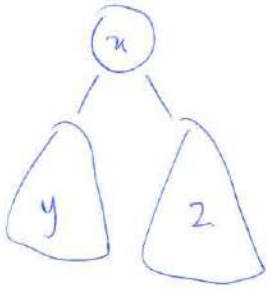
Lecture 16: Binary Search Trees & 2-SAT.

Recall binary search trees: key in left subtree < x.key < key in right subtree



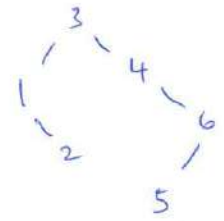
~~[we saw red]~~ basic DS, supporting insert, search, delete, all of which depend on height of the tree.

we saw R-B trees which maintained that any tree with n nodes has height $O(\log n)$.

Will see a simple implementation of a BST that after n insertions, has height $O(\log n)$ in expectation, assuming the order of insertions is random.

Since only relative values of the keys matter, assume we are inserting $1, 2, \ldots, n$ in the tree. Each permutation is equally likely. The algorithm for inserting key k is simple: go down the tree searching for k until you find a NULL, insert k at that point.

say n = 6 , 3, 4, 6, 1, 5, 2



what is expected height of such a tree?

Let X_n be a r.v. denoting height of a tree with n elements $1 \ldots n$

$Y_n := 2^{X_n}$, will show $E[Y_n] = \text{poly}(n)$

Let $Z_{n,i} = 1$ if i is first element to be inserted.
$= 0$ o.w.

If i is first element, then $1 \ldots i-1$ appear in left subtree, $i+1 \ldots n$ appear in right.

Given $Z_{n,i} = 1$. what is height of left-subtree? Since all permutations of $1 \ldots i-1$ are equally likely, this is Y_{i-1}

Similarly height of right-subtree is Y_{n-i}.

Hence $Y_n = \sum_{i=1}^{n} Z_{n,i} \cdot 2 \max(Y_{i-1}, Y_{n-i})$

$\left(X_n = \sum_{i=1}^{n} Z_{n,i} \left[1 + \max(X_{i-1}, X_{n-i}) \right] \right)$

Thus $E[Y_n] = 2 \sum_{i=1}^{n} E[Z_{n,i} \max(Y_{i-1}, Y_{n-i})]$

now note that $Z_{n,i}$ is independent from Y_{i-1} Y_{n-i}.

i.e. height of tree containing $1 \ldots i-1$ is independent of whether i came first or not.

$\Rightarrow E[Y_n] = 2 \sum_{i=1}^{n} E[Z_{n,i}] \, E[\max(Y_{i-1}, Y_{n-i})]$

$\leq \frac{2}{n} \sum_{i=1}^{n} E[Y_{i-1}] + E[Y_{n-i}]$

& we know $Y_1 = 1$, $Y_0 = 0$

~~$\mathbb{E}[Y_1] = 1$~~

~~$\mathbb{E}[Y_2] =$~~

$$\mathbb{E}[Y_n] \leq \frac{2}{n}\left[\mathbb{E}[Y_0] + \mathbb{E}[Y_{n-1}] + \mathbb{E}[Y_1] + \mathbb{E}[Y_{n-2}] + \ldots + \mathbb{E}[Y_{n-2}] + \mathbb{E}[Y_1] + \mathbb{E}[Y_{n-1}] + \mathbb{E}[Y_0]\right]$$

$$= \frac{2}{n} \cdot 2 \sum_{i=1}^{n-1} \mathbb{E}[Y_i] = \frac{4}{n} \sum_{i=0}^{n-1} \mathbb{E}[Y_i]$$

Can show: $\mathbb{E}[Y_n] \leq \frac{1}{4}\binom{n+3}{3}$

using: $\sum_{i=0}^{n-1} \binom{i+3}{3} = \binom{n+3}{4}$ ← not proved, easy proof by induction.

Since: $\mathbb{E}[Y_n] \leq \frac{4}{n} \sum_{i=0}^{n-1} \frac{1}{4}\binom{i+3}{3} = \frac{1}{n} \sum_{i=0}^{n-1} \binom{i+3}{3} = \frac{1}{n}\binom{n+3}{4} = \frac{1}{n} \frac{n+3!}{4!\, n-1!}$

$$= \frac{n+3!}{3!\, n!} \cdot \frac{1}{4}$$

$$= \frac{1}{4}\binom{n+3}{3}$$

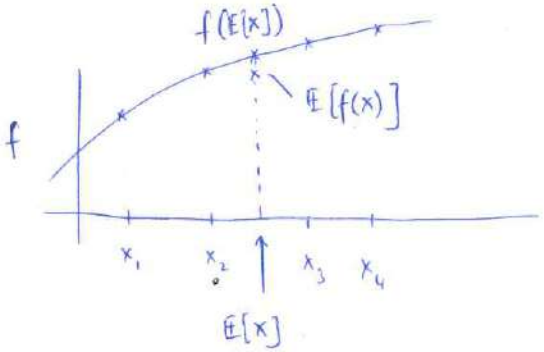
as required

Hence, $\mathbb{E}[Y_n] \leq O(n^3)$

$\Rightarrow \log \mathbb{E}[Y_n] \leq c \log n$

<u>Jensen's inequality</u>: $\mathbb{E}[f(X)] \leq f(\mathbb{E}[X])$ if f is ~~convex~~ concave.

Proof by picture.



but log is concave. Hence $X_n = O(\log n)$ as required.

note: generally a bad idea to assume that input is randomized.

Unless you can't do any better.

(Or at least, distribution should be based on application.)

2-SAT

A Randomized algorithm for 2-SAT.

Given : 2-SAT formula ϕ with m clauses, n variables.

$$\phi = (x_1 \vee \bar{x}_2) \wedge (x_2 \vee x_3) \wedge (\bar{x}_1 \vee \bar{x}_3) \wedge (x_1 \vee x_2)$$

we say a clause is unsatisfied if it evaluates to F.

note that for each clause, there is exactly one way to set the variables in the clause so that the clause is unsatisfied, and the other three ways will satisfy the clause.

eg. for clause 1 :

x_1	x_2	
T	T	→ clause satisfied
T	F	satisfied
F	T	Unsatisfied
F	F	satisfied.

Consider the following algorithm:

```
2SATISFY(φ)
    ∀i, let x_i = T initially
    While (φ not satisfied)
        Let clause C be unsatisfied, w/ variables x_i, x_j
        Pick from {x_i, x_j} w/ equal probability
```

Clearly, if ϕ is unsatisfiable, algo will run forever.

What is expected running time if ϕ is satisfiable?

Fix a satisfying assignment B. We will analyse the # time taken for our current assignment to be the same as B.

To analyze this, let T_i be a r.v. giving # steps taken by the algorithm to reach B, if in current assignment $\underline{i}$ variables are same as in B.

Suppose currently i variables match. In next iteration, we pick an unsatisfied clause and flip a variable. Note that for these variables, initially, at least one variable was set diff. from B (since clause was satisfied in B). Thus w.p. at least $\frac{1}{2}$, # of variables matching decreases by 1.

Let $S_i = \mathbb{E}[T_i]$.

Then for $i \geq 1$, $T_i = 1 + T_{i-1}$ w.p. $\geq \frac{1}{2}$

$= 1 + T_{i+1}$ w.p. $\leq 1/2$

assuming that $S_i \geq S_{i+1}$ (i.e., if more variables are same as in B, we stop sooner)

we get that $\mathbb{E}[T_i] \leq 1 + \frac{1}{2}\mathbb{E}[T_{i+1}] + \frac{1}{2}\mathbb{E}[T_{i-1}]$

& $\mathbb{E}[T_0] = 1 + \mathbb{E}[T_1]$

Thus: $S_0 = 1 + S_1$

$2S_1 = 2 + S_0 + S_2$, or replacing from above, $S_1 = -1 + 2 + S_2$

$2S_2 = 2 + S_1 + S_3 \Rightarrow S_2 = -1 + 2 \cdot 2 + S_3$

$2S_3 = 2 + S_2 + S_4 \Rightarrow S_3 = -1 + 2 \cdot 3 + S_4$

$\vdots$

$2S_{n-2} = -1 + (n-1)2 + S_{n-1}$

$2S_{n-1} = 2 + S_{n-2} + S_n \Rightarrow S_{n-1} = -1 + 2n$, $S_{n-2} = (2n-1) + (2n-3)$

$\downarrow$

0

$\cdots$

$S_0 = n^2$

Thus in expectation, if there is a satisfying assignment, algo will find it in $O(n^2)$ iterations.

To get a bound on the probability of success, we use Markov's inequality:

For a random variable, $\Pr[X \geq k\,\mathbb{E}(X)] \leq \frac{1}{k}$

(you should try & derive proof of this).

Thus if we run the while loop in 2SATISFY $4n^2$ times, and return "ϕ not satisfiable" if we haven't stopped yet, probability of a failure is at most $\frac{1}{4}$.

To amplify this, we can run this algorithm $2\log n$ times. If ~~each~~ any run gives a satisfying assignment, we return this. Else, we say "ϕ not satisfiable".

The prob (failure) = Pr(all $4\log n$ runs fail)

$$= \left(\frac{1}{4}\right)^{2\log n} = \frac{1}{\text{poly}(n)}.$$

Thus, in time $O(n^2 \log n)$, we can succeed w.h.p. Note that our algo only makes "one-sided" error: if ϕ is unsatisfiable, it will always so. If ϕ is satisfiable, it returns a satisfying assignment w.h.p.