Induction

Proof Induction

prove basic(f(0)=x)

prove induction step, assume f(n) is true, prove f(n+1) based on that

proved

$$x_1,\ x_2 = \frac{-b\pm\sqrt{b^2-4ac}}{2a}$$

$$x_1+x_2 = \frac{-b+\sqrt{b^2-4ac}}{2a}+\frac{-b-\sqrt{b^2-4ac}}{2a}=-b/a$$

$$x_1x_2=\frac{-b+\sqrt{b^2-4ac}}{2a}\times\frac{-b-\sqrt{b^2-4ac}}{2a}=c/a$$

LRR (liner recurrence relation) of order 1(the x_n depends on x_{n-1})

- **Equation:** $x_n = ax_{n-1} + b\ \forall n \geq 1$, and x_0 is a constant given value
  - a is a given constant number, i.e., it does not depend on n
  - b is a given number, which may or may not be constant

x_0 is called the **initial value**

- Solution of the recurrence relation of order 1:

If b is **constant**:

$x_n = \left(x_0 - \frac{b}{1-a}\right)a^n + \frac{b}{1-a}$ if $a \neq 1$

$x_n = nb + x_0$ if $a = 1$

If b is **NOT constant**: $x_n = Aa^n + \hat{x}_n$ where

$\hat{x}_n$ is special solution in the **same form/family as b**:

1. From $\hat{x}_n = a\hat{x}_{n-1} + b$, derive equations to determine $\hat{x}_n$
2. $A = x_0 - \hat{x}_0$

$x_n = Aa^n$ + sp.x_n

$x_n=ax_{n-1}+b$

solve b,A using x_n,x_{n-1}

$x_n=2x_{n-1}+2^n$, $x_0=0$

a=2， $b=2^n$

sp.$x_n=c2^n+d$

$A2^n+c2^n+d=2(A2^{n-1}+c2^{n-1}+d)+2^n= A2^n+c2^n+2d+2^n$

$d=-2^n$

d is not constant. not working

use sp.x_n = (cn+d)2^n+e

$A2^n$+ (cn+d)2^n+e =2($A2^{n-1}$+ (cn-c+d)2^{n-1}+e)+2^n

A+cn+d=A+cn-c+d+1

e=2e,e=0

c=1

LRR of order2(x_n depends on x_{n-1},x_{n-2})

SUMMARY: LINEAR RECURRENCE RELATIONS OF ORDER 2

- **Equation:** $x_n = ax_{n-1} + bx_{n-2} + c\ \forall n \geq 2$; x_0, x_1, a, b are given constants, c is given

	Case: $a^2 + 4b > 0$	**Case:** $a^2 + 4b = 0, a \neq 0$
If $c = 0$	1. Solve $s^2 - as - b = 0$: $s_1 = \frac{a+\sqrt{a^2+4b}}{2}$, $s_2 = \frac{a-\sqrt{a^2+4b}}{2}$ 2. $x_n = As_1^n + Bs_2^n$ (A, B are TBD next) 3. $\begin{Bmatrix} x_0 = As_1^0 + Bs_2^0 \\ x_1 = As_1^1 + Bs_2^1 \end{Bmatrix} \Rightarrow A = \frac{x_1 - s_2 x_0}{\sqrt{a^2+4b}}$, and $B = \frac{x_0 s_1 - x_1}{\sqrt{a^2+4b}}$ 4. Final solution: $x_n = As_1^n + Bs_2^n$	1. Solve $s^2 - as - b = 0$: $s = \frac{a}{2}$ 2. $x_n = (A + Bn)s^n$ (A, B are TBD next) 3. $\begin{Bmatrix} x_0 = (A + B \times 0)s^0 \\ x_1 = (A + B \times 1)s^1 \end{Bmatrix} \Rightarrow A = x_0, B = \frac{2x_1}{a} - x_0$ 4. Final solution: $x_n = \left(x_0 + \left(\frac{2x_1}{a} - x_0\right)n\right)\left(\frac{a}{2}\right)^n$
If $c \neq 0$	1. Solve $s^2 - as - b = 0$: $s_1 = \frac{a+\sqrt{a^2+4b}}{2}$, $s_2 = \frac{a-\sqrt{a^2+4b}}{2}$ 2. $x_n = As_1^n + Bs_2^n + \hat{x}_n$ (A, B and $\hat{x}_n$ are TBD next) 3. Set the $\hat{x}_n$ form to be in the same form as c 4. From $\hat{x}_n = a\hat{x}_{n-1} + b\hat{x}_{n-2} + c$ derive equations to determine $\hat{x}_n$ 5. $\begin{Bmatrix} x_0 = As_1^0 + Bs_2^0 + \hat{x}_0 \\ x_1 = As_1^1 + Bs_2^1 + \hat{x}_1 \end{Bmatrix} \Rightarrow \begin{Bmatrix} A + B = x_0 - \hat{x}_0 \\ s_1 A + s_2 B = x_1 - \hat{x}_1 \end{Bmatrix} \Rightarrow$ $A = \frac{x_1 - \hat{x}_1 - s_2(x_0 - \hat{x}_0)}{\sqrt{a^2+4b}}$, and $B = \frac{(x_0 - \hat{x}_0)s_1 - (x_1 - \hat{x}_1)}{\sqrt{a^2+4b}}$ 6. Plug in s_1, s_2, A, B, and $\hat{x}_n$ to get final solution: $x_n = As_1^n + Bs_2^n + \hat{x}_n$	1. Solve $s^2 - as - b = 0$: $s = \frac{a}{2}$ 2. $x_n = (A + Bn)s^n + \hat{x}_n$ (A, B and $\hat{x}_n$ are TBD) 3. Set the $\hat{x}_n$ form to be in the same form as c 4. From $\hat{x}_n = a\hat{x}_{n-1} + b\hat{x}_{n-2} + c$ derive equations to determine $\hat{x}_n$ 5. $\begin{Bmatrix} x_0 = (A + B \times 0)s^0 + \hat{x}_0 \\ x_1 = (A + B \times 1)s^1 + \hat{x}_1 \end{Bmatrix} \Rightarrow$ $A = x_0 - \hat{x}_0$, and $B = \frac{x_1 - \hat{x}_1 - (x_0 - \hat{x}_0)s}{s}$ 6. Plug in s, A, B, and $\hat{x}_n$ to get final solution: $x_n = (A + Bn)s^n + \hat{x}_n$

Introduction

$$T(n) = aT\left(\frac{n}{b}\right) + f(n) \text{ for } n > n_0$$

Then $T(n)$ has the following asymptotic bounds:

Please brush up on logarithms

- If $f(n) = O(n^{\log_b a - \varepsilon})$ for some constant $\varepsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$.
- If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \log n)$.
- If $f(n) = \Omega(n^{\log_b a + \varepsilon})$ for some constant $\varepsilon > 0$, and if $af\left(\frac{n}{b}\right) \leq cf(n)$ for some constant c<1 for all sufficiently large n, then $T(n) = \Theta(f(n))$.

By default, log is base 2: $\log n = \log_2 n$

$$T(n) = 2T(n/2) + cn$$
$$\Rightarrow T(n) = O(n \log n)$$

- **Stirling's Approximation:** $n! \cong \sqrt{2\pi n}\left(\frac{n}{e}\right)^n$, where e=2.718… is the base of natural logarithm
- Useful summation formulas:
  - $1+2+3+\cdots+n = \frac{n(n+1)}{2}$
  - $1^2+2^2+\cdots+n^2 = \frac{n(n+1)(2n+1)}{6}$
  - $1^3+2^3+\cdots+n^3 = \left(\frac{n(n+1)}{2}\right)^2$
  - $1^k+2^k+\cdots+n^k = O(n^{k+1})$, where k is a positive constant integer
  - $1+x+x^2+x^3 \ldots + x^n = \frac{x^{n+1}-1}{x-1}$, for all $x \neq 1$.
  - $1+2x+3x^2 \ldots + nx^{n-1} = \frac{nx^{n+1}-(n+1)x^n+1}{(x-1)^2}$, for all $x \neq 1$.
  - $(a+b)^n = \binom{n}{n}a^n b^0 + \binom{n}{n-1}a^{n-1}b^1 + \binom{n}{n-2}a^{n-2}b^2 + \cdots + \binom{n}{k}a^{n-k}b^k + \cdots + \binom{n}{0}a^0 b^n$

$$n! = 1 \times 2 \times 3 \times \cdots \times n$$
$$\binom{n}{k} = \frac{n!}{k!\,(n-k)!}$$

34

Data structure

stack pop push top, queue dequeue,enqueue

Define a record type ***employee***:
record employee
begin
    char name[1:30];
    int SSN;
    char address[1:100];
    float salary;
    employeePtr next; // a new field
end

Tree height: level-1, (root-> level0)

- perfect binary tree: non-leaf have two child and same level
- Almost complete binary tree: last level has lack leafs from right

Binary search tree

- insert
  - search for a, if not exist, create the node containing a
  - see search and insert example
    - constant time in searching
- delete
  - delete pointer, attach the child to parent.

- pick largest value(rightmost) in left brach,use that node as the deleted
    - make left child to the previous parent
- use hash for maps?

MinHeap

- child must be larger than parent.
  - swap child and parent if not
  - become valid, or child become the root
- delete: swap with the last leaf and delete
  - then swap with the smaller leaf
- logN leve

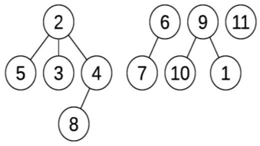
Union find Data structure

- every child has a parent.
- construct an arrry of index(element number) and it's own parent. use number to find which set it is in.
- single tree is the worst case in union find with only one array.
- path compression, as a find, make every node in the path to point directly to the parent.O(n)

-- 2[ND] IMPLEMENTATION (UNION) --

- PARENT array of this collection:

i	1	2	3	4	5	6	7	8	9	10	11
PARENT	9	**-5**	2	2	2	**-2**	6	4	**-3**	9	**-1**



- At the start, PARENT[i]$= -1\ \forall i$, why?

i:	1	2	3	4	5	6	7	8	9	10	11
PARENT[i]:	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1

DC

- mergesort
  - breakdown to single pieces and merge sorted list
- **TIME COMPLEXITY OF MERGESORT**

  -- SOLVING THE RECURRENCE RELATION (2) --

$$T(n) = 2T\left(\frac{n}{2}\right) + cn$$
$$2T\left(\frac{n}{2}\right) = 2^2T\left(\frac{n}{2^2}\right) + 2c\frac{n}{2}$$
$$2^2T\left(\frac{n}{2^2}\right) = 2^3T\left(\frac{n}{2^3}\right) + 2^2c\frac{n}{2^2}$$
$$\ldots$$
$$2^{k-1}T\left(\frac{n}{2^{k-1}}\right) = 2^kT\left(\frac{n}{2^k}\right) + 2^{k-1}c\frac{n}{2^{k-1}}$$

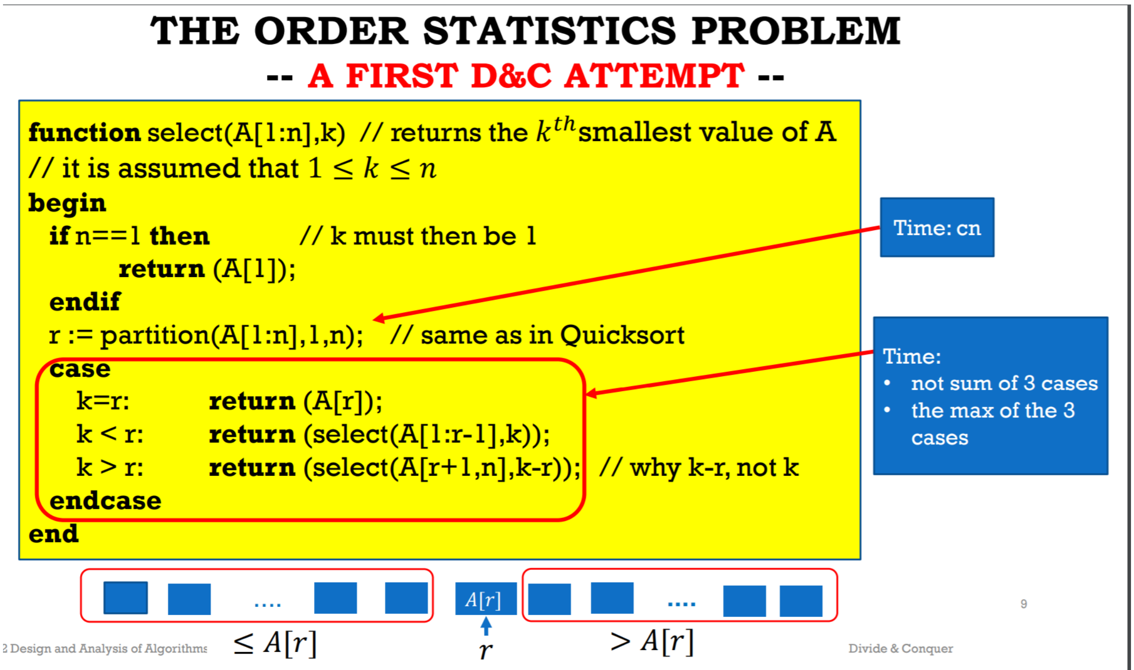
$$T(n) = \cancel{2T\left(\frac{n}{2}\right)} + cn$$
$$\cancel{2T\left(\frac{n}{2}\right)} = \cancel{2^2T\left(\frac{n}{2^2}\right)} + cn$$
$$\cancel{2^2T\left(\frac{n}{2^2}\right)} = \cancel{2^3T\left(\frac{n}{2^3}\right)} + cn$$
$$\ldots$$
$$\cancel{2^{k-1}T\left(\frac{n}{2^{k-1}}\right)} = 2^kT\left(\frac{n}{2^k}\right) + cn$$

  - Sum of left terms = sum of right terms
  - Cancel terms that occur on both sides of "="
  - What remains on the left is: $T(n)$
  - What remains on the right: $2^kT\left(\frac{n}{2^k}\right) + cnk = nT(1) + cnk$
  - Therefore: $T(n) = nT(1) + cnk = cn + cn\log n = O(n\log n)$
  - $\boldsymbol{T(n) = O(n\log n)}$

22

- Quicksort
  - p==q return

- r= partition A[p:q]
- quicksort A[p:r-1]
- quicksort A[r+1:q]

- select(sort)
- 

- Theorem: The time complexity $TT\ nn$ of QuickSelect(A[1:n],k) satisfies: $TT\ nn \leq 20cn$
- **A FEW OTHER QUICK D&C APPLICATIONS**
  -- POLYNOMIAL EVALUATION (2/3) --

  - D&C method:
    - Let $m = \frac{n}{2}$
    - $P(x) = a_0 + a_1x + a_2x^2 + \cdots + a_{m-1}\,x^{m-1} + a_mx^m + \cdots + a_{n-1}\,x^{n-1}$
    - $P(x) = [a_0 + a_1x + a_2x^2 + \cdots + a_{m-1}\,x^{m-1}] + [a_mx^m + a_{m+1}x^{m+1} + \cdots \ldots + a_{n-1}\,x^{n-1}]$
    - $P(x) = [a_0 + a_1x + a_2x^2 + \cdots + a_{m-1}\,x^{m-1}] + x^m[a_m + a_{m+1}x^1 + \cdots + a_{n-1}\,x^{n-m-1}]$
    - $P(x) = \qquad Q(x) \qquad + x^mR(x)$

      Where $Q(x) = a_0 + a_1x + a_2x^2 + \cdots + a_{m-1}\,x^{m-1}$, represented by $a[0{:}\,m-1]$

      and $R(x) = a_m + a_{m+1}x^1 + \cdots + a_{n-1}\,x^{n-m-1}$, represented by $a[m{:}\,n-1]$
    - Now we can call the algorithm recursively on $Q(x)$ and $R(x)$
    - Merging: compute x^m and then $P(x) = Q(x) + x^mR(x)$ and return $P(x)$;

greedy

MST(minimum spanning tree)

Use unionfind for circle check?

known edge:

tree mapping.(union find)

if edge's two nodes are in different trees, no circle problem

if same tree, no add.

- single source shortest path

GREEDY SSSP ALGORITHM

```
Procedure SSSP( in: W[1:n,1:n], s; out: DIST[1:n]);
begin
    for i =1 to n do:  DIST[i] := W[s,i]; endfor
    // implement Y as Boolean array Y[1:n] : Y[i]= 1 if i ∈ Y, 0 otherwise
    Boolean Y[1:n];       // initialized to 0
    Y[s] := 1;            // add s to set Y
    for num =2 to n do
        Select a node u from out of Y (i.e., Y[u]==0) such that
          DIST[u] = min {DIST[i] | Y[i] = 0};
        Y[u] := 1;        // Add u to Y
        // update the DIST values of the other nodes
        for all node v  where Y[v] = 0 do
            DIST[v]= min (DIST[v], DIST[u]+W[u,v]);
        endfor
    endfor
End SSSP
```

60

CS 6212 Design and Analysis of Algorithms   The Greedy method

- Heap sort
  - deletemin, and form a new array
- Proof of optimality
  - assume counter example exist
  - • The contradiction means that the assumption that DIST[u] ≠ distance(s,u) must be false • Hence, DIST[u] = distance(s,u).
-

Examples

Induction

Induction step: Assume $S(n-1) = \frac{(n-1)(n-1+1)(2(n-1)+1)}{6} = \frac{(n-1)n(2n-1)}{6}$, prove that $S(n) = \frac{n(n+1)(2n+1)}{6}$. (Note: The yellow-highlight portion is called the induction hypothesis (I.H.).)

$$
\begin{aligned}
S(n) &= 1^2 + 2^2 + 3^2 + \cdots + n^2 && \text{by definition} \\
&= 1^2 + 2^2 + 3^2 + \cdots + (n-1)^2 + n^2 \\
&= S(n-1) + n\text{^}2 \\
&= \frac{(n-1)n(2n-1)}{6} + n^2 && \text{by the induction hypothesis} \\
&= \frac{(n-1)n(2n-1) + 6n^2}{6} \\
&= \frac{n[(n-1)(2n-1) + 6n]}{6} \\
&= \frac{n[2n^2 - 3n + 1 + 6n]}{6} \\
&= \frac{n[2n^2 + 3n + 1]}{6} \\
&= \frac{n(n+1)(2n+1)}{6}. && \text{Q.E.D.}
\end{aligned}
$$

d. Let $T(n)$ be defined recursively as follows: $T(n) = 4T(\frac{n}{2}) + n^3$ (note that for sufficiently small n, $T(n)$ is bounded by a constant). Use the Master Theorem to find the Θ value of $T(n)$.

Solution:

$T(n) = 4T(\frac{n}{2}) + n^3$, let $f(n) = n^3 = \Omega(n^3) = \Omega(n^{\log_2 4+1})$, and $4f\left(\frac{n}{2}\right) = 4\left(\frac{n}{2}\right)^3 = 4\frac{n^3}{8} = \frac{n^3}{2} \leq \frac{1}{2}f(n)$. So we are in the third case of the Master Theorem.

According to the Master Theorem, $T(n) = \Theta(f(n)) = \Theta(n^3)$.

Bonus Problem:

Let c be a constant > 1, and let $T(1) = c - 1$, $T(2) = c$, and $T(n) = T(\lfloor\sqrt{n}\rfloor) + 1$ for all integer $n \geq 3$. Prove that $T(n) = O(\log\log n)$. Hint: Prove that $\forall n \geq 2, T(n) \leq c + \log\log n$, by induction on n.

Solution:

Basis step: For $n = 2$, $T(2) = c \leq c + \log\log 2$ because $c + \log\log 2 = c + \log 1 = c + 0 = c$.

Induction step:

Assumed that $T(k) \leq c + \log\log k$ for all integer $k \leq n-1$, prove that $T(n) \leq c + \log\log n$.

Since $T(k) \leq c + \log\log k \ \forall k \leq n-1$, then $T(\lfloor\sqrt{n}\rfloor) \leq c + \log\log\lfloor\sqrt{n}\rfloor$ b/c $\lfloor\sqrt{n}\rfloor \leq n-1$. We will use that below.

$T(n) = T(\lfloor\sqrt{n}\rfloor) + 1$	by definition of $T(n)$
$\leq c + \log\log\lfloor\sqrt{n}\rfloor + 1$	using what we conluded above about $T(\lfloor\sqrt{n}\rfloor)$
$\leq c + \log\log\sqrt{n} + 1$	because $\lfloor\sqrt{n}\rfloor \leq \sqrt{n}$
$= c + \log(\frac{1}{2}\log n) + 1$	because $\log\sqrt{n} = \log n^{\frac{1}{2}} = \frac{1}{2}\log n$
$= c + \log(\log n) + \log\frac{1}{2} + 1$	because $\log ab = \log a + \log b$
$= c + \log(\log n) - 1 + 1$	because $\log\frac{1}{2} = -\log 2 = -1$
$= c + \log\log n$	

Therefore, $T(n) \leq c + \log\log n$.

This implies that $T(n) = O(\log\log n)$ by definition of big O.

a)

```
func pow(x, n)
begin
    if n = 0 then return 1; endif
    // now, n ≥ 1.
    y := pow(x, ⌊n/2⌋);
    if n is even then
        return y*y;
    else
        return y*y*x;
    endif
end
```

$T(n) = T(n/2) + c \Rightarrow T(n) = O(\log n)$.