Métodos de Programação II

LESI / LMCC

28 de Junho 2004

Questão 1 Considere duas sequência de chaves armazenadas em dois arrays globais, `pre` e `in`, que correspondem, respectivamente, ao resultado das travessias *preorder* e *inorder* de uma árvore binária. Assumindo que não há chaves repetidas, podemos afirmar que as duas sequências determinam univocamente a árvore binária. O primeiro elemento em `pre` é a raiz, e as duas sub-árvores podem ser determinadas usando o facto de que a raiz parte o array `in` de forma que todos os elementos da sub-árvore esquerda estão em `in` antes da raiz e todos os elementos da sub-árvore direita estão em `in` depois da raiz. Considere as seguintes declarações:

```
typedef struct node {
  int chave;
  struct node *esq;
  struct node *dir;
} Node;

int pre[MAX], in[MAX];
```

1. (a) Desenhe a árvore definida pelas seguintes sequências de chaves:
   `pre` = 1, 2, 4, 5, 3, 6 e `in` = 4, 2, 5, 1, 3, 6.

   (b) Complete a definição da função `buildtree` que constroi a árvore binária a partir das sequências `in` e `pre`, de forma a que se a árvore binária tem n elementos, `buildtree(0, n-1, 0, n-1)` devolve o apontador para a raiz da árvore.

```
Node * buildtree(int i1, int i2, int p1, int p2)
{ Node *t=NULL;
  int k;
                        // assume-se que o input está correcto
  if (i1>i2 && p1>p2)
     ...A... ;
  else {
        t = malloc(sizeof(Node));
        t->chave = ...B... ;
        for( k=i1; ...C... ; k++);
        t->esq = ...D... ;
        t->dir = ...E... ;
        return t;
       }
}
```

2. Escreva o conteúdo dos arrays `pre` e `in` para os dois casos extremos que correspondem a árvores de altura máxima com as chaves 1, 2, 3, 4, 5 e 6. Estas situações correspondem ao melhor e ao pior caso de execução da função `buildtree`. Justifique esta afirmação.

3. Escreva equações de recorrência que exprimem o tempo de execução de `buildtree` no caso genérico, no melhor caso, e no pior caso. Analise o tempo de execução para cada caso.

Nome:_____ **Número:**_____ **Curso:**_____

Métodos de Programação II

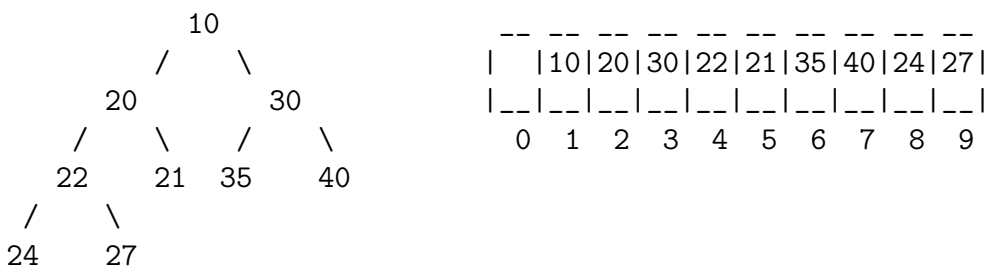
LESI / LMCC

28 de Junho 2004

Questão 2 [*Estruturas de Dados*] Uma *heap* é uma árvore binária que verifica a seguinte propriedade: o conteúdo de cada nó é menor ou igual que o conteúdo dos seus descendentes (não havendo, no entanto, qualquer relação de ordem entre os conteúdos das duas sub-árvores de um mesmo nó).

Uma implementação possível destas estruturas, vantajosa em diversas situações, recorre apenas a um vector, contendo, a partir do índice 1, os elementos de uma *heap* dispostos por níveis, da esquerda para a direita. As heaps assim representadas têm uma forma particular: o nível $k+1$ só pode ter elementos a partir do momento em que o nível k está totalmente preenchido, e cada nível começa a ser preenchido da esquerda para a direita, não havendo nós vazios pelo meio.

Exemplo de uma *heap* e sua representação em vector:



1. Se se acrescentar um elemento no fim do vector, isso corresponderá a acrescentar um folha à árvore, o que pode não resultar numa *heap* uma vez que o conteúdo dessa folha não é necessariamente superior ou igual ao dos seus ascendentes.

   A seguinte função recebe um vector contendo uma heap entre as posições 1 e $n-1$ e um elemento qualquer na posição n. O vector contém, no fim da sua execução, uma *heap* entre as posições 1 e n.

   ```
   void insere_ultimo(int h[], int n)
   {
     int i = n, p;
     while (i!=1) {
       p = ...A...;
       if (...B...) {
         swap(h, p, i);
       }
       ...C...;
     }
   }
   ```

   (`swap` troca o conteúdo de duas posições de um vector).

   Complete a definição da função com as expressões A, B, e C. Sugestão: comece por estabelecer uma relação entre os índices do vector correspondentes a um nó e aos seus dois filhos,

2. Efectue a análise assimptótica de melhor e pior caso da função da alínea anterior. Se não tiver resolvido essa alínea, assuma que A, B, C contêm apenas operações de tempo constante.

Nome:_____ **Número:**_____ **Curso:**_____

3. Estabeleça um invariante para o ciclo `while` da função e prove a correcção do algoritmo.

4. Altere a função anterior por forma a melhorar o funcionamento do algoritmo no melhor caso. Repita a análise assimptótica do tempo de execução para esta nova versão do algoritmo.

Nome:_____ **Número:**_____ **Curso:**_____

Métodos de Programação II

LESI / LMCC

28 de Junho 2004

Questão 3 [*Algoritmos de Grafos*] Considere o algoritmo de Warshall que calcula o fecho transitivo sobre um grafo não pesado.

```
void fechoTrans(int A[][],int F[][], int n) {
    int i,j,k;

    for (i=0;i<n;i++)
       for(j=0;j<n;j++)
          F[i][j]=A[i][j];

    for (k=1 ; k<=n; k++)
        for (i=1 ; i<=n ; i++)
            for (j=1 ; j<=n ; j++)
                if (F[i][k] && F[k][j]) F[i][j] = 1;
 }
```

1. (a) Explique o funcionamento do algoritmo.
   (b) Enuncie (sem provar) um invariante de ciclo para o ciclo mais exterior.
   (c) Identifique e explique a representação do grafo utilizada neste algoritmo.
   (d) Explique por que razão esta representação é utilizada neste caso particular.

2. Apresente uma adaptação deste algoritmo que calcule os caminhos mais curtos entre todos os pares de nós num grafo pesado. Admita que o grafo é fornecido utilizando o mesmo tipo de representação, com o valor `MAXINT` a representar a inexistência de um arco. A matriz resultante deverá conter a distância correspondente ao caminho mais curto entre cada par de nós, caso exista, e `MAXINT` caso não exista.

   ```
   typedef int Weight;

   void distancias(Weight weights[][], Weight dists[][], int n);
   ```

   Diga ainda, justificando, se este algoritmo apresenta vantagens em relação à aplicação exaustiva do algoritmo de Dijkstra, para o mesmo fim.

Nome:_____ **Número:**_____ **Curso:**_____

Questão 4 [*Problemas NP-completos*] Considere o seguinte problema de decisão:

- É dado um conjunto O de n objectos;
- cada objecto i tem dimensão s_i (um número inteiro positivo);
- é dado ainda um número inteiro positivo K;
- pretende-se decidir se existe ou não um sub-conjunto de objectos de O cuja soma de dimensões seja exactamente igual a K.

Prove que se trata de um problema NP, isto é, prove que existe um algoritmo não-determinístico, limitado polinomialmente, que o resolve.

Nome:_____ **Número:**_____ **Curso:**_____