CS-GY 6043 A
Lecture 11
Professor Boris Aronov

Name: Willem Mirkovich
Date: 2023-12-12

1 Randomized Algorithms

1.0.1 Required Reading

Basic probability facts to recall notes: edstem discussion post. Goes over basic notation, expectations, etc.

1.1 Randomized Quicksort

1.1.1 Basic Quicksort

Input: array A of n items

Goal: sort in place

Implementation: If n is small, brute force. Otherwise, select a pivot element. Partition array into two sub arrays: one side of pivot with smaller elements, other side with larger elements. Then, recurse on both sub-arrays with same procedure until sorted.

$$\text{Array pre partition: } [-,-,-,-,-,-,-,-]$$
$$\text{Array post partition: } [<x,<x,<x,<x,<x,x,>x,>x]$$

Worst case runtime: $\Theta(n^2)$ in general (even worse if you consistently pick largest/smallest element as pivot)

1.1.2 Randomize selection of pivot

Claim: if pivot is chosen randomly ($1/n$ probablity for choosing a given element), runtime becomes $\Theta(n \log(n))$

Proof 1: Direct

$T(n)$: expected runtime of input of size n
2 cases to consider: bad and good

1. good: pick pivot in between bottom 25% and top 25% of ranked items
2. bad: pick bad pivot in top/bottom 25%, and need to recurse on approx n

Trivially, since we are looking at 50% of elements in either case, you have $\frac{1}{2}$ probability of either case
NOTE: $O(n)$ is sorting of sub-arrays, $T()$ is subarray recursive call

$$T(n) \leq \frac{1}{2}[\text{runtime good}] + \frac{1}{2}[\text{runtime bad}]$$

$$\leq \frac{1}{2}(T(3/4 * n) + T(1/4 * n) + O(n)) + \frac{1}{2}(T(n) + O(n))$$

$$\leq \frac{1}{2}(T(3/4 * n) + T(1/4 * n) + T(n)) + O(n))$$

$$2T(n) \leq 2 * \frac{1}{2}(T(3/4 * n) + T(1/4 * n) + T(n)) + 2O(n))$$

$$2T(n) \leq T(3/4 * n) + T(1/4 * n) + T(n) + O(n))$$

$$T(n) \leq T(3/4 * n) + T(1/4 * n) + O(n))$$

$$\ldots$$

$$T(n) \leq O(n \log(n))$$

CS-GY 6043 A
Lecture 11
Professor Boris Aronov

Name: Willem Mirkovich
Date: 2023-12-12

Proof 2:

Count of the expected comparisons made by quicksort.
Pick 2 elements x_i, x_j with ranks i, j.
P_{ij}: probability of comparing elements x_i with x_j

$$[-, -, x_i, -, -x_j, -, -]$$

Cases:

1. pivot rank $< i \to 0$ comparisons
2. pivot rank $= i \to 1$ comparison to x_j
3. pivot rank between $i, j \to 0$ comparisons, never again
4. pivot rank $= j \to 1$ comparison to x_i
5. pivot rank $> j \to 0$ comparisons

$$P_{ij} = \frac{2}{j - i + 1}$$

Where denominator is the number of elements in range i and j

$$X_{ij} = \text{indicator of if } x_i \text{ compared to } x_j$$

$$\sum_{\forall i,j \in 1 \le i < j \le n} X_{ij} = \text{total \# copmarisons}$$

$$E[\sum X_{ij}] = \sum_{i,j} E[X_{ij}] = \sum_{i,j} P_{ij}$$

$$= \sum_{i<j} \frac{2}{j - i + 1}$$

$$= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{2}{j - i + 1}$$

$$
\begin{aligned}
&\dots \\
&\leq n * 2[\frac{1}{2} + \frac{1}{3} + .. + \frac{1}{n}] : \text{harmonic series} \\
&\leq n * 2 \ln n \\
&= \boxed{O(n \log(n))}
\end{aligned}
$$

1.2 Randomly Constructed Binary Search Trees (BSTs)

Input: Array $A[1..n]$

Central Idea:

1. Randomly permute A
2. Build standard BST with repeated inserts

Central Questions:

1. What is the expected cost?
   Worst case: $\Theta(n^2)$
2. What is the expected height?
   Worst case: $n - 1$ (single branch)

1.2.1 Expected cost

Intuition/Amazaing Fact: This is same problem as Randomize Qsort

Expected # Comparisons: Same as Randomized Qsort $\boxed{\rightarrow O(n \log(n))}$

1.2.2 Expected height

$H(n)$ = expected height of tree of size n $H(1) = 0$

Root rank k,

2 cases (similar to Qsort):

1. 50% of the time balanced
$$max(H(\frac{3}{4}n), H(\frac{n}{4})) + 1$$
2. 50% of the time unbalanced
$$H(n-1) + 1$$

$$H(n) \leq \frac{1}{2}(max(H(\frac{3}{4}n), H(\frac{n}{4})))\frac{1}{2}(H(n-1)) + 1$$
$$H(n) \leq \boxed{O(\log(n))}$$

1.3 Skip Lists

Central Idea: Linked lists have quick insert/delete, but bad search. This Data Structure compensates for this.

1.3.1 General Design

Level of linked lists, with level 0 has all elements, and each level up has subsection of elements. Set fixed probability p, where $0 < p < \frac{1}{2}$

$$\text{top level} = [-\infty \to 68 \to \infty]$$
$$1 \text{ level} = [-\infty \to 45 \to 68 \to \infty]$$

$$0 \text{ level} = [-\infty \to 1 \to 5 \to 45 \to 68 \to 103 \to \infty]$$

Search: Go from top level down, walk list until you find an element of less rank than current.

Example search for 103, walk top level until hit 68, which is lower. Then hit ∞, and go down level, however start at 68 position. Will only look at ∞ on level 1, then move to bottom. Once here, view 103, and found.

Example search for 5, walk top level until hit 68, which is to high. Same thing for 45 on going down a level, so continue to start at $-\infty$. Hit 5 on final level.

Great visual example

Insert: Do a search for element at bottom level (stop once hit too big on bottom level). Once inserted on bottom level. Once done, recursively add element on level up from 0 with probability p

Example Insert 10. Same path as searching for 5, but stop and insert 10 before 45. Then, if $p = .5$, flip coin to insert on level 1. Heads, so add to level 1. Do coin flip for adding to next level once inserted on 1. Tails, do not add to next level. Done.

Delete: Do a search for element, once found delete at all levels.

1.3.2 Expected cost insertion

Without cap on of levels, bad expected cost (imagine infinitely creating levels with repeated heads flip).

Expected number of levels for a given element $= 1 * 1 + p * 2 + p^2 * 3 + ... + k * p^{k-1}$, where to be on any given level, you must have won each previous level (eg flipped heads k times).

Rewritten by seperating multiples of each level

$$1 + p + p^2 + p^3 + ... = \frac{1}{1-p}$$
$$+(p + p^2 + p^3 + ... = \frac{p}{1-p})$$

$$+(p^2 + p^3 + ... = \frac{p^3}{1-p})$$

Expected cost of insertion is $\boxed{O(\frac{1}{(1-p)^2})}$, which is $O(1)$ on fixed constant p, based on number of levels each element will be on.

1.3.3 Expected size

Sum of all elements with how many times each element shows up on each level. We calculated in cost of insertion that element will show up $O(1)$ (based on p), so size is $\boxed{n * O(1) = O(n)}$

1.3.4 Expected cost delete

Random element: $O(1)$ based on cost of insert (how many levels a random element will be on).

Arbitrary element: worst case $O(k)$ where k is of levels

1.3.5 Expected cost search

Going down levels from top, you will expect to see $n*p^k, n*p^{k-1}...n*p^2, n*p+n$ of elements. This simplifies to $log(n)$ of levels.

We are interested in finding expected cost of level i to $i-1$ (the horizontal number of comparisons before jumping down a level). Each of the elements on the level $i-1$ did not make it to i, each with $(1-p)$ probability. For x many items on $i-1$, probability of them being there and not on i is $(1-p)^x$.

Look back to insertion, and notice the expected number of elements that are NOT on a given levels is

$$= 1 * 1 + (1-p) * 2 + (1-p)^2 * 3 + ... + k * (1-p)^{k-1}$$

$$\boxed{= O(1)}$$

based on p for horizontal walk cost.

Therefore Looking back and putting everything together, cost of search is number of levels, which we found to be $\log(n)$

1.3.6 Final Notes

Overall, we get the following costs:

Search/Insert/Delete: $O(\log(n))$ worst case cost **Delete/Insert**: $O(1)$ expected structure cost per operation

1.4 Max 3SAT

Input: 3SAT formula $((a \vee \neg b \vee c) \wedge ...)$

Goal: Find assignment of a, b, c to maximize of truth clauses (NP-Hard)

Trick: Randomly assign truth values with probability $\frac{1}{2}$

Claim: Expected true clauses $= \frac{7}{8}n$, where n is the number of clauses (think about only case its false is worst outcome, where you flip tails 3 times in a row)

Proof:

Fix a clause with independent variables. We can calculate the probability it is false as

$$(a \vee \neg b \vee c)$$

$$(\frac{1}{2} \vee \frac{1}{2} \vee \frac{1}{2})$$

$$(\frac{1}{2} * \frac{1}{2} * \frac{1}{2}) = \frac{1}{8}$$

So, probability of true is $\frac{7}{8}$ Let $T_j = I$, where I is the indicator of a clause being true. For n clauses, its expected of true clauses is

$$E[\sum_{j=1}^{n} T_j] = \sum_{j=1}^{n} E[T_j] = \frac{7n}{8}$$

Corollary 1: There is ALWAYS an assignment with $\geq \frac{7n}{8}$ true clauses

Corollary 2: OPT $\geq \frac{7n}{8}$ for MAX-3SAT, OPT $< n$

Therefore, 8/7 approximation

1.5 Max Global Cut Problem

Input $G(V, E)$, take global cut of edges $A, B \subset E$, $A \cup B = E$, no intersection. Let weight of cut be w, which is of edges crossing cut (edge with vertices in A and B)

Goal Find A, B such that you maximize $w(A, B)$ (NP-Hard[er])

Easy approximation Randomly assign every $v \in V$ to A or B with probability $\frac{1}{2}$ independently.

Central Question What is the probability that the edge crosses the cut?

Directly work out: $c =$ cross, $o =$ out

$$(A \wedge B) = c$$

$$(A \wedge A) = o$$

$$(B \wedge B) = o$$

$$(B \wedge A) = c$$

$$\frac{2}{4} = \boxed{\frac{1}{2}}$$

$$X_e = I$$

where I is the indicator of an edge crossing the cut

Expectation of edges crossing the cut:

$$E[\sum_e X_e]$$

$$\sum_e E[X_e] = \frac{m}{2}$$

Corollary 1: There is ALWAYS a cut of size $\frac{m}{2}$

Corollary 2: Expected is $\geq \frac{m}{2}$

Corollary 3: OPT is $\geq \frac{m}{2}$

Therefore, random gives you a 2 approximation