CORREÇÃO DA PRIMEIRA PROVA
MAC 122 — B.C.C.

[Questão 1]

```
Elemento * elemento_max (Elemento * prim)
{
  Elemento * pmax = prim,  /* pmax aponta para a chave maxima */
           * p ;           /* p    percorre a lista           */

  for (p=prim; p; p=p->prox)
    if (p->chave > pmax->chave)
      pmax = p;
  return pmax;
}
```

[Questão 2]

A solução é a mesma utilizada no §18 do programa `EX11`.

```
Elemento * inverta_lista (Elemento * prim)
{
  Elemento * ptr,
           * lista_final ;

  lista_final = NULL;
  while (prim) {
     ptr  = prim;
     prim = prim->prox;
     ptr->prox = lista_final;
     lista_final = ptr;
  }
  return lista_final;
}
```

[Questão 3]

A solução é parecida com o algoritmo utilizado no § 14 do programa `freq_ABB`. (Veja também o comentário no §12.)

```
long prof_total_par (No * t, long * n_nos_pares)
{
  long se,      /* soma das profund. dos nós pares da sub-árv. esquerda */
       sd,      /*  "   "     "      "   "    "    "     "   direita  */
       ne,      /* número de nós pares na sub-árvore esquerda */
       nd,      /*  "     "  "    "    "     "      direita  */
       somapp;  /* soma das profundidades dos nós pares       */

  if (t == NULL)
    *n_nos_pares = 0;
  else {
    se = prof_total_par (t->esq, &ne);
    sd = prof_total_par (t->dir, &nd);
    *n_nos_pares = ne + nd;
      /*  A seguir, fazemos uma pequena correção pois,            */
      /*  embora a função devolva -1 para árvores sem nós pares,  */
      /*  o ideal é fazer contas com soma das profundidades = 0   */
    if (se == -1) se = 0;
    if (sd == -1) sd = 0;
      /*  Para atribuir somapp, como "subimos" 1 nível na árvore,   */
      /*  devemos, na profundidade total, somar 1 para cada nó par  */
      /*  encontrado nas sub-árvores esquerda e direita.            */
    somapp = se + ne + sd + nd;
    if (t->chave%2 == 0)
      *n_nos_pares ++;
      /*  Não precisa alterar somapp, pois profundidade(raiz) = 0 */
  }
  if (*n_nos_pares == 0)
       return -1;
  else return somapp;
}
```

[**Questão 4**]

[**item (i)**]

A estrutura que permite acesso aos sucessores explícitos está indicada no §5 do programa `EX11`: é a que começa na estrutura `x->topo_suc`.

A inserção e a remoção são feitas nos §11 e §14 respectivamente, nos quais podemos observar que ambas as operações são feitas do mesmo lado da lista. Por isso, tal estrutura é uma pilha.

[**item(ii)**]

Se usássemos fila ou deque, precisaríamos manter mais um apontador (para a outra extremidade da lista).

Com relação à eficiência, *nada* mudaria, pois todos os acessos continuariam a ser feitos nas extremidades e não seria preciso percorrer a lista (fazendo comparações entre chaves) em nenhum momento.

[**item(iii)**]

São 25 possíveis ordenações, listadas a seguir.

123465	123456	123546	125346	152346
124365	124356	124536	125436	152436
132465	132456	132546	135246	153246
123645	123654	123564	125364	152364
132645	132654	132564	135264	153264

[**item(iv)**]

Seja $\mathcal{C} = \{\lfloor n/2 \rfloor + 1, \ldots, n\}$. Se $j > \lfloor n/2 \rfloor$ então j não divide n; portanto nenhum elemento de $\mathcal{C}$ divide qualquer outro. Logo, se N é o número de ordenações topológicas dos elementos de $\mathcal{C}$, então $N =$ número de permutações em $\mathcal{C} = |\mathcal{C}|! = \lceil n/2 \rceil!$.

Assim sendo, para mostrar que há pelo menos $\lceil n/2 \rceil!$ ordenações topológicas da entrada $\{1, 2, \ldots, n\}$, só falta mostrar que há pelo menos *uma* ordenação topológica de $\{1, 2, \ldots, \lfloor n/2 \rfloor\}$ e isso é fácil, pois basta tomar os elemento em ordem crescente.

[**item(v)**]

Base: $k = 0$. Conjunto de tarefas $= \{1\}$. É suficiente $k + 1 = 1$ rodada.

Hip. de indução: Conjunto de tarefas $= \{1, 2, 3, \ldots, 2^{k-1}\}$. Existe decomposição em $(k-1) + 1 = k$ rodadas.

Passo: Seja $k \geq 1$. Conjunto de tarefas $= \{1, 2, 3, \ldots, 2^k\}$.

Usando raciocínio análogo ao do item (iv), vemos que, entre $2^{k-1} + 1$ e 2^k, nenhum elemento divide outro. Logo, essas tarefas são independentes entre si e podem ser executadas na $(k + 1)$-ésima rodada.

[item(vi)]

Tomemos os números $2^0, 2^1, 2^2, \ldots, 2^k$. Essa é uma seqüência de tamanho $k+1$ em que cada elemento divide o seguinte. Portanto temos $k+1$ tarefas que devem ser executadas seqüencialmente.

[Questão 5]

[item (i)]

Como cada palavra é mapeada a um valor uniformemente distribuído, vamos considerar o valor esperado para construir uma *única* lista de colisão.

Conforme visto em aula, para construir uma lista de r elementos, o número médio de comparações é $E(C) = r/2$. No nosso caso, espera-se que as listas formadas tenham aproximadamente o mesmo tamanho, ou seja, $r = n/M$. Logo

$$E(C) \approx \frac{n}{2M}.$$

[item (ii)]

É preciso percorrer a lista até o fim.

(a) ℓ (lista mais curta).

(b) L (lista mais longa).

[item (iii)]

(a) 1 (achar no início da lista).

(b) L (achar no fim da lista mais longa).

[item (iv)]

Há igual probabilidade $(1/\ell)$ de encontrar o elemento na posição j, para $j = 1, 2, \ldots, \ell$. Se ele estiver na posição j, será encontrado após j comparações. Portanto, o número médio de comparações é

$$\frac{1}{\ell}1 + \frac{1}{\ell}2 + \cdots + \frac{1}{\ell}\ell = \frac{1}{\ell}(1 + 2 + \cdots + \ell) = \frac{1}{\ell}\left((1+\ell)\frac{\ell}{2}\right) = \frac{\ell+1}{2}.$$

Logo, o número esperado de comparações é $\left\lceil \frac{\ell+1}{2} \right\rceil$.

[Questão 6]

[item (i)]

Não usaria. Pois a árvore ficaria muito desbalanceada, com os filhos esquerdos quase inexistentes. Em conseqüência, as buscas na árvore seriam muito próximas a uma busca linear.

[item(ii)]

Quando se insere uma palavra na árvore, o número de comparações é igual à profundidade em que será inserida, portanto, é no máximo h.

No entanto pode haver palavras repetidas e, no pior caso, cada palavra repetida será encontrada em uma folha, o que gastaria $h + 1$ comparações.

Desse modo, para a inserção de n palavras, um limitante superior é $n(h + 1)$ comparações.