Instituto Superior Técnico

Análise e Síntese de Algoritmos

Ano Lectivo 2021/2022 1º Exame

RESOLUÇÃO

Cotação: 4 + 4 + 4 + 4 + 4 = 20 val.

A) Dado um conjunto de inteiros $V = \{v_1, ..., v_n\}$, pretende determinar-se se é possível dividir V em três subconjuntos disjuntos de modo a que a somas dos seus elementos coincidam. Formalmente, pretende determinar-se se existem três conjuntos V_1, V_2 e V_3 tais que: **(1)** $V = V_1 \cup V_2 \cup V_3$; **(2)** $V_1 \cap V_2 = V_2 \cap V_3 = V_1 \cap V_3 = \emptyset$; and **(3)** $\sum_{v \in V_1} v = \sum_{v \in V_2} v = \sum_{v \in V_3} v$. Por exemplo, o conjunto $\{1, 2, 4, 5, 6\}$ pode ser dividido nos subconjuntos: $\{1, 5\}$, $\{2, 4\}$ e $\{6\}$, todos com soma 6.
Nota: Admite-se, para simplificar a formulação do problema, que os elementos do conjunto dado como input estão indexados, sendo que se denota por v_i o i-ésimo elemento do conjunto.

1. Seja $B^{(m)}(k_1, k_2)$ o booleano que indica se existem dois conjuntos disjuntos V_1' e V_2' contidos no conjunto $\{v_1, ..., v_m\}$, com $m \leq n$, tais que: $\sum_{v \in V_1'} v = k_1$ e $\sum_{v \in V_2'} = k_2$. Por exemplo, dado o conjunto $\{1, 2, 4, 5, 6\}$, temos que:

   - $B^{(4)}(6, 6) =$ **true**, basta escolher $V_1' = \{1, 5\}$ e $V_2' = \{2, 4\}$;
   - $B^{(4)}(3, 5) =$ **true**, basta escolher $V_1' = \{1, 2\}$ e $V_2' = \{5\}$;
   - $B^{(4)}(2, 3) =$ **false**.

   Defina $B^{(m)}(k_1, k_2)$ recursivamente completando os campos em baixo:

$$B^{(m)}(k_1, k_2) = \begin{cases} \textbf{false} & \text{se } ((k_1 \neq 0) \vee (k_2 \neq 0)) \wedge m = 0 \\ \boxed{} & \text{se } k_1 = 0 \wedge k_2 = 0 \\ B^{(m-1)}(k_1 - v_m, k_2) \vee \boxed{} \vee \boxed{} & \text{c.c.} \end{cases}$$

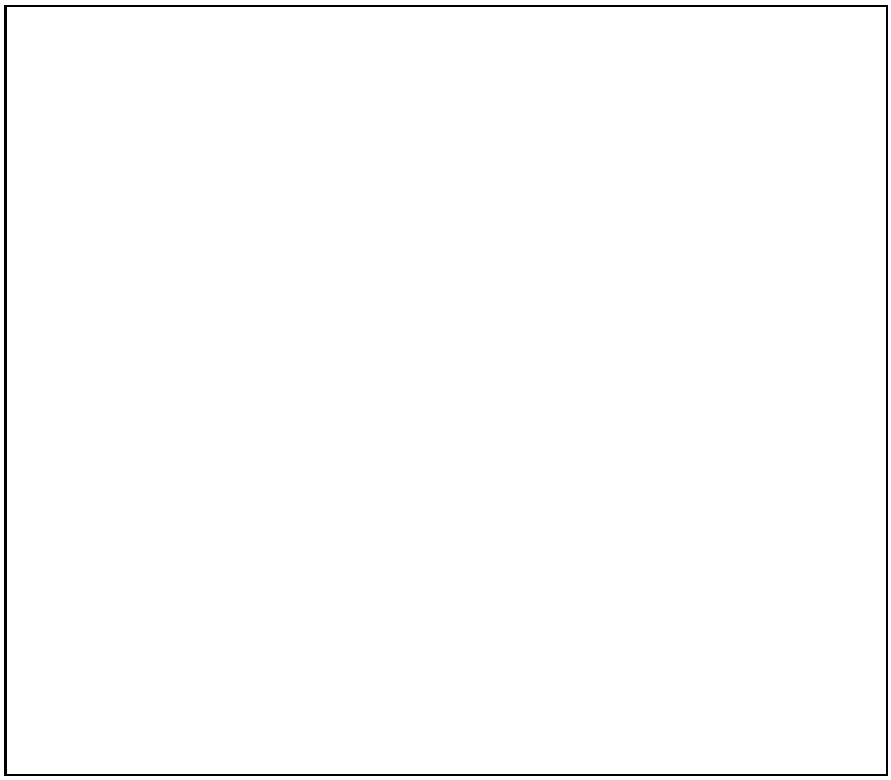
   Admite-se para facilitar a formulação que $B^{(m)}(k_1, k_2) =$ **false** quando $k_1 < 0$ ou $k_2 < 0$.

2. Complete o template de código em baixo que, dados dois inteiros k_1 e k_2, calcula a matriz $B^{(0)}(k_1, k_2)$.

```
InitMatrix(k1, k2)
  let B[0..k1, 0..k2] be a new matrix of size (k1 + 1) × (k2 + 1)
  [                                                              ]
  return B
```

3. Complete o template de código em baixo que, dado um conjunto de inteiros $V = \{v_1, ..., v_n\}$ e dois inteiros k_1 e k_2, calcula o valor booleano $B^{(n)}(k_1, k_2)$.

```
DoublePartition(v[1..n], k1, k2)
  let B^(0) := InitMatrix(k1, k2)
  for l = 1 to n do
    let B^(l) be a new matrix of size (k1 + 1) × (k2 + 1)
```



```
  endfor
  return B^(n)[k1, k2]
```

4. Determine a complexidade assimptótica do algoritmo proposto na alínea anterior.

5. Explique por palavras como obter a solução para o problema original a partir do algoritmo proposto na alínea 3.

Solução:

1.

$$B^{(m)}(k_1, k_2) = \begin{cases} \textbf{false} & \textbf{se } ((k_1 \neq 0) \vee (k_2 \neq 0)) \wedge m = 0 \\ \textbf{true} & \textbf{se } k_1 = 0 \wedge k_2 = 0 \\ B^{(m-1)}(k_1 - v_m, k_2) \vee B^{(m-1)}(k_1, k_2 - v_m) \vee B^{(m-1)}(k_1, k_2) & \textbf{c.c.} \end{cases}$$

2.

```
InitMatrix(k1, k2)
  let B[0..k1, 0..k2]  be a new matrix of size (k1 + 1) × (k2 + 1)
  for i = 0 to k1 do
    for j = 0 to k2 do
      B[i, j] := false
    endfor
  endfor
  B[0, 0] := true
  return B
```

3.

```
DoublePartition(v[1..n], k1, k2)
  let B^(0) := InitMatrix(k1, k2)
  for l = 1 to n do
    let B^(l) be a new matrix of size (k1 + 1) × (k2 + 1)
    for i = 0 to k1 do
      for j = 0 to k2 do
        if(i == 0 ∧ j == 0){
          B^(l)[i, j] := true
        } else{
          b := B^(l−1)[i, j]
          if(i ≥ v[l]){
            b := b ∨ B^(l−1)[i − v[l], j]
          }
          if(j ≥ v[l]){
            b := b ∨ B^(l−1)[i, j − v[l]]
          }
          B^(l)[i, j] := b
        }
      endfor
    endfor
  endfor
  return B^(n)[k1, k2]
```

4. Complexidade: $O(n.k_1.k_2)$. O algoritmo tem de preencher $n + 1$ matrizes cada uma com dimensão $(k_1 + 1) \times (k_2 + 1)$.

5. Dado um conjunto de inteiros $V = \{v_1, ..., v_n\}$, começamos por calcular a soma dos elementos do conjunto: $\bar{v} = \sum_{v \in V} v$. Depois, calculamos a divisão inteira de $\bar{v}$ por 3. Se 3 não divide $\bar{v}$, o algoritmo retorna **false**. Caso contrário, seja k o valor do quociente obtido, i.e. $3 * k = \bar{v}$, a resposta ao problema proposto é dada por: `DoublePartition`$(v[1..n], k, k)$.

B) O gestor de pessoal do Hospital Central de Caracolândia foi encarregue de fazer a atribuição de cirurgiões a blocos operatórios para o próximo mês tendo em conta as restrições indicadas em baixo:

- O hospital dispõe de k cirurgiões $\{C_1, ..., C_k\}$.
- O calendário mensal hospitalar é constituído por n slots para cirurgias $\{S_1, ..., S_n\}$.
- O hospital dispõe de m blocos operatórios $\{B_1, ..., B_m\}$, sendo que cada bloco operatório B_i está apenas disponível nos slots contidos no conjunto de slots $\mathbf{BSlots}(B_i)$.
- Cada cirurgião C_i pode apenas efectuar cirurgias nos blocos operatórios contidos no conjunto $\mathbf{CBlocks}(C_i)$.
- Nenhum cirurgião pode efectuar mais de **max** cirurgias por mês.
- Cada cirurgia envolve um único cirurgião.

O objectivo do gestor hospitalar é maximizar o número de cirurgias efectuadas, respeitando as restrições do problema, admitindo que: $m < k < n$.

1. Modele o problema descrito em cima como um problema de fluxo máximo. A resposta deve incluir o procedimento utilizado para determinar o conjunto de tuplos da forma (C_i, S_j, B_l), indicando que o cirugião C_i vai efectuar uma cirurgia no bloco B_l no slot S_j.
2. Indique o algoritmo que utilizaria para a calcular o fluxo máximo, bem como a respectiva complexidade assimptótica medida em função dos parâmetros do problema: número de cirurgiões k, número de slots n e número de blocos operatórios m. De entre os algoritmos de fluxo estudados nas aulas deve escolher aquele que garanta a complexidade assimptótica mais baixa para o problema em questão.
   Nota: A resposta deverá necessariamente incluir as expressões que definem os limites superiores assimptóticos para o número de vértices e de arcos da rede de fluxo proposta ($|V|$ e $|E|$, respectivamente) em função dos parâmetros do problema, bem como um upper-bound para o valor do fluxo máximo.

Solução:

1. *Construção da rede de fluxo* $G = (V, E, w, s, t)$. Na construção da rede de fluxo consideramos um vértice por cirurgião, um vértice por cada par bloco-slot compatíveis e dois vértices adicionais s e t, respectivamente a fonte e o sumidouro. Formalmente:

   - $$\begin{array}{ll} V = \{C_i \mid 1 \leq i \leq k\} & \text{Um vértice por cirurgião} \\ \quad \cup \{CS_{ij} \mid 1 \leq i \leq k \wedge 1 \leq j \leq n \wedge\} & \\ \quad \cup \{SB_{ij} \mid 1 \leq i \leq n \wedge 1 \leq j \leq m \wedge S_i \in \mathbf{BSlots}(B_j)\} & \text{Um v. por par bloco-slot compatíveis} \\ \quad \cup \{s, t\} & \text{Fonte e sumidouro} \end{array}$$

   - $$\begin{array}{ll} E = \{(s, C_i, \mathbf{max}) \mid_{i=1}^{k}\} & \text{Cada cirurgião pode fazer } \mathbf{max} \text{ cirurgias} \\ \quad \cup \{(C_i, CS_{ij}, 1) \mid_{i=1}^{k} \mid_{j=1}^{n}\} & \text{cirurgiões e slots} \\ \quad \cup \{(CS_{ij}, SB_{jl}, 1) \mid B_l \in \mathbf{CBlocks}(C_i) \wedge S_j \in \mathbf{BSlots}(B_l)\} & \text{Blocos e slots} \\ \quad \cup \{(SB_{ij}, t, 1) \mid_{i=1}^{n} \mid_{j=1}^{m}\} & \text{Blocos e slots} \end{array}$$

   A resposta deve incluir o tuplo (C_i, S_j, B_l) sse $f^*(CS_{ij}, SB_{jl}) = 1$.

2. *Complexidade:*

   - $|V| = k + k.n + m.n + 2 = O(k.n)$
   - $|E| \leq k + k.n + k.n.m + m.n = O(k.m.n)$
   - $|f^*| \leq m.n = O(m.n)$
   - Edmonds Karp (upper bound de FF): $O(|f^*|.E) = O(k.n.k.n.m) = O(k^2.n^2.m)$
   - Edmonds Karp (upper bound EK): $O(E^2.V) = O(k^3.m^2.n^3)$
   - Relabel-To-Front: $O(k^3.n^3)$

   O limite mais apertado é obtido pelo upper bound do método de FF, pelo que podemos utilizar qualquer implementação do método Ford-Fulkerson.

C) Uma família com n membros $\{M_1, ..., M_n\}$ prepara-se para cozinhar a maior pizza de sempre. Para tal têm de escolher os ingredientes a incluir na pizza de entre k ingredientes disponíveis $\{I_1, ..., I_k\}$. Cada familiar M_i deve indicar os ingredientes que não deseja incluir na pizza e os ingredientes que deseja incluir. Assim sendo, associamos a cada familiar M_i um par com dois conjuntos de ingredientes, (C_i, C_i'), onde C_i contém os ingredientes a incluir e C_i' os ingredientes a excluir.

Tratando-se de uma família pouco conflituosa para que um familiar se considere satisfeito basta que uma das suas escolhas seja atendida: um dos seus ingredientes preferidos seja incluído ou um dos preteridos não o seja. Por exemplo, suponha que o pai quer fiambre e queijo e não quer ananás; para que a pizza escolhida satisfaça o pai, basta que contenha fiambre ou queijo ou não contenha ananás.

O problema da escolha de ingredientes para pizza, **PizzaIngredients**, consiste em determinar se existe um conjunto de ingredientes que satisfaça todos os membros da família e é modelado formalmente através do seguinte problema de decisão:

$$\textbf{PizzaIngredients} = \{\langle \mathcal{I} \rangle \mid \text{existe uma escolha de ingredientes compatível com } \mathcal{I}\}$$

Onde $\mathcal{I}$ denota o conjunto de pares que representam as escolhas da família.

1. Mostre que o problema **PizzaIngredients** está em **NP**.
2. Mostre que o problema da escolha dos ingredientes é **NP**-difícil por redução a partir do problema **3-CNFSAT** estudado nas aulas. Não é necessário provar formalmente a equivalência entre os dois problemas; é suficiente indicar a redução e a respectiva complexidade.

Solução:

1.
   - *Certificado:* o conjunto X de ingredientes a incluir.
   - *Tamanho do Certificado:* $|X| \in O(k)$
   - *Algoritmo de verificação:* Verificar se X é compatível com cada elemento do conjunto $\mathcal{I}$. Começamos por calcular o conjunto $\bar{X} = \{I_1, ..., I_k\} \backslash X$. Para cada par (C_i, C_i'), verificamos se: $X \cap C_i \neq \emptyset$ ou $\bar{X} \cap C_i' \neq \emptyset$.
   - *Complexidade do algoritmo de verificação:* A intersecção de conjuntos faz-se em tempo linear, pelo que a verificação de cada par para custa: $O(k)$. Assim, a verificação de todos os n pares faz-se em tempo $O(k.n)$.
2. Há que mostrar que **3-CNFSAT** $\leq_P$ **PizzaIngredients**.
   - *Redução:* Cada cláusula corresponde a um membro da família e cada variável a um ingrediente. As variáveis negadas na cláusula correspondem aos ingredientes a excluir e as variáveis não negadas aos ingredientes a incluir. Por exemplo, a cláusula $(x_1 \vee \neg x_2 \vee x_3)$ é mapeada no par $(\{x_1, x_3\}, \{x_2\})$. A redução tem complexidade: $O(n)$, onde n é o número de cláusulas.

D) Considere a aplicação do algoritmo de Prim a grafos com arcos de peso inteiro não negativo e limitado superiormente por uma dada constante X de valor pequeno. Neste caso, pode utilizar-se uma implementação alternativa da fila de prioridade mínima por forma a melhorar a complexidade assimptótica do algoritmo.

1. Admitindo que as chaves a inserir na fila têm prioridade limitada por uma dada constante Z de valor pequeno, proponha uma implementação de uma fila de prioridade mínima com as seguintes operações e complexidades associadas:

   - `ExtractMin(Queue q)` $\in O(Z)$: extrai o elemento de prioridade mínima;
   - `DecreaseKey(Queue q, Elem elem, int pri)` $\in O(1)$: diminui a prioridade associada ao elemento `elem` na fila `q` para o valor `pri`;
   - `MakeQueue(Elem[] elems)` $\in O(n)$: cria uma fila de prioridade mínima com n elementos `elems` respeitando as prioridades que lhes estão associadas.

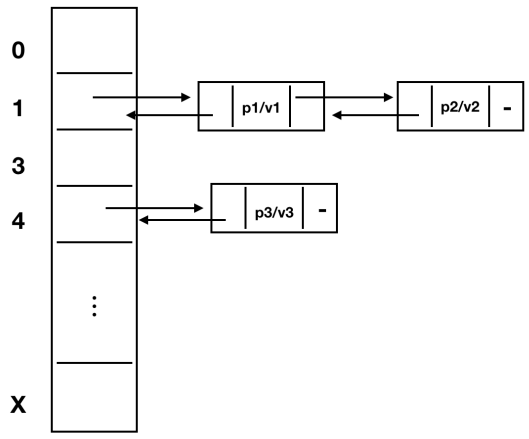
   Não é necessário apresentar o pseudo-código. Basta fazer um diagrama que ilustre a estrutura da fila de prioridade e explicar por palavras a implementação das três operações. Pode admitir-se que cada elemento `elem` está associado a uma estrutura com os campos `key` e `pri`, que guardam respectivamente a sua chave e prioridade.

2. Admitindo que o peso dos arcos do grafo está limitado por uma dada constante X, qual o número máximo de prioridades distintas a manter na fila de prioridade?

3. Admitindo que as complexidades das operações associadas à fila de prioridade mínima são aquelas as dadas na alínea **1.**, indique a complexidade assimptótica do algoritmo de Prim. Deve justificar a resposta.

Solução:

1. Mantemos um vector com uma posição por prioridade e associamos a cada prioridade uma lista duplamente ligada com elementos com essa prioridade. Adicionalmente, guardamos a prioridade mínima associada a um elemento da fila. O diagrama que ilustra a implementação é dado em baixo:



   Operações:

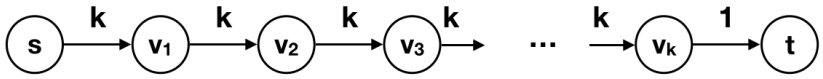
   - `ExtractMin(Queue q)` $\in O(Z)$: remove o primeiro elemento da lista duplamente ligada associada à prioridade mínima.
   - `DecreaseKey(Queue q, Elem elem, int pri)` $\in O(1)$: remove o elemento *elem* da lista duplamente ligada que o contém para depois o inserir na lista duplamente ligada associada à nova prioridade.

- `MakeQueue(Elem[] elems)` $\in O(n)$: todos os elementos são inseridos nas listas duplamente ligadas correspondentes às suas prioridades.

2. A fila de prioridade tem que manter no máximo $X+2$ prioridades distintas porque há que contar com a prioridade 0 e com a prioridade ∞.

3. O algoritmo de *Prim* executa $|V|$ iterações do ciclo principal que remove os vértices da fila de prioridade e $|E|$ iterações do ciclo interior que percorres as adjacências de cada vértice. Todas as operações realizadas em cada iteração dos dois ciclos (`DecreaseKey` e `ExtractMin`) se fazem em tempo constante, pelo que a complexidade do algoritmo é agora $O(V + E)$.

E) Considere a aplicação do algoritmo Push-Relabel à rede de fluxo que se apresenta em baixo:



Durante a resolução do exercício admita que k é um inteiro ímpar superior a 1.

1. Determine o número exacto de operações de push e relabel a ser efectuadas durante a aplicação do algoritmo em função de k, bem como o respectivo limite assimptótico superior.

2. Considere a seguinte heurística, designada por *heurística de intervalo*, a aplicar durante a execução do algoritmo:

   > Se existir uma altura $0 < k < |V|$ tal que nenhum vértice do grafo tem altura k, então é atribuída a todos os vértices $v \in V \backslash \{s, t\}$ com altura superior a k a altura: $\mathbf{max}(v.h, |V| + 1)$.

   Determine o número exacto de operações de push e relabel a ser efectuadas durante a aplicação do algoritmo de Push-Relabel ao grafo da figura em função de k considerando a aplicação da heurística de intervalo, bem como o respectivo limite assimptótico superior.

3. Recorde o invariante de *função de altura* usado para estabelecer a correcção do algoritmo de Push-Relabel. Mostre que a aplicação da heurística de intervalo ao grafo apresentado não viola este invariante.

Solução:

1. Para contar o número de pushes e relabels, é útil estabelecer primeiro o número de travessias de ida-e-volta do grafo. As alturas dos vértices v_1 e v_k variam da seguinte maneira:

Travessia	$h(v_1)$	$h(v_k)$
0	0	0
1	1	2
2	3	4
$\vdots$	$\vdots$	$\vdots$
i	$2i-1$	$2i$

   Paramos de empurrar fluxo da esquerda para a direita quando $h(v_1) = h(s) = k+2$. Resolvendo para i obtemos: $2i - 1 = k + 2 \Leftrightarrow i = \frac{k+3}{2}$. Contamos o número de pushes e relabels por travessia:

Travessia	# Pushes	# Relabels
0	$2.(k-1)+2$	$2.(k-1)+1$
1	$2.(k-1)$	$2.(k-1)$
2	$2.(k-1)$	$2.(k-1)$
$\vdots$	$\vdots$	$\vdots$
$\frac{k+3}{2}$	$2.(k-1)+1$	$2.(k-1)+1$
Total:	$(k+3).(k-1)+3$	$(k-1).(k+3)+2$

   Na primeira travessia temos 2 pushes adicionais (de s para v_1 e de v_k para t) e 1 relabel adicional (de $h(v_k)$ para 1). Na última travessia temos 1 push adicional (de v_1 para s) e 1 relabel adicional (de $h(v_1)$ para $k+3$).

2. No final da primeira ida-e-volta todos os vértices intermédios têm altura 2 excepto o vértice v_1 que tem altura 1. Quando a altura de v_1 é subida para 3, estamos nas condições da heurística de intervalo e a altura de todos os vértices é subida para $k+3$. Contamos apenas os pushes e relabels da primeira ida-e-volta, incluindo os relabels da aplicação da heurística:

   - *Número de pushes:* $2(k-1)+3$
   - *Número de relabels:* $2(k-1)+2+k = 3k$

3. O invariante de altura estabelece que $(u,v) \in E_f$ então $h(u) \leq h(v)+1$. Para mostrarmos que aplicação da heurística não viola o invariante de altura consideramos a rede residual no momento da aplicação da heurística e as novas alturas obtidas. A rede residual é composta por 4 tipos de arcos:

   - (t, v_k): $h(t) = 0 \leq h(v_k)+1 = k+3+1 = k+4$
   - (v_i, v_{i+1}): $h(v_i) = k+3 \leq h(v_{i+1})+1 = k+4$
   - (v_{i+1}, v_i): $h(v_{i+1}) = k+3 \leq h(v_i)+1 = k+4$
   - (v_1, s): $h(v_1) = k+3 \leq h(s)+1 = k+3$