Lecture P7: Advanced Recursion

Overview

What is recursion?
- When one function calls ITSELF directly or indirectly.

Today.
- Quicksort.
- Dragon curve.
- Travelling salesperson problem.

Quicksort

- Partition array so that:
  - some partitioning element \( a[m] \) is in its final position
  - no larger element to the left of \( m \)
  - no smaller element to the right of \( m \)

\[
\begin{align*}
\text{QUICKSORT} & \quad \text{L} \\
\text{ICKICLQUORTSOO} & \\
\leq L & \quad \geq L
\end{align*}
\]

partitioned array

\[
\begin{align*}
\text{QUICKSORT} & \quad \text{L} \\
\text{ICKICLQUORTSOO} & \\
\text{CCIIK} & \quad \text{O} \quad \text{O} \quad \text{O} \quad \text{QRSSTU}
\end{align*}
\]
Quicksort

Partition array so that:
- some partitioning element \( a[m] \) is in its final position
- no larger element to the left of \( m \)
- no smaller element to the right of \( m \)

Sort each “half” recursively.

```c
void quicksort(int left, int right) {
    int m;
    if (right > left) {
        m = partition(left, right);
        quicksort(left, m - 1);
        quicksort(m + 1, right);
    }
}
```

quicksort.c (see Sedgewick Program 7.1)

---

Quicksort : Implementing Partition

```c
int partition(int left, int right) {
    int i = left - 1; /* left to right pointer */
    int j = right;  /* right to left pointer */
    for(;;) {
        while (a[++i] < a[right])
            ;
        while (a[right] < a[--j])
            if (j == left)
                break;
        if (i >= j)
            break;
        swap(i, j);
    }
    swap(i, right);
    return i;
}
```

partition (see Sedgewick Program 7.2)

---

```c
#define N 14
char a[] = "pseudomythical";
void swap(int i, int j) {
    char t;
    t = a[i];
    a[i] = a[j];
    a[j] = t;
}

int main(void) {
    printf("%s", a);
    quicksort(0, N-1);
    printf("%s", a);
    return 0;
}
```

main()
Dragon Curve

Fold a wire in half \( n \) times. Unfold to right angles.

\[
\begin{array}{c}
n = 0 & n = 4 \\
n = 1 & \\
n = 2 & n = 12 \\
n = 3 & \\
\end{array}
\]

Drawing a Dragon Curve

Use simple “turtle graphics.”

- \( F \): move turtle forward one step (pen down).
- \( L \): turn left 90°.
- \( R \): turn right 90°.

Example.

- \( F L F L F \)

Recursive Dragon Curve Program

A dragon curve of order \( n \) is:

- Dragon curve of order \( n-1 \).
- Move left.
- Dragon curve of order \( n-1 \) backwards.

```
dragon(int n) {
    if (n == 0)
        F();
    else {
        dragon(n-1);
        L();
        nogard(n-1);
    }
}
```

```
void F(void) {
    printf("10 0 rlineto\n");
}
```

```
void L(void) {
    printf("90 rotate\n");
}
```

```
void R(void) {
    printf("-90 rotate\n");
}
```

need implementation of nogard()
Drawing a Dragon Curve

To get nogard(n):
1. dragon(2): F L F F R F
2. nogard(2): F L F R F F
3. dragon(3): F L F F R F L F F R F

void nogard(int n) {
    if (n == 0)
        F();
    else {
        dragon(n-1);
        R();
        nogard(n-1);
    }
}

Enumerating All Permutations

Enumerate all permutations of a set of elements.
1. N elements ⇒ N! possibilities
2. If elements named a, b, c, then 6 possible permutations are:
   abc, acb, bac, bca, cab, cba.

Key idea: permutations of abcde are one of the following:
1. End with a preceded by one of 4! permutations of bcde.
2. End with b preceded by one of 4! permutations of acde.
3. End with c preceded by one of 4! permutations of abde.
4. End with d preceded by one of 4! permutations of abce.
5. End with e preceded by one of 4! permutations of abcd.

Reduces enumerating permutations of N elements to enumerating
permutations of N-1 elements.

Recursive solution for trying all permutations:
1. Array a[] to store current permutation.
2. Initially a[] = "abcde"

```c
void swap(char a[], int i, int j) {
    int t;
    t = a[i]; a[i] = a[j]; a[j] = t;
}

void enumerate() { . . . }
int main(void) {
    char a[] = "abcde";
    enumerate(a, 5);
    return 0;
}
```
Application: Traveling Salesperson Problem

Given N points, find shortest tour connecting them.

- Brute force: try all N! possible permutations.

Recursive solution for finding best TSP tour.

- Store coordinates of points in a[].
- Replace printf() with checklength().
- Takes N! steps.
- No computer can run this for N ≥ 100.
  - 100! > 10^{150}.

Is there an efficient way to do this computation?