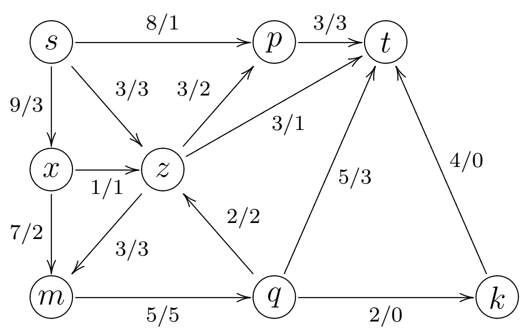
Departamento de Ciência de Computadores FCUP

Desenho e Análise de Algoritmos (CC2001) 2017/18

Exame (29.01.2018) *duração: 3h*

Resolução de questões selecionadas

1. Considere a rede de fluxo seguinte, onde c/f são pares capacidade/fluxo, e s e t são a origem e destino.

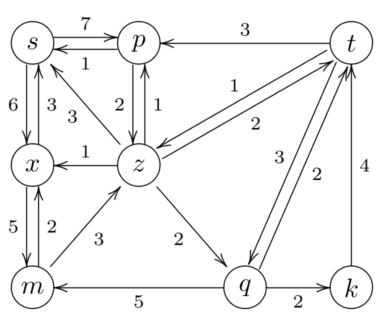


a) [~~0.7~~ 1.0] Indique os valores de:

$f(q,m)$	-5	$f(p,z)$	-2	$f(z,p)$	2
$\lvert f\rvert$	7	$c(q,m)$	0	$c(m,q)$	5
$c_f(q,m)$	5	$c_f(m,q)$	0	$c_f(z,t)$	2
$c_f(p,s)$	1	$c_f(s,z)$	0	$c_f(k,t)$	4

b) [~~1.5~~ 2.0] Partindo do fluxo f, aplique o algoritmo de Edmonds-Karp para obter um fluxo máximo (desenhe a rede residual em cada iteração, represente o fluxo final na rede, e explique sucintamente os passos).

1. Rede residual G_f para f:

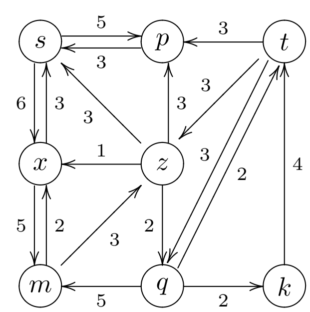


Caminho de s para t em G_f obtido por pesquisa em largura (BFS):

$$\gamma = (s, p, z, t)$$

com capacidade $\min(7, 2, 2) = 2$. Usamos γ para acrescentar 2 unidades ao fluxo.

2. Nova rede G_f:



Caminho de s para t (por BFS):

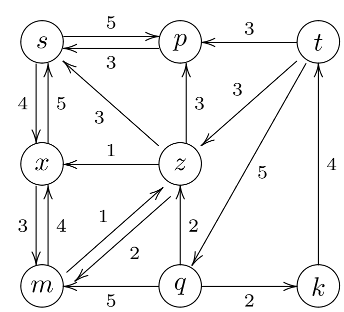
$$\gamma = (s, x, m, z, q, t).$$

Capacidade de γ é

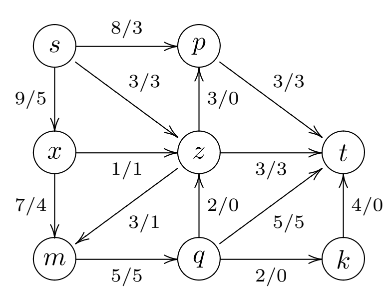
$$\min(6, 5, 3, 2, 2) = 2.$$

Usamos γ para acrescentar 2 unidades ao fluxo.

3. Nova rede G_f:



Não existe caminho de s para t.
O fluxo não pode aumentar.
Logo, $|f^\star| = 7 + 2 + 2 = 11$.
Representação do fluxo final ($f^\star$):



c) [0.6] Complete as frases: *A capacidade do corte* $(\{s,q,t\},\{p,x,z,m,k\})$ *é* $\boxed{(8{+}3{+}9){+}(2{+}2) = 24}$.

$\boxed{(\,\{s,p,x,m,z\},\ \{q,k,t\}\,)}$ *é um corte {S,T} com capacidade mínima, a qual é* $\boxed{11}$.

Para provar o **Teorema de Ford-Fulkerson**, mostrou-se que $S = \{$nós acessíveis de s na rede residual final$\}$ e $T = V \setminus S$ definem um corte $\{S,T\}$ com capacidade mínima.

d) [1.2] Escreva em pseudocódigo uma **função** que determine **um caminho para aumento** numa dada rede residual G_f e o **incremento** de $|f|$. Assuma que $V = \{1, 2, \ldots, |V|\}$. Qual é a sua complexidade?

O caminho para aumento do fluxo é um caminho de s para t no grafo G_f e pode ser determinado por pesquisa em largura (como no algoritmo de Edmonds-Karp) ou em profundidade.

```
CaminhoAumento_BFS(s, t, G_f, pai)
    Para v ← 1 até |G_f.V| fazer
        pai[v] ← 0;  visitado[v] ← false;
    Q ← Mk_Empty_Queue(|G_f.V|);
    EnQueue(s, Q); visitado[s] ← true; cap[s] ← ∞;
    Enquanto (NotEmptyQueue(Q)) fazer
        v ← DeQueue(Q);
        Se v = t então retorna cap[t];
        Para w ∈ G_f.Adjs[v] fazer
            Se visitado[w] = false então
                visitado[w] ← true; pai[w] ← v;
                cap[w] ← min(cap[v], VALOR_ARCO(v, w, G_f));
                EnQueue(w, Q);
    retorna 0;   /* zero indica não existência de caminho */
```

Assume-se que cap é um vetor de inteiros e que VALOR_ARCO(v, w, G_f) é a capacidade residual do arco (v, w). A função tem complexidade $O(|V| + |E_f|)$, sendo E_f o conjunto de ramos de G_f.

Em alternativa, podia começar por determinar um caminho de s para t e, a seguir, a sua capacidade.

2. [~~1.0~~ 1.5] Considere o problema de formar uma certa quantia q com um número **mínimo** de moedas de valores 1, 2, 5, 10, 20, 50, 100, e 200, estando q e esses valores na mesma unidade monetária. Indique a **estratégia *greedy*** que obtém a solução ótima se se dispuser de um número ilimitado de moedas de cada tipo e prove que é incorreta se **for limitado**. Indique **todos** os erros possíveis e instâncias nessas condições.

Estratégia greedy: usar as moedas de valor mais elevado não superior a q o número máximo de vezes possível e aplicar a mesma estratégia para a quantia que sobrar.

Esta estratégia falha se o número de moedas for limitado porque:

- pode não permitir formar a quantia q, embora fosse possível se se usasse outra estratégia (por exemplo, se $q = 6$ e tiver duas moedas de valor 5, três de valor 2 e nenhuma de valor 1; se aplicar a estratégia greedy, não consegue formar q, embora pudesse formar com as moedas de valor 2).
- pode requerer mais moedas para formar q do que seria necessário (por exemplo, se $q = 60$ e tiver pelo menos uma moeda de 50, três de 20 e cem moedas de 1, e nenhuma dos restantes tipos, a estratégia greedy usaria onze moedas – uma de 50 e dez de 1 – mas bastava usar três de 20).

3. [2.0] Usando a definição matemática das ordens de grandeza e das classes indicadas, justifique a veracidade ou falsidade de cada uma das afirmações seguintes.

a) $3n^2 + 100 \in \Omega(6n^2 + 5)$.

Afirmação verdadeira, pois $\exists_{c \in \mathbb{R}^+} \exists_{n_0 \in \mathbb{N}} \forall_{n \geq n_0}\ 3n^2 + 100 \geq c(6n^2 + 5)$. Por exemplo, $c = \frac{1}{2}$ e $n_0 = 1$.

N.º | Nome

b) $100n + 3n\log_2 n \notin \Theta(n\log_2 n)$

Afirmação falsa, pois $\exists_{c_1\in\mathbb{R}^+}\exists_{c_2\in\mathbb{R}^+}\exists_{n_0\in\mathbb{N}}\forall_{n\geq n_0}\ \ c_1(n\log_2 n) \leq 100n + 3n\log_2 n \leq c_2(n\log_2 n)$. Por exemplo, $c_1 = 1$, $c_2 = 103$ e $n_0 = 2$.

4. Considere o algoritmo de Dijkstra, suportado por uma *heap binária de mínimo* Q, para determinação de caminhos mínimos com origem num nó s num grafo dirigido $G = (V, E, d)$, com $d(e) \in \mathbb{Z}^+$, para $e \in E$.

a) [0.3+0.7] O que retorna a operação EXTRACTMIN(Q)? Como é efetuada e de que modo afeta Q na implementação de Q apresentada nas aulas (recorde que se mantém dois *arrays* $Q.a$ e $Q.pos\_a$).

EXTRACTMIN(Q) retorna o **identificador do nó** v que tem $dist[v]$ mínimo (e que está ainda na *heap*).

Cada elemento de $Q.a$ é um par que corresponde a $(v, dist[v])$, sendo $dist[v]$ a chave. EXTRACTMIN(Q) retira o elemento $Q.a[1]$, que tem chave mínima, substituindo-o pelo elemento $Q.a[size]$ (sendo $size$ o número de elementos ainda na heap, o qual é decerementado nesta operação). A seguir, aplica a operação HEAPIFY(1) para repor a condição de *heap-min*, isto é, para garantir que a chave de cada nó da heap não excede as chaves dos seus filhos, se existirem. Se exceder, o nó troca de posição com o filho que tiver chave menor e a operação HEAPIFY prossegue a partir desse filho.

Na implementação, $Q.pos\_a[v]$ indica a posição dos dados correspondentes ao nó v do grafo, isto é, a posição do par $(v, dist[v])$ na heap $Q.a$. Se a posição for alterada, o valor $Q.pos\_a[v]$ é alterado consistentemente (garantindo que, enquanto v está na heap, $Q.pos\_a[v] = i$ sse $Q.a[i] = (v, dist[v])$). Esse *array* é útil na operação DECREASEKEY($Q, v, dist[v]$) para localizar $(v, dist[v])$ na heap em $O(1)$.

b) [1.0] Indique a complexidade de EXTRACTMIN(Q) $\boxed{O(\log_2(size))}$, DECREASEKEY($Q, v, dist[v]$) $\boxed{O(\log_2(size))}$, e do algoritmo de Dijkstra $\boxed{O((|V| + |E|)\log_2(|V|))}$.

Aqui, $size$ é o número de elementos ainda na heap. Podia ser indicado $O(\log_2(|V|))$ em vez de $O(\log_2(size))$.

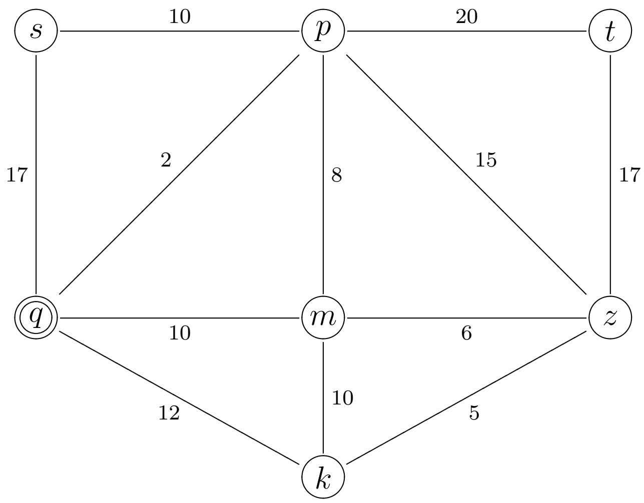
5. [~~0.4~~ 0.0(*)] Uma árvore de pesquisa *red-black* não é uma árvore equilibrada. Que propriedade garante que a operação de procura de um dado valor seja realizada em $O(\log_2 n)$, sendo n o número de valores na árvore?

/* Em alternativa, resolva questão 10. */

Uma árvore “red-black” com n nós internos tem altura menor ou igual a $2\log_2(n+1)$. A operação de pesquisa tem complexidade $O(h)$, sendo h a altura da árvore, e $O(\log_2 n) = O(\log_2(n+1))$.

(Continua, v.p.f.)

6. [2.0] Aplique o algoritmo de Prim para obter uma árvore geradora $\mathcal{T}$ de peso **mínimo** do grafo indicado, com raiz q. Anote os nós com pares $(dist, pai)$, como se definiu nas aulas, de modo a poder reconstruir **os passos intermédios** dessa aplicação. Na caixa à direita, indique os nós em $\mathcal{T}$ após cada iteração.



Resolução omitida (ver a forma das anotações nos apontamentos das aulas ou na correção do 1ºteste).

7. Considere o algoritmo de Kosaraju-Sharir para determinação das componentes fortemente conexas de um grafo dirigido $G = (V, E)$. Pretendemos obter uma lista de listas de nós que definem cada componente.

a) [~~1.5~~ 2.0] Descreva os passos principais, as suas complexidades temporais e as estruturas de dados que usam.

1. Aplicar pesquisa em profundidade (DFS) para visitar o grafo G e colocar numa pilha S os identificadores dos nós por ordem crescente de tempo de finalização (o nó v é colocado em S à saída de $DFS\_Visit(v)$). Este passo usa ainda um *array* de booleanos para assinalar os nós já visitados. Tem complexidade $\Theta(|V| + |E|)$, para G representado por listas de adjacências.

2. Construir o grafo G^T transposto de G. Tem complexidade $\Theta(|V| + |E|)$.

3. Visitar G^T, usando a pilha S para definir a ordem pela qual efetua a pesquisa: enquanto a pilha não ficar vazia, retira o nó v do topo e, se v ainda não estiver visitado, efetua pesquisa em profundidade a partir de v, acrescentando a lista de nós que visita nessa pesquisa à lista das componentes (lista de listas). Tem complexidade $\Theta(|V| + |E|)$.

b) [0.4] Que propriedades da pesquisa e do grafo de componentes são determinantes para a correção?

A pesquisa em profundidade num DAG determina uma ordenação topológica dos nós do DAG.

O grafo das componentes fortemente conexas de G é um DAG. As componentes fortemente conexas de G e G^T são iguais. Uma ordenação topológica do DAG das componentes de G^T corresponde a uma ordenação topológica inversa do DAG as componentes de G.

A pesquisa em DFS de G no passo 1. coloca os nós na pilha S por uma ordem que induz uma ordenação topológica inversa do DAG das componentes de G^T. Se uma componente $\mathcal{C}$ permite aceder a uma componente $\mathcal{C}'$ em G (e, consequentemente, no DAG de componentes de G), com $\mathcal{C}' \neq \mathcal{C}$, então os nós de $\mathcal{C}'$ ficaram abaixo dos de $\mathcal{C}$ na stack S. Assim, ao visitar o DAG de G^T por ordem inversa, garante que os nós que encontra por visitar na pesquisa a partir de um dado v (no passo 2) pertencem à sua componente conexa (dado que as componentes a que conseguiria aceder já estão visitadas).

(Continua, v.p.f.)

N.º | | Nome | |

8. Considere a função ANALISAROTA(s, t, m, L) para verificar se uma rota a dar por um utilizador passa em s e t e tem lugares suficientes entre s e t para um grupo de m elementos, sendo L uma matriz e $L[v, w]$ o número de lugares disponíveis no troço (v, w) (que será -1 se não existir esse troço). A rota é dada pela sequência de nós por onde passa, os quais são todos distintos. O utilizador começa por dar o número de nós da rota e a seguir indicará os nós. Assuma que $s \neq t$ e $m \geq 1$.

ANALISAROTA(s, t, m, L)

1. $d \leftarrow m$;
2. $ok \leftarrow$ `false`;
3. ler(n); ler(v); $k \leftarrow 1$;
4. Se $(v = s)$ então $ok \leftarrow$ `true`;
5. Enquanto $(v \neq t \wedge d = m \wedge k < n)$ fazer
6.   ler(w); $k \leftarrow k + 1$;
7.   Se $(w = s)$ então $ok \leftarrow$ `true`;
8.   senão
9.   se $(ok =$ `true` $\wedge\, d > L[v, w])$ então
10. $d \leftarrow L[v, w]$;
11. $v \leftarrow w$;
12. Se $(v = t \wedge ok =$ `true` $\wedge\, d = m)$ então
13.   retorna `true`;
14. retorna `false`;

a) [~~1.2~~ 1.5] Qual é a complexidade no pior caso? $\boxed{\Theta(n)}$ E, no melhor caso? $\boxed{\Theta(1)}$ Identifique-os e explique.

A complexidade do algoritmo é dominada pelo ciclo "Enquanto", pois os blocos 1–4 e 12–14 têm complexidade $\Theta(1)$.

O **pior caso** acontece quando se tem de processar os n nós da rota. Ocorre, por exemplo, se a rota não passar nem em s nem em t.

O **melhor caso** ocorre, por exemplo, quando os dois primeiros nós são s e t, pois o bloco $6 - 11$ só será executado uma vez. Esse bloco tem complexidade $\Theta(1)$, assim como cada teste da condição de ciclo e transferência de controlo.

b) [0.7] Assuma que não é necessário ler a rota até ao fim. Indique um **invariante de ciclo** que permita demonstrar a correção da função.

Seja $x_1, x_2, \ldots, x_n$ a sequência de nós que o utilizador pretende indicar.

Quando se está a testar a condição de ciclo (linha 5) pela i-ésima vez, para $i \geq 1$ fixo, tem-se:

- já foram lidos $x_1, \ldots, x_i$ e falta ler $x_{i+1}, \ldots, x_n$;
- o valor de v é x_i e o valor de k é i;
- o valor de ok é `true` se s já ocorreu em $x_1, \ldots, x_i$ e é `false` caso contrário;
- o valor de t não ocorreu em $x_1, \ldots, x_{i-1}$;
- o valor de d é o número de elementos do grupo que se poderia transportar de s até x_i, se s já tiver ocorrido ($d = m$ a menos que s tenha ocorrido em $x_1, \ldots, x_{i-1}$ e $L[x_{i-1}, x_i] < m$);
- a variável n mantém o valor dado na linha 3 e m o valor que tem na chamada da função.

(Continua, v.p.f.)

c) [1.1] Usando **indução matemática** (sobre o número de vezes que testa a condição de ciclo), **demonstre o invariante** que indicou e, **aplicando-o**, apresente a dedução de que a função retorna o valor correto.

(i) Caso de base $i = 1$. As instruções 1–4 garantem que n tem o valor indicado, $d = m$, $v = x_1$, $k = 1$ e $ok = \texttt{true}$ se $s = x_1$, sendo $\texttt{false}$, caso contrário, faltando ler $x_2, \ldots, x_n$.

(ii) Hereditariedade. Suponhamos, como hipótese de indução (HI), que o estado das variáveis quando está a testar a condição pela i-ésima vez é o que se definiu (no invariante enunciado). Se a condição de ciclo for satisfeita, então:

- pela HI, concluimos que $d = m$ à entrada da iteração i e, como $t \neq v = x_i$, então t não ocorreu em $x_1, \ldots, x_{i-1}, x_i$;
- executará o bloco 6–11: lê w, o qual pela HI tomará o valor x_{i+1} (ficou por ler $x_{i+2}, \ldots x_n$); incrementa k (usando a HI, conclui-se que k fica com valor $i + 1$); verifica se $w = s$ e, se for, atribuiu $\texttt{true}$ a ok (assinalando o facto de $x_1, \ldots, x_i, x_{i+1}$ passar em s); se $w \neq s$, verifica se $x_1, \ldots, x_i$ já passou em s (pela HI, tal corresponde a verificar o estado da variável ok) e se já tiver passado, em 9–10, reduz d de m para $L[x_i, x_{i+1}]$ se o troço (x_i, x_{i+1}) restringir o número de elementos que se pode transportar de s até x_{i+1}; finalmente, na linha 11, v é atualizado, ficando com o valor de w (ou seja, com x_{i+1});
- Portanto, concluimos que quando se executa o teste da condição de ciclo pela $(i + 1)$-ésima vez, o estado das variáveis satisfaz o invariante (ou seja, a condição que se obtém se substituir i por $i + 1$ no seu enunciado).

Da prova de (i) e (ii) resulta, pelo princípio de indução matemática, que a propriedade (i.e., o invariante) se verifica em todas as iterações do ciclo.

O ciclo termina quando $v = t \vee d \neq m \vee k = n$, passando à execução de 12–14 para dar o resultado.

- Se $v = t \wedge ok = \texttt{true} \wedge d = m$, na linha 12, então, usando o invariante e o facto de $s \neq t$, concluimos que $x_1, \ldots, x_k$ passou em s, que $x_k = t$, e que no percurso de s até t tem lugares suficientes para o grupo.
- Se $v \neq t \vee ok \neq \texttt{true} \vee d \neq m$, na linha 12, então:
  - se $ok = \texttt{false}$ então $x_1, \ldots, x_k$ não passou em s. Assim, se $k = n$ (linha 5), não havia mais nós. Portanto, é correto retornar $\texttt{false}$. Se $k < n$ então $v = t$ (linha 5), isto é $x_k = t$, e também é correto retornar $\texttt{false}$ pois não voltará a encontrar t dado que os nós da rota são todos distintos.
  - Se $ok = \texttt{true}$ e $v = x_k \neq t$, então $k = n$ ou $d < m$ (na paragem do ciclo). Em ambos os casos conclui-se que é correto retornar $\texttt{false}$ pois se $k = n$ a rota terminou sem passar em t e se $d < m$ não será possível transportar o grupo nessa rota. Analogamente se conclui que é correto retornar $\texttt{false}$ se $ok = \texttt{true}$ e $d < m$.

d) [0.3] Assuma que é necessário ler a rota até ao fim. Corrija o programa.

Depois do ciclo "Enquanto" terminar, acrescentar entre a linha 11 e a linha 12, o ciclo seguinte para consumir o que falta ler da rota:

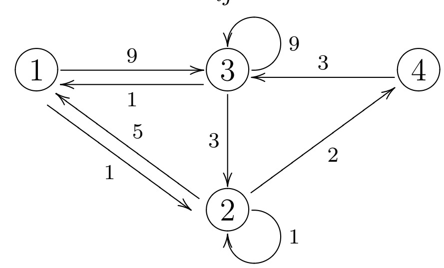
Enquanto $(k < n)$ fazer
    ler(w); $k \leftarrow k + 1$;

N.º [ ] Nome [ ]

9. Seja $G = (V, E, d)$ um grafo dirigido finito, com $V = \{1, 2, \ldots, n\}$, e em que $d(e) \in \mathbb{Z}^+$ define o peso do ramo e, para todo $e \in E$. O peso de um percurso é a soma dos pesos nos ramos do percurso. Considere percursos $\gamma_{ij}^{(k.r)}$ de i para j com no máximo r ramos e que passam num nó k pré-definido (basta que k ocorra, não tem de ser um nó intermédio). Seja $K_{ij}^{(k,r)}$ o **peso mínimo** que um tal percurso pode ter, para k e r fixos. Um percurso tem pelo menos um ramo. Se o percurso não existir, defina $K_{ij}^{(k,r)}$ como ∞.

a) [0.6] Para a instância representada, indique os valores de:

$K_{3,3}^{(3,2)}$	9	$K_{1,4}^{(3,2)}$	∞	$K_{1,4}^{(3,9)}$	10
$K_{1,1}^{(3,6)}$	7	$K_{4,4}^{(3,2)}$	∞	$K_{2,2}^{(3,4)}$	7



b) [0.5] No caso geral, prove que nenhum percurso $\gamma_{ij}^{(k,r)}$ com peso $K_{ij}^{(k,r)}$ contém um ciclo de k para k, a menos que $i = j = k$.

Se contiver um ciclo de k para k então $\gamma_{ij}^{(k,r)} = \gamma_{ik}^{(k,r_1)}\gamma_{kk}^{(k,r_2)}\gamma_{kj}^{(k,r_3)}$, com $\gamma_{kk}^{(k,r_2)}$ não vazio e $r_1+r_2+r_3 = r$ (podendo r_1 ou r_3 ser 0, caso $i = k$ ou $j = k$). O percurso $\gamma_{ik}^{(k,r_1)}\gamma_{kj}^{(k,r_3)}$ teria um peso inferior a $K_{ij}^{(k,r)}$ o que seria absurdo dado que passa em k e tem $r_1 + r_3 \leq r$ ramos, pelo que $\gamma_{ij}^{(k,r)}$ não teria peso mínimo. Portanto, $\gamma_{ij}^{(k,r)}$ só pode conter um ciclo de k para k se for um ciclo de k para k.

c) [~~1.0~~ 0.5(*)] Defina $K_{ij}^{(k,r)}$ por uma recorrência, para todo $(i, j) \in V \times V$ e $r \geq 1$, com $k \geq 1$ fixo, bem como o nó $N_{ij}^{(k,r)}$ que segue o nó i num percurso $\gamma_{ij}^{(k,r)}$ encontrado com peso $K_{ij}^{(k,r)}$. **Explique sucintamente.** (*Sugestão:* como exprimir a matriz $K^{(k,r+1)}$ a partir da matriz $K^{(k,r)}$ e de d?)

Seja $D_{ij} = d(i, j)$ se $(i, j) \in E$ e, caso contrário, $D_{ij} = \infty$. Para $r = 1$, o valor $K_{ij}^{(k,1)}$ requer um percurso de i para j com um ramo e que passe em k, o que obriga a ter $k = i$ ou $k = j$. Assim,

$$K_{ij}^{(k,1)} = \begin{cases} \infty, & \text{se } i \neq k \wedge j \neq k \\ D_{ij} & \text{se } i = k \vee j = k \end{cases} \qquad N_{ij}^{(k,1)} = \begin{cases} 0, & \text{se } (i \neq k \wedge j \neq k) \vee (i,j) \notin E \\ j & \text{se } (i = k \vee j = k) \wedge (i,j) \in E. \end{cases}$$

Para $r \geq 1$, podemos definir $K_{ij}^{(k,r+1)}$ e $N_{ij}^{(k,r+1)}$, distinguindo $i = k$ de $i \neq k$ assim

$$\begin{aligned} K_{kj}^{(k,r+1)} &= \min(K_{kj}^{(k,r)}, \min_{1\leq p\leq n}(K_{kp}^{(k,r)} + D_{pj})) \\ K_{ij}^{(k,r+1)} &= \min(K_{ij}^{(k,r)}, \min_{1\leq p\leq n}(D_{ip} + K_{pj}^{(k,r)})), \qquad \text{para } i \neq k \end{aligned}$$

$$N_{ij}^{(k,r+1)} = \begin{cases} N_{ij}^{(k,r)} & \text{se } K_{ij}^{(k,r+1)} = K_{ij}^{(k,r)} \\ N_{kp}^{(k,r)} & \text{se } i = k \text{ e } K_{kj}^{(k,r+1)} = K_{kp}^{(k,r)} + D_{pj}, \text{ sendo } p \text{ o menor nó nessas condições.} \\ p & \text{se } i \neq k \text{ e } K_{ij}^{(k,r+1)} = D_{ip} + K_{pj}^{(k,r)}, \text{ sendo } p \text{ o menor nó nessas condições.} \end{cases}$$

O percurso mínimo $\gamma_{ij}^{(k,r+1)}$ é um percurso $\gamma_{ij}^{(k,r)}$ com até r ramos ou por um percurso que tem $r + 1$ ramos. Nesse caso, se $i \neq k$, começa por um ramo (i, p), para algum $p \in V$, e os restantes formam um percurso $\gamma_{pj}^{(k,r)}$ ótimo (para passar em k basta que $k = p$). Se $i = k$, então $\gamma_{ij}^{(k,r+1)}$ termina com um ramo (p, j), para algum p, e o percurso de i até p é um percurso ótimo $\gamma_{kp}^{(k,r)}$.

d) [~~0.2~~ 0.0(*)] Indique um valor r_0, dependente de n, tal que $K_{ij}^{(k,r)} = K_{ij}^{(k,r_0)}$, para todo (i,j) e $r \geq r_0$.

$r_0 = 2(n-1)$, pois pode ser preciso passar por todos os nós para chegar de i a k e depois também para chegar de k a j.

e) [~~1.1~~ 0.1 (*)] Escreva (em pseudocódigo) uma função RESOLVE(D,n,k,K), com **complexidade** $O(n^3)$, para obter a matriz K, sendo K_{ij} o peso mínimo de um percurso de i para j que passe por k, com $k \geq 1$ fixo, para todos os pares (i,j). Deve ser baseada na recorrência definida anteriormente e usar **programação dinâmica**. São dados n, k e a matriz D, sendo $D_{ij} = d(i,j)$ se $(i,j) \in E$ (caso contrário, $D_{ij} = \infty$).

Ideia para resolução:

```
RESOLVE(D, n, k, K)
  /* inicializar */
  Para i ← 1 até n fazer
      Para j ← 1 até n fazer
          Se ((i ≠ k ∧ j ≠ 1) ∨ (i, j) ∉ E) então K[i, j] ← ∞;   N[i, j] ← 0;
          senão K[i, j] ← D[i, j];   N[i, j] ← j;
  /* determinar K^(k,n) */
  trocas ← true;
  Enquanto (trocas) fazer
      trocas ← false;
      Para i ← 1 até n fazer
          Para j ← 1 até n fazer
              Para p ← 1 até n fazer
                  Se i ≠ k então
                      Se K[i, j] > D[i, p] + K[p, j] então
                          K[i, j] ← D[i, p] + K[p, j];   N[i, j] ← p;
                          trocas ← true;
                  senão  /* i = k */
                      Se K[k, j] > K[k, p] + D[p, j] então
                          K[k, j] ← K[k, p] + D[p, j];   N[k, j] ← N[k, p];
                          trocas ← true;
```

A complexidade desta função seria $O(n^4)$ porque cada iteração do ciclo "Enquanto" tem complexidade $O(n^3)$ e o número de iterações do ciclo "Enquanto" não excede $2(n-1)+1$, de acordo com 9d).

A complexidade pode ser reduzida para $O(n^3)$, pois podemos evitar o ciclo j. Note-se, por exemplo, que se $i \neq k$ e $j \neq k$, um percurso ótimo de i para j é formado por um percurso ótimo de i para k e um percurso ótimo de k para j. Tendo por base a recorrência, podemos calcular K_{ik}, para todo i, e K_{kj} para todo j em $O(n^3)$, e somar esses valores para obter K_{ij}. A implementação desta ideia requer algum cuidado (podendo usar dois arrays de n inteiros). É de salientar que, uma solução alternativa baseada no algoritmo de Dijkstra poderia ser melhor do que $O(n^3)$.

10. [~~0.4~~ 0.0(*)] Explique de que modo a correção do algoritmo de Kruskal, para cálculo de uma árvore de suporte de peso **máximo** (ou **mínimo**) de um grafo $G = (V,E,d)$, se deduz da correção da estratégia *greedy* que determina um conjunto máximo independente num matróide pesado $(S,\mathcal{F})$. A que corresponde S e $\mathcal{F}$?

/* Em alternativa, resolva questão 5. */

Se se definir $S = E$ e $\mathcal{F}$ como conjunto dos subconjuntos E' de E que definem os subgrafos acíclicos de G (ou seja, cada E' define uma floresta de G), o par $(S,\mathcal{F})$ tem estrutura de matróide.

O algoritmo *greedy* para um matróide pesado determina $E' \in \mathcal{F}$ com peso máximo. Toma $E' = \emptyset$ e, considerando os elementos de S por ordem crescente de peso, acrescenta o próximo elemento $e \in S$ ao conjunto E' desde que $E' \cup \{e\}$ pertença a $\mathcal{F}$ (o que, para o caso considerado, significa que o subgrafo (V,E') é acíclico). Essa estratégia é análoga à que o algoritmo de Kruskal aplica.

(Fim)

Cotações (*): escala corrigida para distribuir parte da cotação de questões de valorização pelas restantes