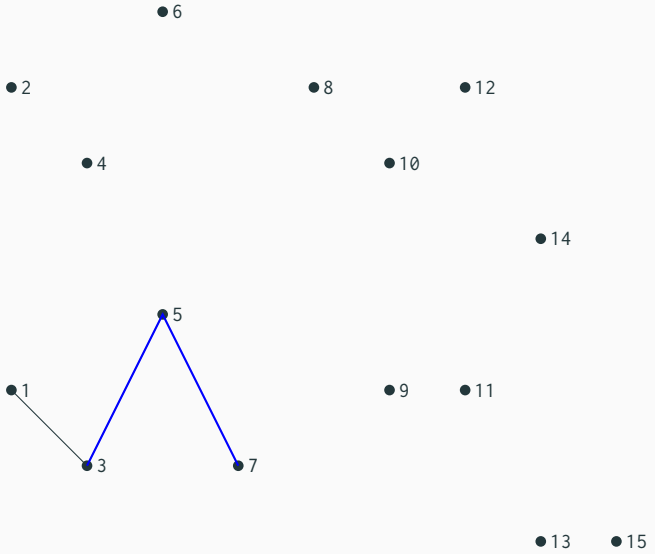
Geração do lower hull



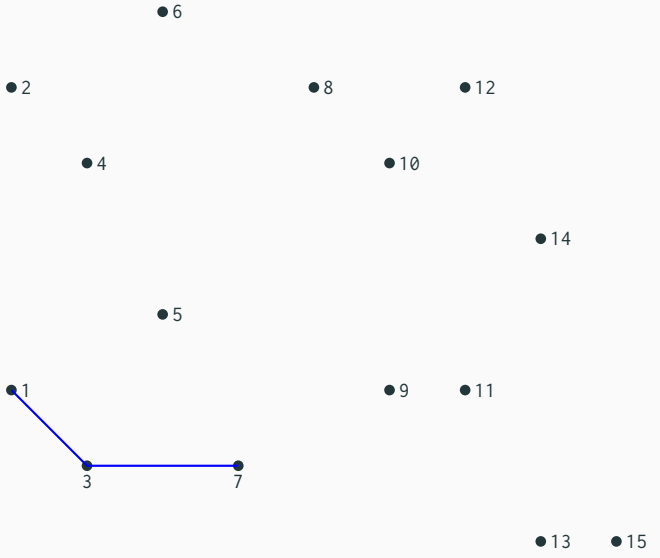
Geração do lower hull



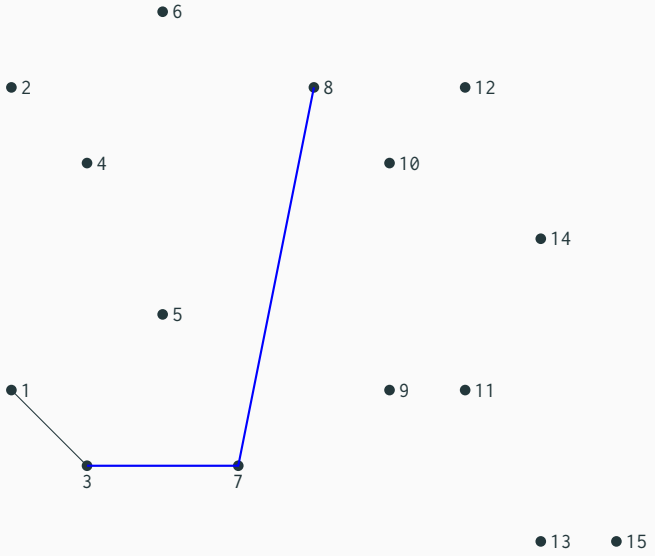
Geração do lower hull



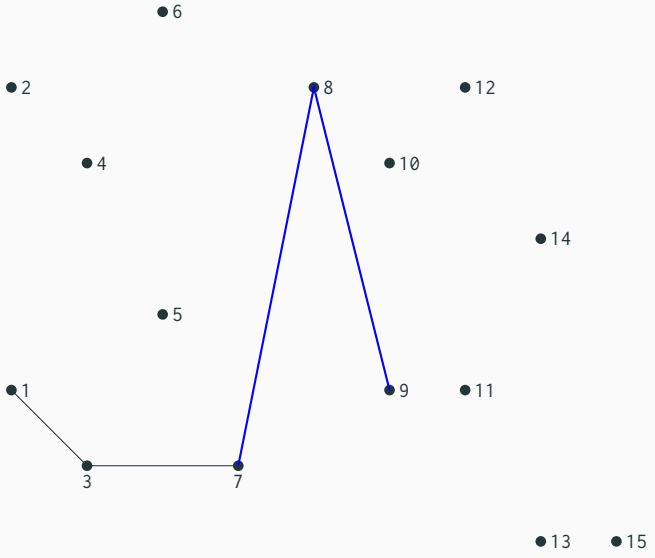
Geração do lower hull



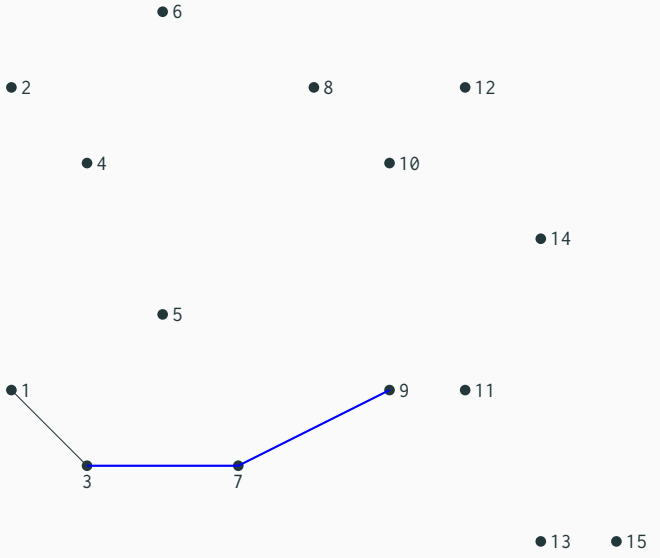
Geração do lower hull



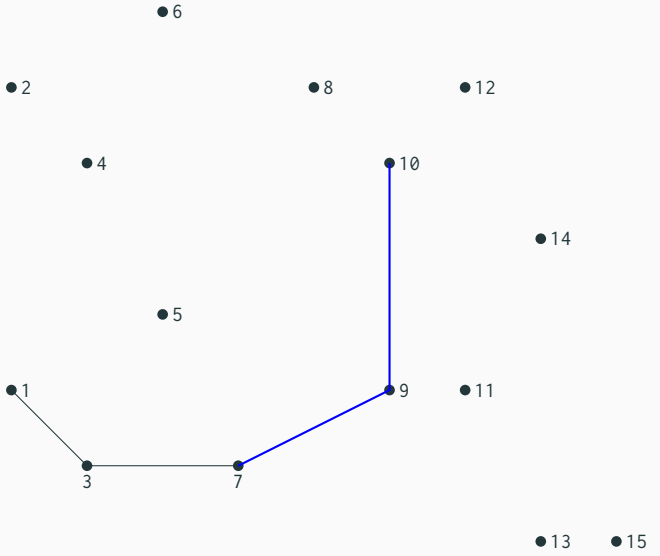
Geração do lower hull



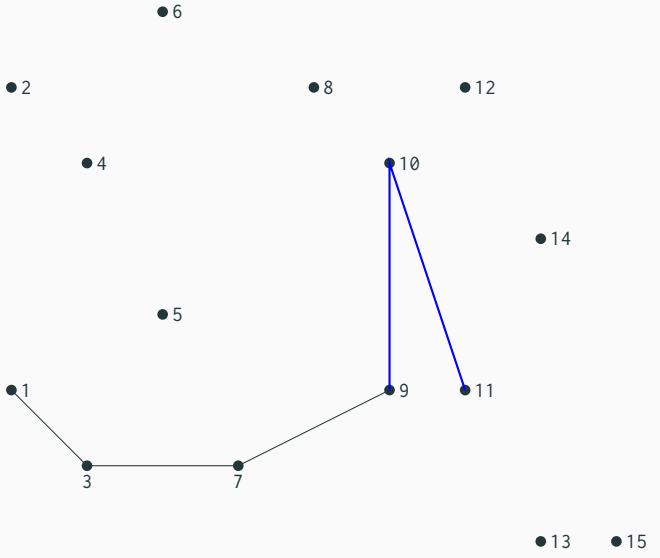
Geração do lower hull



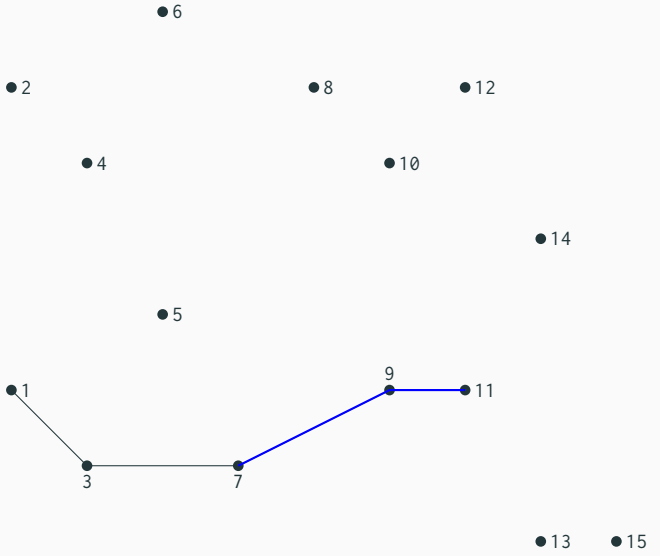
Geração do lower hull



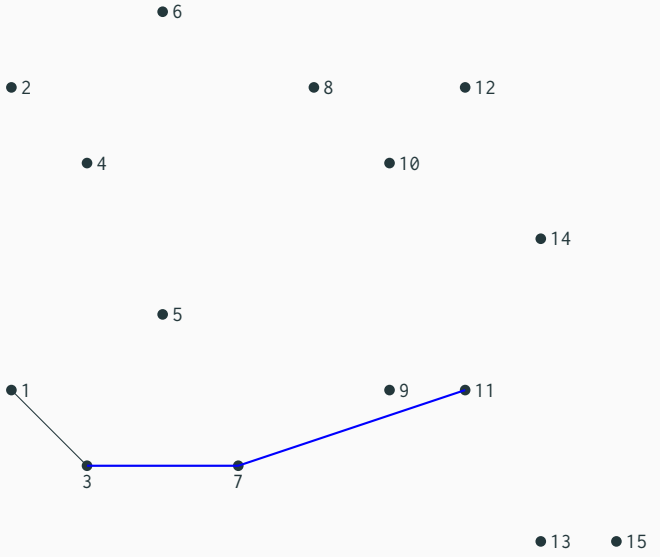
Geração do lower hull



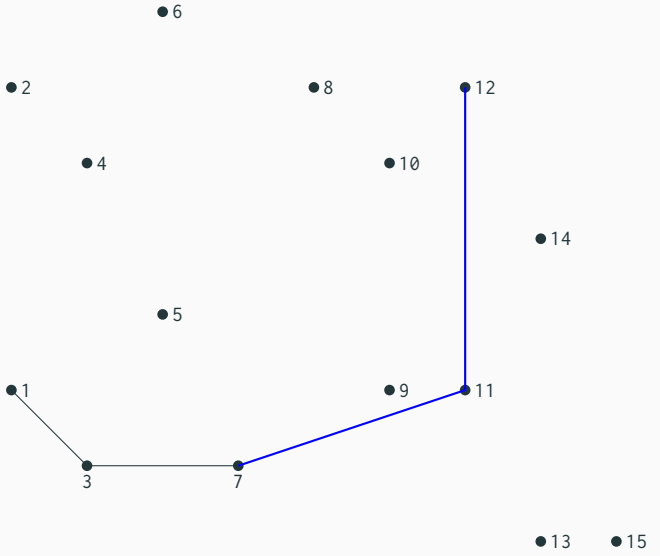
Geração do lower hull



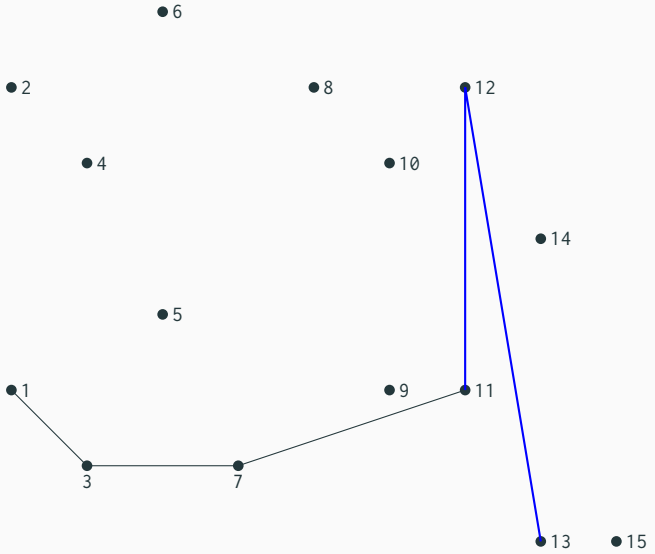
Geração do lower hull



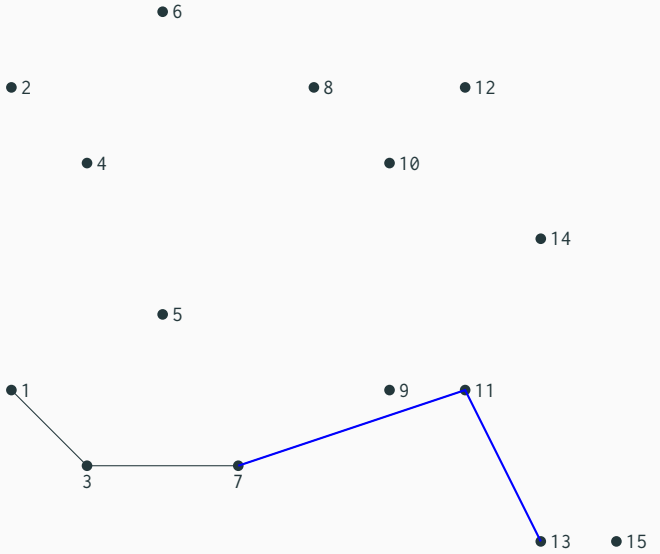
Geração do lower hull



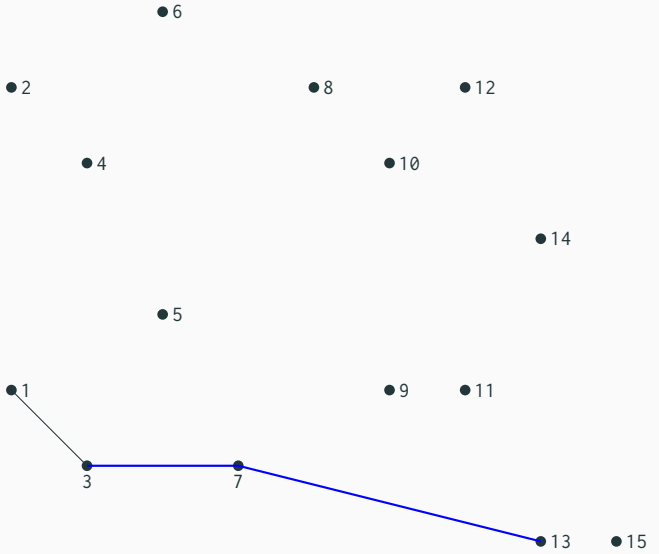
Geração do lower hull



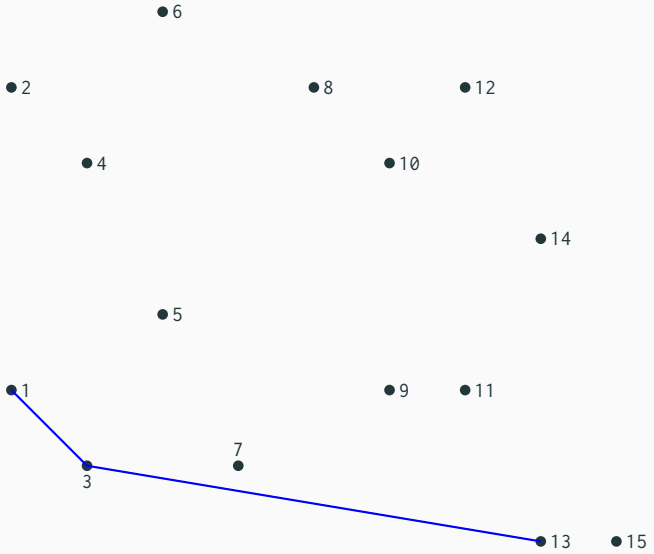
Geração do lower hull



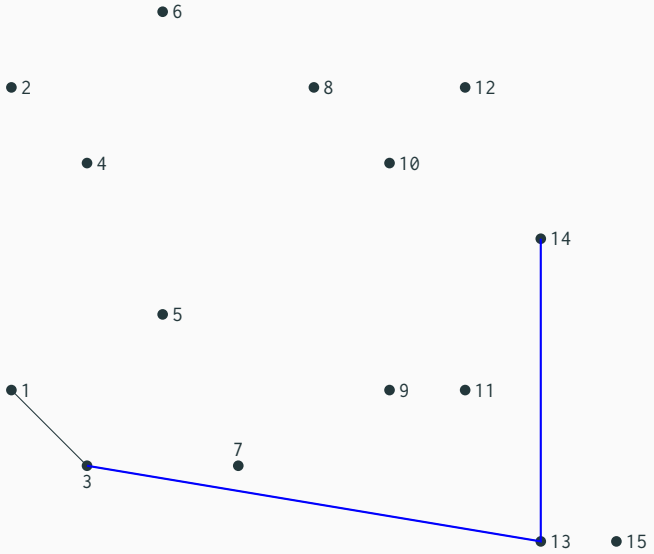
Geração do lower hull



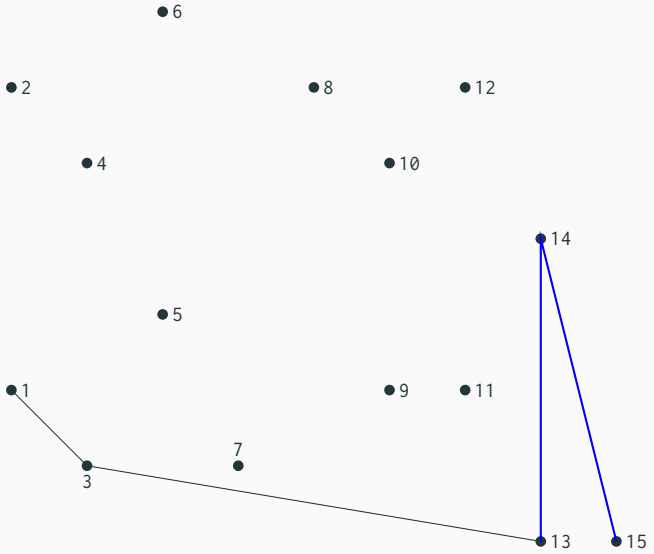
Geração do lower hull



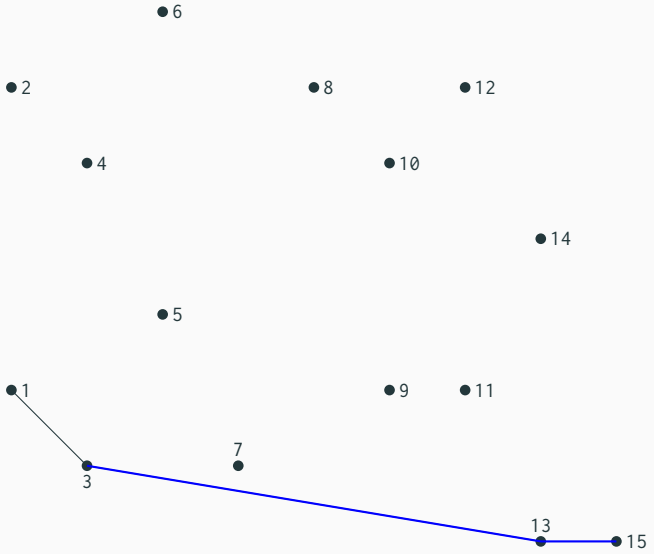
Geração do lower hull



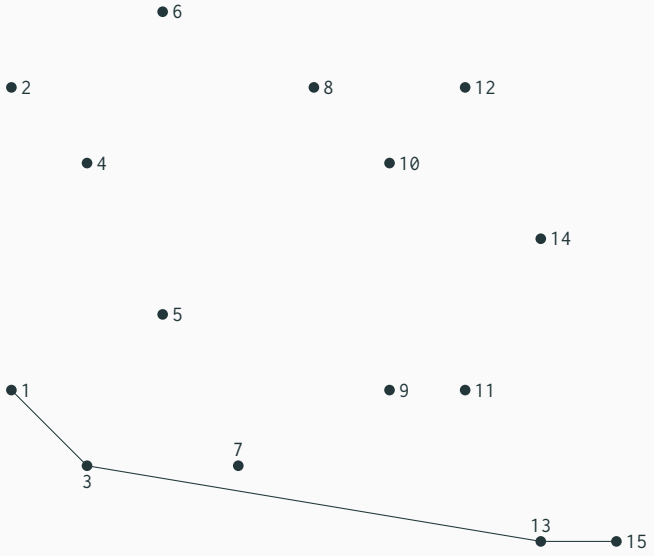
Geração do lower hull



Geração do lower hull



Geração do lower hull



Implementação da geração do envoltório convexo

```
22 template<typename T>
23 vector<Point<T>> make_hull(const vector<Point<T>>& points, vector<Point<T>>& hull)
24 {
25     for (const auto& p : points)
26     {
27         auto size = hull.size();
28 
29         while (size >= 2 and D(hull[size - 2], hull[size - 1], p) <= 0)
30         {
31             hull.pop_back();
32             size = hull.size();
33         }
34 
35         hull.push_back(p);
36     }
37 
38     return hull;
39 }
```

Implementação da geração do envoltório convexo

```
42 vector<Point<T>> monotone_chain(const vector<Point<T>>& points)
43 {
44     vector<Point<T>> P(points);
45
46     sort(P.begin(), P.end());
47
48     vector<Point<T>> lower, upper;
49
50     lower = make_hull(P, lower);
51
52     reverse(P.begin(), P.end());
53
54     upper = make_hull(P, upper);
55
56     lower.pop_back();
57     lower.insert(lower.end(), upper.begin(), upper.end());
58
59     return lower;
60 }
```

Referências

1. **ANDREW**, A. M. *Another Efficient Algorithm for Convex Hulls in Two Dimensions*. Information Processing Letters vol. 9, pg. 216-219, 1979.
2. **DE BERG**, Mark. *Computational Geometry: Algorithms and Applications*, Springer, 3rd edition, 2008.
3. **HALIM**, Felix; **HALIM**, Steve. *Competitive Programming 3*, 2010.
4. **LAAKSONEN**, Antti. *Competitive Programmer's Handbook*, 2018 (*Open Access*).
5. **O'ROURKE**, Joseph. *Computational Geometry in C*, Cambridge University Press, 2nd edition, 1998.