**Instituto de Computação - Unicamp**

**MC558 - Projeto e Análise de Algoritmos II**

# Laboratório 04 - Corrida Maluca

Prazo de entrega: **07/11/2016 às 23:59:59**

*Professor:* Orlando Lee
*Monitores:* Guilherme Bueno Andrade (PAD) e Maycon Sambinelli (PED)

**Aviso: Leia a seção "Tirando 3 no relatório"**

## Descrição

Seu amigo Richard P. Feynman foi sorteado para participar da corrida maluca, que é um evento promocional criado para premiar os clientes de uma grande rede de supermercados.

A corrida funciona da seguinte forma. Você pega um carrinho de compras na entrada do supermercado e deve chegar à saída, passando por corredores de mão única que valem uma certa quantidade de pontos. O objetivo da corrida maluca é chegar à saída sem estourar o limite de pontos, isto é, a soma do número de pontos dos corredores percorridos no trajeto deve menor ou igual ao limite pré-estabelecido. Se você conseguir chegar a saída sem estourar o limite de pontos, então o seu prêmio é igual ao maior número de pontos de um corredor percorrido por você.

Feynman sabe que o valor máximo do prêmio que ele poderá receber não é necessariamente o maior número de pontos de um corredor no supermercado. Sabendo que você é um ótimo programador, Feynman pediu a sua ajuda para criar um programa que calcule o valor máximo de um prêmio que ele poderá ganhar.

## Entrada

A primeira linha da entrada consiste em um único inteiro $p$ $(1 \leq p \leq 1000)$ representando a quantidade de casos de teste da entrada. A primeira linha de cada caso de teste consiste em cinco inteiros: $n$ $(2 \leq n \leq 1000)$, o número de locais (entrada, saída, pontos iniciais/finais de corredores); $m$ $(0 \leq m \leq \frac{n(n-1)}{2})$, o número de corredores; $s$ $(0 \leq s \leq n-1)$, o local de entrada; $t$ $(0 \leq t \leq n-1)$, o

local de saída; e $r$ $(0 \leq r \leq 1000000)$, o valor limite de pontos. As próximas $m$ linhas contêm, cada uma, três inteiros $u$ $(0 \leq u \leq n-1)$, $v$ $(0 \leq u \leq n-1)$ e $c$ $(0 \leq c \leq 500)$, indicando que é possível ir do local $u$ ao local $v$ por um corredor que vale $c$ pontos.

## Saída

Para cada caso de teste, seu programa deve imprimir um único inteiro indicando o valor máximo do prêmio que Feynman pode receber. Se não for possíve ir de $s$ a $t$ sem estourar o limite de pontos, então seu programa deve imprimir $-1$.

## Exemplos

**Teste 01**

Entrada

```
2
4 5 0 3 15
0 1 1
0 2 2
2 1 11
1 3 2
2 3 5
5 6 1 4 6
0 4 4
1 2 2
1 4 10
1 0 3
2 3 1
3 4 5
```

Saída

```
11
-1
```

**Teste 02**

Entrada

```
2
4 4 0 1 16
0 3 1
3 2 10
2 0 1
0 1 2
4 4 0 1 13
0 3 1
3 2 10
2 0 1
0 1 2
```

Saída

```
10
2
```

Para mais exemplos, consulte os testes abertos no Susy.

## Restrições

O seu programa deve atender as seguintes restrições:

- Complexidade: $O((n + m) \lg n)$

## Relatório

Além do programa, você deve escrever um pequeno relatório explicando a ideia utilizada para resolver o problema e fazer a análise de complexidade de tempo do seu programa (não é necessário provar a corretude do algoritmo). O relatório deve conter no máximo uma página, deve estar no formato pdf e deve ser submetido pelo SuSy.

## Tirando 3 no relatório

Esta seção contém algumas dicas para que você tire a nota máxima no laboratório. Ela foi construída levando em consideração os erros mais comuns encontrados durante as correções dos laboratórios.

**Faça a análise enquanto explica**

Apesar de ninguém perder pontos diretamente por fazer a explicação da solução separadamente da explicação da análise, existem algumas vantagens ao se fazer as duas simultaneamente.

A primeira que podemos apontar é com relação a espaço. Se você explica um passo do algoritmo e não apresenta a sua complexidade logo em seguida, posteriormente quando você fizer a análise deste passo, você terá que referenciá-lo novamente. Note que esta segunda referência não ocorreria se a análise fosse apresentada logo em seguida à explicação. Como o relatório é limitado a apenas uma página, este espaço desperdiçado pode fazer falta na hora de explicar a sua solução, deixando-a confusa e consequentemente levando-o a perder pontos.

A segunda vantagem que podemos apresentar é referente à facilidade de compreensão. Por exemplo, analise os dois casos abaixo e veja o que é mais claro de entender:

1. "O terceiro passo consiste em verificar se todos os vértices de $G_1$ estão presentes em $G_2$. Para isso, verificamos se os extremos de cada aresta de $G_1$ estão presentes no dicionário labels. Isso é feito em $O(m_1 n_2)$".

2. Na seção da explicação da solução escrevemos "O terceiro passo consiste em verificar se todos os vértices de $G_1$ estão presentes em $G_2$". Em um lugar aleatório na seção de análise, colocamos antes mesmo da análise de leitura da entrada "A função `verificaG1EmG2` verifica se todos os vértices de $G_1$ estão presentes em $G_2$. Isso é feito em $O(m_1 n_2)$".

Claramente o primeiro caso está mais claro. É provável que no momento da análise do segundo exemplo, o leitor do seu relatório já não lembre mais do funcionamento do seu algoritmo. Além disso, não seguir o fluxo do seu algoritmo durante a análise (o que apareceu com uma certa frequência nos relatórios) ajuda a deixar a sua explicação ainda mais confusa.

**Níveis de detalhes**

Uma dúvida frequente é qual o nível de detalhes que o relatório deve conter. Errar esse nível tanto para mais quanto para menos pode levar a perda de pontos. Ao detalhar mais do que o necessário um passo do algoritmo, você está perdendo espaço que poderia ser usado para explicar melhor um outro passo e, em um relatório de uma página, todo espaço conta. Ao detalhar de menos, o leitor não é capaz de entender a sua solução, o que ocasiona perda de pontos. Uma dica para tentar calibrar a granularidade da explicação é responder a seguinte pergunta:

```
Alguém que ler o meu relatório conseguirá implementar a minha solução?
```

Aqui cabe uma ressalva. A pergunta é "implementar a minha solução" e não "implementar o meu código", ou seja, resolver o problema da mesma forma que você, com a mesma ideia, mas não necessariamente gerando o mesmo código. Por exemplo, você poderia simplesmente dizer "em seguida fazemos uma ordenação topológica usando uma DFS", não importando quem você passa por parâmetro, quem é global, ou se usa um `for` ou `while` para percorrer a vizinhança de um vértice. Esse nível de detalhe de implementação só serve para desperdiçar espaço e deixar o seu relatório mais confuso.

Uma outra maneira de identificar se a granularidade da explicação do passo está adequada é olhando a análise de complexidade. Por exemplo, digamos que você escreva algo como "Na sequência o algoritmo faz X. Este passo é feito em $O(f(n))$". Deveria ser claro o porquê de "X" poder ser feito em $O(f(n))$. Exemplo, "Copiamos os elementos pares do vetor `numeros` no vetor `numeros_pares`. Este passo tem complexidade $O(n)$, onde $n$ é o número de elementos do vetor `numeros` ". É bem claro para alguém que sabe programação que este passo pode ser feito dentro da complexidade apresentada. Existem dois motivos do porque não estar claro que "X" pode ser feito em $O(f(n))$:

1. **Passo Genérico**. O seu passo está muito genérico/geral, por exemplo: "então contamos todos os caminhos de $s$ a $t$, isso pode ser feito em $O(n+m)$". Neste caso, quebre a explicação do passo em passos menores. Este é o principal motivo de perda de pontos na parte da análise.

2. **Malicia na análise**. Em alguns poucos casos, você não tem como detalhar mais e ainda assim não é claro o porque da complexidade do passo. Isso acontece porque é necessário enxergar um detalhe, uma "malicia" no passo. Nesses casos, o que você deve fazer é justificar melhor a análise.

Cabe ressaltar aqui que algoritmos clássicos vistos em sala de aula não precisam de uma análise ou explicação detalhada, ou seja, você pode usar algo como "então aplicamos o algoritmo de Dijkstra cuja lista de prioridades foi implementada com uma heap binária e portanto tem complexidade $O(v \lg v + m \lg n)$". No entanto, se você modificou o comportamento padrão do algoritmo de Dijkstra de alguma forma, você precisa explicar como foi essa modificação e como isso afeta a complexidade do algoritmo.

**Código**

Não assuma que o monitor olhará o seu código! Por esse motivo, referenciar linhas e trechos de código é inútil. O monitor olha o seu código nos seguintes casos:

1. Suspeita que a sua solução violou alguma restrição, por exemplo, de complexidade.
2. Seu relatório está muito confuso.

### Detalhe de implementação

Dissemos que não se deve apresentar detalhes de implementação no relatório. Porém, existe um detalhe de implementação que é crucial e não deve faltar em hipótese alguma: as estruturas de dados utilizadas. Isso se deve pelo fato dela ter um papel fundamental na análise de complexidade. Desta forma, nunca esqueça-se de especificar qual estrutura você está utilizando.

### Estude o gabarito

O relatório fornecido como gabarito dos laboratórios é preparado com muito cuidado de forma que seria considerado um relatório excelente se fosse enviado por um aluno. Portanto, recomendamos que estudem a estrutura desses relatórios, a granularidade dos passos e os usem como modelo na preparação para os próximos relatórios. Cabe notar que não estamos dizendo que o modelo/estrutura dos relatórios apresentados nos gabaritos é o único jeito correto de se fazer o relatório, mas sugerimos fortemente que ele seja seguido.

### Leia o seu relatório antes de enviar

Muitos relatórios são enviados com frases que não fazem nenhum sentido e que seriam facilmente identificadas com uma breve leitura.

## Critérios de avaliação

A nota máxima do laboratório é 10 e é dada pela seguinte formula:

$$NF = NP + NR - PE$$

Onde:

- $NF$ é a nota final.
- $NP$ é a nota do programa. Esse valor é igual a 7 se o programa passou em todos os casos de teste do SuSy. No caso do programa ter falhado em um ou mais casos de teste, temos que $NF = 0$ (Note que neste caso é a nota final que é igual a zero e não a nota do programa).
- $NR$ é a nota do relatório (3 pontos).
- $PE$ é a soma das penalidades aplicadas ao programa. O valor das penalidades é apresentado na tabela abaixo.

| Valor | Descrição |
|---|---|
| -2 | Programa com problemas relevantes de qualidade de código (Falta de comentários, nomes não significativos e etc) |

| Valor | Descrição |
| --- | --- |
| -3 | Programa que violar alguma das restrições apresentadas na seção *Restrições* |

## Observações

- O número máximo de submissões é **15**;
- O seu usuário no SuSy é o seu número de RA (apenas números) e a sua senha é a sua senha da DAC.
- Indente corretamente o seu código e inclua comentários no decorrer do seu programa.
- O SuSy utiliza as seguintes flags de compilação: `-std=c99 -pedantic -Wall -lm` para a linguagem C e `-ansi -pedantic -Wall -lm` para a linguagem C++.
- Para efeito de avaliação será levado em conta apenas a última submissão no SuSy. Arquivos fontes mandados por email não serão levados em conta.

## Plágio

O reaproveitamento de código da Web ou de colegas é considerado plágio e será tratado de acordo com os critérios estabelecidos.