Design and Analysis of Algorithms

Presented by Dr. Li Ning

Shenzhen Institutes of Advanced Technology, Chinese Academy of Science
Shenzhen, China

Dynamic Programming

Solving Subproblems

Subproblems: Disjoint or Overlap

Recall with **Divide-and-Conquer**

1. partition the problem into subproblems
2. solve subproblems recursively
3. combine to get the solution for the original problem

Subproblems: Disjoint or Overlap

- if the subproblems are disjoint, e.g.
  - in MergeSort, a sequence is divided into two disjoint parts
  - in Maximum-Subarray, an array is divided into two disjoint arrays

  not much recalculation by recursively solving subproblems

Subproblems: Disjoint or Overlap

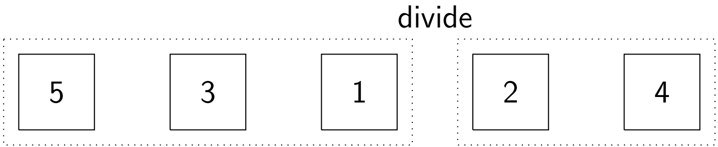
- if the subproblems are disjoint, e.g.
    - in MergeSort, a sequence is divided into two disjoint parts
    - in Maximum-Subarray, an array is divided into two disjoint arrays

  not much recalculation by recursively solving subproblems
- if the subproblems overlap, e.g.
    - in Fibonacci Numbers, then calculation of $F(n-1)$ overlaps a lot with the calculation of $F(n-2)$

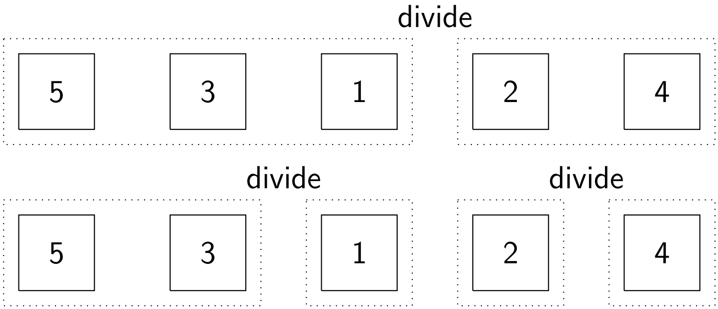
  how to avoid the recalculations?

Revisit: MergeSort

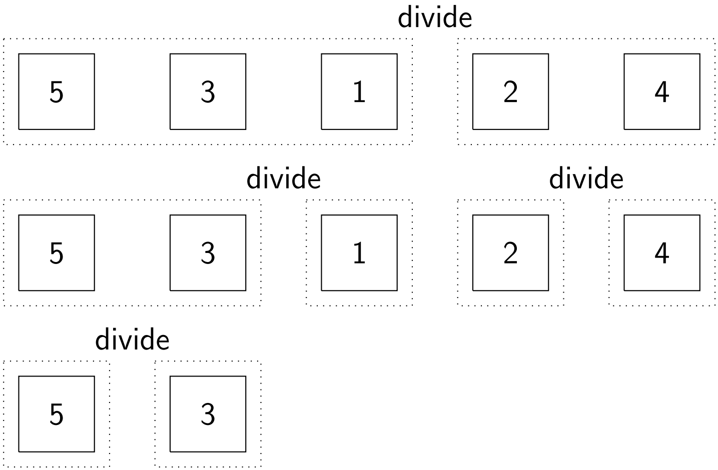
MergeSort: Top Down



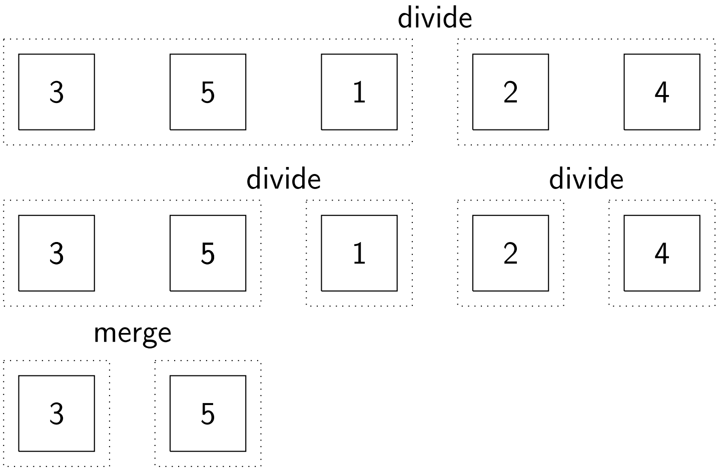
MergeSort: Top Down



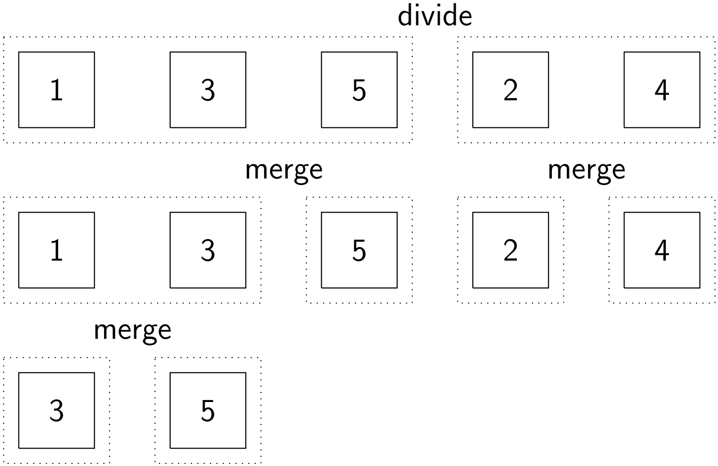
MergeSort: Top Down



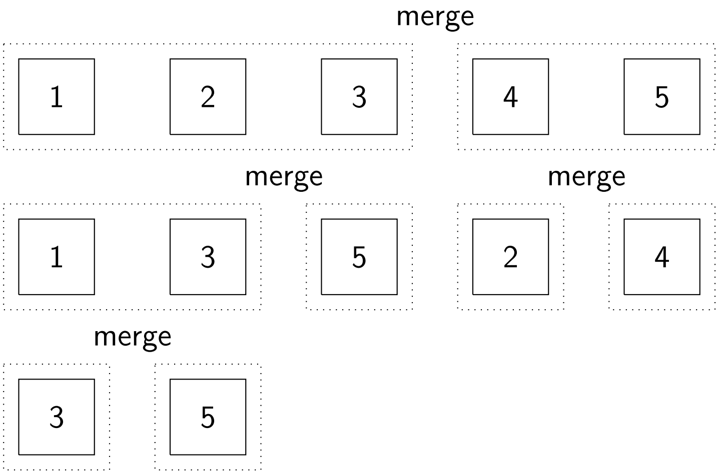
MergeSort: Top Down



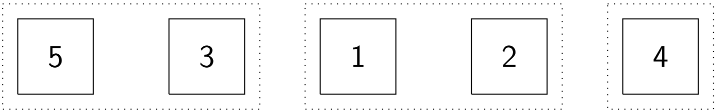
MergeSort: Top Down



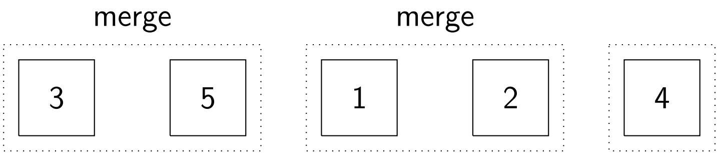
MergeSort: Top Down



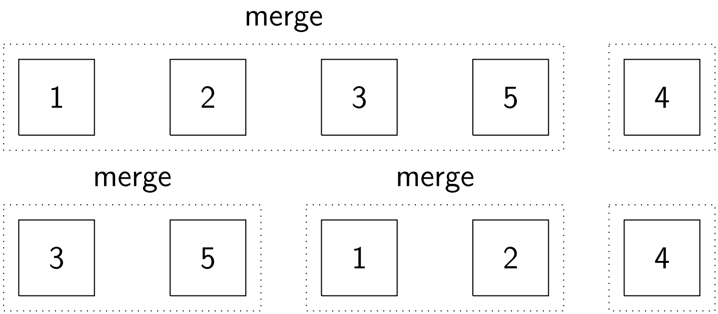
MergeSort: Bottum Up



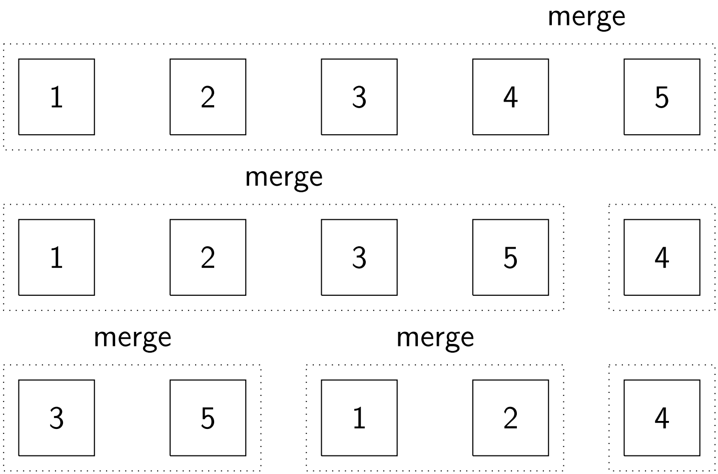
MergeSort: Bottum Up



MergeSort: Bottum Up



MergeSort: Bottum Up



MergeSort: Bottum Up

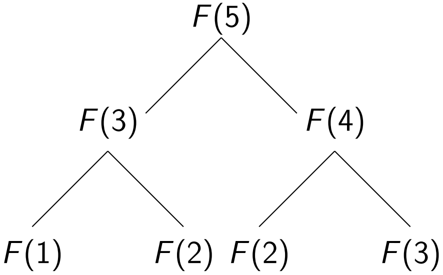
```
void mergesort(int A[], int a, int b) {
  int n = b - a + 1;
  int m = (int) floor(log2(n));
  for (int p = 0; p <= m; p++) {
    int k = (int) pow(2, p), r = 2 * k;
    for (int a1 = a; a1 <= (n / r) * r; a1 += r) {
      int b1 = a1 + k - 1;
      if (b1 < b) {
        int a2 = b1 + 1, b2 = a2 + k - 1;
        if (b2 > b) b2 = b;
        merge(A, a1, b1, a2, b2);
      }
    }
  }
}
```

Revisit: Fibonacci Numbers

Fibonacci Number

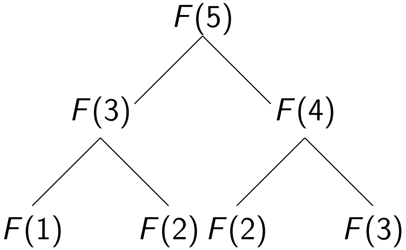
The n-th Fibonacci number is defined by

$$F_n = \begin{cases} F_{n-1} + F_{n-2}, & n \geq 2 \\ 1, & n = 1 \\ 0, & n = 0 \end{cases}$$



Fibonacci Number

Following the definition



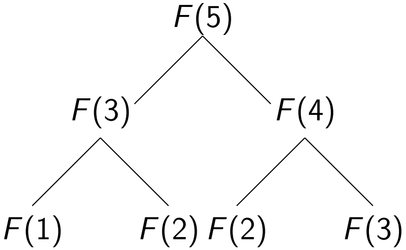
7. $F(5)$

Fibonacci Number

Following the definition

2. ** $F(3)$

6. ** $F(4)$
7. $F(5)$

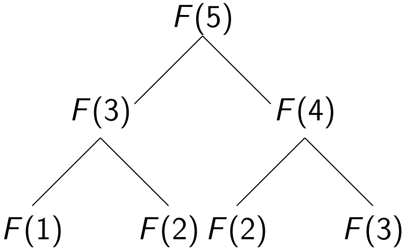


Fibonacci Number

Following the definition

1. **** $F(2)$
2. ** $F(3)$

6. ** $F(4)$
7. $F(5)$

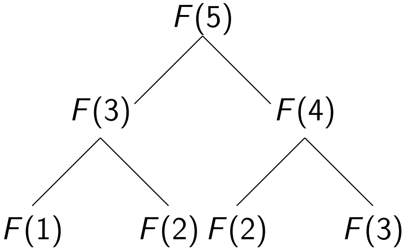


Fibonacci Number

Following the definition

1. **** $F(2)$
2. ** $F(3)$
3. ****** $F(2)$

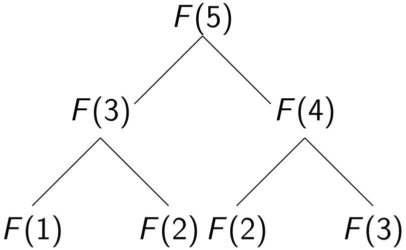
5. ****** $F(3)$
6. ** $F(4)$
7. $F(5)$



Fibonacci Number

Following the definition

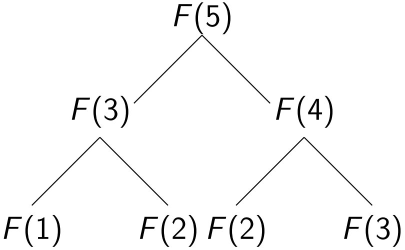
1. **** $F(2)$
2. ** $F(3)$
3. ****** $F(2)$
4. ******** $F(2)$
5. ****** $F(3)$
6. ** $F(4)$
7. $F(5)$



Fibonacci Number

Bottom up.

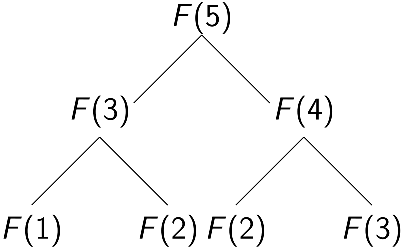
1. $F(2)$: $F(0)$, $F(1)$ by query



Fibonacci Number

Bottom up.

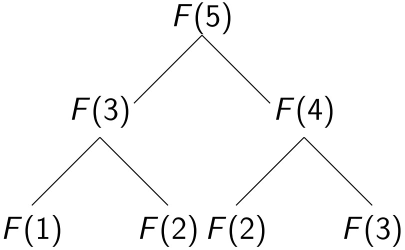
1. $F(2)$: $F(0)$, $F(1)$ by query
2. $F(3)$: $F(1)$, $F(2)$ by query



Fibonacci Number

Bottom up.

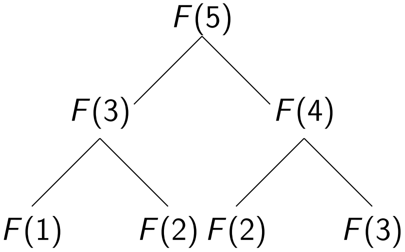
1. $F(2)$: $F(0)$, $F(1)$ by query
2. $F(3)$: $F(1)$, $F(2)$ by query
3. $F(4)$: $F(2)$, $F(3)$ by query



Fibonacci Number

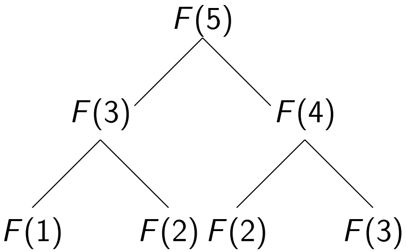
Bottom up.

1. $F(2)$: $F(0)$, $F(1)$ by query
2. $F(3)$: $F(1)$, $F(2)$ by query
3. $F(4)$: $F(2)$, $F(3)$ by query
4. $F(5)$: $F(3)$, $F(4)$ by query



Fibonacci Number

Following the definition, and using memory

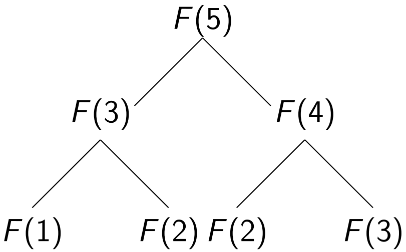


4. $F(5)$: add $F(3)$, $F(4)$

Fibonacci Number

Following the definition, and using memory

2. $F(3)$: add $F(1)$, $F(2)$
4. $F(5)$: add $F(3)$, $F(4)$

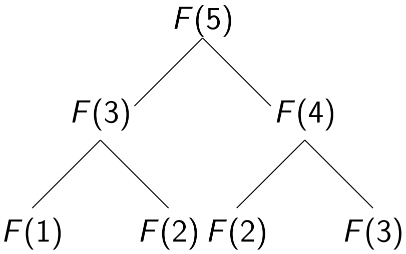


Fibonacci Number

Following the definition, and using memory

1. $F(2)$: add $F(0)$, $F(1)$
2. $F(3)$: add $F(1)$, $F(2)$

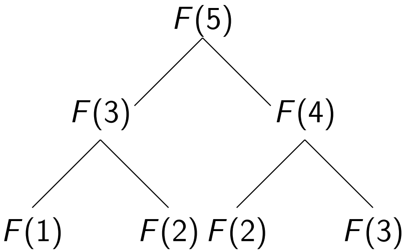
4. $F(5)$: add $F(3)$, $F(4)$



Fibonacci Number

Following the definition, and using memory

1. $F(2)$: add $F(0)$, $F(1)$
2. $F(3)$: add $F(1)$, $F(2)$
3. $F(4)$: add $F(2)$, $F(3)$
4. $F(5)$: add $F(3)$, $F(4)$



Which one is better?

- **BottomUp**: 4 additions
- **Using Memory**: 4 additions

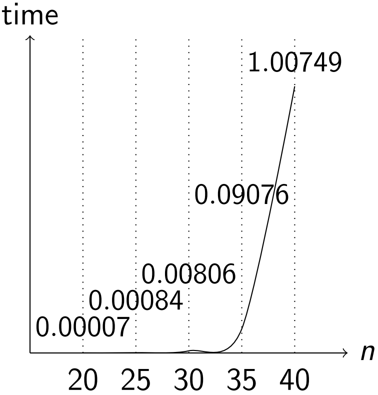
Fibonacci: Following Definition

Listing 1: fib_def.c

```
int fib_def(int n) {
  if (n == 0) return 0;
  if (n == 1) return 1;
  return fib_def(n - 2) + fib_def(n - 1);
}
int main(int argc, char ** argv) {
  clock_t start, end;
  start = clock();
  int f = fib_def(atoi(argv[1]));
  end = clock();
  printf("%d [%lf]\n", f, (double) (end - start) /
      CLOCKS_PER_SEC);
  return 0;
}
```

Fibonacci: Following Definition

```
>> ./fib_def 20
6765 [0.000070]
>> ./fib_def 25
75025 [0.000835]
>> ./fib_def 30
832040 [0.008057]
>> ./fib_def 35
9227465 [0.090762]
>> ./fib_def 40
102334155 [1.007486]
```



Fibonacci: Bottom Up v.s. Using Memory

Listing 2: fib_bu.c

```c
int f[10000];
void fib_bu(int n) {
   for (int i = 2; i <= n; i ++) {
    f[i] = f[i - 2] + f[i - 1];
  }
}
int main(int argc, char ** argv) {
   f[0] = 0;
   f[1] = 1;
   fib_bu(atoi(argv[1]));
}
```

Fibonacci: Bottom Up v.s. Using Memory

Listing 3: fib_mem.c

```
int f[10000];
void fib_mem(int n) {
  if (f[n] == -1) {
    if (f[n - 2] == -1) fib_mem(n - 2);
    if (f[n - 1] == -1) fib_mem(n - 1);
    f[n] = f[n - 2] + f[n - 1];
  }
}
int main(int argc, char ** argv) {
   for (int i = 0; i < 10000; i ++) f[i] = -1;
   f[0] = 0;
   f[1] = 1;
   fib_mem(atoi(argv[1]));
}
```

Fibonacci: Following Definition

```
>> ./fib_bu 1000
[0.000010]
>> ./fib_bu 2000
[0.000019]
>> ./fib_bu 3000
[0.000030]
>> ./fib_bu 4000
[0.000056]
>> ./fib_bu 5000
[0.000045]
```

```
>> ./fib_mem 1000
[0.000019]
>> ./fib_mem 2000
[0.000036]
>> ./fib_mem 3000
[0.000061]
>> ./fib_mem 4000
[0.000083]
>> ./fib_mem 5000
[0.000128]
```

Dynamic Programming

Consider the **optimization problems**

1. Define the optimal solution by recursion.
    - e.g. $F(n) = F(n-2) + F(n-1)$
2. Compute the values in the recursions
    - bottom up, **or**
    - using memory
3. Finish when the original problem is solved

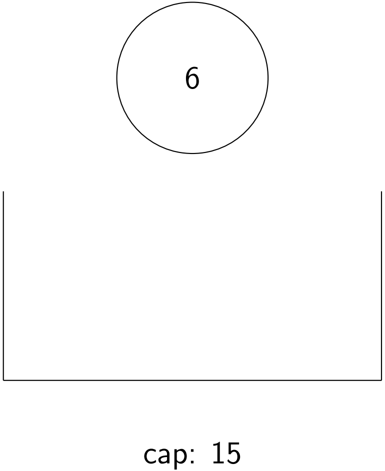
The Knapsack Problem

Knapsack

A container $\mathcal{K}$

- **capacity** $W > 0$

An item x of weight w_x



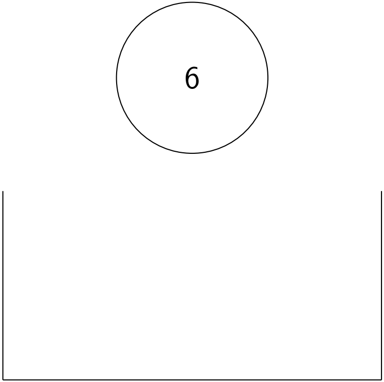
Knapsack

A container $\mathcal{K}$

- **capacity** $W > 0$

An item x of weight w_x

- x can be put into $\mathcal{K}$ iff. $w_x < W$



cap: 15

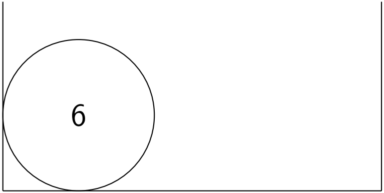
Knapsack

A container $\mathcal{K}$

- **capacity** $W > 0$

An item x of weight w_x

- x can be put into $\mathcal{K}$ iff. $w_x < W$
- the capacity is updated to $W - w_x$.



cap: 9

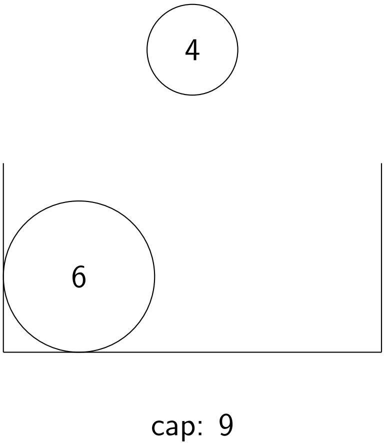
Knapsack

A container $\mathcal{K}$

- **capacity** $W > 0$

An item x of weight w_x

- x can be put into $\mathcal{K}$ iff. $w_x < W$
- the capacity is updated to $W - w_x$.



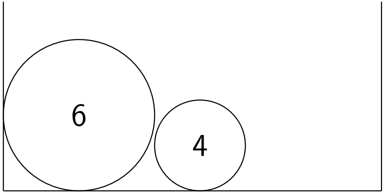
Knapsack

A container $\mathcal{K}$

- **capacity** $W > 0$

An item x of weight w_x

- x can be put into $\mathcal{K}$ iff. $w_x < W$
- the capacity is updated to $W - w_x$.



cap: 5

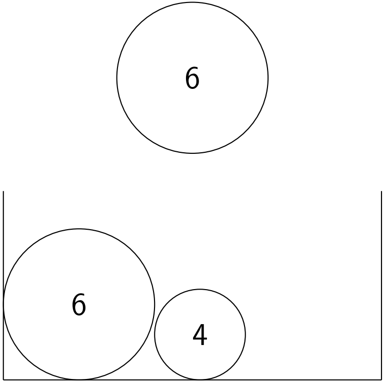
Knapsack

A container $\mathcal{K}$

- **capacity** $W > 0$

An item x of weight w_x

- x can be put into $\mathcal{K}$ iff. $w_x < W$
- the capacity is updated to $W - w_x$.



cap: 5

The Knapsack Problem: 0/1 Version

Given

- a container $\mathcal{K}$ of capacity W
- n items $\{x_0, x_1, \ldots, x_{n-1}\}$
  - integral weight $w_i > 0$
  - value $v_i > 0$

Fill the knapsack so as to maximize the total value.

The Knapsack Problem: Example

$\mathcal{K}$ of capacity $W = 11$

- $x_0 : w_0 = 1, \; v_0 = 1$
- $x_1 : w_1 = 2, \; v_1 = 6$
- $x_2 : w_2 = 5, \; v_2 = 18$
- $x_3 : w_3 = 6, \; v_3 = 22$
- $x_4 : w_4 = 7, \; v_4 = 28$

The Knapsack Problem: Example

Greedy: maximum v_i/w_i

- $v_0/w_0 = 1/1 = 1$
- $v_1/w_1 = 6/2 = 3$
- $v_2/w_2 = 18/5 = 3.6$
- $v_3/w_3 = 22/6 = 3.666$
- $v_4/w_4 = 28/7 = 4$

$\{x_4, x_1, x_0\}$, total value is 35.

Opt: $\{x_3, x_2\}$, 40.

$\mathcal{K}$ of capacity $W = 11$

- $x_0 : w_0 = 1,\ v_0 = 1$
- $x_1 : w_1 = 2,\ v_1 = 6$
- $x_2 : w_2 = 5,\ v_2 = 18$
- $x_3 : w_3 = 6,\ v_3 = 22$
- $x_4 : w_4 = 7,\ v_4 = 28$

Dynamic Programming: First Try

1. Define the optimal solution by recursion.
2. Compute the values in the recursions
3. Finish when the original problem is solved

The Knapsack Problem

1. Define $OPT(i)$ = max-subset-total-value of items $\{x_0, \ldots, x_{i-1}\}$, with the given capacity.
2. Cases

Dynamic Programming: First Try

1. Define the optimal solution by recursion.
2. Compute the values in the recursions
3. Finish when the original problem is solved

The Knapsack Problem

1. Define $OPT(i)$ = max-subset-total-value of items $\{x_0, \ldots, x_{i-1}\}$, with the given capacity.
2. Cases
   - if $OPT(i)$ does not select x_{i-1}, then $OPT(i) = OPT(i-1)$.

Dynamic Programming: First Try

1. Define the optimal solution by recursion.
2. Compute the values in the recursions
3. Finish when the original problem is solved

The Knapsack Problem

1. Define $OPT(i) =$ max-subset-total-value of items $\{x_0, \ldots, x_{i-1}\}$, with the given capacity.
2. Cases
   - if $OPT(i)$ does not select x_{i-1}, then $OPT(i) = OPT(i-1)$.
   - if $OPT(i)$ selects x_{i-1}, then $OPT(i)?OPT(i-1) + v_{i-1}$.

Dynamic Programming: First Try

1. Define the optimal solution by recursion.
2. Compute the values in the recursions
3. Finish when the original problem is solved

The Knapsack Problem

1. Define $OPT(i)$ = max-subset-total-value of items $\{x_0, \ldots, x_{i-1}\}$, with the given capacity.
2. Cases
   - if $OPT(i)$ does not select x_{i-1}, then $OPT(i) = OPT(i-1)$.
   - if $OPT(i)$ selects x_{i-1}, then $OPT(i)?OPT(i-1) + v_{i-1}$.
     - $W(i)$ = total weight of items in $OPT(i)$.

Dynamic Programming: First Try

1. Define the optimal solution by recursion.
2. Compute the values in the recursions
3. Finish when the original problem is solved

The Knapsack Problem

1. Define $OPT(i)$ = max-subset-total-value of items $\{x_0, \ldots, x_{i-1}\}$, with the given capacity.
2. Cases
    - if $OPT(i)$ does not select x_{i-1}, then $OPT(i) = OPT(i-1)$.
    - if $OPT(i)$ selects x_{i-1}, then $OPT(i)?OPT(i-1) + v_{i-1}$.
        - $W(i)$ = total weight of items in $OPT(i)$.
        - what about $W(i-1) + w_{i-1} > W$?

Dynamic Programming: Add a New Variable

The Knapsack Problem

1. Define $OPT(i, w)$ = max-subset-total-value of items $\{x_0, \ldots, x_{i-1}\}$, with weight limit w.

Dynamic Programming: Add a New Variable

The Knapsack Problem

1. Define $OPT(i, w)$ = max-subset-total-value of items $\{x_0, \ldots, x_{i-1}\}$, with weight limit w.
2. Cases for $OPT(i, w)$
   - if $OPT(i, w)$ does not select x_{i-1}, then

$$OPT(i, w) = OPT(i-1, w).$$

Dynamic Programming: Add a New Variable

The Knapsack Problem

1. Define $OPT(i, w)$ = max-subset-total-value of items $\{x_0, \ldots, x_{i-1}\}$, with weight limit w.
2. Cases for $OPT(i, w)$
   - if $OPT(i, w)$ does not select x_{i-1}, then
   $$OPT(i, w) = OPT(i-1, w).$$
   - if $OPT(i, w)$ selects x_{i-1}, then
   $$OPT(i, w) = OPT(i-1, w - w_{i-1}) + v_{i-1}.$$

Dynamic Programming: Add a New Variable

The Knapsack Problem

1. Define $OPT(i, w)$ = max-subset-total-value of items $\{x_0, \ldots, x_{i-1}\}$, with weight limit w.
2. Cases for $OPT(i, w)$
   - if $OPT(i, w)$ does not select x_{i-1}, then
   $$OPT(i, w) = OPT(i-1, w).$$
   - if $OPT(i, w)$ selects x_{i-1}, then
   $$OPT(i, w) = OPT(i-1, w - w_{i-1}) + v_{i-1}.$$
3. Solution: $OPT(n, W)$, where
   - n is the number of items
   - W is the capacity

Knapsack: Dynamic Programming

Algorithm: Knapsack_DP(W, v, w)

$n := |v|$;
for $w = 0$ *to* W **do** OPT[0, w] $= 0$;
for $i = 1$ *to* n **do** OPT[i, 0] $= 0$;
for $i = 1$ *to* n **do**
    for $w = 1$ *to* W **do**
        if $w_{i-1} > w$ **then** OPT(i, w) = OPT($i-1$, w);
        else OPT(i, w) = max{OPT($i-1$, w), OPT($i-1$, $w - w_{i-1}$) + v_{i-1}};
    end
end
return $OPT(n, W)$;

The Knapsack Problem: Example

w / i	0	1	2	3	4	5
11						
10						
9						
8						
7						
6						
5						
4						
3						
2						
1						
0						

$\mathcal{K}$ of capacity $W = 11$

- x_0 : $w_0 = 1$, $v_0 = 1$
- x_1 : $w_1 = 2$, $v_1 = 6$
- x_2 : $w_2 = 5$, $v_2 = 18$
- x_3 : $w_3 = 6$, $v_3 = 22$
- x_4 : $w_4 = 7$, $v_4 = 28$

$OPT(i, w)$

- if $w_{i-1} > w$, $OPT(i, w) = OPT(i-1, w)$
- if $w_{i-1} \leq w$, $OPT(i, w) =$ max of
  - $OPT(i-1, w)$, and
  - $OPT(i-1, w - w_{i-1}) + v_{i-1}$

The Knapsack Problem: Example

11	0					
10	0					
9	0					
8	0					
7	0					
6	0					
5	0					
4	0					
3	0					
2	0					
1	0					
0	0	0	0	0	0	0
w/i	0	1	2	3	4	5

$\mathcal{K}$ of capacity $W = 11$

- x_0 : $w_0 = 1$, $v_0 = 1$
- x_1 : $w_1 = 2$, $v_1 = 6$
- x_2 : $w_2 = 5$, $v_2 = 18$
- x_3 : $w_3 = 6$, $v_3 = 22$
- x_4 : $w_4 = 7$, $v_4 = 28$

$OPT(i, w)$

- if $w_{i-1} > w$, $OPT(i, w) = OPT(i-1, w)$
- if $w_{i-1} \leq w$, $OPT(i, w) = \max$ of
  - $OPT(i-1, w)$, and
  - $OPT(i-1, w - w_{i-1}) + v_{i-1}$

The Knapsack Problem: Example

w / i	0	1	2	3	4	5
11	0					
10	0					
9	0					
8	0					
7	0					
6	0					
5	0					
4	0					
3	0					
2	0					
1	0	1	1	1	1	1
0	0	0	0	0	0	0

$\mathcal{K}$ of capacity $W = 11$

- $x_0 : w_0 = 1, \ v_0 = 1$
- $x_1 : w_1 = 2, \ v_1 = 6$
- $x_2 : w_2 = 5, \ v_2 = 18$
- $x_3 : w_3 = 6, \ v_3 = 22$
- $x_4 : w_4 = 7, \ v_4 = 28$

$OPT(i, w)$

- if $w_{i-1} > w$, $OPT(i, w) = OPT(i-1, w)$
- if $w_{i-1} \leq w$, $OPT(i, w) = \max$ of
  - $OPT(i-1, w)$, and
  - $OPT(i-1, w - w_{i-1}) + v_{i-1}$

The Knapsack Problem: Example

11	0					
10	0					
9	0					
8	0					
7	0					
6	0					
5	0					
4	0					
3	0					
2	0	1	6	6	6	6
1	0	1	1	1	1	1
0	0	0	0	0	0	0
w/i	0	1	2	3	4	5

$\mathcal{K}$ of capacity $W = 11$

- $x_0 : w_0 = 1,\ v_0 = 1$
- $x_1 : w_1 = 2,\ v_1 = 6$
- $x_2 : w_2 = 5,\ v_2 = 18$
- $x_3 : w_3 = 6,\ v_3 = 22$
- $x_4 : w_4 = 7,\ v_4 = 28$

$OPT(i, w)$

- if $w_{i-1} > w$, $OPT(i, w) = OPT(i-1, w)$
- if $w_{i-1} \leq w$, $OPT(i, w) =$ max of
  - $OPT(i-1, w)$, and
  - $OPT(i-1, w - w_{i-1}) + v_{i-1}$

The Knapsack Problem: Example

11	0	1	7	25	40	40
10	0	1	7	25	29	35
9	0	1	7	25	29	34
8	0	1	7	25	28	29
7	0	1	7	24	24	28
6	0	1	7	19	22	22
5	0	1	7	18	18	18
4	0	1	7	7	7	7
3	0	1	7	7	7	7
2	0	1	6	6	6	6
1	0	1	1	1	1	1
0	0	0	0	0	0	0
w/i	0	1	2	3	4	5

$\mathcal{K}$ of capacity $W = 11$

- x_0 : $w_0 = 1$, $v_0 = 1$
- x_1 : $w_1 = 2$, $v_1 = 6$
- x_2 : $w_2 = 5$, $v_2 = 18$
- x_3 : $w_3 = 6$, $v_3 = 22$
- x_4 : $w_4 = 7$, $v_4 = 28$

$OPT(i, w)$

- if $w_{i-1} > w$, $OPT(i, w) = OPT(i-1, w)$
- if $w_{i-1} \leq w$, $OPT(i, w) = \max$ of
  - $OPT(i-1, w)$, and
  - $OPT(i-1, w - w_{i-1}) + v_{i-1}$

Knapsack: Dynamic Programming

Listing 4: knapsack_dp(...)

```
int knapsack_dp(int W, int * va, int * wa, int n) {
  int OPT[n + 1][W + 1];
  for (int w = 0; w <= W; w ++) OPT[0][w] = 0;
  for (int i = 1; i <= n; i ++) OPT[i][0] = 0;
  for (int i = 1; i <= n; i ++) {
    for (int w = 1; w <= W; w ++) {
        if (wa[i - 1] > w) OPT[i][w] = OPT[i - 1][w];
        else {
            if (OPT[i - 1][w] > OPT[i - 1][w - wa[i - 1]] + va[i - 1])
              OPT[i][w] = OPT[i - 1][w];
            else
              OPT[i][w] = OPT[i - 1][w - wa[i - 1]] + va[i - 1];
        }
    }
  }
  return OPT[n][W];
}
```

Knapsack: Dynamic Programming

Listing 5: knapsack_dp.c

```c
int main(int argc, char ** argv) {
  int W = 11;
  int n = 5;
  int va[5] = {1, 6, 18, 22, 28};
  int wa[5] = {1, 2, 5, 6, 7};
  int opt = knapsack_dp(W, va, wa, n);
  printf("OPT: %d\n", opt);
  return 0;
}
```

The Knapsack Problem: Example

w/i	0	1	2	3	4	5
11	0	1	7	25	40	40
10	0	1	7	25	29	35
9	0	1	7	25	29	34
8	0	1	7	25	28	29
7	0	1	7	24	24	28
6	0	1	7	19	22	22
5	0	1	7	18	18	18
4	0	1	7	7	7	7
3	0	1	7	7	7	7
2	0	1	6	6	6	6
1	0	1	1	1	1	1
0	0	0	0	0	0	0

```
>> ./knapsack_dp
0 1 7 25 40 40
0 1 7 25 29 35
0 1 7 25 29 34
0 1 7 25 28 29
0 1 7 24 24 28
0 1 7 19 22 22
0 1 7 18 18 18
0 1 7 7 7 7
0 1 7 7 7 7
0 1 6 6 6 6
0 1 1 1 1 1
0 0 0 0 0 0
OPT: 40
```

The Knapsack Problem

- Dynamic programming: $O(nW)$
  - OPT has $(n+1) \times (W+1)$ entries.
- Pseudo-polynomial: input size $\Theta(n) + \log W$
- Polynomial: if W is polynomial of n.

Special Case: When $v_i = w_i$

Will the problem be easier, if $v_i = w_i$?

It is still hard.

Subset-Sum Problem: Given a set of numbers, find the maximum(-size) subset of size k.

Algorithms with Sequences

Increasing Subsequence

Given a sequence of numbers

$$a_0, a_1, \ldots, a_{n-1}$$

an increasing subsequence is any subset of these numbers

$$a_{i_1}, a_{i_2}, \ldots, a_{i_k}$$

such that

- taken in order: $0 \leq i_1 < i_2 < \cdots < i_k \leq n-1$
- increasing: $a_{i_1} < a_{i_2} < \cdots < a_{i_k}$

Longest Increasing Subsequence

Problem: Given a sequence of numbers

$$a_0, a_1, \ldots, a_{n-1}$$

find the increasing subsequence of greatest length.

Example:

$$5 \quad \color{red}{2} \quad 8 \quad 6 \quad \color{red}{3} \quad \color{red}{6} \quad \color{red}{9} \quad 7$$

Define The Optimal Solution by Recursion

Problem: Given a sequence of numbers

$$a_0, a_1, \ldots, a_{n-1}$$

find the increasing subsequence of greatest length.

$L[i] =$ len of the longest increasing subsequence ending at a_{i-1}

Solution: $\max_i L[i]$

5 2 8 6 3 6 9 7

Define The Optimal Solution by Recursion

Problem: Given a sequence of numbers

$$a_0, a_1, \ldots, a_{n-1}$$

find the increasing subsequence of greatest length.

$L[i] =$ len of the longest increasing subsequence ending at a_{i-1}

Solution: $\max_i L[i]$

$$\underset{\substack{\uparrow \\ L[1]}}{\color{red}{5}} \quad 2 \quad 8 \quad 6 \quad 3 \quad 6 \quad 9 \quad 7$$

Define The Optimal Solution by Recursion

Problem: Given a sequence of numbers

$$a_0, a_1, \ldots, a_{n-1}$$

find the increasing subsequence of greatest length.

$L[i] =$ len of the longest increasing subsequence ending at a_{i-1}

Solution: $\max_i L[i]$

$$\begin{array}{cccccccc} 5 & \underset{\substack{\uparrow \\ L[2]}}{2} & 8 & 6 & 3 & 6 & 9 & 7 \end{array}$$

Define The Optimal Solution by Recursion

Problem: Given a sequence of numbers

$$a_0, a_1, \ldots, a_{n-1}$$

find the increasing subsequence of greatest length.

$L[i]$ = len of the longest increasing subsequence ending at a_{i-1}

Solution: $\max_i L[i]$

$$\begin{array}{cccccccc} 5 & 2 & \underset{\underset{L[3]}{\uparrow}}{8} & 6 & 3 & 6 & 9 & 7 \end{array}$$

Define The Optimal Solution by Recursion

Problem: Given a sequence of numbers

$$a_0, a_1, \ldots, a_{n-1}$$

find the increasing subsequence of greatest length.

$L[i] =$ len of the longest increasing subsequence ending at a_{i-1}

Solution: $\max_i L[i]$

$$\begin{array}{cccccccc} 5 & \color{red}{2} & 8 & 6 & \color{red}{3} & \color{red}{6} & \color{red}{9} & 7 \\ & & & & & & \uparrow & \\ & & & & & & L[7] & \end{array}$$

Calculate $L[i]$

Assuming we have the values $L[j]$ for all $j < i$, what is the value of $L[i]$?

$$\underbrace{a_? \qquad a_? \qquad a_? \qquad a_{k-1} \quad a_{i-1}}_{L[i]}$$

Calculate $L[i]$

Assuming we have the values $L[j]$ for all $j < i$, what is the value of $L[i]$?

$$\underbrace{a_? \qquad a_? \qquad a_? \qquad a_{k-1}}_{L[k]} \quad a_{i-1}$$

Calculate $L[i]$

Assuming we have the values $L[j]$ for all $j < i$, what is the value of $L[i]$?

$$\underbrace{a_? \quad a_? \quad a_? \quad a_{k-1}}_{L[k]} \quad a_{i-1}$$

$$L[i] = 1 + \max_k \{ L[k] \mid k < i \text{ and } a_{k-1} < a_{i-1} \}$$

LIS: Dynamic Prgoramming

Listing 6: lis_dp(...)

```
int lis_dp(int * a, int n) {
  int m = 0;
  int L[n + 1];
  L[0] = 0;
  for (int i = 2; i <= n; i ++) {
    L[i] = 1;
    for (int j = 1; j < i; j ++) {
      if (a[j - 1] < a[i - 1] && L[j] + 1 > L[i])
        L[i] = L[j] + 1;
    }
    if (L[i] > m) m = L[i];
  }
  return m;
}
```

LIS: Dynamic Prgoramming

Listing 7: lis_dp.c

```c
int main(int argc, char ** argv) {
  int n = 8;
  int a[8] = {5, 2, 8, 6, 3, 6, 9, 7};
  int l = lis_dp(a, n);
  printf("LIS: %d\n", l);
  return 0;
}
```

```
>> gcc lis_dp.c -o lis_dp
>> ./lis_dp
0 0 1 2 2 2 3 4 4
LIS: 4
```

LIS: Dynamic Prgoramming

Time Cost: $O(n^2)$

- first **for**-loop: n
- second **for**-loop: i for $2 \leq i \leq n$

Common Subsequence

Given two sequences

$$x_0, x_1, \ldots, x_{n-1}$$

$$y_0, y_1, \ldots, y_{m-1}$$

a common subsequence is a subset of numbers

$$x_{i_0} = y_{j_0}, x_{i_1} = y_{j_1}, \ldots, x_{i_k} = y_{j_k}$$

such that

- $0 \leq i_0 < i_1 < \cdots < i_k \leq n - 1$
- $0 \leq j_0 < j_1 < \cdots < j_k \leq m - 1$

Longest Common Subsequence

Problem: Given two sequences

$$x_0, x_1, \ldots, x_{n-1}$$

$$y_0, y_1, \ldots, y_{m-1}$$

find the common subsequence of the greatest length.

Example:

$$A \quad B \quad C \quad B \quad D \quad A \quad B$$

$$B \quad D \quad C \quad A \quad B \quad A$$

Longest Common Subsequence

Problem: Given two sequences

$$x_0, x_1, \ldots, x_{n-1}$$

$$y_0, y_1, \ldots, y_{m-1}$$

find the common subsequence of the greatest length.

Example:

$$\begin{array}{ccccccc} A & \color{red}{B} & C & B & D & A & B \\ \color{red}{B} & D & C & A & B & A & \end{array}$$

Longest Common Subsequence

Problem: Given two sequences

$$x_0, x_1, \ldots, x_{n-1}$$

$$y_0, y_1, \ldots, y_{m-1}$$

find the common subsequence of the greatest length.

Example:

$$A \quad B \quad C \quad B \quad D \quad A \quad B$$

$$B \quad D \quad C \quad A \quad B \quad A$$

Longest Common Subsequence

Problem: Given two sequences

$$x_0, x_1, \ldots, x_{n-1}$$

$$y_0, y_1, \ldots, y_{m-1}$$

find the common subsequence of the greatest length.

Example:

$$\begin{array}{ccccccc} A & \color{red}{B} & \color{red}{C} & \color{red}{B} & D & A & B \\ \color{red}{B} & D & \color{red}{C} & A & \color{red}{B} & A & \end{array}$$

Longest Common Subsequence

Problem: Given two sequences

$$x_0, x_1, \ldots, x_{n-1}$$

$$y_0, y_1, \ldots, y_{m-1}$$

find the common subsequence of the greatest length.

Example:

$$A \quad \color{red}{B} \quad \color{red}{C} \quad B \quad D \quad \color{red}{A} \quad B$$

$$\color{red}{B} \quad D \quad \color{red}{C} \quad \color{red}{A} \quad B \quad A$$

Longest Common Subsequence

Problem: Given two sequences

$$x_0, x_1, \ldots, x_{n-1}$$

$$y_0, y_1, \ldots, y_{m-1}$$

find the common subsequence of the greatest length.

Example:

$$A \quad B \quad C \quad B \quad D \quad A \quad B$$

$$B \quad D \quad C \quad A \quad B \quad A$$

Longest Common Subsequence

Problem: Given two sequences

$$x_0, x_1, \ldots, x_{n-1}$$

$$y_0, y_1, \ldots, y_{m-1}$$

find the common subsequence of the greatest length.

Example:

$$A \quad \color{red}{B} \quad C \quad B \quad \color{red}{D} \quad \color{red}{A} \quad \color{red}{B}$$

$$\color{red}{B} \quad \color{red}{D} \quad C \quad \color{red}{A} \quad \color{red}{B} \quad A$$

LCS: Brute Force

In

$$x_0, x_1, \ldots, x_{n-1}$$

for each subsequence

$$x_{i_0}, x_{i_1}, \ldots, x_{i_k}$$

check if it is also a subsequence of

$$y_0, y_1, \ldots, y_{m-1}$$

Time cost: $O(m2^n)$

- there are 2^n subsequences to check
- $O(m)$ for one check

LCS: Dynamic Programming

Problem: Given two sequences

$$x_0, x_1, \ldots, x_{n-1}$$

$$y_0, y_1, \ldots, y_{m-1}$$

find the common subsequence of the greatest length.

$C[i, j] =$ len of the longest common subsequence of

$$x_0, x_1, \ldots, x_{i-1}$$

$$y_0, y_1, \ldots, y_{j-1}$$

Solution: $C[n, m]$

Calculate $C[i, j]$

Assuming we have the values $C[i, j]$ for all $i' + j' < i + j$, what is the value of $C[i, j]$?

$$
\begin{array}{ccccc|c|}
\cline{6-6}
x_? & x_? & x_? & x_? & & x_{i-1} \\
y_? & y_? & y_? & y_? & & y_{j-1} \\
\cline{6-6}
\end{array}
$$

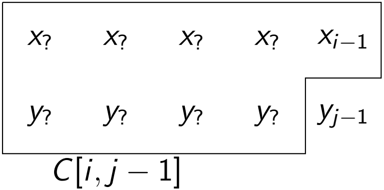
Calculate $C[i, j]$

Assuming we have the values $C[i, j]$ for all $i' + j' < i + j$, what is the value of $C[i, j]$?

$$\underset{C[i-1,j-1]}{\boxed{\begin{array}{cccc} x_? & x_? & x_? & x_? \\ y_? & y_? & y_? & y_? \end{array}}} \quad \underset{x_{i-1}=y_{j-1}}{\boxed{\begin{array}{c} x_{i-1} \\ y_{j-1} \end{array}}}$$

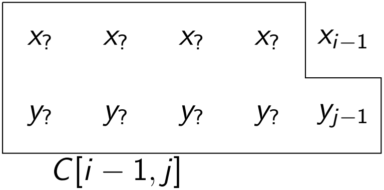
Calculate $C[i, j]$

Assuming we have the values $C[i, j]$ for all $i' + j' < i + j$, what is the value of $C[i, j]$?



Calculate $C[i, j]$

Assuming we have the values $C[i, j]$ for all $i' + j' < i + j$, what is the value of $C[i, j]$?



Calculate $C[i, j]$

Assuming we have the values $C[i, j]$ for all $i' + j' < i + j$, what is the value of $C[i, j]$?

$$
\begin{array}{ccccc}
x_? & x_? & x_? & x_? & \boxed{\begin{array}{c} x_{i-1} \\ \\ y_{j-1} \end{array}} \\
y_? & y_? & y_? & y_? &
\end{array}
$$

$$
C[i, j] = \begin{cases} C[i-1, j-1] + 1 & \text{if } x_{i-1} = y_{j-1} \\ \max\{C[i-1, j], C[i, j-1]\} & \text{otherwise} \end{cases}
$$

LCS: Dynamic Programming

Time Cost: $O(nm)$

- C: $(n+1) \times (m+1)$ entries
- constant time to calculate the value for one entry

Sequence Alignment

Given two sequences

$$x_0, x_1, \ldots, x_{n-1}$$

$$y_0, y_1, \ldots, y_{m-1}$$

an alignment is just a way to write the letters column by column

$$\begin{array}{c:c:c:c:c:c} - & x_0 & x_1 & \cdots & x_{n-1} & - \\ y_0 & y_1 & - & \cdots & - & y_{m-1} \end{array}$$

In the alignment, it is allowed to have **mismatches** and **gaps**.

Similarity between Two Sequences

Mismatch

w	o	r	d
w	o	o	d

Gap (or **Unmatched**)

w	o	r	-	d
w	o	r	l	d

Edit Distance

Penalty

- mismatch: α
- gap: δ

Problem: Given two sequences

$$x_0, x_1, \ldots, x_{n-1}$$

$$y_0, y_1, \ldots, y_{m-1}$$

find the alignment of the minimum total penalty.

Edit Distance: Example

w	o	r	d
w	o	o	d

edit distance: α (1 mismatch)

Edit Distance: Example

w	o	r	-	d
w	o	-	o	d

edit distance: 2δ (2 gaps)

Sequence Alignment

Problem: Given two sequences

$$x_0, x_1, \ldots, x_{n-1}$$

$$y_0, y_1, \ldots, y_{m-1}$$

find the alignment of the minimum total penalty.

$P[i, j] =$ min penalty of the aligning

$$x_0, x_1, \ldots, x_{i-1}$$

$$y_0, y_1, \ldots, y_{j-1}$$

Solution: $P[n, m]$

Calculate $P[i, j]$

Assuming we have the values $P[i', j']$ for all $i' + j' < i + j$, what is the value of $P[i, j]$?

$$
\begin{array}{cccc|c|}
\cline{5-5}
x_? & x_? & x_? & x_? & x_{i-1} \\
 & & & & \\
y_? & y_? & y_? & y_? & y_{j-1} \\
\cline{5-5}
\end{array}
$$

Calculate $P[i, j]$

Assuming we have the values $P[i', j']$ for all $i' + j' < i + j$, what is the value of $P[i, j]$?

$$\underset{P[i-1, j-1]}{\boxed{\begin{array}{cccc} x_? & x_? & x_? & x_? \\ & & & \\ y_? & y_? & y_? & y_? \end{array}}} \quad \boxed{\begin{array}{c} x_{i-1} \\ | \\ y_{j-1} \end{array}}$$

Calculate $P[i,j]$

Assuming we have the values $P[i',j']$ for all $i'+j' < i+j$, what is the value of $P[i,j]$?

$$\begin{array}{c}
\boxed{\begin{array}{ccccc} x_? & x_? & x_? & x_? & x_{i-1} \\ \\ y_? & y_? & y_? & y_? & y_{j-2} \end{array}}
\begin{array}{c} - \\ \\ y_{j-1} \end{array} \\
P[i,j-1]
\end{array}$$

Calculate $P[i, j]$

Assuming we have the values $P[i', j']$ for all $i' + j' < i + j$, what is the value of $P[i, j]$?

$$
\begin{array}{|ccccc|c}
\hline
x_? & x_? & x_? & x_? & x_{i-2} & x_{i-1} \\
y_? & y_? & y_? & y_? & y_{j-1} & - \\
\hline
\end{array}
$$

$$P[i-1, j]$$

Calculate $P[i, j]$

Assuming we have the values $P[i', j']$ for all $i' + j' < i + j$, what is the value of $P[i, j]$?

$$
\begin{array}{ccccc}
x_? & x_? & x_? & x_? & \boxed{\begin{array}{c} x_{i-1} \\ \\ y_{j-1} \end{array}} \\
y_? & y_? & y_? & y_? &
\end{array}
$$

$P[i, j] =$ min of

- align x_{i-1} with y_{j-1}: $c + P[i-1, j-1]$
  - if $x_{i-1} = y_{j-1}$, $c = 0$
  - if $x_{i-1} \neq y_{j-1}$, $c = \alpha$
- x_{i-1} is unmatched: $\delta + P[i-1, j]$
- y_{j-1} is unmatched: $\delta + P[i, j-1]$

Sequence Alignment: Dynamic Programming

Time Cost: $O(nm)$

- P: $(n+1) \times (m+1)$ entries
- constant time to calculate the value for one entry

Matrix Chain Multiplication

Matrix Multiplication

Consider a matrix A of size $n_1 \times m$, and a matrix B of size $m \times n_2$, the multiplication $C = A \times B$ is a matrix of size $n_1 \times n_2$, and

$$c_{i,j} = \sum_{k=0}^{m-1} a_{i,k} b_{k,j}$$

It requires $n_1 \times m \times n_2$ operations in total.

Matrix Chain Multiplication

Multiply three matrices $A \times B \times C$

- $A : 50 \times 20$
- $B : 20 \times 1$
- $C : 1 \times 10$

Number of the operations

- $(A \times B) \times C$ requires 1500 operations
  - $50 \times 20 \times 1 = 1000$
  - $50 \times 1 \times 10 = 500$
- $A \times (B \times C)$ requires 10200 operations
  - $20 \times 1 \times 10 = 200$
  - $50 \times 20 \times 10 = 10000$

Matrix Chain Multiplication

Problem: Given a chain of matrix multiplications, determine the optimal calculation order, i.e. the order that requires the minimum number of operations.

$$M_0 \times M_1 \times \cdots \times M_{n-1}$$

- M_i is of size $m_i \times m_{i+1}$

Matrix Chain Multiplication

$$M_0 \times M_1 \times \cdots \times M_{n-1}$$

$C[i,j] =$ min number of operations of

$$M_i \times M_1 \times \cdots \times M_j$$

Solution: $C[0, n-1]$

$$C[i,j] = \min_k (C[i, k-1] + C[k,j] + m_i \times m_k \times m_{j+1})$$

Matrix Chain Multiplication

Time Cost: $O(n^3)$

- C has n^2 entries
- $O(n)$ cases to check for each entry

Matrix Chain Multiplication: Recursion

$$M_0 \times M_1 \times \cdots \times M_{n-1}$$

Define function $COST([M_0, \ldots, M_{n-1}])$, to return the minimum number of operations.

$COST([M_0, \ldots, M_{n-1}]) =$ min of the sum of

- $COST([M_0, \ldots, M_{k-1}])$, and
- $COST([M_k, \ldots, M_{n-1}])$, and
- $m_0 \times m_k \times m_n$

over all possible k.

Matrix Chain Multiplication: Recursion

Algorithm: COST$([M_0, M_1, \ldots, M_{n-1}])$

if $n = 0$ **then return** 0;
if $n = 1$ **then return** $m_0 \times m_{n-1} \times m_n$;
$max = 0$;
for $k = 1$ *to* $n - 1$ **do**
    $cost = COST([M_0, \ldots, M_{k-1}]) +$
    $COST([M_k, \ldots, M_{n-1}]) + m_0 \times m_k \times m_n$;
    if $cost > max$ **then** $max = cost$;
end
return max;

Matrix Chain Multiplication: Recursion

Listing 8: cost(...)

```
int cost(int * ma, int n) {
  int m = 0, c = 0;
  if (n == 2) return 0;
  if (n == 3) return ma[0] * ma[1] * ma[2];
  m = ma[0] * ma[n - 1] * ma[n];
  for (int k = 1; k < n; k ++) {
    c = cost(& ma[0], k) + cost(& ma[k], n - k) +
        ma[0] * ma[k] * ma[n];
    if (c < m) m = c;
  }
  return m;
}
```

Matrix Chain Multiplication: Using Memory

Listing 9: cost_mem(...)

```
int C[4][4]; // C[i][j] is initialized to -1
...
if (j == i) C[i][i] = 0;
else {
  C[i][j] = ma[i] * ma[j] * ma[j + 1];
  for (int k = i + 1; k <= j; k ++) {
    if (C[i][k - 1] == -1) cost_mem(i, k - 1);
    if (C[k][j] == -1) cost_mem(k, j);
    c = C[i][k - 1] + C[k][j] + ma[i] * ma[k] * ma[j
        + 1];
    if (c < C[i][j]) C[i][j] = c;
  }
}
```

Matrix Chain Multiplication: Using Memory

Listing 10: cost_mem.c

```c
int m[5] = {50, 20, 1, 10, 100};
```

```
>> gcc -c cost_mem.c -o cost_mem
>> ./cost_mem
i/j     0     1     2     3
0 :     0  1000   500  7000
1 :    -1     0   200  3000
2 :    -1    -1     0  1000
3 :    -1    -1    -1     0
```

Matrix Chain Multiplication: Dynamic Programming

Time Cost: $O(n^3)$ for the calculation of C

- C has $n \times n$ entries
- $O(n)$ to calculate the value for one entry

How many times have cost_mem been called?

Optimal Binary Search Tree

Binary Search

a c e f h l w

Query: is 'd' in the list?

Binary Search

1

a c e f h l w

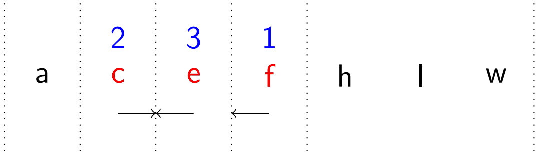
Query: is 'd' in the list?

Binary Search

	2		1			
a	c	e	f	h	l	w
	$\longrightarrow$		$\longleftarrow$			

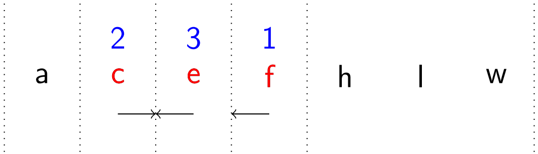
Query: is ‘d’ in the list?

Binary Search



Query: is ‘d’ in the list?

Binary Search



NO

Query: is ‘d’ in the list?

Binary Search Tree

Keys:

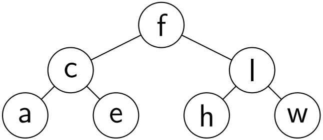
$$k_0, k_1, \ldots, k_{n-1}$$

Query: is k in the list?

Expected:

- if $\exists k_i,\ s.t.\ k = k_i$, then answer YES.
- if $\forall k_i,\ s.t.\ k \neq k_i$, then answer NO.

Assumption: $k_0, k_1, \ldots, k_{n-1}$ are sorted in ascending order.



is ‘d’ in the list?

Binary Search Tree

Keys:

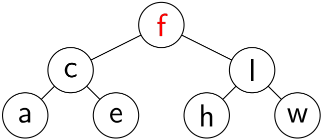
$$k_0, k_1, \ldots, k_{n-1}$$

Query: is k in the list?

Expected:

- if $\exists k_i,\ s.t.\ k = k_i$, then answer YES.
- if $\forall k_i,\ s.t.\ k \neq k_i$, then answer NO.

Assumption: $k_0, k_1, \ldots, k_{n-1}$ are sorted in ascending order.



is 'd' in the list?

Binary Search Tree

Keys:

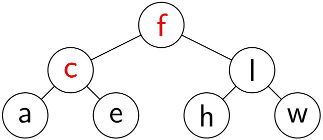
$$k_0, k_1, \ldots, k_{n-1}$$

Query: is k in the list?

Expected:

- if $\exists k_i,\ s.t.\ k = k_i$, then answer YES.
- if $\forall k_i,\ s.t.\ k \neq k_i$, then answer NO.

Assumption: $k_0, k_1, \ldots, k_{n-1}$ are sorted in ascending order.



is ‘d’ in the list?

Binary Search Tree

Keys:

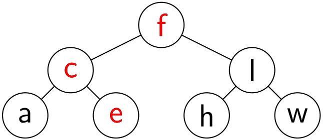
$$k_0, k_1, \ldots, k_{n-1}$$

Query: is k in the list?

Expected:

- if $\exists k_i,\ s.t.\ k = k_i$, then answer YES.
- if $\forall k_i,\ s.t.\ k \neq k_i$, then answer NO.

Assumption: $k_0, k_1, \ldots, k_{n-1}$ are sorted in ascending order.



is ‘d’ in the list?

Binary Search Tree

Keys:

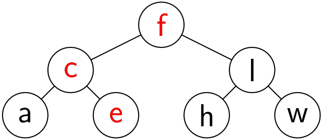
$$k_0, k_1, \dots, k_{n-1}$$

Query: is k in the list?

Expected:

- if $\exists k_i,\ s.t.\ k = k_i$, then answer YES.
- if $\forall k_i,\ s.t.\ k \neq k_i$, then answer NO.

Assumption: $k_0, k_1, \dots, k_{n-1}$ are sorted in ascending order.



is 'd' in the list?

Cost:

- YES: $\#$ visited nodes
- NO: $1 + \#$ visited nodes

Binary Search Tree

Keys:

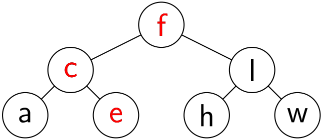
$$k_0, k_1, \ldots, k_{n-1}$$

Query: is k in the list?

Expected:

- if $\exists k_i,\ s.t.\ k = k_i$, then answer YES.
- if $\forall k_i,\ s.t.\ k \neq k_i$, then answer NO.

Assumption: $k_0, k_1, \ldots, k_{n-1}$ are sorted in ascending order.

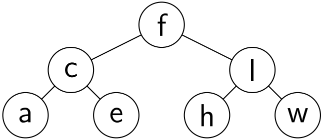


'd' : cost 4

Cost:

- YES: # visited nodes
- NO: 1+ # visited nodes

Search Cost

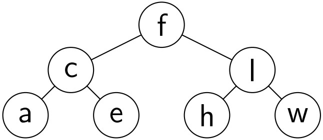


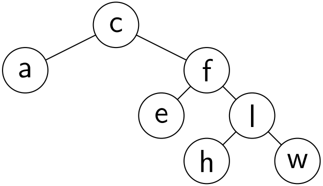
Costs for {'b', 'e', 'a', 'c', 'l'}

'b' : 4
'e' : 3
'a' : 3
'c' : 2
'l' : 2

Total: 14

Search Cost





Costs for {'b', 'e', 'a', 'c', 'l'}

'b' : 4
'e' : 3
'a' : 3
'c' : 2
'l' : 2

Total: 14

Costs for {'b', 'e', 'a', 'c', 'l'}

'b' : 3
'e' : 3
'a' : 2
'c' : 1
'l' : 3

Total: 12

Query Frequency

Given n keys

$$
\begin{array}{cccccccccc}
 & k_0 & & k_1 & & \cdots & & k_{n-1} & \\
q_0 & p_0 & q_1 & p_1 & q_2 & \cdots & q_{n-1} & p_{n-1} & q_n
\end{array}
$$

There are totally 100 queries, in which

- $100 \times p_i$ queries for k_i
- $100 \times q_i$ queries for keys between k_{i-1} and k_i
- $100 \times q_0$ queries for keys smaller than k_0
- $100 \times q_n$ queries for keys larger than k_{n-1}

Thus

$$\sum_{i=0}^{n-1} p_i + \sum_{j=0}^{n} q_j = 1$$

Optimal Binary Search Tree

Problem: Given keys

$$k_0 \qquad k_1 \qquad \cdots \qquad k_{n-1}$$

and the frequencies

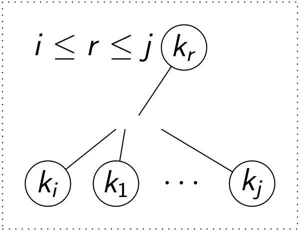
$$\begin{array}{ccccccccc} & p_0 & & p_1 & & \cdots & & p_{n-1} & \\ q_0 & & q_1 & & q_2 & \cdots & q_{n-1} & & q_n \end{array}$$

construct the binary search tree that minimizes the search cost.

Sum of Queries

$w[i,j] = \sum_{l=i}^{j} p_l + \sum_{l=i}^{j+1} q_l.$

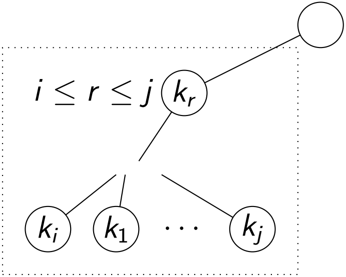
$$\begin{array}{ccccccccc} & k_i & & k_{i+1} & & \cdots & & k_j & \\ \color{blue}{q_i} & \color{red}{p_i} & \color{blue}{q_{i+1}} & \color{red}{p_{i+1}} & \color{blue}{q_{i+2}} & \cdots & \color{blue}{q_j} & \color{red}{p_j} & \color{blue}{q_{j+1}} \end{array}$$



Sum of Queries

$w[i,j] = \sum_{l=i}^{j} p_l + \sum_{l=i}^{j+1} q_l$.

$$
\begin{array}{ccccccccc}
 & k_i & & k_{i+1} & & \cdots & & k_j & \\
\color{blue}{q_i} & \color{red}{p_i} & \color{blue}{q_{i+1}} & \color{red}{p_{i+1}} & \color{blue}{q_{i+2}} & \cdots & \color{blue}{q_j} & \color{red}{p_j} & \color{blue}{q_{j+1}}
\end{array}
$$

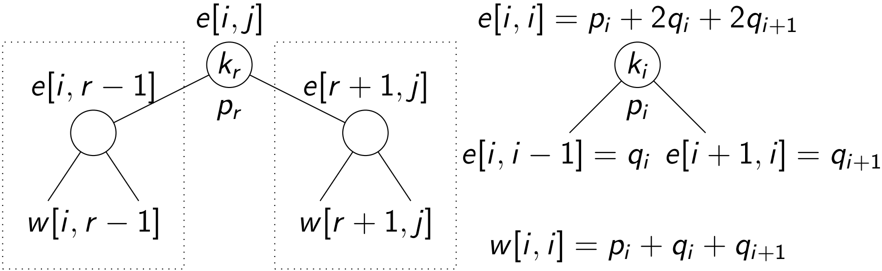


layer: $+1$

cost: $+w[i,j]$

Cost in The Subtree

$$
\begin{aligned}
e[i,j] &= p_r \\
&\quad + e[i,r-1] + w[i,r-1] \\
&\quad + e[r+1,j] + w[r+1,j] \\
&= e[i,r-1] + e[r+1,j] + w[i,r-1] + p_r + w[r+1,j] \\
&= e[i,r-1] + e[r+1,j] + w[i,j]
\end{aligned}
$$



Calculation of $e[i, j]$

Assuming we have the values $e[i', j']$ for all $j' - i' < j - i$, what is the value of $e[i, j]$?

$$e[i,j] = \begin{cases} q_i & \text{if } j = i - 1 \\ \min_{i \le r \le j}(e[i, r-1] + e[r+1, j] + w[i, j]) & \text{if } i \le j \end{cases}$$

$O(n^3)$

THANK YOU