IS 2610: Data Structures

Discuss HW 2 problems
Binary Tree (continued)
Introduction to Sorting

Feb 9, 2004
Binary tree

- A binary tree with $N$ internal nodes has $2N$ links
  - $N-1$ to internal nodes
    - Each internal node except root has a unique parent
    - Every edge connects to its parent
  - $N+1$ to external nodes

- Level, height, path
  - Level of a node is $1 +$ level of parent (Root is at level 0)
  - Height is the maximum of the levels of the tree’s nodes
  - Path length is the sum of the levels of all the tree’s nodes
  - Internal path length is the sum of the levels of all the internal nodes
Examples

- Level of D?
- Height of tree?
- Internal length?
- External length?

- Height of tree?
- Internal length?
- External length?
Binary Tree

- External path length of any binary tree with \( N \) internodes is \( 2N \) greater than the internal path length.

- The height of a binary tree with \( N \) internal nodes is at least \( \lg N \) and at most \( N-1 \).
  - Worst case is a degenerate tree: \( N-1 \).
  - Best case: balanced tree with \( 2^i \) nodes at level \( i \).
    - Hence for height: \( 2^{h-1} < N+1 = 2^h \) – hence \( h \) is the height.
Binary Tree

- Internal path length of a binary tree with $N$ internal nodes is at least $N \lg (N/4)$ and at most $N(N-1)/2$
  - Worst case: $N(N-1)/2$
  - Best case: $(N+1)$ external nodes at height no more than $\lceil \lg N \rceil$
    - $(N+1) \lceil \lg N \rceil - 2N < N \lg (N/4)$
Tree traversal (binary tree)

- Preorder
  - Visit a node,
  - Visit left subtree,
  - Visit right subtree

- Inorder
  - Visit left subtree,
  - Visit a node,
  - Visit right subtree

- Postorder
  - Visit left subtree,
  - Visit right subtree
  - Visit a node
Recursive/ Nonrecursive Preorder

```c
void traverse(link h, void (*visit)(link))
{
    if (h == NULL) return;
    (*visit)(h);
    traverse(h->l, visit);
    traverse(h->r, visit);
}
```
```c
void traverse(link h, void (*visit)(link))
{
    STACKinit(max);
    STACKpush(h);
    while (!STACKempty())
    {
        (*visit)(h = STACKpop());
        if (h->r != NULL) STACKpush(h->r);
        if (h->l != NULL) STACKpush(h->l);
    }
}
```
Recursive binary tree algorithms

- Exercise on recursive algorithms:
  - Counting nodes
  - Finding height
### Sorting Algorithms: Selection sort

- **Basic idea**
  - Find smallest element and put in the first place
  - Find next smallest and put in second place
  - ..
- **Try for**
  
2 6 3 1 5

```c
#define key(A) (A)
#define less(A, B) (key(A) < key(B))
#define exch(A, B) { Item t = A; A = B; B = t; }

void selection (Item a[], int l, int r) {
    int i, j;
    for (i = l; i < r; i++) {
        int min = i;
        for (i = i+1; i <= r; j++)
            if (less(a[j], a[min]) min = j;
        exch(a[i], a[min]);
    }
}
```
Sorting Algorithms: Selection sort

- Recursive?
  - Eliminate the outer loop
  - Find the minimum and place it in the first place
  - Make recursive call to sort the remaining parts of the array

- Complexity
  - $i$ goes from 1 to $N-1$
  - There is one exchange per iteration; total $N-1$
  - There is $N-i$ comparisons per iteration
    - $(N-1) + (N+2) + \ldots + 2 + 1 = N(N-1)/2$
Sorting Algorithms: Bubble sort

- Bubble sort
  - Move through the elements exchanging adjacent pairs if the first one is larger than the second
  - Try out!
    - 2 6 3 1 5

```c
#define key(A) (A)
#define less(A, B) (key(A) < key(B))
#define exch(A, B) { Item t = A; A = B; B = t; }

void selection (Item a[], int l, int r) 
{
    int i, j;
    for (i = l; i < r; i++)
        for (j = r; j > r; j--)
            if (less(a[j-1], a[j]) exch(a[j-1], a[j]);
}
```
Sorting Algorithms: Bubble sort

- Recursive?
- Complexity
  - $i^{th}$ pass through the loop – $(N-i)$ compare-and-exchange
  - Hence $N(N-1)/2$ compare-and-exchange is the worst case
  - What would be the minimum number of exchanges?
Sorting Algorithms: Insertion sort

- Insertion sort
  - “People” method

2 6 3 1 5

- Complexity
  - About $N^2/4$
  - About half of the left array

```c
#define less(A, B) (key(A) < key(B))
#define exch(A, B) { Item t = A; A = B; B = t; }
#define compexch(A, B) if (less(B, A)) exch(A, B)

void insertion(Item a[], int l, int r) {
  int i;
  for (i = r; i > l; i--) compexch(a[i-1], a[i]);
  for (i = l+2; i <= r; i++) {
    int j = i; Item v = a[i];
    while (less(v, a[j-1])) {
      a[j] = a[j-1]; j--;
    }
    a[j] = v;
  }
}
```
Sorting Algorithms: Shell sort

- Extend of insertion sort
  - Taking every $h^{th}$ element will
  - Issues
    - Increment sequence?

$h = 13$

```c
void shellsort(Item a[], int l, int r)
{
    int i, j, h;
    for (h = 1; h <= (r-l); h = 3*h+1) ;
    for ( ; h > 0;)
        h = h/3;
    for (i = l+h; i <= r; i++)
        {
            int j = i; Item v = a[i];
            while (j >= l+h && less(v, a[j-h]))
                {
                    a[j] = a[j-h]; j -= h;
                }
            a[j] = v;
        }
}
```
void shellsort(Item a[], int l, int r)
{ int i, j, h;
  for (h = 1; h <= (r-l); h = 3*h+1) ;
  for ( ; h > 0;)
    h = h/3;
  for (i = l+h; i <= r; i++)
  { int j = i; Item v = a[i];
    while (j >= l+h && less(v, a[j-h]))
      { a[j] = a[j-h]; j -= h; }
    a[j] = v;
  }
}
Sorting Algorithms: Quick sort

- A divide and conquer algorithm
  - Partition an array into two parts
  - Sort the parts independently
  - Crux of the method is the partitioning process
    - Arranges the array to make the following three conditions hold
      - The element a[i] is in the final place in the array for some i
      - None of the elements in a[l]….a[i-1] is greater than a[i]
      - None of the elements in a[i+1]….a[r] is less than a[i]
void quicksort(Item a[], int l, int r)
{
    int i;
    if (r <= l) return;
    i = partition(a, l, r);
    quicksort(a, l, i-1);
    quicksort(a, i+1, r);
}

int partition(Item a[], int l, int r)
{
    int i = l-1, j = r; Item v = a[r];
    for (;;)
    {
        while (less(a[++i], v));
        while (less(v, a[--j]) if (j == l) break;
        if (i >= j) break;
        exch(a[i], a[j]);
    }
    exch(a[i], a[r]);
    return i;
}
Sorting Algorithms: Quick sort

```c
int partition(Item a[], int l, int r)
{
    int i = l-1, j = r; Item v = a[r];
    for (;;)
    {
        while (less(a[++i], v)) ;
        while (less(v, a[--j])) if (j == l) break;
        if (i >= j) break;
        exch(a[i], a[j]);
    }
    exch(a[i], a[r]);
    return i;
}
```