1 Data Structures and Algorithm Analysis (based on slides of Harry Zhou)

Algorithm:

A finite sequence of operations that solves a given task

Good characteristics:

- correctness
- efficiency
- elegance, simplicity, robustness

Data Structures:

A way to organize data so that common operations (e.g. insert a new item, search, etc.) are performed efficiently

Examples: linked lists, queues, stacks, trees, graphs

2 The maximum contiguous subsequence sum problem

Given integers A_1 A_2 .. A_n.

Find the sequence that produces the max value $\sum^{j}_{k=i} A_k$

The sum is a zero if all integers < 0

Example:

-2 11 -4 13 -5 2

The sum of the sequence from the 2nd to the 4th number is 20, which is the largest possible value

1 -3 4 -2 -1 6

The sum of the sequence from the 3rd number to the 6th number (4 -2 -1 6)

is the largest possible value: 7

3 Algorithm 1: conduct an exhaustive search(a brute force algorithm)

It calculates all subsequences, such as the sequence of 1 number, the sequence of 2 numbers, and so on, starting at the first position, then at the 2nd position, until the last position

```
max = 0
for(i=0; i<length;  i++)
  for(j=i;  j<length; j++)
            sum = 0
            for(k=i; k<=j; k++)
              sum += a[k]
            if(sum > max)
                    max = sum
                    start = i
                    end =  j
```

Time complexity: O(n^3)

Example: The following numbers are in the array a:

-2 11 -4 13 -5

Trace:

(1) i=0

j=0 sum = -2

j=1 sum = -2+11

j=2 sum = -2+11-4

j=3 sum = -2+11-4+13

j=4 sum = -2+11-4+13-5

max = 18

i=0

j=3

4 Cont. trace

(2) i=1

j=1 sum = 11

j=2 sum = 11-4

j=3 sum = 11-4+13

j=4 sum=11-4+13-5

max = 20

i=1

j=3

(3) i=2

j=2 sum = -4

j=3 sum = -4+13

j=4 sum = -4+13-5

max, i and j remain unchanged

(4) i =3

j=3 sum = 13

j=4 sum = 13-5

max, i and j remain unchanged

(5) i=4

j=4 sum = -5

max, i an j remain unchanged

SOME SUMS

$$1 + 2 + \cdots + n = \frac{n(n+1)}{2}$$

$$1^2 + 2^2 + \cdots + n^2 = \frac{n(n+1)(2n+1)}{6}$$

```
max = 0
for(i=0; i<length;  i++)
  for(j=i;  j<length; j++)
        sum = 0
        for(k=i; k<=j; k++)
          sum += a[k]
        if(sum > max)
              max = sum
              start = i
              end =  j
```

[redacted]

ALG.1

$7\,(j-i+1)$ operations

$$\sum_{j=i}^{n-1} \left(4+7(j-i+1)\right) = (4+7\cdot 1)+(4+7\cdot 2)+\dots+(4+7(n-i))$$

$$= 4\cdot(n-i) + 7\,(1+2+\dots+(n-i))$$

$$= 4\,(n-i) + 7\cdot\frac{(n-i)\,(n-i+1)}{2}$$

$$\sum_{i=0}^{n-1} \left(3+4\,(n-i)+7\cdot\frac{(n-i)\cdot(n-i+1)}{2}\right)$$

$$= 3\cdot n + 4\,(n+(n-1)+\dots+1) + \frac{7}{2}\cdot\left(n(n+1)+(n-1)\cdot n+(n-2)(n-1) + \dots\ 1\cdot 2\right)$$

$$= 3n + 4\cdot\frac{n\cdot(n+1)}{2} + \frac{7}{2}\cdot\left(n^2+n+(n-1)^2+(n-1)+\dots+1^2+1\right)$$

$$= 3n + 2n\cdot(n+1) + \frac{7}{2}\,\frac{n(n+1)(2n+1)}{6} + \frac{7}{2}\cdot\frac{n\cdot(n+1)}{2} =$$

$$\boxed{= \frac{7}{6}\,n^3 + \frac{11}{2}\cdot n^2 + \frac{22}{3}\cdot n + 0}$$

5 Algorithm 2

Observation: $\sum^{j}_{k=i} A_k = A_j + \sum^{j-1}_{k=i} A_k$

Sequence: -2 11 -4 13 -5

Example: once we calculate -2+11-4+13=18, we need to perform only one addition to calculate the entire sequence

that is: 18-5 = 13

The algorithm

```
max=0
for(i=0; i<length; i++)
        sum =0
        for(j=i; j<length; j++)
         sum += a[j]
         if sum > max
                max = sum;start=i;end=j;
```

Time complexity: $O(n^2)$

The statement sum +=a[j] adds one additional number to the sum reducing redundant work

ALG.2

```
max=0
for(i=0; i<length; i++)
        sum =0
        for(j=i; j<length; j++)
          sum += a[j]
          if sum > max
                max = sum;start=i;end=j;
```

$7(n-i)$

$$\sum_{i=0}^{n-1} 3 + 7(n-i) = 3n + 7\cdot\left(n + (n-1) + \ldots + 1\right)$$

$$= 3n + 7\cdot\frac{n\cdot(n+1)}{2}$$

$$= \frac{7}{2}\cdot n^2 + \frac{13}{2}\cdot n$$

TOTAL: $\frac{7}{2}\cdot n^2 + \frac{13}{2}\cdot n + 2$

ALGORITHM 3

We define:

$d[j]$ = max sum of a contiguous subsequence that ends with A_j.

$A_0 \quad A_1 \quad \ldots \quad \boxed{\ldots \quad A_j} \quad \ldots \quad A_{n-1}$

$$d[0] = A_0$$

$$d[j] = \begin{cases} d[j-1] + A_j, & \text{if } d[j-1] > 0 \\ A_j, & \text{if } d[j-1] \leq 0 \end{cases}$$

So it is possible to calculate in order

$d[0], d[1], \ldots, d[n-1]$,

and then find $\max\{d[i], i = 0, \ldots, n-1\}$

EXAMPLE $2, -1, -3, 6$

$d[0] = 2$, $d[1] = 1$, $d[2] = -2$, $d[3] = 6$

<u>Improvement:</u> Since d[j] depends only on d[j-1], we do not need an array to keep the d[] values. In the pseudocode (next slide), we keep the current value of d[j] in the variable *sum*, we use *max* to determine the largest d[j] and we calculate *sum* and *max* in the same loop.

we called it d[j]

7 Cont. algorithm 3

```
i=0; max =0; sum = 0;
for(j=0; j<length; j++)
        sum +=a[j]
    if sum > max
        max = sum
        start=i; end = j;
    else
        if(sum <0)
            i=j+1
            sum=0
```

Time complexity: O(n)

sequence: -2 11 -4 13 -5

(1) j=0 a[j] = -2

i=0   sum -2   max =0

i=1 (reset)

(2) j=1 a[j] = 11

i=1   sum=11 max=11

start=1   end =1

(3) j=2 a[j]= -4   sum = 7 max =11

(4) j=3 a[j] =13   sum =20 max = 20

start =1   end = 3

(5) j=4 a[j]=-5   sum = 15 max =20

start =1   end =3

Basic idea: as long as the sum >0, add another number to it. If not, start over again

Dynamic programming

Algorithm 3 is an example of a dynamic programming algorithm.

Dynamic programming – a general technique used in the design of algorithm.

IDEA: break the problem into many subproblems

Each subproblem can be solved using some of the smaller subproblems

Store the solutions of the subproblems in a table (memoization), so that when we need a solution we have it handy.

In our example: subproblems are each d[i]; in the end we did not use a table for memoization, because for each d[i] we only needed d[i-1].

9 Summary:

Algorithm 1 is $O(n^3)$

Algorithm 2 is $O(n^2)$

Algorithm 3 is $O(n)$

and all of them are correct algorithms

One importance issue in the design of algorithm is efficiency

Runtime efficiency is important

Let us assume we use a computer capable of 10^6 instructions/sec.

	n	n^2	n^3
n=10	< 1 sec	< 1 sec	< 1 sec
n=1000	< 1 sec	1 sec	18 min
n=10000	< 1 sec	2 min	12 days
n= 10^6	1 sec	12 days	31710 years

11 Steps in designing software

- Specification

  input, output, expected performance, features and sample of execution

- System Analysis

  chose top-down design or OOD (sort of bottom-up)

- Design (OUR FOCUS IN THIS COURSE)

  * choose data representation: create ADTs

  * algorithm design

- Refinement and Coding

  implementation

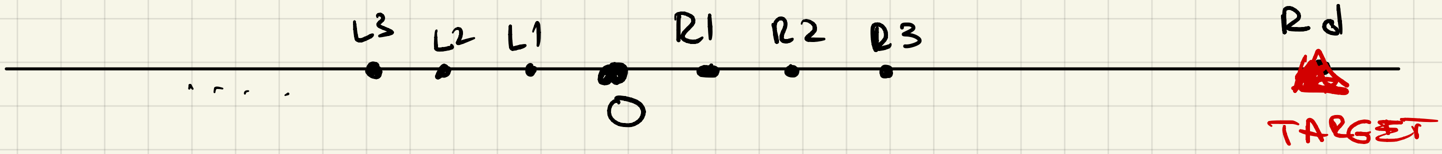
- Verification

  correctness analysis and testing

- Documentation

  manual and program comments

ANOTHER PROBLEM: FINDING LOCATION ON A LINE



. Robot starts at O; target is d steps away from O left or right (unknown which), d is not known

ALG. 1

GO TO

$R1 \rightarrow 1$ step

$L1 \rightarrow 2$ steps

$R2 \rightarrow 3$ steps

$L2 \rightarrow 4$ steps

$\vdots$

R_{d-1}

$L_{d-1} \rightarrow 2d-2$ steps

$R_d \rightarrow 2d-1$ steps

$$\text{TOTAL} = 1+2+3+\dots+(2d-1) = \frac{(2d-1)\cdot 2d}{2} = 2d^2-d$$

If target is at L_d:

$$\text{TOTAL} = 1+2+\dots+2d = \frac{2d(2d+1)}{2} = 2d^2+d$$

$$= O(d^2).$$

ALGORITHM 2

go to powers of 2

Let k be so that: $2^{k-1} < d \leq 2^k$

Go to

$$\begin{array}{lcl}
R\ 1 & \longrightarrow & 0 + 1 \\
L\ 1 & \longrightarrow & 1 + 1 \\
R\ 2^1 & \longrightarrow & 1 + 2 \\
L\ 2^1 & \longrightarrow & 2 + 2 \\
R\ 2^2 & \longrightarrow & 2 + 4 \\
L\ 2^2 & \longrightarrow & 2^2 + 4 \\
 & \vdots & \\
R\ 2^{k-1} & \longrightarrow & 2^{k-2} + 2^{k-1} \\
L\ 2^{k-1} & \longrightarrow & 2^{k-1} + 2^{k-1} \\
R\ d & \longrightarrow & 2^{k-1} + d \leq 2^{k-1} + 2^k
\end{array}$$

TOTAL = COLUMN 1 + COLUMN 2 =

$$= 0 + (1+1) + (2+2) + \dots + (2^{k-1} + 2^{k-1}) +$$

$$+ (1+1) + (2+2) + \dots + (2^{k-1} + 2^{k-1}) + 2^k$$

$$= 4\,(1 + 2 + \dots + 2^{k-1}) + 2^k$$

$$= 4\,(2^k - 1) + 2^k = 5 \cdot 2^k - 4 \leq 10 \cdot d - 4$$

$$= O(d).$$

USEFUL FORMULA

$$1+q+\dots+q^{k-1} = \frac{q^k-1}{q-1}, \text{ if } q \neq 1.$$

In particular, for $q=2$:

$$1+2+\dots+2^{k-1} = 2^k - 1$$