Graph←→Digraph

<u,v>: u→v: <u,v> is incident from u and is incident to v(Dafaq…)

Component: Maximal subgraph bla

Acyclic: no cycle; δ(s,v): the dist from s to t.

W:未遍歷 G:相鄰皆未遍歷 B:相鄰皆已遍歷

※BFS 可找到最短證明:
引: edge(u,v)∈E,δ(s,v)≦δ(s,u)+1(反證)
引: BFSv.d=u.d+1≧δ(s,u)+1≧δ(s,v)(歸納)
引: rear.d≦front.d+1 and q_i.d≦q_{i+1}.d(歸納)
推: v_i 在 v_j 前 enq→j enq 後，v_i.d≦v_j.d
證明: (反證)v 是不合法中 δ(s,v)最小者
找 v 的前一個 u，分析 u 的顏色得到矛盾

parenthesis structure: 子孫會被祖先括住

white-path(v 是 u 子孫 iff 設 u.d 時有 u→v 白路

白路證明:" ←": (反證)取 v 在白路上前一個點 w，證明 u.f≧w.f>v.d>u.d，故 v 爲 u 子孫。

Tree Edge→DFS Tree 上

Back Edge→連接到祖先 or Self-loop

Forward Edge→連接到子孫、non-tree edge

Cross Edge→其他

(u,v)看 v 的顏色: W→T、G→B、B→F or C

Activity on Vertex vs Activity on Edge

Topological Sort 正確性：(依照 v.f)
引: G is acyclic iff G's DFS no back edge.
證明: 對一個<u,v>，v.f<u.f:
v not G→否則<u,v>爲 back edge
if v is W→v 是子孫、if v is B→v 已經結束

Biconnected→no Articulation Point

low(u)=min{u.d, min{low(w)|w is child}, min{w.d|(u,w)is back edge}}

當有 child u 使得 low(u) ≧v→v is AP

Kruskal 正確性(1)產生出的是 SP(2)最小
(1) 有 SP iff connected.
(2) T、U(MST)有 k 條邊差異: (對 k 歸納)
將不在 U 中之邊加入 U 找圈 比較 cost

Dijkstra 正確性: 若從 v_0 到 v 有經過一個未選過點 w，因爲 δ(v_0,w)< δ(v_0,v)，w 應該已被選。

Bellmanford:
d^{k}[u]=min{d^{k-1}[u], min{d^{k-1}[i]+cost<I,u>}}

Internal Sort←→External Sort

Stabilty: same key; In-place: O(1)add memory

Adaptability: 部份已序→複雜度較低

Comparison-based sorting:

Decision tree of N! element→lg(N!)=nlgn

選擇:in-place

插入:常數低, stable, in-place, adaptive, online

Quick: choose a pivot

T(n)=2T(n/2)+O(n) →T(n)=O(nlgn)

1. 只要切分有比例便是 nlgn
2. 分得好得會吸收分得壞得，只是常數大

Radix: Most Significant Digit first sort

Bucket: 從 uniform distribution 取出、分 bucket 各自 insertion。 Since E[n_i^2]=2-1/n,
E[T(n)]=E[O(n)+ΣO(n_i^2)]=O(n)+nO(2-1/n)=O(n)

Hash: |K|<<|U|, Direct-address 造成浪費

Open addressing:
1. Linear probing: h(k)+i % m→clustering(集結)
2. Quadratic probing: h(k,i)=h'(k)+c_1i+$c_2$$i^2$ % m
3. Double hashing: h(k,i)=h_1(k)+ih_2(k) % m

接近 uniform hashing(任意 probing seq 機率近)

uniform hashing's expected num of probes:

α:load factor=n/m < 1

失敗: 1+α+α^2+α^3+⋯=1/(1-α)

成功: (1/α)ln(1/(1-α))

Chaining: 可用平衡樹 bla, avg. chain 大小:α

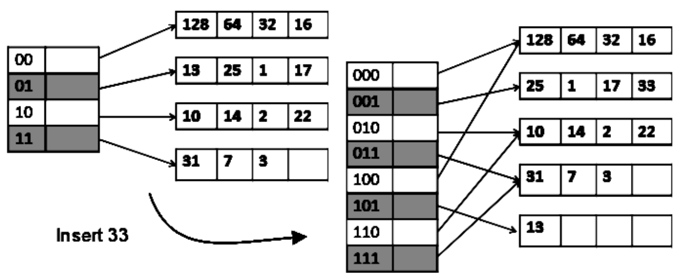
失敗: O(1+α)

成功: 1+(α/2)- (α/2n)=O(1+α)

Hash func:
1. Division: h(k)=k%D
2. Mid-square: h(k)=bits$_{i,\ i+r-1}$(k^2), 櫃子數 2^r
3. Shift Folding: 切好幾段、總和% D
4. Folding at the boundaries: 奇段反轉
5. Digit Analysis: 已知 key,分析最平均前幾 digit
6. Multiplication Method: h(k)=[m(kA%1)],0<A<1

Dynamic hashing: (extendible hashing)

Using directories:



Directoryless:

$0\sim q-1$:	$q\sim 2^r-1$:	$2^r\sim 2^r+q-1$
h(k,r+1)	h(k,r)	h(k,r+1)

當前櫃滿: 開一個新的櫃(2^r+q), split 櫃 q
當前櫃若仍超出, 暫時用 chain

String Matching:
Σ^*: 使用Σ的有限長度字串(包含空)
|x|: the length of string x
xy: concatenation string x and y
$w \sqsubset x$: w is x's prefix, i.e. x=wy, $y\in\Sigma^*$
$w \sqsupset x$: w is x's suffix, i.e. x=yw, $y\in\Sigma^*$
Naïve: worst O(|T||P|), avg O(|T|-|P|)
Rabin-Karp: O(|T|)preprocessing+O(|T|-|P|)
$t_0=T[m]+10(T[m-1]+10(\cdots+10T[1])\cdots)\ \%\ q$
$t_{s+1}=10(t_s-10^{m-1}xT[s+1])+T[s+m+1]\ \%\ q$
Knuth-Morris-Pratt:
Failure function π :
π [q]=max{k: k<q and P_k is a suffix of P_q}

m←P.length	
let π [1…m] be a new array	
π [1] ←0, k←0	
for q=2 to m	
while k>0 and $P_{k+1}\neq P_q$	k 至少-1
k←π [k]	
if $P_{k+1}=P_q$	→k<q
k←k+1	
π [q]←k	
return π	total: O(m)

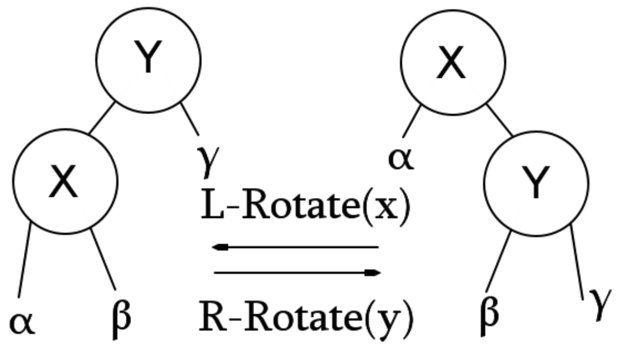
Red-black Tree

Properties:
1. Every node is either red or black.
2. The root is Black.
3. a node is black→ both its children is red
4. All the leaves (external node) are black.
5. For each node, all path from the node to descendant leaves contain the same number of black nodes.

Black height: bh(x): x 到任意子孫 leaf 遇到的 black node 個數(不包含 x 自己)
The bh of a external node(NIL) is 0.

證明: n 個 node 的 RBT, h 最深爲 2lg(n+1)
引: x 的 subtree 最少有 $2^{bh(x)}-1$ 個 int.node
(對 bh 歸納)$(2^{bh(x)-1}-1)+(2^{bh(x)-1}-1)+1=2^{bh(x)}-1$
Prop4. = node 到 leaf 至少一半黑的.
bh(root)≧h/2, root 下至少有 $2^{h/2}-1$ 個 node.

Rotate:



Note that $\alpha<x<\beta<y<\gamma$.
Insert z(red): might violate Prop.2, Prop.4

violate Prop.2: 整棵樹只有 z.

violate Prop.4: 爸爸是紅的, so 爺爺黑的
不失一般性假設爸爸是左小孩(右小孩鏡射)
Cond1: 叔叔紅的→
抹黑爸爸跟叔叔, 爺爺紅了, $\boxed{\text{向上傳遞}}$
Cond2: 叔叔黑的, 自己是右小孩→
L-Rotate(爸爸), R-Rotate(爺爺).
Cond3: 叔叔黑的, 自己是左小孩→
R-Rotate(爺爺).

Delete z:
Def. x:移動到 y 位置的, z:要刪的
y: (Deg_z=2: 移到 z 的, else:被砍的), 記顏色

y: 紅→因爲 y 上下都黑, y 自己紅, 非 root
→不會 violate property.

y: 黑→可能 violate property
1. y 是 root, 他的紅 child 變 root
2. y 上下皆紅
3. y 的祖先 bh 不一致

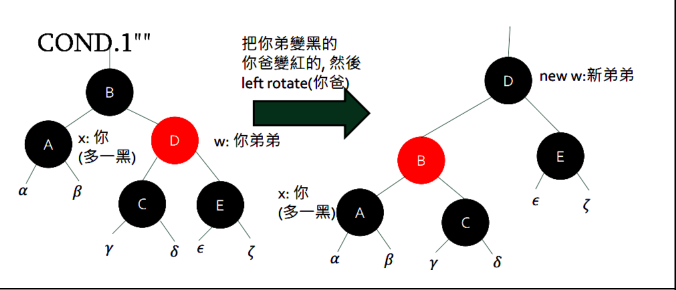
y 的黑色被趕到 x, 但 x 可能原本爲紅

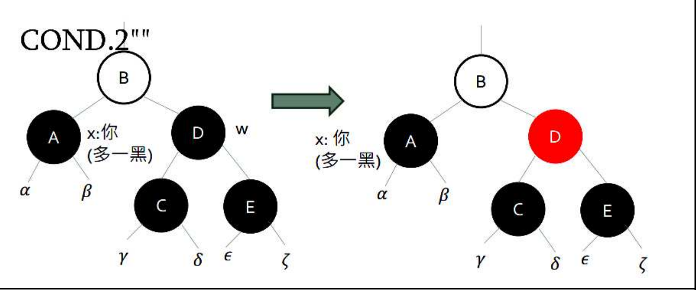
x: 紅, y: 黑→直接將 x 改成黑
x: 黑, y: 黑且 x 是 root→直接將 x 改成黑
x: 黑, y: 黑且 x 非 root: 在 x 角度考慮：
一樣只考慮你是左小孩(下頁)

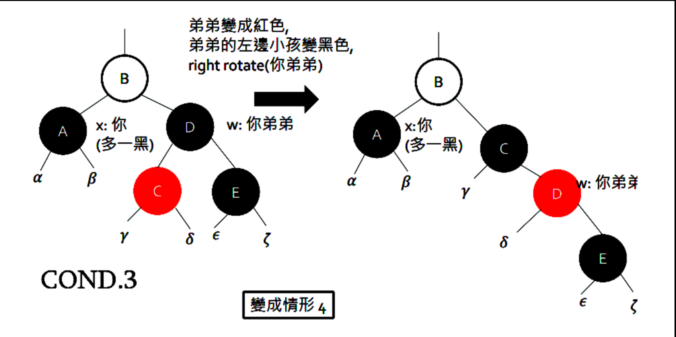
Cond1: 弟弟是紅
把弟弟抹黑，爸爸紅了，然後 L-Rotate(爸爸)
→轉成其他 Cond.
Cond2: 弟弟是黑，姪子們都是黑
弟弟紅了
→(爸黑)結束 or (爸紅)往上傳
Cond3: 弟弟是黑，姪子左紅右黑
弟弟紅了，抹黑弟弟左小孩, R-Rotate(弟弟)
→轉成 Cond4.
Cond4: 弟弟是黑，右邊姪子紅的
弟弟變爸爸的顏色，爸爸抹黑，弟弟右邊小孩抹黑, L-Rotate(爸爸)

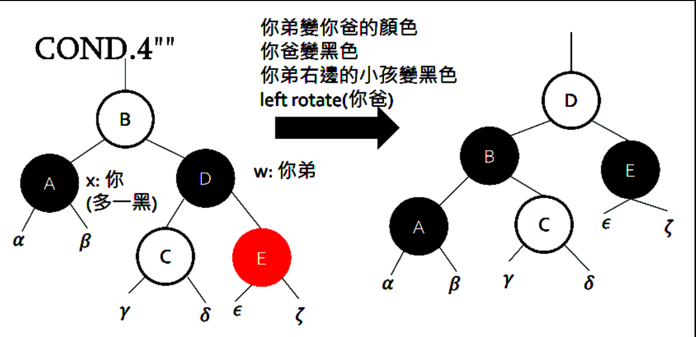
About GDB:
Compile:
gcc -g [your program] -o [executable file]
Start:
gdb / gdb [executable file]

```
>    file [executable file]
>    start(開始執行，停在 main 入口)
>    run(開始執行至遇到 break point)
>    list(list part of the code of your program)
>    break [location(行號)function] (set 斷點)
>    break [location/function] if [condition]
>    print [variable]
>    continue (continue executing)
>    watch [variable] (set watchpoint)
>    set [variable] = [value]
>    list break (list break&watch point)
>    delete [point number] (delete point)
```









At any given time, I can only remember two bugs.
Keeping a database of bugs is one of the hallmarks of a good software team.
Without an organized database listing all known bugs in the code, you are simply going to ship low quality code.

Good bug report:

1. Steps to reproduce,
2. What you expected to see, and
3. What you saw instead.

Ten tips

1. Reduce the repro steps to the minimal steps.
2. The only person who can close a bug is the person who opened it in the first place. Anyone can resolve it, but only the person who saw the bug can really be sure that what they saw is fixed.
3. There are many ways to resolve a bug. For example: fixed, won't fix, postponed, not repro, duplicate, or by design.
4. Not Repro means that nobody could ever reproduce the bug. Programmers often use this when the bug report is missing the repro steps.
5. You'll want to keep careful track of versions.
6. If you're a programmer, and you're having trouble getting testers to use the bug database, just don't accept bug reports by any other method.If your testers are used to sending you email with bug reports, just bounce the emails back to themwith a brief message: "please put this in the bug database. I can't keep track of emails."
7. If you're a tester, and you're having trouble getting programmers to use the bug database, just don't tell them about bugs - put them in the database and let the database email them.
8. If you're a programmer, and only some of your colleagues use the bug atabase, just start assigning them bugs in the database. Eventually they'll get the hint.
9. If you're a manager, and nobody seems to be using the bug database that you installed at great expense, start assigning new features to people using bugs. A bug database is also a great "unimplemented feature" database, too.
10. Avoid the temptation to add new fields to the bug database. For the bug database to work, everybody needs to use it, and if entering bugs "formally" is too much work, people will go around the bug database.