Randomized Algorithms

What to Randomize?



The world behaves randomly – randomly generated input solved by traditional algorithm

Average-case Analysis

The algorithm behaves randomly – make random decisions as the algorithm processes the worst-case input



Randomized Algorithms

Why Randomize?

Efficient deterministic algorithms that always yield the correct answer are a special case of —



efficient randomized algorithms that only need to yield the correct answer with *high probability*

randomized algorithms that are always correct, and run efficiently *in expectation*



Symmetry-breaking among processes in a distributed system

Simpler

A Quick Review

Pr[A] := the probability of the even A

 \overline{A} := the *complementary* of the even A (A did not occur)

$$Pr[A] + Pr[\overline{A}] = 1$$

E[X] := the*expectation* (the "average value") of the random variable X

$$E[X] = \sum_{j=0}^{\infty} j \cdot \Pr[X = j]$$

Example The Hiring Problem

- Fire an office assistant from headhunter
- $\ \ \,$ Interview a different applicant per day for N days
- \mathcal{F} Interviewing Cost = C_i << Hiring Cost = C_h
- Analyze interview & hiring cost instead of running time

Assume M people are hired.

Total Cost: $O(NC_i + MC_h)$

Naïve Solution

```
int Hiring ( EventType C[ ], int N )
{ /* candidate 0 is a least-qualified dummy candidate */
  int Best = 0;
  int BestQ = the quality of candidate 0;
  for ( i=1; i<=N; i++ ) {
     Qi = interview(i); /* C_i */
     if ( Qi > BestQ ) {
        BestQ = Qi;
        Best = i;
        hire( i ); /* C<sub>h</sub> */
  return Best;
```

Worst case: The candidates come in increasing quality order $O(NC_h)$

Assume candidates arrive in random order

X = number of hires

$$E[X] = \sum_{j=1}^{N} j \cdot \Pr[X = j]$$

Randomness assumption: any of first i candidates is equally likely to be bestqualified so far

$$X_i = \begin{cases} 1 & \text{if candidate } i \text{ is hired} \\ 0 & \text{if candidate } i \text{ is NOT hired} \end{cases}$$

$$X = \sum_{i=1}^{N} X_{i}$$

$$E[X_{i}] = \Pr[candidate \ i \ is \ hired]$$

$$E[X] = E[\sum_{i=1}^{N} X_{i}] = \sum_{i=1}^{N} E[X_{i}] = \sum_{i=1}^{N} 1/i = \ln N + O(1)$$

$$\rightarrow O(C_h \ln N + NC_i)$$

Radomized Algorithm

```
int RandomizedHiring (EventType C[], int N)
{ /* candidate 0 is a least-qualified dummy candidate */
  int Best = 0;
  int BestQ = the quality of candidate 0;
  randomly permute the list of candidates;
                                                     takes time
  for ( i=1; i<=N; i++ ) {
    Qi = interview(i); /* C_i */
     if ( Qi > BestQ ) {
                                      no longer need to
       BestQ = Qi;
                                 assume that candidates
       Best = i;
       hire( i ); /* C<sub>h</sub> */
                                 are presented in random
                                 order
```

Radomized Permutation Algorithm



Assign each element A[i] a random priority P[i], and sort

```
void PermuteBySorting ( ElemType A[ ], int N )
{
   for ( i=1; i<=N; i++ )
        A[i].P = 1 + rand()%(N³);
        /* makes it more likely that all priorities are unique */
        Sort A, using P as the sort keys;
}</pre>
```

Claim: PermuteBySorting produces a uniform random permutation of the input, assuming all priorities are distinct.

Online Hiring Algorithm – hire only once

```
int OnlineHiring ( EventType C[ ], int N, int k )
  int Best = N;
  int BestQ = -\infty;
  for ( i=1; i<=k; i++ ) {
    Qi = interview(i);
     if ( Qi > BestQ ) BestQ = Qi;
  for ( i=k+1; i<=N; i++ ) {
                               What is the probability
    Qi = interview( i );
                                  we hire the best qualified
     if ( Qi > BestQ ) {
                                  candidate for a given k?
       Best = i;
       break;
                               What is the best value of
                                  k to maximize above
  return Best;
                                  probability?
```

Online Hiring Algorithm

 $S_i :=$ the *i*th applicant is the best

What needs to happen for S_i to be TRUE?

[independent]

 $\{A:= \text{ the best one is at position } i\}$

 \cap { B:= no one at positions $k+1 \sim i-1$ are hired }

$$\Pr[S_i] = \Pr[A \cap B] = \Pr[A] \cdot \Pr[B] = \frac{k}{N(i-1)}$$
$$= 1/N = k/(i-1)$$

$$\Pr[S] = \sum_{i=k+1}^{N} \Pr[S_i] = \sum_{i=k+1}^{N} \frac{k}{N(i-1)} = \frac{k}{N} \sum_{i=k}^{N-1} \frac{1}{i}$$

Discussion 13: Prove $\int_{k}^{N} \frac{1}{x} dx \le \sum_{i=k}^{N-1} \frac{1}{i} \le \int_{k-1}^{N-1} \frac{1}{x} dx$

$$\Pr[S] = \sum_{i=k+1}^{N} \Pr[S_i] = \sum_{i=k+1}^{N} \frac{k}{N(i-1)} = \frac{k}{N} \sum_{i=k}^{N-1} \frac{1}{i}$$

What is the probability we hire the best qualified candidate for a given *k*?

$$\frac{k}{N}\ln\left(\frac{N}{k}\right) \le \Pr[S] \le \frac{k}{N}\ln\left(\frac{N-1}{k-1}\right)$$

What is the best value of *k* **to maximize the above probability?**

Discussion 19: What is the maximum value of $f(k) = \frac{k}{N} \ln \left(\frac{N}{k} \right)$? And what is the best k?

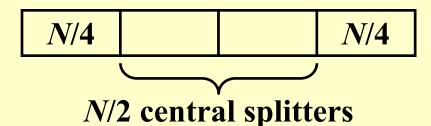
[Example] Quicksort

- **Deterministic Quicksort**
 - $\mathfrak{P}(N^2)$ worst-case running time
 - $\mathfrak{P}(N \log N)$ average case running time, assuming every input permutation is equally likely
- How about choosing the pivot uniformly at random?

Central splitter := the pivot that divides the set so that each side contains at least n/4

Modified Quicksort := always select a central splitter before recursions

Claim: The expected number of iterations needed until we find a central splitter is at most 2.



 $Pr[find\ a\ central\ splitter\] = 1/2$

Type j: the subproblem S is of type j if $N\left(\frac{3}{4}\right)^{j+1} \le |S| \le N\left(\frac{3}{4}\right)^{j}$

Claim: There are at most $\left(\frac{4}{3}\right)^{j+1}$ subproblems of *type j*.

$$E[T_{type\ j}] = O(N\left(\frac{3}{4}\right)^{j}) \times \left(\frac{4}{3}\right)^{j+1} = O(N)$$
Number of different types = $\log_{4/3} N = O(\log N)$



Research Project 6 Skip Lists (26)

Skip list is a data structure that supports both searching and insertion in O(logN) expected time.

This project requires you to introduce the skip lists, and to implement insertion, deletion, and searching in skip lists.

Detailed requirements can be downloaded from https://pintia.cn/

Reference:

Introduction to Algorithms, 3rd Edition: Ch.5, p.114-145; Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest and Clifford Stein. The MIT Press. 2009

Data Structure and Algorithm Analysis in C (2nd Edition): Ch.10, p.399-401; M.A. Weiss 著、陈越改编,人民邮件 出版社, 2005