DSNP report

1. **資料結構的實做**
   1. 三種資料結構的比較
      1. Array : 動態陣列，有capacity(容量)、size(資料量)，當capacity不夠用時就必須換一個container，做法就是新創一個array並把舊的array的data copy過去，並且新array的capacity是old array的兩倍。
      2. Dlist : 每個node都有兩個pointer，一個指到前面一個指到的node。至於在最後一個node後面還會有一個dummy node，他的previous指到最後一個node，然後next指到第一個node。
      3. Bst : 用tree的形式，像樹狀排列，有一個root是最頂端(沒有parent)，另外左邊的node比右邊的node小。
   2. 寫法
      1. Array : push\_back的部分就是在capacity == size的時候，就要換容器，因此就new一塊記憶體給新的array，並且capacity是兩倍；至於delete則是把最後一個拿到刪掉的位置，因此可以提升速度。Sort寫好了這裡就不討論。
      2. Dlist : 是不是empty()用head->\_prev是否 == \_head來判斷；至於push\_back()較複雜，需考慮empty的情況，若不是empty則new一個node並把他插進去，用pointer將其指到對的node就行。

(總共有4個pointer需要assign)。而delete則是把前後node的pointer互指，再把想delete的那個node delete掉。

* + 1. Bst : 我的寫法是有parent也有tail，而有用這兩個東西是讓我在寫的時候比較容易思考，有parent也不需要keep trace，但是在maintain上就比較複雜，例如每次delete牽扯到tail的時候都需要特別思考，因為tail要去指到最後一個；而parent的問題第一個就是maintain上的麻煩，指的時候就要多assign一個parent，另一個問題則是root的parent是null，所以要注意node是root的情況不能再去使用他的parent(會有error)。也因此在寫successor的時候就變得比較簡單，如果有right child就是right child底下指到的最左邊，而如果沒有就是他向上找第一個往右的(有parent就簡單)。

1. **比較三種的表現**

**1. 首先用DO2的表現來做比較，並做簡短的評論 :**

Array ->

Total time used : 0.04 seconds

Total memory used: 4.312 M Bytes

Array幾乎是咻咻就跑完了，完全不耗時間，速度超快，但記憶體就用了比較多，不過就像老師說的，記憶體很便宜……

Dlist ->

Total time used : 28.14 seconds

Total memory used: 3.32 M Bytes

Dlist在sort卡了好久，畢竟是n平方，所以最後跑了28秒，不過記憶體就耗的比array少

BST->

Total time used : 15.39 seconds

Total memory used: 3.328 M Bytes

Bst的sort就完全不需要花時間，但是random delete就花了非常久，幾乎都是卡在random delete，但速度還是比dlist快一些(sort實在差太多)，記憶體用量就跟dlist相去不遠。

自己的測資

我主要是測試add、delete、sort這三者所需的時間，有的數字不同是因為太大可能會跑不出來。以下為我的測資，最後是總結。

**Array 的測試 :**

Atda –r 1000000

Total time used : 0.2 seconds

Total memory used: 64.29 M Bytes

Adtd –r 1000000

Period time used : 0.06 s

Adta –r 1000000

Adts

Period time used : 0.38 seconds

Adtr 6

Period time used : 0 seconds

Adta –r 1000000

Adtd –f 1000000

Period time used : 0.02 seconds

Adta –r 1000000

Adtd –b 1000000

Period time used : 0 seconds

Q -f

Adta –r 5000000

total time used : 1.18 seconds

Total memory used: 512.3 M Bytes

Adtd –r 5000000

Period time used : 0.51 seconds

Adta –r 5000000

Adts

Period time used : 2.08 seconds

Adtd –b 5000000

Period time used : 0 seconds

Adta –r 5000000

Adts

Adtd –f 5000000

Period time used : 0.04 seconds

**ARRAY :**

從以上可以看出，因為push\_back是constant time，所以需要花的時間不多；至於sort由於是nlogn，因此花的時間雖然會因為數字大而增加一些，但也不會卡著很久(比起dlist的sort快很多)**；**另外由於pop\_back只要改變size就好，因此pop\_back幾乎不需要時間；至於pop\_front就需要花一些時間，不過比起erase少了一點，但因為都是constant time，因此比其他兩種(dlist&bst)快很多。Erase因為還需要traverse的時間，因此需多一點時間；但array的erase相對比其他兩個快很多。

**Dlist的測試 :**

Adta –r 1000000

Total time used : 0.15 seconds

Total memory used: 61.27 M Bytes

Q -f

Adta –r 100000

Adtd –r 100000

Period time used : 57.2 seconds

Q -f

Adta –r 5000000

Total time used : 1 seconds

Total memory used: 305.4 M Bytes

Q –f

Adta –r 10000

Adts

Total time used : 2.15 seconds

Total memory used: 0.8398 M Bytes

Adta –r 20000

Adts

Period time used : 16.94 seconds

Adta –r 1000000

Adtd –f 1000000

Period time used : 0.02 seconds

Adta –r 1000000

Adtd –b 1000000

Period time used : 0.02 seconds

**DLIST :**

從以上可以看出dlist的問題是sort太久，我原本還想測100000他會sort多久但跑不出來，另外他的erase也是超級慢，不過比bst好一點；而pop\_back和pop\_front都滿快，大約跟array的pop\_front差不多。另外他的add好像比array快一點點，但是其實差別不大，因此整體效能還是array比較好，但到了數字比較大的時候，memory的用量還是有點差別，array還是用了較多的memory。

**BST的測試 :**

adta -r 1000000

Total time used : 0.96 seconds

Total memory used: 61.28 M Bytes

Adtd –b 1000000

Period time used : 0.11 seconds

Q -f

adta -r 1000000

adtd -f 1000000

Period time used : 0.12 seconds

Q –f

Adta –r 100000

Adtd –r 100000

Total time used : 149.6 seconds

Total memory used: 6.355 M Bytes

Adta –r 1000000

Adtd –b 1000000

Period time used : 0.11 seconds

Adta –r 1000000

Adtd –f 1000000

Period time used : 0.13 seconds

Adta –r 1000000

Adts

Period time used : 0 seconds

Adta –r 5000000

Total time used : 6.87 seconds

Total memory used: 305.4 M Bytes

Adtr 6

Period time used : 0.81 seconds

**BST :**

Bst主要的問題是erase需要花比較多的時間，因此通常都是在random delete的地方卡住；而pop\_back或pop\_front效能就只比另兩個稍慢一些，但還是比較慢，可能也因為有一些pointer需要maintain(例如我的bst是有用\_tail的)。而ADD方面bst也比另兩個還慢，尤其數字大的時候更明顯，主要也是因為他在add的時候就已經sort好了；但使用的記憶體上就跟dlist差不多。當然就如前述所說的sort在add時就已經做好，因此sort的時間是0。而reset方面用的時間也比前兩者還久一些些，主要也是因為他是一個一個做pop\_back()(也是跟pop\_back()的快慢有關)。

**總結** : 比較三者的表現，array還是比另兩者優許多，或許用了較多的記憶體，但效能真的好上許多(尤其數字大的情況)；而至於dlist是最慢的，根本原因是因為他的sort是n平方，而他的erase也需要花一些時間，雖然比bst稍快，但也很久；而bst花最多時間的就是erase，但優點就是資料永遠都是sort好的。