PxNP

Trabalho 1 de MC458 - Entrega: 14/05

Um parque famoso pela sua grande quantidade de montes e pelas belas vistas que estes proporcionam é o Parque exótico Nacional dos Picos (PxNP). Uma das atividades que são realizadas no parque é uma excursão pelos montes para apreciar as paisagens. Para tornar a excursão mais interessante e prática, os montes são interligados por tirolesas. Assim, para cada par de montes, existe exatamente uma tirolesa que os conecta, que logicamente só pode ser usada em um sentido.

Você ficou responsável por determinar uma rota para a excursão no parque. A excursão pode começar por qualquer monte, e não se deseja passar por um mesmo monte mais de uma vez. O PxNP é composto por um total de **N** montes, e você é capaz de consultar se há uma tirolesa que conecta um monte ao outro. Sua tarefa é determinar uma sequência de montes que podem ser visitados de forma a visitar o maior número de montes possível.

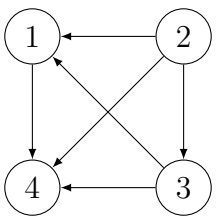
Entrada e Saída

Neste trabalho você não tem acesso direto às instâncias. Ao invés disso, lhe é fornecido uma função `has_edge(i,j)` que informa em tempo constante se há uma tirolesa que conecta os montes **i** e **j**, nesse sentido.

De posse dessa função, você deve completar uma rotina `solve(int N)`, onde **N** é o número de montes, e os montes são rotulados com inteiros de um **1** a **N**. Tal rotina deve retornar um vetor que representa a sequência de montes a serem visitados pelo caminho.

Exemplo

Considere o seguinte exemplo com $\mathbf{N} = 4$ montes:



Neste caso existe uma rota que passa pelos 4 montes, dada pela sequência $2, 3, 1, 4$.

Restrições

- 50% dos casos de teste: $1 \leq \mathbf{N} \leq 10^3$;
- 50% dos casos de teste: $1 \leq \mathbf{N} \leq 10^5$;
- Tempo limite de execução: 1 segundo.

Sua nota será dada pela quantidade de casos de teste corretos e dentro do tempo limite que seu algoritmo é capaz de resolver. Note que os testes não vão apenas verificar corretude, mas também a complexidade de tempo para resolver o teste. Assim, programas corretos mas com complexidade de tempo alta provavelmente não resolverão a tempo os testes maiores. Para obter uma estimativa da complexidade de tempo esperada, pode-se estimar que a máquina é capaz de realizar até uma quantidade da ordem de 10^7 instruções dentro de 1s. Com isso, note que um algoritmo com complexidade $\mathcal{O}(\mathbf{N}^2)$ não deve passar os casos de teste maiores. **Se seu algoritmo tem uma complexidade assintótica superior à esperada e mesmo assim passe testes maiores dentro do tempo limite, tais testes poderão não ser considerados certos.**

Instruções para submissão

- Você deve entregar um código chamado `solution.cpp`, além de um relatório que explica seu algoritmo.
- A entrega do código será feita pelo SuSy, pelo link `https://susy.ic.unicamp.br:9999/mc458a/1`.
- O código deve ser escrito em C++ e deve ter o nome `solution.cpp`. Serão utilizados os parâmetros de compilação `-std=c++11 -Wall -lm`.
- Na página do SuSy, em Arquivos auxiliares, é fornecido um template com os arquivos `solution.cpp` e `lib.hpp`. Em `solution.cpp` você deve completar a rotina `solve(int n)`, como explicado anteriormente. No header `lib.hpp` é definido a função `has_edge(int i, int j)`. Este header **não** é o mesmo que será utilizado na submissão. Ele tem como propósito apenas a realização de testes locais. Portanto, você pode modificar as funções de `lib.hpp` à vontade para testar instâncias específicas localmente. Você deve submeter apenas o arquivo `solution.cpp`. **Não modifique** o `main` nem a linha `#include "lib.hpp"` de `solution.cpp`.
- É permitido um total de 20 submissões. **Apenas a submissão mais recente será considerada**.

- A entrega do relatório será feito pelo Classroom, na tarefa correspondente, através de um documento em `pdf`. O relatório deve contemplar:

    1. **A descrição do algoritmo**: explique a ideia do seu algoritmo textualmente (**não** coloque um print do seu código; o objetivo do relatório é justamente ser capaz de entender o funcionamento do seu algoritmo com palavras, sem ter que ler um código). Também justifique a corretude do seu algoritmo.
    2. **A análise da complexidade de tempo**: faça a análise de pior caso do algoritmo e de eventuais estruturas de dados não triviais que tenham sido usadas. Se for pertinente, dê a complexidade de caso médio também.

    Na correção do relatório, será avaliado a corretude da explicação do algoritmo e da análise da complexidade de tempo, bem como a clareza da explicação.