**Data Structure and Algorithm notes**

1. **Supplementary knowledge ( kiến thức phụ trợ hoy)**

* Pointer ( trong C/C++ ), reference trong Java
* <http://alumni.cs.ucr.edu/~pdiloren/C++_Pointers/> (tựa vậy chứ đọc hay)
* Iterator là 1 biến thể của pointer
* Big O, theta notation ( tính thời gian chạy thuật toán )
* <https://www.khanacademy.org/computing/computer-science/algorithms/asymptotic-notation/a/big-o-notation>
* Ví dụ for(int i=0;i<n;i++){ for(int j =0;j<n;j++)} => O(n^2)
* for(int i=0;i<n;i++) =>O(n);
* 2 lần for ko nest ( đan nhau ) => O(n)
* Chặt nhị phân => log2(n)
* Để hiểu rõ them về big O notation thì nó cũng như là chặn trên của tính toán độ phức tạp thuật toán. Giả dụ thuật toán ta có tổng các hành động là 2n^3+4n^2+10n => nó là O(n^3) do chắc chắn sẽ có 1 số c đủ lớn làm cho cn^3 > 2n^3+4n^2+10n.
* 1s thì máy có thể chạy tầm 10^8 hoạt động
* C++ knowledge ( cái này bác Loan biết rồi )

1. **STL trong C++**

* Phần thứ nhất là container. Nói nôm na cho dễ hiểu. Container cứ như là các array vậy. Nhưng chúng nó thật ra là các object có khả năng chứa các object khác ( cái này liên quan OOP ). Ở 1 cấp độ thấp hơn object như là 1 cái struct vậy. struct có khả năng chứa biến và hàm con trong struct.
* Các container thường dùng string vector map set pair
* <https://www.topcoder.com/community/data-science/data-science-tutorials/power-up-c-with-the-standard-template-library-part-1/>
* <https://www.topcoder.com/community/data-science/data-science-tutorials/power-up-c-with-the-standard-template-library-part-2/>
* Phần thứ 2 là các thuật toán có trong thư viện algorithm
* Sort : <http://www.studytonight.com/cpp/stl/stl-sorting-algorithms>
* Lower\_bound,upper\_bound : <http://www.studytonight.com/cpp/stl/stl-searching-lower-upper-bound>
* Các thuật toán chỉ trả kết quả ko làm thay đổi dữ liệu : <http://www.studytonight.com/cpp/stl/stl-non-modifying-algorithms>

<http://www.studytonight.com/cpp/stl/stl-modifying-algorithms>

Đọc chơi thôi chứ chủ yếu xài swap , min , max thôi chứ ko đụng mấy cái này :v

1. **Linked List**

* Cấu trúc dữ liệu bắt đầu tất cả ( cần phải biết về pointer/ reference). Note bác Loan chỉ cần hiểu thôi còn code thì có trong thư viện C++.
* <https://www.cs.cmu.edu/~adamchik/15-121/lectures/Linked%20Lists/linked%20lists.html>

Nhìn hình cho hiểu là được.

* <http://codeforces.com/problemset/problem/264/A> Bài này làm 2 cách insert của vector và insert của list để hiểu sự khác biệt của 2 nhaaaaaa. Xài vector và list có sẵn trong C++

1. **Stack and Queue**

* Stack: Last in first out
* Queue : First in first out
* Có sẵn thư viện trong C++
* <https://www.cs.cmu.edu/~adamchik/15-121/lectures/Stacks%20and%20Queues/Stacks%20and%20Queues.html>
* Bài tập về stack: <https://www.hackerearth.com/practice/data-structures/arrays/1-d/practice-problems/algorithm/nice-arches-1/>, <https://www.hackerearth.com/practice/data-structures/arrays/1-d/practice-problems/algorithm/the-amazing-race-1/> (Có thể xem editorial)
* Bài tập về queue khá ít

1. **Priority Queue**

* Priority Queue là 1 cái queue nó có độ ưu tiên. Ví dụ min priority queue thì nó sẽ ưu tiên thằng nhỏ nhất đứng trước. Max thì thằng max
* Priority Queue có thể đc áp dụng = array ( chậm) hoặc heap (nhanh) (heap là cái gì thì bác Duy chưa học 😐 )
* Priority queue có sẵn trong C++
* Độ phức tạp cho 1 lần insert, delete, pop, push là O(logN), N là size của priority queue
* Bài tập : <https://www.hackerearth.com/practice/data-structures/trees/heapspriority-queues/practice-problems/algorithm/monk-and-multiplication/>, <https://www.hackerearth.com/practice/data-structures/trees/heapspriority-queues/practice-problems/algorithm/monk-and-champions-league/>

1. **Cây ( cơ bản )**

* Lại 1 cấu trúc quan trọng khác.
* Implement = linked list
* Tại sao cây lại quan trọng ? Nó sẽ cho ta cách truy cập 1 phần tử cực nhanh mà ( nhanh tương đương binary search nhưng nó sẽ ko dính các giới hạn về insert sort của các cấu trúc khác.
* Về cơ bản thì set và map thì là binary search tree => Hầu hết các hoạt động sẽ theo O(logN), N => tất cả các node
* <https://www.hackerearth.com/practice/data-structures/trees/binary-and-nary-trees/tutorial/>
* <https://www.hackerearth.com/practice/data-structures/trees/binary-search-tree/tutorial/>
* Bài tập : <http://codeforces.com/problemset/problem/792/D> <http://codeforces.com/problemset/problem/526/B>

1. **Binary Indexed Tree ( Bác Duy chưa học update sau )**
2. **Segment Tree ( Bác Duy chưa học update sau )**
3. **Heap ( Bác Duy chưa học update sau )**
4. **Sort**
5. **Buble Sort**

* Giờ là 10h50 16/04 rồi… Đã gần 10 tiếng từ lúc cái mail đó đc gửi. 20p từ lúc tui nhận đc cái mail đó… Mà thôi tui sẽ vẫn viết cái note này. Như là 1 lời hứa với con mèo ko biết có còn / sẽ là của tui ko…
* Bubble sort. Đúng như tên gọi, nó sẽ sort theo kiểu phần tử thấp nhất, lớn nhất đc trồi lên. Thuật toán này có độ phức tạp là O(n^2).
* Code của thuật toán này là

for(int i=0;i<n-1;i++){

for(int j=i+1;j<n;j++){

if(a[i]>a[j]) swap(a[i],a[j]) }}

* Thuật toán này rất chậm nhưng trong cuộc sống bình thường vẫn đc áp dụng.
* <https://www.hackerearth.com/practice/algorithms/sorting/bubble-sort/practice-problems/algorithm/benny-and-segments-marcheasy/>

1. **Selection Sort**

* Thuật toán này nhìn chung cũng dở như thuật toán ở trên vậy 😐
* Thời gian vẫn là O(n^2)
* Nó khá dễ hiểu. Chỉ là ở vị trí i thì tìm phần tử thứ i nhỏ nhất rồi swap thôi.

1. **Insertion Sort**

* Thuật toán này mạnh hơn 2 thuật toán trên.
* Vẫn là thời gian O(n^2) nhưng thời gian thực tế nhanh hơn rất nhiều.
* <https://drive.google.com/file/d/0B6HjbPlQCGF1dHowRFVvX0plV0k/view?usp=sharing>
* Link trên để xem them về 2 thuật toán này.
* Bài tập chắc hem có 😐 Tại bài tập nó chủ yếu là sort dãy. Mà sort dãy thì mình xài hàm C++ quách cho nhanh. Chỉ hiểu cái idea là đc rồi.

1. **Merge Sort**

* Đây là thuật toán có idea tương tự như chặt nhị phân vậy. Cứ chia từng phần nhỏ ra rồi sort. Ý tưởng chính của thuật toán này chia để trị. Chia nhỏ từng phần ra để trị.
* Ví dụ muốn ghép 2 array đã sort sẵn rồi thì ta phải làm sao ? Chúng ta phải làm trong thời gian như thế nào O(n^2) hay O(n). Câu trả lời là O(n).
* Vậy chúng ta làm sao đạt được giải thuật ở thời gian O(n) bác Loan tự hỏi ????. Bằng cách này đây đầu tiên đặt 2 biến index1 và index2 = 0 để chỉ index ở 2 array. Sử dụng vòng while(index1<arr1.size()||index2<arr2.size()). Nếu arr1[index1] <arr2[index2] thì gán arr1[index1] vào array trả lời. Sau đó tăng index1 lên. Ngược lại tăng index2. Cứ vậy. Lưu ý nếu index1==arr1.size() hoặc index2==arr2.size() thì cứ nhét cho đến khi nào cái index còn lại đụng cái size.
* <https://drive.google.com/file/d/0B6HjbPlQCGF1bWI2VDZ0N2xhSE0/view?usp=sharing> **Code**
* <https://drive.google.com/file/d/0B6HjbPlQCGF1VHpZajZsVWFKd1E/view?usp=sharing> **Slide**

1. **Counting Sort**

* Sort đếm. Thuật toán này là thuật toán sort nhanh nhất. Thời gian chỉ tốn O(n) hoyyyyyyyyyyyyyy
* Tuy nhiên nó chỉ dùng được khi mà mọi phần tử bé hơn 10^5 hoặc 10^6. Còn lại thì không dùng đc
* Vậy tại sao nó lại ko dùng đc bác Loan bánh bao của tui hỏi ??? Tại vì chúng ta sẽ đếm số lần xuất hiện của 1 phần tử. Chúng ta sẽ tạo 1 cái array counting[N]={0} với N là max(a[i]) với 0<i<n (n số phần tử). Sau đó chúng ta sẽ chạy từ 1->N. Chỗ nào counting[i] != 0 thì chúng ta in ra số i tương ứng với counting[i];
* Link code: <http://www.sourcetricks.com/2013/03/counting-sort.html#.WPoMszG0nIU>
* Bài tập: <http://codeforces.com/problemset/problem/768/C>
* Bài tập này sử dụng counting sort là thuật toán nền cho nó. Vì sao bác Loan hỏi. Vì a[i]<=1023 và x<=1000. 2 số <=1023 xor sẽ ra 1 số bé hơn 1023. Vậy chúng ta sẽ áp dụng counting sort thế nào????? Hint nhaaaaaaaaaaaaa. Bài tập này có thể chạy trong thời gian O(1024\*k). Suy nghĩ suy nghĩ nàooooooooooooooooo
* XOR: Bit giống nhau => 0, bit khác nhau =>1

1. **Quick Sort ( bữa t5 cúp chưa họcccccccccc để thầy up slide sẽ update lại )**
2. **Two pointer algorithm**

* Thế nào là thuật toán 2 con trỏ Bác Loan bánh bao hỏi ? Hai con trỏ có như 2 con bò tụi mình hemmm ? Đây là 1 thuật toán khá hay có thể giảm thời gian từ O(n^2) xuống thành O(n).
* Giả dụ có 1 array. Đề cho ra câu hỏi là tìm xem có đoạn liên tục nào có tổng = sum ko. Vậy chúng ta sẽ làm = cách nào ? Cách làm đơn giản đầu tiên là O(n^3) gồm 3 vòng for chồng nhau. {i=0->n{j=i->n{k=i->j } } } . Trong vòng loop k sẽ tính tổng của của đoạn i và j. Chúng ta có thể cải thiện thuật toán lên thành O(n^2) với prefix sum. Tuy nhiên cách nhanh nhất là 2 pointer algorithm. Nó như là 1 cách làm của 1 người thông thường vậy. Đầu tiên tạo hai biến idx1 và idx2. idx1 chỉ bắt đầu và idx2 chỉ kết thúc. Đầu tiên idx1 = idx2 = 0. Sau đó chúng ta cho idx2 chạy lên và tính tổng. Nếu tổng > sum cần tìm thì idx1 sẽ chạy lên và tổng sẽ trừ dần những phần tử idx1 đã bỏ qua. Nếu tổng bé hơn thì idx2 lại chạy lên. Cứ như thế đến khi nào ko chạy đc nữa.
* Chắc cách làm hiệu quả nhất là để bác Loan làm bài tập chứ bác Duy chắc giải thích hơi khó hiểu 😐 Hiểu hemmmmmmmmm ?
* **Bài tập:** <https://www.hackerearth.com/practice/algorithms/searching/linear-search/practice-problems/algorithm/the-normal-type/>**,** <http://codeforces.com/problemset/problem/701/C>

1. **Search**
2. **Linear search**

* Thuật toán này chắc bác Loan cũng quá quen thuộc rồi. Chỉ là mình chạy từ đầu tới cuối tìm 1 phần tử thôi.

1. **Binary search**

* Về căn bản chắc bác Loan quen thuật toán này rồi phải hemmmmm. Vậy hem cần nhắc lại nữa. Ở đây bác Duy sẽ nói về 1 nhánh khác của thuật toán này. Đó là tìm câu trả lời tốt nhất = chặt nhị phân.
* Thế nào là câu trả lời tốt nhất bác Loan bánh bao hỏi. Bác Duy đù trả lời đó là câu trả lời bé nhất hoặc nhỏ nhất. Ví dụ như sau tìm số ngày tối thiểu để học để qua đc tất cả các môn. Hay là tìm số tiền lớn nhất mà mình có thể kiếm đc. Vậy chúng ta sẽ làm dạng bài này bằng cách nào ?
* Chúng ta sẽ làm dạng bài này bằng cách đặt ra range trên và range dưới của câu trả lời có thể. Sau đó từ 2 range đó sẽ chặt nhị phân tìm câu trả lời = 1 hàm check. Hàm check đó kiểm tra xem là câu trả lời đó có đúng hay ko thôi. Chỉ là có đúng chưa hay ko thôi nhaaaaaaaaaaaaaaaa. Còn tìm câu trả lời tốt nhất sẽ do binary search thực hiện.
* Chắc khó hiểu lắm phải hem 😐
* Chờ bác Duy tí nha để cài window xong update tiếppppppp
* Update tiếp nàooooooooooooooooooooooooooo. Giải thích bằng ví dụ là dễ hiểu nhất.
* <http://codeforces.com/contest/782/problem/B>
* Ở trên là 1 bài chặt nhị phân để áp dụng tìm câu trả lời tốt nhất. Nhiệm vụ của chúng ta là phải tìm khoảng thời gian ngắn nhất để mọi người có thể đến đc điểm gặp mặt. Vậy chúng ta cứ chặt nhị phân hoy. <http://codeforces.com/contest/782/submission/25267721>

Đây là code của bác Duy. Theo như bác Loan thấy khúc chặt nhị phân sẽ là khúc này đây

while(abs(Ti-pre)>0.000001){

//cout<<Ti<<" "<<check(Ti)<<endl;

if(check(Ti)) r=Ti;

else l=Ti;

pre = Ti;

Ti=(l+r)/2;

//cout<<l<<" "<<r<<endl;

//cout<<Ti<<endl;

}

Đây là 1 code chặt nhị phân thường mà phải hem. Tại mình tìm thời gian min. Nên nếu thời gian chặt thỏa điều kiện => r=Ti để lát Ti giảm xuống xem có đc ko. Nếu ko thỏa điều kiện l=Ti để lát Ti tang lên.

bool check(double t){

double ri,le=-1e9;

ri=f[0].fi+f[0].se\*t;

le=f[n-1].fi-f[n-1].se\*t;

for(int i=1;i<n-1;i++){

ri=min(ri,f[i].fi+f[i].se\*t);

le=max(le,f[i].fi-f[i].se\*t);

}

if(ri>=le) return 1;

else return 0;

}

Còn đó là hàm check. Nhìn hàm check chắc bác Loan cũng hiểu mà đúng hem. Đó là xét xem có khả năng thằng chạy về phải yếu nhất đụng đc thằng chạy về trái yếu nhất ko. Đó là cách làm của bài này.

* Bài tập: <http://codeforces.com/problemset/problem/807/C>, <http://codeforces.com/problemset/problem/732/D>