



- Graphs
 - Breadth-First Search(BFS)
 - Bipartite Graph
 - Depth-First Search (DFS)
 - Articulation Points
 - Node
- Greedy Algorithm
 - Huffman Algorithm
- Divide and conquer
 - Merge sort
 - Closest pair algorithm
 - Recurrence
 - Karatsuba multiplication
 - Master Recurrence
- Dynamic Programming
 - Intro
 - Schedule
 - Weighted Interval Scheduling
 - Knapsack Algorithm
 - RNA

Week 1 Class 2

Runtime

Upper Bounds:

$f(n)$ is $O(g(n))$ if there is a constant $c > 0$ so that $f(n)$ is eventually always $\leq cg(n)$

Lower Bounds:

$f(n)$ is $\Omega(g(n))$ if there is a constant $c > 0$ so that $f(n)$ is eventually always $\geq cg(n)$

Both:

$f(n)$ is $\Theta(g(n))$ if there are constants $c_1, c_2 > 0$ so that eventually always $c_1g(n) \leq f(n) \leq c_2g(n)$

Week 1 Class 3

Logarithms

Change-of-base formula:

$$\log_a(x) = \frac{\log_b(x)}{\log_b(a)}$$

$$\log_b(a) * \log_a(x) = \log_b(x)$$

For all $x > 0$, (no matter how small) $\log n = O(n^x)$

+ Week 2 Class 1

Graphs

Undirected Graph:

Directed Graph:

Sparse graphs: $m \ll n^2$

+ Week 2 Class 2

Breadth-First Search(BFS)

level by level search

runtime:

- $O(n^2)$ in general.

Assume edge-list
representation

BFS: Analysis, I

$O(n)$ Global initialization: mark all vertices "undiscovered"
+ BFS(s)
 $O(1)$ mark s "discovered"
+
 $O(n)$
 \times
 $O(n)$
= $O(n^2)$

```
queue = { s }
while queue not empty
    u = remove_first(queue)
    for each edge {u,x}
        if (x is undiscovered)
            mark x discovered
            append x on queue
mark u fully explored
```

Simple analysis:
2 nested loops.
Get worst-case
number of
iterations of each;
multiply.

34

- $\$(n+m)$ if sparse graphs
- Application: Shortest Paths

Bipartite Graph

odd cycle is,

Depth-First Search (DFS)

go one deep first and go back

Articulation Points

DFS To Find Articulation Points

Global initialization: $\text{dfscounter} = 0$; $v.\text{dfs\#} = -1$ for all v .

$\text{DFS}(v)$:

```
v.\text{dfs\#} = \text{dfscounter}++  
v.\text{low} = v.\text{dfs\#}           // initialization  
for each edge {v,x}  
  if (x.\text{dfs\#} == -1)      // x is undiscovered  
    \text{DFS}(x)  
    v.\text{low} = \min(v.\text{low}, x.\text{low})  
    if (x.\text{low} >= v.\text{dfs\#})  
      print "v is art. pt., separating x"  
  else if (x is not v's parent)  
    v.\text{low} = \min(v.\text{low}, x.\text{dfs\#})
```

Except for root. Why?

What if G is not connected?

Equiv: "if({v,x} is a back edge)"
Why?

Node

- Leaf nodes(Leaves): The bottom nodes in every way of the tree. Node without children
- Internal nodes: Node have at least one child
- Root: Node without parent

+ Class 4.1

Greedy Algorithm

+ Class 4.3

Huffman Algorithm

use the less bits to represent the every elements (the data has probability)

+ Class 5.2

Divide and conquer

The basic idea is divide one full size probelem to two half

Merge sort

sort the array

Closest pair algorithm

```
Closest-Pair( $p_1, \dots, p_n$ ) {
    if( $n \leq ??$ ) return ??  
  
    Compute separation line  $L$  such that half the points  
    are on one side and half on the other side.  
  
     $\delta_1 = \text{Closest-Pair(left half)}$   
     $\delta_2 = \text{Closest-Pair(right half)}$   
     $\delta = \min(\delta_1, \delta_2)$   
  
    Delete all points further than  $\delta$  from separation line  $L$   
  
    Sort remaining points  $p[1] \dots p[m]$  by y-coordinate.  
  
    for  $i = 1 \dots m$ 
         $k = 1$ 
        while  $i+k \leq m \ \&\& \ p[i+k].y < p[i].y + \delta$ 
             $\delta = \min(\delta, \text{distance between } p[i] \text{ and } p[i+k]);$ 
             $k++;$   
  
    return  $\delta$ .
}
```

+ Class 5.3

$$C(n) \leq \begin{cases} 0 & n=1 \\ 2C(n/2) + kn \log n & n>1 \end{cases} \Rightarrow C(n) = O(n \log^2 n)$$

for some constant k

Sort center strip

$C(n)$ is number of comparisons

Recurrence

Merge sort

+ Class 6.1 2/7

Karatsuba multiplication

To multiply two n -bit integers:

Add two pairs of $\frac{1}{2}n$ bit integers.

Multiply **three** pairs of $\frac{1}{2}n$ -bit integers.

Add, subtract, and shift n -bit integers to obtain result.

$$\begin{aligned}x &= 2^{n/2} \cdot x_1 + x_0 \\y &= 2^{n/2} \cdot y_1 + y_0 \\xy &= 2^n \cdot x_1 y_1 + 2^{n/2} \cdot (x_1 y_0 + x_0 y_1) + x_0 y_0 \\&= 2^n \cdot x_1 y_1 + 2^{n/2} \cdot ((x_1 + x_0)(y_1 + y_0) - x_1 y_1 - x_0 y_0) + x_0 y_0\end{aligned}$$

A B A C C

Theorem. [Karatsuba-Ofman, 1962] Can multiply two n -digit integers in $O(n^{1.585})$ bit operations.

$$T(n) \leq \underbrace{T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + T(1 + \lceil n/2 \rceil)}_{\text{recursive calls}} + \underbrace{\Theta(n)}_{\text{add, subtract, shift}}$$

Sloppy version : $T(n) \leq 3T(n/2) + O(n)$

$$\Rightarrow T(n) = O(n^{\log_2 3}) = O(n^{1.585})$$

$$\sum_{i=0}^k x^i = \frac{x^{k+1} - 1}{x - 1}$$

+ Class 6.2 2/9

Master Recurrence

divide and conquer – master recurrence

$$T(n) = d \quad \text{for } n < b,$$

$$T(n) = aT(n/b) + cn^k \text{ for } n \geq b \text{ then}$$

$$a > b^k \Rightarrow T(n) = \Theta(n^{\log_b a}) \quad [\text{many subprobs} \rightarrow \text{leaves dominate}]$$

$$a < b^k \Rightarrow T(n) = \Theta(n^k) \quad [\text{few subprobs} \rightarrow \text{top level dominates}]$$

$$a = b^k \Rightarrow T(n) = \Theta(n^k \log n) \quad [\text{balanced} \rightarrow \text{all } \log n \text{ levels contribute}]$$

Fine print:

$a \geq 1; b > 1; c > 0; d, k \geq 0; n = b^t$ for some $t > 0;$

a, b, k, t integers. True even if it is $\lceil n/b \rceil$ instead of n/b when t is not an integer.

+ Class 6.3 2/11

Dynamic Programming

Intro

+ Class 7.1 2/14

Eg. Give the elements 1, 4, 5, Find the min number of element to get 27.

5	4	1	total number
5	0	2	7
4	1	3	8
3	3	0	6

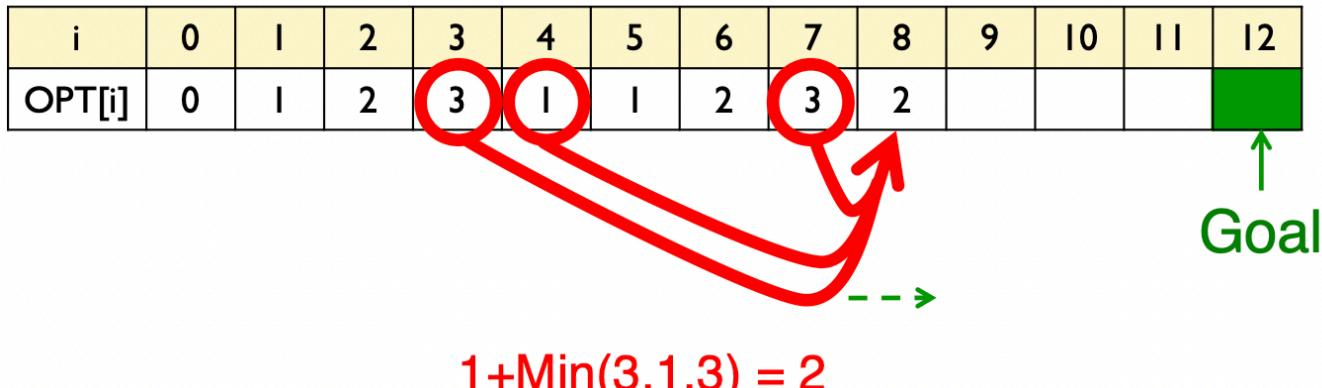
for $i = 0, \dots, N$ do

$$OPT(i) = \min \left\{ \begin{array}{ll} 0 & i=0 \\ 1+OPT(i-1) & i \geq 1 \\ 1+OPT(i-4) & i \geq 4 \\ 1+OPT(i-5) & i \geq 5 \end{array} \right\}$$

New Array Entry

Old Array Access

$OPT(i)$ is the min number of stamps totaling i



Run time: $O(n)$

In this way, we do not know which number is the elements, but we can use trace back

Trace-Back

i	0	1	2	3	4	5	6	7	8	9	10	11	12
OPT[i]	0	1	2	3	1	1	2	3	2				

I + Min(3, I, 3) = 2

Schedule

Weighted Interval Scheduling

Jobs with start time, end time, and value. Time may be conflict, find most valueable sequency.

Input: $n, s_1, \dots, s_n, f_1, \dots, f_n, v_1, \dots, v_n$

Sort jobs by finish times so that $f_1 \leq f_2 \leq \dots \leq f_n$.

Compute $p(1), p(2), \dots, p(n)$

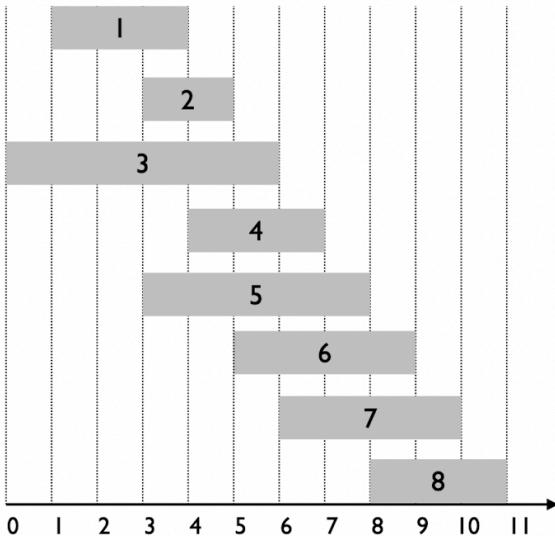
Iterative-Compute-Opt {

```

    OPT[0] = 0
    for j = 1 to n
        OPT[j] = max(vj + OPT[p(j)], OPT[j-1])
    }
```

Output OPT[n]

Exercise: try other concrete examples:
If all $v_j=1$: greedy by finish time $\rightarrow 1,4,8$
what if $v_2 > v_1$, but $< v_1+v_4$?
 $v_2>v_1+v_4$, but $v_2+v_6 < v_1+v_7$, say? etc.



j	p _j	v _j	$\max(v_j + \text{opt}[p(j)], \text{opt}[j-1]) =$	opt[j]
0	-	-		0
1	0	2	$\max(2+0, 0) =$	2
2	0	3	$\max(3+0, 2) =$	3
3	0	1	$\max(1+0, 3) =$	3
4	1	6	$\max(6+2, 3) =$	8
5	0	9	$\max(9+0, 8) =$	9
6	2	7	$\max(7+3, 9) =$	10
7	3	2	$\max(2+3, 10) =$	10
8	5	?	$\max(?+9, 10) =$?

The job is sorted by the end time. p_j is last possible job. v_j is the value.

+ Class 7.2 2/16

That can only find the optimal value, so we need use traceback to find the solution which job has been counted.

```

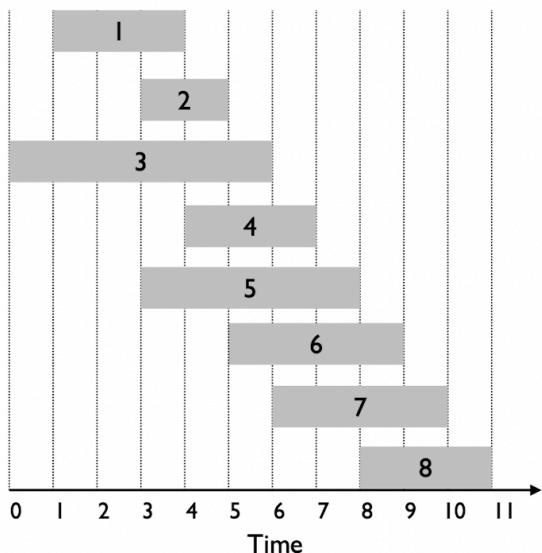
Run M-Compute-Opt(n)
Run Find-Solution(n)

Find-Solution(j) {
    if (j = 0)
        output nothing
    else if (vj + OPT[p(j)] > OPT[j-1])
        print j
        Find-Solution(p(j))
    else
        Find-Solution(j-1)
}

```

the condition determining the max when computing OPT[]

the relevant sub-problem



j	p _j	v _j	$\max(v_j + \text{opt}[p(j)], \text{opt}[j-1]) = \text{opt}[j]$
0	-	-	-
1	0	2	$\max(2+0, 0) = 2$
2	0	3	$\max(3+0, 2) = 3$
3	0	1	$\max(1+0, 3) = 3$
4	1	6	$\max(6+2, 3) = 8$
5	0	9	$\max(9+0, 8) = 9$
6	2	7	$\max(7+3, 9) = 10$
7	3	2	$\max(2+3, 10) = 10$
8	5	.1	$\max(0.1+9, 10) = 10$

V8 = 0.1 is excluded; opt solution is v6+v2

Knapsack Algorithm

Knapsack Algorithm

	W + I											
	0	1	2	3	4	5	6	7	8	9	10	11
ϕ	0	0	0	0	0	0	0	0	0	0	0	0
{1}	0	1	1	1	1	1	1	1	1	1	1	1
{1,2}	0	1	6	7	7	7	7	7	7	7	7	7
{1,2,3}	0	1	6	7	7	18	19	24	25	25	25	25
{1,2,3,4}	0	1	6	7	7	18	22	24	28	29	29	40
{1,2,3,4,5}	0	1	6	7	7	18	22	28	29	34	35	40

$$\text{OPT: } \{ 4, 3 \}$$

$$\text{value} = 22 + 18 = 40$$

W = 11

```

if (wi > w)
    OPT[i, w] = OPT[i-1, w]
else
    OPT[i, w] = max{OPT[i-1,w], vi+OPT[i-1,w-wi]}
```

Item	Value	Weight
1	1	1
2	6	2
3	18	5
4	22	6
5	28	7

Last number in line 5: $40 = \max(40, 28+7)$

Red line: $24 = \max(6+18, 7) = 24$ *where 6: 2{1,2} 18: v_3

+ Class 7.3 2/18

RNA

Dynamic Programming Over Intervals: (R. Nussinov's algorithm)

Notation. $\text{OPT}[i, j]$ = maximum number of base pairs in a secondary structure of the substring $b_i b_{i+1} \dots b_j$.

- Case 1a. If $i \geq j - 4$ (and base b_j is not paired):

$\text{OPT}[i, j] = 0$ by no-sharp turns condition.

- Case 1b. If $i < j - 4$, but base b_j is not paired:

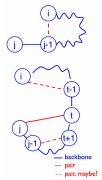
$\text{OPT}[i, j] = \text{OPT}[i, j-1]$

- Case 2. Base b_j pairs with b_t for some $i \leq t < j - 4$.
non-crossing constraint decouples resulting sub-problems

$$\text{OPT}[i, j] = 1 + \max_t \{ \text{OPT}[i, t-1] + \text{OPT}[t+1, j-1] \}$$

Key point:
Either last base
is unpaired
(case 1a,b) or
paired (case 2)

take max over t such that $i \leq t < j-4$ and
 b_t and b_j are Watson-Crick complements



- + Class 8.1 holiday
- + Class 8.2 2/23