12. Randomized Algorithms

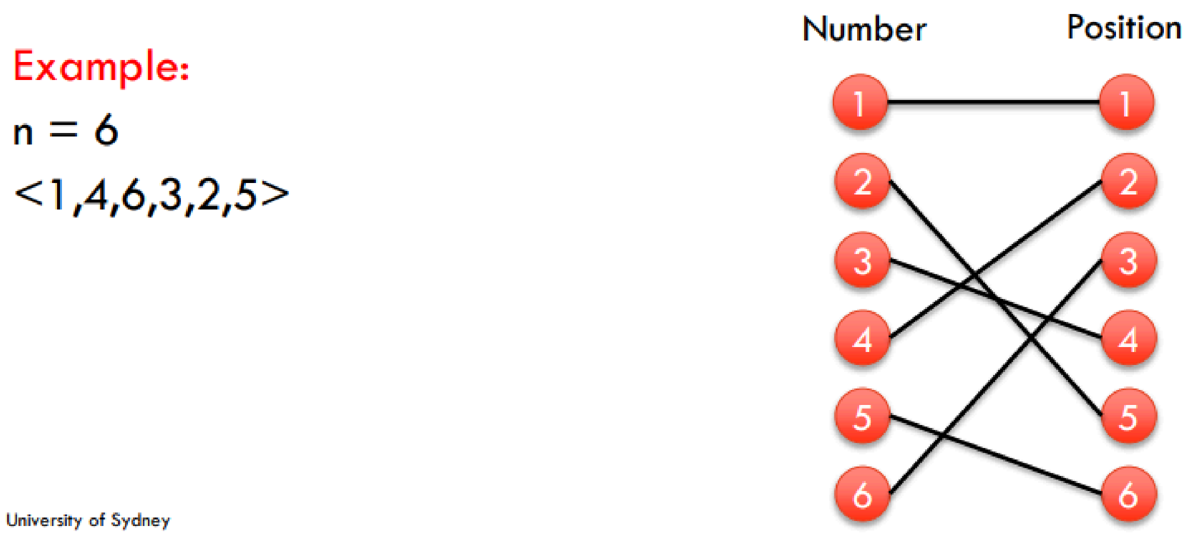
12.1 Definition

Behaviour doesn’t solely on the input. It also depends on random choices or the value of a number of random bits.

- like random seed
- Useful, but skip the case

12.2 Random Permutation

- Input : An integer n
- Output : {1,....,n} but random sequence



- Incorrect attempt:

```
def permute(A):
   n = len(A)
   for i in range(0,n):
      j = random number in {0..n-1}
      switch A[i],A[j]
   return A
```

Incorrect reason :

if A = [1,2,3]

```
A = [1,2,3]
there will be 3*2*1 (3!) = 6 potential result:
[1,2,3]
[1,3,2]
[2,1,3]
[2,3,1]
[3,1,2]
[3,2,1]
```

```
But there will be 3 * 3 * 3 potential switch part!
(every loop it has 3 potential switch and there will be 3 loops overall)
```

- Fisher - Yates

```
def fisher_yate(A):
    n = len(a)
    for i in range(0,n):
        j = random number in (i..n-1)
        switch A[i], A[j]
    return A
```

- Let's calculate!
  A = [1,2,3...n]
  n! potential results

  potential switch part：

  i = 0 —— n

  i = 1 —— n - 1

  i = 2 —— n - 2

  ....

  i = n —— 0

  which is n !

- Example

  [1,2,3,4]

  0 : switch with 2 : [3,2,1,4]

  1: switch with 1 : [3,2,1,4]

  2: switch with 3 : [3,1,2,4]

  3 : switch with 3 : [3,1,2,4]

- Why is that correct?

  We can see that in this case, the possible result is `n !` and the there will be `n!` potential switch !

  - Noticed that, they are equals to each other, and 1 potential switch can cover ever possible results!

  So it is 1 / n! probability for all the different permutation.

  - We can call each sequence of random choices(several potential switch) as an execution that generates a unique permutation.

12.3 Skip lists

Another way implementing Map (not hash)

- simple data structure that's built in a randomized way.
- No need of rebalancing.
- Still has O(logn) worth case time.

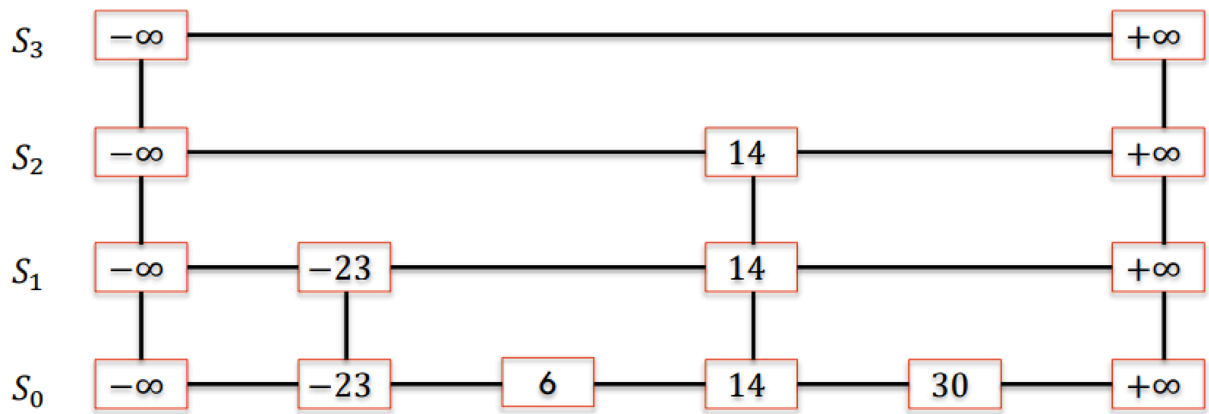
12.3.1 ADT

- get(k)
- put(k,value)
- remove(k)
- size(), isEmpty()
- entrySet() : iterable collection of the entries in M
- keySet() : iterable collection of the keys in M
- values() : .... of values

12.3.2 Structure

联想一下类似一站到底的游戏！ 每个人从level 0 开始抛硬币，赢了就去下一层！

和pivot 有点类似！



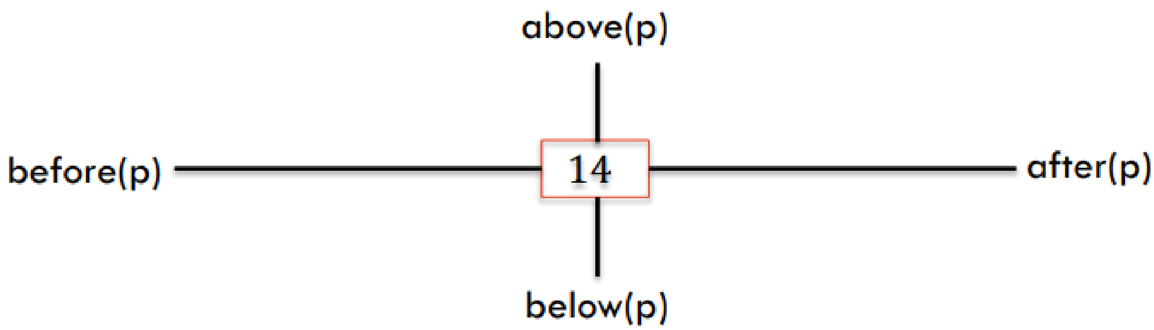
- The start of skip lists :

  only have -无穷 and + 无穷, and several level.

```
Level 2:  -∞  ------⟶  +∞
          |         |
Level 1:  -∞  ------⟶  +∞
          |         |
Level 0:  -∞  ------⟶  +∞
```

the node has pointer

- after(p) : on the right, same level
- before(p) on the left, same level
- above(p) on the upper level
- below(p) on the lower level



- Search

```
def search(p,k):
  while below(p) ≠ null do
        p ← below(p)
    while key(after(p)) ≤ k do
          p ← after(p)
  return p
```

go down → go right until we reach the key that `best smaller/equal` than the given value

continual this process until the bottom level.

in Chinese : 寻找每层最接近target的值(从小的方面来说，或者说 target - p 最小)，然后继续向下找，有点贪心的意思。非底层元素必有below,具体见插入。

```
Level 2:   -∞ ----------⟶ 30 ----------⟶ +∞
           |          |
Level 1:  -∞ ⟶ 10 ---⟶ 30 ----------⟶ +∞
           |     |     |
Level 0:   -∞ → 5 →10 →20→30 →40 →50 → +∞
```

We want to search all the element in given key.

- For example, we search 20.

- Starting with level 2.
- go level 1 , go right until 10, go below.
- go level 1 , go right until we see 20.
- below(10) is null , end loop.

- Insertion

```
def insert(p,k):
 p ← search(p,k)
 q ← insertAfterAbove(p,null,k)
 while coin flip is heads do
       while above(p) = null do
    p ← before(p)
       p ← above(p)
   q ← insertAfterAbove(p,q,k)
```

insertAfterAbove(p,q,k), means that insert or create a new node on right of p and top of q, if q = null, then put it on the level 0.

First use search to find the place to insert, insert on the right of the p at the level 0 and then using 抛硬币 to check which level should it max to (that is random!)

- Removal

Removal

```
def remove(p,k):
  p ← search(p,k)
  if key(p) ≠ k then return null
  repeat
    remove p
        p ← above(p)
  until above(p) = null
```

First find whether have this key ,

if there is, remove all of them and remove all of the above.

- Top layer

The pointer or entry to the skip lists is the top left node.

- How to choose ?
    - One way : max(10,3[log(n)]) —— by experience.
    - The other way : by coins

      Don't need to check it is reach the highest level, if there is not, just add a new level.

      Hardly can it be more than O(logn)

- Analysis
    - Expected height : O(logn)
        - Due to coins, the probability of every node appear in heigh `i` is 1/2^i
        - There are several height, so in level i , there should be at most n / 2^ i percentage that it has one node.

    - Assume that there are level bigger than logn, we can use clogn level (because the level is start from 0, so the clogn is actually clogn + 1 !).
    - The probability of having at least one node of the level is

$$\frac{n}{2^{c \log n}} = \frac{n}{n^c} = \frac{1}{n^{c-1}}$$

    - which is nearly 0.

- Search
    - Two part : horizontal (right) and vertical (down).
    - vertical : logn.
    - horizontal：Because the 1/2 of coins, every level should be / 2 than the down level.
    - So, when we go right on the top level , we actually skip 指数级 node on the 0 level！

      假设有5层，而第五层正好是数组中间值，如果直接照着中间值往下找，就相当于直接忽略了前半部分！而这只需要O(1) 的时间就能做到！
    - We expect O(1) on every level.
    - Total is O(logn) time.

- Space Analysis

  O(n)

$$\sum_{i=0}^{h} \frac{n}{2^i} = n \sum_{i=0}^{h} \frac{1}{2^i} < 2n$$