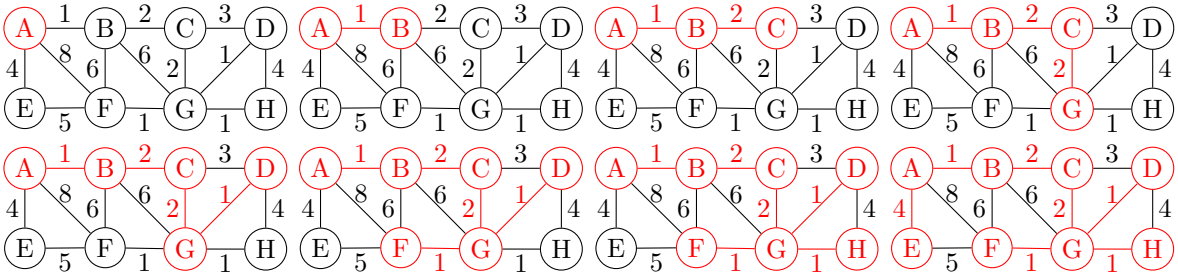
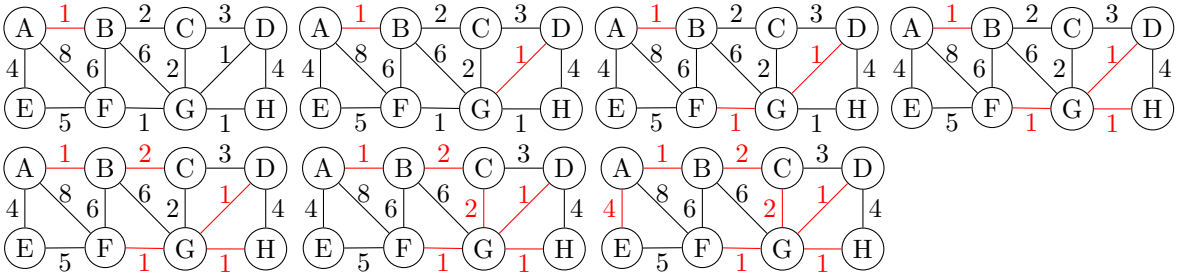
Lista de soluções 2

Exercício 1

a) O algoritmo de Prim passa pela seguinte sequência de passos



b) O algoritmo de Kruskal passa pela seguinte sequência de passos



Exercício 2

Vamos supor que todos pesos são diferentes. Usaremos a seguinte propriedade útil nas provas:

Lema 0.1 (Propriedade do ciclo)
Seja C um ciclo em G. A aresta mais pesada de C não faz parte de nenhuma árvore espalhada mínima.

Prova. Seja T uma árvore espalhada mínima que contém a aresta mais pesada e de um ciclo C. A deleção de e separa T em dois componentes com vértices S e $V \setminus S$. No ciclo C existe uma outra aresta e' com um vértice em S e outro em $V \setminus S$ com peso menor que e. Por um argumento similar com aquela da propriedade de corte podemos ver que o conjunto $T' = T \setminus \{e\} \cup \{e'\}$ é conexo e acíclico, i.e., uma árvore espalhada. Mas T' tem peso menor que T, uma contradição com a minimalidade de T. ■

a) Caso o peso de uma aresta que não faz parte de T aumenta, T continua a ser mínima. Prova: O peso de T é constante, e o peso de uma árvore geradora arbitrária é maior ou igual após o aumento. Logo T continua ser mínima.

b) Caso o peso de uma aresta que não faz parte de T diminuiu, é possível que existe uma nova árvore geradora de menor peso total. Caso ela existe, ela tem que conter a aresta e. Senão teríamos uma contradição com a minimalidade de T antes da diminuição. Logo, adicionaremos a aresta e à árvore T. Isso fecha um ciclo. Desse ciclo removeremos a aresta de maior peso e' para obter uma nova árvore T'. Isso pode ser feito tempo $O(m+n)$. A árvore T' é uma árvore geradora mínima.

   Prova: Considera uma aresta $e'' \notin T'$ arbitrária. Provaremos que e'' é a aresta mais pesada no ciclo C obtido pela inserção de e'' em T'. Neste caso, pela propriedade do ciclo e'' não faz parte de nenhuma árvore geradora mínima. Logo T' tem que ser a árvore geradora mínima.

   Caso C contém somente arestas de T, e'' é a aresta mais pesada em C, porque já foi isso antes da diminuição. Caso contrário, C contém algumas arestas do ciclo C' formado inserindo e'' em T, e do ciclo C'' formado inserindo e em T, com exceção de e'. Mas e'' pesa mais que e', por não fazer parte de T, e e' é a aresta mais pesada em C''. Logo e'' pesa mais que todas arestas em C (ver também Figura 1(a)).

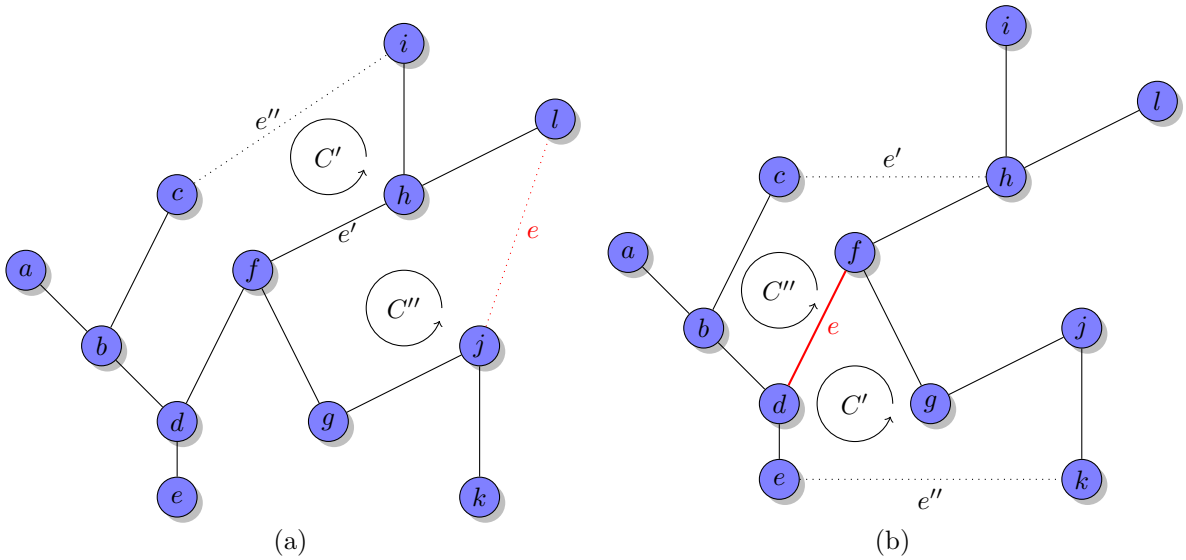


Figura 1: (a) Diminuir o peso de uma aresta fora da árvore geradora mínima; (b) Aumentar o peso de uma aresta que faz parte da árvore geradora mínima.

c) Caso o peso de uma aresta da árvore geradora mínima T diminui, T continua a ser mínima. Prova: O peso total de T diminui por $\Delta = p_e - p'_e$, e o peso de uma árvore geradora arbitrária diminui no máximo por Δ (caso ela contém e). Logo T continua ser mínima.

d) Caso o peso de uma aresta da árvore geradora mínima T aumenta, é possível que existe uma nova árvore geradora de menor peso total. Removendo a aresta e resulta em dois componentes conectadas S e $V \setminus S$, com as duas sub-árvores conectando cada componente com o menor peso total. Conectar os dois componentes com a aresta mais leve e' obtemos uma árvore T', que é a árvore geradora mínima.

   Prova: Novamente provaremos que uma aresta $e'' \notin T'$ arbitrária é a mais pesada no ciclo C obtido pela inserção de e'' em T'. Similar com o caso b) isso é claro caso C contém somente arestas de T. Caso contrário C contém algumas arestas do ciclo C' formado pela inserção de e'' em T, e do ciclo C'' formado pela inserção de e' em T. Mas e'' pesa mais que e', porque os dois conectam S e $V \setminus S$, mas e' é aresta mais leve desse tipo. Com e'' não fez parte de T ele pesa mais que as arestas em C' e como e' não fez parte de T ele pesa mais que as arestas em C''. Logo e'' é a aresta mais pesada em C (ver também figura 1(b)).

Exercício 3

a) A sequência 1234 com atrasos $L_1 = 2$, $L_2 = 4$, $L_3 1$, e $L_4 = 6$ e atraso máximo 6 é ótima.

b) Considere duas tarefas i e j subsequentes em alguma sequência. O atraso máximo é o atraso do i ou o atraso do j ou o atraso de alguma outra tarefa t:

$$L_{\max} = \max\{L_i, L_j, L_t\}.$$

   Agora considera o que acontece se trocamos a ordem de i e j. O atraso de i e do j pode ser diferente, mas o atraso das outras tarefas é o mesmo. Logo

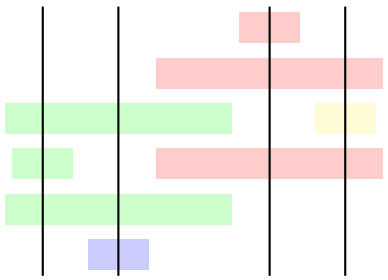
$$L'_{\max} = \max\{L'_i, L'_j, L_t\}.$$

Com a tarefa j termina mais cedo agora sabemos que o seu atraso diminui: $L'_j \leq L_j$. Qual o novo atraso de i? Seja t o tempo de início de i antes da troca: Logo

$$L_i = \max\{t + p_i - d_i, 0\}$$
$$L'_i = \max\{t + p_i + p_j - d_i, 0\}$$

Como $L_j = \max\{t + p_i + p_j - d_j, 0\}$ antes da troca, temos também $L'_i \leq L_j$ caso $d_i \geq d_j$. Logo, caso um processo com um *deadline* maior precede um processo com deadline menor, podemos trocar os dois sem aumentar o *deadline*. Em resumo (por indução), uma sequência ordenada por *deadlines* não-decrescente tem que ser ótima.

Exercício 4
Podemos ordenar os intervalos pelo tempo de término e processar nessa ordem. Para o próximo intervalo nessa ordem, escolhe o seu término como ponto de pica e remove este intervalo e todos outros intervalos picados. Isso gera a menor número de pontos possíveis. Prova: Provaremos por indução que para cada ponto t que o algoritmo guloso escolha, ele usava menor número de pontos para os intervalos que começaram antes de p. A afirmação então é consequência do fato que na última escolha, esse conjunto contém todos intervalos. Base da indução: Na primeira escolha, a afirmação com certeza é correta, já que precisamos pelo menos um ponto para picar o intervalo atual. Passo da indução: Considere o $i + 1$-ésimo ponto escolhido. Pela hipótese da indução foram necessárias pelo menos i pontos para picar os intervalos que começaram até o ponto anterior t_i. Mas o intervalo atual começou depois de t_i senão teria sido excluído no passo anterior. Logo precisamos pelo menos $i + 1$ pontos para picar todos intervalos que começaram antes de t_{i+1}.
Segue um contra-exemplo para a estratégia de escolher gulosamente o ponto que pica o maior número de intervalos.



Exercício 5
Caso multiplicamos uma linha ou coluna com soma negativa por -1, a soma total dos coeficientes da matriz aumenta. Como a soma total é limitado pela soma total de valores absolutos, um processe que repetidamente executa essa operação tem que terminar. Como ele só termina caso não existe mais linha ou coluna com soma negativa, provamos que isso é um algoritmo que resolve o problema.