Introdução à Otimização Combinatória Aplicada

Primeiro Trabalho Prático
Entrega Sugerida: 06/06/2020

1 Introdução

No primeiro Trabalho Prático (TP01) será solicitada a entrega de um arquivo **.zip** contendo um programa que solucione o problema apresentado na próxima seção, bem como, para cada instância, um arquivo **.sol** contendo a solução que seu programa encontrou para a mesma. Por conta de detalhes do corretor, no **.zip** também devem estar os arquivos **Makefile** e **verifier.py**, que são disponibilizados junto do trabalho. Além disso, os itens a seguir devem ser respeitados:

- O TP01 deve ser feito individualmente e plágio não será tolerado;
- O TP01 deve ser entregue no run codes (https://run.codes);
- Seu programa deve apresentar um código em uma das seguintes linguagens (C, C++, Java, Python 3) e deve conter um cabeçalho com informações do estudante, como: nome, curso (caso se aplique) e RA (ou RG, se for membro externo);
- Cada estudante deve se cadastrar no run codes (https://run.codes) informando Nome Completo, escolhendo "UFSCar - Universidade Federal de São Carlos" no campo Universidade e colocando seu RA (ou RG, se for membro externo) no campo Núm. Matrícula. Depois de cadastrado, basta logar no run codes e se matricular na disciplina "IOCA - Introdução à Otimização Combinatória Aplicada" usando o Código de Matrícula EZMQ.
- Caso utilize PL ou PLI, pode usar resolvedores como OR-Tools, CPLEX e Gurobi.

2 Agente em Teste

Você é um agente iniciante da CIA e está se preparando para sua primeira missão ultrasecreta. A Agência te oferece uma série de pequenos utensílios que são categorizados por dois valores: um determinando quanto ele pode ajudar em sua missão, e outro indicando seu custo monetário. Como você é um novato e está em teste, seria irresponsável colocar em seu poder um conjunto muito valioso de itens, e assim foi determinado um limite para o custo monetário total dos itens que você poderá carregar. Além disso, você precisa tomar cuidado, pois existem pares de itens tão perigosos quando juntos, que escolher os dois poderia custar a sua vida!

Considerando que você quer sair vivo dessa (e impressionar a Agência), escolha o conjunto de itens que será o mais útil possível, sem ultrapassar o custo máximo estipulado, e que não coloque sua vida em mais risco ainda!

Entrada:

Cada instância contém um único caso de teste. A primeira linha contém três inteiros: o número de utensílios disponíveis N, o valor máximo que você poderá carregar K, e a quantidade R de pares de itens que não podem ser escolhidos juntos. As próximas N linhas contêm as informações dos utensílios. Especificamente, cada linha tem dois números inteiros, o primeiro indicando quanto o item poderá te auxiliar, e o segundo correspondendo ao custo monetário do item. Por fim, as R linhas seguintes guardam as informações dos pares conflitantes. Cada linha possui novamente dois números inteiros, A ($0 \leq \text{A} < N$) e B ($0 \leq \text{B} < N$), indicando que você não deve levar para a missão os itens A e B ao mesmo tempo (ou seja, caso tenha escolhido levar A, não poderá levar B, e vice-versa).

Saída:

A saída de seu programa deve ser composta por duas linhas. A primeira contendo obj, a utilidade total que indica o quanto seu conjunto de itens poderá te auxiliar. A próxima linha é uma lista de N valores, um para cada utensílio u: u_i deverá ser 1 caso você tenha escolhido levar o item de índice i, e 0 caso contrário.

Exemplo de entrada

No exemplo abaixo temos um cenário com 4 itens, custo monetário máximo igual a 11, e 1 conflito entre o primeiro e segundo item.

```
4 11 1
8 4
12 5
15 8
4 3
0 1
```

Saída esperada para esse exemplo

Neste resultado estamos escolhendo os dois últimos itens, que juntos resultam em um valor 19 de utilidade.

```
19
0 0 1 1
```

3 Instruções

Disponibilizamos o trabalho no arquivo **tp01.zip**. Este contém o arquivo **solver.py** que realiza a leitura de uma instância, executa um algoritmo básico para o problema, desconsiderando os

conflitos, devolve uma solução no formato de saída descrito acima, e grava essa solução no arquivo <**nome_da_instancia**>**.sol** na mesma pasta da instância lida. Modifique a função **knapsackNaive()** do **solver.py** para resolver o problema proposto. Sua implementação pode ser testada com o comando

```
python ./solver.py ./data/<nome_da_instancia>
```

Você pode escolher entre implementar sua solução diretamente em python, modificar a função para chamar uma aplicação externa, ou desenvolver um código na linguagem que desejar, que substitua o **solver.py**.

As instâncias que devem ser resolvidas também estão no **tp01.zip**, mais especificamente na pasta **./data/**. Também disponibilizamos algumas instâncias menores na pasta raiz, cujas soluções não serão cobradas, mas que podem ser úteis nos seus testes durante a implementação.

Sua submissão deve corresponder a um arquivo **.zip** que contém, além do seu programa, um arquivo <**nome_da_instancia**>**.sol** para cada instância. O conteúdo desses deve ser a solução encontrada por seu programa ao resolver a instância correspondente. No **.zip** também devem estar os arquivos **Makefile** e **verifier.py**, que estão disponíveis na pasta **./sols/** do **tp01.zip**. Em particular, vocês podem usar o programa **verifier.py**, que valida as soluções encontradas, para facilitar seus testes durante a implementação.

Cálculo da Nota

Soluções inviáveis (i.e., aquelas que não seguem o formato de saída ou violam as restrições do problema) irão receber 0 pontos. Soluções viáveis irão receber pelo menos 3.33 pontos. Soluções viáveis superando um primeiro padrão de qualidade irão receber 6.66 pontos, e soluções que superam um alto padrão de qualidade irão receber 10 pontos. No run codes existem testes distintos, com a mesma instância, para verificar esses diferentes critérios. Para ser aprovado, o estudante deve obter pelo menos 70% dos pontos do trabalho.