Q1. [1.5 pontos] Suponha que implementamos listas ligadas com elementos do tipo `Elemento`, definido assim:

```
typedef struct el_struct {
  int chave;
  struct el_struct *prox;
} Elemento;
```

Escreva uma função em `C` de protótipo

```
Elemento *elemento_max(Elemento *prim);
```

que recebe como entrada um ponteiro para o primeiro elemento de uma lista, e devolve um ponteiro para um elemento desta lista que tem campo `chave` máximo. Se a lista dada for vazia (isto é, `prim` for `NULL`), a função deve devolver `NULL`.

Q2. [2 pontos] Suponha que uma lista ligada tem elementos do tipo `Elemento`, declarado como na questão anterior. Escreva uma função em `C` de protótipo

```
Elemento *inverta_lista(Elemento *prim);
```

que inverte a ordem em que se encontram os elementos da lista cujo primeiro elemento é (apontado por) `prim`. O procedimento deve devolver um ponteiro para o primeiro elemento desta lista invertida. Não reproduza as estruturas presentes na lista dada (isto é, não faça cópias destas estruturas); apenas "acerte os ponteiros." (Se a lista dada for vazia, a sua função não faz nada, a não ser devolver `NULL`.)

Q3. [2 pontos] Suponha que os nós de uma árvore binária são do tipo `No`, declarado assim:

```
typedef struct no_struct {
  int chave;
  struct no_struct *esq, *dir;
} No;
```

Chame um nó (apontado por) `p` desta árvore binária de *nó par* se `p->chave` for par. Escreva uma função em `C` de protótipo

```
long prof_total_par(No *t, long *n_nos_pares);
```

que devolve *a soma das profundidades dos nós pares* da árvore binária cuja raiz é (apontada por) `t`. (Devolva -1 se a árvore não tiver nenhum nó par.) Use o parâmetro `n_nos_pares` para devolver o número de nós pares nesta árvore.

Q4. [2.5 pontos] Esta é uma questão sobre o programa `ex11`, que encontra ordenações topológicas. Naquele programa, mantemos uma estrutura que nos permite fazer acesso eficiente aos 'sucessores explícitos' de um elemento. Aqui, chamamos de *sucessores explícitos* de i aqueles elementos j para os quais o par (i, j) foi dado na entrada. Esta estrutura é implementada como uma lista ligada.

(i) Qual é o regime de inserção e remoção nesta estrutura, isto é, esta estrutura é uma fila, pilha, ou deque?

(ii) Qual seria o efeito no programa se alterássemos este regime para um dos outros dois regimes mencionados acima?

(iii) Suponha que as tarefas de uma dada entrada para o programa `ex11` são os inteiros de 1 a 6, e que a tarefa j depende da tarefa i se i divide j (por exemplo, os pares da entrada que envolvem 3 seriam $(1, 3)$ e $(3, 6)$). Liste todas as possíveis ordenações topológicas desta entrada.

(iv) Suponha agora que as tarefas de uma dada entrada para o programa `ex11` são os inteiros de 1 a n, e que a relação de dependência é a mesma de (iii). Prove que o número de ordenações topológicas desta entrada é pelo menos

$$\left\lceil \frac{n}{2} \right\rceil ! \, .$$

(*Sugestão*: considere os elementos $\lfloor n/2 \rfloor + 1, \ldots, n$.)

Considere agora o EP2.

(v) Suponha que temos as tarefas $1, \ldots, 2^k$ ($k \geq 0$), e que a relação de dependência é aquela de (iii) e (iv). Prove por indução em k que existe uma decomposição destas tarefas em $k+1$ rodadas, no sentido do EP2. (*Sugestão*: considere as tarefas $2^{k-1}+1, \ldots, 2^k$.)

(vi) Mostre que, para todo $k \geq 0$, não existe uma decomposição em rodadas das tarefas em (v) com um número menor de rodadas.

Q5. [1.5 pontos] Considere o programa `freq`. Suponha que damos a ele como entrada uma lista de n palavras distintas. Nesta questão, admitimos que qualquer palavra fornecida a este programa é mapeada pela função de espalhamento a um valor independente e uniformemente distribuído (isto é, 'aleatório') do conjunto $\{0, \ldots, M-1\}$, onde $M =$ `HASH_PRIME`

(i) Seja C o número de comparações entre palavras feitas por palavra por `freq` ao montarmos o 'dicionário' com estas n palavras. A esperança de C é, aproximadamente,

$$\frac{n}{2M}.$$

Explique por quê! (Neste item, uma explicação intuitiva é suficiente.)

Suponha agora que o nosso 'dicionário' com as n palavras já está montado. Suponha também que as listas ligadas mais curtas que surgiram em nossa estrutura têm ℓ elementos, e que as mais longas têm L elementos.

(ii) Suponha que buscamos uma palavra que *não* ocorre em nosso dicionário. Quantas comparações entre palavras vamos fazer (a) no melhor caso e (b) no pior caso? Justifique.

(iii) Suponha que buscamos uma palavra que *ocorre* em nosso dicionário. Quantas comparações entre palavras vamos fazer (a) no melhor caso e (b) no pior caso? Justifique.

(iv) Suponha agora que $\ell = L$ (isto é, todas as listas têm o mesmo número de elementos). Suponha também que escolhemos, ao acaso, uma das n palavra presentes em nosso dicionário, e que fazemos a busca desta palavra. Qual é o número esperado de comparações entre palavras que fazemos? Justifique. (Procure ser preciso em sua argumetação.)

Q6. [1.5 pontos] Árvores de busca binária podem ser usadas para ordenar uma lista de elementos. Observe que, por exemplo, o programa `freq_ABB` devolve em sua saída as palavras da entrada em ordem alfabética (além de outras informações).

(i) Você usaria este método de ordenação baseado em árvores de busca binária se soubesse que as entradas para os quais o seu programa será usado têm uma forte tendência de serem 'quase-ordenadas' (isto é, praticamente já vêm ordenadas)? Justifique.

(ii) Suponha que, por algum motivo qualquer, sabemos que a entrada a `freq_ABB` é sempre constituída de n palavras que sempre geram uma árvore de altura no máximo h. Dê um limitante superior simples, em função de n e de h, para o número de comparações entre palavras que `freq_ABB` executa nestas entradas. Justifique.