Red-Black Tree

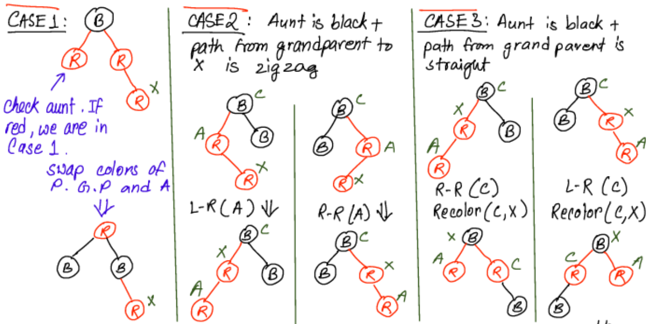
① Every node is red or black
② Root is black
③ Leaves (nil) are black
④ If a node is red, both children are black
⑤ All simple path, from any node x, excluding x, to a descendant leaf has the same # of black nodes. [black-height (x)]

Time complexity = $O(\log n)$

Theorem

→ A Red back-tree with n keys has height $h \leq 2 \cdot \log(n+1)$
→ For each path of length h, there are at most ½ red nodes. (Property 4) so, $bh \geq h/2$
→ The # of nil leaves is $\leq n+1$
→ $n+1 \geq 2^{bh} \Rightarrow \log_2(n+1) \geq bh$
$\log_2(h+1) \geq bh \geq h/2$
$\Rightarrow h \leq 2\log_2(n+1)$

Insertion : Insert node as a red node. Only property 4 can be violated.



Number of vertices in a red black tree with all black nodes : $2^{bh} - 1$
Number of vertices in a red black tree with alternating black-red nodes : $2^{2bh} - 1$
Numer of internal red nodes in alternating R-B tree : $2 + 8 + 32 \cdots + n$
$\Rightarrow 2(1+4+16)$
h = black height $\Rightarrow 2 \cdot \sum_{i=0}^{h} 4^i = 2\left(\frac{4^h - 1}{3}\right)$

Number of internal black nodes : $1 + 4 + 16 + 64 \Rightarrow \sum_{i=0}^{h} 4^i = \frac{4^h - 1}{3}$

Sort

k = range of each element

→ Counting sort : if n is bounded and small then use counting sort. Running time : $O(n+k)$. It is a stable sort. $k = n$

→ Radix Sort : Sort from LSD to MSD. if n is not bounded and range is given for each digit the use Radix Sort. $O(n)$

→ Merge Sort : $O(n \cdot \log n)$ → Insertion sort : $O(n^2)$

if we convert decimal to binary or hexadecimal.
* decimal → binary ① # of digits for radix sort increases.
② Range of values for counting sort decreases.
* decimal → hex ① # of digits for radix sort decreases
② Range of values for counting sort increases.

Q1) An array of n binary numbers: Binary numbers can have an unbound range, so we can use merge sort for $\Theta(n \log n)$
Q2) An array of n credit card numbers: 16 digits between 0 to 9. So we use radix sort. Runtime will be $O(n+k) = O(n)$
Q3) An ranking of n candidates for a job: Here $k = n$ (ranking are 1, 2, 3, 4 . . . n) which implies radix or counting sort. Run time = $O(n)$.
Q4) A list of n UTSA students by their banner IDs: 8 digits between 0 to 9. So we use radix sort. Runtime will be $O(n+k) \simeq O(n)$
Q5) An array of n natural numbers: Range is unknown. Therefore we use merge sort to get $O(n \log n)$.
Q6) An array of n students by their grades on an exam: n students with range 0 to 100; $k = 101$. Therefore counting sort will be $O(n)$ time.
Q7) A list of n sports team according to their rank: Here $k = n$ (ranking are 1, 2, 3, 4, ... n) which implies radix or counting sort. Runtime = $O(n)$.
Q8) An array of n rational numbers: Range is not given, which is unbound. So we use merge sort, runtime = $O(n \cdot \log n)$
Q7) A phone book consisting of n telephone numer: 10 digit number, each ranging between 0 to 9. So we use radix sort, runtime = $O(n+k) \simeq O(n)$
Q8) A array of n numbers in the range $[0 \ldots n^2]$: Radix Sort, Convert numbers to base $\log n$. Runtime $O(n)$

DP : ① Hotel problem:

1) There are 2^n different subsets of hostel and each subset takes $O(n)$ time to check its feasibility and costs. Therefore $O(n \cdot 2^n)$
2) $a[i] = \begin{cases} 0 & \text{if hostel } i \text{ is} \leq 20 \text{ miles from start} \\ \min\limits_{k \in H(i)} (C_k + a[k]) & \text{otherwise} \end{cases}$

$H(i)$ are the hostels within 20 miles before hostel i.

3) Assuming a dummy hostel h_{n+1} at the very end of the trail.

```
for (i = 1 to n+1)
{ if (hi is ≤ 20 miles from the start)
  { a[i] = 0 } end if
  else
  { mincost = ∞
    j = i - 1
    while (hi - hj ≤ 20)
    { if (a[j] + Cj < mincost)
      { mincost = a[j] + Cj } endif
      j--
    } end while
    a[i] = min cost
  } end else
} end for
return a[n+1]
```

#LRS :
$LRS[i][j] = \begin{cases} 0 & \text{if } (i = 0 \text{ or } j = 0) \\ LRS[i-1][j-1] + 1 & \text{if } (x[i-1] = x[j-1] \,\&\, x_i = x_j) \\ \max(LRS[i-1][j], LRS[i][j]) & \text{if } (x[i-1] \,!= x[j-1]) \end{cases}$

LC Sub-str
$C[i][j] = \begin{cases} C[i-1][j-1] + 1 & \text{if } (x[i-1]) = y[j-1]) \\ 0 & \text{otherwise} \end{cases}$

Trip problem:

1) There are 2^n different ways of picking station and it takes $O(n)$ time to determine if a choice is feasible and compute the cost. Therefore, $O(n \cdot 2^n)$
2) $C[1] = 20$, $C[2] = 50$, $C[3] = 20 + 70 = 90$, $C[4] = 50 + 30 = 80$
3) $C[i] = \begin{cases} C_i & \text{if } M_i \leq 300 \\ C_i + \min\limits_{k \in S(i)} \{C[k]\} & \text{otherwise} \end{cases}$

$S(i)$ set of all gas station within 300 mile before St_i

4)
```
for (i = 1 to n)
{ if (Mi ≤ 300)
  { C[i] = Ci } endif
  else { j = i - 1
    min = i - 1
    while (mi - mj ≤ 300)
    { if (C[j] < C[min])
      { min = j } end if
      j--;
    } end while
    C[i] = C[min] + Ci
  } end else
} end for
```

```
j = n - 1
min = n - 1
while (M - mj ≤ 300)
{ if (C[min] > C[j])
  { min = j } end if
  j--;
} end while
return C[min]
```

B-tree : Values for each node = $2t - 1$, Root must store at least 1.

① Every node except the root, stores $t - 1 \leq$ # of element $\leq 2t - 1$
② each node has at most $2t$ child nodes, minimum t child nodes.
③ children of a node = # of elements + 1 (except leaf)

Theorem: A B-tree with minimum degree $t > 2$ which stores n values has height $h \leq \log_t \frac{n+1}{2}$.

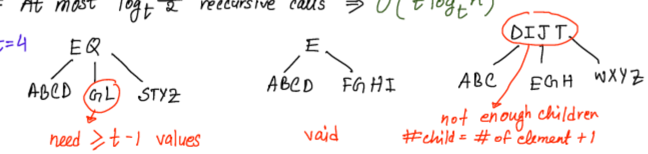
Proof: # of nodes $\geq 1 + 2 + 2t + 2t^2 + 2t^3 \ldots + 2t^{i-1} + \ldots + 2t^{h-1}$
$= 1 + \sum_{i=0}^{h} 2t^{i-1} = 1 + \sum_{i=1}^{h-1} 2t^i = 1 + 2\left[\frac{t^h - 1}{t - 1}\right]$

# of values $= n \geq 1 \cdot 1 + 1 \cdot 2\left[\frac{t^h - 1}{t - 1}\right] \cdot t - 1 = 2t^h - 1$

$\Rightarrow n \geq 2t^h - 1 \Rightarrow \frac{n+1}{2} \geq t^h \Rightarrow \log_t\left(\frac{n+1}{2}\right) \geq h$

At most $\log_t \frac{n+1}{2}$ recursive calls $\Rightarrow O(t \log_t n)$

#t=4



Subsequence palindrome

1) There are 2^n different ways we can create a subset win n characters. Then to determine whether it's a palindrome or not, we need $O(n)$ time. Therefor $O(n \cdot 2^n)$
2) $C[1,1] = 1$, $C[2,2] = 1$, $C[3,3] = 1$, $C[1,2] = 1$, $C[2,3] = 1$, $C[1,3] = 3$
3) $C[i][j] \begin{cases} 1 & \text{if } i == j \\ \max(C[i][j-1], C[i+1][j]) & \text{if } S_1[i] \neq S_2[j] \\ \max(C[i][j-1], C[i+1][j]) + 2 & \text{if } S_1[i] == S_2[j] \end{cases}$

LCS Theorem:

$C[i][j] = \begin{cases} C[i-1][j-1] + 1 & \text{if } X[i] = Y[j] \\ \max(C[i-1][j], C[i, j-1]) & \text{otherwise} \end{cases}$

Matrix-mul Theorem

$M[i,j] \begin{cases} 0 & \text{if } i == j \\ \min\limits_{i \leq k < j} (M[i,k] + M[k+1, j] + P_{i-1} \times P_k \times P_j) \end{cases}$

$P[n]$ = array sequence $\{3, 20, 5, 8\}$

```
Mul (i, j)
{ if (i == j)
  { M[i, j] = 0 } end if
  else
  { min = ∞
    for k = 1 to k < j
    { X = Mul (i, k) + Mul (k+1, j) + P[i-1] × P[k] × P[k+1]
      if (X < min)
      { min = X } end if
    } end for
    M[i, j] = min
  } end else
  return M[i, j]
}
```

coin change:
$DP[i][j] = (i / K[j]) + dp[i][j-1]$
Since i represents n and $k[j]$ represen the current coin

coin-change

① We can make multiple copies of each coin. Certainly, we would not need more than n/d_i copies of coin d_i. Considering every way of choosing these coins, Therefore, n^m where at most n copies of each coin over m different coins
② $a[1] = 1$, $a[2] = 2$, $a[3] = 3$, $a[4] = 1$, $a[5] = 1$, $a[6] = 2$, $a[7] = 3$
③ $a[i] = \begin{cases} 0 & \text{if } i = 0 \\ 1 & \text{if } i = d_j \text{ for some } j \\ \min\limits_{j : d_j \leq i} \{ a[i - d_j] + 1 \} & \text{otherwise} \end{cases}$

④
```
a[0] = 0
for j = 1 to m
  { a[dj] = 1 }
for (i = 2 to n)
{ if (a[i] == NULL)
  { min = n;
    for (j = 1 to m)
    { if (dj ≤ i AND a[i-dj] < min)
      { min = a[i-dj] j. } end if
    } end for
    a[i] = min + 1
  } end if
} end for
return a[n]
```

Runtime = $O(n^2)$

Rod cutting

$dp[i] = \begin{cases} C_i & \text{if } i == \text{inch } (1, 2 \text{ or } 3) \\ \max\{\max(C_k + C_{n-k}), C_n\} \end{cases}$