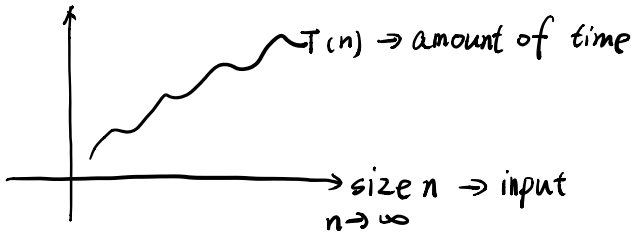
Chapter 1.1: Runtime Complexity



O(1) 是 constant time

asymptotic notations: O, Ω, Θ 渐近的.

1.1.1 Upper-Bound (Big-O)

{ 常数项可忽略: $O(f(n)) = O(c \times f(n))$
低次项可忽略: $O(n^a + n^b) = O(n^a)$, $a > b > 0$

For any monotonic functions f, g from the positive integers to the positive integers.

we say $f(n) = O(g(n))$ 不是唯一的, is a family

if $g(n)$ eventually domainates $f(n)$

$\exists$ a positive real number, $c > 0$, that for all sufficient large n: $f(n) \le c * g(n)$ ☆

eg. $n^2 + 2n + 1 = O(n^2)$ ∵ $\underbrace{n^2 + 2n + 1}_{f(n)} \le n^2 + 2n^2 + n^2 = \underbrace{4n^2}_{g(n)}$

eg. $1 < \log n < n \le n^2 \le n^2 \cdot \log n \le n^3 \le 2^n \le n! \le n^n$

Discussion

1. 排序

$\log^2 n$ < $2^{\log n}$ < $n \cdot \log\log n$ < $\log n^n$ < $n^{\sqrt{n}}$ < n^2 < $n^{\log n}$

$\log^2 n$: 比linear小. $2^{\log n}$: 公式, $\log n = x$, $2^x = n$

2. suppose $f(n)$ and $g(n)$ are two positive non-decreasing functions such that $f(n) = O(g(n))$. Is it true that $2^{f(n)} = O(2^{g(n)})$?

$f(n) = 2n$, $g(n) = n$, $f(n) = ? = O(g(n))$

$2^{f(n)} = 2^{2n} = 4^n$

$2^{g(n)} = 2^n$

$f(n) \le c \cdot g(n)$

$2n \le c \cdot n$

$c = 2$

$4^n \neq O(2^n)$ False ☆

1.1.2 Lower Bound Ω

$f(n) = \Omega(g(n))$

if $f(n)$ eventually dominates $g(n)$

formally: $f(n) \geq c * g(n)$ ☆

eg. $n^2+2n+1 = \Omega(n^2)$ ∵ $n^2+2n+1 \geq n^2$ for $n \geq 1$, we choose $c=1$ and $n_0 = 1$

1.1.3 Exact Bound Θ

$f(n) = \Theta(g(n))$

if: $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$

Discussion

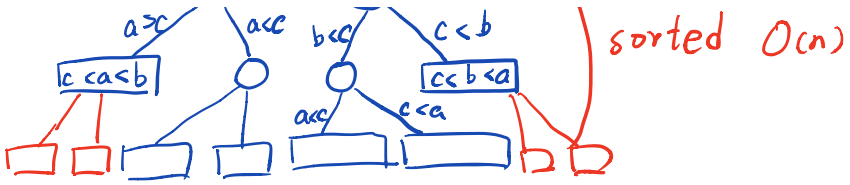
1. $n = \Omega(n^2)$ ✗ $n \geq c*n^2$ ✗
2. $n = \Theta(n + \log n)$ ✓ $n = \Theta(n)$ (不care)
3. $\log n = \Omega(n)$ ✗ $\log n \geq c*n$ ✗
4. $n^2 = \Omega(n \log n)$ ✓ $n^2 \geq c*n\cdot\log n$
5. $n^2 \log n = \Theta(n^2)$ ✗ $\neq 0$
6. $3n^2 + 4n + 5 = \Theta(n^2)$ ✓
7. $2^n + 100n^2 + n^{100} = \Omega(n^{101})$ ✓ 2^n的增速 > n^{101}
8. $(\frac{1}{3})^n + 100 = \Theta(1)$ ✓ ($(\frac{1}{3})^n \to 0$)

Chapter 1.2. Sorting Lower bound.

We will show that any deterministic comparison-based sorting algorithm must take $\Omega(n \log n)$ time to sort an array of n elements in the worst-case.

[a, b, c]

$a \neq b \neq c$



leaves : $n!$ $3! = 3 \times 2 = 6$

add 4 leaves : leaves : 2^h , $h=3$, 高度. $2^h = 8$ $\quad n! \sim n^n$

$$2^h > n! \Rightarrow h \geq \log(n!)$$
$$h = \Omega(\log n^n)$$
$$h = \Omega(n \log n)$$

Discussion

1. void bigOh1 (int[] L, int n)

```
n: while (n>0)
log4n [ find-max (L, n);
        n = n/4;
```

$O(n \cdot \log n)$

$$n + \frac{n}{4} + \frac{n}{16} + \cdots + 1 \leq n\left(1 + \frac{1}{4} + \frac{1}{16} + \cdots\right)$$
$$= n \cdot \frac{1}{1-\frac{1}{4}} = n \cdot \frac{4}{3} = O(n)$$
$$\sum_{k=0}^{\infty} x^k = \frac{1}{1-x}$$

2. string bigOh2 (int n)

```
if (n == 0) return "a";
string str = bigOh2(n-1);
return str + str;
```

→ 递归 : n.

bigOh2(0) → 'a' 2^0
bigOh2(1) → 'aa' 2^1
bigOh2(2) → 'aaaa' 2^2
bigOh2(3) → "aaaaaaaa" 2^3 $\quad \therefore O(n \cdot 2^n)$ ☆

Chapter 1.3. Trees and Graphs.

A **graph** G is a pair (V, E) where V is a set of vertices (or nodes) E is a set of edges connecting the vertices. An **undirected** graph is connected when there is a path between every pair of vertices.

A **tree** is a connected graph with no cycles.

A path in a graph is a sequence of distinct vertices.

A cycle is a path that starts and ends at the same vertex.

Theorem. Let G be a graph with V vertices and E edges.

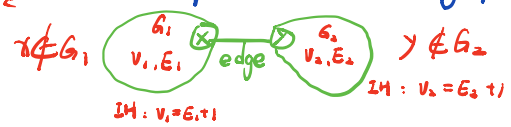
1. G is a tree (a connected graph with no cycles)
2. every two vertices of G are connected by a unique path.
3. G is connected and $V = E+1$ $\quad |V| = |E|+1$
4. G is acyclic and $V = E+1$. (acyclic: 非循环的)
5. G is acyclic and if any two non-adjacent vertices are joined by an edge, the resulting graph has exactly one cycle.

proof: 1→2

2→3 proof by Induction (归纳) on vertics

Induction hypothesis
1. Base case: o—o $V=2, E=1.$ $d=E+1$
2. IH: assume $V=E+1$ is true for graphs with $V<n$.

Induction step
3. IS: prove $V=E+1$ for graphs with $V=n$



$V = V_1 + V_2 = (E_1 + 1 + E_2) + 1 = E + 1$

3→4. proof that G is an acyclic graph by contradiction.

Assum G has a cycle with k vertices in it., it contains k edges.

the number of edges in graph is at least V,

So ⇒ k edges in cycle, at least $V-k+1$ outside the cycle

Theorem: Prove that in an undirected simple graph $G = (V, E)$. there are at most $V(V-1)/2$ edges. In short. using the asymptotic notation, $E = O(V^2)$.

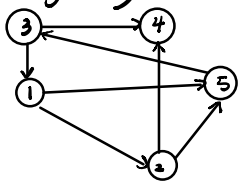
$(V-1) + (V-2) + (V-3) + \dots + 1 = \frac{(1+V-1)(V-1)}{2}$ 求和.

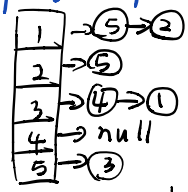
Representing Graphs.

1. Adjacency List or Adjacency Matrix

Vertex x is adjacent to vertex y if and only if there is an edge (x, y) between them.

1. Adjacency List → 用于 repersent sparse graph $E = O(V)$





1. Is vertex 1 adjacent to 3? No

2. It takes linear time to figure it out

2. Adjacency Matrix → 用于 repersent dense graph $E = O(V^2)$

	1→1	1→2	1→3	1→4	1→5
1	0	1	0	0	1
2	0	0	0	1	1
3	1	0	0	1	0
4	0	0	0	0	0
5	0	0	1	0	0

Note: A connected graph is maximally sparse if it's a tree.

A graph is maximally dense if it's complete

↑ 每两个V之间都有 edge, binomial coefficient.

Graph Traversals (遍历)

visit all vertices in a systematic order, not all edges.

the result of traversals are trees. (spanning tree)

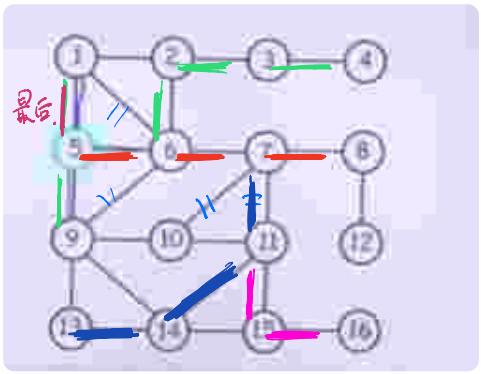
遍历分类:

DFS: Depth-first search: stack for backtracking

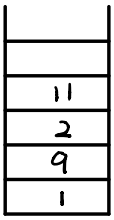
BFS: Breadth-first search: queue for book-keeping

时间复杂度: $O(V+E)$ ☆

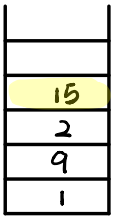
DFS:



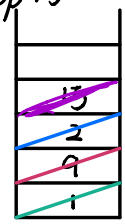
(1)



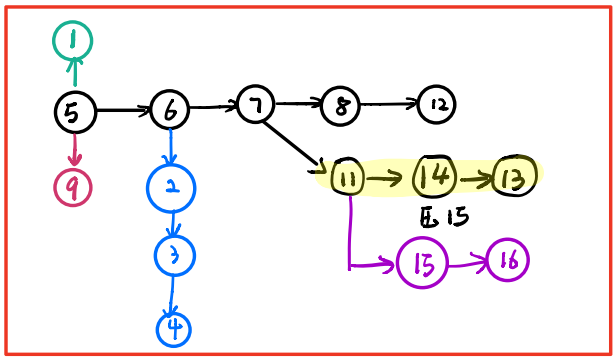
(2) pop 11 push 15



(3) pop 15

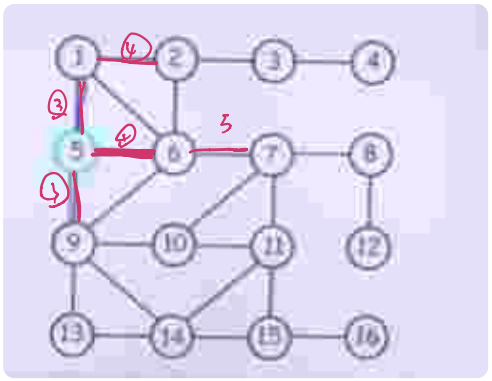


依次出栈，最后一个不多



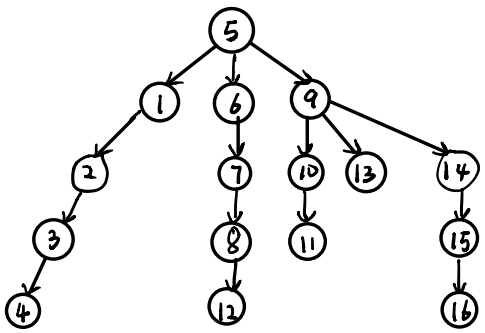
tall spanning tree

BFS



queue:

← 11 9 2 7 ←



short spanning tree

Discussion

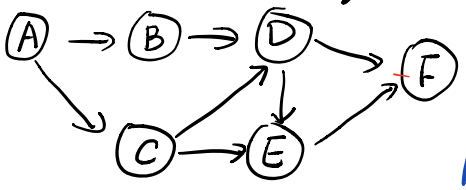
The complete graph on vertices, denoted K_n, is a simple graph in which there is an edge between every pair of distinct vertices.

1. The height of the DFS tree for K_n? $n-1$ (short tall) 一条通走到底，不用POP

2. The height of the BFS tree for K_n? 1

Topological Sort for DAG (direct acyclic graph) 拓扑结构

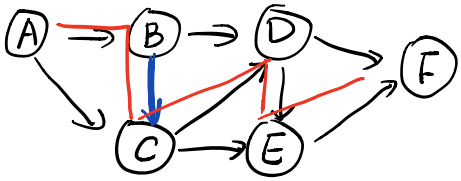
V: tasks. a direct edge (u, v) indicates that task v depends on task u. ∋ u → v. The topological ordering of the vertices is a valid order in which you can complete the tasks.



longest, give a linear-time.

ACEBDF

Discussion:



ABCDEF

① 选择无前趋点并入栈

② 栈 ≠ null 时，pop 一个 vertix，并删除与之相连的所有有向边

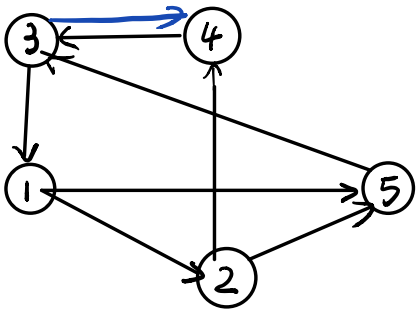
③ 重复以上步骤，直至无 vertix

Linear Time Algorithm

May run DFS several times FEDBCA

Strongly Connected Graphs

a directed graph. It's strangly connected if each vertex is reachable from any other vertex. 每个点都能到达另一个点，



如果 4→3 换成 3→4

4 就不能到任何地方.

weakly connected graph

How do you test if a graph is strongly connected?

Brute Force Algorithm: run DFS/BFS from each vertex.

Complexity: $O(V(V+E)) = O(V^2 + V \cdot E)$

$O(V \cdot E)$ ← $\neq O(V^3)$

$V^2 + VE = V \cdot E$

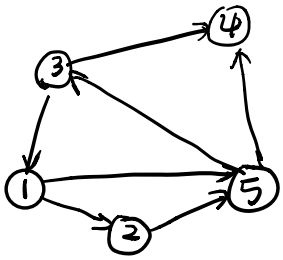
$\because V^2 < VE$

$E = O(V^2)$ for dense graph

E 最大为 V^2

$\therefore V \cdot E > V^2$

Q.E.D



Review

1. Any func. $f(n)$ which is $\Omega(\log n)$ is also $\Omega(\log(\log n))$

T. $\Omega(\log n) = \Omega(\log(\log n))$

$\exists c,\ f(n) \geq c \cdot \log n \geq c_2 \log\log n$

2. If $f(n) = \Theta(g(n))$ then $g(n) = \Theta(f(n))$.

T $\exists c_1, c_2,\ c_1 g(n) \leq f(n) \leq c_2 \cdot g(n)$ Θ

Θ 的定义☆

$\therefore \quad g(n) \geq \frac{1}{c_2} f(n)$ $\qquad f(n) = \Omega(g(n))$

$g(n) \leq \frac{1}{c_1} f(n)$ $\qquad f(n) = O(g(n))$

$\therefore \quad \frac{1}{c_2} f(n) \leq g(n) \leq \frac{1}{c_1} f(n)$ 符合自己的定义.

3. If $f(n) = \Omega(g(n))$ then $2^{f(n)} = \Omega(2^{g(n)})$

T

$\exists c,\ f(n) \geq c \cdot g(n) \Rightarrow 2^{f(n)} \geq 2^{c \cdot g(n)}$

$f(n) = n$, $g(n) = 5n$, $c = \frac{1}{50}$

4. $n^{\frac{1}{\log n}}$, $\sqrt{\log n}$, $2^{\sqrt{2\log n}}$, $(\sqrt{2})^{\log n}$, $4^{\log n}$, $(\log n)!$, $n^{\log\log n}$.

$n^{\frac{1}{\log n}} \to n^{\log_n 2} \to 2$

$2^{\sqrt{2\log n}} \to < \sqrt{n}$

$(\sqrt{2})^{\log n} \to 2^{\frac{1}{2}\log n}$

$4^{\log n} \to n^2$

① $\log_b a = \frac{\log a}{\log b}$, $\therefore \log_n 2 = \frac{1}{\log n}$

② $2^{\sqrt{2\log n}} < \sqrt{n}$

$\sqrt{2\log n} < \log \sqrt{n} = \frac{1}{2}\log n$ true.

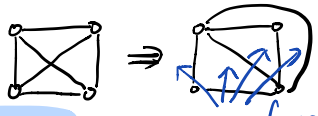
③ $n^{\log\log n}$ $\qquad (\log n)!$

$\log n = t$, $n = 2^t$

$n^{\log t}$? $t!$

planar graph

a graph is planar if it can be drawn in the plane with each edge drawn in the plane without crossing edges.



4 faces.

Euler's Formula

A planar graph when drawn in the plane, splits the plane into disjoint faces.

If G is a connected planar graph with V vertices, E edges and F faces.

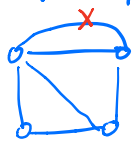
then $V - E + F = 2$

proof by Induction on # of edges.

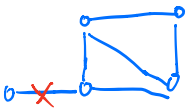
Base case : o—o $E=1, V=2, F=1$; $V-E+F = 2-1+1=2$

Inductive hypothesis: Assume it holds for E edges.

Inductive step: prove $V-E+F$ for $E+1$ edge



$E--, V, F--$; $V-E+F=2$



$E--, V--, F$; $V-E+F=2$

$-1 \quad -1$

Coloring Planar Graphs

A coloring of a graph is an assignment of a color to each vertex such that no neighboring vertices have the same color

Theorem : Every planar graph can be colored with at most six colors.

proof by Induction on # of vertices.

Base case: $G=(V,E)$, $V \leq 6$

IH : assume that all graphs with $V-1$ vertices are six-colorable

IS : prove it for V vertices

remove a vertex with <6 degree. I will put a graph with $V-1$ vertices.

Apply IH, Color

E (a . . . . . . b)

E

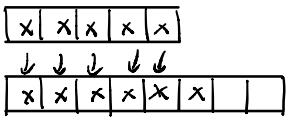
T = (a . . . b)

T=1 → T=2 → W

log E (V+E)

Amortized Analysis

Unbounded Array



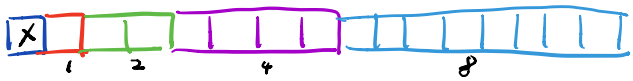
insert $O(1)$

$\begin{cases} \text{copy } 5 & O(n) \\ \text{insert} & O(1) \end{cases} \Rightarrow O(n)$

worst-case $O(n)$

Ins	Old size	new size	Copy	
1	1	–	–	# insert $s = 9$
2	1	2	1	# copy $i = 1+2+4+8 = 16$
3	2	4	2	$AC = \frac{16}{9}$
4	4	–	–	
5	4	8	4	
6			–	
7			–	
8			–	
9	8	16	8	

成倍增长



# of insertions $= 2^n + 1$

# of copy $= 1+2+4+\cdots+2^n = 2^{n+1}-1$

Total work : $2^n + 1 + 2^{n+1} - 1 = 2^n + 2^{n+1} = 3 \cdot 2^n$

$\sum_{k=0}^{n} 2^k = 2^{n+1} - 1$

$AC = \frac{\text{Total work}}{\text{\# insertions}} = \frac{3 \cdot 2^n}{2^n + 1} = O(3) = O(1)$

Average Cost

Binary Counter

8 2^3 8有3bit的二进制

Given a binary number with $\log(n)$ bits, sorted as an array, where each entry $A[i]$ stores the i-th bit. 3

The cost of incrementing a binary number is the number of bits fliped with n bits

change n次 change n次 change n次

0	0	0	—
0	0	1	1
0	1	0	2
0	1	1	1
1	0	0	3
1	0	1	1
1	1	0	2
1	1	1	1
0	0	0	3
			14.

$ac = \frac{\#flips}{n}$

最后一次变 $3n$ 次，倒数第二个就会变 $\frac{n}{2}$ 次

$n + \frac{n}{2} + \frac{n}{4} + \dots + 2 = O(n) \le n(1 + \frac{1}{2} + \frac{1}{4} + \dots + t) = n\sum_{k=0}^{\infty} \frac{1}{2^k} = O(n)$

$ac = \frac{O(n)}{n} = O(1)$

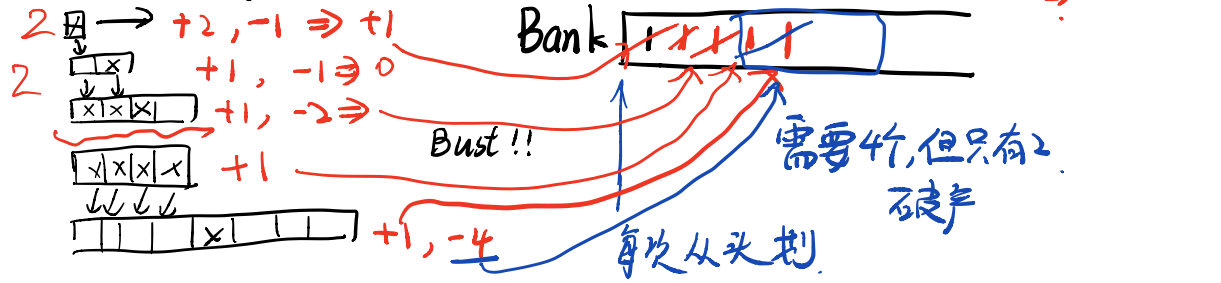
↖ 一共变 $3n$ 次

a "Ponzi" scheme

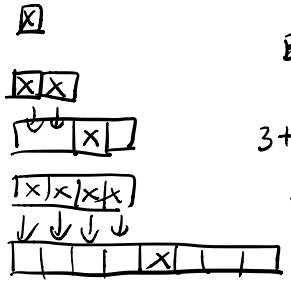
1) each insert gets 1 tokens Bank 0

+1 insert -1 copy -1 | 0 | -1 破产

2) each insert gets 2 tokens 每次 +1 ⇒ (insert 和 2 tokens)



3) each insert gets 3 tokens



Bank 2 2 3+2-2=3

3+2·3=9

3+2-2=3

3+2=5

5+2-4=3

3+2=5

5+2=7

7+2=9

花不完

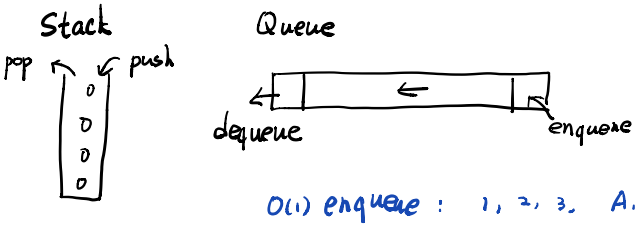
9个

Discussion

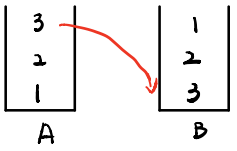
You have a stack data type, and you need to implement a FIFO queue. The stack has the usual POP and PUSH operations, and the cost of each operation is 1. The FIFO has two operations: ENQUEUE and DEQUEUE.

We can implement a FIFO queue using two stacks. What is the amortized cost of ENQUEUE and DEQUEUE operations?

AC = O(1)



O(1) enqueue : 1, 2, 3. A.push()



O(n)

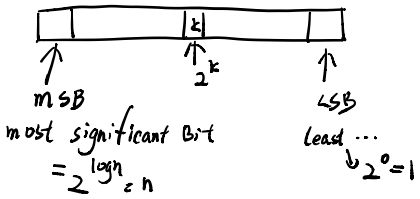
dequeue : A.pop } loop
B.push

B.pop

dequeue : B.pop()

1. How many tokens to enqueue? O(3) O(1) token Bank 6-3-3=0
2. How many tokens to dequeue? O(1) B.pop. O(n) tokens

number n with log n bits



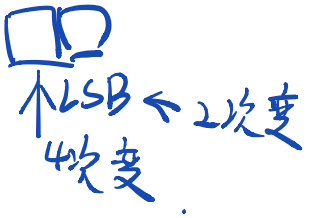
cost of each flip is 2^k ⇒ 每一个.

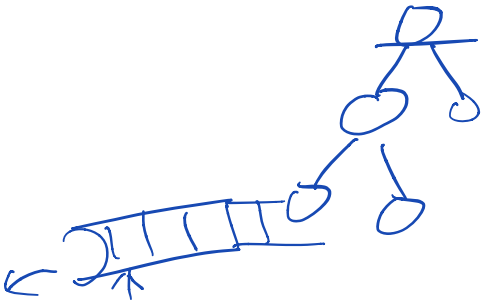
find AC of each flip O(log n)

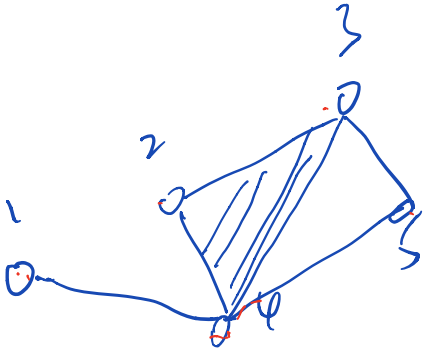
$n \cdot 2^0 + \frac{n}{2} \cdot 2 + \frac{n}{4} \cdot 2^2 + \cdots + 2 \cdot n = O(n \log n)$

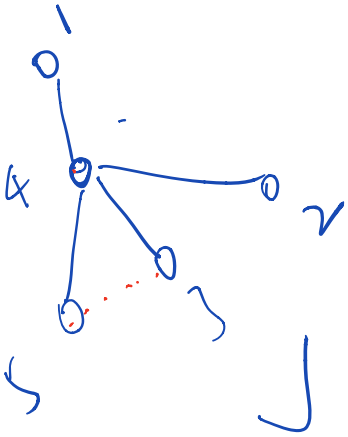
AC per flip = $\frac{O(n \log n)}{O(n)} = O(\log n)$

↑ number of flips 上面算的









REVIEW QUESTIONS

1. What is the definition of the amortized cost using the aggregate method?
2. **(T/F)** Amortized analysis is used to determine the average runtime complexity of an algorithm. F → 应是 a sequence of operations
3. **(T/F)** Compared to the worst-case analysis, amortized analysis provides a more accurate upper bound on the performance of an algorithm. T
4. **(T/F)** The total amortized cost of a sequence of n operations gives a lower bound on the total actual cost of the sequence. F — upper bound
5. **(T/F)** Amortized constant time for a dynamic array is still guaranteed if we increase the array size by 5%. T
6. **(T/F)** If an operation takes $O(1)$ expected time, then it takes $O(1)$ amortized time. F → 不一样
7. Suppose you have a data structure such that a sequence of n operations has an amortized cost of $O(n \log n)$. What could be the highest actual time of a single operation? Total $n \log n$
8. What is the worst-case runtime complexity of searching in an amortized dictionary?

7. $\underbrace{1, 1, \ldots 1}_{n-1}, n^2 = O(n^2)$

$O(n \log n)$ ✓

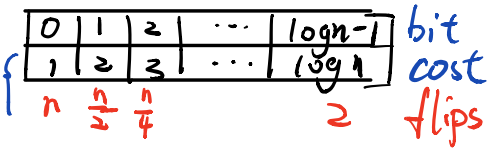
~~$O(n^2)$~~

~~$O(n^2 \log n)$~~ } 不可能为 n^2.

could → $O(\log n)$

could → $O(n)$

2. We are incrementing a binary counter, where flipping the i-th bit costs $i + 1$. Flipping the lowest-order bit costs $0 + 1 = 1$, the next bit costs $1 + 1 = 2$, the next bit costs $2 + 1 = 3$, and so on. What is the amortized cost per operation for a sequence of n increments, starting from zero?



$$n \cdot 1 + \frac{n}{2} \cdot 2 + \frac{n}{4} \cdot 3 + \dots + 2 \cdot \log n = n \sum_{k=0}^{\log n - 1} \frac{(k+1)}{2^k} \leq n \cdot \sum_{k=0}^{\infty} = O(n)$$

AC per increment is $\frac{O(n)}{n} = O(1)$

Heaps

Heap and Priority Queue

1. sort $O(n \log n)$
2. findMax $O(n)$
3. heap $O(\log n)$

PQ

1. insert
2. deleteMin.

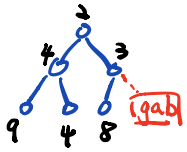
3.1 Binary Heap

A binary heap is a complete binary tree which satisfies the heap ordering property.

1. Structure Property

   中间无gap，只有最后的level have gap

   从左往右填.



2. Ordering Property

0	1	2	3	4	5	6	7
X	2	4	3	9	4	8	gap

   ① min-heap : $P \leq L$ and $P \leq R$

   ② max-heap : $P \geq L$ and $P \geq R$

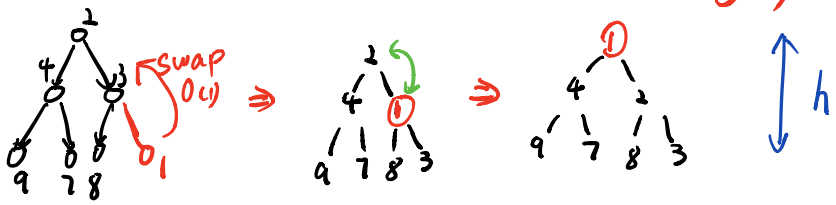
Consider k-th element of the array

① its left child is located at $2k$ index

② its right child is located at $2k+1$ index

③ its parent is located at $k/2$ index

3.1.2 insert (tree reps) ~ O(logn)



1	2-1
3	4-1
7	8-1
15	16-1
31	62-1 $2^{h+1}-1$

worst-case complexity $O(h) = O(\log n)$

0	1	2	3	4	5	6	7
X	2	4	3	9	7	8	1
X	2	9	1	9	7	8	3
X	1	4	2	9	7	8	3

Discussion Problem 1.

The values 1, 2, 3, ..., 63 are all inserted (in any order) into an initially empty min-heap. What is the smallest number that could be a leaf node?

A) 32

B) 33

C) 1

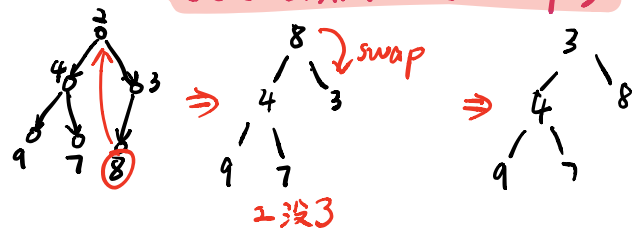
D) 2

E) 6

$2^{h+1} - 1 \geq 63 \quad 63 = 2^6 - 1$

$h = 5$

3.1.3 deleteMin (tree reps) O(log n)



worst-case complexity - O(log n)

0	1	2	3	4	5	6	7
X	2	4	3	9	7	8	

X	8	4	3	9	7		
X	3	4	8	9	7		

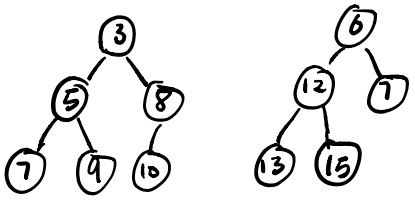
Discussion Problem 2.

two binary min-heaps, A and B. with a total of n elements between them. find if A and B have a key in common. Give a solution to this problem that takes O(nlogn). Do not use the fact that heaps are implemented as arrays.

HW1, pro. 2.

first : interface.
DFS.

two : BFS visited.



A.delmin ⑤ B.delmin ①

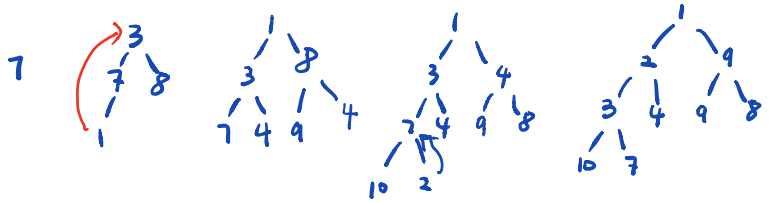
A.delmin ⑦ ⑦=⑦

最坏情况：del 所有的，n次，每次 logn. ∴ nlogn

buildHeap - by insertion

give an array - turn it into a heap.

7, 3, 8, 1, 4, 9, 4, 10, 2, 0

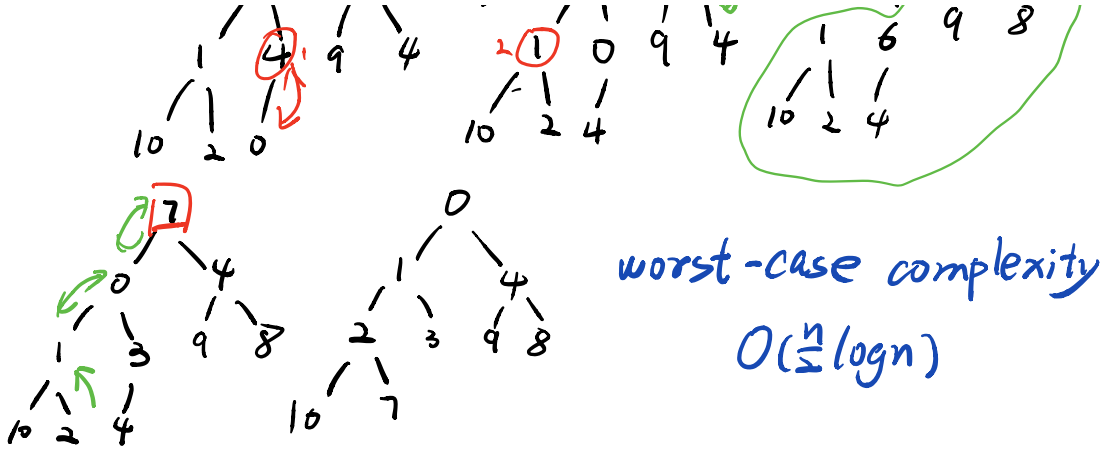


worst-case complexity: O(nlogn)

log1 + log2 + log3 + ⋯ logn = log n! = Ω(nlogn) ⇒ lecture 1.

Heapify.

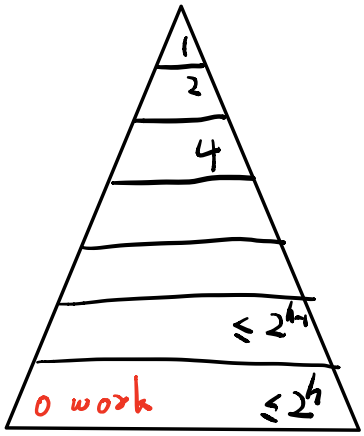
O(n)



worst-case complexity

$O(\sum_{}^{n} \log n)$

complexity of heapify. h高度 = logn.



height	# of nodes	# of swaps
0	1	h
1	2	h-1
2	4	h-2
⋮	⋮	⋮
h-2	2^{h-1}	2
h-1	2^{h-1}	1

$$T(n) = h\cdot 1 + (h-1)\cdot 2 + (h-2)\cdot 3 + \cdots + 2^{n-1}\cdot 1 = \sum_{k=1}^{h} k\,2^{h-k} \quad (h = \log n)$$

$$= 2^h \sum_{k=1}^{h} \frac{k}{2^k} \le 2^h \sum_{k=1}^{\infty} \frac{k}{2^k} = O(2^h) = O(2^{\log n}) = O(n)$$

$T(n) = O(n)$

Discussion Problem 3.

How would you sort using a binary heap?

What is it runtime complexity?

Run deleteMin n times, $O(n \log n)$

Space Complexity: to store sorted data $O(n)$

$O(1)$

in-beace

stable sort?

BST, $O(n)$

Discussion Problem 4

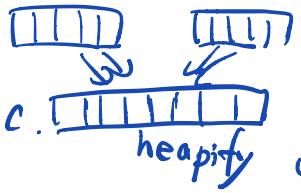
How would you merge two binary min-heaps?

What is it runtime complexity?

方法一: A B

B.insert(A.deleteMin) $O(n \log n)$

方法二:



$O(n)$

$O(n)$

$\Rightarrow O(n)$

Discussion Problem 5

give a unsorted array of size n. Devise a heap-based algorithm that finds the k-th largest element in the array. What is its runtime complexity?

法一: offline algorithm:

1. build max-heap of n elements $O(n)$
2. call deleteMax k times $O(\log n)$

$O(n + k \log n)$

法二: Online algorithm:

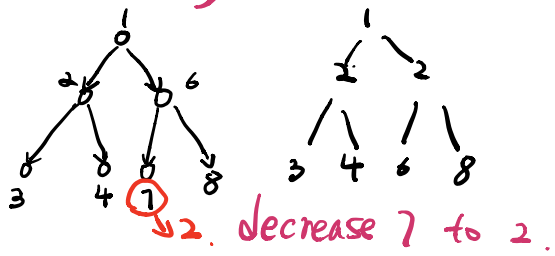
1. build min-heap of k elements. $O(k)$

wait for (k+1)

root > 第(k+1)个, then do nothing

否则, deletmin, insert $O((n-k)\log(k))$

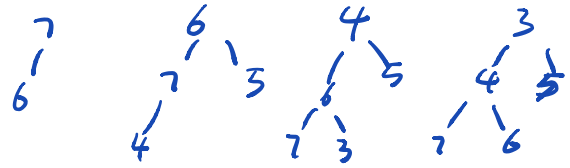
DecreaseKey 换一个值 $\rightarrow O(\log n)$



3.2 Binomial Heaps

We want to create a heap with a better amortized complexity of insertion.

Insert: 7, 6, 5, 4, 3, 2, 1



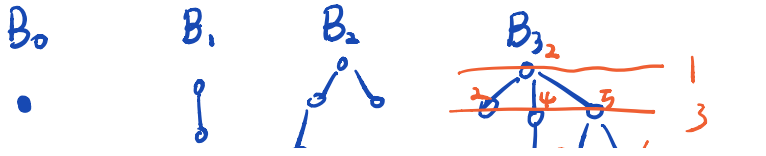
$\sum_{k=0}^{\log n - 1} 2^k \cdot k = O(n \log n)$ = worsed-cost

The binomial tree B_k is defined as

1. B_0 is a single node
2. B_k is formed by joining two B_{k-1} trees.

$(1+x)^3 = 1 + 3x + 3x^2 + x^3$

1 3 3 1

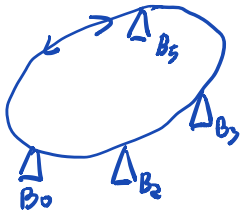


Binomial Heaps

A binomial heap is a collection (a linked list) of at most Ceiling($\log n$) binomial trees (of unique rank) in increasing order of size where each tree has a heap ordering property.

Queue

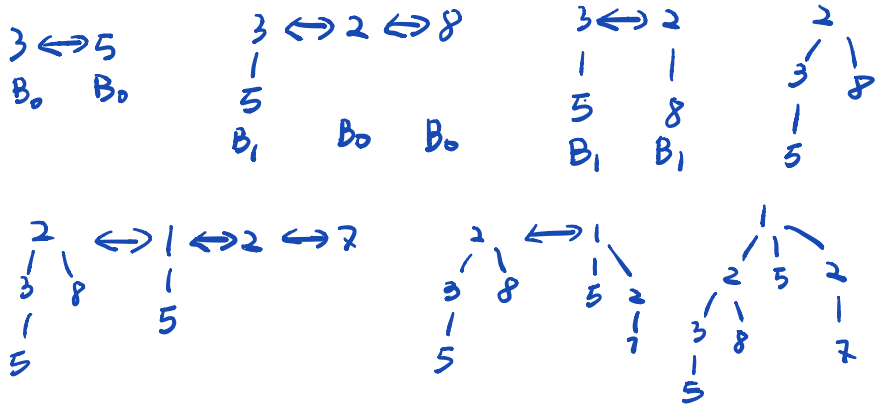
each tree of unique rank.



Discussion Problem 6

Given a sequence of numbers: 3, 5, 2, 8, 1, 5, 2, 7

Draw a binomial heap by inserting the above numbers reading them left to right.



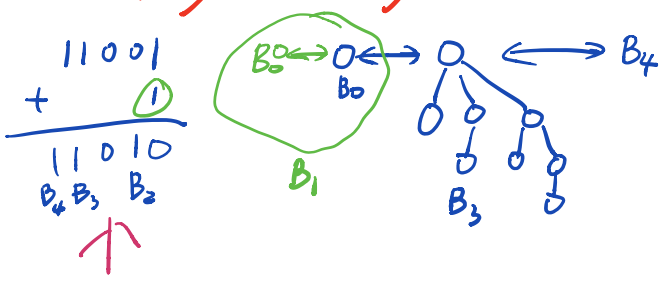
Discussion Problem 7

How many binomial trees does a binomial heap with 25 elements contain?

What are the rank of those trees?

$25_2 = (16+8+1) = 1\ 1\ 0\ 0\ 1$
$2^4\ 2^3\ 2^0$ $B_4\ B_3$ B_0

Insertion is binary addition by 1



Insertion

What is its worst-case runtime coplexity? ?

$O(\log n)$

What is its amortized runtime complexity?

insert 2 tokens $O(2) = O(1)$

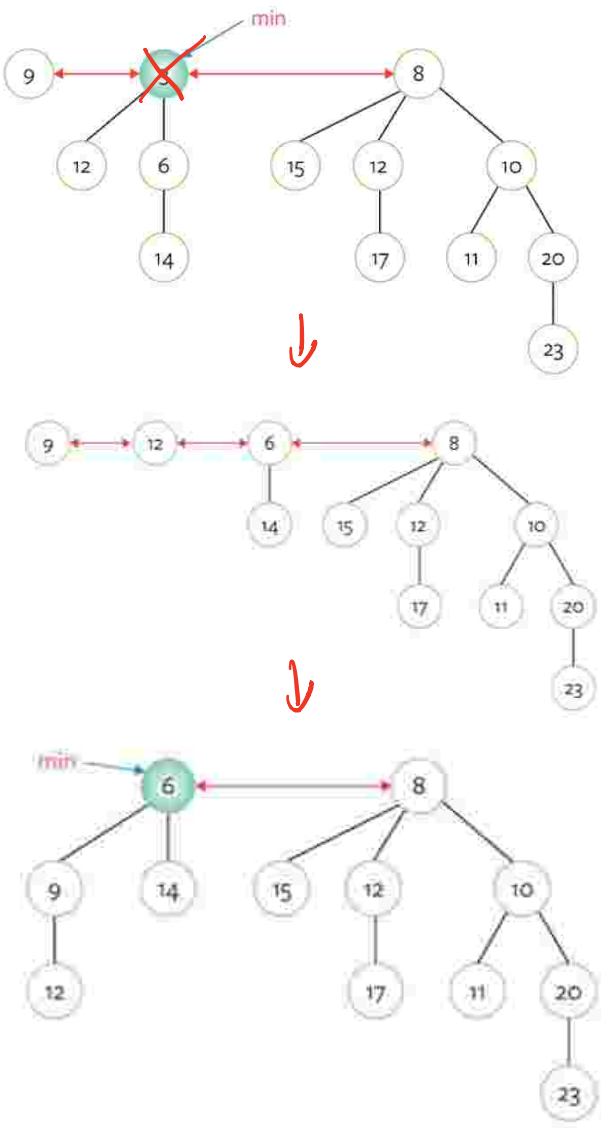
Binary vs Binominal heaps

① insert n elements
worst-case $O(n \log n)$ online

② if n is known advance.
run heapify, binary heap
can be constructed in time $O(n)$ offline

Insert n elements, one after other.
is $O(n)$ (amortized cost). even
if n is not known in advance.

deleteMin() $O(\log n)$



Discussion Problem 8

Devise an algorithm for merging two binomial heaps and discuss its complexity. Merge $B_0 B_1 B_2 B_4$ with $B_1 B_4$.

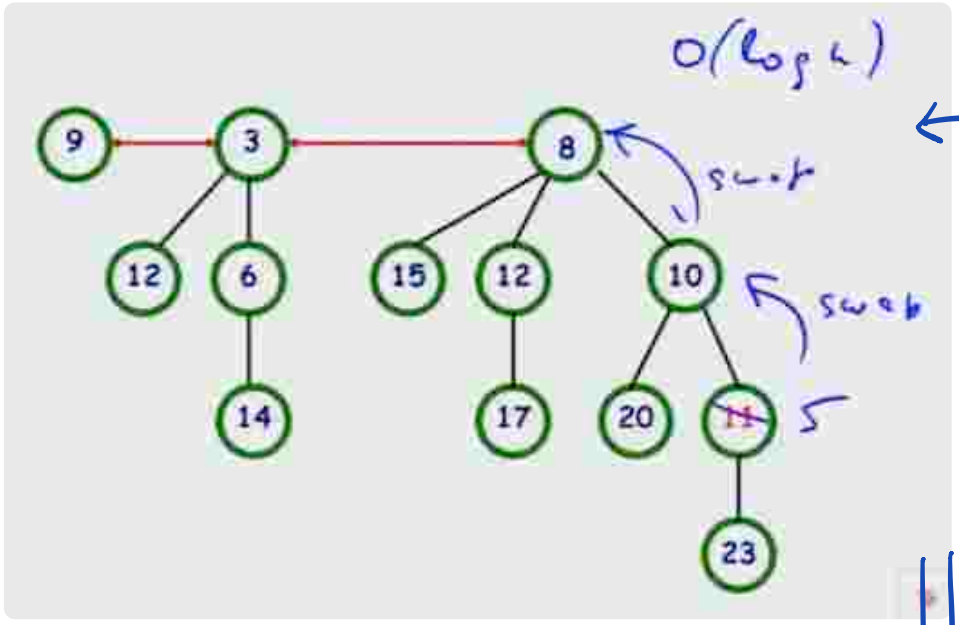
$$\begin{array}{r} 10111 \\ +10010 \\ \hline 101001 \end{array}$$

$O(\log n)$

3.3 Fibonacci Heaps

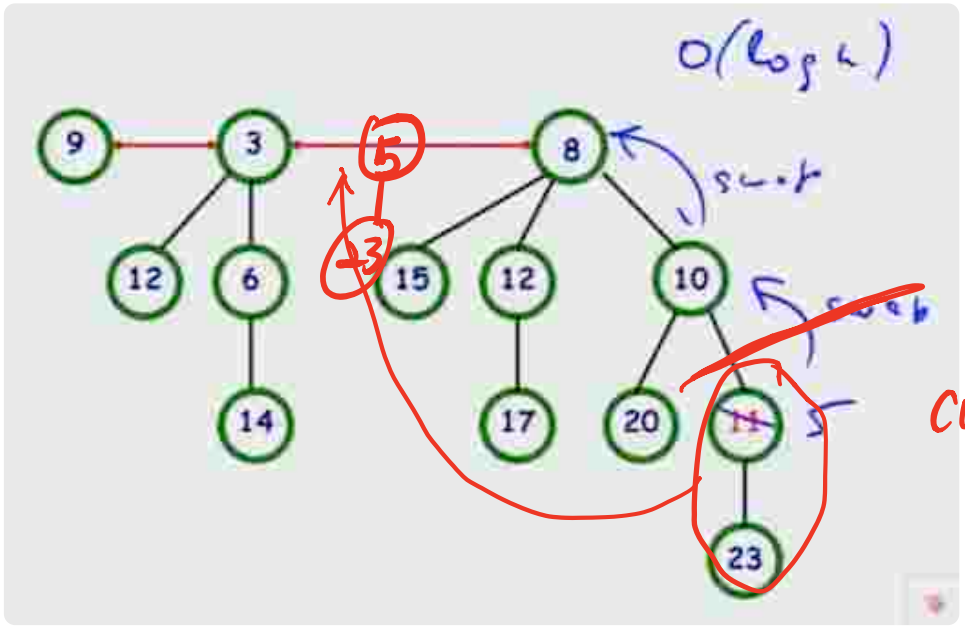
Idea : relaxed (lazy) binomial heaps

Goal : decreasekey in O(1) ac.



← binomial heap

改进



cut !!

Summary

TABLE 3.4 Running times for heap operations

	Binary	Binomial	Fibonacci
findMin	$\Theta(1)$	$\Theta(1)$	$\Theta(1)$
deleteMin	$\Theta(\log n)$	$\Theta(\log n)$	$O(\log n)$ (ac)
insert	$\Theta(\log n)$	$\Theta(1)$ (ac)	$\Theta(1)$
decreaseKey	$\Theta(\log n)$	$\Theta(\log n)$	$\Theta(1)$ (ac)
merge	$\Theta(n)$	$\Theta(\log n)$	$\Theta(1)$ (ac)

Review Question

1. What is the worst-case runtime complexity of finding the largest item in a binary min-heap?

   O(n log n) deleteMin, 每次log n, n次

2. What is the a.c runtime complexity of inserting into a binomial heap?

   O(1) one insert : 2 tokens, cost 1 tokens
   each tree have a token

3. In a binary min-heap with n elements, the worst-case runtime complexity of finding the second smallest element is ~~O(1)~~. O(log n)

   ① deletmin (log n)
   ② find min (O(1))

4. In a binomial min-heap with n elements, the worst-case runtime complexity of finding the

second smallest element is O(1). F

O(logn)

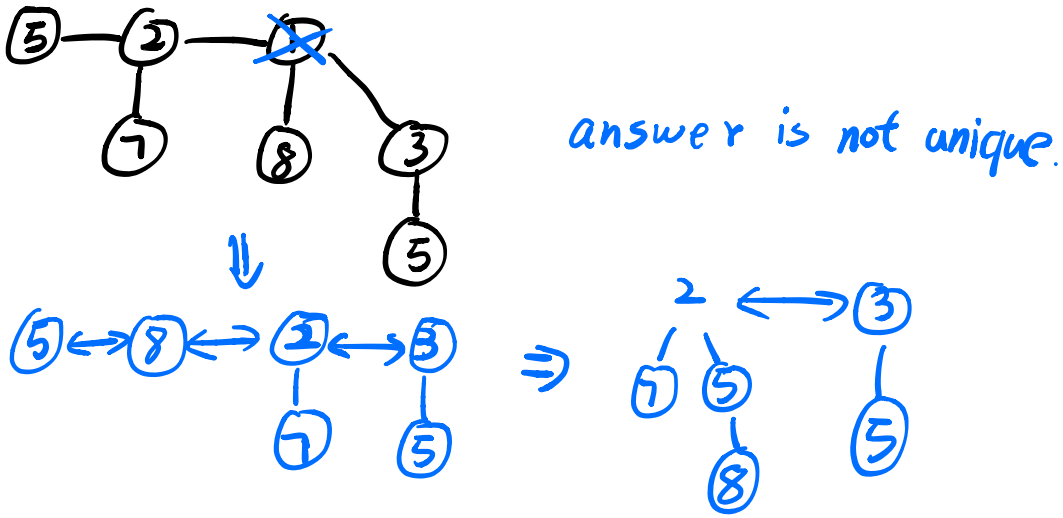
5. Given a Fibonacci heap of size n, the maximum number of trees is that heap is n. F.

2

所有的都是 single.

*. a collection of tree with binary min-heap

6. Show a heap that would be the result after the call to deleteMin() on this heap.



answer is not unique.

5. Prove that it is impossible construct a min-heap (not necessarily binary) in a comparison-based model with *both* the following properties:
   a. deleteMin() runs in $O(1)$
   b. buildHeap() runs in $O(n)$, where n is the input size

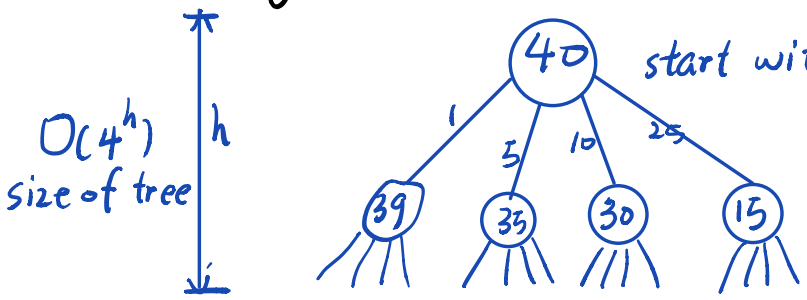
If I build heap in linear time, called deletmin n times. then we got sorting in n times

∵ comparison-based model, sort O(nlogn)

Greedy Algorithms.

4.1 The Money Changing Problem.

we are to make a change of $0.40 use US currency and assuming that there is an unlimited supply of coins.



$40 = 25 + 30 + 5$. greedy always choose the largest avaliable coins.

$O(h)$ is the size the real branch.

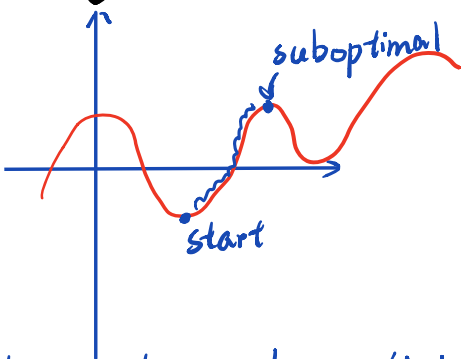
Suboptimal solution

Greedy Algorithm does not always yield the global optimal solution.

money option: 1, 5, 10, 20, 25

Greedy : $25 + 20 + 5 = 40$

but optimal : $20 + 20 = 40$



Greedy Algorithm

- They make a sequence of choices. , it is used to solve optimization problems.
- Each choice is the best available at each step.
- Earlier decisions made during execution are never undone. can not go back
- They do not always yield the optimal solution.

Elements of the greedy structure

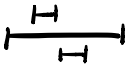
two property : { ① greedy choice property
② optimal substructure

Scheduling Problem

There is a set of n requests. Each request i has a starting time s(i) and finish time f(i). Assume that all request are equally important and $s(i) \le f(i)$. our goal is to develop a greedy algorithm that finds the largest compatible (non-overlapping) subset of requests

How do we choose requests?

1. sort s(i)



2. sort f(i) - s(i)

3. sort f(i) ✡ ✓

proof by induction:

solution by Greedy algorithm : $i_1, i_2, \dots i_k$ intervals.

optimal solution (assume) : $O_1, O_2, O_3, \dots O_m$

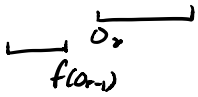
need to prove : $f(i_r) \le f(o_r)$, $\forall r \le k$

① Base case : $r = 1$, $f(i_1) \le f(O_1)$

② IH : assume $f(i_{r-1}) \le f(O_{r-1})$ for (r-1) intervals.

IS : prove $f(i_r) \le f(O_r)$

$f(i_{r-1}) \le f(O_{r-1}) \le S(O_r)$ (next; not overlap)



prove $k = m$, By contradiction

Assum $k < m$

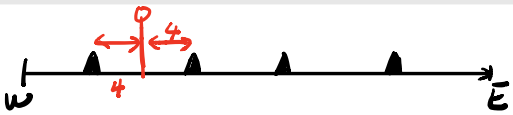
上面证明

$f(O_k) \le S(O_{k+1})$ and $f(i_k) \le f(O_k)$

$f(i_k) \le S(O_{k+1})$

Discussion Problem 1.

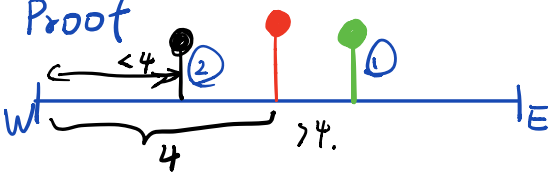
Let's consider a long, quiet country road with houses scattered very sparsely along it. We can picture the road as a long line segment, with an eastern endpoint and a western endpoint. You want to place cell phone base stations at certain points along the road so that every house is within four miles of one of the base stations. Give an efficient algorithm that achieves this goal and uses as few base stations as possible.



① Algorithm:

1. walk from w to the 1st house
2. count 4 miles
3. plot a base station
4. walk again, count 4 miles, repeat

② Proof

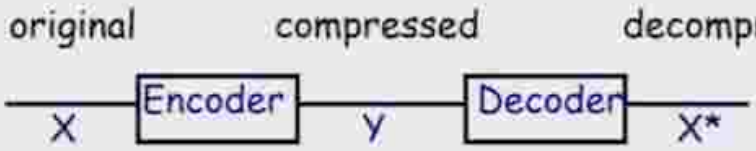


By contradiction:

if is ①, not satisfy the requirement ("within 4 m)

if is ②, will have more stations.

Basic Data Compression Conceps.



目的
- Compression -> bit reduction
- byte (char) -> codeword (bit string) ⇒ 'c' ⇒ 100101
  8 bits

种类
- Lossless compression 无损 X = X* (zip, gzip, bzip)
- Lossy compression 有损 X != X* jpg, mp3, mp4...)

Huffman Tree.

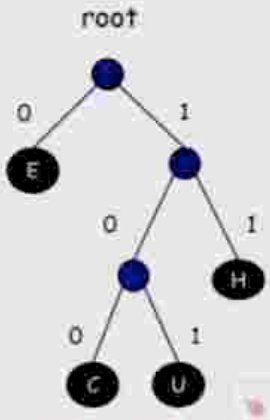
Algorithm is used to assign a prefix-free codeword to each char (byte) in the text according to their *frequencies*.

A prefix-free code is one where NO codeword is a prefix of another codeword.

A codeword is a path from the root to the character.

A codeword for C is 100.

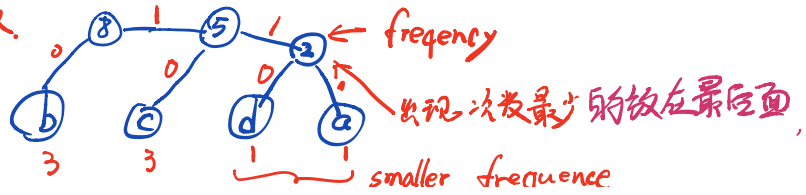
A codeword for H is 11.



* All codewords are different.

example : encode : abcbcdbc

a → 1
b → 3
c → 3
d → 1
} 出现次数



could be a = 111 (3位数)
could be b = 0
could be c = 10
could be l = 110 (3位数)

How many bits I have

3+3+6+3 = 15 bits = $\min \sum f(x_i) \cdot depth(x_i)$

3×1 1×3 2×3 3×1

original size: 8 bits × 8 = 64

abcbcdbc = 111 0 10 0 10 110 0 10 ⇐ tree

a b c b c d b c

build heap : O(n)

+Insert : O(nlogn) 总costs.

Huffman Tree

In general, we want to minimize the overall length of encoding.

$$\text{cost of tree} = MIN \sum_{k=0}^{n} f(X_k) d(X_k)$$

$f(X_k)$ ↓ frequency of x char/node.

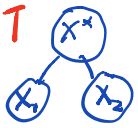
depth of x char/node ↓ is the number of bits

proof by induction on # chars

Basic case: 1.

IH: assum optimality for n-1 characters. A^*, T^* (opt)

IS: prove it for n chars. $|A| = n$, T



$A^* = A \setminus \{x_1, x_2\} \cup x^*$

$|A^*| = n-2+1 = n-1$

$cost(T) = \sum_{k=1}^{n} f \cdot d = f(x_1)d(x_1) + f(x_2)d(x_2) + \sum_{k=3}^{n} f \cdot d = cost(T^*) - f(x^*)d(x^*)$

$f(x^*) = f(x_1) + f(x_2)$, $d(x^*) = d(x_1) - 1 = d(x_2) - 1$

$Cost(T) = f(x_1)d(x_1) + f(x_2)d(x_2) + cost(T^*) - f(x_1)d(x_1) + f(x_2)$

$$+ f(x_2)\, d(x_1) + f'(x_2)$$

$$= cost(T^*) + f(x_1) + f(x_2)$$

proof of opt by contradiction.

proof that T is optimal.

Assume $\exists\ T_1$, such that $cost(T_1) < cost(T)$

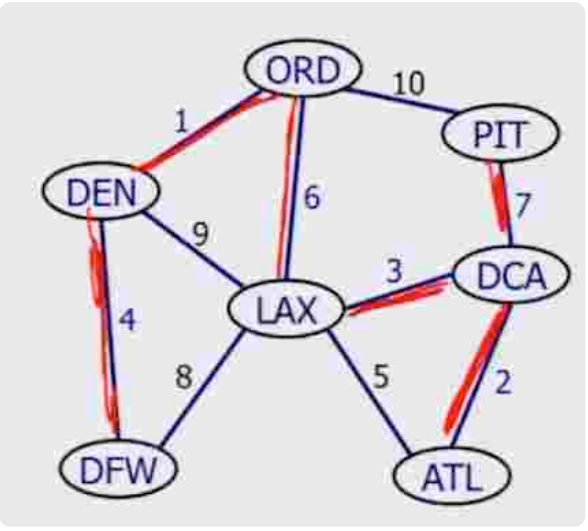
$$cost(T_1) = cost(T_1^*) + f(x_1) + f(x_2)$$

$$cost(T_1) < cost(T) \Rightarrow cost(T_1^*) < cost(T^*)$$

4.4 The minimum spanning trees.

Find a spanning tree of the minimum total weight

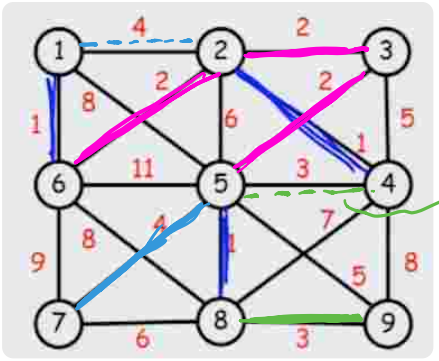
the cost is unique.



① 先找最短的

② 排除连结过的点

Kruskal's Algorithm



1. Sort edges → deletmin $O(E \log E)$
把1找出，把2找出。

2. cycle detection on each edge $O(V)$
找到4后，stop.
个有E个 $O(VE)$

Total : $O(E \log E + VE)$

T/F Questions

1. Every graph has spanning tree. F. 有的图不connected
2. A Minimum Spanning Tree is unique. F.
3. Kruskal's algorithm can fail in the presence of negative cost edges. F. sort doesn't care.

Discussion Problem 2.

Suppose we are given a graph G with all distinct edge costs. Let T be a minimum spanning tree for G. Now suppose that we replace each edge cost c_e by its square, c_e^2, thereby creating a new graph but with the different distinct costs. Prove or disprove whether T is still an MST for this new graph

{ have negetive weight : False
all positive weight edges : True.

if $c_e \to c_e + 1$, always True

Discussion Problem 3

You are given a minimum spanning tree T in a graph G = (V, E). Suppose we add a new edge (without introducing any new vertices) to G creating a new graph G_1. Devise a ~~linear time~~ algorithm to find a MST in G_1.

∵ MST = T, T has already connected all vertices

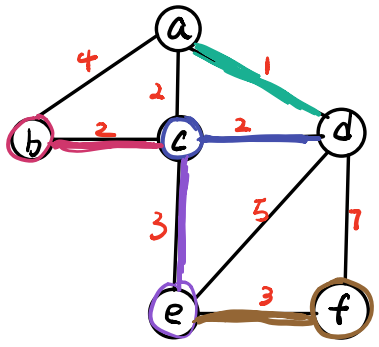
∴ if we add an edge on T, it must be cycle

find maximum edge and remove it

Prim's Algorithm

algorithm builds a tree one vertex at a time.

1. Start with an arbitrary (任意的) vertex as a sub-tree C.
2. Expand C by adding a vertex having the minimum weight edge of the graph having exactly one end point in C
3. Update distances from C to adjacent vertices.



① a →(1) d
② d →(2) c
③ c →(3) b
④ c →(3) e
⑤ e →(3) f

从一个点出发，找距离最小的点

Steps:

1. We start at a, $T = \{a\}$; build Heap $= \{d_1, c_2, b_4\}$ adjacent vertices

   weight. ef

   deletmin.

   . we pick d, $T = \{a, d\}$, $H = \{c_2, b_4\}$ e f.

2. 下一个循环, $T = \{a, d\}$, build. $H = \{c_1, b_4, e_5, f_7\}$

   pick shortest, c_1, $T = \{a, d, c\}$, $H = \{b_4, e_5, f_7\}$

3. 下一循环. $T = \{a, d, c\}$, build. $H = \{b_2, e_5, f_7\}$

   pick b_2, $T = \{a, d, c, b_2\}$, $H = \{e_5, f_7\}$

Total tree: $1+1+2+3+3 = 10$

Complexity of Prim's Algorithm

1. Build binary heap $O(v)$

Loop: a) deleteMin $O(v \log v)$

b) decreasekey $O(E \cdot \log v)$ decreasekey 的次数不可能超过 E

$O(v \log v + E \log v)$

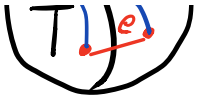
$O(E \log v)$ Usually write.

Proof of the correctness.

G.

$e' \neq e$.

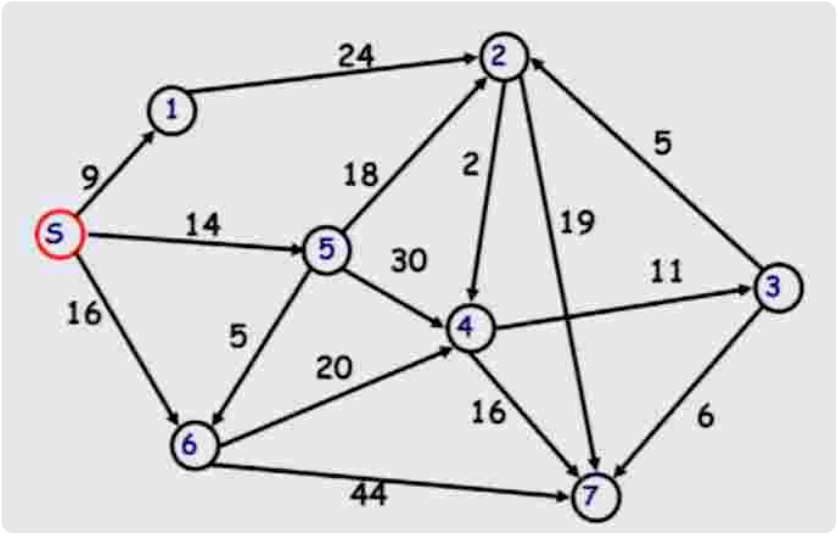
∴ take red edge, add to T



by contradiction:
assume, we don't add e, but add e', which e'>e.
then we take red edge, add to tree T, there will be a cycle, so we remove e' and add e.
the cost would be smaller, contradiction.

The shortest Path Problem ⇒ Dijkstras Algorithm

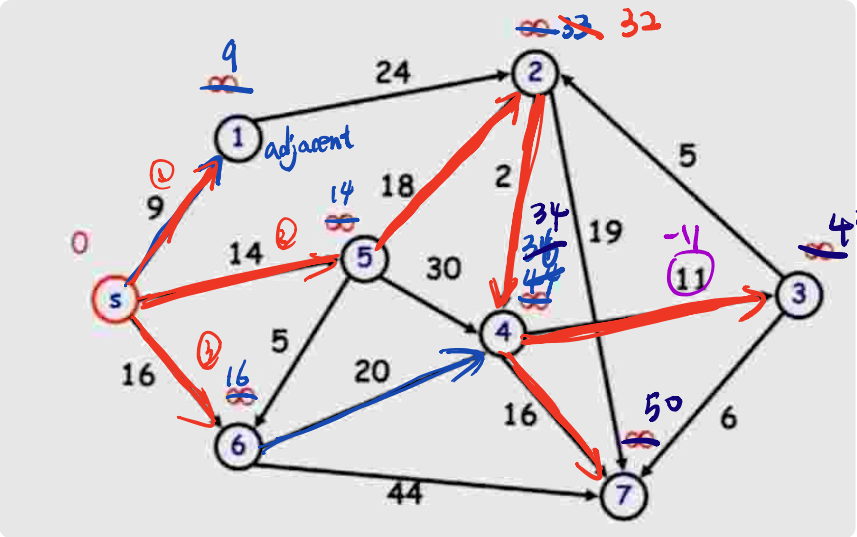
Give a positive weight graph G with a source vertex s, find the shortest path from s to all other vertices in the graph.



* note, 从一点找到另一点速度并不会变快.

start with s.

heap = {1, 2, 3, 4, 5, 6, 7}



红色为最短路

step 1，从 s 出发，到 1，5，6，距离都有值 3.

	1	2	3	4	5	6	7
s	9	∞	∞	∞	14	16	∞

step 2. 选距离最短的，1，1 到 2

	1	2	3	4	5	6	7
s	9	33	∞	∞	14	16	∞

到 5 最短.

step 3. 从 5，到 2，4，6.

	1	2	3	4	5	6	7
s	9	32	∞	44	14	16	∞

到 6 最短

Step 4，从 6，到 4，7.

	1	2	3	4	5	6	7
s	9	32	∞	34	14	16	∞

⋮

complexity.

Let D(v) denote a length from the source s to vertex v. We store distances D(v) in a binary heap.

1. Build heap : $O(v)$

Loop:

a) deleteMin $O(v \log v)$ v次.

b) decreasekey $O(E \log v)$

$O(v \log v + E \log v)$ = Prim's

different between Prim's and shortest.

heap: { Prim's : 存的 edge
shortest : 存的 distances

{ kruskal and Prim's can have negative edge.
shortest : 不能

如果有一条 edge 上的 weight 变成 negative

可能就有 cycle

Assume that an unsorted array is used instead of a binary heap. What would the algorithm's running time in this case?

be

delete_Min : $O(V \cdot V)$ 循环 V次.

decreasekey: $O(E \cdot 1)$

find and change directly through Hashmap

$\therefore T(n) = O(V^2 + E)$

is heap always better? No

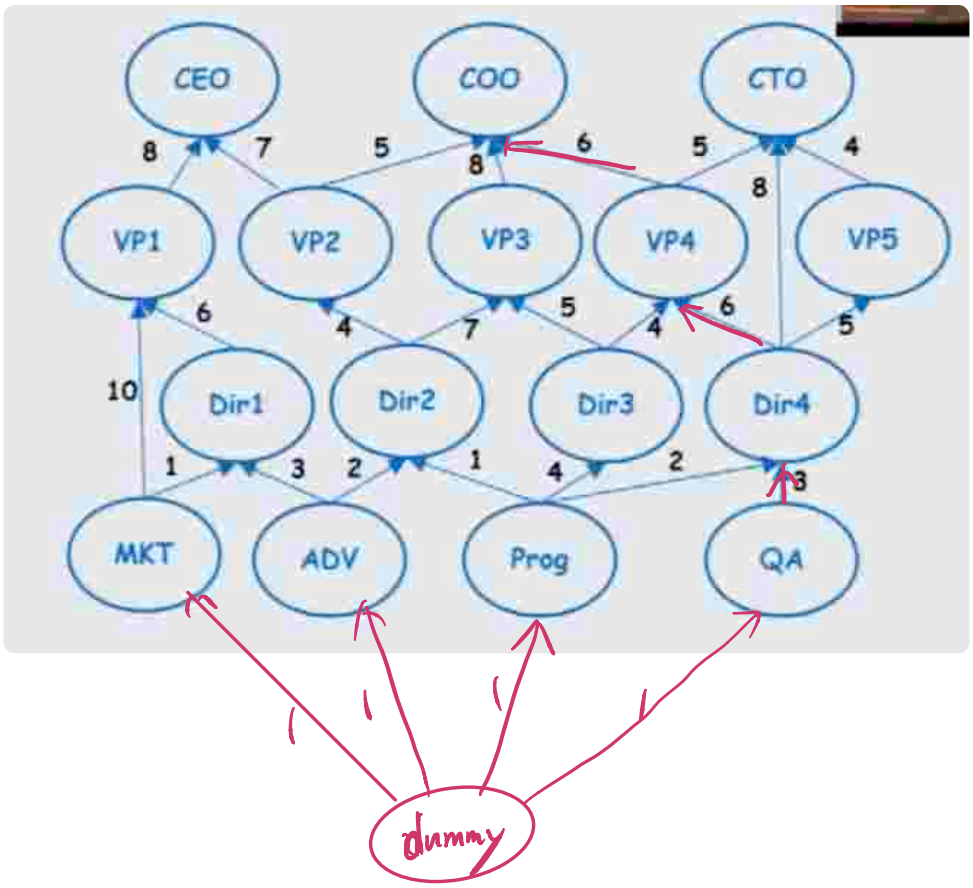
- graph is dense $E = O(V^2)$
  - array: decreaseKey, $O(V^2)$
  - heap: decreaseKey, $O(V^2 \log V)$
  - Fibonacci heap. get $O(V \log V + E \cdot 1)$ decreaseKey
- graph is sparse $E = O(V)$
  - for array, decrease $O(V^2)$
  - for heap, de.. $O(V \log V)$

不能判断.

Discussion 5

You are given a graph representing the several career paths available in industry. Each node represents a position and there is an edge from node v to node u if and only if v is a pre-requisite for u. Top positions are the ones which are not pre-requisites for any positions. Start positions are the ones which have no pre-requisites. The cost of an edge (v,u) is the effort required to go from one position v to position u. Salma wants to start a career and achieve a top position with minimum effort. Using the given graph can you provide an algorithm with the same run time complexity as Dijkstra's?

GAG

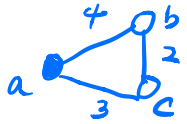


只用一次Dijkstra

Review. Question.

(T/F) The first edge added by Kruskal's algorithm can be the last edge added by Prim's algorithm.

True. ex.



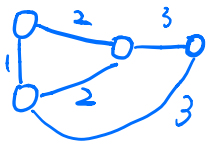
Prim's : a→c→b

Kruskal : c-b, a→c

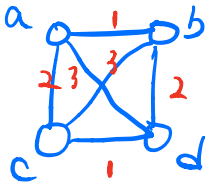
(T/F) If path P is the shortest path from u to v and w is a node on the path, then the part of p from u to w is also the shortest path.

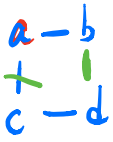
True u ~~ w ~~ v optimal.

(T/F) Suppose we have a graph where each edge weight value appears at most twice. Then, there are at most two minimum spanning trees in this graph.



4 spanning trees.



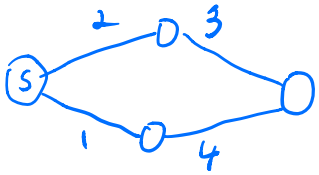


(T/F) If a connected undirected graph $G = (V, E)$ has $n = |V|$ vertices and $n + 5$ edges, we can find the minimum spanning tree of G in $O(n)$ runtime.

True, HW2. BST → Tree, deleteMax

(T/F) If all edges in a connected undirected graph have distinct positive weights, the shortest path between any two vertices is unique.

False,



5 = 2+3

5 = 1+4

(T/F) Suppose we have calculated the shortest paths from a source to all other vertices. If we modify the original graph, G, such that weights of all edges are increased by 2, then the shortest path tree of G is also the shortest path tree of the modified graph.

False

☆ shortest path 会变

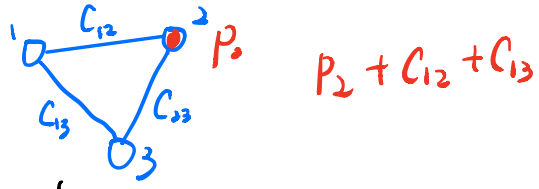
(T/F) Suppose we have calculated the shortest paths from a source to all other vertices. If we modify the original graph G such that weights of all edges are doubled, then the shortest path tree of G is also the shortest path tree of the modified graph.

True

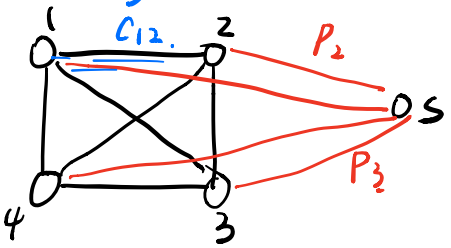
$a+b+c < x+y$

$2a+2b+2c < 2x+2y$

In this problem you are to find the most efficient way to deliver power to a network of n cities. It costs p_i to open up a power plant at city i. It costs $c_{ij} \geq 0$ to put a cable between cities i and j. A city is said to have power if either it has a power plant, or it is connected by a series of cables to some other city with a power plant. Devise an efficient algorithm for finding the *minimum cost* to power all the cities.



$p_2 + c_{12} + c_{13}$



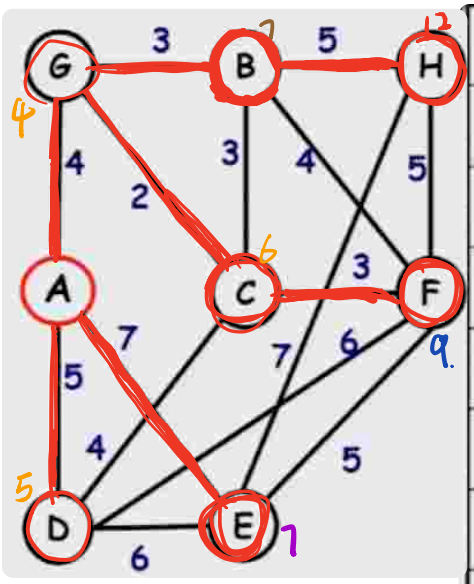
run MST

∵ prim's 无法遍历点上的 weight.

∴ 把点上的值转化到边上，添一个 supernode.

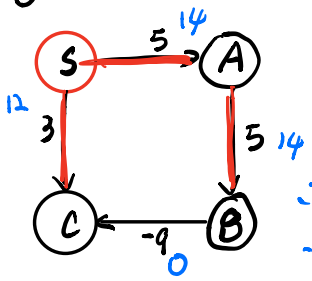
改变 input; 再执行 prims

Find a shortest path from A to each node using Dijkstra's algorithm.



	A	B	C	D	E	F	G	H
	0	∞	∞	∞	∞	∞	∞	∞
A				5	7		4	
G		7	6					
D						11		
C						9		
B								12
E								
F								
H								

Why Dijkstra's greedy algorithm <u>does not</u> work on graphs with negative weights?



d(c) = 3

S - A - B - C = 1 ← 不会执行这句

∴ Dijkstra is not the optimal choice

∵ greedy approach

S → C , S → A , A → B stop.

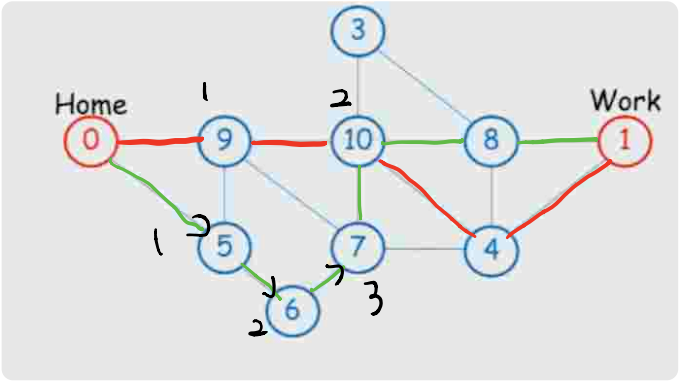
我选择了 C 之后，删掉它，我之后不会再访问到 C.

Solve①: we need to allow we can visit a node (V-1) times, 一个 node 可以进 queque V-1 times.

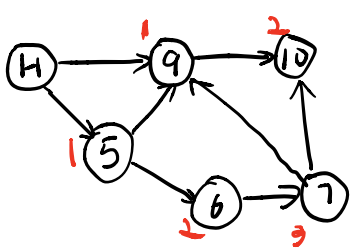
edges

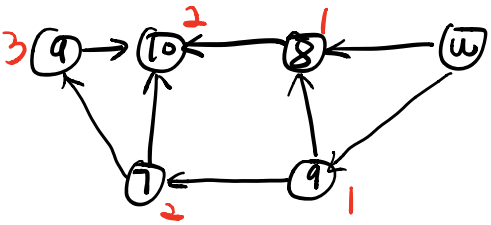
② 每条边都 +9, $s \to a \to b \to c$

Hardy decides to start running to work in San Francisco to get in shape. He prefers a route to work that goes first entirely uphill and then entirely downhill. To guide his run, he prints out a detailed map of the roads between home and work. Each road segment has a positive length, and each intersection has a distinct elevation. Assuming that every road segment is either fully uphill or fully downhill, give an efficient algorithm to find the shortest path that meets Hardy's specifications.



change inputs



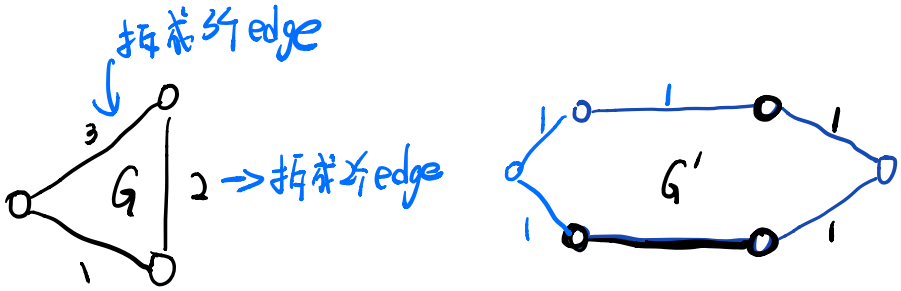


看有多少 common vertices.

- 7: 3+2=5
- 9: 1+3=4
- 10: 2+2=4

Given a graph G=(V,E) whose edge weights are integers in the range [0, W], where W is a relatively small integer number compare to the number of vertices, W = O(1). We could run Dijkstra's algorithm to find the shortest distances from the start vertex to all other vertices. Design a new linear time algorithm that will outperform Dijkstra's algorithm.

我们不能做DFS，因为边上有 weight



$G=(V,E) \Rightarrow G'=(V',E')$

$V' = V + \sum_{i=1}^{E} w_i - E$

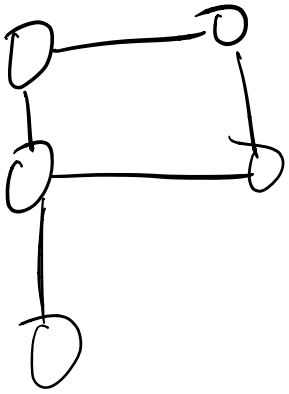
$E' = \sum_{i=1}^{E} w_i$

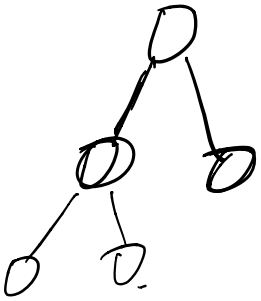
BFS

$V'+E' = V + \sum_{i=1}^{E} w_i - E + \sum_{i=1}^{E} w_i = 2\sum_{i=1}^{E} w_i + V - E$

$\leq 2E \cdot W + V - E = O(V+E)$

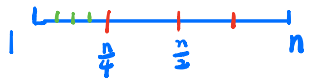
(W 是 $O(1)$)





Divide-and-Conquer Algorithms

A divide-and-conquer algorithm consists of
- dividing a problem into smaller subproblems
- solving (recursively) each subproblem
- then combining solutions to subproblems to get solution to original problem.



Binary Search

2, 5, 9, (11), 15, 27

Find (5)

Let $T(n)$ runtime complexity on input n.

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

$T(n) = O(\log n)$ BST

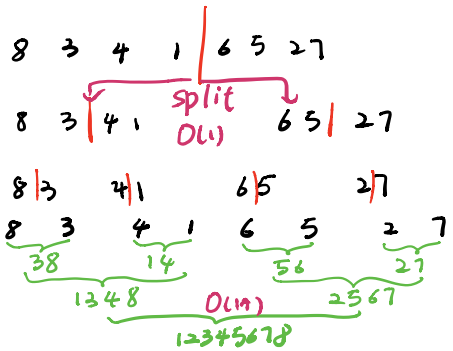
case 2, $T(n) = O(\log n)$

$a=1$ into a subproblem.

each subproblem of size $n/2$. $b=2$

$f(n) = O(1)$. ⇒ find middle and compare to decide which half.

Mergesort



$T(n) = 2T(\frac{n}{2}) + O(n)$ ← merge

$T(1) = O(1)$

$a = 2$, 2 subproblems

$b = 2$, each size $= \frac{n}{2}$

$f(n) = O(n)$. merge two sorted arrays in linear time

case 2: $f(n) = n$.

total: $T(n) = O(n \log n)$

D & C Recurrences

Suppose $T(n)$ is the number of steps in the worst case needed to solve the problem of size n.

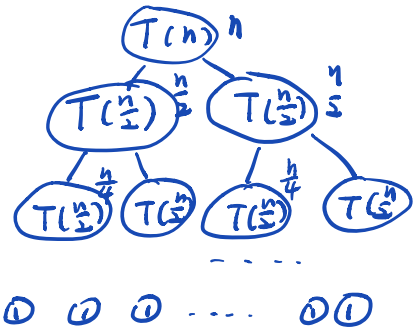
Let us divide a problem into $a \geq 1$ subproblems, each of which is of the input size n/b where $b > 1$.

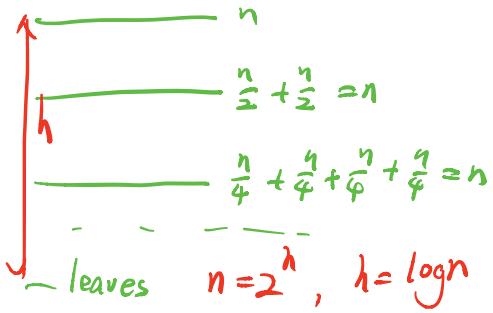
The total complexity $T(n)$ is obtained by

$$T(n) = a \cdot T(n/b) + f(n)$$

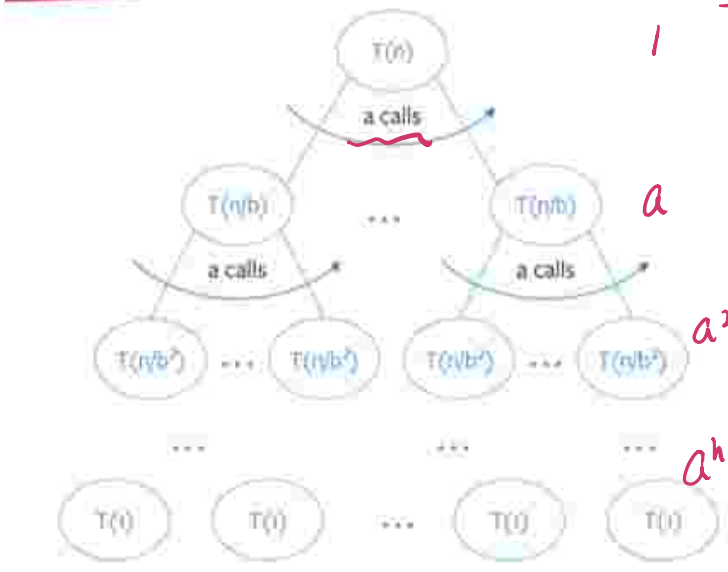
a: subproblems, n/b: size of sub.

Mergesort : tree of recursive calls





$$T(n) = h \cdot n = n \log n$$



$$T(n) = a \cdot T\left(\frac{n}{b}\right) + f(n)$$

$$T(1) = O(1)$$

$$\log_b n = \frac{\log_a n}{\log_a b} = \log_a n \log_b a$$

$$a^h = (a^{\log_a n})^{\log_b a} = n^{\log_b a}.$$

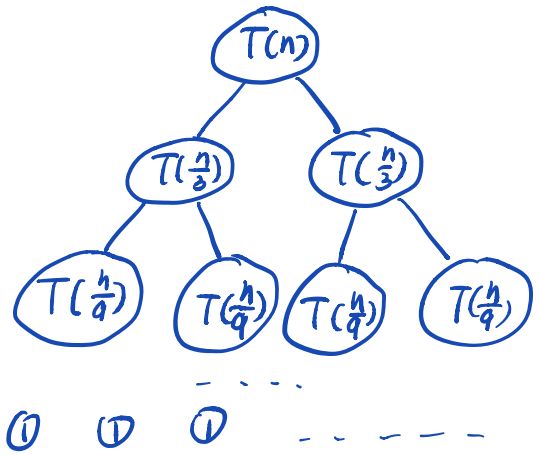
# of leaves : $a^h = a^{\log_b n} = (a^{\log_a n})^{\log_b a} = n^{\log_b a}$

$\log_b n = \frac{\log_a n}{\log_a b} = \log_a n \cdot \log_b a$ $\qquad n^{\log_b a} \Leftarrow$ leaves

Exercise:

Draw a recursive tree for: $T(n) = 2T(n/3) + O(1)$

and compute the total work. $T(1) = 1$



$a = 2$ $\qquad b = 3$

2 subproblems $\qquad$ size: $\frac{n}{3}$

$h = \log_3 n$

leaves $= n^{\log_b a} = n^{\log_3 2}$

$T(n) = \underbrace{n^{\log_3 2}}_{\text{leaves}} + \underbrace{(1+2+4+\cdots+2^{h-1})}_{\text{levels before } h.}$

↑ total work, 把每个 level 加起来

$\frac{(1-2^h)}{-1} = O(2^h)$

$\therefore T(n) = n^{\log_3 2} + O(2^h) \approx O(n)$

$T(n) = O(2^h)$

The Master Theorem

The master method provides a straightforward ("cookbook") method for solving recurrences of the form $T(n) = aT(n/b) + f(n)$

where $a \geq 1$, $b > 1$ are constants and $f(n)$ is a positive function

$c = \log_b a$:

条件.

Case 1: (only leaves) 忽略所有 internal nodes

if $f(n) = O(n^{c-\varepsilon})$, then $T(n) = O(n^c)$ for some $\varepsilon > 0$

Case 2: (all nodes) add one more log.

if $f(n) = O(n^c \log^k n)$, $k \geq 0$, then $T(n) = O(n^c \log^{k+1} n)$

Case 3: (only internal nodes) the leaves can be drop

if $f(n) = \Omega(n^{c+\varepsilon})$, then $T(n) = O(f(n))$ for some $\varepsilon > 0$

1 and 2 ; always choose the largest

if $f(n)$ > #of leaves $\Rightarrow O(f(n))$ 3.

if $f(n)$ < #of leaves $\Rightarrow O(n^c)$ 1 (#of leaves)

Discussion

1. Solve the recurrence by the Master Theory

$$T(n) = 16\,T(n/4) + 5n^3$$

$T(n) = a \cdot T(n/b) + f(n)$

$a = 16$ $f(n) = 5n^3$

$b = 4$.

$c = \log_b a = \log_4 16 = 2$

$\therefore f(n) = 5n^{2+1}$

case 1 : $n^3 = O(n^2)$ ✗

case 3: $n^3 = \Omega(n^2)$ ✓

case3: $T(n) = O(f(n)) = O(5n^3) = O(n^3)$

2. $A(n) = 3A(n/3) + 15$.

$c = \log_b a = \log_3 3 = 1$ $\quad 15 = O(n)$

$1 = n^{c-1}$

case 1, $A(n) = O(n)$

3. $B(n) = 4B(n/2) + n^3$

$c = \log_b a = \log_2 4 = 2$ $\quad n^3 = n^{c+1}$

$n^3 = n^{2+1}$

case 3, $B(n) = O(n^3)$

3. $C(n) = 4C(n/2) + n^2$

$n^2 = n^c$

$c = \log_b a = \log_2 4 = 2$.

case 2, $C(n) = O(n^2 \log n)$

4. $D(n) = 4D(n/2) + n$

$c = \log_b a = \log_2 4 = 2$.

$n = n^{c-1} = n^{2-1}$

case 3: $D(n) = D(n^c) = O(n^2)$

5. $E(n) = 4E(n/2) + n^2 \log n$

$c = \log_b a = \log_2 4 = 2$

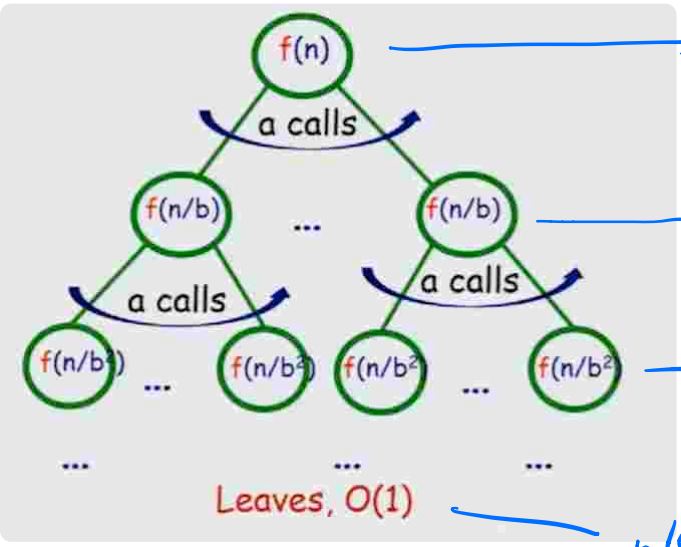
$n^2 \log n = n^{\alpha} \log n$.

case 2: $E(n) = O(n^2 \log^2 n)$

6. $F(n) = F(n/2) - n \log n$ ✗

↑ can't be negative

Proof:



$f(n)$

$a \cdot f(n/b)$

$a^2 f(n/b^2)$

$a^{h-1} f(n/b^{h-1})$

$n^{\log_b a}$ leaves

$$T(n) = n^{\log_b a} + \sum_{k=0}^{h-1} a^k f(n/b^k)$$

Case 1: $f(n) = O(n^{c-\varepsilon}) \Rightarrow f(n) \le c_1 \cdot n^{\log_b a - \varepsilon}$

$$\sum_{k=0}^{h-1} a^k f\left(\frac{n}{b^k}\right) \le c_1 \sum_{k=0}^{h-1} a^k \left(\frac{n}{b^k}\right)^{c-\varepsilon} = c_1 n^{c-\varepsilon} \sum_{k=0}^{h-1} a^k b^{k\varepsilon} \left(\frac{1}{b^k}\right)^{\log_b a}$$

$$= c_1 n^{c-\varepsilon} \sum_{k=0}^{h-1} a^k b^{k\varepsilon} \left(b^{\log_b a}\right)^{-k} = c_1 n^{c-\varepsilon} \sum_{k=0}^{h-1} b^{k\varepsilon}$$

a

$O(b^{\varepsilon h})$

$$\le c_2 n^{c-\varepsilon} b^{\varepsilon h} = c_2 n^{c-\varepsilon} n^{\varepsilon}$$

$h = \log_b n$

$$= c_2 n^c = c_2 n^{\log_b a}$$

$$T(n) = O(n^{\log_b a}) + O(n^{\log_b a}) = O(n^{\log_b a})$$

Integer Multiplication

Given two n-digit integers a and b, compute $a \times b$.

Brute force solution: $O(n^2)$ bit operations.

```
   1 2 3 4
 x 1 1 1 1
 ---------
   1 2 3 4
 1 2 3 4
 1 2 3 4
+1 2 3 4
 ---------
 1 3 7 0 9 7 4
```

$5705\,67571 = \underbrace{5705}_{x_1} \cdot 10^5 + \underbrace{67571}_{x_0}$

$a = x_1 \cdot 10^{\frac{n}{2}} + y_1$

$b = y_1 \cdot 10^{\frac{n}{2}} + y_0$

$a \times b = x_1 y_1 10^n + (x_1 y_0 + y_1 x_0) 10^{\frac{n}{2}} + x_0 y_0$

$O(n)$

$T(n) = 4T(\frac{n}{2}) + O(n)$ 加法的复杂度.

$c = \log_2 4 = 2$. $n^{c-1} = n^c$

如果变成3.

就会变快

$T(n) = O(n^2)$

Karatsuba's algorithm

Consider the product of two integers

$$(x_1 \cdot 10^{\frac{n}{2}} + x_0) \cdot (y_1 \cdot 10^{\frac{n}{2}} + y_0)$$

$x_1 y_0 + x_0 y_1 = (x_1 + x_0)(y_1 + y_0) - x_1 y_1 - x_0 y_0$ (⑤ ④ ③)

$a \times b = x_1 y_1 10^n + [\quad] 10^{n/2} + x_0 y_0$ (① ②)

$T(n) = 3 \cdot T(\frac{n}{2}) + O(n) \Rightarrow T(n) = O(n^{\log 3})$

5个乘法.

实际有3个.

3个有重复的

Exercise: 3-way spliting

The key idea is to divide a large integer into 3 parts (rather than 2) of size approximately n/3 and then multiply those parts. What would be the runtime complexity of this multiplication?

$a = x_2 \cdot 10^{\frac{n}{3}} + x_1 \cdot 10^{2n/3} + x_0$

$T(n) = 9T(\frac{n}{3}) + O(n)$ (9个相乘, 相加)

$c = \log_b a = \log_3 9 = 2$

$T(n) = O(n^2)$

570 | 567 | 571
x_2 x_1 x_0

$a \times b$ = 9 parts

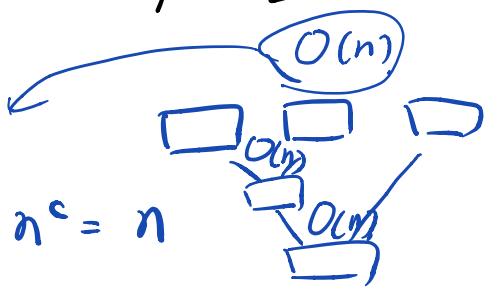
$T(n) = x \cdot T(\frac{n}{3}) + O(n)$, $T(n) = O(n^{\log_3 x})$

$\log_3 x < \log_2 3 \Rightarrow x = 5$

fastest multiplications $O(n \log n)$ FFT

3-way splitting Mergesort

The key idea is to divide an array into 3 parts of size approximatly n/3 and then sort each parts. What would be the runtime complexity of this algorithm?



$T(n) = 3T(\frac{n}{3}) + O(n)$

$c = \log_3 3 = 1$. $n^c = n$

$\therefore T(n) = O(n \log n)$

Quicsort ⇒ two part, not same size

$T(n) = T(\frac{n}{4}) + T(\frac{3}{4}n) + O(n)$

Discussion Problem

There are 2 sorted arrays A and B of size n each. Design a D&C algorithm to find the median of the array obtained after merging the above 2 arrays (i.e. array of length 2n). Discuss its runtime complexity

A = [1, 3, 5, 16, 18, 21, 30]

B = [2, 13, 17, 20, 23, 29, 35]

AUB = [1,2,3,5,13,16,17,18,20,21,23,29,30,35]

sort $O(n \log n)$ merge sort.

merging $O(n)$

goal $O(\log n)$ binary search.

compare medians. 16, 20.

16 < 20 the new median is > 16 & < 20

A_1 = [16, 18, 21, 30] B_1 = [2, 13, 17, 20]

13 < 18

A_2 = [16, 18] B_2 = [13, 17, 20]

$T(n) = 1T(n/2) + O(1)$ ↑ compare median

using pointer. lazy delete.

$T(n) = O(\log n)$

Review

1. $T(n) = 16\,T(n/4) + 5n^3 + \log n$

   $c = \log_b a = \log_4 16 = 2$

   case 1, $T(n) = O(5n^3 + \log n) = O(n^3)$

2. $T(n) = 4\,T(n/2) + n^2 \log n$

   $c = \log_b a = \log_2 4 = 2$. n^c

   case 2, $T(n) = O(n^2 \log^2 n)$

3. $T(n) = 4\,T(n/8) - n^2$ NA

   ↑ ✗

4. $T(n) = 2^n\,T(n/2) + n$ ✗ NA

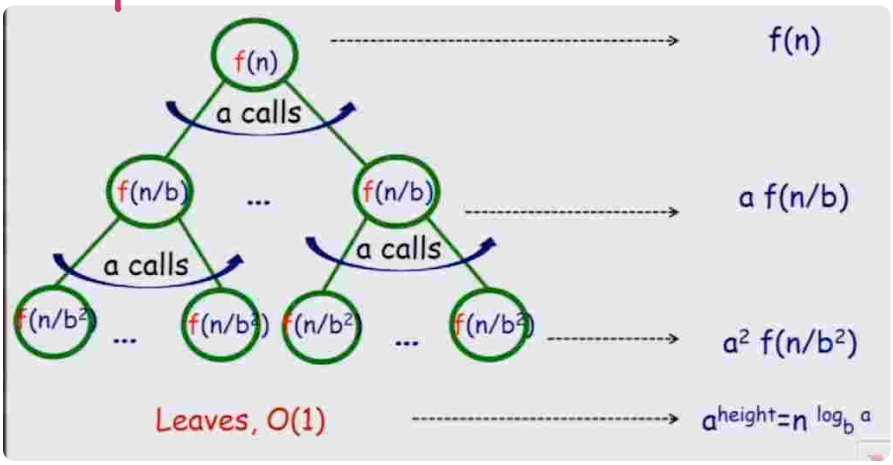
   2^n 不是 constant.

5. $T(n) = 0.2\,T(n/2) + n \log n$ ✗ NA

   ↑

   should be ≥1

Proof case 2.



$T(n) = O(n^{\log_b a}) + \sum_{k=0}^{h-1} a^k f(\frac{n}{b^k})$ where $h = \log_b n$

case 2 ($k=0$): if $f(n) = O(n^{\log_b a})$, then $T(n) = O(n^{\log_b a} \log n)$

$f(\frac{n}{b^k}) = (\frac{n}{b^k})^{\log_b a} = n^{\log_b a} (\frac{1}{b^{\log_b a}})^k = n^{\log_b a} \cdot (\frac{1}{a})^k$

$\sum_{k=0}^{h-1} a^k f(\frac{n}{b^k}) = \sum_{k=0}^{h-1} a^k \frac{n^{\log_b a}}{a^k} = n^{\log_b a} \sum_{k=0}^{h-1} 1 = h \cdot n^{\log_b a}$

$\sum_{k=0}^{h-1} 1 = O(h)$, $h = \log_b n$

$= \log_b n \cdot n^{\log_b a}$

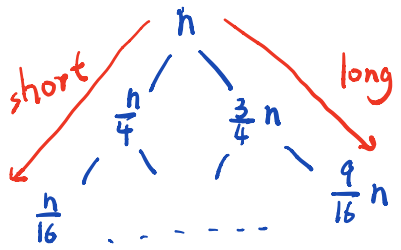
$\sum_{k=0}^{h-1} a^k f(\frac{n}{b^k})$

$\therefore T(n) = O(n^{\log_b a}) + O(\log n \cdot n^{\log_b a})$

$T(n) = O(n^{\log_b a} \cdot \log n)$

$$T(n) = T(n/4) + T(3n/4) + n$$

We design a new Mergesort algorithm in which instead of splitting in half we split it with 1 to 3 ratio. What is the runtime complexity of this approach?



height: $n \log_4 n \leq T(n) \leq n \log_{\frac{4}{3}} n$

$T(n) = 4T(n/2) + n/\log n$ 改成4, case 1, $T(n) = O(n^2)$

Case 1, $f(n) = O(n^{1-\varepsilon})$ No

$\therefore T(n) = O\left(\frac{n}{\log n}\right)$

$\frac{n}{\log n} \le c \cdot n^{1-\varepsilon} \Rightarrow n^{\varepsilon} \le c \cdot \log n$ can not be true

$\varepsilon \cdot \log n \le c_2 \log\log n$?

Case 2. No

Case 3 $f(n) = \Omega(n^{1+\varepsilon})$ No

Matrix Multiplication

A B $C = A \times B$

$$\begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix} = \begin{pmatrix} a_{11}b_{11} + a_{12}b_{21} & a_{11}b_{12} + a_{12}b_{22} \\ a_{21}b_{11} + a_{22}b_{21} & a_{21}b_{12} + a_{22}b_{22} \end{pmatrix}$$

$n \times n$ $n \times n$

$T(n) = O(n^3)$ $n \times n$?

The usual rules of matrix multiplication holds for Block matrices

$$\begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix} = \begin{pmatrix} A_{11}B_{11} + A_{12}B_{21} & A_{11}B_{12} + A_{12}B_{22} \\ A_{21}B_{11} + A_{22}B_{21} & A_{21}B_{12} + A_{22}B_{22} \end{pmatrix}$$

8 次 multiplication

D&C Algorithm (8 次 ×, 4 次 +)

Let $n = 2^k$ and M(A,B) denote the matrix product

1. if A is 1x1 matrix, return $a_{11} * b_{11}$.
2. write $A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}$ $B = \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}$

   where A_{ij} and B_{ij} are n/2 x n/2 matrices.
3. Compute $C_{ij} = M(A_{i1},B_{1j}) + M(A_{i2},B_{2j})$
4. Return $\begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix}$

Runtime complexity? $T(n) = 8\,T(\frac{n}{2}) + O(n^2)$

$\therefore T(n) = O(n^3)$

Strassen's Algorithm (7↑ ×, 18↑ +)

$$\begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix} = \begin{pmatrix} S_1+S_2-S_4+S_6 & S_4-S_5 \\ S_6+S_7 & S_2-S_3+S_5-S_7 \end{pmatrix}$$

7↑

$S_1 = (A_{12} - A_{22})(B_{21} + B_{22})$

$S_2 = (A_{11} + A_{22})(B_{11} + B_{22})$

$S_3 = (A_{11} - A_{21})(B_{11} + B_{12})$

$S_4 = (A_{11} + A_{12})B_{22}$

$S_5 = A_{11}(B_{12} - B_{22})$

$S_6 = A_{22}(B_{21} - B_{11})$

$S_7 = (A_{21} + A_{22})B_{11}$

It takes 7 multiplications

eg. $S_6 + S_7 = a_{22}b_{21} - a_{22}b_{11} + a_{21}b_{11} + a_{22}b_{11}$

$= a_{22}b_{21} + a_{21}b_{12}$

$\therefore T(n) = 7\,T(n/2) + O(n^2)$

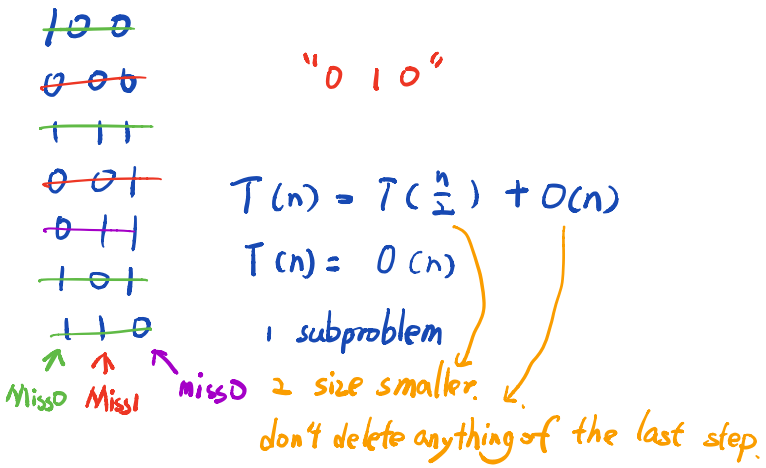
↑ 7↑ multiplications

$T(n) = O(n^{\log_2 7}) = O(n^{2.8})$

Discussion Problem

You are given an unsorted array of ALL integers in the range $[0,\ldots, 2^k-1]$ except for one integer, denoted the missing number by M.

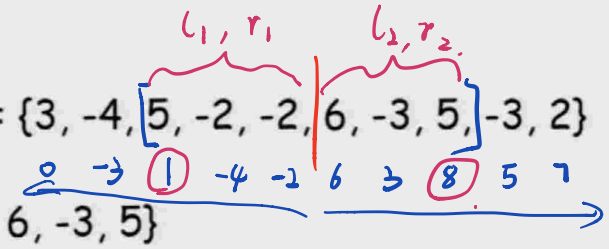
Describe a divide-and-conquer to find the missing number M, and discuss its the worst-case runtime complexity in terms of $n = 2^k$.



Finding the Maximum Subsequence Sum (subproblems overlap)

Given an array A[0,..., n-1] of integers, design a D&C algorithm that finds a subarray A[i, ..., j] such that A[i] + A[i + 1] + ... + A[j] is the maximum.

For example,



Output: {5, -2, -2, 6, -3, 5}

Sum = 5-2-2+6-3+5 = 9

left1 right1

$(l_1, r_1, max_1) = MSS(leftpart)$

$(l_2, r_2, max_2) = MSS(rightpart)$

} 从中间分开，算 $max_1 = 1$, $max_2 = 8$

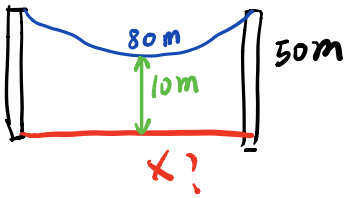
$(l_3, r_3, max_3) = span(array)$ ⇒ two parts together. $max_3 = 1+8-9$

$span(n) = O(n)$

return $MAX(max, max_1, max_3)$

$T(n) = 2T(\frac{n}{2}) + O(n)$, $T(n) = O(n \log n)$

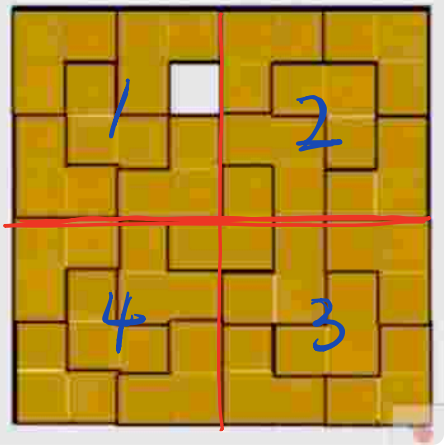
Amazon interview problem.



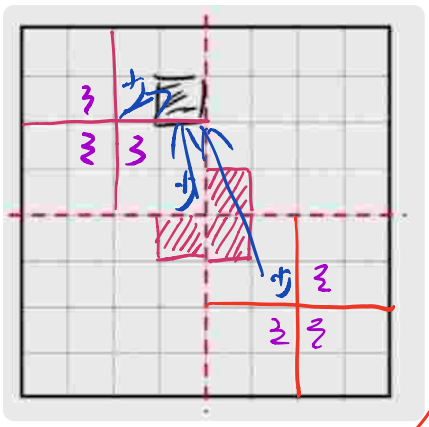
Discussion Problem

A tromino is a figure composed of three 1x1 squares in the shape of an L.

Given a $2^n \times 2^n$ checkerboard with 1 missing square, tile it with L-trominoes.
Design a D&C algorithm and discuss its runtime complexity.



find center



$T(2^n) = 4\,T(2^{n-1}) + O(1)$

$2^n = m$

$T(m) = 4\,T(\frac{m}{2}) + O(1)$

$T(m) = O(m^2)$

$\therefore T(2^n) = O(4^n)$

input size :

a) 2^n

b) n , each piece $n/4$

c) 4^n

(b). $T(n) = 2 \cdot T(\frac{n}{2}) + O(n)$

$T(n) = O(n \log n)$

(c). $T(4^n) = 4T(4^{n-1}) + O(1)$, $4^n = k$

$T(k) = 4T(\frac{k}{4}) + O(1)$

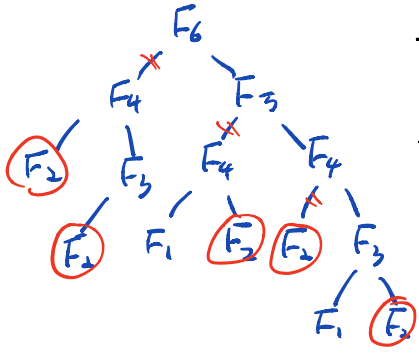
$T(k) = O(k)$

$T(4^n) = O(4^n)$

Overlapping Subproblems

exponential

Fibonacci Numbers : $F_n = F_{n-1} + F_{n-2}$, $F_0 = F_1 = 1$



$T(n) = T(n-1) + T(n-2) + O(1)$

$T(n) = O(2^n)$

$T(n) = T(n-1) + T(n-2) + O(n)$

$\log F_n = O(\log 4^n) = O(n \log 4) = O(n)$

Memorization : a top-down approach

$T(n) = T(n-1) + n$

$T(n-1) = T(n-2) + n-1$

$T(n-2) = T(n-3) + n-2$

$T(n) = n + (n-1) + (n-2) + \cdots + 1 = O(n^2)$

```
int table [50];   //initialize to zero
table[0] = table[1] = 1;

int fib(int n)
{
      if (table[n] != 0) return table[n];
      else
      table[n] = fib(n-1) + fib(n-2);

      return table[n];
}
```

use hash table

Runtime complexity ? $T(n) = T(n-1) + O(n)$

$T(n) = O(n^2)$

Tabulation : a bottom-up approach

```
int table [n];

void fib(int n)
{
      table[0] = table[1] = 1;
      for(int k = 2; k < n; k++)
         table[k] = table[k-1] + table[k-2];

      return;
}
```

$T(n) = n\,O(n) = O(n^2)$

Review Questions:

1. (T/F) For a divide-and-conquer algorithm, it is possible that the dividing step takes asymptotically longer time than the combining step.

T eg. binary search of linkedlist : divide ($>O(1)$), Combine ($O(n)$)

2. (T/F) A divide-and-conquer algorithm acting on an input size of n can have a lower bound less than $\Theta(n \log n)$.

T, binary search $O(\log n)$

3. (T/F) There exist some problems that can be efficiently solved by a divide-and-conquer algorithm but cannot be solved by a greedy algorithm.

T. eg. Merge Sort. D.C

4. (T/F) It is possible for a divide-and-conquer algorithm to have an exponential runtime. 指数.

T, Fibonacci numbers

5. (T/F) A divide-and-conquer algorithm is always recursive.

F recursive can be implement by iteration

6. (T/F) The master theorem can be applied to the following recurrence: $T(n) = 1.2\,T(n/2) + n$.

T. "a" is any number ≥ 1

7. (T/F) The master theorem can be applied to the following recurrence: $T(n) = 9\,T(n/3) - n^2 \log n + n$.

is negative

F.

8. (T/F) Karatsuba's algorithm reduces the number of multiplications from four to three.

T. $4 \to 3$

9. **(T/F)** The runtime complexity of mergesort can be asymptotically improved by recursively splitting an array into three parts (rather than into two parts).

F. don't matter how many parts you divide.

10. **(T/F)** Two $n \times n$ matrices of integers are multiplied in $\Theta(n^2)$ time.

F.

n^3, 最好是 2.4

11. **(Fill in the blank)** Let A, B be two 2×2 matrices that are multiplied using the standard multiplication method and Strassen's method.
    a. Number of multiplications (×) in the standard method: 8
    b. Number of additions (+) in the standard method: 4 $\Rightarrow 4(n^2)$
    c. Number of multiplications (×) using Strassen's method: 7
    d. Number of additions (+) using Strassen's method: 18

12. **(Fill in the blank)** The space complexity of Strassen's algorithm is: $O(n^2)$.

$(A+)(B+) = (S+)$

$S_1, \dots, S_7$ S_1 is a 乘法, 有2个+法.

一共 9个, $sp(n) = sp(\frac{n}{2}) + 9(\frac{n}{2})^2$

Exercises.

9. We know that mergesort takes $\Theta(n \log n)$ time to sort an array of items. Design a divide-and-conquer mergesort algorithm for sorting a singly linked list. Discuss its worst-case runtime complexity.

$T(n) = 2 \cdot T(\frac{n}{2}) + split(\frac{n}{2}) + merge(\frac{n}{2})$

split : 2 pointers $O(n)$

merge : compare + compine $O(n)$

$\therefore T(n) = 2T(\frac{n}{2}) + O(n) = O(n \log n)$

Dynamic Programming

General approach: in order to solve a larger problem, we solve smaller subproblems and store their values in a table.

DP is applicable when the subproblems are greatly overlap. Compare with Mergesort.

DP is not greedy either. DP tries every choice before solving the problem. It is much more expensive than greedy.

DP : two attributes

Optimal substructure means that the solution can be obtained by the combination of optimal solutions to its subproblems. Such optimal substructures are usually described recursively.

Overlapping subproblems means that the space of subproblems must be small, so an algorithm solving the problem should solve the same subproblems over and over.

0-1 knapsack Problem

Given a set of n unbreakable unique items, each with a weight w_k and a value v_k, determine the subset of items such that the total weight is less than or equal to a given knapsack capacity W and the total value is as large as possible.

w=0

items	value	weight
1	30	6
2	16	4

3	9	1
4	9	2

For greedy (by value): 30 + 16 = 46.

stop because 6+4 = 10 can add item in snack anymore

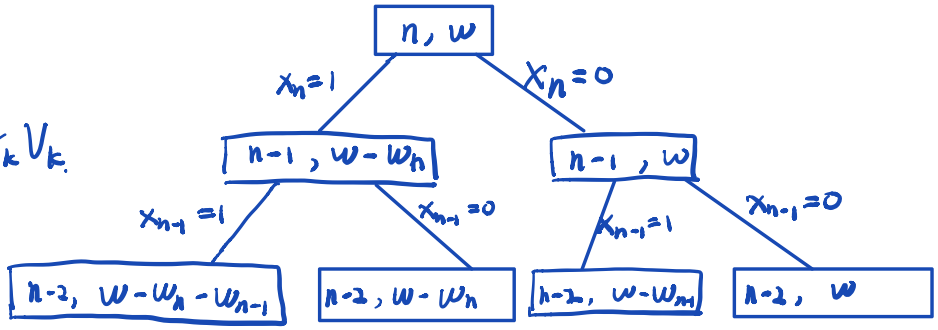
but optimal is : 30 + 9 + 9 = 48

Decision Tree

$k = 1, 2, \cdots, n$: $X_k = \begin{cases} 1 & k\text{th item is chosen} \\ 0 & k\text{th item is not chosen} \end{cases}$

$\sum_{k=1}^{n} X_k w_k \le w$

goal: $\max \sum_{k=1}^{n} x_k v_k$



Let OPT[k, x] be the max value of knapsack of $0 \le$ size $\le w$, and k item, $0 \le k \le n$.

DP Approach

solve using the following four steps:

1. Define (in plain English) subproblems to be solved.
2. Write the recurrence relation for subproblems.
3. Write pseudo-code to compute the optimal value.
4. Compute the runtime of the above DP algorithm in terms of the input size.

① Subproblems.

$OPT[k, x]$

1. $x_k = 1$ (pick kth item) ← k的value.

   $OPT[k, x] = OPT[k-1, x - w_k] + v_k$

2. $x_k = 0$ (not choose kth item)

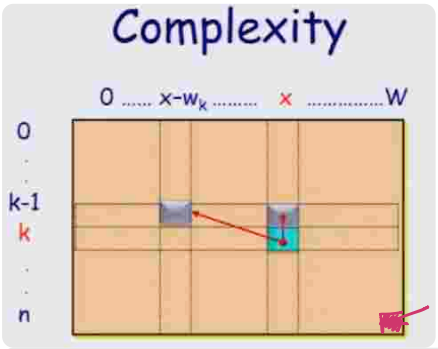
   $OPT[k, x] = OPT[k-1, x]$

② recurrence :

$OPT[k,x] = MAX[\ OPT[k-1, x-w_k] + v_k\ ,\ OPT[k-1, x]]$!!

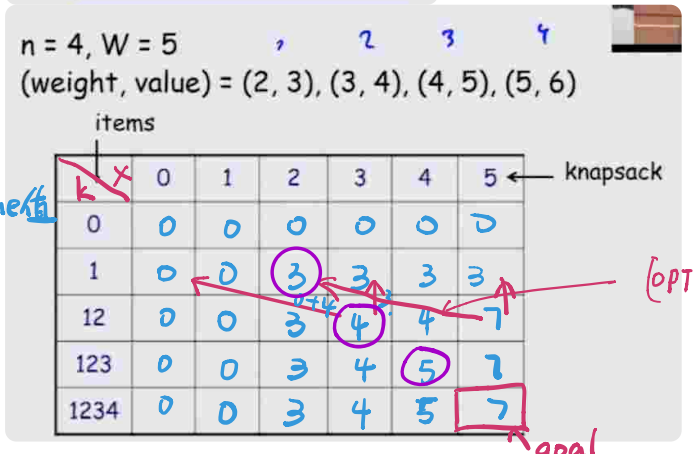
recursive must be terminated by base cases :

$OPT[k,x] = 0$, if $k=0$ or $x=0$ (stop)



Runtime Complexity ? $O(n \cdot w \cdot 1)$ ← array size

Space Complexity ? $O(n \cdot w)$ ← size log n, array of bits



k \ x	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	3	3	3	3
12	0	0	3	4	4	7
123	0	0	3	4	5	7
1234	0	0	3	4	5	7

$OPT[2,3] = \max[OPT[1,3-3]+4, OPT[1,3]] = 0$

knapsack

$OPT[2,4] = \max[\underbrace{OPT[1,5-3]+4}_{7}, \underbrace{OPT[1,4]}_{3}] = 7$

③ pseudo-code

```
int knapsack(int W, int w[], int v[], int n) {
  int Opt[n+1][W+1];                k=1
  for (k = 0; k <= n; k++) {        //k
    for (x = 0; x <= W; x++) {      x=  //x: 背包里能放入的剩余的容量.
      if (k==0 || x==0) Opt[k][x] = 0;
      if (w[x] > x) Opt[k][x] = Opt[k-1][x];    若x的weight.
      else
        Opt[k][x] = max( v[k] + Opt[k-1][x - w[k-1]],
                         Opt[k-1][x] );
    }
  }
  return Opt[n][W];
}
```

Complexity: $O(nW \cdot 1) = O(nW)$ = space complexity.

Discussion Problem

The table does not show the optimal items, but only the maximum value. How do we find which items give us that optimal value?

n = 4, W = 5

(weight, value) = (2, 3), (3, 4), (4, 5), (5, 6)

	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	3	3	3	3
2	0	0	3	4	4	7
3	0	0	3	4	5	7
4	0	0	3	4	5	7

5-3=2

① $OPT(2,5) = \max[\overbrace{opt[1,5-w_2]+v_2}^{7}, \underbrace{opt[1,5]}_{3}) = 7$

traversal this table $(O(n))$

General approach: in order to solve a larger problem, we solve smaller subproblems and store their values in a table.

DP is applicable when the subproblems are greatly overlap.

DP can be implemented by means of memoization or tabulation.

```
int table [n];
table[0] = table[1] = 1;
int fib(int n)
{
  if (table[n] != 0)
    return table[n];
  else
    table[n] = fib(n-1) + fib(n-2);
  return table[n];
}
```

Memoization:
a top-down approach.

```
int table [n];

int[] fib(int n)
{
  table[0] = table[1] = 1;
  for(int k = 2; k < n; k++)
    table[k]=table[k-1]+table[k-2];

  return table;
}
```

Tabulation:
a bottom-up approach.

Discussion Problem

You are to compute the minimum number of coins needed to make change for a given amount m. Assume that we have an unlimited supply of coins.

Denominations are denoted by d_k and $d_1 = 1$.

Let $OPT[k, x]$ be the min number of coins to give a change to x ($0 \le x \le m$) using k denominations ($1 \le k \le n$)

$OPT[k, x] = MIN(OPT[k-1, x], OPT[k, x-d_k] + 1)$

Base cases :

$OPT[1, x] = x$

$OPT[k, 0] = 0$

Complexity : $O(m \cdot n)$ ← array size; ↑ number. (size is $\log n$)

When their respective sport is not in season, USC's student-athletes are very involved in their community, helping people and spreading goodwill for the school. For the upcoming semester, we have **S** student-athletes who want to volunteer their time, and **B** buses to help get them between campus and the location of their volunteering. There are **F** projects under consideration: project **i** requires s_i student-athletes and b_i buses to accomplish, and will generate $g_i > 0$ units of goodwill for the university. Our goal is to maximize the goodwill generated for the university subject to these constraints.

Input :

1. array of projects. $(P_1, P_2, \cdots, P_F)$
2. array of buses $(b_1, b_2, \cdots b_F)$ $b_1 + b_2 + \cdots + b_F > B$
3. array of students $(S_1, S_2, \cdots, S_F)$ $S_1 + S_2 + \cdots + S_F > S$
4. array of goodwills $(g_1, g_2, \cdots, g_F]$

Let $OPT[P, b, s]$ be the max goodwill we can get from P projects $(0 \le P \le F)$ with b buses $(0 \le b \le B)$ and s students $(0 \le s \le S)$

$$OPT[P, b, s] = MAX\ (g_p + OPT[P-1, b-b_p, s-s_p],\ 0 + OPT[P-1, b, s])$$

OPT[P,b,s] = 0, if P = 0 or b = 0 or s = 0

OPT[P,b,s] = OPT[P-1, b, s], if we cannot do project

complexity : O(BFS)

constant . array size

Pseudo - Polynomial Algorithm

Definition. A numeric algorithm runs in pseudo-polynomial time if its running time is polynomial in the numeric value of the input, but is exponential in the length of the input.

0-1 Knapsack is pseudo-polynomial, $T(n) = \Theta(n \cdot W)$

```
int knapsack(int W, int w[], int v[], int n) {
  int Opt[n+1][W+1];
  for (k = 0; k <= n; k++) {
    for (x = 0; x <= W; x++) {
      if (k==0 || x==0) Opt[k][x] = 0;
      if (w[x] > x) Opt[k][x] = Opt[k-1][x];
      else
       Opt[k][x] = max( v[k] + Opt[k-1][x - w[k-1]],
                        Opt[k-1][x] );
    }
```

input size ≤ constant + logW + n·logW + nlogV + logn (< n.)

= O(nlog WV)

= O(n logW)

time complexity O(nW)

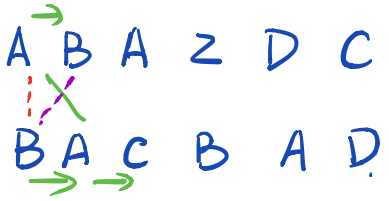
Longest Common Subsequence

We are given string S_1 of length n, and string S_2 of length m.

Our goal is to produce their longest common subsequence.

A subsequence is a subset of elements in the sequence taken in order (with strictly increasing indexes.) Or you may think as removing some characters from one string to get another.

Intuition



AB

BA

Subproblems

Let $OPT[i,j]$ be the longest subsequence of $S_1[1 \cdots i]$ and $S_2[1 \cdots j]$

Two choices

1) $S_1[i] = S_2[j]$ $\quad OPT[i,j] = OPT[i-1, j-1] + 1$

2) $S_1[i] \neq S_2[j]$ $\quad OPT[i,j] = \max(OPT[i-1,j], OPT[i,j-1])$

Base cases: $OPT[i,0] = 0$, $OPT[0,j] = 0$

complexity: $O(mn)$ polynomial ? Yes

we have m characters, input-size $= n \cdot 8 + m \cdot 8$

对 n 来说，polynomial

Example S = ABAZDC T = BACBAD O(n)

		B	A	C	B	A	D
	0	0	0	0	0	0	0
A	0	0	1	1	1	1	1
B	0	1	1	1	2	2	2
A	0	1	2	2	2	3	3
Z	0	1	2	2	2	3	4
D	0	1	2	2	2	3	4
C	0	1	2	3	3	3	4

we need

to compute

```
int LCS(char[] S1, int n, char[] S2, int m)
{
int table[n+1, m+1];
table[0...n, 0] = table[0, 0...m] = 0; //init
for(i = 1; i <= n; i++)
  for(j = 1; j <= m; j++)
     if (S1[i] == S2[j])  table[i, j] = 1 + table[i-1, j-1]
     else
     table[i, j] = max(table[i, j-1], table[i-1, j]);

return table[n, m];
}
```

Discussion Problem

A subsequence is palindromic if it reads the same left and right. Devise a DP algorithm that takes a string and returns the length of the longest palindromic subsequence (not necessarily contiguous).

2^n

For example, the string

QRAECCETCAURP

has several palindromic subsequences, RACECAR is one of them.

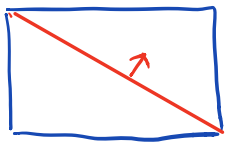
Let $OPT[i,j]$ be the longest palindromic in a substring from i to j.

$OPT[i,j] = OPT[i+1, j-1] + 2, \quad S[i] = S[j]$

$OPT[i,j] = \max[OPT[i+1,j], OPT(i,j-1)), \quad o.w.$

Base cases: $OPT(i,i) = 1$

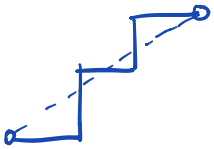
$OPT(i,i+1) = 2$, if $S[i] = S[i+1]$

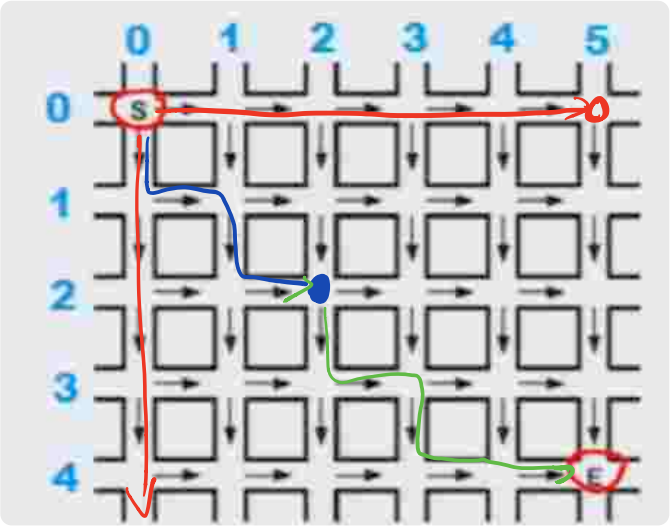


Complexity: $O(n^2)$

Discussion Problem

You are in Downtown of a city where all the streets are one-way streets. You can only go east (RIGHT) and south (DOWN) on the streets. This is called a Manhattan walk. How many unique ways are there to go from (0,0) to (n,m)? Formulate the solution to this problem as a dynamic programming problem.





$$\binom{9}{5} = \frac{9!}{5!4!}$$

$$= \frac{6 \cdot 7 \cdot 8 \cdot 9}{2 \cdot 3 \cdot 4}$$

$$= 126$$

Let $w[i,j]$ be the # of paths from $(0,0)$ to (i,j)

$w[i,j] = w[i-1,j] + w[i,j-1]$

$w[i,0] = w[0,j] = 1$

Complexity : $O(nm)$ Polynomial?

Static Optimal Binary Search Tree.

Build a binary search tree which gives a minimum search cost, assuming we know the frequencies p_i ← probability with which data k_i is accessed. The tree cannot be modified after it has been constructed.

Want to build a binary search tree with minimum expected search cost:

Expected Cost = $\sum_{i=1}^{n} P_i \, \text{depth}(k_i)$

depth (root) = 1

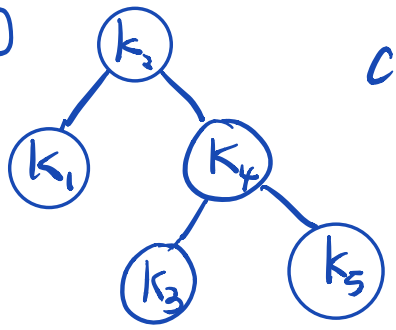
Example

Consider 5 items:

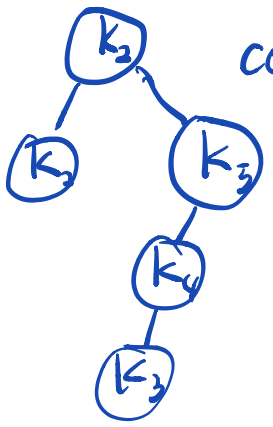
$$k_1 < k_2 < k_3 < k_4 < k_5$$

and their search probabilities

$P_1 = 0.25$, $P_2 = 0.2$, $P_3 = 0.05$, $P_4 = 0.2$, $P_5 = 0.3$

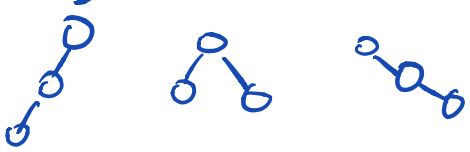
① 

$$\text{cost} = 2 \cdot 0.25 + 0.2 + 3 \times 0.05 + 2 \cdot 0.2 + 3 \times 0.3$$
$$= 0.5 + 0.2 + 0.15 + 0.4 + 0.9$$
$$= 2.15$$

② 

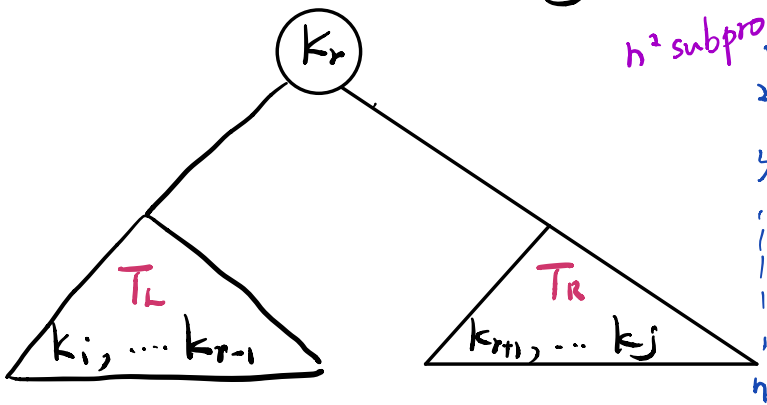
$$\text{cost} = 0.5 + 0.2 + 0.2 + 0.6 + 0.6 = 2.1$$
$$= 2.1 < 2.15$$

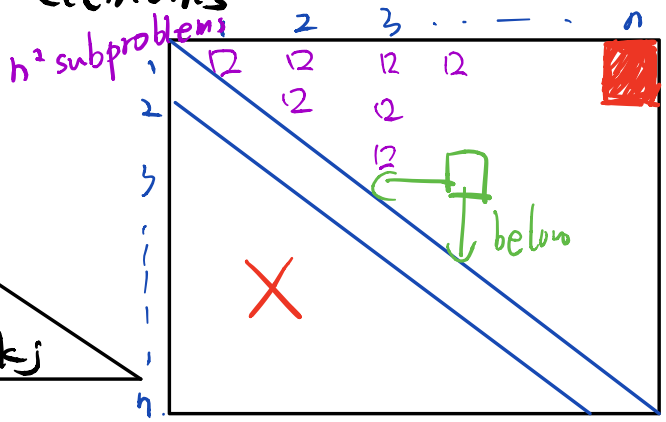
How many BST with n nodes? $\binom{2n}{n} = O(4^n)$



Optimal substructure:

Let $OPT[i,j]$ be the min cost tree built out of $k_i, k_{i+1}, \dots k_j$ elements





$k_1 \le k_2 \le \dots \le k_n$

complexity : $O(n^3)$

$$OPT[i,j] = 1 \cdot P_r + \sum_{s=i}^{r-1} P_s \, depth_T(k_s) + \sum_{s=r+1}^{j} P_s \, depth_T(k_s)$$

$$depth_T(k_s) = depth_{T_L}(k_s) + 1$$

$$OPT[i,j] = P_r + \sum P_s (depth_{T_L}(k_s) + 1) + \sum P_s (depth_{T_R}(k_s) + 1)$$

$$= P_r + (P_i + \dots + P_{r-1}) + (P_{r+1} + \dots + P_j) + \sum_{s=i}^{r-1} P_s \, depth_{T_L}(k_s) + \sum_{s=r+1}^{j} P_s \, depth_{T_R}(k_s)$$

$$OPT[i,j] = \sum_{s=i}^{j} P_s + OPT[i, r-1] + OPT[r+1, j]$$

$$OPT[i,j] = \min_{i \le r \le j} \left(\sum_{i}^{j} P_s + OPT[i, r-1] + OPT[r+1, j] \right)$$

r can be anything — takes linear time : find min

★ Base case : $OPT[i,i] = P_i, \quad i = 1, 2, \dots, n$

exam: $OPT[i, i-1] = 0, \quad i = 1, 2, \dots, n$

time complexity : There are n^2 subproblems, and each subproblem takes $O(n)$ time to compute. ∴ total $O(n^3)$

Example :

n = 5

(prob,value)=(0.25,1), (0.2,2), (0.1,3), (0.15,4), (0.3,5)

	0	1	2	3	4	5
1	0	0.25	0.65	0.9	1.3	2.15
2		0	0.2	0.4	0.8	1.45
3			0	0.1	0.35	0.9
4				0	0.15	0.6
5					0	0.3

base case 1

base case 2.

$OPT[1,1] = P_1 = P_1 = 0.25$

2.15需要所在行和所在列

斜着算.

eg. 0.9 : 0.65, 0.4, 0.25+0.1

选最小.

$$OPT[1,2] = \min_{1 \le r \le 2} (OPT[1,r-1] + OPT[r+1,2]) + P_1 + P_2.$$

$$= \min(\overbrace{0 + P_2}^{r=1}, \overbrace{P_1 + 0}^{r=2}) + P_1 + P_2 = 0.2 + 0.25 + 0.2 = 0.65$$

$$OPT[2,3] = \min_{2 \le r \le 3} (OPT[2,r-1] + OPT[r+1,3]) + P_2 + P_3$$

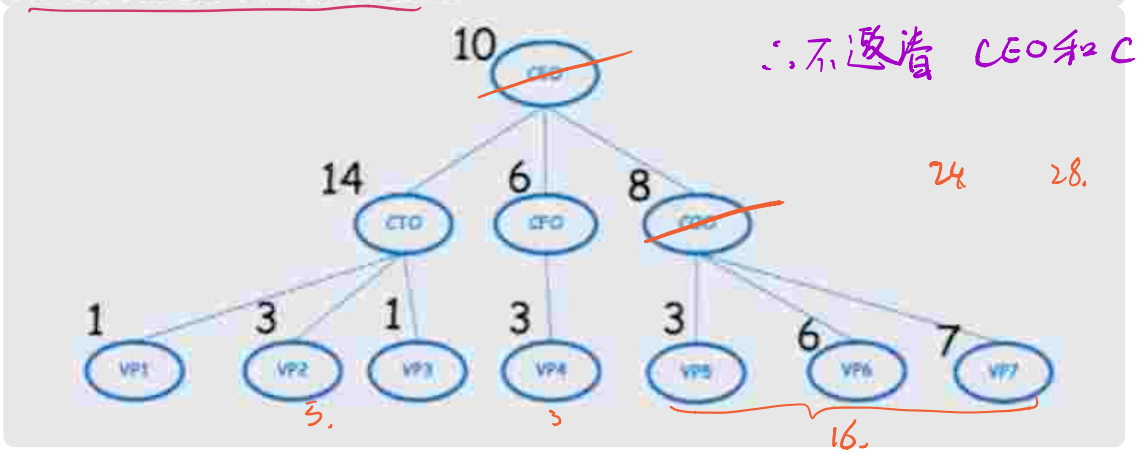
$$= \min(0 + P_3, OPT[1,2] + 0) + P_2 + P_3 = 0.1 + 0.2 + 0.1 = 0.4$$

$$OPT[3,4] = \min_{3 \le r \le 4} (OPT[3,r-1] + OPT[r+1,4]) + P_3 + P_4$$

$$= \min(0 + P_4, \underset{0.1}{P_3} + 0) + P_3 + P_4 = 0.1 + 0.1 + 0.15 = 0.35$$

Discussion Problem

Suppose you are organizing a company party. The corporation has a hierarchical ranking structure; that is, the CEO is the root node of the hierarchy tree, and the CEO's immediate subordinates are the children of the root node, and so on in this fashion. To keep the party fun for all involved, you will not invite any employee whose immediate superior is invited. Each employee j has a value v_j (a positive integer), representing how enjoyable their presence would be at the party. Our goal is to determine which employees to invite, subject to these constraints, to maximize the total value of invitees.



∴不选着 CEO和COO

24 28.

Let OPT[node] be the MAX joy of a subtree rooted of this node.

choices:

1. invite "node": $OPT[node] = V_{node} + \sum_g OPT[g]$

2. otherwise: $OPT[node] = \sum_c OPT[c]$

1. invite CEO, 34 10 + 5 + 3 + 16

2. donot, 36, 14 + 6 + 3 + 6 + 7 = 36.
grandchildren children

$OPT[node] = MAX(V_{node} + \sum_g OPT[g], \sum_c OPT[c])$

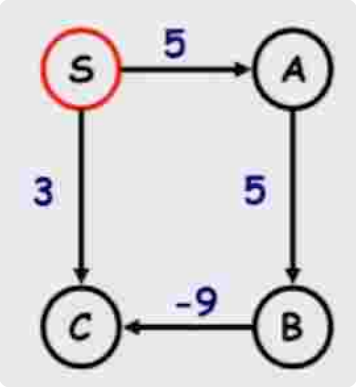
Base case:

OPT[NULL] = 0

How do you fill the table? DFS, post-order. LRP

Complexity: $O(n^2)$ n 个 node，每个 $O(n)$ $\Rightarrow O(n^2)$

The shortest Path Problem $O(VE)$



Dijkstra's greedy algorithm does not work on graphs with negative weights.

Intuition.

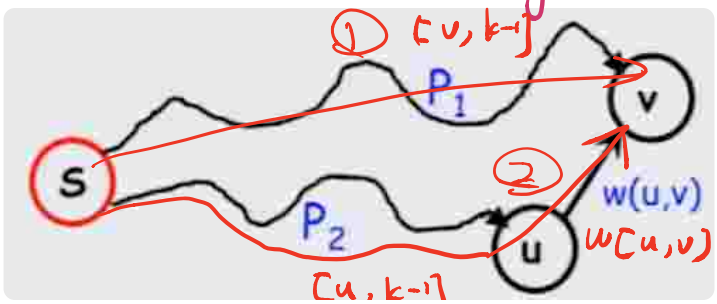
Consider the shortest path

$$v = w_0, w_1, \cdots w_{k-1}, w_k = u$$

To have an optimal substructure the following path. $v = w_0, w_1, \cdots, w_{k-1}$

must be the shortest path to w_{k-1}.

The bellman-Ford Algorithm



P_1 & P_2 has at most k edges

how to create a path of (k+1) edges

Let D[v, k] be the shortest pat from s to v with at most k edges.

$$D[v,k] = \min[D[v,k-1], \min_{\text{adjcent } w}(cost(w,v) + D[w,k-1])]$$

Base case : D[s,0]=0 D[v,0] = ∞

Implementation

```
D[v,k] denotes the length of the shortest path
from s to v that uses at most k edges

  D[v,0] = INFINITY; v != s
  D[s,k] = 0; for all k

V for k=1 to V-1:
  V for each v in V:
    E for each edge (u,v) in E:
        D[v, k] = min(D[v,k-1] , w(u,v) + D [u,k-1])
```

k=V 时 若是3，有 negative cycle

edge adjacent to v.

E

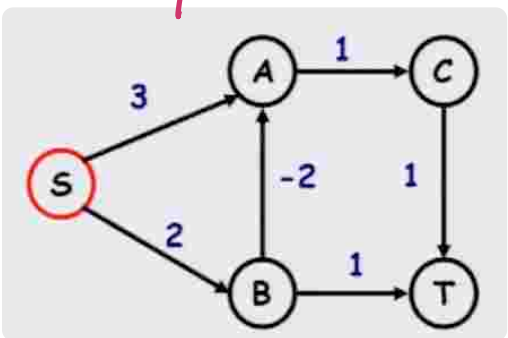
Complexity : $O(V^2E)$, do better?

$O(V \cdot E)$ → 如果 graph is undirected. $O(V \cdot 2E)$

$O(E \cdot \log V)$ → Dij

different.

Example.



k=1 : D[A] = 3, D[B] = 2

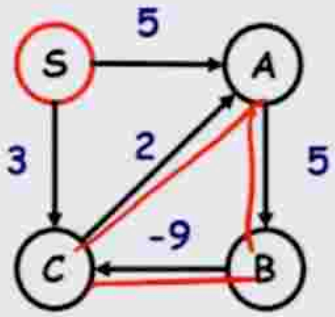
k=2 : D[A] = 0, D[B] = 2

D[C] = 4 D[T] = 3

k=3 : D[C] = 1

k=4 : D[T] = 2

How would you apply the Bellman-Ford algorithm to find out if a graph has a negative cycle?



$k = V$

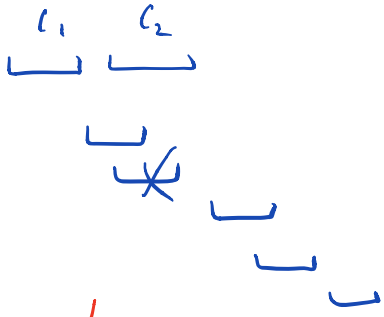
Do not stop V-1 iterations, perform extra round, let k=V, if anything changes in the table, there we know there is a negative cycle.

Discussion Problem

You are to plan the spring 2020 schedule of classes. Suppose that you can sign up for as many classes as you want, and you'll have infinite amount of energy to handle all the classes, but you cannot have any time conflict between the lectures. Also assume that the problem reduces to planning your schedule of *one particular day*. Thus, consider one particular day of the week and all the classes happening on that day: $c_1, \ldots, c_n$. Associated with each class c_i is a start time s_i and a finish time f_i and you also assign a score v_i to that class based on your interests and your program requirement. Assume $s_i < f_i$. You would like to choose a set of courses C for that day to maximize the total score. Devise an algorithm for planning your schedule.

sort classes by f_i, $O(n \log n)$ ☆ already sorted.

Let $OPT[i]$ be the max score achived from $c_1, c_2, \dots c_i$ classes.



Cases:

1. do not take c_i.

$$OPT[i] = OPT[i-1]$$

2. take c_i

$$OPT[i] = v_i + OPT[g(i)]$$ ← function

$$g(i) = \max\{j \mid f_j < s_i\}$$ ← binary search

f_j s_i.

Base case: $OPT[0] = 0$

Complexity: $O(n^2)$ size of table is n, find max of 2 value is $O(1)$, but, to find value, we need to do sequential search, worst case size of input.

can we do better? Binary Search $O(\log n)$

$O(n \log n)$

All-pairs, Shortest Path Problem.

Can we use Bellman-Ford's formulation for the shortest path subproblem?

$D[v, u, k]$: distance from vertex v to u using k edges

runtime complexity: $O(V^2 \cdot E)$ count vertexs

The Floyd-Warshall Algorithm

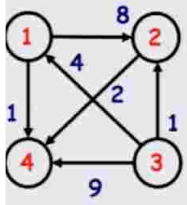
enumerate all vertices: 1, 2, 3, ... n

Let D[i,j,k] be the shortest path from i to j with intermediate vertices from {1, 2, ..., k}

D[i,j,0] — adjcent matrix

Example :

Consider a path from 3 to 4, using {1, 2, ..., k} as intermediate vertices.



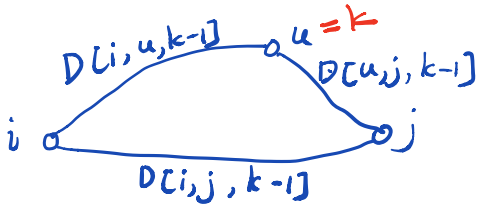
k=0, 9

k=1, min (5,9)=5

k=2, path 3-1-2-4 = 14

3-2-4 =3

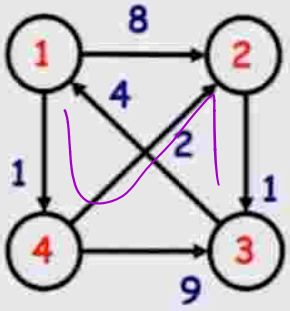
Recurrence Relation



D[i,j,k]

$$D[i,j,k] = \min_u [D[i,j,k-1], D[i,u,k-1] + D[u,j,k-1]]$$

Example



k=0 ⇒ adjcent matrix

	1	2	3	4
1	0	3	8	1
2	5	0	1	6
3	4	12 7	0	5
4	7	2	3	0

k=1

k=2

k=3

k=4

k=1, 2-1-3 ✗ 3-1-2 ✓ 4-1-2 ✗

2-1-4 ✗ 3-1-4 ✓ 4-1-3 ✗

k=2, 1-2-3 ✓ 3-2-1 4-2-1

1-2-4 3-2-4 4-2-3

k=3 1-3-2 ✓ (9+12)

1-3-4 ✓ (9+5)

2-3-1 ✓

4-3-1 ⇒ 7

k=4 1-4-3 ✓ ⇒ 4

1-4-2 ✓ 2-3-1

D[i,j,0] = c[i,j] for all i and j,

B-F : $O(V^2E)$

F-W : $O(V^3)$

} 不一定比较, 取决于图的形

```
for k=1 ... V do
  for i=1 ... V do
    for j=1 ... V do
      D[i,j,k] = min (D[i,j,k-1], D[i,k,k-1] + D[k,j,k-1] )
```

D[i,i,k] < 0 , inside outer loop

Actually, we don't need a separate matrix for each k: we will overwrite the existing values

Question: How do we handle the possibility that the graph has a negative cycle?

D[1, 1, 3] < 0 , then have negative cycle

Question: How do we extract the shortest path?

P[i,j] = k

↑ k 的值

	1	2	3	4
1	0	4	4	0
2	3	0	0	3
3	0	4	0	1
4	3	0	2	0

find path (i,j)

if p(i,j) = 0, then output (i,j)

else { find path (i, p(i,j));

find path (p(i,j), j);

}

path from 1 to 3.

1 to 3

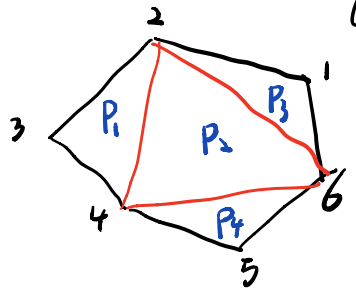
1 to 4 | 4 to 3.

stop 0 (1,4)

4 to 2 | 2 to 3

stop (4,2)

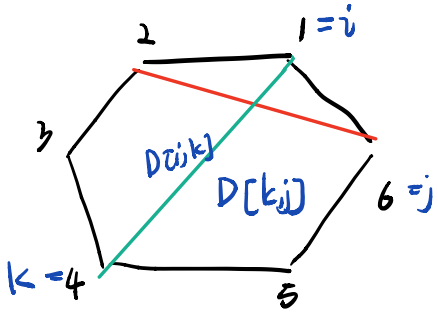
stop. (2,3) ← edge



Goal : minmize perlimeter of each triangles.

Solve with DP.

D[i,j] be the min triangulation for 三角划分.
(i, ··· j)

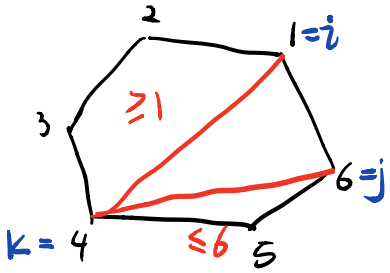


$D[i,j] = \min_{i,k} [D[i,k] + D[k,j]]$

$D[i,i+1] = \infty$

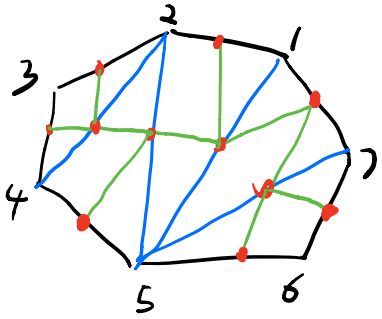
D[i,i+2] = perimeter 周长.

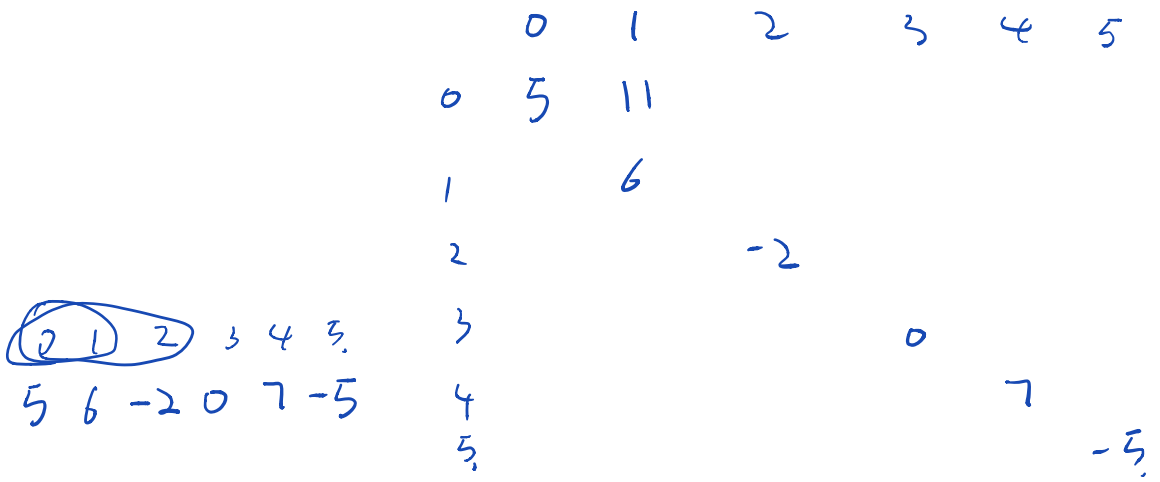
time complexity : $O(n^2)$ all combination.



$D[i,j] = \min_{k} [D[i,k] + D[k,j] + \text{perimeter}]$

Time complexity : $O(n^3)$




0 1 2 3 4 5
0 5 11
1 6
2 -2
3 0
4 7
5 -5
2 1 2 3 4 5
5 6 -2 0 7 -5