CSC236 Cheat Sheet

Lecture 1 & lecture 2 Inductions*Make sure the "chain of implications" is connected **everywhere. Note: 会分 case 讨论，比如分奇偶**

Simple induction

1. Define the predicate P(n)
2. Base Case: show that P(1) is True
3. Induction Step: Assume P(k) is true (I.H.), show that P(k+1) is true.

EX. Using 6 cents and 11 cents only, you can make any amount greater than 60 cents.
要点:至少一个 11 cents 和没有 11 cents 这两个情况你都能凑出一个 1
Base case: 60 cents 可以用 6 cents 和 11 cents 凑出
Induction step: assume p(k)is true, show p(k+1)
Case1:至少一个 11 cents, 可以通过两个 6 cents 来制造出 12 cents(比原来的多了个 1)
Case2：no 11 cents，那至少就有 9 个 6, 6*9=54，通过 5 个 11 可以凑出 55（比原来的多了个 1）

Complete Induction

1. Define the predicate P(n)
2. Base Case: show that P (base cases) are True
3. Induction Step: Assume P(1), p(2)….p(k) are true (I.H.), show that P(k+1) is true.(the only difference with simple induction)

EX. Prove "Prime or Product of Primes"
Base case: n=2, 2 is already a prime.
Induction Step: Assume $P(2) \wedge P(3) \wedge \ldots \wedge P(n)$, all numbers from 2 to n can be written as a product of primes. (I.H.). Show P(n+1), can be written as a product of primes

- Case 1: n+1 is prime, then n+1 is already a product of primes, done
- Case 2: n+1 is composite (not prime), then n+1 can be written as a*b, where 2 <= a, b <= n, According to I.H., each of a and b can be written as a product of primes. So n = a x b can be written as a product of primes.

Structural Induction (recursively defined set)

1.Base element
2.recursive rules that generates new elements of the set from the existing element of the set (the smallest set which contains nothing else)

Suppose that S is a recursively-defined set and P is some predicate

- Base case: If P is true for each base element of S
- Induction Step: under the assumption that P(e) is true for element e of S, we find that each recursive rule generates an element that satisfies P. Then P is true for all elements of S.

We must do the induction step for every recursive rule!

EX. Empty string and 1 are in S, if w is a string in S, then so are w00 and w01. Prove that S does not contain two consecutive 1s.
Base Case: empty string: no consecutive 1's, "1": no consecutive 1's, check Induction Step: check all the recursive rules:

- Rule 1: w → w00: show P(w) ⇒ P(w00)
- Rule 2: w → w01: show P(w) ⇒ P(w01)

(这里的 w 就是 empty string 或 1，100 和 101 都没有 consecutive 1s)

Lecture 3 big oh, big theta, big omega and Iterative runtime (重点就是找 c 和 n0, n0 the breakpoint 之后的 runtime，n0 一般选 1 都 work)

***(slowest)** $1 < \log n < \sqrt{n} < n < n\log n < n^2 < n^3 < 2^n < n^n$ **(fastest)**

Big oh (upper bound-no faster than) Over-Estimate

f(n) is Big-O of g(n), $f(n) \in O(g(n))$, iff $\exists c \in \mathbb{R}^+, \exists n_0 \in \mathbb{N}$, such that $\forall n \geq n_0$, $f(n) \leq c\, g(n)$

Tricks：remove the negative term or multiply a positive term(一定要确保他是的，如果有可能是正的就不能 remove，一定要看好 n0 的值)

EX. Proof that $100n + 10000$ is in $O(n^2)$
Choose $n_0=1$, c = 10100, then for all $n \geq n_0$,
$100n + 10000 \leq 100n^2 + 10000n^2$ (**because** $n \geq 1$) $= 10100n^2 = cn^2$
Therefore, by definition of big-Oh, $100n + 10000$ is in $O(n^2)$

Big omega (lower bound-no slower than) Under-Estimate

Function $f(n)= \Omega(g(n))$ iff $\exists c \in \mathbb{R}^+, \exists n_0 \in \mathbb{N}$, such that $\forall n \geq n_0$, $cg(n) \leq f(n)$

tricks: 1.remove a positive term or multiply a negative term(一定要确保他的正负性，需要 n0 多大就选多大)

2. 把 higher order term 拆开去减那个负 term 后一起删掉

EX. Prove that $2n^3 - 7n + 1 = \Omega(n^3)$
$2n^3 - 7n + 1 \rightarrow n^3 + (n^3 - 7n) + 1$ (Pick $n_0 = 3$ we want $n^3 - 7n > 0$) $\rightarrow n^3 + 1$ (remove a positive term) $\rightarrow n^3 \rightarrow$ pick c = 1

Big theta (tight bound-no faster and slower)

Function $f(n) = \Theta(g(n))$ iff $f(n) \in O(g(n))$ and $f(n)= \Omega(g(n))$
Proof: big oh 和 big omega 都 prove 一遍

Worst case analysis of iterative runtime:把每层 loop 运行的次数用 summation∑表示出来，一般一个 loop 就是 0 到 n-1

Lecture 4 recursion (find the closed form thru repeated substitution, develop a recurrence)

我们是通过带有 F(n)的 recursive function，用 repeated substitution 得到一个 closed form，也就是通式，知道了 n 就直接知道了 runtime，通过 closed form 去找这个 recursive function 的 runtime，而带有 T(n)的式子就是 recurrence 了，是以 recursive 的形式来表示的 runtime

repeated substitution: substitute 几次，substitution 的次数就是 k，substitute 的时候<u>**常数项尽量不要拆开方便找规律**</u>，找到一个 pattern 以后以 k 和 n 的形式总结出来 guessed closed form**, <u>for each base case,</u>** 把 function 用 base case 代进去替换掉，得出 k，plug k back into the formula, prove the closed form is equivalent to the recurrence by induction

Prove the closed form using induction

EX. prove T1(n) = 2T(n-1)+1 (the recurrence) is equivalent to $T0(n) = 2^n-1$(closed form)
Predicate P(n): T1(n) = T0(n)
Base case: $T1(1) = 1 = 2^1-1=1$(把 base case 分别带入 recurrence 和 closed form 中)
Induction step：suppose n＞1 and that T1(k)=T0(k), prove T1(k+1) = T0(k+1)

Develop a recurrence:1. Find the base case(s) that can be evaluated directly 2. break the large problem into small problems, so that you can define f(n) in terms of f(m) for some m < n.(e.g. f(n): The number of 2-element subsets of n elements. 分成 the subsets that contain e_n and the subsets that do NOT contain e_n, 有 e_n 的那组，他和除了他自己的都可以组，所以有 n-1 组，那没有 e_n 的那组，就少了一个 element，就是有 f(n-1)组啦！所以最后 f(n)=n-1+f(n-1))

Lecture 5 Recursion and Master theorem (Prove the runtime of recursive functions)

如果直接给你一个 **recursive function** 怎么看他的 runtime？constant 就写 constant，F(n)就写 T(n)，F(n-1)就写 T(n-1)，和 base case 结合起来，<u>写成花括号的形式</u>，，然后再继续用 repeated substitution 等接下去一系列的操作求出这个 recursive function 的 runtime.

求 recurrence 的 runtime 的三种方法：

1. find the recurrence and use repeated substitution to find the closed form, prove the theta bound.
2. 直接猜一个 runtime，用 induction 证（例题在下面）
3. Master theorem

<u>**Master Theorem:**</u> （找的是 asymptotic bound）

Let T(n) be defined by the reccurence $T(n) = aT(n/b) + f(n)$, for some constants $a \geq 1$，$b > 1$ and $k \geq 1$， then we can conclude the following about the asymptotic complexity of T(n): 注意一定要满足形式

(1) If $k = \log_b a$, then $T(n) = O(n^k \log n)$.
(2) If $k < \log_b a$, then $T(n) = O(n^{\log_b a})$.
(3) If $k > \log_b a$, then $T(n) = O(n^k)$.

When master theorem does not directly apply,把小的合到大的那个里在用

Lecture 6 Divide and Conquer (写满足这种形式 $T(n) = aT(n/b) + f(n)$**的 function)**

一定要注意的是最终的结果是不是可以直接来源于左右 sub-problems，还是要 cross the middle point!一般最后的 n 就是用来 check middle point 的

Lecture 7 & Lecture 8 Recursive program correctness and Iterative program correctness

Recursive Program Correctness (for each program path)

○ if there is no recursive call, analyze the code directly (like base cases)
○ if there are recursive calls

1. show that the precondition holds when making each recursive call
2. Show that the recursive call occurs on "smaller" values than the original call. (So it will terminate eventually)
   Note: sometimes measure of the size 是两个 parameter 的 sum！
3. You can then assume that the recursive call satisfies the post-condition (by Induction Hypothesis)
4. Show that the post-condition of the function is satisfied based on the assumption

Iterative Program Correctness (partial correctness and termination**)**

<u>**Partial Correctness**</u>**: (induction step 时可能需要分奇偶讨论)**

loop invariant: relationship of parameters that's not going to change in every iteration，和 loop guard 两部分 (track the code for several iteration to find the invariant)

1. Base case: Argue that the loop invariant is true when the loop is reached（还没进到 loop 之前）
2. Induction Step: assume that <u>the invariant</u> and <u>guard</u> are true at the end of an arbitrary iteration **i0** (by I.H.), show that the invariant remains true after one iteration **i1**(invariant 和 loop guard 两个部分分别证，<u>表示出 **variable** 在两个 **iteration** 之间的变化</u>，比如 `i1=i0+1`，`sum1=sum0+A[i0]`)
3. Check post-condition: Argue that the <u>invariant</u> and the <u>negation of the loop guard</u> together let us conclude the program's post-condition.

<u>**Termination:**</u>**(loop variant)**

Show that the loop variant is a natural number that's decreasing on every iteration

If the loop variant decreases on each iteration yet cannot drop below 0, then we can conclude that at some point the loop must terminate. (sometimes it's 一个 variable 加/减另一个 variable)

Lecture 9 Regular Languages and Regular Expressions (基本只是 terminologies， description to regex, regex to description, simplify regex)

1. Alphabet: a finite set of symbols
2. A string w over alphabet ∑ is a finite sequence of symbols from ∑, 由 alphabet 里的 elm 组成的一个 string
3. Empty string “” which we denote with ε， it’s a string over any alphabet.
4. Length of string: number of characters in w， |ε| = 0， $\sum^n$ is set of strings over ∑ of length n， $\sum^*$ is set of all strings over ∑
5. A language L is only a subset of $\sum^*$， L ⊆ ∑*
6. Operations on Languages: Given two languages L, M ⊆ ∑*, three operations can be used to generate new languages.
- Union, L ∪ M
- Concatenation, LM: L 里的每个 elm 都和 M 里的每个 elm 相连
- L*: all strings that can be formed by concatenating 0 or more strings from L.

7. A regular expression (regex) is a string representation of a regular language. A regex “matches” a set of strings (the represented regular language).

Definition of regular language
1. ∅, the empty set, is a regular language
2. {ε}, the language consisting of only the empty string, is a regular language
3. For any symbol a ∈ ∑, {a} is a regular language.
4. If L, M are regular languages, then so are L ∪ M, LM, and L*(recursive rule)

EX. Prove that language L = {a, aa} is a regular language.
Proof: {a} is regular by definition So {aa} = {a}{a} is regular (concatenation rule) So {a, aa} = {a} U {aa} is regular (union rule)

Definition of regular expression (for a regex r , L(r) is the language matched by r)
1. ∅ is a regex, with L(∅)string with the single character ∅ = ∅ empty set. (matches no string)
2. eps is a regex, with L(epsilon) = {eps} (matches only the empty string)
3. For all symbols a ∈ Σ, a is a regex with L(a) = {a} (matches only single string a)
4. Let r1, r2 be regexes. Then r1 + r2, r1r2, and r ∗ 1 are regexes, with L(r1 + r2) = L(r1) ∪ L(r2), L(r1r2) = L(r1)L(r2), and L(r ∗1) = (L(r1))* (matches union, concatenation, and star, respectively)

For a regex to correctly represent a language L, it must match every string in L, and nothing else.

Lecture 10 DFA（the DFA diagram, prove correctness of DFA, prove minimum number of states）

Note：DFA 非常严谨，每个 state 都要有 exactly |Σ| transitions，有 trapping state 等

1.DFA’s Definition: D = (Q, Σ, δ, s, F) where Q: the (finite) set of states in D, Σ: the alphabet of symbols used by D, δ: Q × Σ → Q is **the transition function,** s ∈ Q is the initial state of D, F ⊆ Q is the **set** of accepting (final) states of D

2. number of transition is |Q|*|Σ|; **each state must have exactly |Σ| transitions leading out of it**, each labelled **with a unique symbol** in Σ (only one transition)

3. Proving the correctness of a DFA with state invariant (每个 state 都有自己的 unique state invariant)

Format for proving correctness of state invariant: (structural induction)
1.Base case: Show that ε (the empty string) satisfies the state invariant of the initial state.
2.Induction step: For **each transition** from state q to state r on symbol a,
-assume that the invariant of state q holds for some string w (I.H.)
-show that the invariant of state r holds on string wa.
3.“Postcondition”: Show that the state invariant(s) of the accepting state(s) exactly describe the languages that we want the DFA to accept.

Notes on state invariants predicate p(x): **covers all cases and no overlapping**
1. All strings w reaching q must be satisfy P(w), All strings w satisfying P(w) must reach q.
2. No string should satisfy more than one invariant
3.Every string should satisfy one of the invariants
EX. P(w): w has an even number of 1’s and P(w): w has an odd number of 1’s

Format for proving minimum number of states (contradiction)
1.suppose for contradiction, 先 assume 我们需要比 minimal 还少一个 state
2.根据每个必要的 state invariant 来找找几个 strings， 各 state invariant 找一个, then at least two of those strings must end at the same state (pigeonhole)
3. for **any pair**(**每个 pair 都要证**) of them, find some x (自己找的一个 string), that wix is NOT in L (rejected) and wjx is in L (accepted), then we have a contradiction, and what must assumed (we can find a correct DFA with 3 states) must be FALSE.

EX. {{0,1}| w has at least 3 ones}. w0 = epsilon, w1=1, w2 = 11, w3 = 111,
Pair1: w0 and w1->we choose x to be 11. So w0x = 11, rejected; w1x = 111, accepted. That’s what we want.
Pair2: w2 and w3->No need to choose x, w2 =11 is rejected by the DFA; w3 =111 is accepted, contradiction again.

4. prove a language is not regular is basically to prove the minimum number of states needed by the DFA is **infinite.**

Lecture 11 NFA (draw NFA diagram, NFA to DFA, DFA to Regex)

Note: 只要有一条 path 能通向 accepting state 就 accept, 一个 state 可以有 more than one transitions thru one symbol, 一个 state 可以没有任何 outgoing transition, 可以没有 trapping state, given a state and a symbol, it returns the set of states to which that symbol transitions, 有 ε-transition

NFA to DFA (Subset Construction-DFA 里的每个 state 都代表 a set of states in NFA)
1.先把 NFA 的 initial state 通过 0 or more ε-transitions 可以得到的 set of states 设为 DFA 的 initial state
2.再从已经得到的 DFA 的 initial state 出发，每个 symbol 的 transition 都走一遍，把 set 里面的每一个 state 可以通过 symbol 得到的下一个 state 组成新的 set of state，记住**再 check 有没有通过 ε-transitions 可以得到 free state**！如果是新的 state，继续这个走每个 symbol 的 transition，一直 repeat 到没有 new states 出现
Note：如果一个 state 通过一个 symbol 去不到任何 state，那就是 empty set（也算是一个 state）
3. Accepting states of the DFA is any states that contains an accepting state of the NFA.

DFA to Regex (State Elimination)
1. If the initial state q1 has incoming edges, create a new start state s and add an ε-transition to q1.
2. If there are multiple accepting states or if the final state qn has outgoing edges, create a new accepting state f and add ε-transition(s) to f from all former accepting states. (Former accepting states become non-accepting.)
3.删掉 trapping state
4. Eliminate state by state until only the initial and the accepting state remain.
(q1)—r1—(q2)—r2—(q3)变成(q1)—r1r2—(q3)
(q1)—r1—(q2 自绕 r2)—r3--(q3)变成(q1)—r1r2*r3—(q3) 星号是 kleen star
(q1)—r1r2 两条—(q2)变成(q1)—r1+r2—(q2)
(q1)—r1—(q2)—r2—(q1)变成 q2 自绕 r1r2

*Euclidean algorithm：GCD(a, b) = GCD(b, a % b) where b<a by precondition and a%b < b, because a%b <= b-1 by definition

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

$\sum_{k=1} k^3 = \frac{n^2(n+1)^2}{4}$

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

$\sum^n_{k=1} r^k = \frac{r(1-r^n)}{1-r}$

$\sum^k_{i=0} n^i = \frac{n^{k+1}-1}{n-1}$

$\log_b(mn) = \log_b(m) + \log_b(n)$

$\log_b(^m/_n) = \log_b(m) - \log_b(n)$

$\log_b(m^n) = n \cdot \log_b(m)$

$a^{\log b(c)} = c^{\log b(a)}$ $\log_b(1) = 0$

some regular expressions (我们只涉及得到+，*和括号)
1. 5^{th} has to be 1
⇒(0+1)*1(0+1)(0+1)(0+1)(0+1)
2. L = {w∈{0,1}*| w represents a binary number divisible by 2} ⇒(0+1)*0
3. L = {w∈{0,1}*| w starts and ends with the same symbol}
⇒0(0+1)*0+1(0+1)*1+0+1+ε
4. L = {w∈{0,1}*|w has a substring 010}
⇒(0+1)*010(0+1)*

证 recurrence runtime without using repeated substitution，直接猜一个 runtime，用 induction 证. EX.

$$T(n) = \begin{cases} 2T\left(\frac{n}{2}\right) + kn & if\ n > 1 \\ a & if\ n = 1 \end{cases}$$
Prove by complete induction that T(n) = O(nlogn).

Assume n is a power of 2. **<u>We must find positive constants n0 and c such that T(n)≤ cnlgn for all n ≥n0.</u>** Choose no = 2. (一般就选让 function 满足 base case 的值)
Base case: n0 = 2, by the recursive definition, T(2) = 2a+2k. we must prove that 2a+2k ≤cnlgn = 2c. We can do this by letting c = a + k. (as long as c ≥ k)
Inductive step: let n >2 and suppose that T(n/2) ≤ c(n/2) lg(n/2).
T(n) = 2T(n/2) + kn ≤ 2(c(n/2) lg(n/2)) + kn = cnlg(n/2) + kn
= cnlgn − cnlg2 + kn = cnlgn − cn + kn = cnlgn + (k − c)n ≤ cnlgn
Therefore, we choose c = a + k and no = 2 to complete the proof.

*midterm2 上的 TA 让我们自己查的那题