Resumo OBI

Aleardo Manacero

2020

1 Ordenação com quicksort

A função qsort() faz a ordenação dos elementos de um vetor. Dentro do seu programa ela é chamada assim:

```
qsort (nome_vetor, num_elementos, tamanho_elemento, funcao_escolha);
```

A `funcao_escolha` compara dois elementos e decide qual deles deve aparecer antes no vetor. Seu código é algo do tipo:

```
int funcao_escolha (const void *a, const void *b)
{ return ( (int)(*a) - (int)(*b) ); // se o vetor for de inteiros
}
```

Mas qsort() pode ser usado para mais coisas, como ordenar um vetor de inteiros colocando primeiro os pares e depois os ímpares, sempre ordenados. Para isso a função fica:

```
int funcao_escolha (const void *p, const void *q)
{   // Recebe os valores dos dois inteiros
    int l = *(const int *)p;
    int r = *(const int *)q;

    // Se os dois forem ímpares ou os dois forem pares coloque o menor antes
    if ( ((l&1) && (r&1)) || ( !(l&1) && !(r&1) ) )
        return (l-r);

    // Se um for ímpar e o outro par, coloque primeiro o par
    if (!(l&1))
        return -1;
    // l é ímpar, coloque depois
    return 1;
}
```

2 Caminho mínimo

O código a seguir resolve o problema do caminho mínimo. No caso ele está aplicado ao problema FRETE da OBI de 2017.

```
#include "stdio.h"

#define MAX 1001
#define BIG 1000000 // valor "infinito" inicial

int frete[MAX], marca[MAX], cidade[MAX][MAX];

// função para encontrar vértice com menor valor de frete
// e que ainda nao foi "percorrido"
int mincusto(int N)
{int i, menor=N;

  for (i=N-1; i>1; i--) // procura vértice com menor custo
     if (frete[i] < frete[menor] && marca[i] == 0)
         menor = i;
  if (menor != N)  // se não existirem mais vértices faz menor = -1
      return(menor);
    else
      return (-1);
}

int main()
{int i, j, N, M, A, B, C, cand;

  scanf("%d %d", &N, &M);
  for (i = 0; i <= N; i++) // Preenche tudo como custo "infinito"
    for (j = 0; j <= N; j++)
        cidade[i][j] = BIG;

  for (i = 0; i < M; i++ )  // Lê as ligações e custos conhecidos
  { scanf("%d %d %d", &A, &B, &C);
    cidade[A][B] = cidade[B][A] = C;
  }
  for (i = 1; i <= N; i++)
  { cidade[i][i] = 0;
    frete[i] = cidade[1][i]; // Determina fretes iniciais
    marca[i] = 0;  // Indica que não foi percorrida ainda
  }
  // começa dizendo que o frete da cidade 1 para ela mesma é 0
  // e diz que a primeira candidata é a própria cidade 1
  frete[1] = 0;     cand = 1;
```

```
// Aqui começa o algoritmo de caminho mínimo (é o laço while)
  while (cand>=0)
  { for (i=1; i<=N; i++)
    { if (cidade[cand][i] < BIG) // se tem caminho entre cand e i
      { if (frete[i] > frete[cand] + cidade[cand][i])
            { frete[i] = frete[cand] + cidade[cand][i];
              cidade[cand][i] = frete[i];
            } // troca o valor de frete se o caminho diminui o valor
      }
    }
    marca[cand] = 1;  // marca o vértice como "percorrido"
    cand = mincusto(N);  // procura novo vértice candidato
  }
  printf("%d\n",frete[N]);
  return 0;
}
```

3 Uso de produto vetorial

Aplicações envolvendo posições relativas entre segmentos de reta podem ser resolvidas com a aplicação de produto vetorial. Considerando dois vetores, p_1 e p_2, com origem em um ponto comum (aqui será o ponto (0,0)), temos que o produto vetorial deles será:

$$\begin{aligned} p_1 \times p_2 &= \det \begin{pmatrix} \mathrm{x}_1 & \mathrm{x}_2 \\ \mathrm{y}_1 & \mathrm{y}_2 \end{pmatrix} \\ &= x_1.y_2 - x_2.y_1 \\ &= -p_2 \times p_1 \end{aligned}$$

3.1 Determinando posição relativa entre segmentos

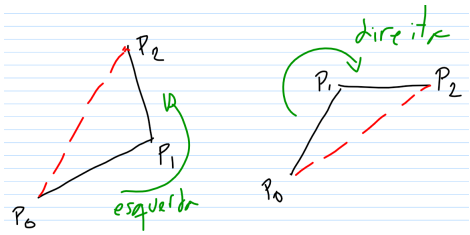
Considerando os segmentos a e b, que tenham um ponto de partida em comum, digamos p_0, podemos determinar a posição entre eles com o cálculo do produto vetorial. Assim, se $a \times b$ é positivo, então a está a direita de b, isto é, se considerarmos o movimento dos ponteiros de um relógio, passamos primeiro por b e depois por a. Se esse produto for negativo, então a está a esquerda. Se o produto for zero, então são colineares.

Considerando que o segmento a tem origem em $p_0 = (x_0, y_0)$ e termina em $p_1 = (x_1, y_1)$ e o segmento b tem origem em p_0 e termina em $p_2 = (x_2, y_2)$, então o produto vetorial é dado por:

$$a \times b = (x_1 - x_0).(y_2 - y_0) - (x_2 - x_0).(y_1 - y_0)$$

3.2 Determinando a direção de um caminho

Supondo agora que os segmentos a e b tem como intersecção o ponto final do primeiro e inicial do segundo, de forma a que a comece em p_0 e termine em p_1 e que b comece em p_1 e termine em p_2, então a direção da "curva" feita em p_1 pode ser determinada calculando o produto vetorial entre a e o segmento formado pelos pontos p_0 e p_2, que chamaremos de p_0p_2. Assim, se $p_0p_2 \times a$ for positivo, então p_0p_2 está a direita e a curva foi para a direita. Se for negativo, então a curva foi para a esquerda.



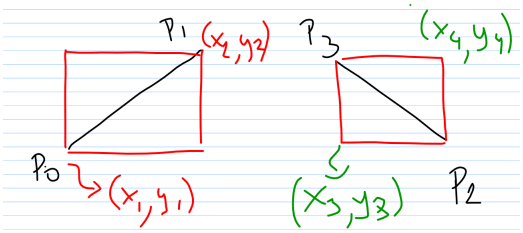
3.3 Interseção entre dois segmentos

Podemos usar o produto vetorial também para determinar se dois segmentos se interceptam, ou seja, apresentam um ponto de interseção. Para determinar isso se aplica primeiro o chamado teste de rejeição, que procura verificar se as "caixas de limitação" dos segmentos apresentam interseção. Se as caixas não tiverem intersecção (em x e em y), então é impossível que os segmentos se interceptem.

3.3.1 Teste de rejeição

Considerando os segmentos da figura abaixo, os segmentos passariam no teste de rejeição, observando que os pontos na equação são os vértices marcados e não as extremidades dos segmentos, se e somente se:

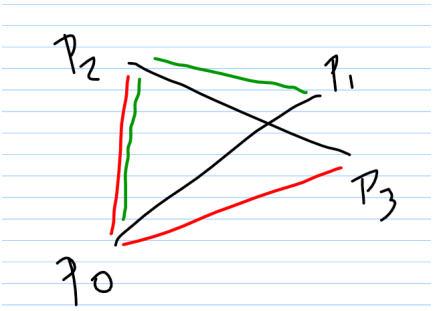
$(x_2 \geq x_3) \wedge (x_4 \geq x_1) \wedge (y_2 \geq y_3) \wedge (y_4 \geq y_1)$



3.3.2 Determinando se existe intersecção

Se os segmentos passarem no teste de rejeição, então é necessário verificar se os mesmos se interceptam ou não. Isso é feito calculando o produto vetorial

entre um segmento e segmentos imaginários que levam ao outro segmento, como representado graficamente a seguir.



Então, considerando os segmentos P_0P_1 e P_2P_3, primeiro calculamos os produtos $P_0P_1 \times P_0P_2$ e $P_0P_1 \times P_0P_3$ (segmentos em vermelho). Para que exista a possibilidade de cruzamento entre os segmentos, então esses produtos devem ter sinais contrários.

Depois temos que fazer o mesmo a partir do ponto P_2, com os segmentos em verde. Se os sinais também forem contrários, então os segmentos se interceptam.

4 Algoritmos para números

4.1 Cálculo do MDC entre dois números

Para isso se usa o algoritmo de Euclides, que basicamente determina o valor do MDC recursivamente, do seguinte modo:

$MDC(a, b) = MDC(b, a\%b)$

Isso prossegue até que tenhamos b=0. Em C isso fica:

```
int mdc(int a, int b)
{ if (b == 0)
      return(a);
    else
      return( mdc(b, a%b) );
}
```