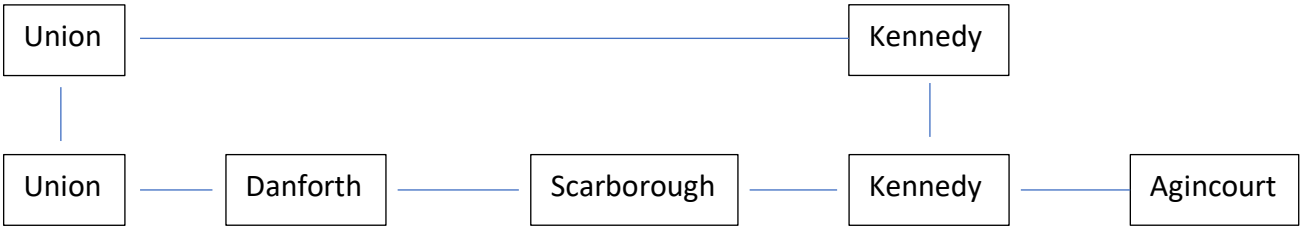
Part 1

Problem:

Consider GoTrain Stouffville Line.

There are Express line that stops at Union, Kennedy, Unionville and Stouffville.

There are also regular line that stops at each station.

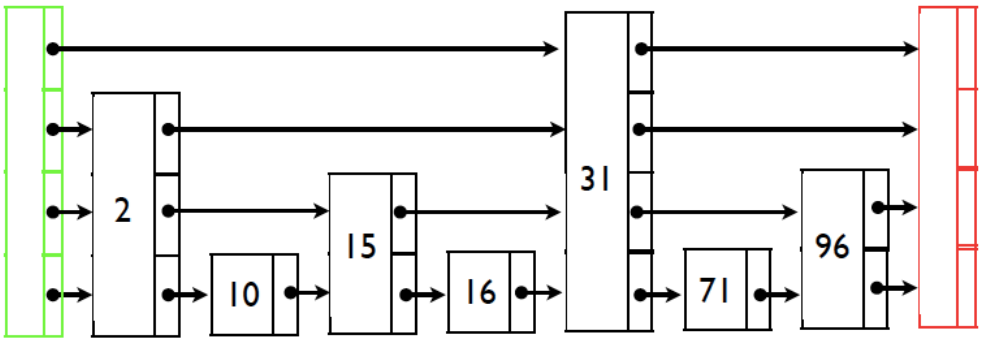


If I want to go to Agincourt, the faster one is to take the express line to Kennedy and then take the regular line to Agincourt. What if there are super express line (stops at fewer stations) ? Or semi-express line (stop at more stations than express one but less than regular) ?

If we take each station as a node, then we can view each line as a sorted linked list, where each line are connected at the common stations (where you can take transfer).

ADT: A sorted linked list where each node connected in various way.

There are n + 2 nodes, the additional ones are dummy nodes for head and tail. Each node contain between 1 and O(log n) pointers. *Why O(log n) ? Explains at the next page!*



So we can also consider it as ADT that links together O(log n) linked list. We can denote the linked list that connect every node as L_0 , the next level as L_1 … etc.

Each node store the following information:

- key : the value it holds
- forward : the list of pointers to the next node at different level, initialized as null
- level : the highest level where this node get promoted
- prev : a pointer to the previous node

William Pugh . July, 1989 . A Skip List Cookbook , University of Maryland
https://www.cs.cmu.edu/~ckingsf/bioinfo-lectures/skiplists.pdf

This list store the following information:

- MAXLEVEL : the highest level of linked list it connects
- head : a dummy node that store $-\infty$
- tail : a dummy node that store ∞

♦ How to decide how often the express line stops ?

Assume each stop has equal probability to be chosen as destination, then
the best approach is to evenly space the regular line, so if we have to take a transfer, we only need to take one portion of regular line, which is $\frac{|regular|}{|express|}$.

- Denote the total number of stations as n, express line as L_1 , regular line as L_0 .

  In worst case, the search cost is $|L_1| + \frac{|L_0|}{|L_1|}$.
  This cost will be minimized when $|L_1|^2 = |L_0| = n$; $|L_1| \Longrightarrow \sqrt{n}$.

  $|L_1| + \frac{|L_0|}{|L_1|} = \sqrt{n} + \frac{n}{\sqrt{n}} = \sqrt{n} + \sqrt{n} = 2\sqrt{n}$

- If we have a super express line, denote as L_2 .
  Then the worst cost of search is $|L_2| + \frac{|L_1|}{|L_2|} + \frac{|L_0|}{|L_1|}$.
  The cost will be minimized when $|L_2|^3 = |L_1|^2 = |L_0| = n$
  $|L_2| + \frac{|L_1|}{|L_2|} + \frac{|L_0|}{|L_1|} = \sqrt[3]{n} + \frac{\sqrt[3]{n^2}}{\sqrt[3]{n}} + \frac{n}{\sqrt[3]{n^2}} = \sqrt[3]{n} + \sqrt[3]{n} + \sqrt[3]{n} = 3\sqrt[3]{n}$

♦ How many line should we have ?

2 sorted linked list : $2\sqrt{n}$
3 sorted linked list : $3\sqrt[3]{n}$
k sorted linked list : $k\sqrt[k]{n}$
$\log n$ sorted linked list : $\log n \sqrt[\log n]{n} = 2 \log n$
$\Rightarrow$ If we have $\log n$ sorted linked list, we can search in $O(2\log n) = O(\log n)$.

Operations:

- Search(x)
  Walk right in top linked list (that stops at fewest stations) until going further will pass our destination, walk down to the second linked list (take transfer), and so on so forth until we reach our destination.

- Insert(x)
  Search(x) to see where x fits in the L_0 , insert it. And flip a fair coin to decide if we want to promote x to the next level (L_1 , L_2 ,…..).

- Delete(x)
  Remove x from all linked list the contains it.

William Pugh . July, 1989 . A Skip List Cookbook , University of Maryland
https://www.cs.cmu.edu/~ckingsf/bioinfo-lectures/skiplists.pdf

Part 2

Operations

- Search : return the node holds the value of key or its predecessor
  1. If x = key : found!
  2. If x < next key, take transfer. (Go down a level)
  3. If k ≥ next key, go right.

```
Node search(List,x):

      curr = List.head
      # start from the top level of skip list, update current forward
      for i in range(self.level, -1, -1):
            while curr.forward[i] & curr.forward[i].key < x
                  curr = curr.forward[i]

      return curr.forward[0]
```

- Insert(x)

1. search(x) to see where to insert x into L_0
2. How to decide if x should be ***promote*** x to L_1 , L_2, and where does it stop ?

$\Rightarrow$ Flip a fair coin to decide ! So ***P(***promote x to next level***)*** = *p* = $\frac{1}{2}$

On average:

$\frac{1}{2}$ of the nodes get promoted 0 levels

$\frac{1}{4}$ of the nodes get promoted 1 levels

$\frac{1}{8}$ of the nodes get promoted 2 levels

$\frac{1}{2^k}$ of the nodes get promoted 2^{k-1} levels

```
void insert(L,x):
      Find the place to insert x at level 0 by search(x).
      Insert x in level 0
      let i = 1
      while FLIP() == "heads" && i< L.MAXLEVEL:
            insert node into level i
            i++
```

- Delete(x)

Remove x from all lists containing it

```
void delete(L,x):
      Find x by search(x)
      For i in (0, x.level):
            x.prev.forward[i] = x.forward[i].forward[i]
            x.forward[i].prev = x.prev
```

William Pugh . July, 1989 . A Skip List Cookbook , University of Maryland
https://www.cs.cmu.edu/~ckingsf/bioinfo-lectures/skiplists.pdf