Universidade Federal do Ceará, Departamento de Estatística e Matemática Aplicada
Programação Inteira (CC0399), período 2023.1
Professor: Tibérius O. Bonates (`tb@ufc.br`).

Atividade 1 – Preprocessamento e algoritmo guloso – Set Cover.

1 Descrição

Esta atividade consiste em escrever um programa em Python ou C/C++[1] que implementa o algoritmo guloso para o problema *Minimum Set Cover*, além das técnicas de preprocessamento discutidas em sala de aula e enunciadas na seção 3. Se seu código for escrito em Python, não envie um Python notebook; envie apenas o(s) arquivo(s) "`.py`".

Seu programa deve necessariamente realizar tudo que está descrito neste documento. Como a tarefa é simples e boa parte do código solicitado pode ser encontrado na Web ou obtido via chatbots, a correção será rigorosa, assim como será a penalização por qualquer desvio em relação às instruções presentes neste documento. Não haverá exceções.

Arquivos de entrada serão gerados por meio do gerador fornecido, no seguinte formato: uma matriz de 0s e 1s, contendo n linhas e m colunas, onde n é o número de objetos e m é o número de subconjuntos. Seu código deve ler um arquivo com nome fixo, chamado `entrada.txt`, localizado na mesma pasta que o seu código. Seu código não deve solicitar o nome do arquivo, nem deve usar outro nome de arquivo.

2 Notação

Na descrição desta atividade, vamos usar a seguinte definição do problema:

- Conjunto-base finito, contendo n elementos: $S = \{e_1, \ldots, e_n\}$;
- Família contendo m subconjuntos não-vazios de S: $\mathcal{F} = \{S_1, \ldots, S_m\}$;
- Deseja-se minimizar o número de subconjuntos selecionados, cuja união deve ser igual a S.

3 Tarefa

A seguir, detalhes sobre a tarefa.

3.1 Parte 1: preprocessamento

Seu código deve realizar três tipos de simplificação.

Etapa 1: Se um objeto pertencer a apenas um dos subconjuntos S_j, então a variável x_j deve necessariamente assumir valor 1 em qualquer solução do problema. O resultado desta detecção deve ser o de que a variável x_j é removida do problema. Da mesma forma, todas as restrições que envolvem a variável x_j devem ser removidas do problema, uma vez que estão automaticamente satisfeitas.

Etapa 2: Levando em conta as restrições e variáveis que sobrarem após esta simplificação, seu programa deve detectar todos os casos em que uma restrição é *implicada* por outra. Isto é, quando existirem objetos e_i e e_j tais que o conjunto de subconjuntos que cobrem e_i é um subconjunto (próprio ou não!) do conjunto de subconjuntos que cobre e_j. Neste caso, a restrição correspondente ao objeto e_j deve ser removida. Dizendo ainda de uma terceira forma, isto significa que as variáveis presentes na restrição correspondente ao objeto e_i estão todas presentes na restrição correspondente ao objeto e_j.

[1] Se sua equipe desejar usar outra linguagem de programação, consulte o professor com urgência.

Etapa 3: Levando em conta as restrições e variáveis que sobrarem após estas simplificações, seu programa deve detectar todos os casos em que um subconjunto S_a é um subconjunto (próprio ou não!) de um subconjunto S_b. Neste caso, a variável correspondente ao subconjunto S_a deve ser removida do problema.

Repita as três simplificações até que nenhuma variável ou restrição tenha sido removida em nenhuma das três etapas. Isto é necessário porque pode haver interações entre as etapas: quando uma etapa realiza uma simplificação, isto pode fazer com que outra etapa possa realizar uma nova simplificação que antes não era possível.

Ao final do processo, seu código deve fornecer o número total de variáveis e de restrições remanescentes. Nada deve ser impresso antes desta informação.

3.2 Algoritmo guloso

Seu código deve aplicar, após a fase de preprocessamento, o algoritmo guloso clássico para o problema *Minimum Set Cover*, visto em sala de aula e apresentado no pseudocódigo do Algoritmo 1.

Algorithm 1: Pseudocódigo do algoritmo guloso para *minimum set cover*.

```
INPUT  : Instância preprocessada: conjunto-base S e família F = {S_1, ..., S_m}.
OUTPUT: Solução viável C.
C := ∅ // Solução a ser construída
U := S // Elementos ainda não cobertos
Enquanto ainda houver elementos não cobertos
while U ≠ ∅ do
    S* ∈ argmax{|S_i| : S_i ∈ F} // Subconjunto com maior qtde de elementos não cobertos
    C := C ∪ {S*} // Incluir S* na solução em construção
    F := F − {S*} // Remover S* da lista de subconjuntos disponíveis
    U := U − S* // Atualizar conjunto de elementos não cobertos
    for S_j ∈ F do
        S_j := S_j − S* // Remover de cada subconjunto os elementos recém-cobertos
        // Se algum subconjunto se tornar vazio nesse processo...
        if S_j = ∅ then
            F := F − {S_j} // Remova-o da família F
Retornar C
```

Ao final da execução do algoritmo guloso, a única impressão na tela deve ser o tamanho da solução (isto é, o tamanho do conjunto C). Nada mais deve ser impresso como parte do algoritmo guloso.

Revise o que deve ser impresso e não tome a decisão sobre imprimir mais (ou menos) do que aquilo que for pedido.

4 Entrega

- Pontuação da atividade: 2,0 (dois pontos) na primeira nota. Essa pontuação não é extra.

- Entregável: código-fonte comentado e funcional.
- Data de entrega: 15 abril de 2023, 23:59:59, via SIGAA.
- Não será aceito envio via e-mail ou fora do prazo. Evite decepções. O prazo é dimensionado com tempo mais do que suficiente para resolução da atividade. Para evitar surpresas, como interrupções de conectividade, você deve considerar submeter sua resolução com antecedência.

5 Mais detalhes

- A atividade pode ser feita individualmente ou em dupla. A definição dos membros da equipe deve estar no código. Os comentários no formulário de envio do SIGAA não serão levados em conta.
- Se for necessário submeter múltiplos arquivos, compacte-os em um só arquivo antes do envio.