Geometria Computacional

Pontos

Prof. Edson Alves

Faculdade UnB Gama

Sumário

Definição de ponto

Definição

- Ponto é um termo primitivo da geometria

Definição

- Ponto é um termo primitivo da geometria
- No primeiro livro dos elementos, Euclides define ponto como "*a point is that which has no part*", que numa tradução livre diz que "ponto é aquilo que não tem parte"

Definição

- Ponto é um termo primitivo da geometria
- No primeiro livro dos elementos, Euclides define ponto como "*a point is that which has no part*", que numa tradução livre diz que "ponto é aquilo que não tem parte"
- Os pontos são elementos adimensionais, isto é, não tem dimensão alguma

Definição

- Ponto é um termo primitivo da geometria
- No primeiro livro dos elementos, Euclides define ponto como "*a point is that which has no part*", que numa tradução livre diz que "ponto é aquilo que não tem parte"
- Os pontos são elementos adimensionais, isto é, não tem dimensão alguma
- Em C/C++, pontos podem ser representados como classes ou estruturas, como pares ou como vetores

Definição

- Ponto é um termo primitivo da geometria
- No primeiro livro dos elementos, Euclides define ponto como "*a point is that which has no part*", que numa tradução livre diz que "ponto é aquilo que não tem parte"
- Os pontos são elementos adimensionais, isto é, não tem dimensão alguma
- Em C/C++, pontos podem ser representados como classes ou estruturas, como pares ou como vetores
- Cada representação possível tem suas vantagens e desvantagens

Representação de pontos como classes

- Representar um ponto utilizando uma classe ou estrutura tem a vantagem da legibilidade

Representação de pontos como classes

- Representar um ponto utilizando uma classe ou estrutura tem a vantagem da legibilidade
- A sintaxe para o uso é mais natural

Representação de pontos como classes

- Representar um ponto utilizando uma classe ou estrutura tem a vantagem da legibilidade
- A sintaxe para o uso é mais natural
- Porém esta representação demanda a implementação dos operadores relacionais para comparações entre pontos

Representação de pontos como classes

- Representar um ponto utilizando uma classe ou estrutura tem a vantagem da legibilidade
- A sintaxe para o uso é mais natural
- Porém esta representação demanda a implementação dos operadores relacionais para comparações entre pontos
- O uso de estruturas simplifica a implementação, uma vez que em competição não há necessidade de encapsulamento dos membros

Representação de pontos como classes

- Representar um ponto utilizando uma classe ou estrutura tem a vantagem da legibilidade
- A sintaxe para o uso é mais natural
- Porém esta representação demanda a implementação dos operadores relacionais para comparações entre pontos
- O uso de estruturas simplifica a implementação, uma vez que em competição não há necessidade de encapsulamento dos membros
- O tipo usado para representar os valores das coordenadas pode ser parametrizado, permitindo o uso da mesma implementação seja com variáveis inteiras, seja com variáveis em ponto flutuante

Exemplo de implementação de ponto usando uma estrutura

```cpp
#include <iostream>

template<typename T>
struct Point {
    T x = 0, y = 0;
};

int main() {
    Point<int> p { 1, 2 }, q;   // Declaração
    p = q;                      // Atribuição

    if (p == q) {               // Erro de compilação: o operador == não está definido!
        p.x = q.y + 1;
    }

    return 0;
}
```

Representação de pontos como pares e tuplas

- A biblioteca padrão do C++ possui o tipo paramétrico `pair` (par), que pode ser usado para representar pontos

Representação de pontos como pares e tuplas

- A biblioteca padrão do C++ possui o tipo paramétrico `pair` (par), que pode ser usado para representar pontos
- Usar pares tem como vantagem herdar os operadores de comparação dos tipos escolhidos

Representação de pontos como pares e tuplas

- A biblioteca padrão do C++ possui o tipo paramétrico `pair` (par), que pode ser usado para representar pontos
- Usar pares tem como vantagem herdar os operadores de comparação dos tipos escolhidos
- A desvantagem é a notação, que utiliza `first` e second para acessar as duas coordenadas, ao invés de `x` e `y`, como nas classes e estruturas

Representação de pontos como pares e tuplas

- A biblioteca padrão do C++ possui o tipo paramétrico `pair` (par), que pode ser usado para representar pontos
- Usar pares tem como vantagem herdar os operadores de comparação dos tipos escolhidos
- A desvantagem é a notação, que utiliza `first` e second para acessar as duas coordenadas, ao invés de `x` e `y`, como nas classes e estruturas
- Esta desvantagem pode ser contornada com o uso da decomposição estruturada (C++17 em diante):

```
1 Point2D p;
2 auto [x, y] = p;
```

Representação de pontos como pares e tuplas

- A biblioteca padrão do C++ possui o tipo paramétrico `pair` (par), que pode ser usado para representar pontos
- Usar pares tem como vantagem herdar os operadores de comparação dos tipos escolhidos
- A desvantagem é a notação, que utiliza `first` e second para acessar as duas coordenadas, ao invés de `x` e `y`, como nas classes e estruturas
- Esta desvantagem pode ser contornada com o uso da decomposição estruturada (C++17 em diante):

```
1 Point2D p;
2 auto [x, y] = p;
```

- Além disso, tuplas (`tuple`) podem ser utilizados diretamente para representar pontos tridimensionais

Exemplo de implementação de ponto usando pares e tuplas

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

using Point2D = std::pair<int, int>;          // C++11 em diante
using Point3D = std::tuple<int, int, int>;

int main()
{
    Point3D p { 1, 2, 3 }, q;                 // Declaração
    p = q;                                    // Atribuição

    if (p == q) {                             // Ok! Operador == para ints utilizado
        auto [x, y, z] = q;                   // Decomposição estruturada para tuplas
        auto w = x*x + y*y + z*z;
    }

    return 0;
}
```

Representação de pontos como vetores

- Representar pontos como vetores bidimensionais permite a travessia de conjuntos de pontos em laços por coordenada

Representação de pontos como vetores

- Representar pontos como vetores bidimensionais permite a travessia de conjuntos de pontos em laços por coordenada
- Além disso, é a implementação mais curta para pontos multidimensionais

Representação de pontos como vetores

- Representar pontos como vetores bidimensionais permite a travessia de conjuntos de pontos em laços por coordenada
- Além disso, é a implementação mais curta para pontos multidimensionais
- Porém, a legibilidade fica comprometida, uma vez que as coordenadas são acessadas por índices

Representação de pontos como vetores

- Representar pontos como vetores bidimensionais permite a travessia de conjuntos de pontos em laços por coordenada
- Além disso, é a implementação mais curta para pontos multidimensionais
- Porém, a legibilidade fica comprometida, uma vez que as coordenadas são acessadas por índices
- Esta representação não herda a atribuição e ainda pode gerar confusão com o uso de operadores relacionais

Exemplo de implementação de ponto usando vetores

```cpp
#include <iostream>

using Point = double[2];

int main()
{
    Point p, q {0, 0};      // Declaração
    p = q;                  // Erro de compilação: a atribuição não está definida

    if (p < q) {            // Perigo: comparação entre endereços de ponteiros
        p[0] = q[1];        // O código compila sem erros
    }

    return 0;
}
```

Comparação entre pontos

Operadores relacionais

- Os operadores relacionais que estarão disponíveis dependem da representação escolhida

Operadores relacionais

- Os operadores relacionais que estarão disponíveis dependem da representação escolhida
- Os operadores fundamentais são a igualdade (operador ==) e a desigualdade (operador !=)

Operadores relacionais

- Os operadores relacionais que estarão disponíveis dependem da representação escolhida
- Os operadores fundamentais são a igualdade (operador ==) e a desigualdade (operador !=)
- Os operadores < e > também podem ser definidos, embora a semântica destas comparações dependa do contexto e da implementação utilizada

Operadores relacionais

- Os operadores relacionais que estarão disponíveis dependem da representação escolhida
- Os operadores fundamentais são a igualdade (operador ==) e a desigualdade (operador !=)
- Os operadores < e > também podem ser definidos, embora a semântica destas comparações dependa do contexto e da implementação utilizada
- Mesmo no caso dos pares, que herda a igualdade, é importante implementá-la caso o tipo utilizado para armazenar as coordenadas seja o ponto flutuante, para que seja usado o limiar ϵ

Exemplo de implementação da igualdade

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

template<typename T>
bool equals(T a, T b)
{
    constexpr double EPS { 1e-9 };

    return std::is_floating_point<T>::value ?  fabs(a - b) < EPS : a == b;
}

template<typename T>
struct Point {
    T x = 0, y = 0;

    bool operator==(const Point& p) const noexcept {
        return equals(x, p.x) && equals(y, p.y);
    }

```

Exemplo de implementação da igualdade

```
19      bool operator!=(const Point& p) const noexcept {
20          return not (*this == p);
21      }
22  };
23
24  int main()
25  {
26      Point<double> p { 1, 2 }, q { 3*1.0/3, 2 };
27
28      if (p == q)
29          p.x = q.y;
30
31      std::cout << "p = (" << p.x << ", " << p.y << ")\n";
32
33      return 0;
34  }
```

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. David E. Joyce. *Euclid's Elements*. Acesso em 15/02/2019[1]
5. Wikipédia. *Geometria Euclidiana*. Acesso em 15/02/2019[2].

[1] https://mathcs.clarku.edu/ djoyce/elements/bookI/defI1.html
[2] https://pt.wikipedia.org/wiki/Geometria_euclidiana