Trabalho Prático 2 — Ordenação de Sistemas Solares

Observações:

1. Comece a fazer este trabalho imediatamente. Você nunca terá tanto tempo para resolvê-lo quanto agora!
2. Data de entrega: 04/12.
3. Submissão: via Moodle.
4. Plataforma computacional: o seu trabalho deve conseguir ser executado em alguma máquina do ambiente computacional do Departamento de Ciência da Computação da UFMG (Linux), onde os monitores irão avaliá-lo.
5. Linguagem: você deve escrever o seu programa obrigatoriamente na linguagem de programação C.
6. Qualquer dúvida converse com os monitores ou professor, o objetivo é que todos consigam fazer o trabalho.

1 Introdução

Na terra da Computação, os avanços tecnológicos estão a todo vapor. Os cientistas desenvolveram um novo dispositivo, capaz de identificar estrelas: o telescoptero. Como se trata de um dispositivo muito simples e divertido de usar, ele passou a ser amplamente adotado pelos computistas. O problema? Com a constante descoberta de novos sistemas solares, os cientistas não sabem por quais astros eles devem começar seus estudos.

No entanto, existe um consenso entre os astrônomos das características que tornam um sistema interessante. Sua tarefa é, ordenar os sistemas solares recém descobertos com base nessas características, assim salvando a vida de milhares de acadêmicos. O governo da Computolândia pretende cortar a verba de pesquisa caso os cientistas não façam uma grande descoberta em breve. Ou seja, será necessário que **você implemente um algoritmo rápido** para a ordenação.

2 Informações Adicionais

O consenso entre os cientistas é que um **sistema solar A é MAIS interessante que um sistema solar B, se:**

1. O raio do sol do sistema A é maior que o raio do sol do sistema B.
2. O sistema A possui mais planetas que o sistema B.
3. O maior planeta do sistema A é maior que o maior planeta do sistema B.
4. O sistema A possui mais luas que o sistema B.
5. A maior lua do sistema A é maior que a maior lua do sistema B.
6. O tempo de descoberta de A é **menor** que o de B.

Perceba que os critérios estão ranqueados, ou seja: na comparação entre dois sistemas, os critérios são comparados em ordem. Por exemplo, se um sistema x possui 100 planetas, mas o raio de seu sol é 5, ele é menos interessante que um sistema y que possui 1 planeta, mas cujo sol tem raio 999, pois o raio é o primeiro critério. Dessa maneira, **os critérios subsequentes SÓ SÃO usados** em caso de empate dos critérios anteriores. Assim, o critério de comparação da maior lua somente será usado se ambos os sistemas possuírem o mesmo raio dos sóis, a mesma quantidade de planetas, o mesmo tamanho máximo de planetas e também a mesma quantidade de luas. É possível que mesmo com todos esses critérios ainda possa haver empate; nesse caso sua função de comparação deve levar em conta o sexto critério: o tempo de descoberta. Como a Computolândia possui apenas um telescoptero, o tempo de descoberta de cada sistema é único e não existem empates nesse caso.

E falando em algoritmo, para atender à demanda dos cientistas de que a classificação dos sistemas deve ser rápida. Por sorte, temos disponível um algoritmo capaz de atender esse requisito desenvolvido pelo astrônomo e cientista da computação Bromero Rito: o *bromerosort*.

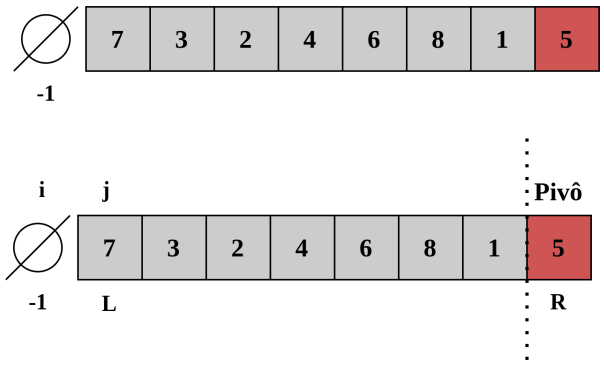
A seguir, é explicado seu funcionamento para que possemos implementá-lo em nossa missão: Considere um vetor $A[l, ..., r]$ de inteiros de tamanho N e defina um elemento de A como sendo o **'pivô'** q a partir de uma política de escolha qualquer. Em seguida, particione os demais elementos em dois subvetores $A[l, ..., q-1]$ e $A[q+1, ..., r]$ onde respectivamente o primeiro contêm os elementos menores que q e o segundo contêm os elementos maiores que q. Agora é hora da magica! Vamos repetir esse mesmo processo para cada subvetor considerando esses como um novo vetor A, ou seja aplicando um procedimento recursivo. Essa operação é realizada sucessivamente até chegarmos no chamado caso base onde temos um subvetor com apenas um elemento que já é trivialmente ordenado. A magia do *bromerosort* está no fato de que estamos combinando de maneira super eficiente a ordenação parcial de vários subvetores de A e os combinando em um único vetor ordenado. Além disso, o algoritmo pode considerar varias politicas de escolha do pivô, que pode ter um resultado mais ou menos eficiente dependo de sua aplicação.

Nosso colega Bromero, também nos forneceu um método base para a a escolha do pivô e a partição do vetor. Esse método que ele chamou de *Ritação*, o algortimo funciona de maneira simples: Primeiro selecionamos como pivô o ultimo elemento do vetor considerado, ou seja, $q = r$. Depois criamos dois índices $i = l - 1$ e $j = l$. Então, iremos mover j iterativamente até antes do pivô na posição $r - 1$. Cada vez que encontrarmos um elemento menor que q incrementaremos i, e realizamos a troca dos elementos $A[i]$ com $A[j]$. Após o *loop*, iremos trocar $A[q]$ com $A[i + 1]$ e retornar o índice $i + 1$ para indicar a fronteira dos dois subvetores repartidos. O segredo para o funcionamento desse procedimento é que estamos movendo a cada iteração os elementos menores que o pivô para direita com o índice j e utilizando i como uma fronteira entre os elementos maiores e menores que o pivô.

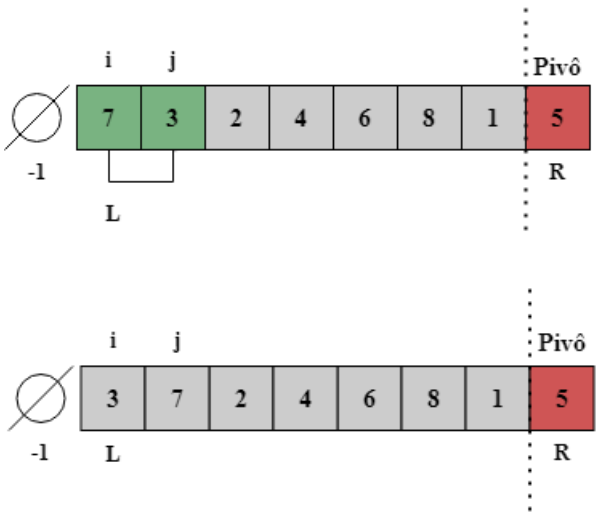
Passo a passo da *Ritação*

Para ilustrar o funcionamento da *Ritação*, vamos particionar uma lista com 8 inteiros. Note, que em nossas imagens a posição -1 está fora dos limites da nossa lista e está indicada com um sinal de vazio.

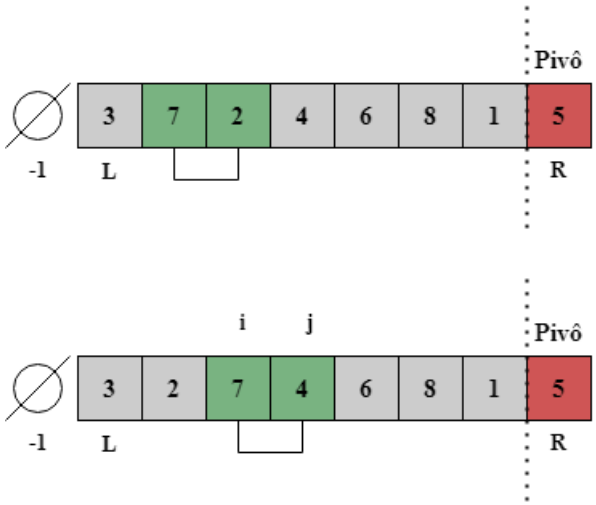
Bem, vamos começar posicionando cada variável de índice que iremos utilizar.



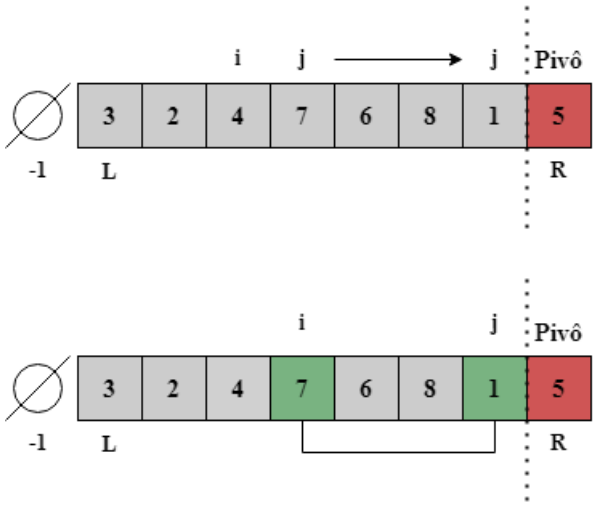
Agora, vamos buscar com o índice j um elemento menor que nosso pivô 5. Uma vez encontrado vamos incrementar o índice o i e trocar os elementos apontados chegando ao resultado da imagem abaixo.



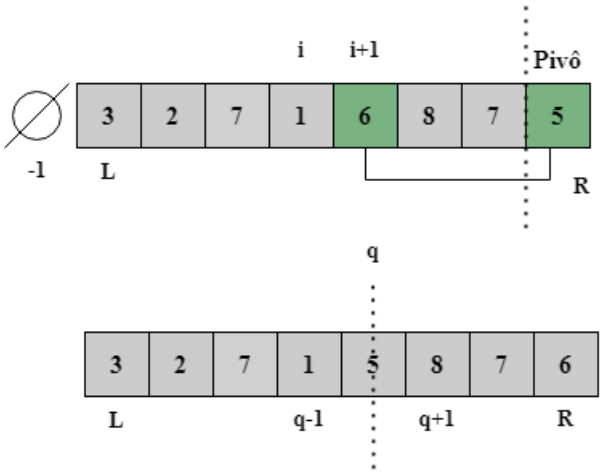
De maneira similar iremos mover o 7 passando pelo 4 e pelo 3. Veja abaixo:



Na sequencia, iremos realizar nossa ultima troca antes do índice j atingir seu limite.

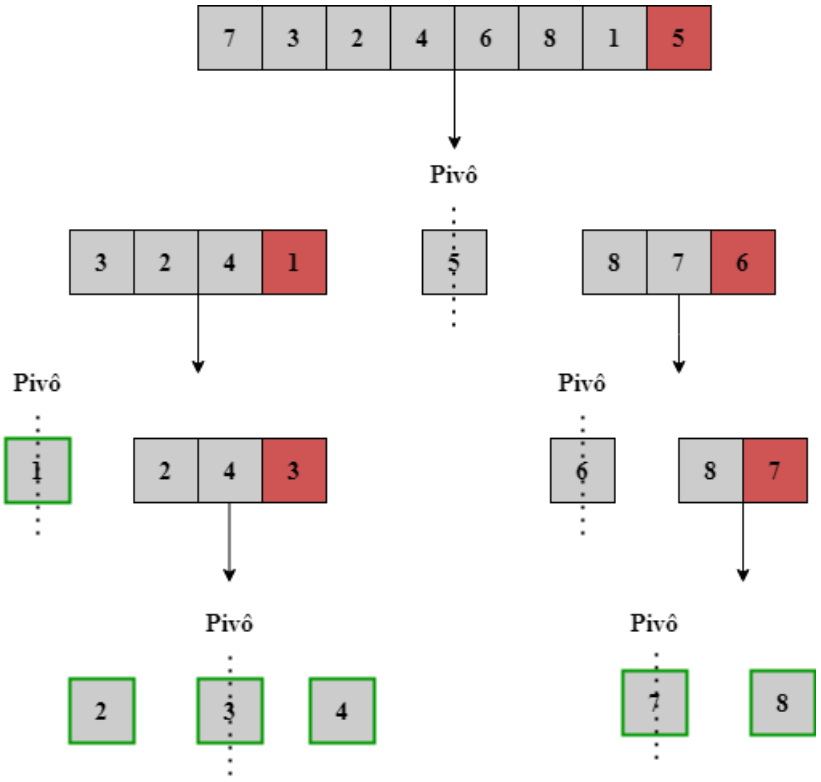


Ao final, basta trocar o elemento pivô com elemento no índice $i + 1$.


i
i+1
Pivô
3
2
7
1
6
8
7
5
-1
L
R
q
3
2
7
1
5
8
7
6
L
q-1
q+1
R

Passo a passo do *BromeroSort*

Uma vez com o procedimento de *Ritação* entendido, podemos visualizar como é o passo a passo do *BromeroSort*. Note que na imagem os quadrados com contornos verdes indicam o caso o base da recursão onde nenhuma troca de elemento é realizado.



Um outro jeito de visualizar método é através de seu pseudocódigo. Veja como é simples:

```
BromeroSort(sistemas, l, r) {
  if l < r:
      q = Ritacao(sistemas, l, r)
      BromeroSort(sistemas, l, q - 1)
      BromeroSort(sistemas, q + 1, r)
}
```

3 Entrada

Convenientemente, os cientistas te enviaram a lista de sistemas solares em um formato padronizado. Primeiro você deve ler a quantidade de sistemas solares, N. Depois você deve ler o tempo de descoberta do sistema T_i, nome do sistema S_i, o raio do seu sol R_i, e a quantidade de planetas que ele possui P_i, tudo em uma linha. Em seguida, é a hora de ler os planetas *daquele* sistema: o nome do planeta, $p_{i,j}$, o raio do planeta $r_{i,j}$ e a quantidade de luas, $L_{i,j}$. Por fim, você deve ler as luas *daquele* planeta: o nome da lua $\ell_{i,j,k}$ e seu raio $q_{i,j,k}$. Perceba que assim que você terminar de ler as luas de um planeta, você vai ler o próximo planeta e suas luas. O mesmo vale para os sistemas: primeiro você lê tudo que tem em um sistema (seus planetas e luas) e **depois** lê o próximo sistema.

A entrada possui as seguintes restrições. Para facilitar, os nomes, raios e quantidades possuem as mesmas restrições[1], ou seja, não existe uma restrição quanto ao raio dos planetas ou das luas, e sim uma restrição de *raios*, no geral. A exceção é o número de sistemas, S, que pode ser maior que o número de planetas / luas (indicados por *length*).

- $1 \leq N \leq 1000000$.
- $1 \leq T_i \leq 1000000$ para todo i.
- todos os nomes têm pelo menos 1 e no máximo 100 caractere s.
- todos os raios são no máximo de tamanho 10000.
- cada sistema tem no máximo 100 planetas e cada planeta tem no máximo 100 luas.

A entrada usa *tabs* (`'\t'`) para facilitar a distinção do que é estrela, planeta, ou lua, como se fosse um código em C. Estrelas não possuem nenhum *tab* na frente, planetas apenas 1 e luas, duas tabulações. Você deve considerar a leitura dos `'\t'` quando for fazer seu programa.

[1] Apesar disso não fazer sentido astronômico (bem como o resto do trabalho).

3.1 Exemplo

Não se preocupe com o nome dos astros, eles foram gerados aleatoriamente!

```
6                          // # Sistemas Solares
3 X1dhcdfx35 557 0           // Sistema #1 | Raio | # Planetas
1 nmok0lee2 257 2            // Sistema #2 | Raio | # Planetas
  4V8VP5P 743 2            // Planeta #1 | Raio | # Luas
    SKZJqLg 538            // Lua #1 | Raio
    23VFgWcFM3 978         // Lua #2 | Raio
  ueJSO 805 1              // Planeta #2 | Raio | # Luas
    phuVmf 484             // Lua #1 | Raio
2 jW2Ol 889 1                // Sistema #3 | Raio | # Planetas
  LUTmBvC8C 901 1          // ...
    zzMpnv 636
20 wP3OHu 944 3
  nBSykm3G 187 1
    Hbf1J58 932
  1pUA1xD1wO 980 4
    bfFx 357
    0MkEM 438
    cSLdX 222
    XHbj8C 137
  60yzYmj 316 3
    WZepWXuYaW 471
    u4Fz2 418
    kqYsyi 9
11 kNahmOcQLr 298 3
  71wIO 468 3
    Ryl4D3A 358
    QBPGgLD 659
    6bsIGvPap 311
  8ke2ice 348 0
  YxjJRC 645 4
    r6TwFh 873
    u8bBOSu 736
    3dxmIJWNT 377
    ih6PPWGt 611
40 9YRGgCqY 453 1
  t7ipn3 207 1
    8CwDd 268
```

Saída Esperada

```
wP3OHu      // Maior tamanho
X1dhcdfx35  // Segundo maior tamanho
jW2Ol       // Empate no tamanho e # de planetas, mas planeta MAIOR
9YRGgCqY    // Empate no tamanho e # de planetas, mas planeta menor
kNahmOcQLr  // Empate no tamanho, mas mais planetas
nmok0lee2   // Empate no tamanho, mas menos planetas
```

Obviamente você não deve imprimir os comentários, no exemplo eles são apenas didáticos. Perceba, também que os sistemas devem ser impressos na ordem inversa da ordenação ("menor para o maior"), uma vez que os cientistas estão interessados em estudar os sistemas MAIS importantes ("maiores") primeiro. O nome dos planetas e das luas não deve ser impresso, **apenas** os nomes dos sistemas.

4 Avaliação

O trabalho deve ser entregue em um único arquivo com extensão `.c`. O trabalho vale 16 pontos e sua avaliação é divida em implementação e documentação. A implementação vale 12 pontos e é calculada com base no número de acertos nos casos de teste, definidos pelos monitores. Você não terá acesso a todos os casos de teste, mas deve garantir o funcionamento correto do seu programa independentemente disso. Sua implementação possui algumas restrições:

- Você **DEVE** usar **ALOCAÇÃO DINÂMICA**.
  Você **NÃO PODE** usar alocação estática.
- Você **DEVE** usar o algoritmo de ordenação descrito no enunciado.
  Você **NÃO PODE** usar outro algoritmo.

Se você usar alocação estática ou implementar outro algoritmo, **você terá a implementação do seu trabalho ZERADA** (você ainda pode conseguir pontos de documentação). Neste trabalho, performance é importante e será definido um tempo máximo de execução para os programas de **5 segundos por caso de teste**.

A documentação vale 4 pontos. Ela é analisada conforme a organização do código, que deve possuir comentários explicativos e significativos, variáveis com nomes intuitivos e estruturação clara, que permita maior facilidade de compreensão.

Casos de cópia entre alunos não serão tolerados. Os códigos serão comparados.