Estrutura de Dados - MC202D
1º Semestre 2017

Instituto de Computação - UNICAMP

Segunda Lista de Exercícios

Entregue **apenas** os exercícios 1, 2 e 3 até a aula de 23/05/2017 presencialmente

1. Faça uma versão do MergeSort para listas ligadas (simples e sem cabeça) que execute em $O(n \lg n)$, sem alocar memória adicional. Para tanto:

1. Considere que `No` tem apenas os campos `struct No *prox` e `int dado` e queremos ordená-la pelo campo dado.

2. Apresenta a implementação de uma função com protótipo `No *merge(No *lista1, No *lista2)` que devolve um ponteiro para a intercalação de duas listas **já ordenadas** dadas pelos ponteiros `lista1` e `lista2`.

   Dica: o uso de recursão pode simplificar bastante a função `merge`.

3. Apresente a implementação de uma função com protótipo `No *mergesort(No *lista)` que devolve um ponteiro para a lista ordenada usando o algoritmo MergeSort.

   Dica: você pode encontrar o meio de uma lista ligada de uma forma simples. Basta ter dois ponteiros, um que anda um nó por vez e outro que anda dois nós por vez. Quando o segundo ponteiro chegar ao final da fila, o primeiro estará na metade. Porém, tome cuidado para não tentar acessar o campo `prox` de um `No` apontando para `NULL`.

Ambas as funções devem modificar os ponteiros da(s) lista(s) dada(s) ao invés de alocar uma nova lista. Ambas as funções devolvem ponteiros para o primeiro nó da lista computada (evitando assim passagem por referência).

2. Um heap d-ário é um heap onde cada elemento tem até d filhos. Mostre como representar um heap terciário ($d = 3$) de máximo usando um vetor. Em particular,

- Diga como definir as quatro macros a seguir: `PAI(i)`, `FILHO_ESQUERDO(i)`, `FILHO_DO_MEIO(i)` e `FILHO_DIREITO(i)` onde, dado `i`, é computado o índice do pai de `i` no heap e os índices dos filhos esquerdo (primeiro), do meio (segundo) e direito (terceiro) de `i` respectivamente.

- Mostre como implementar a função `void desce_no_heap(FilaP *fp, int k)` que desce o elemento que está na posição `k` até sua posição correta no heap.

3. Apresente um algoritmo para ordenar n inteiros de 0 até $n^2 - 1$ dados em um vetor em tempo $O(n)$ e prove que o seu algoritmo consome tempo $O(n)$.

4. Faça uma implementação do SelectionSort para listas ligadas.

```
No *selection_sort(No *lista) {
  No *t, *max, *ant, *ordenado = NULL;
  while(lista != NULL) {
    /*encontra o maximo*/
    max = lista;
    ant = NULL;
    for (t = lista; t->prox != NULL; t = t->prox) {
      if (t->prox->dado > max->dado) {
        max = t->prox;
        ant = t;
      }
    }
    /*tira da lista*/
    if(ant != NULL)
      ant->prox = max->prox;
    else
      lista = max->prox;
    /*insere no comeco da lista ordenada*/
    max->prox = ordenado;
    ordenado = max;
  }
  return ordenado;
}
```

5. Mostre um esquema para tornar qualquer algoritmo em um algoritmo estável. Quanto espaço e tempo adicional é necessário para o seu esquema?

Discussão em sala.

6. Faça versões mais eficiente de sobe_no_heap e desce_no_heap como feito no InsertionSort, ao invés de fazer trocas todas as vezes.

```
void desce_no_heap_original(FilaP *fp, int k) {
  int filho = FILHO_ESQUERDO(k);
  if (filho < fp->n) {
    if (filho < fp->n - 1 &&
        fp->v[filho].chave < fp->v[filho+1].chave)
      filho++;
    if (fp->v[k].chave < fp->v[filho].chave) {
      troca(&fp->v[k], &fp->v[filho]);
      desce_no_heap_original(fp, filho);
    }
  }
}

void desce_no_heap_iterativo(FilaP *fp, int k) {
  int filho = FILHO_ESQUERDO(k);
  while (filho < fp->n) {
    if (filho < fp->n - 1 && fp->v[filho].chave < fp->v[filho+1].chave)
      filho++;
    if (fp->v[k].chave < fp->v[filho].chave) {
      troca(&fp->v[k], &fp->v[filho]);
      k = filho;
      filho = FILHO_ESQUERDO(k);
    } else
      break;
```

```
  }
}

void desce_no_heap(FilaP *fp, int k) {
  Item t = fp->v[k];
  int filho = FILHO_ESQUERDO(k);
  while (filho < fp->n) {
    if (filho < fp->n - 1 && fp->v[filho].chave < fp->v[filho+1].chave)
      filho++;
    if (t.chave < fp->v[filho].chave) {
      fp->v[k] = fp->v[filho];
      k = filho;
      filho = FILHO_ESQUERDO(k);
    } else
      break;
  }
  fp->v[k] = t;
}
```

7. Mostre que o heapsort não é estável.

Basta simular o heapsort para $0, 1, 1, 2$**.**

8. Suponha que você tenha um heap de máximo com todos os elementos distintos e com n elementos. Quais são as posições possíveis que o elemento mínimo pode ocupar nesse vetor?

Discussão em sala de aula

9. Faça uma versão do QuickSort que é eficiente para vetores com muitos elementos repetidos. Em particular, modifique o algoritmo de partição para quebrar o vetor em três partes, uma parte menor do que o pivô (na esquerda), uma parte maior do que o pivô (na direita) e uma parte igual ao pivô (no meio). Em seguida, mostre como alterar o QuickSort para utilizar essa informação.

```
void partition(int *v, int l, int r, int *ini, int *fim) {
  int pivo = v[l];
  int i = l, f = r;
  int a = l-1, b = r+1, k;
  while (i < f) {
    while (v[f] >= pivo && i < f) {
      if (v[f] == pivo) {
        troca(&v[f], &v[b-1]);
        b--;
      }
      f--;
    }
    if (i < f)
      v[i] = v[f];
    while (v[i] <= pivo && i < f) {
      if (v[i] == pivo) {
        troca(&v[i], &v[a+1]);
        a++;
      }
      i++;
    }
    if (i < f)
      v[f] = v[i];
  }
  v[i] = pivo;
  for (k = l; k <= a; k++) {
```

```
    troca(&v[k], &v[i-1]);
    i--;
  }
  for (k = r; k >= b; k--) {
    troca(&v[k], &v[f+1]);
    f++;
  }
  *ini = i;
  *fim = f;
}

void quicksort(int *v, int l, int r) {
  int i, f;
  if(r <= l) return;
  partition(v, l, r, &i, &f);
  quicksort(v, l, i-1);
  quicksort(v, f+1, r);
}
```

10. Qual é o maior número de trocas envolvendo um determinado elemento do vetor na execução da Ordenação por Seleção?

11. Dê um exemplo de um vetor com n elementos (para um n arbitrário) para o qual o BubbleSort faz o número máximo de trocas.

12. Qual é o tempo de execução (em notação $O(.)$) do BubbleSort adaptativo e InsertionSort adaptativo para os seguintes casos:

a) Quando temos um vetor ordenado

b) Quando temos um vetor ordenado invertido

13. Quantas comparações o QuickSort faz ao ordenar um vetor de n elementos iguais?

14. Faça uma versão top-down do MergeSort que realiza as mesmas intercalações que a versão bottom-up.

15. Um vetor que está ordenado em ordem não-crescente é um heap de máximo? Justifique sua resposta.

16. Qual é a menor altura de uma folha na árvore de decisão de um algoritmo de ordenação?

17. Faça uma versão do MergeSort para listas ligadas que recebe o tamanho da lista como um parâmetro da recursão.

18. Implemente uma versão do BubbleSort (chamada ShakerSort) que alterna entre passar pelo vetor da esquerda para a direita e passar pelo vetor da direita para a esquerda.

19. Faça uma versão do MergeSort que divide o vetor em três subvetores ao invés de dois. Qual é a altura da árvore obtida a partir das chamadas recursivas? Qual é o número de comparações no pior caso?