Geometria Computacional

Sweep line: algoritmos envolvendo intervalos

Prof. Edson Alves

2019

Faculdade UnB Gama

Sumário

Sweep line algorithms

Definição

- Também denominados *sweep plane algorithms*, estes algoritmos formam um paradigma de solução de problemas geométricos
- A ideia principal é mover uma reta (geralmente vertical) se movendo pelo espaço e parando em pontos chaves
- A cada parada, são processados os pontos ou que interceptam a reta ou que estejam em sua vizinhança
- Outra interpretação possível é interpretar um conjunto de pontos como eventos, e estes eventos devem ser processados em ordem crescente de coordenada x ou y
- Exemplos de algoritmos desta categoria são o algoritmo de Graham e a cadeia monótona de Andrew

Interseção de intervalos

Interseção de intervalos

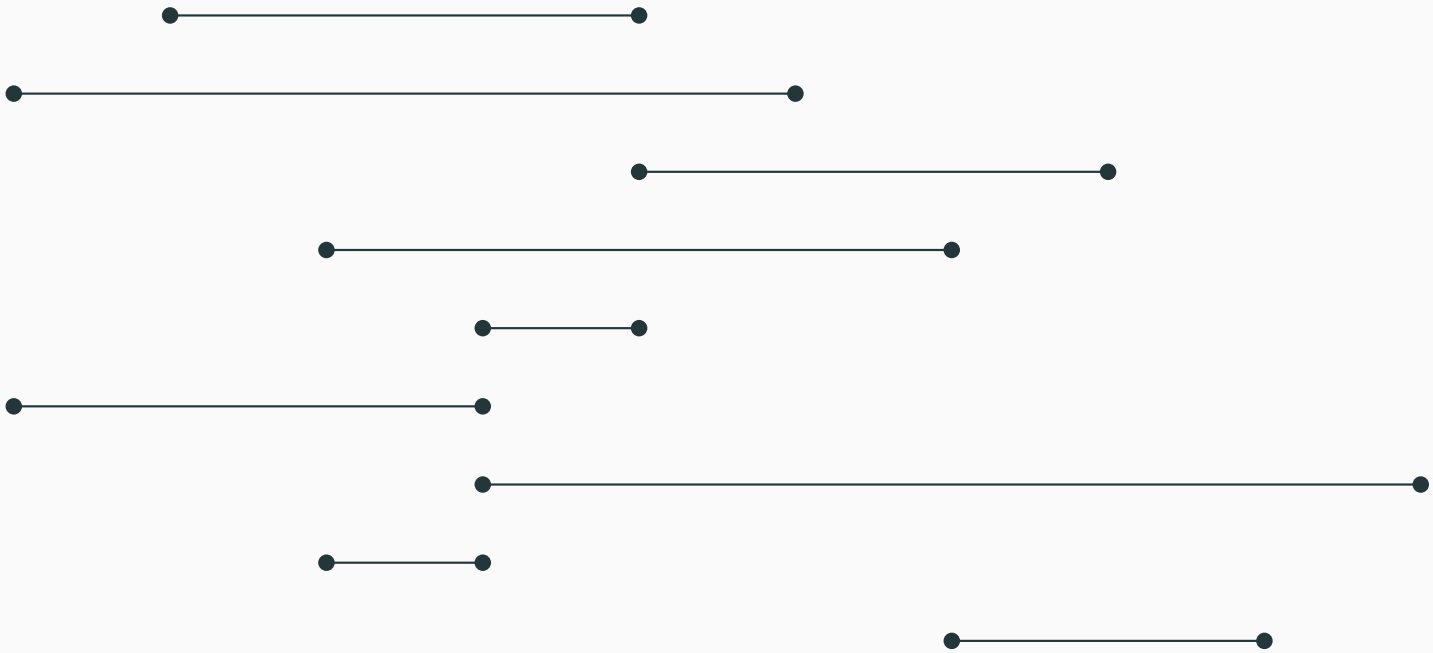
- Um problema que pode ser resolvido usando o paradigma *sweep line* é o de contabilizar, dentre um conjunto de intervalos $I_i = (a_i, b_i)$, o tamanho do maior subconjunto S destes intervalos tal que $I_j \cap I_k \neq \emptyset$ para todo par $I_j, I_k \in S$
- Uma aplicação prática deste problema seria: cada intervalo representa o início e o fim de um espetáculo que acontecerá em determinado dia. Qual é o número máximo de espetáculos que acontecerão simultaneamete?
- A solução é criar, para cada intervalo, dois eventos: um evento de ínicio do espetáculo $(a_i, 1)$ e um evento de final $(b_i, 0)$
- Uma vez ordenados estes eventos em ordem lexicográfica (primeiro por coordenada x, depois por coordenada y), basta processar todos eles, um por vez

Interseção de intervalos

- Um evento de início incrementa o número de eventos em andamento, o evento de fim decrementa
- Com a representação de eventos escolhida, os intervalos (a, b) e (b, c) não tem interseção
- A complexidade deste algoritmo é $O(N \log N)$, por conta da ordenação, pois cada ponto será processado uma única vez
- A representação dos eventos pode ser modificada, de tal modo que é possível identificar quais são os intervalos simultâneos
- Basta fazer (a_i, i) e $(b_i, -i)$ e manter os índices dos intervalos em exibição em um conjuntos, removendo-os a cada evento de encerramento
- Veja que, nesta represenção, os intervalos devem ser numerados a partir de 1, pois o zero geraria ambiguidade

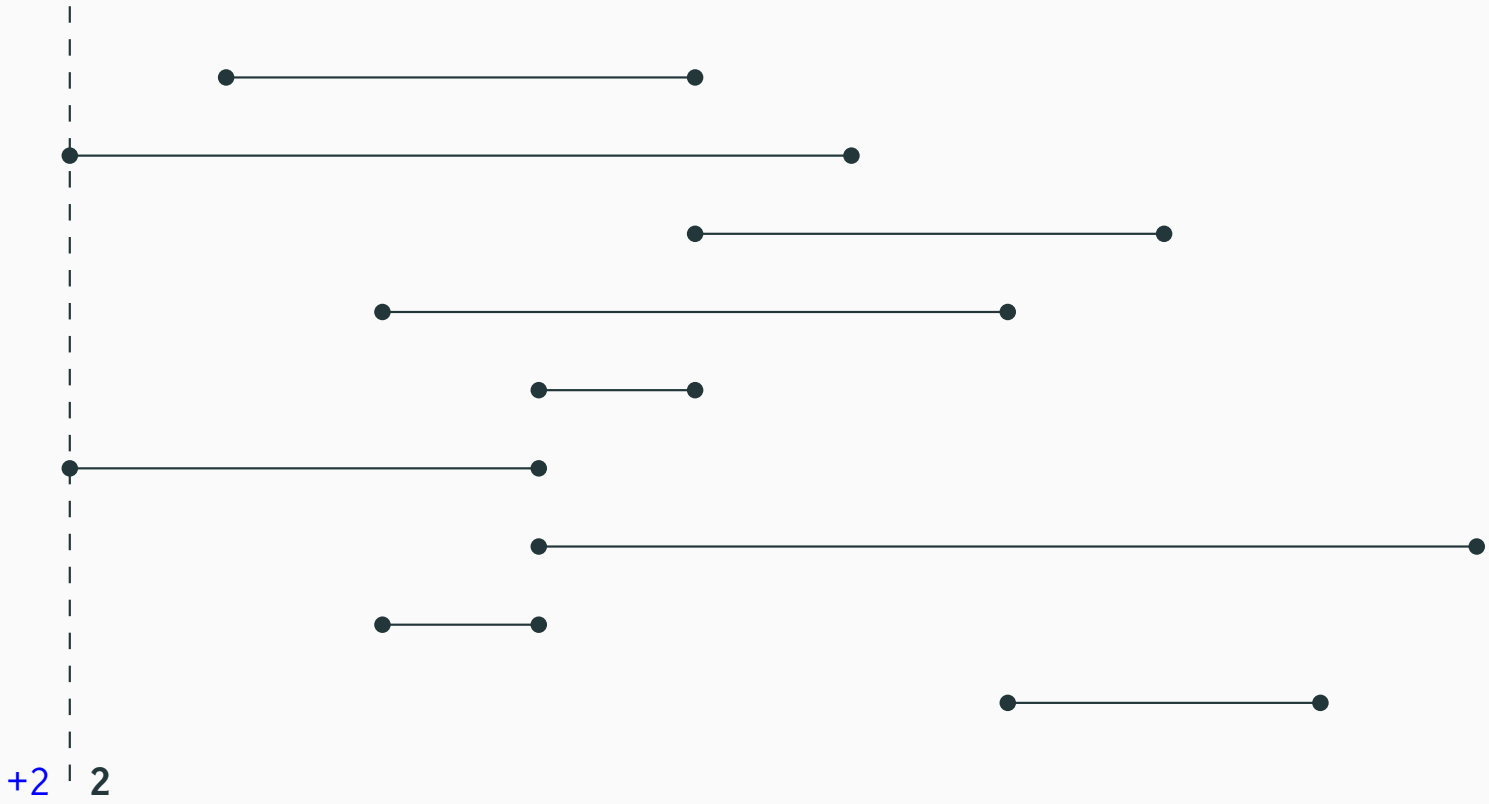
Visualização do algoritmo de interseção de intervalos

Intervalos: $(2,5), (1,6), (5,8), (3,7), (4,5), (1,4), (4,10), (3,4), (7,9)$



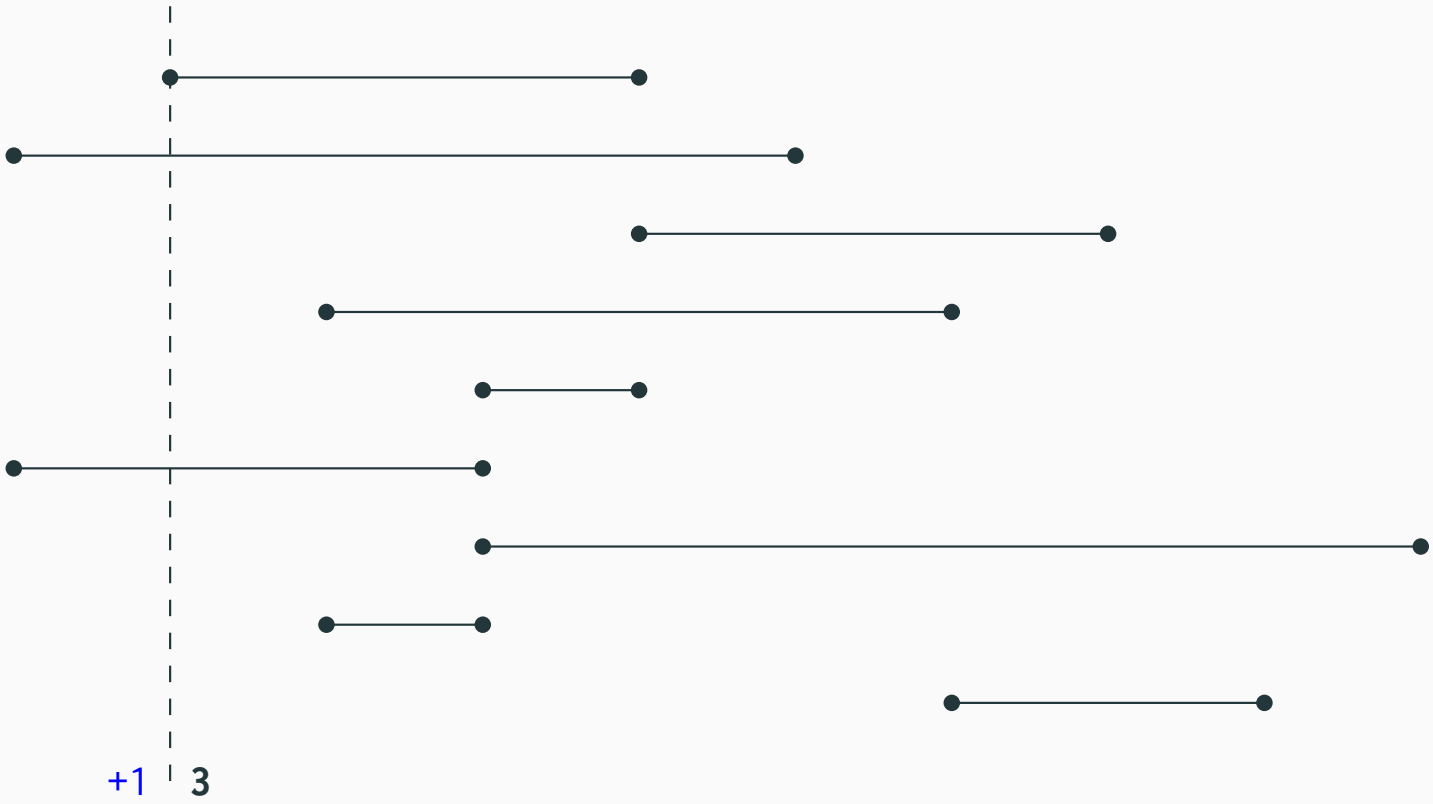
Visualização do algoritmo de interseção de intervalos

Intervalos: $(2,5), (1,6), (5,8), (3,7), (4,5), (1,4), (4,10), (3,4), (7,9)$



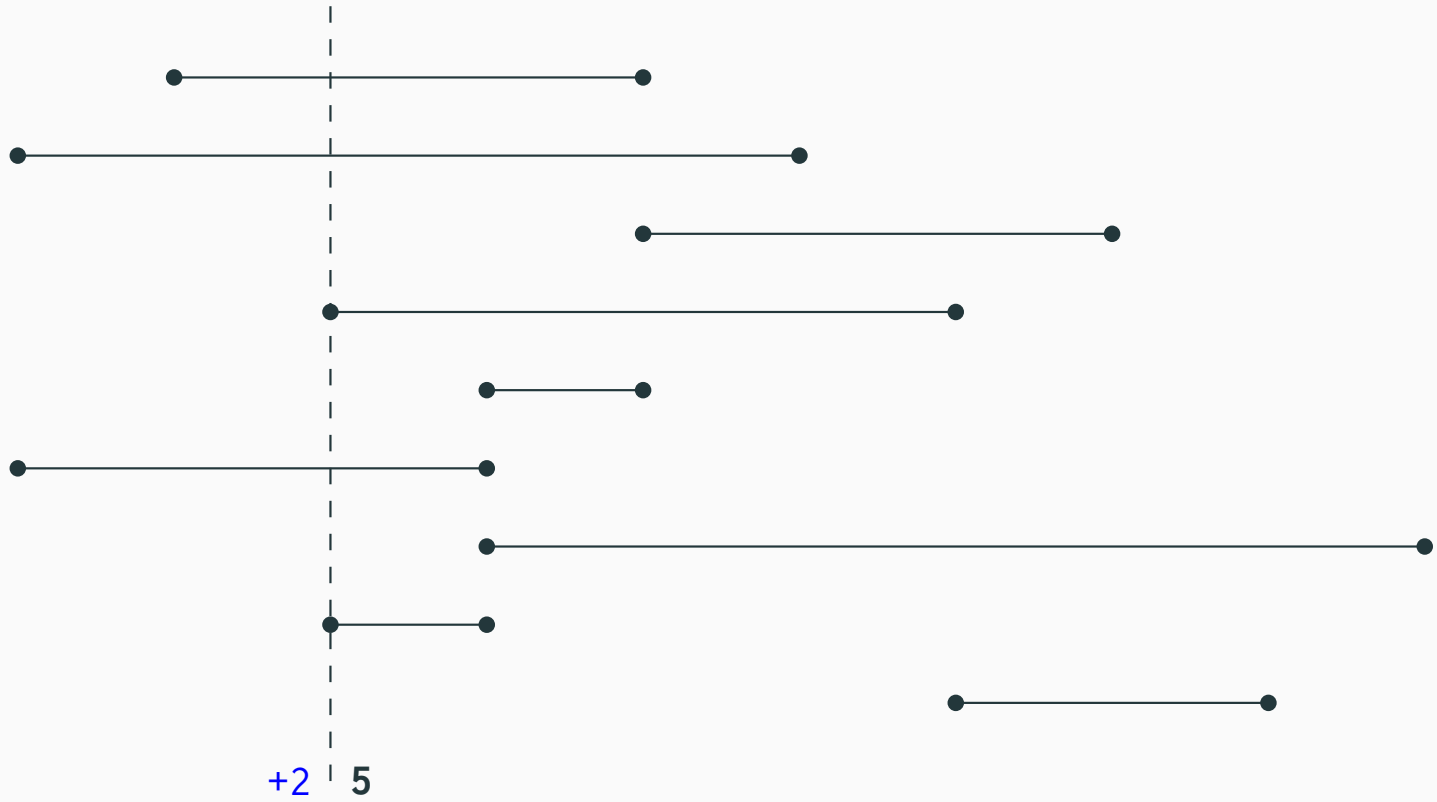
Visualização do algoritmo de interseção de intervalos

Intervalos: $(2,5), (1,6), (5,8), (3,7), (4,5), (1,4), (4,10), (3,4), (7,9)$



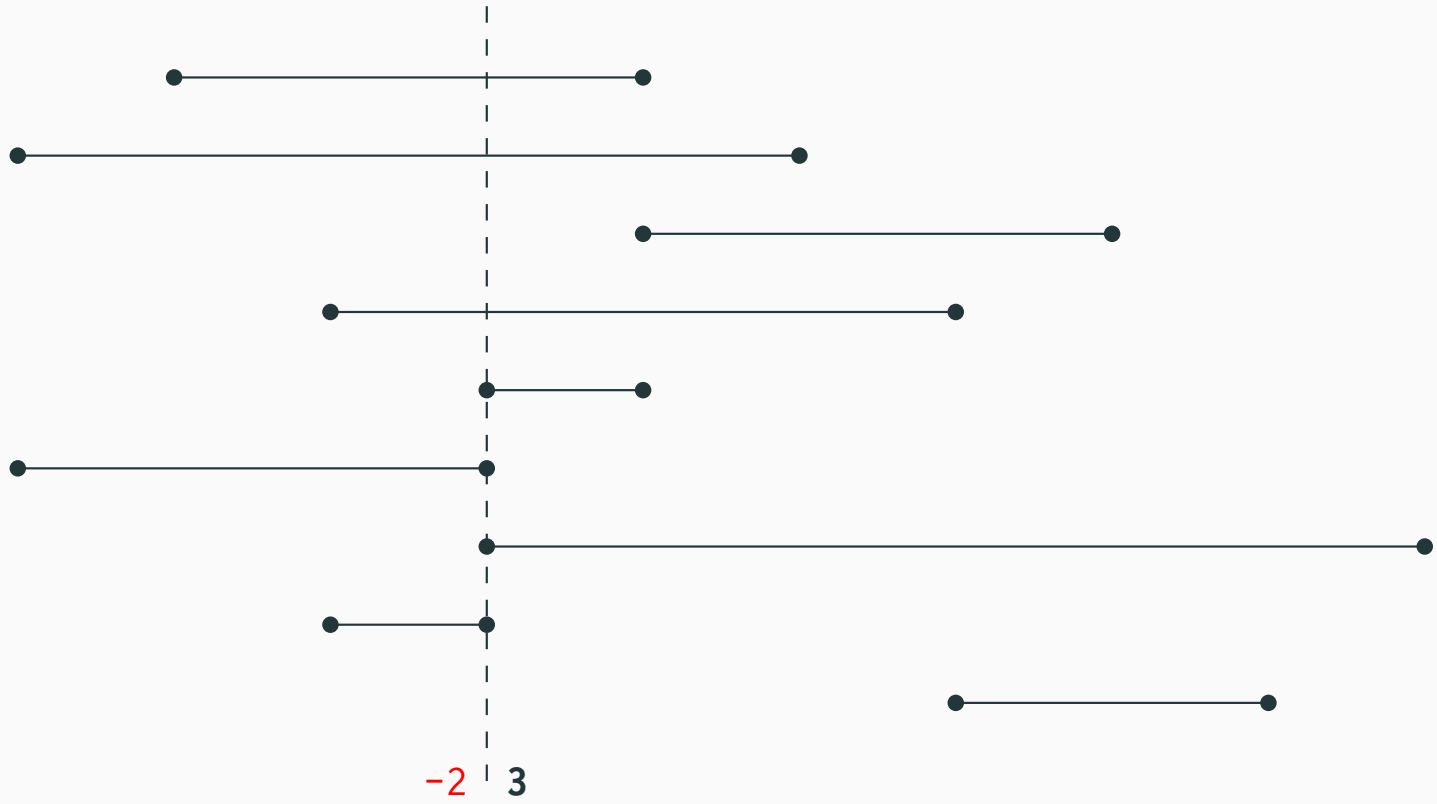
Visualização do algoritmo de interseção de intervalos

Intervalos: $(2,5), (1,6), (5,8), (3,7), (4,5), (1,4), (4,10), (3,4), (7,9)$



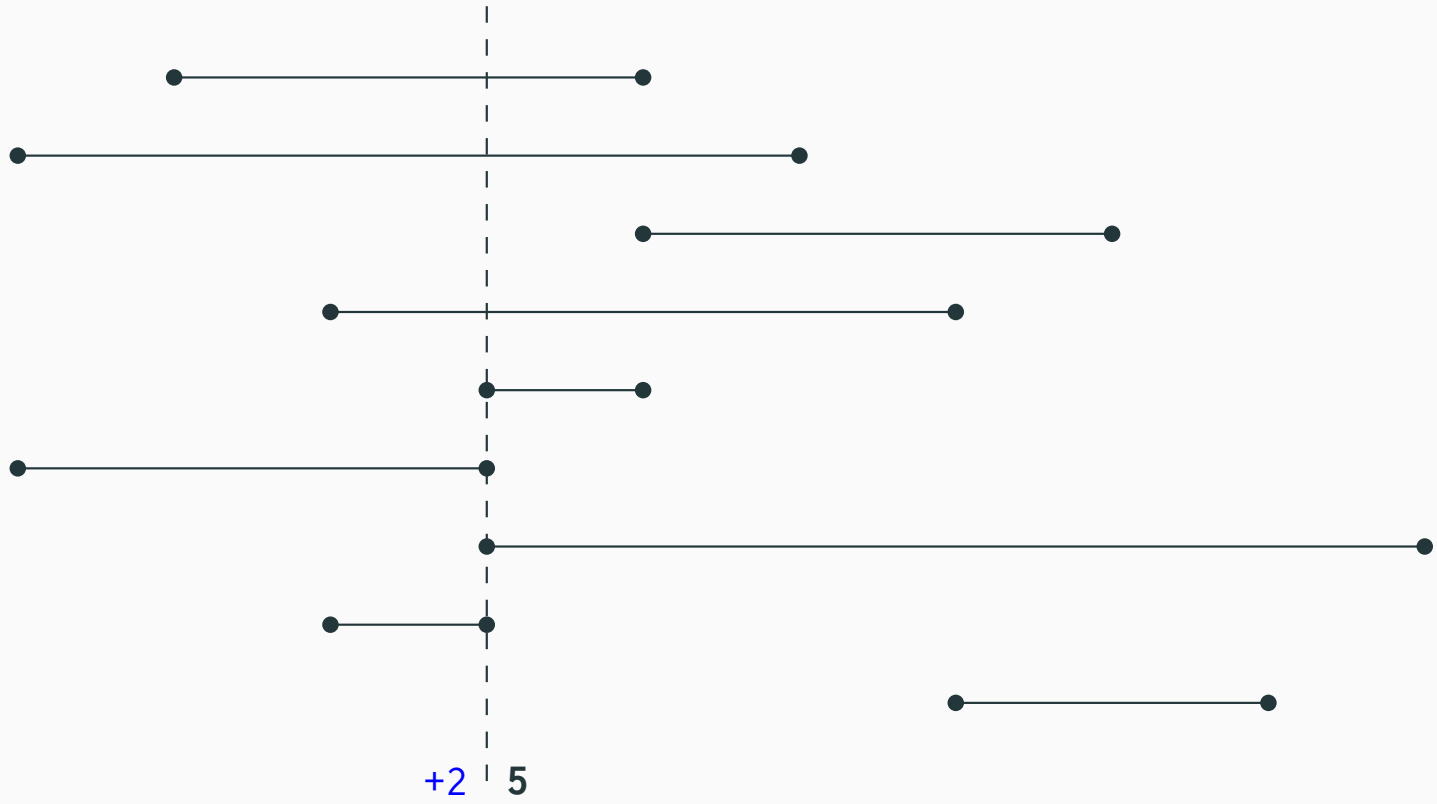
Visualização do algoritmo de interseção de intervalos

Intervalos: $(2,5), (1,6), (5,8), (3,7), (4,5), (1,4), (4,10), (3,4), (7,9)$



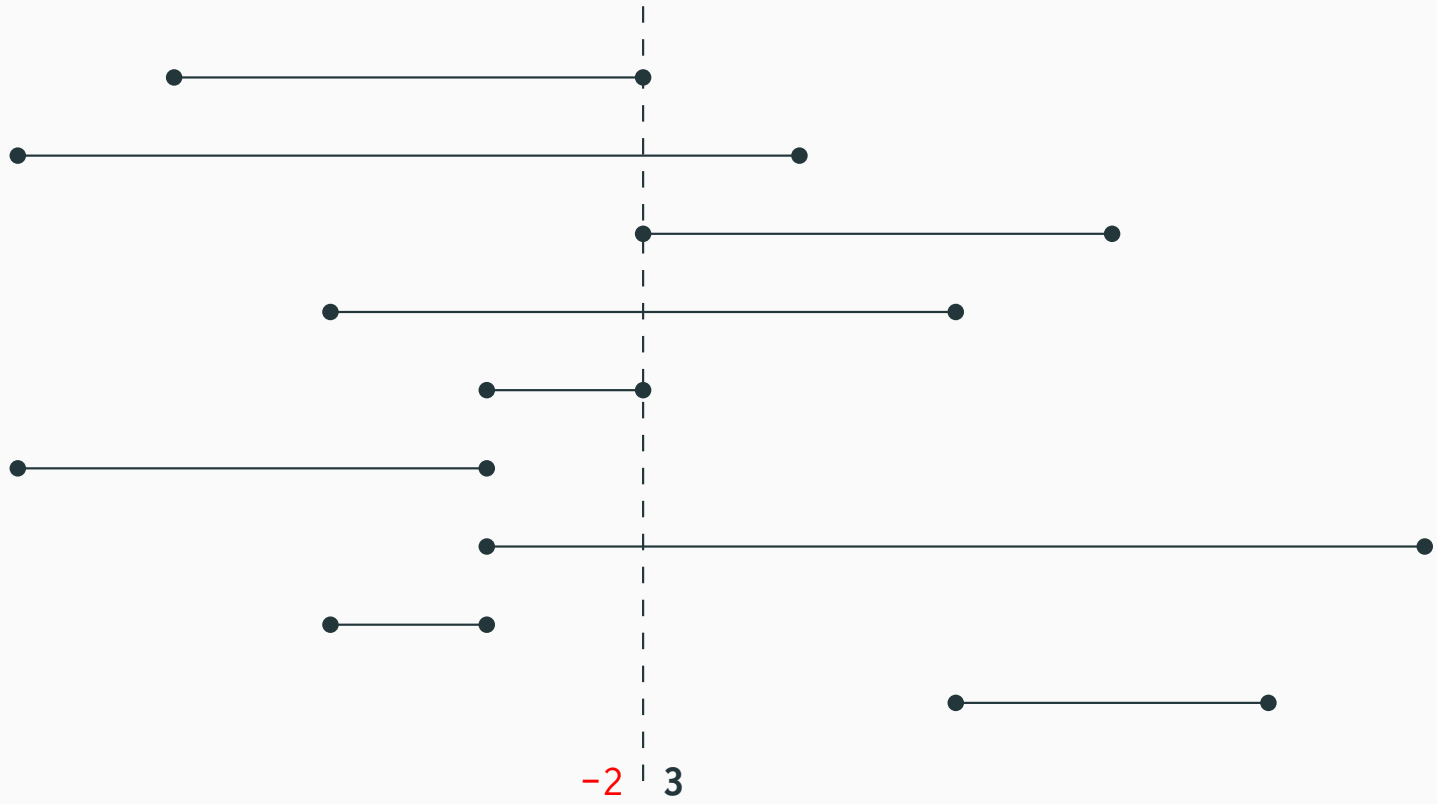
Visualização do algoritmo de interseção de intervalos

Intervalos: $(2,5), (1,6), (5,8), (3,7), (4,5), (1,4), (4,10), (3,4), (7,9)$



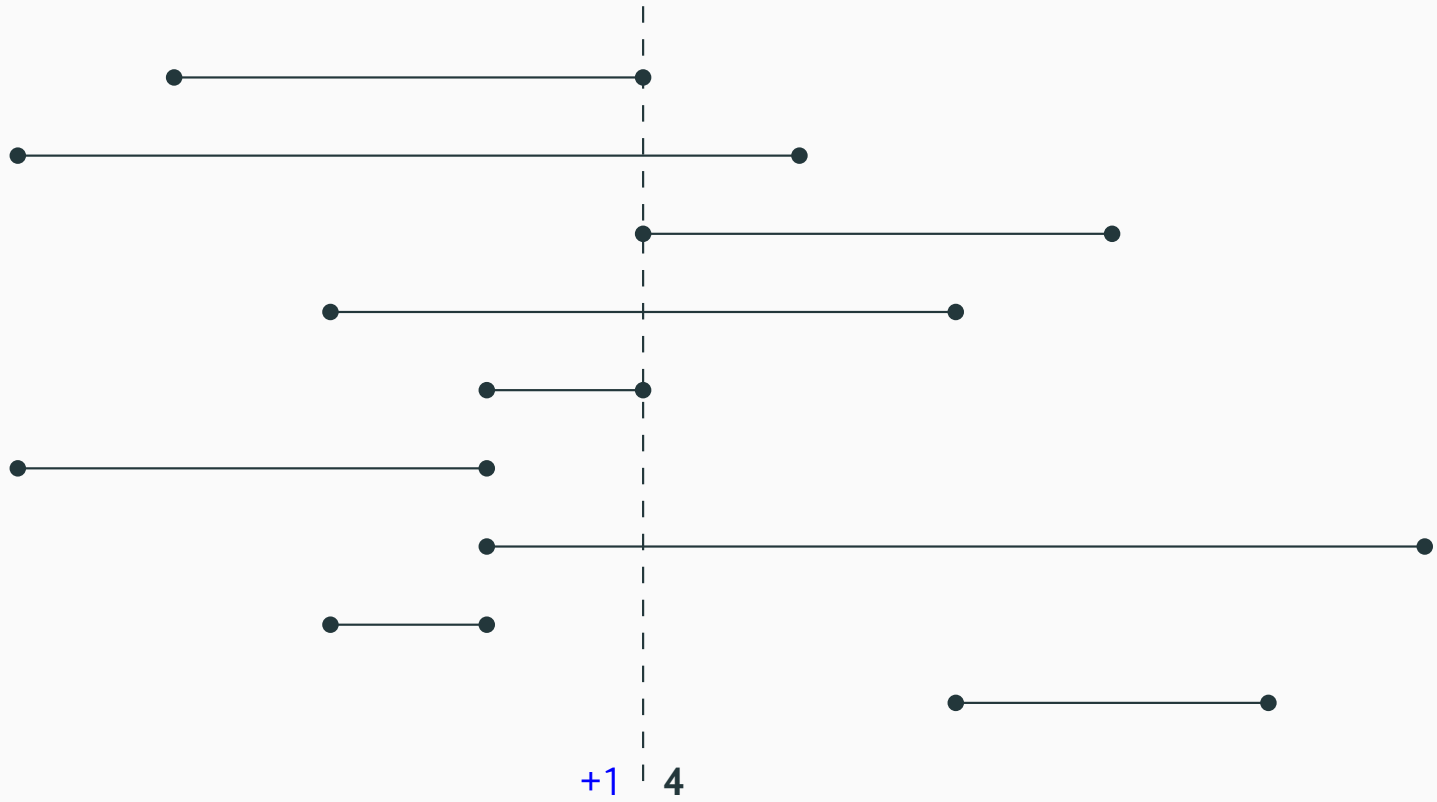
Visualização do algoritmo de interseção de intervalos

Intervalos: $(2,5), (1,6), (5,8), (3,7), (4,5), (1,4), (4,10), (3,4), (7,9)$



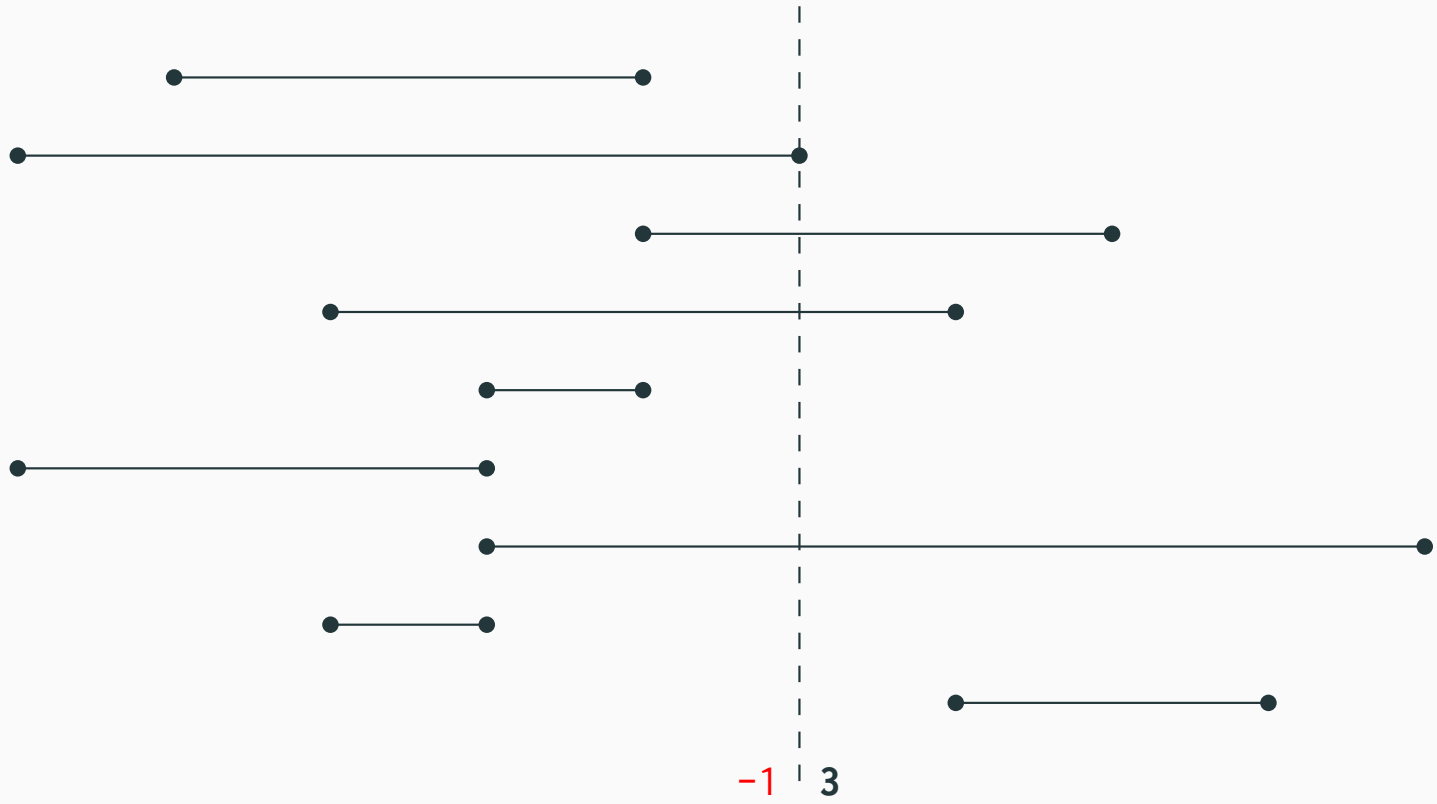
Visualização do algoritmo de interseção de intervalos

Intervalos: $(2,5), (1,6), (5,8), (3,7), (4,5), (1,4), (4,10), (3,4), (7,9)$



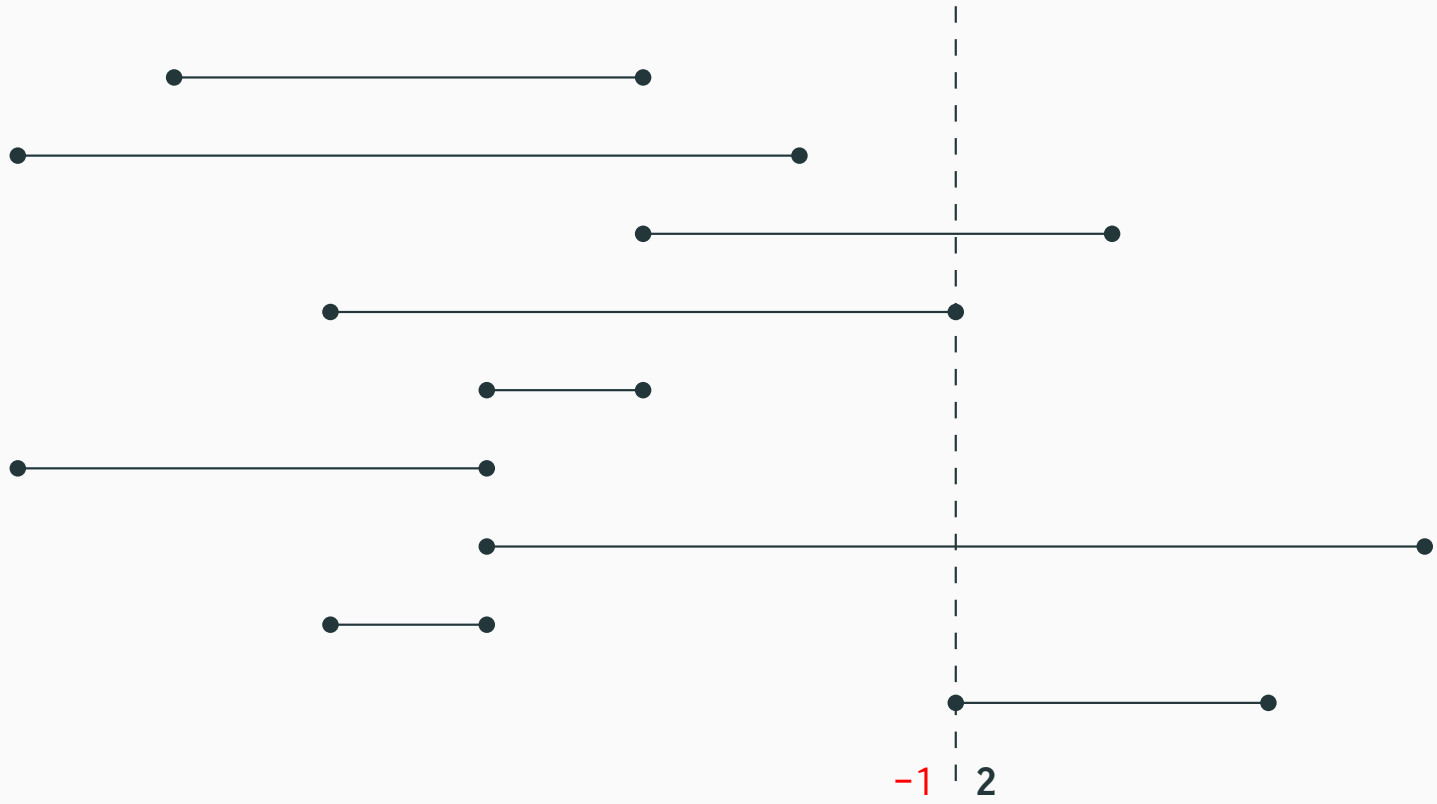
Visualização do algoritmo de interseção de intervalos

Intervalos: $(2,5), (1,6), (5,8), (3,7), (4,5), (1,4), (4,10), (3,4), (7,9)$



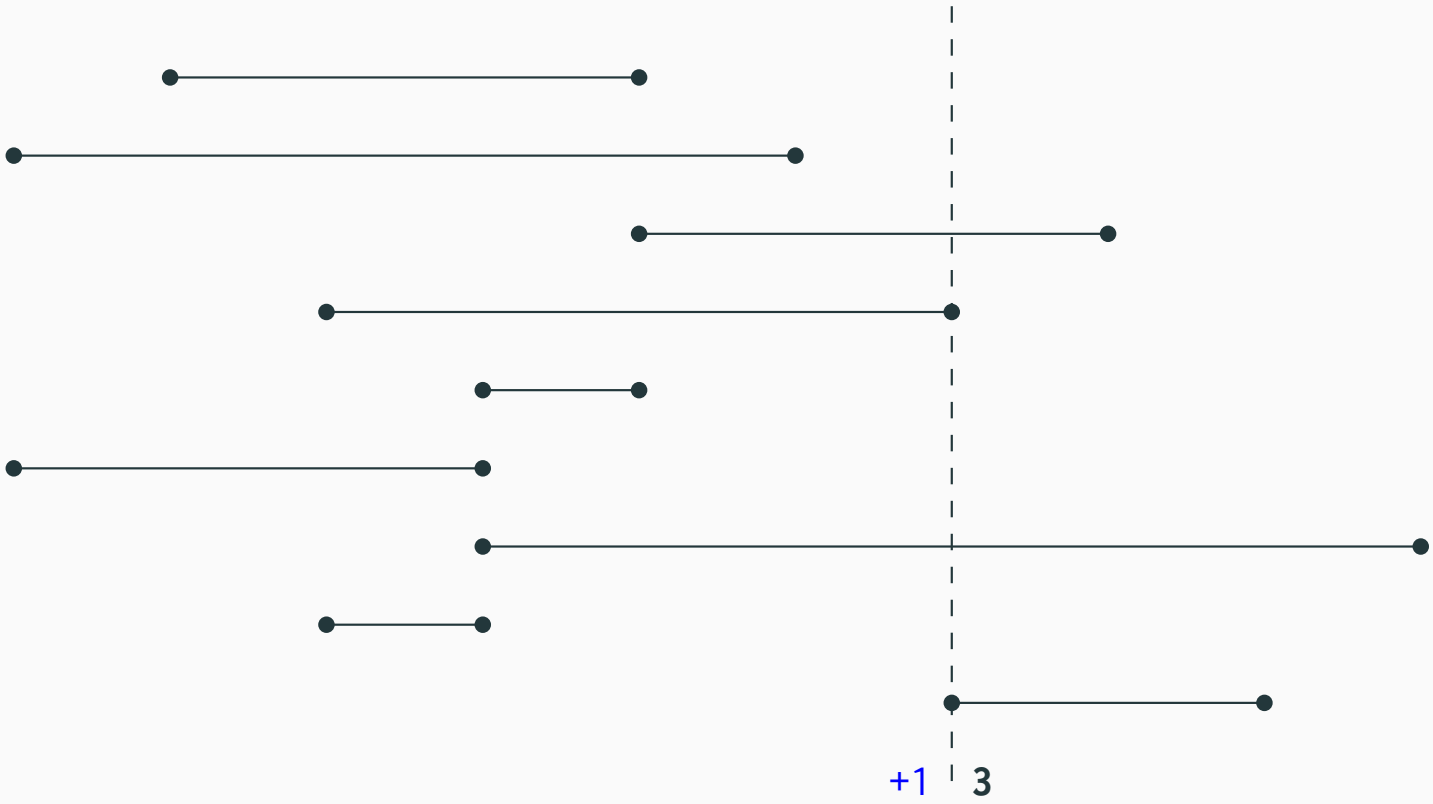
Visualização do algoritmo de interseção de intervalos

Intervalos: $(2,5), (1,6), (5,8), (3,7), (4,5), (1,4), (4,10), (3,4), (7,9)$



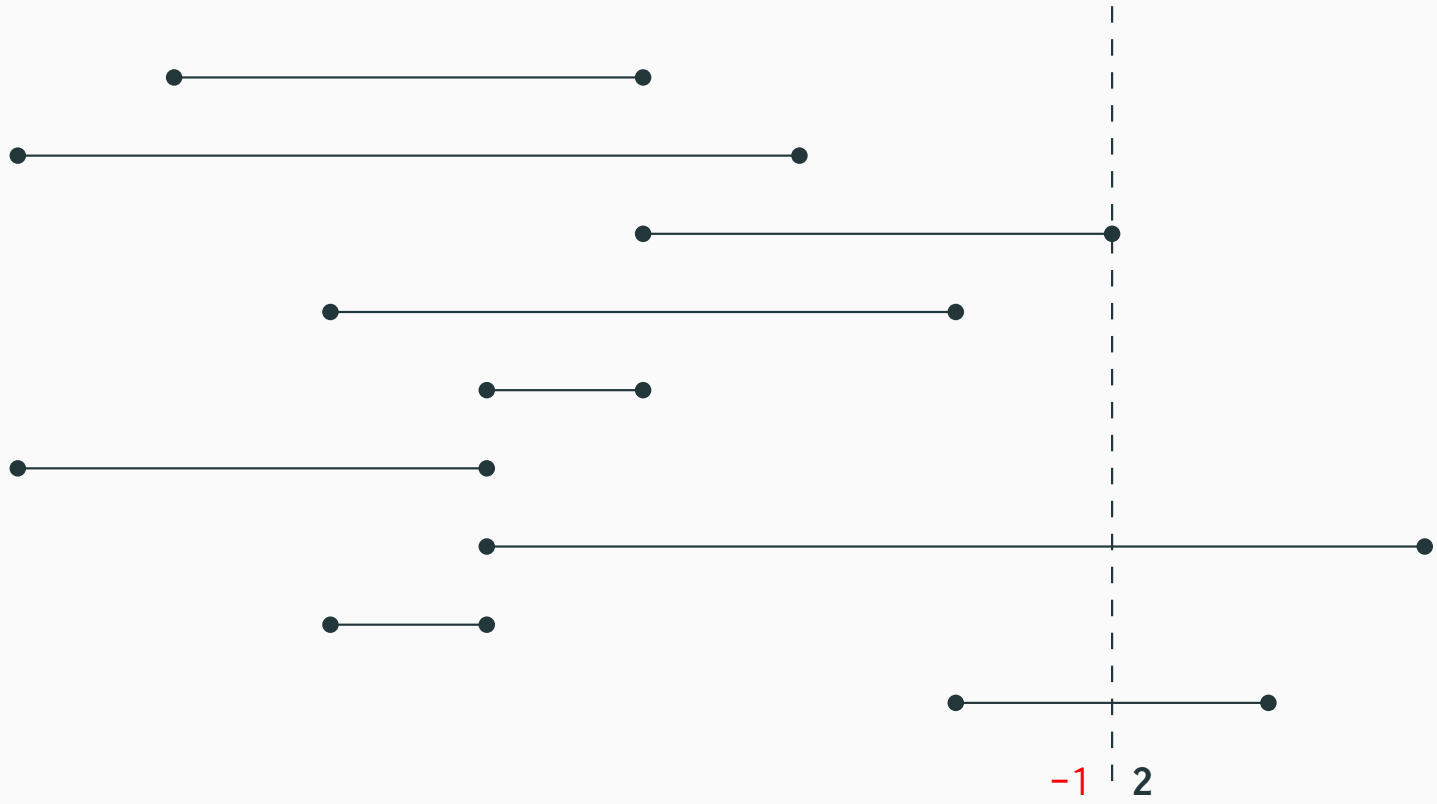
Visualização do algoritmo de interseção de intervalos

Intervalos: $(2,5),(1,6),(5,8),(3,7),(4,5),(1,4),(4,10),(3,4),(7,9)$



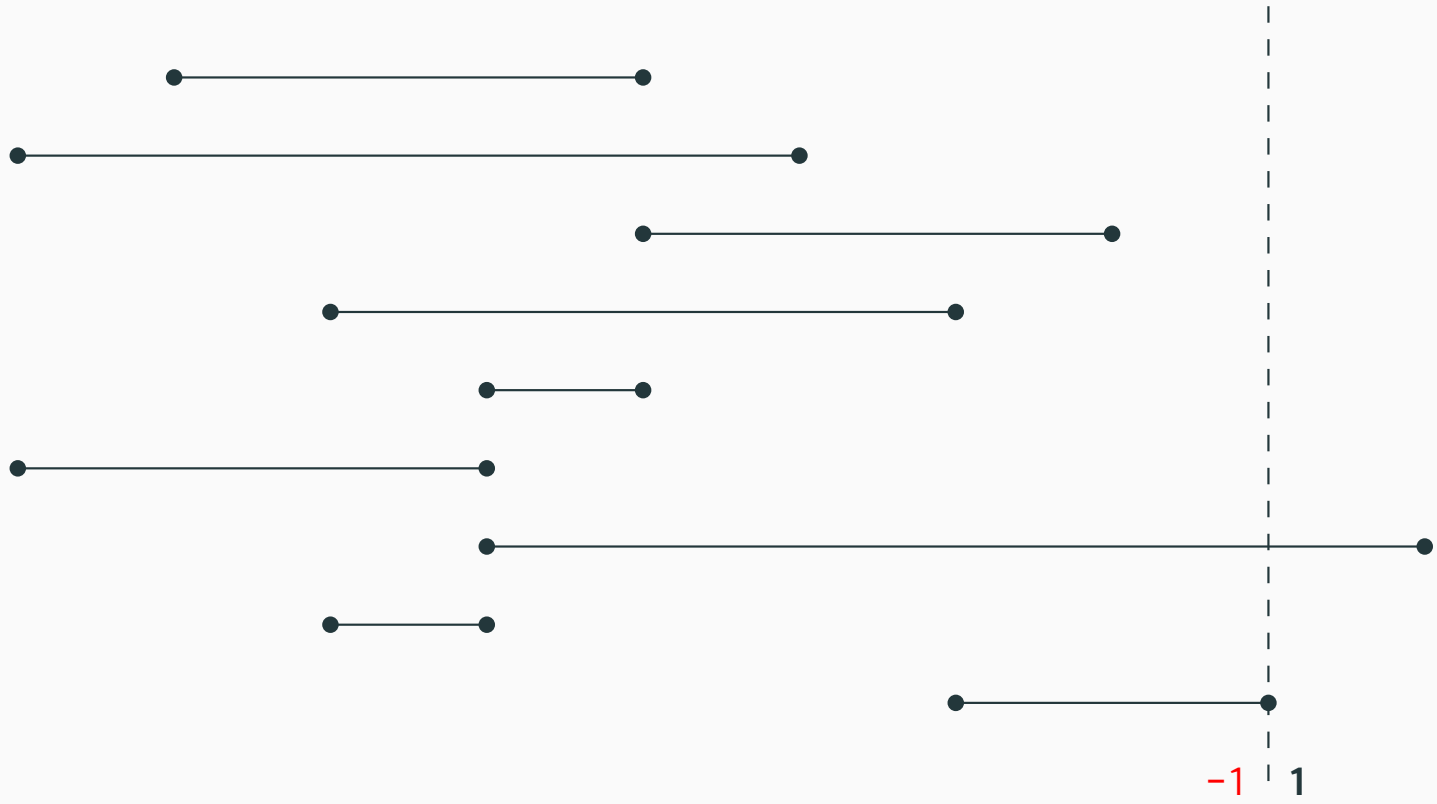
Visualização do algoritmo de interseção de intervalos

Intervalos: $(2,5), (1,6), (5,8), (3,7), (4,5), (1,4), (4,10), (3,4), (7,9)$



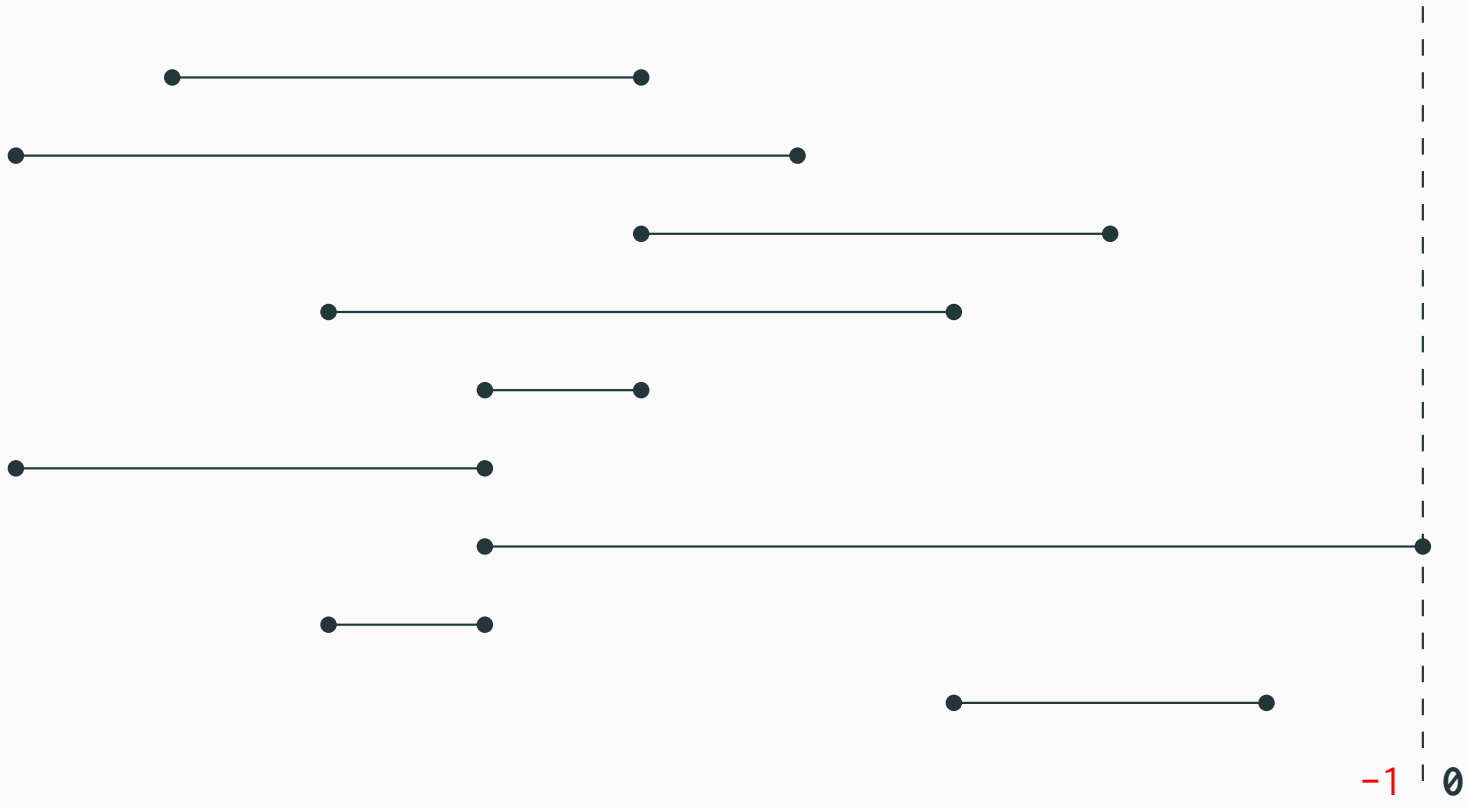
Visualização do algoritmo de interseção de intervalos

Intervalos: $(2,5), (1,6), (5,8), (3,7), (4,5), (1,4), (4,10), (3,4), (7,9)$



Visualização do algoritmo de interseção de intervalos

Intervalos: $(2,5), (1,6), (5,8), (3,7), (4,5), (1,4), (4,10), (3,4), (7,9)$



Implementação da interseção de intervalos

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

using namespace std;
using interval = pair<int, int>;

vector<int> max_intersection(const vector<interval>& is)
{
    using event = pair<int, int>;

    vector<event> es;

    for (size_t i = 0; i < is.size(); ++i)
    {
        auto a = is[i].first, b = is[i].second;

        es.push_back({ a, i + 1});      // Evento de início
        es.push_back({ b, -(i + 1)});   // Evento de fim
    }

    sort(es.begin(), es.end());

```

Implementação da interseção de intervalos

```
22      set<int> active, max_set;
23
24      for (const auto& e : es)
25      {
26          if (active.size() >= max_set.size())
27              max_set = active;
28
29          if (e.second > 0)
30              active.insert(e.second);
31          else
32              active.erase(-e.second);
33      }
34
35      return vector<int>(max_set.begin(), max_set.end());
36  }
37
38  int main()
39  {
40      vector<interval> is { { 2, 5 }, { 1, 6 }, { 5, 8 }, { 3, 7 },
41          { 4, 5 }, { 1, 4 }, { 4, 10 }, { 3, 4 }, { 7, 9 } };
42
```

Implementação da interseção de intervalos

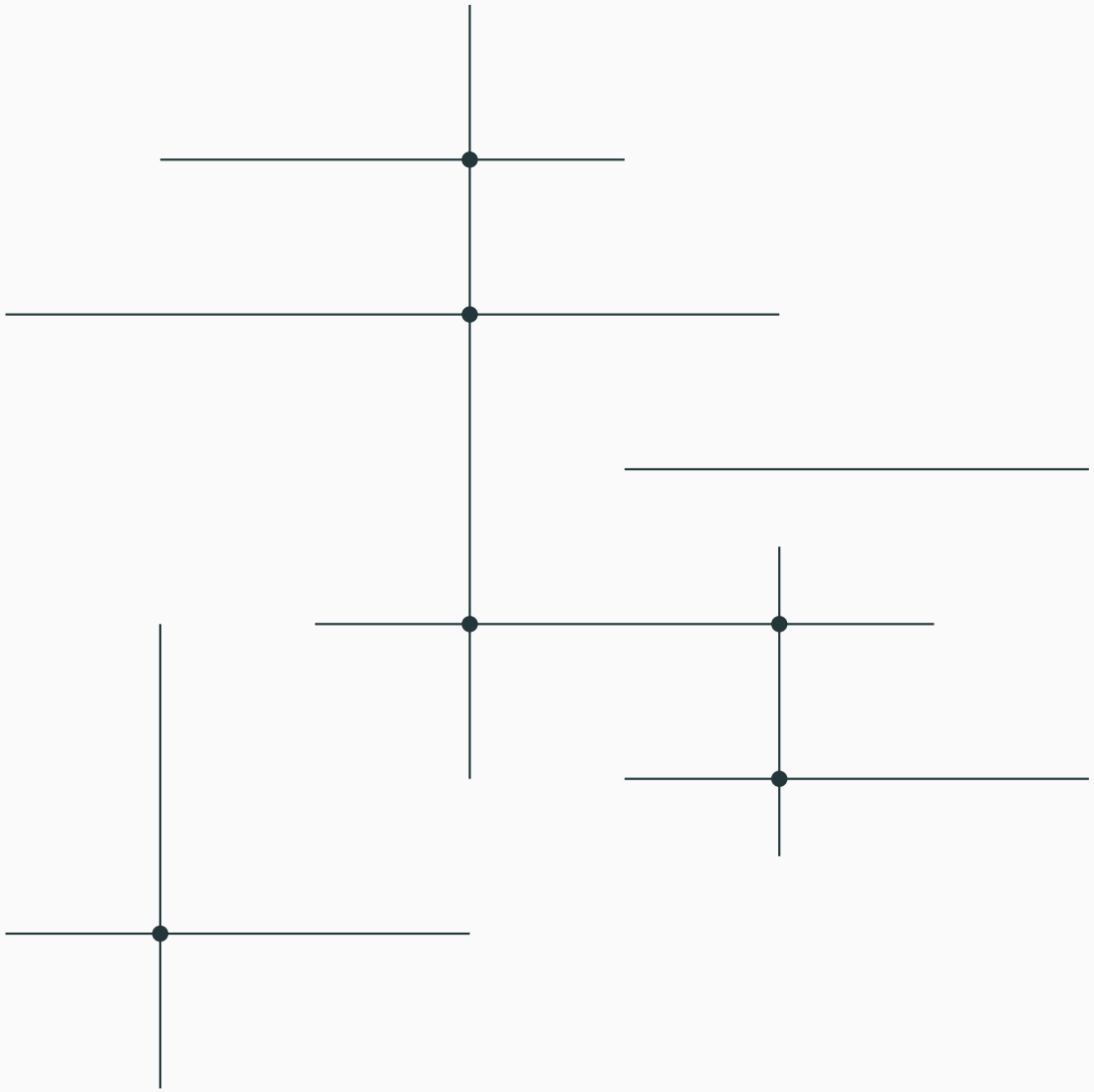
```
43      for (size_t i = 0; i < is.size(); ++i)
44          cout << i + 1 << ": (" << is[i].first << ", "
45              << is[i].second << ")\n";
46
47      auto S = max_intersection(is);
48
49      for (size_t i = 0; i < S.size(); ++i)
50          cout << S[i] << (i + 1 == S.size() ? '\n' : ' ');
51
52      return 0;
53  }
```

Pontos de interseção

Pontos de interseção entre segmentos verticais e horizontais

- Considere o seguinte problema: seja S o conjunto de N segmentos de reta, horizontais ou verticais. Determine o conjuntos dos pontos de interseção entre estes segmentos
- Um ponto estará no conjunto das interseções se ele pertence, simultaneamente, a um segmento vertical e a um segmento horizontal de S
- Assuma que os segmentos horizontais não tenham interseção entre si; e o mesmo para os segmentos verticais
- O algoritmo de força bruta compara um segmento com todos os demais, tendo complexidade $O(N^2)$

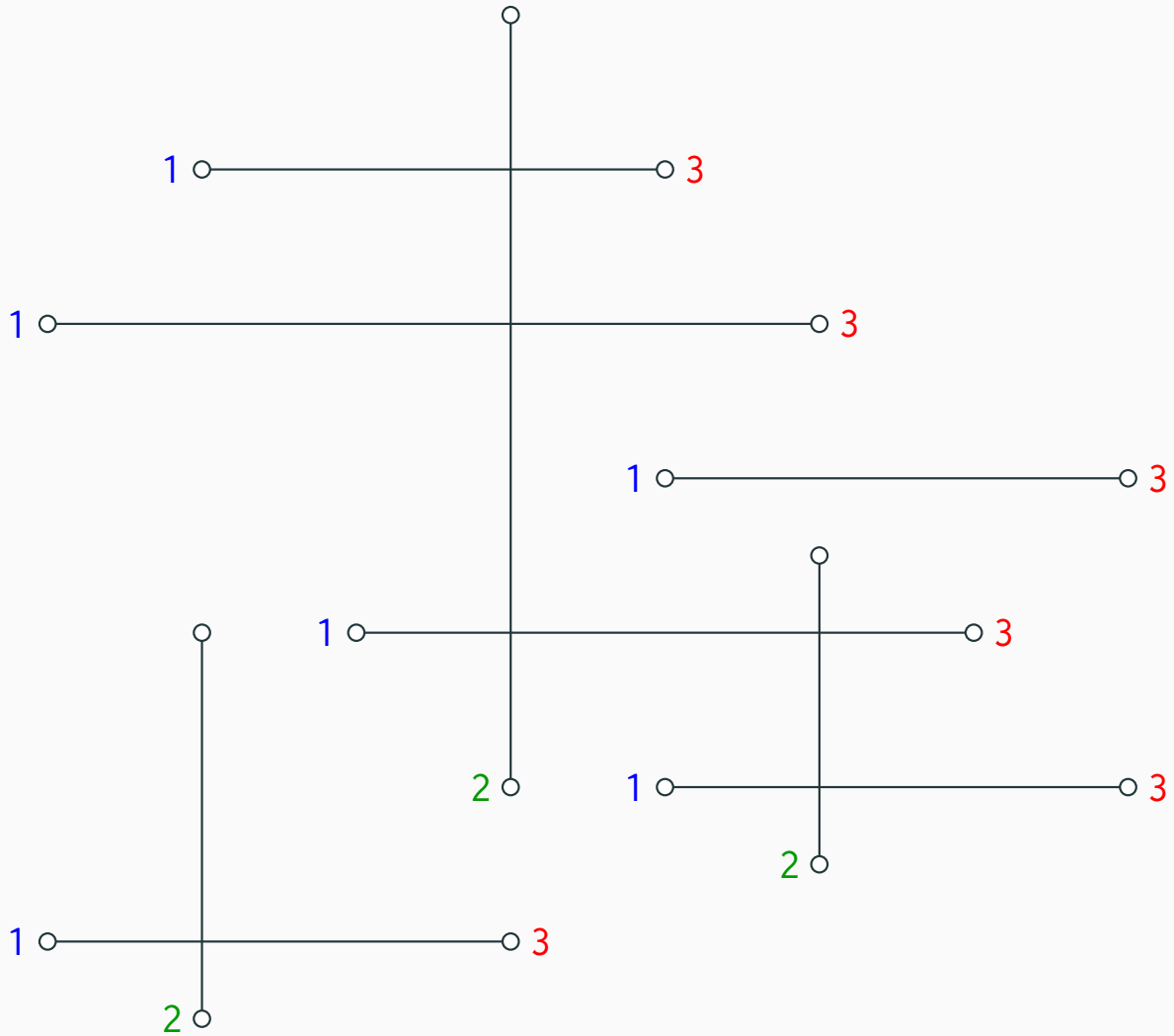
Visualização do problema de pontos de interseção



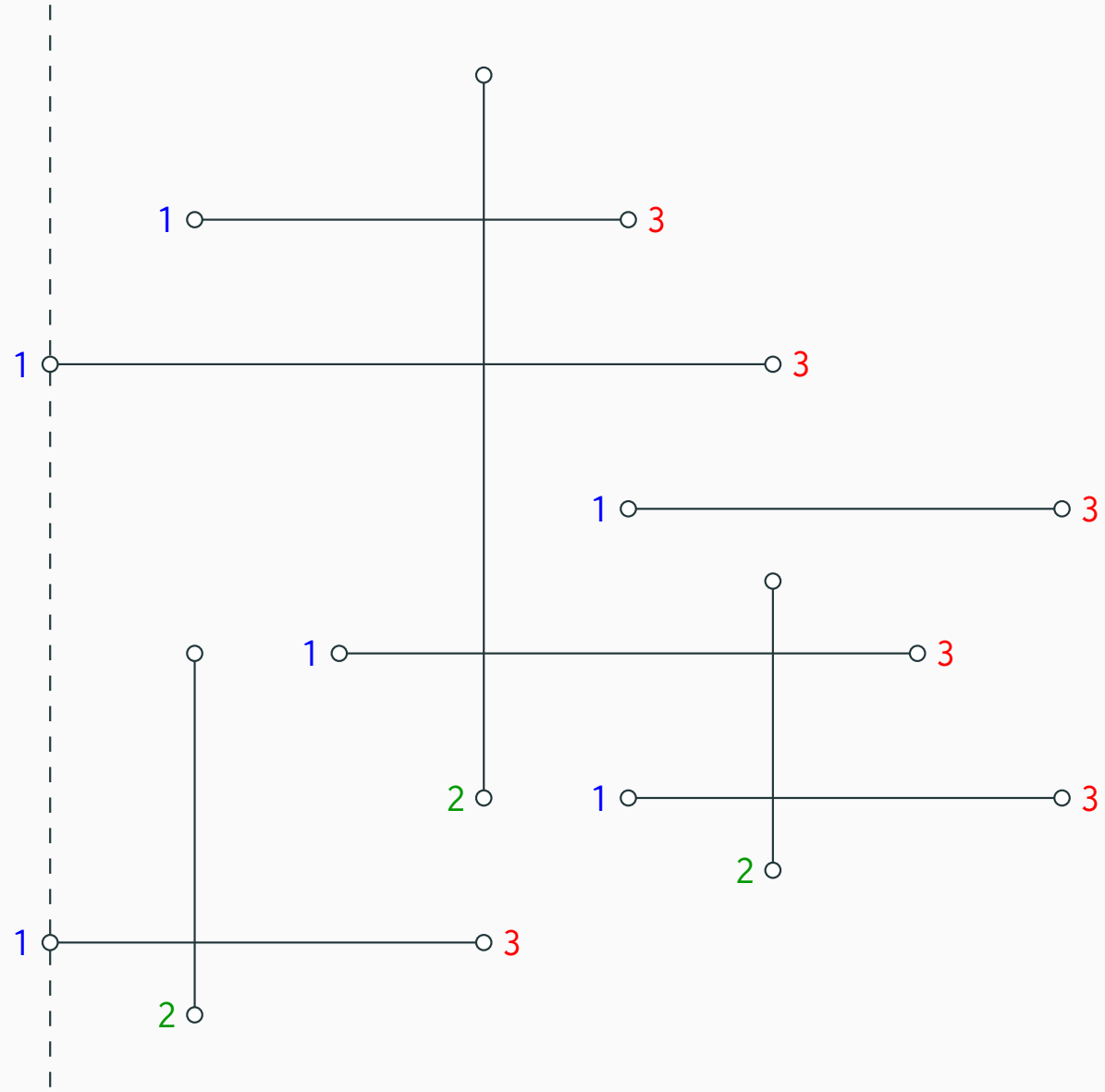
Solução utilizado sweep line

- O *sweep line* pode ser utilizado em uma solução $O(N \log N)$
- Os intervalos podem gerar três tipos de eventos:
  1. Início de um intervalo horizontal com altura y
  2. Intervalo vertical $[a, b]$
  3. Fim de um intervalo horizontal com altura y
- Cada evento do tipo 1 deve armazenar as coordenadas y dos intervalos abertos
- Em cada evento do tipo 2 deve ser contabilizados quantos valores estão no conjunto e que pertencem ao intervalo $[a, b]$
- Para determinar esta quantia os valores y devem ser armazenados em uma árvore de Fenwick
- Cada evento do tipo 3 remove a coordenada y do intervalo a ser fechado

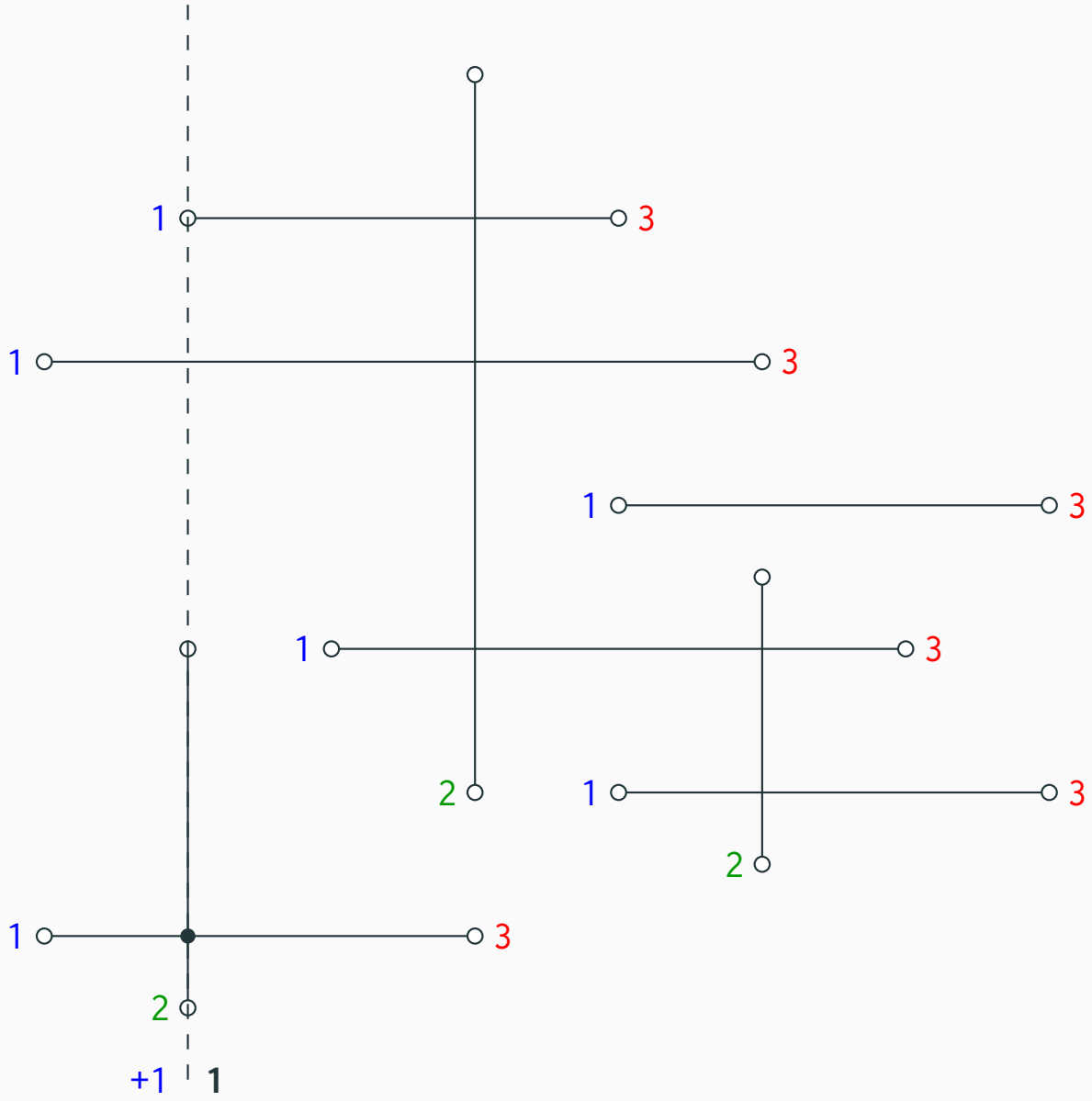
Visualização do algoritmo de pontos de interseção



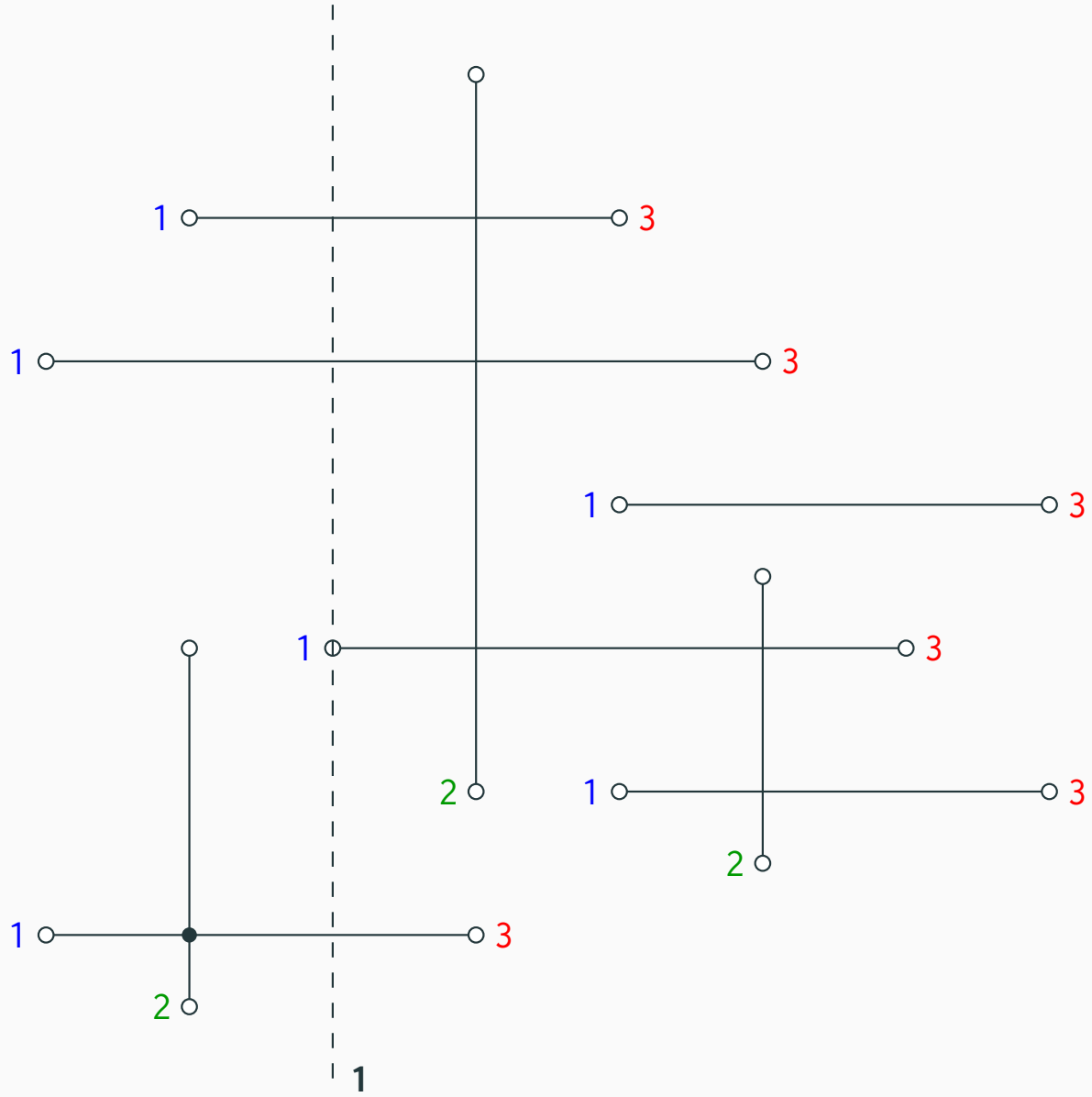
Visualização do algoritmo de pontos de interseção



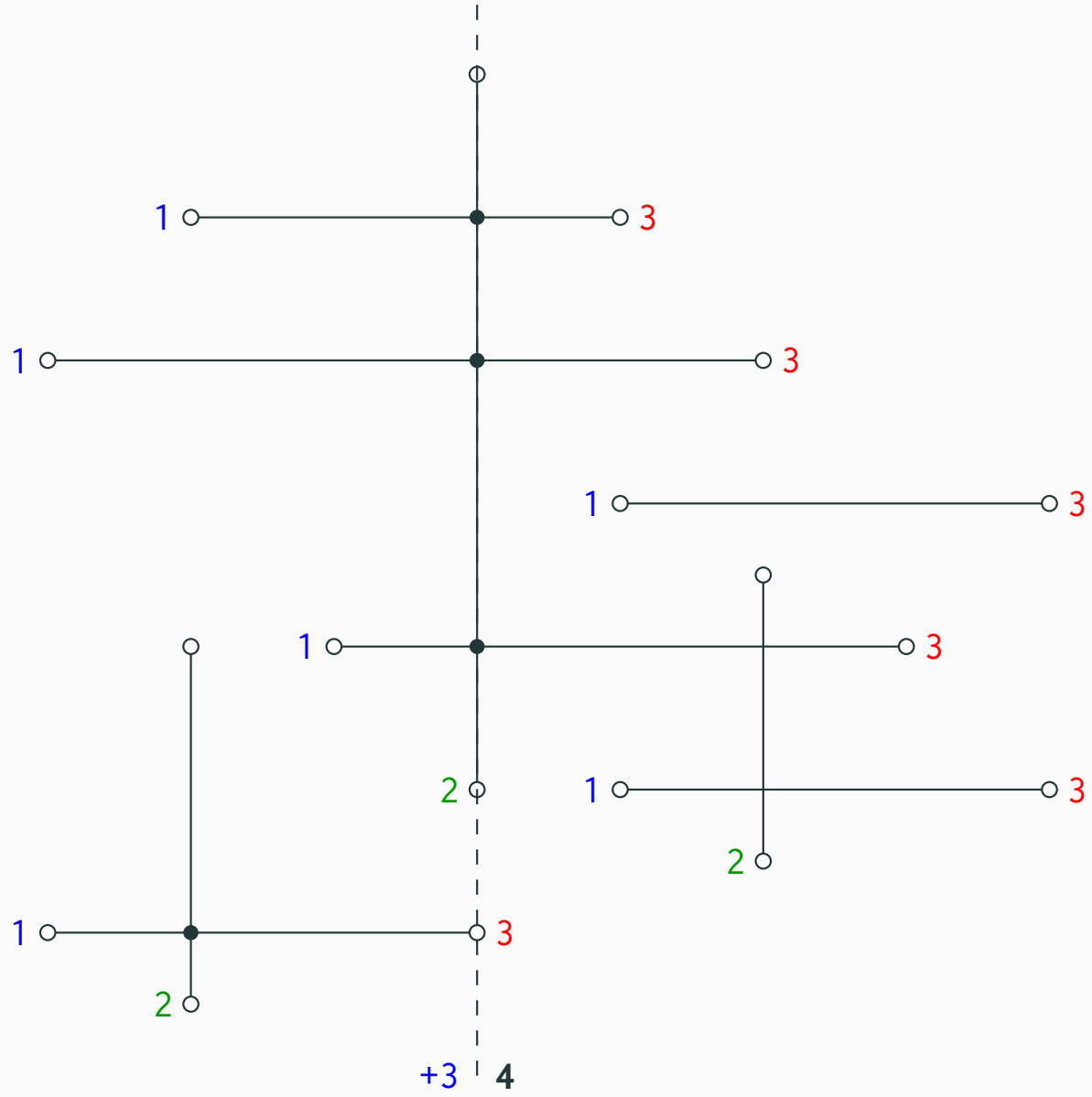
Visualização do algoritmo de pontos de interseção



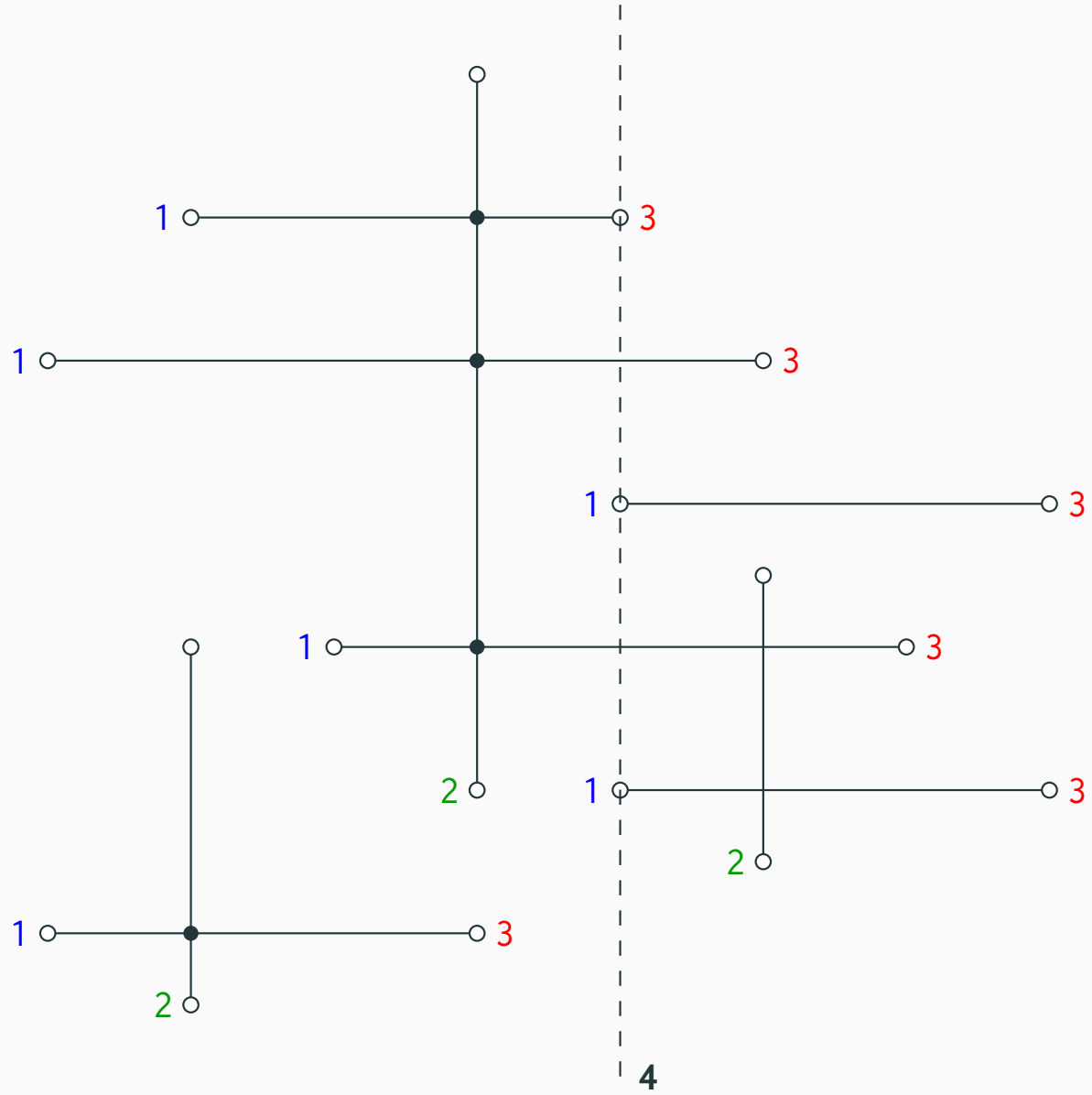
Visualização do algoritmo de pontos de interseção



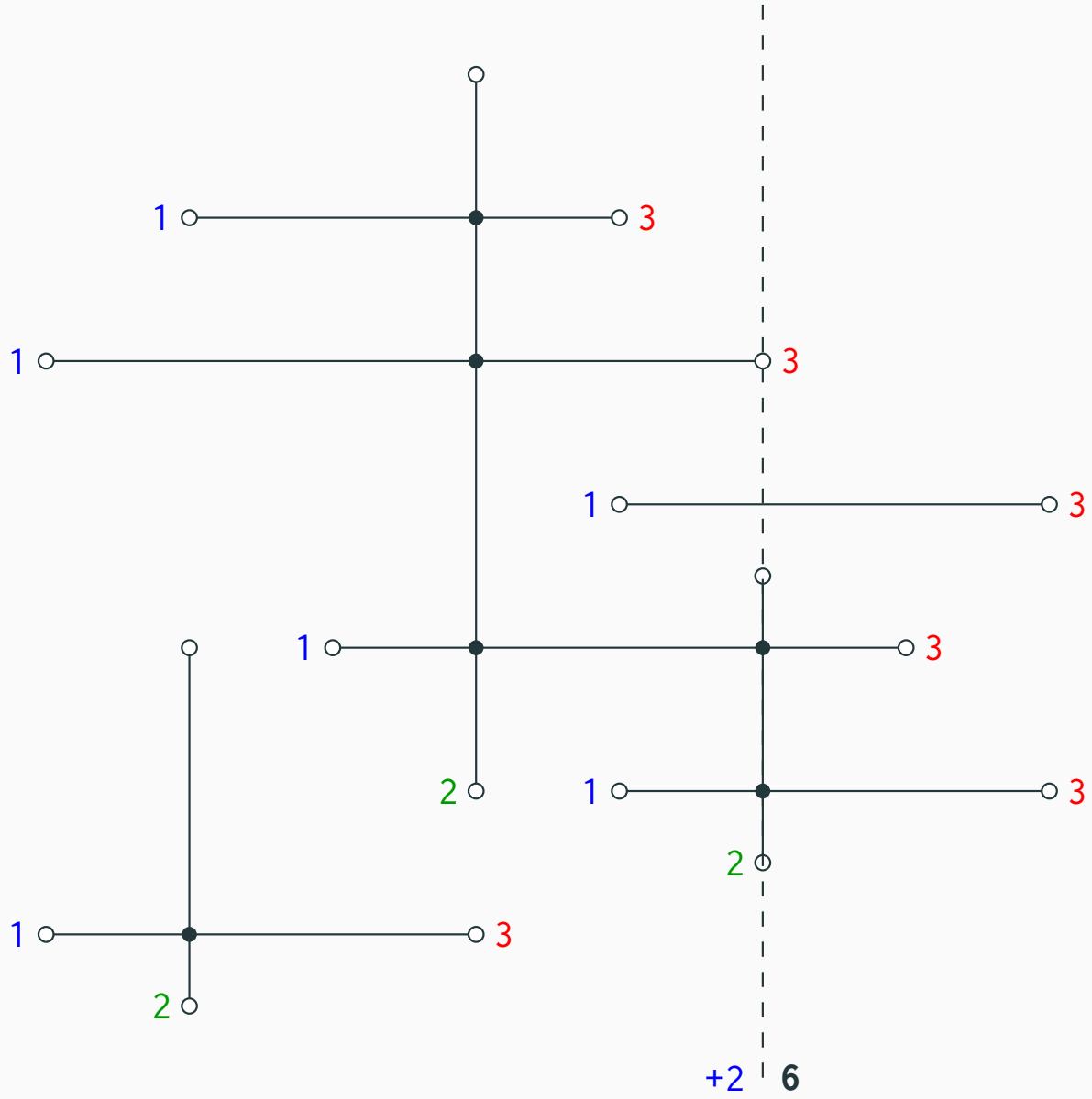
Visualização do algoritmo de pontos de interseção



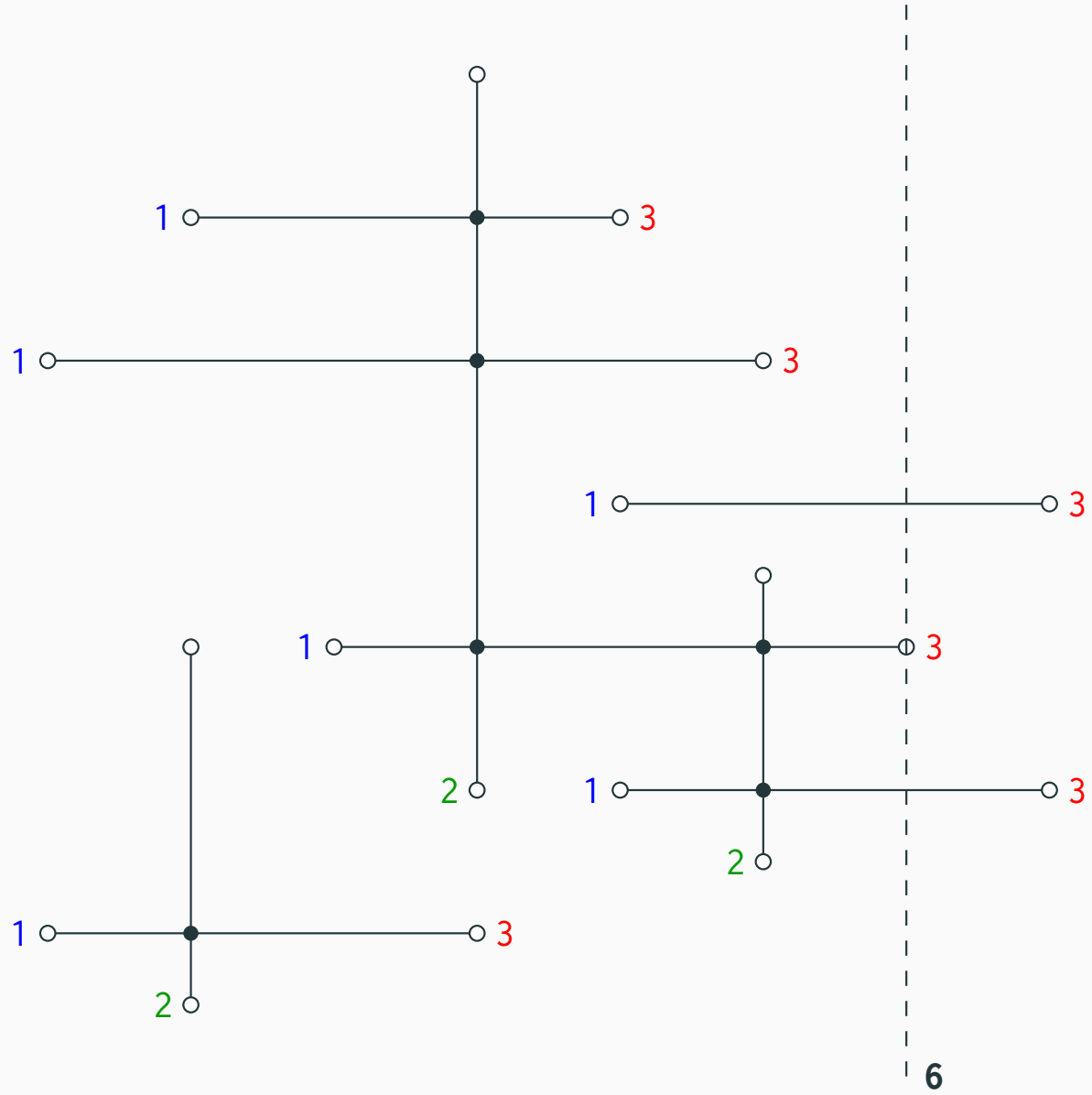
Visualização do algoritmo de pontos de interseção



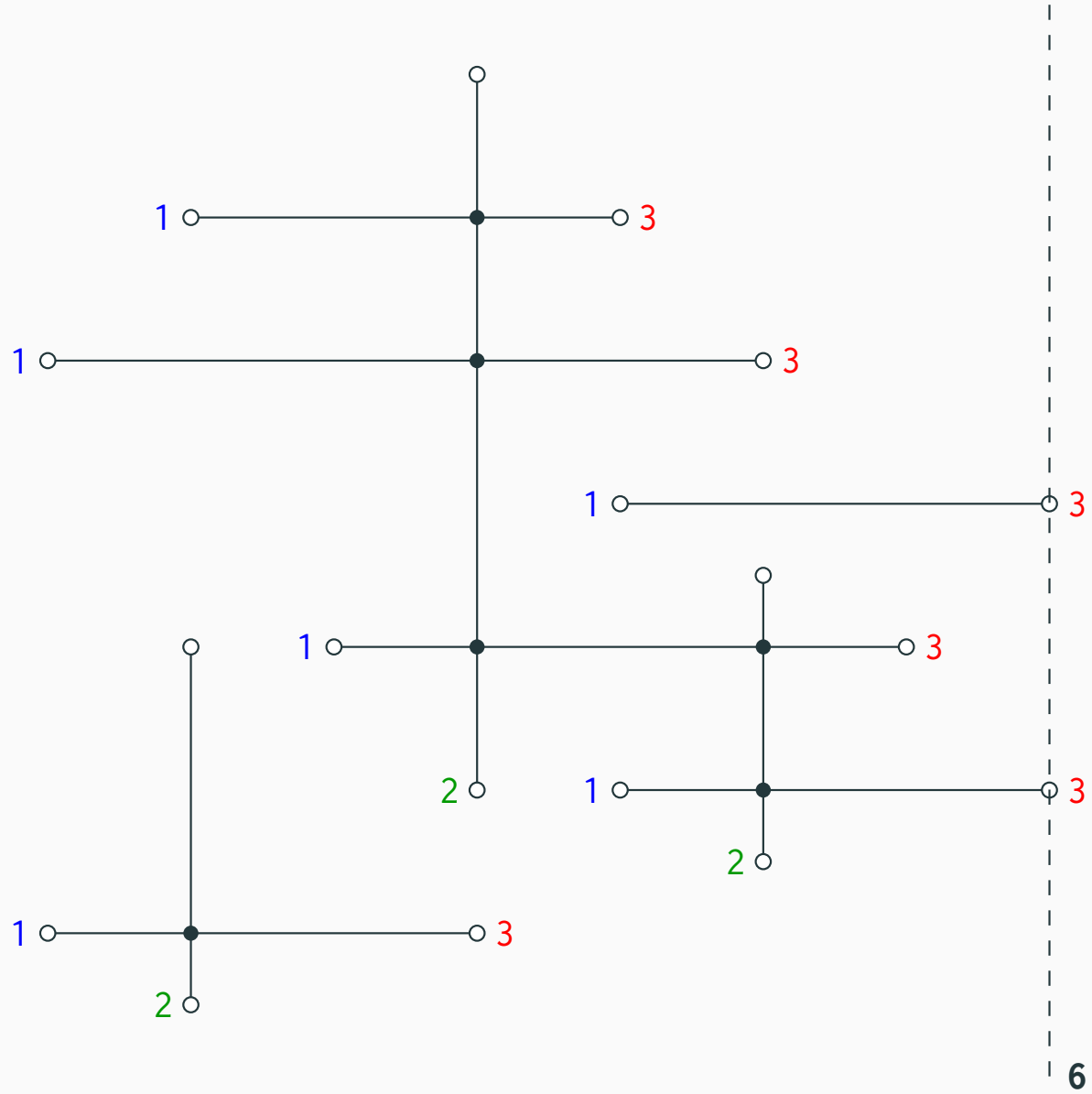
Visualização do algoritmo de pontos de interseção



Visualização do algoritmo de pontos de interseção



Visualização do algoritmo de pontos de interseção



Implementação da contagem dos pontos de interseção

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

using namespace std;

class BITree {
private:
    vector<int> ts;
    size_t N;

    int LSB(int n) { return n & (-n); }

    int RSQ(int i)
    {
        int sum = 0;

        while (i >= 1)
        {
            sum += ts[i];
            i -= LSB(i);
        }

```

Implementação da contagem dos pontos de interseção

```
22          return sum;
23      }
24
25  public:
26      BITree(size_t n) : ts(n + 1, 0), N(n) {}
27
28      int RSQ(int i, int j)
29      {
30          return RSQ(j) - RSQ(i - 1);
31      }
32
33      void add(size_t i, int x)
34      {
35          if (i == 0)
36              return;
37
38          while (i <= N)
39          {
40              ts[i] += x;
41              i += LSB(i);
42          }
```

Implementação da contagem dos pontos de interseção

```
43      }
44  };
45
46  struct Point {
47      int x, y;
48  };
49
50  struct Interval {
51      Point A, B;
52  };
53
54  int index(const vector<int>& hs, int value)
55  {
56      auto it = lower_bound(hs.begin(), hs.end(), value);
57
58      return it - hs.begin() + 1;      // Contagem inicia em 1
59  }
60
```

Implementação da contagem dos pontos de interseção

```
61 int intersections(const vector<Interval>& is)
62 {
63     struct Event {
64         int type, x;
65         size_t idx;
66 
67         bool operator<(const Event& e) const
68         {
69             if (x != e.x)
70                 return x < e.x;
71 
72             if (type != e.type)
73                 return type < e.type;
74 
75             return idx < e.idx;
76         }
77     };
78 
79     vector<Event> es;
80     set<int> ys;        // Conjunto para compressão das coordenadas
81 
```

Implementação da contagem dos pontos de interseção

```
82      for (size_t i = 0; i < is.size(); ++i)
83      {
84          auto I = is[i];
85
86          ys.insert(I.A.y);
87          ys.insert(I.B.y);
88
89          auto xmin = min(I.A.x, I.B.x);
90          auto xmax = max(I.A.x, I.B.x);
91
92          if (I.A.x == I.B.x)     // Vertical
93              es.push_back( { 2, xmin, i });
94          else                    // Horizontal
95          {
96              es.push_back( { 1, xmin, i });
97              es.push_back( { 3, xmax, i });
98          }
99      }
100
101     sort(es.begin(), es.end());
102
```

Implementação da contagem dos pontos de interseção

```
103     vector<int> hs(ys.begin(), ys.end());   // Mapa de compressão
104     BITree ft(hs.size());
105     auto total = 0;
106
107     for (const auto& event : es)
108     {
109         auto I = is[event.idx];
110
111         switch (event.type) {
112         case 1:
113             {
114                 auto y = index(hs, I.A.y);
115                 ft.add(y, 1);
116             }
117             break;
118
119         case 2:
120             {
121                 auto ymin = min(index(hs, I.A.y), index(hs, I.B.y));
122                 auto ymax = max(index(hs, I.A.y), index(hs, I.B.y));
123
```

Implementação da contagem dos pontos de interseção

```
124                 total += ft.RSQ(ymin, ymax);
125             }
126             break;
127
128         default:
129             {
130                 auto y = index(hs, I.B.y);
131                 ft.add(y, -1);
132             }
133         }
134     }
135
136     return total;
137 }
138
```

Implementação da contagem dos pontos de interseção

```
139 int main()
140 {
141     vector<Interval> is {
142         { Point { 2, 6 }, Point { 5, 6 } },
143         { Point { 1, 5 }, Point { 6, 5 } },
144         { Point { 5, 4 }, Point { 8, 4 } },
145         { Point { 3, 3 }, Point { 7, 3 } },
146         { Point { 5, 2 }, Point { 8, 2 } },
147         { Point { 1, 1 }, Point { 4, 1 } },
148         { Point { 4, 7 }, Point { 4, 2 } },
149         { Point { 2, 3 }, Point { 2, 0 } },
150         { Point { 6, 3 }, Point { 6, 1 } },
151     };
152
153     auto ans = intersections(is);
154
155     cout << ans << '\n';
156
157     return 0;
158 }
```

Referências

1. **HALIM**, Felix; **HALIM**, Steve. *Competitive Programming 3*, 2010.
2. **LAAKSONEN**, Antti. *Competitive Programmer's Handbook*, 2018.
3. **De BERG**, Mark; **CHEONG**, Otfried. *Computational Geometry: Algorithms and Applications*, 2008.
4. Wikipedia. Sweep line algorithm, acesso em 22/05/2019.