lesson13 Randomized Algorithms

1.introduction

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

key words: efficient and deterministic

application: Symmetry-breaking among processes in a distributed system(分布式系统中进程之间的对称性破坏)

为什么随机化？因为简单

概率 一些复习：

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

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

2.hiring problem

雇佣问题：

〖Example〗 The Hiring Problem

- **Hire an office assistant from headhunter**
- **Interview a different applicant per day for N days**
- **Interviewing Cost = C_i << Hiring Cost = C_h**
- **Analyze interview & hiring cost instead of running time**

Assume M people are hired.

Total Cost: $O(NC_i + MC_h)$

naive 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 ); /* Ci */
        if ( Qi > BestQ ) {
            BestQ = Qi;
            Best = i;
            hire( i );  /* Ch */
        }
```

```
    }
    return Best;
}
```

worst case: The candidates come in increasing quality order

时间复杂度O(N*Ch)

best case: The candidates come in decreasing quality order

random case:

X: number of hires

$$X_i = \begin{cases} 1, & \text{hired} \\ 0, & \text{not hired} \end{cases}, \quad X = \sum_{i=1}^{N} X_i$$

$$E(X) = \sum_{i=1}^{N} E(X_i)$$

assumption: any of first i candidates is equally likely to be the best-qualified so far.

$$\Rightarrow E(X_i) = 1/i$$

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

$$\Rightarrow O(C_h \cdot \ln N + C_i \cdot N)$$

randomized 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;
//对输入进行随机排列！！！优点：不用担心输入的好坏；缺点：费时间
    randomly permute the list of candidates;

    for ( i=1; i<=N; i++ ) {
        Qi = interview( i ); /* Ci */
        if ( Qi > BestQ ) {
            BestQ = Qi;
            Best = i;
            hire( i );  /* Ch */
        }
    }
```

Radomized Permutation(排列) Algorithm

Target : Permute array A[ ]

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;
}
```

claim: : Permute By Sorting produces a uniform(N!个排列的发生可能性相同) random permutation of the input, assuming all priorities are distinct.

offline algorithm interview之前，得到priority

Online Hiring Algorithm – hire only once

只雇佣一个，前k个，用来确定 BestQ， 后面一旦遇到Qi大于 BestQ ,就直接雇佣，退出;

```
int OnlineHiring ( EventType C[ ], int N, int k )
{
    int Best ;//雇佣他
    int BestQ ;//前k个里最好的
    for ( i=1; i<=k; i++ ) {
        Qi = interview( i );
        if ( Qi > BestQ )   BestQ = Qi;
    }
    for ( i=k+1; i<=N; i++ ) {
        Qi = interview( i );
        if ( Qi > BestQ ) {
            Best = i;
            break;
        }
    }
    return Best;
}
```

两个问题:

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

定义: S_i = the ith applicant is the best

$$P(S_i) = P(A \cap B)$$

A : the best one is at position i } independent

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

$$P(S_i) = P(A)\cdot P(B)$$
$$= (1/N)\cdot\left[1-\frac{1}{i-1}(i-1-(k+1)+1)\right]$$
$$= 1/N \cdot k/(i-1)$$
$$= \frac{k}{N(i-1)}$$

$$\Rightarrow P(S) = \sum P(S_i) = \frac{k}{N}\sum_{i=k+1}^{N}\frac{1}{i-1} = \frac{k}{N}\sum_{i=k}^{N-1}\frac{1}{i}$$

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

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

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

$f(k) = \frac{k}{N}\ln\left(\frac{N}{k}\right)$ 求 k so that $\max(f(k))$

$$f'(k) = \frac{1}{N}\ln\left(\frac{N}{k}\right) + \frac{k}{N}\cdot\frac{k}{N}\cdot\frac{-N}{k^2} = \frac{1}{N}\left(\ln\frac{N}{k} - 1\right)$$

$k = \frac{N}{e}$ 时, $f(k)$ 取max $= \frac{1}{e}$

so the probability of hiring the best is **1/e**

3.quick sort

Deterministic Quicksort

- $\Theta(N^2)$ worst-case running time
- $\Theta(N \log N)$ average case running time, *assuming every input permutation is equally likely*

random: How about choosing the pivot uniformly at random?

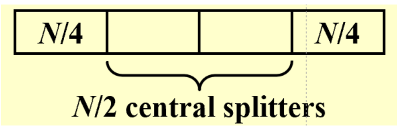
- **Central splitter** : = the pivot that divides the set so that each side contains** **at least n/4**
- **Modified Quicksort** := always select a central splitter **before recursions**

Quicksort主要包含以下3步：

1. 从数组中取出一个元素，叫做主元（pivot）
2. 重排序数组，使得所有小于pivot的元素在它前面，所有大于pivot的元素在它后面，等于pivot的元素放在哪面都行。这样的划分以后，pivot的位置已经排好了。这个过程叫做partition操作
3. 递归地应用步骤2到小于pivot的子数组和大于pivot的子数组

claim1:

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



找到的概率50%嘛

run-time

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

3/4： 如果pivot刚好选在两个1/4,3/4分界点上，那么对应最大子集

下界是由他自己得到的最大子问题，所以肯定比他的子问题大；

claim2: **Claim: There are at most $\left(\frac{4}{3}\right)^{j+1}$ subproblems of *type j*.**

最多有那么多个子问题，每个子问题size最小是他的倒数，那么相乘后肯定要满足不等式左边；

所以时间期望 = 子问题内部遍历调整的时间 * 子问题的数量；

$$E[T_{type\ j}] = O(N\left(\frac{3}{4}\right)^{j}) \times \left(\frac{4}{3}\right)^{j+1} = O(N)$$

再考虑 j 的不同取值，$\log_{4/3} N = O(\log N)$

所以，总时间 $O(N\log N)$