Algoritmos e Estruturas de Dados BCC 2018/1

Exercício Programa 2: Trem-bala de Bangladânia

Rodrigo de Souza

Entrega: quando a peste negra deixar de assolar a Europa

A grande nação de Bangladânia pretende, como parte de grandes empreendimentos previstos para o campeonato mundial de truco que sediará, construir um trem-bala ligando duas de suas maiores cidades. Após anos de batalhas políticas, re-engenharias financeiras e imbróglios judiciais, os políticos de lá compreenderam que um trem-bala só seria viável se ligasse o pair mais próximo de suas cidades, e olhe lá.

A empresa que mais vem agradando no processo licitatório afirma que esse problema é comum nos (muitos) países onde já implantou linhas de trem-bala, por isso possui um sistema automatizado que, a partir do mapa das cidades de um país, encontra o par mais próximo de cidades (em termos de distância euclidiana no plano, sem desvios).

Mas a empresa, que estava mentindo só para ganhar a licitação, não tem sistema algum. Então você foi contratado às pressas para construir esse sistema. Além disso, para que sua empresa continue com chances de ganhar a licitação de grandes conglomerados industriais também interessados no projeto, sua solução deve ter um mínimo de sofisticação.

1 Entrada

A entrada deve ser lida de um arquivo texto `trembala.dat`, que representa o mapa de um país. Esse mapa é representado pelas coordenadas de suas cidades no plano cartesiano. A primeira linha do arquivo é o número n de cidades no mapa, ou seja, as cidades são numeradas como $1, 2, \ldots, n$. A segunda linha apresenta as abscissas e a terceira as ordenadas das cidades (separadas por espaços), ou seja, as coordenadas das cidades são $(x[1], y[1]), (x[2], y[2]), \ldots, (x[n], y[n])$, onde x e y são os valores apresentados na segunda e terceira linhas respectivamente. Pode supor que as coordenadas são números inteiros. O arquivo não tem mais nada além dessas três linhas. Dois exemplos de arquivos de entrada:

```
9
1 2 3 4 5 6 7 8 8
10 3 1 2 5 11 14 9 1
```

```
5
7 8 5 4 3
4 3 2 1 1
```

No primeiro arquivo, temos nove cidades, no segundo, cinco cidades. As coordenadas das cidades no primeiro arquivo são $(1, 10)$, $(2, 3)$, $(3, 1)$, etc. Note que não há nenhuma ordenação a priori para esses valores.

Seu programa deve imprimir (na tela) um par de índices, indicando um par mais próximo de cidades no mapa, e a distância entre elas. A saída não deve conter nenhuma formatação além desses três números, separados por espaços.

Sua solução, para que atinja a sofisticação necessária, deve implementar uma estratégia de divisão-e-conquista que consiste em:

- Ordenar previamente as abscissas das cidades (isso só é feito uma vez; é um pré-processamento) usando algum algoritmo de ordenação de complexidade $\mathcal{O}(n \lg n)$;
- Dividir o conjunto de pontos em dois subconjuntos de mesmo tamanho separados por uma linha vertical, usando para isso a ordenação produzida no pré-processamento;
- Resolver recursivamente o problema em cada um dos sub-conjuntos de pontos, e juntar as duas soluções para construir a solução do conjunto original.

A terceira etapa é um pouco delicada, e envolve também ordenar o eixo das ordenadas. Você pode encontrar uma descrição sucinta mas bastante completa dessa solução em

`https://www.ime.usp.br/~cris/aulas/10_1_6711/slides/aula2pmp.pdf`

Você deve implementar a solução descrita nesse link, que resulta em um algoritmo de complexidade $\mathcal{O}(n \lg n)$ (a ordenação do eixo y serve para garantir que a etapa de combinar as soluções tenha complexidade linear, e isso é essencial para a complexidade final). A primeira coisa a se fazer é ler os slides e tentar entender a solução. Vamos ter tempo para discutí-la em aula e remotamente.

Um comentário importante: ordenando-se um dos eixos, a permutação dos elementos deve envolver os valores do outro eixo também. Ou seja, cuidado para não trocar valores no eixo x de lugar e não fazer a troca correspondente no eixo y; vice-versa. Uma sugestão: implemente um tipo ponto usando `strucs` (que contém as coordenadas de um ponto, basicamente), e ao ler o arquivo, você criará um vetor de pontos ao invés de dois vetores de inteiros. Essa é uma excelente oportunidade para se habituar com o uso de `structs`, que serão empregados extensivamente na segunda parte da disciplina.

2 Instruções gerais

- Seu programa deve ser feito em C.
- Não tente embelezar a saída do seu programa com mensagens, formatações, etc. Seu programa será julgado segundo a adequação de sua saída à descrição do exercício.
- Seu programa deve consistir de um único arquivo, e deve ser razoavelmente modular, ou seja: deve consistir de diversas funções que, juntas, realizam a tarefa descrita.

- Documente cada função dizendo o quê ela faz.
- Escreva no início do código um cabeçalho com comentários, indicando nome, número do EP, data, nome da disciplina.
- A entrega será eletrônica no ambiente Moodle da disciplina (não receberei exercícios impressos ou via email).

3 Correção

- Trabalhos copiados (relatório ou programa) são anulados.
- Um programa que não compila vale no máximo 3. Neste caso, leio o código para tentar julgar as tentativas do aluno.
- Um programa mal escrito, desorganizado, recebe 2 pontos de desconto, *mesmo que funcione*. Ler o código leva muito tempo, mas tento fazer o melhor, lendo tudo o que vocês me entregam. Se o programa está organizado, documentado, a leitura é rápida. Se está confuso, às vezes é muito difícil compilar na cabeça o que o aluno faz, e isso leva muito tempo, então, *não vou compilar na cabeça códigos difíceis de ler*. É por isso que esses códigos recebem desconto. Daí a importância de caprichar no código também.

4 Plágio

O problema do plágio de código é endêmico em disciplinas que envolvem programação. Essa prática será fortemente combatida. Você está se enganando se copiar o programa de seu colega. O maior prejudicado é você mesmo.

Neste semestre, vou usar uma ferramenta automática para detecção de plágios em código. Essa ferramenta faz uma análise de um conjunto de arquivos fonte (os EP's) e emite um relatório, indicando a semelhança entre pares de arquivos. Em particular, essa ferramenta (que já tem anos de extrada, e uma *expertise* subjacente) detecta tentativas de burlar o plágio como mudança de nomes de variáveis e alterações na indentação. Ou seja, é muito difícil copiar e não ser descoberto. E infelizmente preciso ser bastante rigoroso caso a ferramenta detecte cópias (veja critérios de correção acima).

Vocês podem e devem pensar em soluções em conjunto. Mas uma vez encontrada a solução, cada um faz seu programa. O máximo que podem fazer é, eventualmente, trocar pequenos trechos sem importância para a solução, por exemplo, código para entrada e saída. E caso o façam, devem indicar com comentários ("esse trecho foi desenvolvido em conjunto com fulano de tal", etc.).

Ao copiar, você também prejudica seu colega que fez o código, porque ambos os trabalhos serão anulados. Melhor evitar essa prática. Se não fez o exercício, melhor não entregar e se concentrar no próximo.