**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 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 | **Complete Induction**  1. Define the predicate P(n)  2. Base Case: show that P (1) are True  3. Induction Step: Assume P(1), p(2)….p(k) are true (I.H.), show that P(k+1) is true.  EX. Prove “Prime or Product of Primes”  Base case: n=2, 2 is already a prime.  Induction Step: Assume P(2) ∧ P(3) ∧ … ∧ 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 by I.H., 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＜√n＜n＜nlogn＜n2＜n3＜2n＜nn (fastest)**

|  |  |  |
| --- | --- | --- |
| **Big oh (upper bound-no faster than) Over-Estimate**  f(n) is Big-O of g(n), f(n) ∈ O(g(n)), iff ∃c ∈R+, ∃n0∈N, such that ∀n≥n0, f(n) ≤ 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(n²)  Choose n₀=1, c = 10100, then for all n >= n₀,  100n + 10000 <= 100n² + 10000n² (**because n ≥ 1**) = 10100n² = cn²  Therefore, by definition of big-Oh, 100n + 10000 is in O(n²) | **Big omega (lower bound-no slower than) Under-Estimate**  Function f(n)= Ω(g(n)) iff ∃c∈R+, ∃n0∈N, such that ∀n≥n0, cg(n) ≤ 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 2n³ - 7n + 1 = Ω(n³)  2n³ - 7n + 1🡪n³ +(n³ - 7n)+ 1(Pick n₀ = 3 we want n³ - 7n > 0)🡪n³ + 1(remove a positive term)🡪 n³🡪pick c = 1 | **Big theta (tight bound-no faster and slower)**  Function f(n) =Θ(g(n)) ifff(n) ∈ O(g(n)) and f(n)= Ω(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**常数项尽量不要拆开方便找规律**，找到一个pattern以后以k和n的形式总结出来guessed closed form**, for each base case,** 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) = 2n-1(closed form)  Predicate P(n): T1(n) = T0(n)  Base case: T1(1) = 1 = 21-1=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 en and the subsets that do NOT contain en，有en的那组，他和除了他自己的都可以组，所以有n-1组，那没有en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，**写成花括号的形式**，，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 | **Master Theorem:** （找的是asymptotic bound）  Let T(n) be defined by the reccurence T(n) = aT(n/b) + f(n), for some constants a ≥1，b＞1 and k ≥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 = logb a, then T(n) = O(nk log n).  (2) If k < logb a, then T(n) = O(nlogb a ).  (3) If k > logb a, then T(n) = O(nk ).  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  **5.conclusion** | **Iterative Program Correctness (**partial correctness and termination**)**  **Partial Correctness: (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 the invariant and guard 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, 表示出variable在两个iteration之间的变化, 比如i1=i0+1，sum1=sum0+A[i0])**  3. Check post-condition: Argue that the invariant and the negation of the loop guard together let us conclude the program’s post-condition.  **Termination:(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)  4.**conclusion** |

**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，∑n is set of strings over ∑ of length n，∑\* is set of all strings over ∑ | 5. A language L is only a subset of ∑\*,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 (a set of strings)**  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, correctness of DFA, minimum number of states）Note: 每个state都要有exactly |Σ| transitions，有trapping state等**

1.DFA’s Definition: D = (Q, Σ, δ, s, F)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Q: the (finite) set of states in D | Σ: the alphabet of symbols | 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.  Note: state invariant是只有这个state才对的东西, 非常的specific，start with 再前方的  edge, ends with incoming edge，q0通常是empty string  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，只有这个state才对的东西  3.Every string should satisfy one of the invariants，非常的specific  EX. P(w): w has an even/odd number of 1’s or length of string that reaches the state | **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 regex is incorrect simply find a counter-example

**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.) eliminate一个环绕式的state时要确保他的incoming和outgoing edge都被taken care了！  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)—r1(r2\*)r3—(q3) 星号是kleene star零or多次重复  (q1)—r1r2两条—(q2)变成(q1)—r1+r2—(q2)  (q1)—r1—(q2)—r2—(q1)变成q2自绕r1r2（eliminate掉的q1） |

\*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

|  |  |  |
| --- | --- | --- |
| ∑k=1 k2  ∑k=1 k3 =  ∑k=1 k =  ∑nk = 1 rk =  ∑ki = 0 ni =  logb(mn) = logb(m) + logb(n)  logb(m/n) = logb(m) – logb(n)  logb(mn) = n · logb(m)  alogb(c) = clogb(a) logb(1) = 0 | **some regular expressions**  1. last 5th digit 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)\*  **(我们只涉及得到 +，\* ，括号)** | **证recurrence runtime without using repeated substitution，直接猜一个runtime，用induction证. EX.**  T(n) = Prove by complete induction that T(n) = O(nlogn). Assume n is a power of 2. **We must find positive constants n0 and c such that T(n)≤cnlgn for all n ≥n0.** Choose no = 2. (一般就选让function满足base case的值,或者直接选n=1时，哪个好算选哪个)  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, c不一定要完全满足这个等式，inequality满足就行)  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. |

|  |  |  |
| --- | --- | --- |
| **Prove that if L is a regular language, then so is Rev(L).**  Proof:根据regular language的definition来证  **Base cases:**  1. {} is a regular L. There are no strings to reverse, so rev({}) = {}, so rev({}) is a regular language.  2. By the same reason, rev({epsilon}) is a regular language.  3. {a} is a regular language, for any symbol a. But a = aR, so rev({a}) = {a}, and so rev({a}) is regular.  **Inductive case:** suppose that L and M are regular, and that rev(L) and rev(M) are regular.  1. L∪M is regular; we must prove that rev(L∪M) is regular.  rev(L∪M) contains strings that are the reversal of a string in L or M. That is, Rev(L∪M) = Rev(L) ∪ Rev(M). These are regular by IH, and Rev(L) ∪Rev(M) is regular by definition.  2. LM is regular; we must prove rev(LM) is regular. rev(LM) consists of strings of the form (ab)R, where a is in L and b is in R. Now use the identity (ab)R = bR aR. That is, Rev(LM) consists of strings that are the reversal of a string from M followed by the reversal of a string in L: Rev(LM) = rev(M)rev(L); by IH, rev(L) and rev(M) are regular, and by the definition, rev(M)rev(L) is regular.  3. L\* is regular; we must prove rev(L\*) is regular. A string w is in rev(L\*) iff w = (w1w2w3...wn)R, where each wi is in L.  But (w1w2w3...wn)R = wnR w\_{n-1}^R w\_{n-2}^R ... w3^Rw2^Rw1^R, which is exactly those strings in rev(L)\*. By the IH, rev(L) is regular, so rev(L)\* is regular by the recursive definition. | **Iterative partial correctness**  mystery(n):  “Pre: n is an integer >= 0 Post: returns count = n // 2”  count = 0  m = n  while m > 1:  m = m - 2  count = count + 1  return count  **Prove the loop invariant m + count ∗ 2 = n and m ≥ 0.**  **Base case:** When we reach the loop, we must prove  n+0\*2 = n and n >= 0 which holds from the precondition.  **Induction step: assume the invariant and guard are true**  Assume m0+count0\*2 = n and m0 >= 0andm0 >= 2 are true, 同时满足这些条件的就是m0 >= 2， we must prove m1+count1\*2 = n and m1 >= 0 (两个部分分别证)  表示出variable的变化: m1 = m0-2 count1 = count0+1  First, we prove that m1 >= 0 where m1= m0-2 >= 0.m1能进到loop说明至少是2，so this is true by the loop guard.  Now we prove m1+count1\*2 = n, 就把1的term用之前我们得到的m1和m0的关系式代掉  m1+count1\*2  =(m0-2) + (count0+1)\*2  = m0 - 2 + 2count0 + 2  = m0 + count0\*2 = m0  = n, by invariant  **Check the post-condition: use invariant and ~loop guard**  m+count\*2 = n and m >= 0 and m ≤ 1同时满足这几个式子的就是m + count\*2 = n and (m = 0 or m = 1) 有两个条件！  If m = 0, then count\*2 = n (this means n is even),  So, count = n//2.  If m = 1, then 1+count\*2 = n (this means n is odd)  = n//2, as n is odd | **Karatsuba Multiplication**  假如x=5678，y=1234, 把这两个数分成原本length的一半  n = max(len(x), len(y))  nby2 = n / 2 这个就是我们要从什么位置分x和y  a = x / 10^(nby2) 🡪a=56  b = x % 10^(nby2)🡪b=78  c = y / 10^(nby2)🡪c=12  d = y % 10^(nby2)🡪d=34  Step1:算ac=672 Step2：算bd=2652 都是直接recursion  Step3：算(a+b)(c+d)=6164 step4:算step3-2-1=2841  Step5：prod = ac \* 10^(2\*nby2) + (step4\*10^nby2) + bd  也就是6720000(加4个0)+2652+284000(加两个0)=results  **prove a language is not regular by proving the minimum number of states needed by the DFA is infinite.**  Prove no DFA accepts L = {0n1n | n ∈ N}. We use the following infinite set of strings: 0, 00, 000, 0000, . . .; i.e. one or more 0s. We let 0j be the string of j 0s. Any DFA is required to have a finite number of states; so any DFA must place at least two of these strings into the same state. Take i < j, and suppose for contradiction that a correct DFA for L exists and puts 0i and 0j into the same state. Now, add the suffix 1i to each string, yielding 0i1i and 0j1i. The DFA puts these two strings into the same state but, as i≠ j, one of the two strings is in the language while the other is not. This is a contradiction: a state cannot be both an accepting and non-accepting state. Therefore, there is no DFA for L. |