RODRIGO ALVES DE ALMEIDA

CES-12 EXAME COMP-22

01) A ordenação bitônica baseia-se no funcionamento de um meio-limpador, que possui a seguinte propriedade:

- todos os elementos da metade de cima na saída não serão maiores que os elementos da metade de baixo
- as metades de saída serão bitônicas

Assim, dada uma entrada de n bits bitônica, é possível ordená-la recursivamente utilizando log n meio-limpadores.

Desse modo, pode-se dividir uma entrada de n bits em grupos de 2 bits, de modo que esses grupos são entradas bitônicas.

Paralelamente, ordena-se esses grupos com ordenadores (2), formando assim sequências bitônicas maiores, até que toda sequência seja ordenada:

$$T(n) = T(n/2) + \log n$$

profundidade de cada ordenador

$n = 2^k$

$$T(2^k) = T(2^{k-1}) + k = k + (k-1) + (k-2) + \dots + 1 = \frac{(k+1)k}{2} = \frac{k^2}{2} + \frac{k}{2}$$

$$T(n) = \frac{\log^2 n}{2} + \frac{\log n}{2}$$

$$\Theta(n) = \log^2 n$$

02) A técnica de memoization consiste no armazenamento na memória de sub-problemas que já foram calculados para evitar sobreposição.

Ela pode ser utilizada em problemas onde é necessário resolver subproblemas (particionamento) ou realizar recursões.

Vantagens:

-> a técnica pode ser mais eficiente que algoritmos PD pois resolve os subproblemas "top-down", ou seja, somente resolve aqueles que são necessários para a resposta final

-> pode ser mais eficiente que os algoritmos DC pois evita a sobreposição de subproblemas

Desvantagem:

-> maior ocupação de memória estática para guardar os resultados das variáveis

-> em relação ao DC, pode ter um número excessivo de chamadas recursivas

(03) A partir da árvore de comparações de um vetor de tamanho n, é possível comprovar que, em pior caso, são necessários no mínimo $\Theta(n \log n)$ comparações em um vetor para ordená-lo.

Desse modo, não há nenhum algoritmo que consegue realizar a ordenação em $o(n \log n)$ no pior caso, pois isso implicaria uma resolução em tempo menor que $\Theta(n \log n)$.

O merge Sort, por exemplo, possui tempo de pior caso $\Theta(n \log n)$, portanto sua ordem é ótima.

04 Funcionalidade voltar:

Pilha: a cada visita realizada, colocar o endereço da página em uma pilha. Assim, a volta ocorre para as páginas mais recentemente visitadas

Funcionalidade avançar:

Pilha: cada vez que voltar uma página colocar a página em uma pilha. Desse modo, os avanços seguem a mesma ordem

Assim, são necessárias 2 pilhas.

05) Sim, pode-se percorrer a lista de adjacências ($\Theta(|V|+|E|)$) e, para cada arco $\langle u,v \rangle$, $u \neq v$; criar um novo arco $\langle v,u \rangle$ (se a inserção do arco na lista não for seguindo uma ordem, ela ocorre em $\Theta(1)$).

Nesse novo arco criado, inserir uma flag para evitar que sejam realizadas inserções repetidas (por exemplo, o arco $\langle 2,1 \rangle$ é criado e, ao visitar esse arco, a flag impedirá que o arco $\langle 1,2 \rangle$ seja duplicado)

(06) A soma de matrizes de tamanho n ocorre em tempo n^2 (número de elementos)

Assim, o tempo de operação é:

$$T(n) = 6T(n/2) + 20\left(\frac{n}{2}\right)^2 = 6T(n/2) + 5n^2$$

Fazendo $n = 2^k$

$$T(2^k) = 6T(2^{k-1}) + 5\cdot 2^{2k} = 6\left(6T(2^{k-2}) + 5\cdot 2^{2(k-1)}\right) + 5\cdot 2^{2k} =$$

$$= 6^k T(1) + 5\left(2^{2k} + 6\cdot 2^{2(k-1)} + 6^2\, 2^{2(k-2)} + \dots + 6^k\right) =$$

$$= 6^k T(1) + 10\left(\left(3/2\right)^{k+1} - 1\right)$$

$$T(2^k) = \Theta(6^k) = \Theta(2^{k \log_2 6})$$

$$T(n) = \Theta(n^{\log_2 6})$$

Scanned with CamScanner

07) De acordo com esse método, devemos escolher sempre a atividade compatível que acaba antes:

→ Conjunto com atividade escolhida

1. {8-10; 11-13; 13-16; 12-17; 8-12; 14-20; 10-12; 13-15; 16-18} → {8-10}

2. {11-13; 13-16; 12-17; ~~8-12~~; 14-20; 10-12; 13-15; 16-18} → {8-10; 10-12}

↳ incompatível

3. {~~11-13~~; 13-16; 12-17; 14-20; 13-15; 16-18} → {8-10; 10-12; 13-15}

4. {~~13-16~~; ~~12-17~~; ~~14-20~~; 16-18} → {8-10; 10-12; 13-15; 16-18}

(08) O tempo de resolução de um problema por meio de um algoritmo definido pode variar de acordo com a entrada fornecida. Dado uma entrada de tamanho fixo n, a forma com que o algoritmo lida com as diferentes possíveis entradas é o que dita seu upper bound e lower bound.

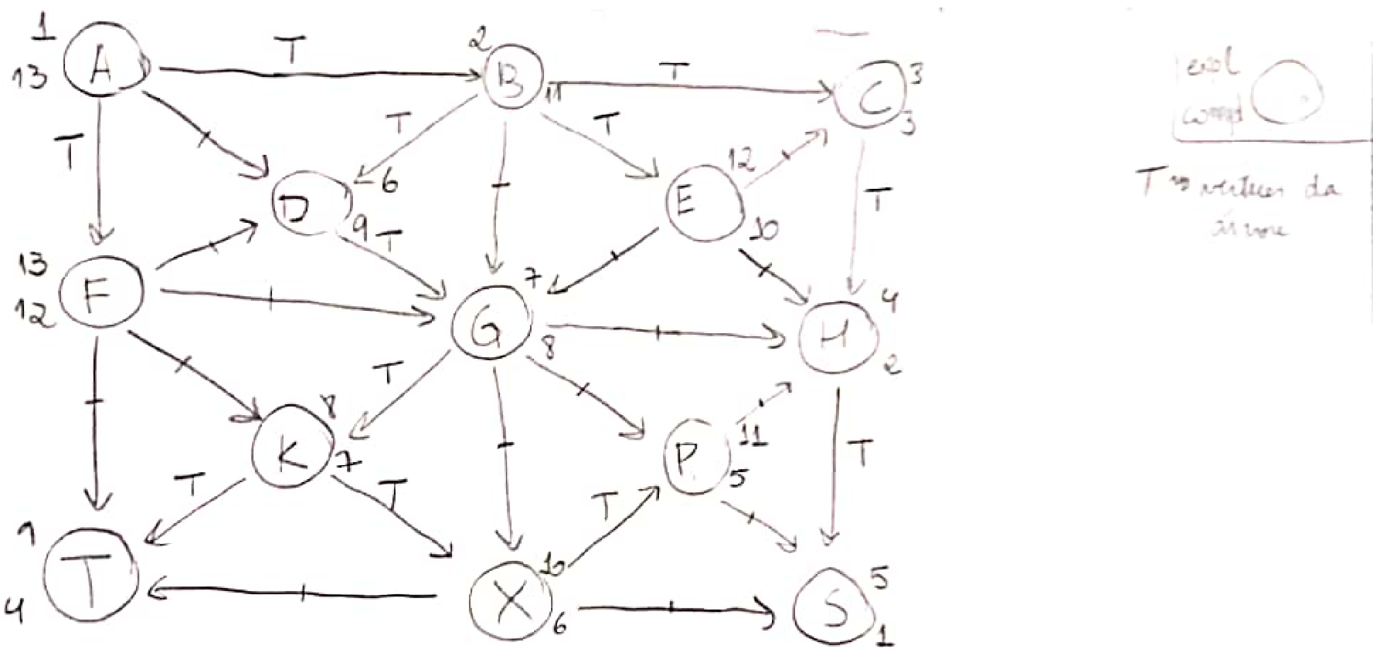
lower bound: menor complexidade de tempo possível gerada por uma entrada

por exemplo: alguns algoritmos de ordenação têm lower bound $\Omega(n)$ quando a entrada já está ordenada

upper bound: maior complexidade de tempo possível gerada por uma entrada

por exemplo. o algoritmo Quick Sort tem $O(n^2)$ se o vetor estiver quase ordenado e a escolha de pivô não for eficiente

09



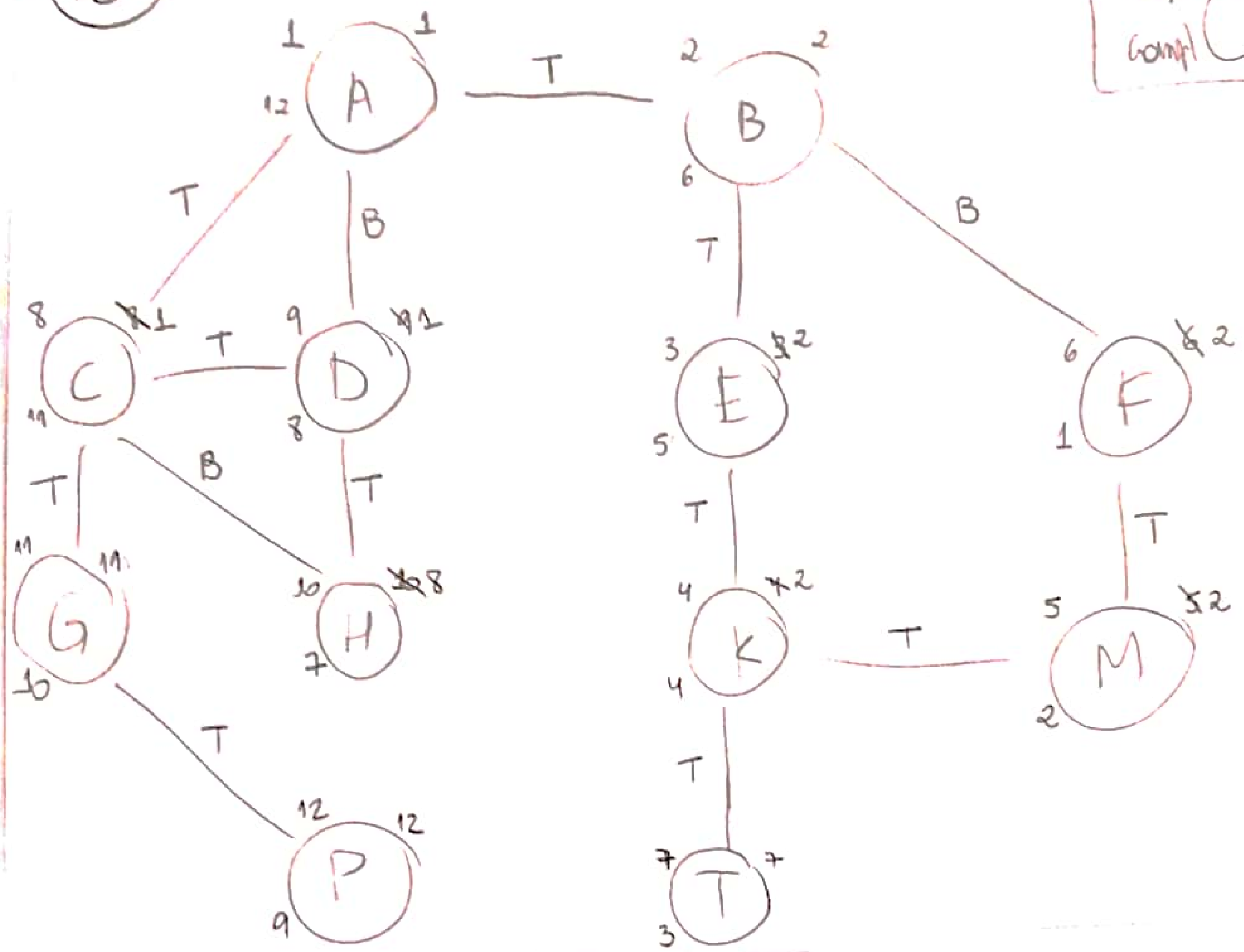
ORdem VISITA → A B C H S D G K T X P E F

ORDEM COMPLETUDE → S H C T P X K G D E B F A

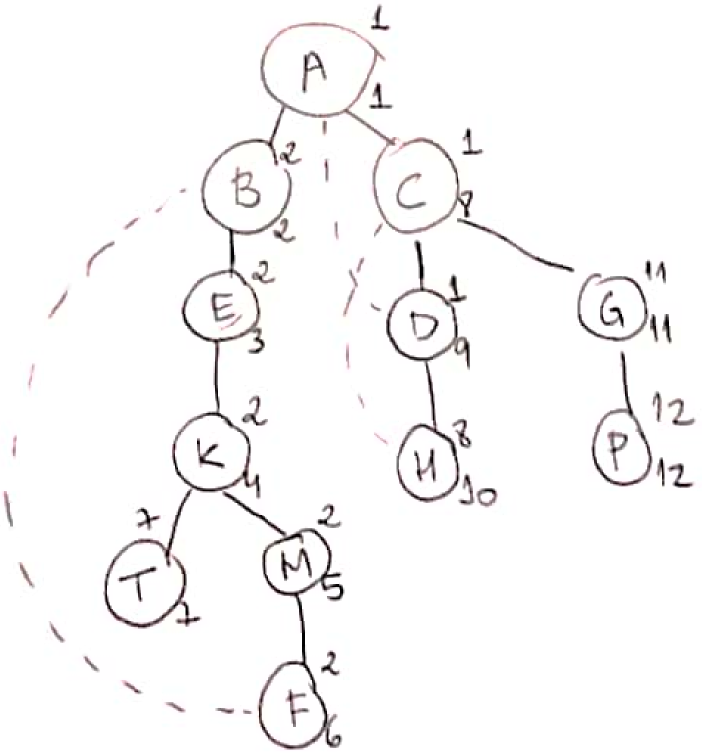
ordenação topológica: A F B E D G K X P T C H S

10

expl
compl



árvore



vértices de corte: A (dois filhos)

B ($m[E] = 2$)

K ($m[T] = 7$)

C ($m[G] = 11$)

G ($m[P] = 12$)

aresta de corte: $\langle A, B \rangle$ ($m[B] = expl[B]$)

$\langle K, T \rangle$ ($m[T] = expl[T]$)

$\langle C, G \rangle$ ($m[G] = expl[G]$

$\langle G, P \rangle$ ($m[P] = expl[P]$)