Algorithm Class Notes

Danny Wang

Autumn, 2016

1 Introduction & Notation *September 13, 2016*

1.1 Class Information

- 4 homeworks (before 10:20 *am* in the class)
- 1 mid, 1 final (A4 paper cheat sheet)
- There is no class on *Dec. 13th*
- Collaboration Policy is IMPORTANT !
- Course announcement will be available on CEIBA.

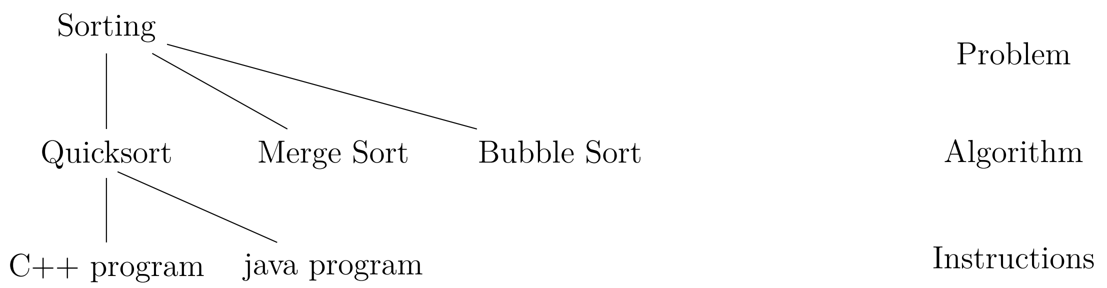
1.2 What is Algorithm ?

Algorithm: A **step-by-step** procedure to solve *problem*s.

Problem: Input $\rightarrow$ Output

Example. multiplication $a, b \rightarrow a \times b$

Example. sorting $a_1, a_2, ..., a_n \rightarrow b_1 \leq b_2 \leq,,, \leq b_n$



There are many important features for a program! (not just running time)

For exmaple: Scalability, resource allocation, user friendliness, readability, robust.

1.3 Why study algorithms ?

1. Performance determines feasibility.
2. Provide a language to talk about program behaviors.
3. Generalize to other resources.
4. Fun !

1.3.1 Example: Multiplication

Computing :

$$1234 \times 4321$$

Question: How many single-digit multiplication is required ?

```
T(n) = # of single-digit multiplication
       required to multiply 2 n-digit-#s
```

$$\begin{array}{cccccc} & & 1 & 2 & 3 & 4 \\ \times & & 4 & 3 & 2 & 1 \\ \hline & & 1 & 2 & 3 & 4 \\ & 2 & 4 & 6 & 8 & \\ . & . & . & . & & \\ \hline \end{array} \qquad \begin{array}{l} \Rightarrow T(4) \\ \Rightarrow T(n) = n^2 \end{array}$$

Question: fewer ?

idea 1. Use recursion. For n-digit numbers x, y :

$$x = a \times 10^{\frac{n}{2}} + b$$
$$y = c \times 10^{\frac{n}{2}} + d \qquad a, b, c, d : \frac{n}{2}\text{-digit numbers.}$$
$$\text{then} \quad xy = ac \times 10^n + (ad + bc) \times 10^{\frac{n}{2}} + bd$$

Example.

$$1234 = 12 \times 10^2 + 34$$

$$4321 = 43 \times 10^2 + 21$$

Compute $xy \Leftrightarrow$ compute ac, ad, bc, bd and shift digits, sum up.

$$T(n) = 4T(\frac{n}{2}) \;\Rightarrow\; T(n) = n^2$$

idea 2. (Karatsuba, 1962)

$$\begin{aligned} xy &= ac \times 10^n + (ad + bc) \times 10^{\frac{n}{2}} + bd \\ &= ac \times 10^n + [(a+b)(c+d) - (ac+bd)] \times 10^{\frac{n}{2}} + bd \end{aligned}$$

We only need to compute $(a+b)(c+d), ac, bd$!

$$T(n) = 3T(\frac{n}{2}) \Rightarrow T(n) \propto n^{1.585}$$

idea 3. Extensions: Toom-Cook scheme about $n^{1.465}$

idea 4. FFT: $n \log n \log \log n$
May be $n \log n$ in recent research.

Whether exists a better algorithm is a much harder problem in usual.

1.3.2 Example: Sequence alignment

```
Input:   2  sequences
            (1) CGTTCAT
            (2) CGTTAC
Output: Similarity score

Alignments:
      C G T T C A T
      C G T T   A C
```

There are gaps or mismatches.

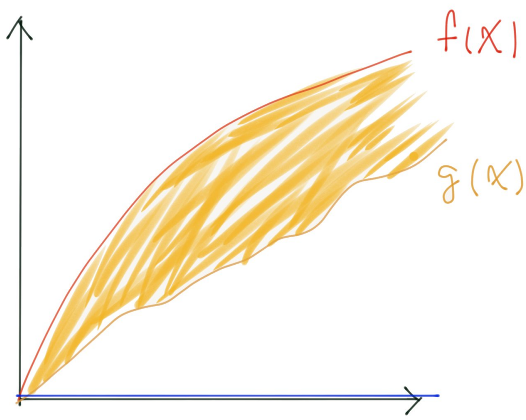
score $= C_1($ # of gaps $) + C_2($ # of mismatches $)$

Find the min score in all alignments.
one mismatch is better than two gaps.
human genome: 3×10^9 bases.

idea 1. (brute force) Try all alignments $\rightarrow$ pick the min score.
For length $200 \sim 300$, takes longer than age of universe.

idea 2. Greedy.
We can get one answer fast.

1.4 Running time (time complexity) of an algorithm



1. Worst case complexity.
   guarantee.
   used in the class unless otherwise specified.

2. Best-case complexity
   can cheat. (blue line, always 1)

3. Average-case complexity
   can depend on input distribution.
   randomized algorithms

1.5 Notation for complexity

1.5.1 Big-O Notation

Definition: (non-neagtive functions only)

$$O(g(n)) = \{ \ h(n) \mid \ \exists \ c, n_0 > 0 \quad \text{s.t.} \quad h(n) \leq c \cdot g(n), \ \forall n \geq n_0 \ \}$$

$$f(n) \in O(g(n))$$

$$f(n) = O(g(n))$$

Example. $f(n) = 5n + 10 \Rightarrow f(n) = O(n)$

Proof. Need to find c, n_0 such that

$$f(n) \leq cn, \ \forall n \geq n_0$$
$$\Leftrightarrow 5n + 10 \leq cn, \ \forall n \geq n_0$$

Pick $c = 6$, $n_0 = 10$ suffices. □

Example. $10^{10}n + 100^{100} \in O(n)$

Example. $n^2 \notin O(n)$

Proof. Need to show for any given c, n_0

$$n^2 \leq cn, \ \forall n \geq n_0 \text{ is never true.}$$

For any given c, n_0, pick $n_1 = max(2c, n_0)$

$$cn_1 \ngeq n_1^2 \geq 2cn_1$$

$\square$

1.5.2 Big-Omega Notation

$$\Omega(g(n)) = \{\ h(n) \mid \ \exists\ c, n_0 > 0 \quad \text{s.t.} \quad c \cdot g(n) \leq h(n), \ \forall n \geq n_0 \ \}$$

1.5.3 Big-Theta Notation

$$\Theta(g(n)) = \{\ h(n) \mid \ \exists\ c_1, c_2, n_0 > 0 \quad \text{s.t.} \quad c_1 \cdot g(n) \leq h(n) \leq c_2 \cdot g(n), \ \forall n \geq n_0 \ \}$$

$$\Theta(g(n)) = O(g(n)) \cap \Omega(g(n))$$

1.5.4 little-o Notation

$$o(g(n)) = \{\ h(n) \mid \ \forall c > 0, \ \exists\ n_0 > 0 \quad \text{s.t.} \quad h(n) \leq c \cdot g(n), \ \forall n \geq n_0 \ \}$$

$$f(n) \in o(g(n)) \ \text{ iff } \quad \lim_{n \to \infty} \frac{f(n)}{g(n)} = 0$$

1.5.5 little-omega Notation

$$\omega(g(n)) = \{\ h(n) \mid \ \forall c > 0, \ \exists\ n_0 > 0 \quad \text{s.t.} \quad c \cdot g(n) \leq h(n), \ \forall n \geq n_0 \ \}$$

$$f(n) \in \omega(g(n)) \ \text{ iff } \quad \lim_{n \to \infty} \frac{f(n)}{g(n)} = \infty$$

Note: For all functions above we only consider where $f(n), g(n), h(n) \geq 0$

Rule. 1 $f(n) = O(f(n))$

Rule. 2 If c is a positive constant then $c \cdot O(f(n)) = O(f(n))$

Rule. 3 If $f(n) = O(g(n))$ then $O(f(n)) = O(g(n))$

Rule. 4 $O(f(n)) \cdot O(g(n)) = O(f(n) \cdot g(n))$

Rule. 5 $O(f(n) \cdot g(n)) = f(n) \cdot O(g(n))$

2 Notation properties and Recurrence *September 20, 2016*

2.1 Recap: Asymptotic Notations

Example. $\sqrt{n} \in o(n)$

Proof. Given any $c > 0 \Rightarrow$ can find n_0 s.t. $\sqrt{n} \leq cn,\ \forall n \geq n_0$

$$\Leftrightarrow 1 \leq c\sqrt{n},\ \forall n \geq n_0$$

$$\Leftrightarrow \sqrt{n} \geq \frac{1}{c},\ \forall n \geq n_0$$

$$\Leftrightarrow n \geq (\frac{1}{c})^2,\ \forall n \geq n_0$$

so we will set $n_0 = (\frac{1}{c})^2$, and then go through the definition:

$$\Rightarrow n \geq (\frac{1}{c})^2,\ \forall n \geq n_0$$

$$\Rightarrow cn \geq \sqrt{n},\ \forall n \geq n_0$$

□

Proof. (Alternative)

$$\lim_{n\to\infty} \frac{\sqrt{n}}{n} = 0$$
$$\Rightarrow \sqrt{n} \in o(n)$$

□

Example. $\frac{1}{2}n \notin o(n)$

Proof. (Alternative)

$$\lim_{n\to\infty} \frac{\frac{1}{2}n}{n} = \frac{1}{2}$$
$$\Rightarrow \frac{1}{2}n \notin o(n)$$

□

Intuitively:

$$f(n) \in O(g(n)) \quad \text{means} \quad \text{"}f \leq g\text{"}$$

$$f(n) \in \Omega(g(n)) \quad \text{means} \quad \text{"}f \geq g\text{"}$$

$$f(n) \in \Theta(g(n)) \quad \text{means} \quad \text{"}f \approx g\text{"}$$

$$f(n) \in o(g(n)) \quad \text{means} \quad \text{"}f < g\text{"}$$

$$f(n) \in \omega(g(n)) \quad \text{means} \quad \text{"}f > g\text{"}$$

Properties:

(1) Transitivity:

$$f(n) \in O(g(n)),\ g(n) \in O(h(n)) \Rightarrow f(n) \in O(h(n))$$

$$f(n) \in o(g(n)),\ g(n) \in o(h(n)) \Rightarrow f(n) \in o(h(n))$$

and it is true for the other three Notations !

(2) Reflexivity

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

(3) Symmetry

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

(4) Transpose Symmetry

$$f(n) \in O(g(n) \Leftrightarrow g(n) \in \Omega(f(n)))$$

$$f(n) \in o(g(n) \Leftrightarrow g(n) \in \omega(f(n)))$$

Proof. (1) O by definition.

$$f \in O(g) \Rightarrow \exists\ c_1, n_1 \quad \text{s.t.} \quad f(n) \leq c_1 \cdot g(n),\ \forall n \geq n_1$$

$$g \in O(h) \Rightarrow \exists\ c_2, n_2 \quad \text{s.t.} \quad g(n) \leq c_2 \cdot h(n),\ \forall n \geq n_2$$

$$\forall n \geq \max(n_1, n_2) = n_0,\ f(n) \leq c_1 g(n) \leq c_1 c_2 h(n) = ch(n)$$

$$\Rightarrow f \in O(h)$$

□

Proof. (1) o by definition.

$$\text{Given any } c > 0, \quad \text{Let} \quad c_1 = c_2 = \sqrt{n}$$

$$f \in o(g) \Rightarrow \exists\ n_1 \quad \text{s.t.} \quad f(n) \leq c_1 \cdot g(n),\ \forall n \geq n_1$$

$$g \in o(h) \Rightarrow \exists\ n_2 \quad \text{s.t.} \quad g(n) \leq c_2 \cdot h(n),\ \forall n \geq n_2$$

$$\forall n \geq \max\{n_1, n_2\} = n_0,\ f(n) \leq c_1 g(n) \leq c_1 c_2 h(n) = ch(n)$$

$$\Rightarrow f \in o(h)$$

□

Example. For any two functions f, g, either $f \in O(g)$ or $g \in O(f)$

Not true !

Counter example:

$$f(n) = n \qquad g(n) = \begin{cases} n^2 & , n \quad \text{odd} \\ 1 & , n \quad \text{even} \end{cases}$$

Abuse of notation:

(1) $f(n) = O(g(n))$ means $f(n) \in O(g(n))$ LHS $\subseteq$ RHS

(2) $n^2 + \Theta(n) = O(n^2)$ means $\forall f(n) \in \Theta(n), \exists\, g(n) \in O(n^2)$ s.t. $n^2 + f(n) = g(n)$

(3) $f(n) = O(n^{2+o(1)})$ means $\exists\, g(n) \in o(1)$ s.t. $f(n) = O(n^{2+g(n)})$

2.2 Recurrence Ralations

Example. Integer Multiplication

$$\begin{aligned} x &= a \cdot 10^{\frac{n}{2}} + b \\ y &= c \cdot 10^{\frac{n}{2}} + d \\ xy &= ac \cdot 10^n + (ad + bc)10^{\frac{n}{2}} + bd \end{aligned}$$

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

$$xy = ac \cdot 10^n + [(a+b)(c+d) - ac - bd]10^{\frac{n}{2}} + bd$$

$$T(n) = 3T(\frac{n}{2}) + 4n,\ T(1) = 1$$

Example. Solve $T(n) = 4T(\frac{n}{2}) + n,\ T(1) = 1$

Method 1: Substitution Method

(1) Guess the answer.

(2) Verify by induction.

(3) Solve for constants (in your guess).

Example. Guess $T(n) = 2n^2 - n$ and prove by induction.

$$\text{Guess 1:} \quad T(k) \le c \cdot k^4,\ \forall k. \ \text{ for some } \ c > 0$$

Induction: base case $k = 1$:

$$1 = T(1) \le c \cdot 1^4, \quad \text{true if} \quad c \ge 1$$

Assume $T(k) \leq c \cdot k^4,\ \forall k < n$:

$$\begin{aligned}
T(n) &= 4T(\frac{n}{2}) + n \\
&\leq 4\left[c \cdot (\frac{n}{2})^4\right] + n \\
&= \frac{1}{4}cn^4 + n \\
&= c \cdot n^4 + \left[-\frac{3}{4}cn^4 + n\right] \\
&\leq c \cdot n^4 \qquad\qquad \text{is true for } c \geq \frac{4}{3} \\
T(n) &\in O(n^4)
\end{aligned}$$

$$\text{Guess 2:} \quad T(k) \leq c \cdot k^2,\ \forall k.\ \text{ for some } \ c > 0$$

Induction: base case $\ k = 1$:

$$1 = T(1) \leq c \cdot 1^2, \quad \text{true if} \quad c \geq 1$$

Assume $T(k) \leq c \cdot k^2,\ \forall k < n$:

$$\begin{aligned}
T(n) &= 4T(\frac{n}{2}) + n \\
&\leq 4\left[c \cdot (\frac{n}{2})^2\right] + n \\
&= cn^2 + n \\
&\leq cn^2
\end{aligned}$$

$$\text{Guess 3:} \quad T(k) \leq c_1 \cdot k^2 + c_2 \cdot k,\ \forall k.\ \text{ for some } \ c > 0$$

Induction: base case $\ k = 1$:

$$1 = T(1) \leq c_1 \cdot 1^2 + c_2, \quad \text{true if} \quad c_1 + c_2 \geq 1$$

Assume $T(k) \leq c_1 \cdot k^2 + c_2 \cdot k,\ \forall k < n$:

$$\begin{aligned}
T(n) &= 4T(\frac{n}{2}) + n \\
&\leq 4\left[c_1 \cdot (\frac{n}{2})^2 + c_2(\frac{n}{2})\right] + n \\
&= c_1n^2 + 2c_2n + n \\
&= c_1n^2 + c_2n + [c_2n + n] \\
&\leq c_1n^2 + c_2n \qquad\qquad \text{need} \ \ c_2 \leq -1
\end{aligned}$$

Any c_1, c_2 satisfy the condition : $\begin{cases} c_1 + c_2 \geq 1 \\ c_2 \leq -1 \end{cases}$ will work!

Method 2: Repeated Substitution Method

Example. Solve $T(n) = 4T(\frac{n}{2}) + n,\ T(1) = 1$

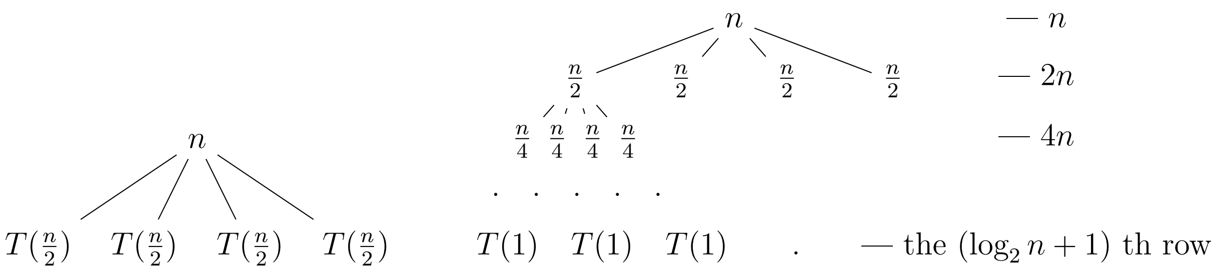
$$\begin{aligned} T(n) &= 4T(\frac{n}{2}) + n \\ &= 4\left[4T(\frac{n}{4}) + \frac{n}{2}\right] + n \\ &= 4^2 \cdot T(\frac{n}{4}) + 2n + n \\ &= 4^2\left[4T(\frac{n}{8}) + \frac{n}{4}\right] + 2n + n \\ &= 4^i T(\frac{n}{2^i}) + (2^i - 1)n \end{aligned}$$

for $n = 2^k$, we can pick $i = k$

$$T(n) = 4^k T(\frac{2^k}{2^k}) + (2^k - 1)\cdot 2^k = 4^k + 4^k - 2^k = 2n^2 - n$$

Method 3: Recursion Tree Method

$T(n) =$



$$\begin{aligned} T(n) &= \left[n + 2n + \ldots + 2^{(\log_2 n) - 1}\right] + 4^{\log_2 n} \cdot T(1) \\ &= (n^2 - n) + n^2 \cdot 1 \\ &= 2n^2 - n \end{aligned}$$

2.3 Master Theorem

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

$$a \geq 1,\ b > 1,\ f(n):\ \exists\ n_0 \quad \text{s.t.} \quad f(n) > 0,\ n \geq n_0$$

Compare $f(n),\ n^{\log_b a}$:

case 1: $f(n) = O(n^{\log_b a} \cdot n^{-\epsilon})$ for some $\epsilon > 0$

$\Rightarrow T(n) = \Theta(n^{\log_b a})$

case 2: $f(n) = \Theta(n^{\log_b a} \cdot (\log n)^k)$ for some $k \geq 0$

$\Rightarrow T(n) = \Theta(n^{\log_b a} \cdot (\log n)^{k+1}) = \Theta(f(n) \log n)$

case 3: $f(n) = \Omega(n^{\log_b a} \cdot n^{\epsilon})$ for some $\epsilon > 0$

$a \cdot f(\frac{n}{b}) \leq c \cdot f(n)$ for some $c < 1$, sufficiently large n

$\Rightarrow T(n) = \Theta(f(n))$

3 Divide and Conquer, Sorting

October 4, 2016

3.1 Some examples

Example. $T(n) = T(\frac{2}{3}n) + T(\frac{1}{3}n) + n$

By recursion tree method, we know max # of levels is $\log_{\frac{3}{2}} n$ and min # of levels is $\log_3 n$

$$\Rightarrow T(n) \in O(n \log n)$$

Example. $T(n) = 2T(\sqrt{n}) + \log_2 n$

Let $m = \log_2 n \ \Rightarrow \ T(2^m) = 2T(2^{\frac{m}{2}}) + m$

Let $S(m) = T(2^m) \ \Rightarrow \ S(m) = 2S(\frac{m}{2}) + m$

$$S(m) \in \Theta(m \log m) \ \Rightarrow \ T(n) \in \Theta(\log n \log \log n)$$

3.2 Divide and Conquer

1. Split into subproblems
2. Solve subproblems recursively
3. Merge solutions

Example. Integer multiplication

Example. Given $a_1, a_2, ..., a_n$, want to find MAX & MIN

1. Find `MAX` — $n-1$ comparisons

   Find `MIN` — $n-1$ comparisons

   Toal — $2n-2$

2. Divide and Conquer

```
MaxMin(a_1, a_2, ..., a_n)
  if n = 1  trivial
  if n = 2  directly compare
  else (M_1, n_1) = MaxMin(a_1, ..., a_⌊n/2⌋)
       (M_1, n_1) = MaxMin(a_⌊n/2⌋+1, ..., a_n)
    return (max(M_1, M_2), min(n_1, n_2))
```

Example. Max Subsequence Sum

Problem: Given $a_1, a_2, ..., a_n$, want to find Max$(a_i + a_{i+1} + ... + a_j)$

$$-5, 12, -3, -4, 14, -2$$

——19——

$$-5, 12, -10, -11, 14, -2$$

14

1. Brute force:

   C_2^n choices for i, j, compute all sums. $\Rightarrow$ $\Theta(n^3)$ time.

2. Divide and Conquer

```
MaxSubseqSum(a_1, a_2, ..., a_n)
  if n = 1  trivial
  else m_1 = MaxSubseqSum(a_1, ..., a_⌊n/2⌋)
       m_2 = MaxSubseqSum(a_⌊n/2⌋+1, ..., a_n)
       m_3 = max(a_i + a_i+1 + ... + a_⌊n/2⌋)
       m_4 = max(a_⌊n/2⌋+1 + ... + a_j)
    return (max(m_1, m_2, m_3 + m_4))
```

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

Let $b_i = a_i + a_{i+1} + ... + a_{\lfloor \frac{n}{2} \rfloor}$

Want to compute `max` b_i $\Rightarrow$ It's in $\Theta(n)$ time !

3.3 Sorting

Problem: Given n numbers $a_1, a_2, ..., a_n$ want rearrange into $b_1 \leq b_2 \leq ... \leq b_n$

Comparison sorting : Only allowed to compare 2 numbers $a_i, \ a_j$

1. **Insertion Sort**

```
for i = 2 to n
  swap a_i forward until
  finding a smaller number
```

Worst case: $n, \ n-1, \ n-2, \ ... \ , \ 2, \ 1$

$$\Rightarrow \Theta(n^2) \text{ running time}$$

Simple, quick on almost sorted arrays !

2. **Bubble Sort**

```
for i = n down to 1
  for j = 2 to i
    if a_{j-1} > a_j
      swap
```

$$\Theta(n^2) \text{ comparisons for all sequences}$$

3. **Stupid Sort**

   (1) Compare $(a_1, a_2),\ (a_2, a_3), \ldots$
   until $a_i > a_{i+1}$

   (2) Swap $a_i,\ a_{i+1}$

   (3) Repeat (1), (2) until done.

   Worst case: $n,\ n-1,\ n-2,\ \ldots,\ 2,\ 1$

$$\Theta(n^3) \text{ comparisons}$$

   No needs to remember indexes ! When data changes constantly, it still works.

4. **Stooge Sort**

```
StoogeSort(i, j)
  StoogeSort(i, i + ⌈2(j-i)/3⌉)
  StoogeSort(i + ⌊(j-i)/3⌋, j)
  StoogeSort(i, i + ⌈2(j-i)/3⌉)
```

$$T(n) = 3T(\frac{2}{3}n),\ T(2) = 1$$
$$T(n) = \Theta(n^{\log_{\frac{3}{2}} 3})$$

   It's not really good as seen.

4 Decision Tree & Order Statistics

October 11, 2016

4.1 Recap

- Recurrence relation
  — Master Theorem
  — Change of variables
- Divide and Conquer
- Comparison Sorting

4.2 Quicksort — NOT stable

Definition: stable sort, equal elements keep their ordering in the input during sorting.

1. Divide:

$$\boxed{\le x} \qquad x \qquad \boxed{\ge x}$$

   (1) Pick $a[i]$ as the pivot.

   (2) Pointers $\quad i = 2, \quad j = n$

   (3) Move i backwards until $a[i] > a[1]$
   Move j forward until $a[j] < a[1]$

   (4) Swap $a[i], \ a[j]$

   (5) Repeat (3), (4) until $i = j$

   (6) Swap $a[1]$ with $a[i]$ or $a[i-1]$

2. Quicksort(”$\le x$”)
   Quicksort(”$\ge x$”)

3. Merge: trivial

Running time:

$$T(n) = T(" \le x") + T(" \ge x") + O(n)$$

Best case: pivot always divide the sequence into equal parts.

$$T(n) = 2T(\frac{n}{2}) + n \Rightarrow T(n) = \Theta(n \log n)$$

Almost Best case: divide into $\frac{99}{100}n$, $\frac{1}{100}n$ each time

$$T(n) = T(\frac{99}{100}n) + T(\frac{1}{100}n) + n \Rightarrow T(n) = \Theta(n \log n)$$

Worst case: $1, 2, 3, ..., n$

$$T(n) = T(n-1) + n \Rightarrow T(n) = \Theta(n^2)$$

Randomized version: pick an element uniformly at random as the pivot.

$\Rightarrow$ **Average running time** $T(n)$**:**

$T(n) = \sum_{i=1}^{n} \frac{1}{n}$ (expected time for picking the i-th smallest a_i as the pivot in step 1.)

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

$$T(n) = \frac{2}{n}\left[\sum_{i=0}^{n-1} T(i)\right] + n$$

$$T(n-1) = \frac{2}{n-1}\left[\sum_{i=0}^{n-2} T(i)\right] + n - 1$$

$$\Rightarrow nT(n) = (n+1)T(n-1) + 2n - 1$$

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

$$S(n) = \frac{T(n)}{n+1} = S(1) + 3(\sum_{i=1}^{n-1} \frac{1}{i}) + (-1)(\sum_{i=2}^{n} \frac{1}{i}) = \Theta(\log n)$$

It is an in place sorting method.

4.3 Merge Sort

```
MergeSort(i, j)
  if j - i = 0, 1   directly compare
    return Result
  else
    A = MergeSort(1, ⌊n/2⌋)
    B = MergeSort(⌊n/2⌋ + 1, n)
    return Merge(A, B)
```

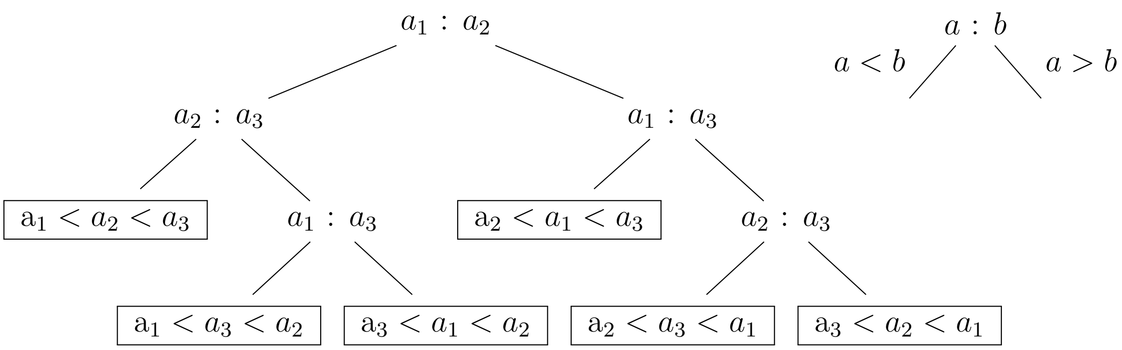
$$T(n) = 2T(\frac{n}{2}) + \Theta(n) \Rightarrow T(n) = \Theta(n \log n)$$

It requires extra space (not in place) !

4.4 Time complexity of comparison sorting

Decision Tree:

Example. Insertion Sort: input a_1, a_2, a_3, we have six consequences of their relation !



Facts:

1. # of leaves $\geq$ # of possible outputs $= n!$ for comparison sorting.
2. running time $=$ (height of tree) -1
3. # of leaves $\leq 2^{\text{height}-1}$

For every decision tree for comparison sorting

$$n! \leq \#\text{ of leaves } \leq 2^{\text{height}-1} = 2^{\text{running time T}}$$

$$\Rightarrow T \geq \log_2(n!) \geq \frac{n}{2} \log \frac{n}{2} = \Omega(n \log n)$$

4.5 Order Statistics

Problem: Select the i-th smallest of $a_1, a_2, ..., a_n$ using pairwise comparison only !

Randomized Divide & Conquer:

Pick a pivot x, it's the k-th smallest number

$$\boxed{\quad \leq x \quad} \qquad x \qquad \boxed{\quad \geq x \quad}$$

(1) $i = k \Rightarrow$ output x

(2) $i < k \Rightarrow$ find the i-th smallest in "$\leq x$"

(3) $i > k \Rightarrow$ find the $i - k$-th smallest in "$\geq x$"

Worst case: always pick MAX/MIN as the pivot $\Rightarrow T(n) = \Theta(n^2)$

Not so bad case: at each round, 1% of the elements removed $\Rightarrow T(n) = \Theta(n)$

Average case:

$$\begin{aligned} T(n) &= \sum_{m=1}^{n} \frac{1}{n} [\ T(\max(m-1, n-m)) + n\ ] \\ &= \frac{2}{n} \sum_{m=\lceil \frac{n}{2} \rceil}^{n} [\ T(m-1) + n\ \ ] \end{aligned}$$

Substitution Method, Guess $T(n) \le c \cdot n$ success, when $c \ge 4$

Worst-case Linear Time Algorithm:

```
SELECT(i, n)
  1. Divide n elements into groups of 5,
     sort each group.
  2. Recursively SELECT the median of ⌊n/5⌋ medians,
     use it as pivot x
  3. Divide & conquer [≤ x] x [≥ x]
```

$$T(n) = T(\frac{n}{5}) + T(\ 3.\ ) + \Theta(n)$$

If $\#(\ \boxed{\le x}\ ) \le \frac{3}{4}n$ and $\#(\ \boxed{\ge x}\ ) \le \frac{3}{4}n$ then

$$T(n) \le T(\frac{n}{5}) + T(\frac{3}{4}n) + \Theta(n)$$

Substitution Method: $T(n) \le c \cdot n$

5 Greedy & Dynamic Programming *October 18, 2016*

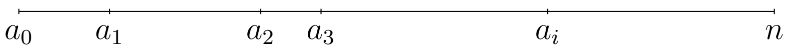
5.1 Recap

1. Quicksort: Worst $\Theta(n^2)$, Average $\Theta(n \log n)$
2. Merge Sort: Worst $\Theta(n \log n)$
3. Comparison sorting: $\Omega(n \log n)$
4. Order Statistics: $\Theta(n)$

5.2 Greedy Algorithms

1. Build up a solution in small steps.
2. At each step, optimize locally.

Example. Trip Planning: Planning a trip with total distance n km.



1. ≤ 20 km each day.
2. can only stop at distance $a_1, a_2, ..., a_k$

Goal: Minimize # of days.
Algorithm: Walk as close to 20 km as possible every day.
Proof. By contradiction:
Suppose Algorithm: stop at $b_1, b_2, ..., b_m = n$
Optimal: stop at $c_1, c_2, ..., c_{m-x} = n$
Find the first j such that $c_j > b_j \Rightarrow c_{j-1} \leq b_{j-1}$

$$c_j - b_{j-1} \leq c_j - c_{j-1} \leq 20\text{km} \Rightarrow \text{alg should select } c_j$$

□

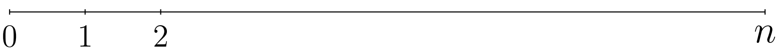
Example. Execute n jobs on a single machine, Job i has execution time t_i
Want: Minimize total waiting time.
If $t_1 = 6, t_2 = 5, t_3 = 3$
ordering: 1 2 3: $6 + (6+5) + (6+5+3) = 31$
3 2 1: $3 + (3+5) + (3+5+6) = 25$
Algorithm: Execute the job with smallest t_i first.
Proof. If there are only two jobs t_1, t_2
t_1 first: $t_1 + (t_1 + t_2)$
t_2 first: $t_2 + (t_2 + t_1)$
putting the smaller one first is better! □

5.3 Dynamic Programming

1. Solve a pre-determined set of subproblems.

2. Memorize solutions to subproblems.

Example.



1. ≤ 20 km per day.

2. Stopping at location i costs c_i.

Want: Minimize total cost.

```
A(i) = the min cost to reach i & stay
B(i) = the best stop right before i

A(i) = c_i          i = 1, 2, ... , 20
for i = 21 to n
  A(i) = min(A(i-1), A(i-2), ... , A(i-20)) + c_i
  B(i) = j        if the above min picks a_j
Output: A(n)
```

Example. Max Subsequence Sum.

Given $a_1, ..., a_n$.

$$-4, 10, -3, -2, 15, -1$$

——20——

Want: $\max_{i,j}(a_i + ... + a_j)$

```
M[i] = MaxSubseqSum(a_1, ... , a_i)
R[i] = Max Subsequence Sum containing a_i

M[1] = R[1] = a_1
for i = 2 to n
  if R[i-1] < 0
    R[i] = a_i
  else
    R[i] = a_i + R[i-1]
Output: M[n]
```

Example. Given two sequences, find the minimum difference.
Mismatch costs c_1, gap costs $c_2 > \frac{1}{2}c_1$
Want: Find the alignment with min total cost

$$X = A\ C\ A\ A\ T$$
$$Y = A\ G\ A\ T\ G$$

1. $\begin{array}{l} X = A\ \boxed{C}\ A\ \boxed{A}\ \boxed{T} \\ Y = A\ \boxed{G}\ A\ \boxed{T}\ \boxed{G} \end{array}$ There are 3 mismatches.

2. $\begin{array}{l} X = A\ \boxed{C}\ A\ \boxed{A}\ T\ \boxed{\_} \\ Y = A\ \boxed{G}\ A\ \boxed{\_}\ T\ \boxed{G} \end{array}$ There are 1 mismatch and 2 gaps.

3. $\begin{array}{l} X = A\ C\ A\ A\ T\ \_\ \_\ \_\ \_ \\ Y = A\ \_\ \_\ \_\ \_\ G\ A\ T\ G \end{array}$ There are 8 gaps

```
C[i,j] = min cost to align
         (first i symbols of X) & (first j symbols of Y)
B[i,j] = which case is X[i] & Y[j] being aligned
d(i,j) = X[i] == Y[j] ? 0 : c_1

Compute C[i,j] :
  case 1: X[i] matched to Y[j]
    C[i,j] = C[i-1,j-1] + d(i,j)
  case 2: either X[i] or Y[j] is aligned to gap
    C[i,j] = min(C[i-1,j],C[i,j-1]) + c_2
  return min of above 2 cases

C[i,0] = ic_2, C[0,j] = jc_2
for i = 1 to m
  for j = 1 to n
    Compute C[i,j]
    Maintain table C[m][n]
Output: C[m,n]
```

	Y		A	A	G
X		0	1	2	3
	0	0	1	2	3
A	1	1	0	1	2
T	2	2	1	1.5	2.5
A	3	3	2	1	2

6 Sorting

October 25, 2016

6.1 Recap

1. Greedy algorithm

2. Dynamic programming

6.2 Sorting in Linear Time

1. **Counting Sort**

   Input: $a_1, a_2, ..., a_n \quad a_i \in \{1, ..., k\}$

   Output: $b_1 \leq b_2 \leq ... \leq b_n$

   Example. $4, 4, 3, 1, 2, 3, 4, 1, 1, 2$

   $$\text{Step 1: count,} \quad \text{Step 2: accumulated sum}$$

   $$\begin{array}{ccc}
   1 \times 3 & — & \leq 1 \times 3 \\
   2 \times 2 & — & \leq 2 \times 5 \\
   3 \times 2 & — & \leq 3 \times 7 \\
   4 \times 3 & — & \leq 4 \times 10 \\
   O(n) \text{ time} & & O(k) \text{ time}
   \end{array}$$

   Finally put them into a new array with right indexes, total time $O(n+k)$

2. **Radix Sort**

   Apply counting sort digit by digit, least-significant digit first.

   Example.

   $$\begin{array}{ccc}
   3 & 2 & 9 \\
   4 & 5 & 7 \\
   6 & 5 & 7 \\
   8 & 3 & 9 \\
   4 & 3 & 6 \\
   7 & 2 & 0
   \end{array}
   \qquad
   \begin{array}{ccc}
   7 & 2 & 0 \\
   4 & 3 & 6 \\
   4 & 5 & 7 \\
   6 & 5 & 7 \\
   3 & 2 & 9 \\
   8 & 3 & 9
   \end{array}
   \qquad
   \begin{array}{ccc}
   7 & 2 & 0 \\
   3 & 2 & 9 \\
   4 & 3 & 6 \\
   8 & 3 & 9 \\
   4 & 5 & 7 \\
   6 & 5 & 7
   \end{array}
   \qquad
   \begin{array}{ccc}
   3 & 2 & 9 \\
   4 & 3 & 6 \\
   4 & 5 & 7 \\
   6 & 5 & 7 \\
   7 & 2 & 0 \\
   8 & 3 & 9
   \end{array}$$

   Running time $= O(d(n+k))$

   Given n numbers in $\{0, 1, 2, ..., n^d - 1\}$

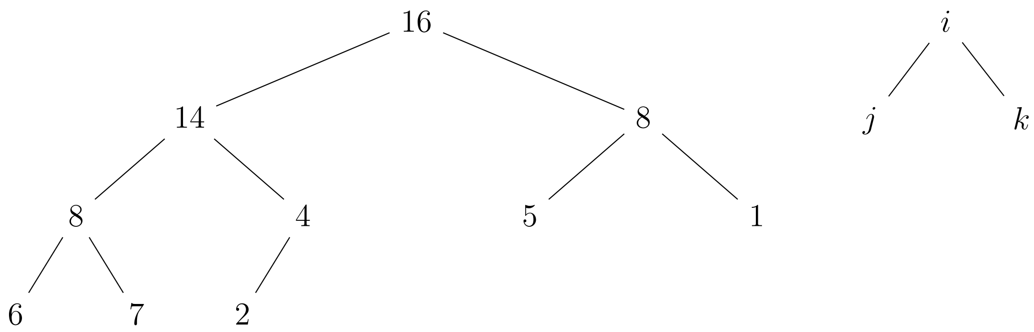
   1. Rewrite them into base-n numbers ($\Rightarrow d$ digits)

   2. Apply radix sort

      Running time $= O(d(n+n)) = O(dn)$

3. Heapsort

   Heap (binary max heap) :



node i is node j & k 's parent

Max Heap $\Leftrightarrow$ value(parent(i)) $\geq$ value(i)

Array Representation:

parent(i) $= \lfloor \frac{i}{2} \rfloor$ left-child(i) $= 2i$ right-child(i) $= 2i + 1$

Given n numbers:

1. arrange into a binary max heap.
2. use heap-operations to sort.

```
MaxHeapify(A, i)
  compare A[i], A[2i], A[2i+1]
  if A[i] is largest
    return
  if A[2i] or A[2i+1] is largest
    swap A[i], A[x]
    MaxHeapify(A, x)
    return
```

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

```
BuildMaxHeap(A)
  for i = n down to 1
    MaxHeapify(A, i)
  return
```

$$\Theta(n)$$

```
Heapsort(A)
  BuildMaxHeap(A)
  for i = n down to 2
    swap A[1], A[i]
    heapsize--
    MaxHeapify(A,1)
  return
```

$$O(n \log n)$$

6.3 Min-Heap

Wants to support:

1. Insert(k) : Insert a data with key k.
2. MIN : Find the data with MIN key.
3. Extract-MIN : Find MIN & delete MIN.
4. Union : Combine two heaps into one.
5. Decrease(x, k) : Pointer x points to data$D$ decrease key(D) to k.
6. Delelte(x) : Delete the data that x points to.

Amortized cost	Insert	MIN	Extract	Union	Decrease	Delete
binary MIN heap	O($\log n$)	O(1)	O($\log n$)	O(n)	O($\log n$)	O($\log n$)
Fibonacci Heap	O(1)	O(1)	O($\log n$)	O(1)	O(1)	O($\log n$)

6.4 Amortized Analysis

Gives an upper bound on the average cost of the operations.

Example. A Stack S. two operations.

1. push(d): push d onto the stack.
2. multi-pop(k): pop min(k,size(s)) elements from the stack.

Will prove $\begin{cases} \text{push: } 2 \\ \text{multi-pop: } 0 \end{cases}$ amortized cost.

Meaning: starting from empty, perform n_1 push operations, n_2 multi-pop operations

Then total cost $\leq 2n_1 + 0n_2$

1. Aggregate Method

   Directly find an upper bound of the total cost.

   With n_1 push, only n_1 elements are added to, can only pop $\leq n_1$ elements.

   $\Rightarrow$ total cost(multi-pop) $\leq n_1$

2. Accounting Method

   Overcharge some operations to pay for later operations

   Example. push(d): cost \$ 2

   \$ 1 actual cost, \$ 1 saved for popping d in the future

   $\Rightarrow$ cost \$ 0 for multi-pop.

3. Potential Function

   Φ: A potential function that maps configuration D_i to a real number $\Phi(D_i)$

   Suppose operation $O_i : D_{i-1} \to D_i$

   Amortizes cost for O_i : $\hat{c_i} = c_i + \Phi(D_i) - \Phi(D_{i-1})$

   Example. $\Phi(S) = \text{ size } (S)$

   push: $\hat{c_i} = c_i + \Delta\Phi = 1 + 1 = 2$

   multi-pop: $\hat{c_i} = c_i + \Delta\Phi = \min(k, \text{ size } (S)) + (-\min(k, \text{ size } (S))) = 0$

   ※ $\Phi(D_i) \geq \Phi(D_0) = 0$

7 Midterm Note & Fibonacci Heap

November 1, 2016

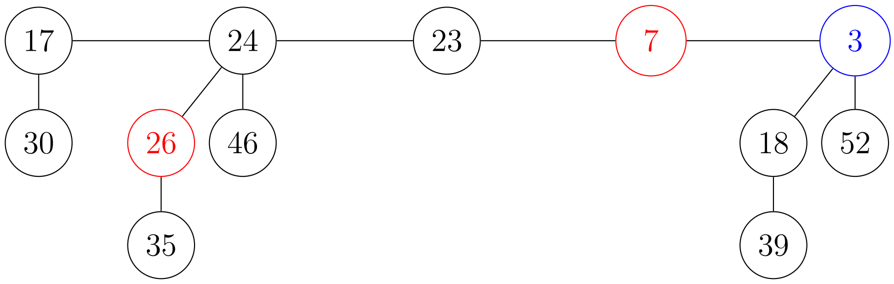
7.1 Recap

1. Counting Sort $O(n+k)$
2. Radix Sort $O(d(n+k))$
3. Heap Sort $O(n \log n)$
4. Binary Heap
5. Amortized Analysis

7.2 Midterm

1. You may bring one A4-size hand-written note
2. You may redo midterm as a homework and submit online
3. 1. Sym 2. Recursion 3. $\sim$ 6. DP + Greedy + Divide & Conquer

7.3 Fibonacci Heap



1. Many trees linked together on the roots.
2. Every tree is a MIN-heap.
3. Some nodes are marked. (RED nodes)
4. Pointer to MIN. (BLUE one)

$$\Phi = c_1 \cdot t(H) + c_2 \cdot m(H) \qquad D : \text{max-degree}$$

7.3.1 Operations on Fibonacci Heap

1. MIN: Trivial! $O(1)$

2. Union: Link roots together. Find min. $O(1)$

3. Insertion(k): Create a new tree to the left of MIN. $O(1) + c_1 = O(1)$

4. Extract-MIN:

   (1) Remove MIN. $O(1)$

   (2) Add all children to the root list. $O(D)$

   (3) **Consolidate**. $O(D) + c_3(k = \#$ of trees combined)

   $$\delta\Phi = -c_1(k),\ \text{set } c_1 = c_3 \to O(D)$$

   (4) Find MIN. $O(D)$

5. Decrease(x, k): 3 cases

   (1) After decreasing. key(k) $\geq$ key($parent(x)$) $\Rightarrow$ direct decreasing it.

   (2) $k <$ key($parent(x)$), parent unmarked
   $\Rightarrow$ Mark parent. $O(1) + c_1$
   $\Rightarrow$ Move the subtree rooted at x to the root list. $O(1) + c_2$

   (3) $k <$ key($parent(x)$), parent marked.
   $\Rightarrow$ Cut x, move to root list. $O(1)$ $O(1) + c_1$
   $\Rightarrow$ Recursively cut the parent, move to root unmarked until finding an unmarked node **m**. $c_4 + c_1 - c_2$ Set $c_2 = c_1 + c_4$
   $\Rightarrow$ Mark **m**.

6. Delete(x): Decrease(x, $-\infty$) and Extract-MIN $O(1) + O(D)$

$$\text{Theorem: } D \leq \log_\phi N \qquad \phi = \frac{1+\sqrt{5}}{2}$$

8 Midterm Riview & Data Structure

November 22, 2016

8.1 Midterm

Problem 2

2. Be careful $T(2) = T(1) = 5$!

$$T(n) = (\log_2 n + 1)! \cdot \frac{T(1)}{2}$$

Problem 6

1. Bread $b_i \Rightarrow$ time t_i, profit p_i

```
A[t] = max profit with t units of time
A[0] = 0, A[-1] = A[-2] = ... = -inf

for t = 1 to T
  A[t] = max{A[t - t_i] + p_i, 0}   ∀i ∈ {1, 2, ..., m}

output: A[T]
```

2. Maintain a cube.

```
A[i, j, k] = min time to make profit k,
             use j types of bread {b_1, b_2, ..., b_i}

A[i, j, k] = min{A[i - 1, j, k],
                 A[i - 1, j - 1, k - xp_i] + xt_i, ∀x}

output: A[n, k, p]
```

8.2 Disjoint Set

Maintain a collection $S_1, S_2, ..., S_i$ of disjoint sets.
Each S_i has a leader(element).

(1) Make-set(x): Makes a new set $\{x\}$.

(2) Find(x): Find the leader of x's set.

(3) Union(x, y): Union x's and y's sets.

1. Linked List: every element have one pointer to next and top pointer to leader.
   (1) — $\mathcal{O}(1)$, (2) — $\mathcal{O}(1)$, (3) — $\mathcal{O}(|S_2|)$
   Idea: Change the top pointer for the smaller set when union.
   Lemma: If an element switched top pointer k times, it belongs to a set of size $\geq 2^k$

   Proof. By induction, $k = 0$ trivial.
   Assume it's true for $k = k_0 - 1$
   The k_0-th change happens in the union operation from $k_0 - 1$-th
   the size of set $\geq 2 \cdot (2^{k_0-1}) \geq 2^{k_0}$ □

   Theorem. Starting from 0 elements.
   $$\text{Perform} \begin{cases} n & \text{,make-set} \\ \leq n-1 & \text{,union} \\ m & \text{,find} \end{cases} \Rightarrow \text{total cost} = \mathcal{O}(m + n \log n)$$

   Proof. For set size $\leq n$,
   all elements switched top pointer $\leq \log_2 n$ times.
   $\Rightarrow$ total $\leq n \log n$ top pointer swithings. □

2. Disjoint Set Forests
   (1) — $\mathcal{O}(1)$, (2) — 2 Find $+ \mathcal{O}(1)$, (1) — $\mathcal{O}$(height of the tree)

   Idea:
   - When union, link the shorter tree to the taller tree.
   - path compression.

   total time $= \mathcal{O}((m+n)\log n) = \mathcal{O}((m+n) \cdot \alpha(n))$ s.t. $\alpha(10^{80}) = 4$

8.3 Graph

$G = (V, E)$

$V = \{v_1, v_2, ..., v_n\}$ the set of vertices(nodes).

$E = \{e_1, e_2, ..., e_n\}, e_i = (v_j, v_k)$ the set of edges.

undirected graph: v_j, v_k interchangeable.

directed graph: v_j, v_k not interchangeable.

In the class, focus on *simple graphs*

Definition:

simple graph: graphs with no loops/double edges.

path from u to v : $(v_0, v_1, ..., v_k)$ where $v_0 = u$, $v_k = v$, $(v_i, v_{i+1}) \in E$

simple path: all nodes are distinct.

cycle: path from v to v

1. Adjacency Matrix $(n \times n)$

   $$A = \begin{bmatrix} .... \\ ..A_{ij}.. \\ .... \end{bmatrix}, \quad A_{ij} = \begin{cases} 0 & ,\text{if}(i,j) \notin E \\ 1 & ,\text{if}(i,j) \in E \end{cases}$$

2. Adjacency list

   $Adj[v] = \{$ vertices adjacent to $v\} = \{u | (v, u) \in E\}$

   $|V|$ sets, $2|E|$ elements.