Weekly Report July 30,2018-Aug 5,2018

Haoyu Wang
School of Software Engineering
Shanghai Jiao Tong University
gogowhy@sjtu.edu.cn

Abstract

Big chunk of summer holiday time is coming , due to my everyday arrangement and my plan of this summer vacation, I am going to have my weekly report turned out in a diary style,which may help me record what I've done in a week more efficiently.

1 Monday

1.1 current state

1. I finished week 2 of Algorithm(devide and conquer on coursera)

2. I finished Unit 0 of possibility and statistics

1.2 the record of my Algorithm

1. Firstly, I learned about the Algorithm of counting the inversions in an array,using the divide and conquer method.Instead of the brute-force with the running time of $\theta(n^2)$,the usage of devide and conquer method can enable the running time to decrease to $\theta(n \lg n)$,let me show the step briefly:
   $right : ifi, j \geq \frac{n}{2}$
   $split : ifi < \frac{n}{2} < j$

   We combine the merge sort with this recursive method:

   $so : x = sortandcount(left)$
   $y = sortandcount(right)$
   $z = countsplitinversion(x + y)$
   whenever there is an inversion , there should be a number in array y which hasn't been put into the long array before all of the members in array x put into the long array.

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} \mathbf{B} = \begin{bmatrix} b_{11} & b_{12} & \cdots & b_{1n} \\ b_{21} & b_{22} & \cdots & b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n1} & b_{n2} & \cdots & b_{nn} \end{bmatrix}$$

2. Secondly, I learned the Strassen's Alogorithm, which have two matrices with the column of n*n,let me show it in a mathematical way
   we can just make it easier into a simple form, which means matrix A and matrix B

$$\begin{Bmatrix} a & b \\ c & d \end{Bmatrix}$$

$$\begin{Bmatrix} e & f \\ g & h \end{Bmatrix}$$

   What we should actuallt do is changing the 8 recursive calls (AB BG AF BH CE DG CF DH) into 7 ,which can absolutely decrease the time cost:
   1.p1=A(F-H)
   2.p2=(A+B)H
   3.p3=(C+D)E
   5.p4=D(G-E)
   6.(B-D)(G+H)
   7.(A-C)(E+H)
   then it can be replaced like:

$$\begin{Bmatrix} p5_p4 - p2 + p6 & p1 + p2 \\ p3 + p4 & p1 + p5 - p3 - p2 \end{Bmatrix}$$

3. Thirdly, I learned The closest pair problem,which means: we input a long array with (p1,......pn) which has two coordinate px and py
   $(p1, q1) = closestpair(Qx, Qy)$;
   $(p2, q2) = closestpair(Rx, Ry)$;
   $(p3, q3) = closestpair(px, py)$;
   the key is that though we need$n \lg n$ running times to do the first two recursive calls in two array, we just need a 6*n time at most to find out if there is a split distance which is less than the current least distance

4. Forthly, I learned about the master method to count the time complexity,let me show you in just pure math
   $T(n) = aT(\frac{n}{b}) + O(n^d)$
   The a means: number of recursive calls (≤ 1)
   The b means :input size shrinkage factor($>$1)
   The d means:outside the recursive call operation(≤ 0)
   and the result is rather simple:
   $T(n) = O((n^d) * \log n)$ if a=b^d
   $T(n) = O(n^d)$ if a$<$ b^d
   $T(n) = O(n^(logba))$ if a$>$ b^d
   The proof is quite simple so i will cover its detail, but the core thought of this proof is that : the total works $\leq C(n^d)((\frac{a}{b^d})^(logba))$

1.3 the record of my Posibility and Statistics

Unit 0 is quite fundamental that only tells the basic requirement and what we may learn in the following lectures.

1.4 summary

today is really enriched and may I persist in my plan in the following days!

2 Tuesday

2.1 current state

1. I finished week 3 of Algorithm(devide and conquer on coursera)

2. I finished Unit 1 of possibility and statistics

2.2 the record of my Algorithm

1. Firstly, I learned about the quick sort, actually this is what I've been learning for day long, including it's proof and some theorem ,which has an average running time of O(n$\log n$), but it also has very certain advantage over the traditional merge sort–Quick sort needs no more storage area, which contents the programmer a lot.Let me display the algorithm with a kind of pseudo-code at first:
   if n=1,return n
   p=choosepivot(A,n)
   patition A around p
   recursively sort 1st part
   recursively sort 2nd part
   to actually fulfill this expectation, we need to add two variables into this subroutine–i (the pivot mark)and j(the time counter),so the code would be like that:
   Partition(A,l,r) (l means the very left element and r means the very right element)
   -p:A[l];
   -i=l+1;
   -for j=i+1 to r:
   if A[j]¡p(else we do nothing except add 1 to i)
   -swapA[i] and A[j]
   i++
   -swap A[l] and A[i-1]
   That's actually how Quick sort works, but the key is that how we choose a pivot, the method is we choose a pivot in a random way,so actually this is a random algorithm, so how to measure it's running time with totally different excution each time?
   Obviously it takes us $\theta(n^2)$ running time if the pivot is as bad as a smallest element, while it takes us $\theta(n \log n)$ if the pivot ranks from $25\%$to 75%in this array
   Here comes our key claim:
   ∀i,j,PZi,Zj get compared=$\frac{2}{j-i+1}$, Zi means the ith smallest element in the array.
   the proof is quite simple:
   we first l ist from Zi,Z(i+1),.......toZj
   Sooner or later a pivot will occur in this list
   if:pivot is between Zi and Zj, thenthey will never ever be compared, cause each of the element can only be compared 0 time or once because of the disappair of the pivot after a single recursive call, and from i to j elements are sorted, once chosen a pivot from them ,they will be separated permanently
   if:pivot is either Zi or Zj,then they will be compared once and separate from each other since the pivot will be fixed and deleted in the following recursive calls
   E(c)=$\sum_{i=1}^{n-1}\sum_{j=1+1}^{n}\frac{2}{j-i+1}$ (we use a linearity of expectation here)
   E(c)$\leq 2*n*(\frac{1}{2}+\frac{1}{3}+......\frac{1}{n})$
   According to some basic knowlege of calculas, the function of $\frac{1}{2}+\frac{1}{3}+......\frac{1}{n}$ is definitely smaller than $\ln n$, so the running time is around O(n$\log n$).

2.3 the record of my Posibility and Statistics

1. Today I learned the basic concepts of probability, namely sample space ,events, subsets and some probability axioms
   P(A)$\geq$0
   P(Ω)=1
   P(A)+P(A^c)=1

P(A∪ B)=P(A)+P(B)-P(A∩B)
P(A∪B∪)=P(A)+P(A^c∩B)+P(A^c∩B^c∩C)
S^{c^c}=S

2. Then i met De Morgan's Laws, which is quite similar to that in discrete math:
   $(S \cap T)^c$=$S^c \cup T^c$
   That is to say:
   $(\cup Sn)^c$=$\cap Sn^c$ and$(\cap Sn)^c$=$\cup Sn^c$

3. I learned about definite series and grometric series, countable and uncountable sets. $\sum_{i=1}^{\infty}(\sum_{j=1}^{\infty} aij)$=$\sum_{j=1}^{\infty}(\sum_{i=1}^{\infty} aij)$
   P((A∩B^c)∪(A^c∩B))=P(A)+P(B)-2P(A∩B)

4. I learned the Bonferroni's inequality:
   P(A1∩A2)≥P(A1)+P(A2)-1
   which can be popularized as:
   P($A1 \cap A2 \cap$ ......An)≥P(A1)+P(A2)+......+P(An)-(n-1)

2.4 summary

today is really more enriched and may I persist in my plan in the following days!

3 Wednesday

3.1 current state

1. I finished week 3 of Algorithm(devide and conquer on coursera)

2. I finished Unit 2 of possibility and statistics

3.2 the record of my Algorithm

1. I firstly learned about the Selection problem ,which can actually be solved by Merge sort or Quick sort,but can we do better? Definitely yes, we use a recursive call on each recurrence and find the pivot at random ,then we will make it:
   Select(array A,length n, orderstatistic i)
   -if n=1 return A[i];
   -choose pivot at random;
   -partition A around P
   let j = new index of p;
   -if j=i , return p;
   if j>i,return select (1st of array A,j-1,i)
   ifj<i,return select (2nd of array A,n-j,i-j)

2. The worst running time can be about O(n),while the actual average speed ofO(n);
   The proof is also kind of simple, by using the induction method,
   The running time $\leq \sum_{phasej} Xjnc\frac{3}{4}^j \leq 8cn$
   so it's an linear algorithm

3. Then i learned the deterministic selection algorithm ,which also has the time complexity of O(n),the operation is like a magic:
   select(array A,length n, order statistic i)
   -Logically break A into n/5 groups and sort them (around 120 operations times n= O(n))
   -p=select(c,n/5,n/10)

-partition A around p
-if j=i,return p
-if j¡i,return select(1st of A,j-1,i)
-if j¿i,return select(2nd of A,n-j,i-j)
The core thought of the deteministic algorithm is choosing a pivot from the mediam of the mediam.
Surprisingly, the running time is also an O(n) despite of our usage of two recursive calls in a single recurrence.

4. The next step, i started some graph algorithms, I reviewed the knowledge of graph ,namely cuts, crossing edges and so on, and then learned about The minimum cut problem– to figure out the fewest number of crossing edges in an undirected graph, before the algorithm was taught , i first knew that the graphs are stored in the memory mainly in two different ways: Way1:adjacency Matrix, it's good for dense graph,with O(n^2),but is really a waste when we store a sparse graph(almost linear)
   Way 2:adjacency list,it's good for sparse graph like internet or something else,with the space of only O(n+m)
   how to make an adjacency list?
   -We form an array(or list) of vertices
   -We form an array(or list) of edges
   -each edge points to its end points
   -each vertex points to edges incident on it
   the time complexity is obviously O(n+m)

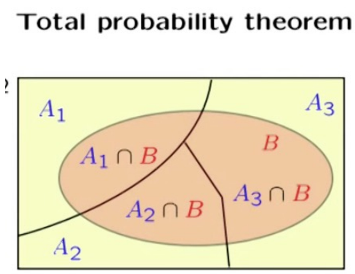
5. after figuring out this issues, I met the Randomized contraction algorithm, which is the first algorithm i met that use a repetitive way to compensate it's low accuracy, that's awesome!
   -While there are more than 2 vertices:
   -pick a remaining edge at random
   -merge u and v into a single vertex
   -remove self-loop
   It's acuuracy is very low like $\frac{1}{n^2}$,but after repeating it $n^2 \ln n$ times, the rate of an error answer can be reduced to $\frac{1}{n}$

6. The eventual problem today is what's the maximum number of minimum cuts in a graph?
   The answer is $C^2 n$

3.3 the record of my Posibility and Statistics

1. First I learned about the total probablity therorem
   DEF: P(B$\|$A)=P(B$\cap$A)/P(A)

2. Then I knew the Bayer's rule: P(Ai$\|$B)=$\frac{P(Ai)P(B\|Ai}{\sum_j P(Aj)P(B\|Aj)}$



3. ThenI learned about the total probablity therorem

$$P(B) = P(B \cap A_1) + P(B \cap A_2) + P(B \cap A_3)$$
$$= P(A_1)P(B|A_1) + \cdots + \cdots$$

4. And another part of today's knowledge is independence
   DEF: P(B$\|$A)=P(B)
   -if A and B are independent ,then A and B^c are independent
   conditional independence:
   $P(A \cap B \| C) = P(A \| C) P(B \| C)$
   event independence: $P(Ai \cap Aj \cap ......Am) = P(Ai)P(Aj)......P(Am)$

5. Finally, i learned somthing of the relationship between independence and reliability

3.4 summary

I started probability formally today , may I arrange my time in a more scientific way in the following days of study!

4 Thursday

4.1 current state

Pitifully, my compulsory curricular has a lab which is due to 3rd Aug, so today I did nothing except from the stupid homework. :(

4.2 summary

Cheer up tomorrow!!!

5 Friday

5.1 current state

Today, I finished week 5 of Algorithm ,graph searching.

5.2 the record of my Algorithm

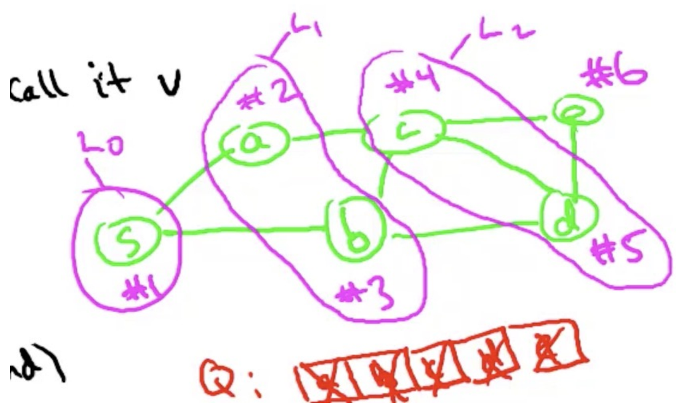
1. Firstly, I started the generic graph search algorithm ,the Goals are literally simple:
   1.find everything findable;
   2.Do not explore anything twice ,because we want to control the time complexity of the algorithm at the stage of O(m+n),which is actually linear thus can run quickly.

There are totally two different searching methods to solve the problem put forward above ,namely BFS(breadth first search)and DFS(depth first search),the core thought of these two searching methods are as follows:
BFS:-explore nodes in ”layers”
- compute the shortest paths
-with a linear running time O(m+n)
DFS:-explore agressively, onky look back when necessary;
-O(m+n) running time also

2. To manage the two searching methods, we need first learn about two different data structures, the queue stucture and the stack structure:
QUEUE structure : a FIFO principle(first in and first out)
STACK struxture: a LIFO principle (last in and first out)

5.2.1 BFS

1. Breadth first search: the code is as follows:
-BFS (graph G, start vertex S)
mark S as explored
-Let Q = queue data structure(FIFO) initialized with S
-While Q $\neq \emptyset$
remove the fisrt node of Q, call it V;
for each edge(V,w)
if w unexplored,
mark w as explored
add W to Q(at the end)



We manage to get two claims from the BFS alforithm
Claim1:At the end of BFS, V explored : $\iff$ G has a path from S to V
Claim2: running time of the BFS is linear O(m+n) ,which is the result of the fact that we chache each vertex once and check each edge at most twice

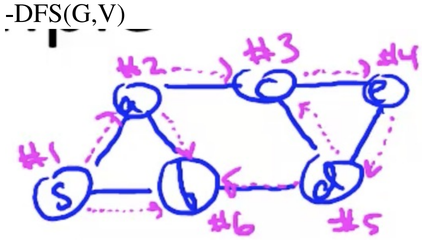
2. Then I met some applications of the BFS, for example ,the calculation of the shorteset path,what we need to do is just adding a few extra codes:
When considering edge(V,W):
If W unexplored set dist(w)=dist(v)+1;
Termination dist(V)=i shows which layer v lies.

3. Another application of BFS is the Undevided connectivity, we say that U,V are equivalence class only when there is at least a U-V path in G
Our goal is to compute all connected components (chack if a net is broken):
-All nodes unexplored(labelled from 1 to n)
-for i =1 to n
-if i not yet explored,

-BFS(G,i)
-count the time of BFS operated

5.2.2 DFS

1. Secondly , I learned about the code of DFS (known as depth first search),the code is as follows:
   -DFS(graph G,start vertex S)
   -mark S as explored
   -for every edge (S¡V)
   -if V unexplored
   -DFS(G,V)

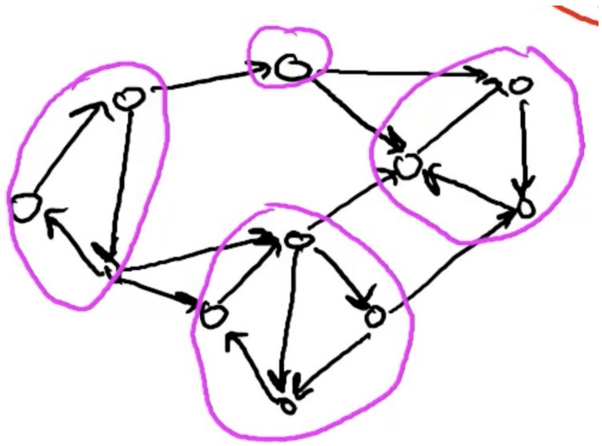
   

   We manage to get two claims from the BFS alforithm
   Claim1:At the end of BFS, V explored : $\iff$ G has a path from S to V
   Claim2: running time of the BFS is linear O(m+n) ,which is the result of the fact that we chache each vertex once and check each edge at most twice (same as the BFS)

2. The first apllication of DFS is Topological sort:
   To manage the Topological sort ,we should first figure out the following two principles:
   1.If G has a directed cycle ,then it;s impossible for us to compute a topological sort
   2.If ther is no directed cycle ,which means there is a sink vertex(no outgoing edges), definitely we can compurte the topological sort in a linear way O(m+n)
   The code is as follows:
   Part A:
   DFS(graph G, start vertex S)
   -mark S explored
   \- for every edge (S,V)
   -if V not yet explored
   -DFS(G,V)
   -set f(s)=current-label
   -current-label–

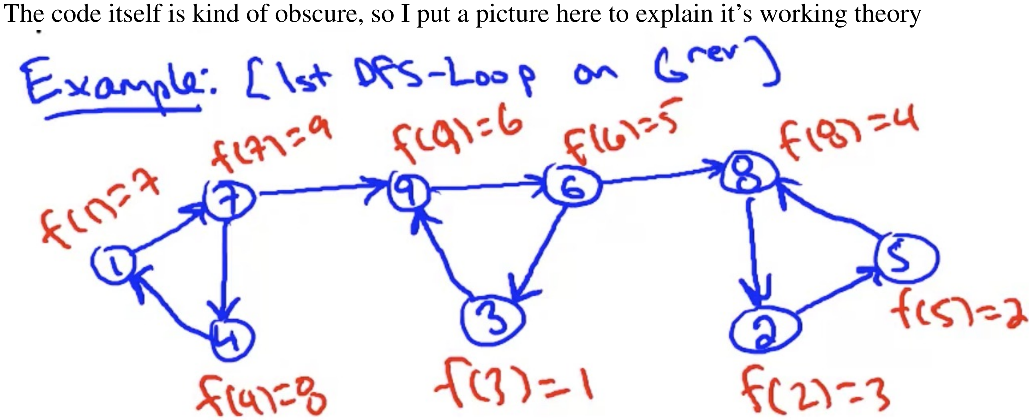
   Part B:
   DFS-LOOP(graph G) -mark all nodes unexplored
   -current label = n
   -for each vertex V $\in$G -if V not yet explored
   -DFS(G,V)
   The running time of this algorithm is also O(m+n)

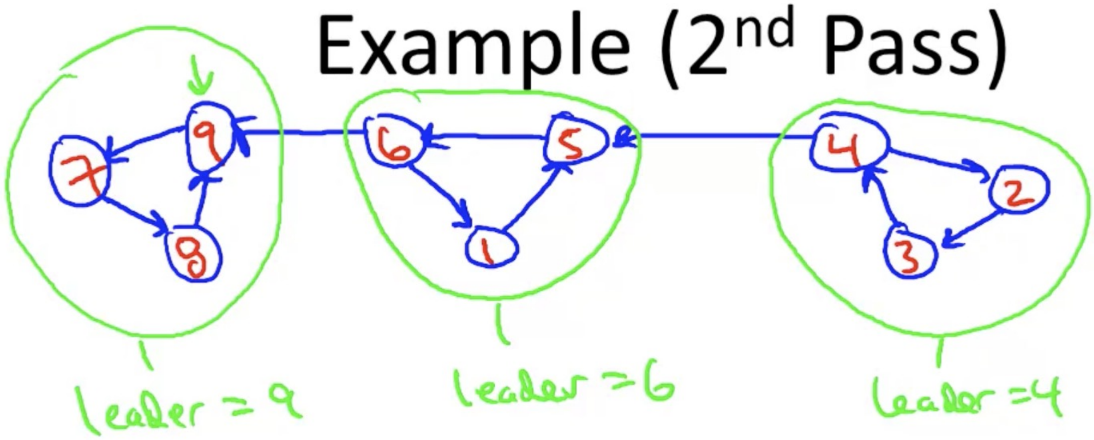
3. another application of DFS is the Strongly connected components (come from any point to any other point);The Strongly connected components can be taken as SCC for short.
   For example, these are four SCCs:



It's like a magic that we can easily figure out the problem by using the DSF only twice, the code and examples are as follows:

Pseudocode PART:
-Algorithm(given direxted graph G)
-Let $G^{r}ev$ = G with all arcs reserved
-run DFS-loop on $G^{r}ev$ (The goal is to compute f(x) = finishing time of each vertex)
-run DFS-loop on G (The goal is to dicover the SCCS one by one, processing nodes in decreasing order of finishing codes with a reverse arc)
DFS-loop PART:
-DFS-loop) graph G) -global variable t=0 (for finishing times in 1st pass)
-gloabal variable s=NULL(for leaders in 2nd pass)
'assum nodes are labelled 1 to n
-for i=n down to 1
if i not yet explored
s=i
DFS PART:
DFS(graph G,i)
-mark i explored
-set leader(i)=node s
for each arc(i,j)∈G if j not explored
DFS(G,j)
t++
setf(i)=t
The code itself is kind of obscure, so I put a picture here to explain it's working theory



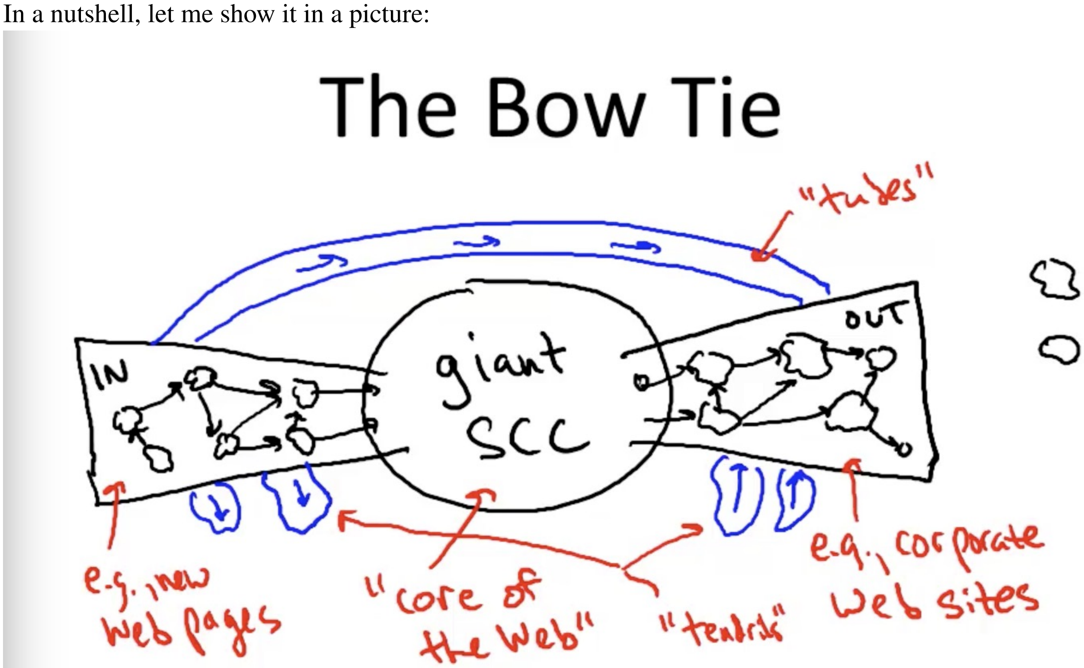


Running Time: 2 * DFS = O(m+n).

Then we can easily find the three SCCs in this situation

5.2.3 web

Then I learned about the relationship between web and graph. it's kind of hard to explain. In a nutshell, let me show it in a picture:



That's the bow-tie thoery, in which it's author explains the relationship between graph and web(internet)

5.3 summary

Today I've got no time to learn start Lec3 of probability and statistics, hope I would finsh both two of then tomorrow!

6 Saturday

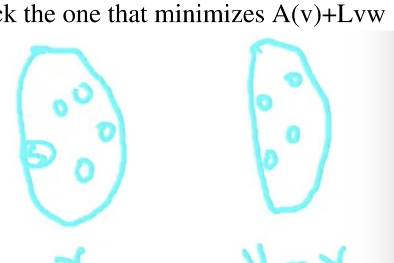
6.1 current state

1. Today, I finished week 6 of Algorithm ,graph searching.

2. I finished Unit3 of probabilities and statistics

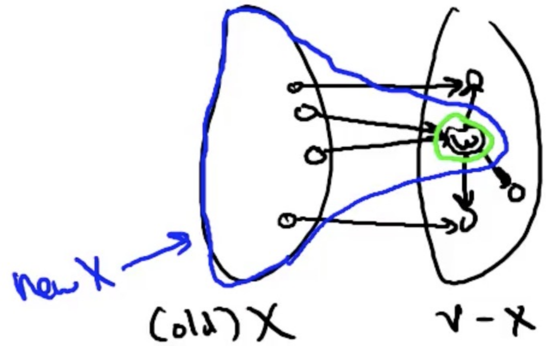
6.2 the record of my Algorithm

1. Today I learned about the shortest way problem ,with its application in the map calculation, named SINGLE SOURCE SHORTEST PATHES, we input a graph and then input a vertex x, we want to calculate the shortest path from each other vertices to the vertex X, There are two principles that we should obey:
   a. (for convenience) each has a path (or we can use DFS of BFS to check) to the vertex
   b.nonnegative length $\forall le \geq 0$
   Question: Why not just change the length into 1 in BFS? Answer:Because from highway to neighborhood ,the D-value are too big
   here is the pseudo code
   INITIALIZE PART:
   -x=(s) (vertices processed so far)
   -A(s)=0 (computed shortest path distances)
   -B(s) empty path (computed shortest paths)
   MAINLOOP PART:
   -while x$\neq$ V (forw x by one node each loop)
   -among all edges (v,w)$\in$ with v$\in$x,w$\notin$x:
   -pick the one that minimizes A(v)+Lvw

   -set A(W^*)=A(v^*) +Lv^*w^*

2. The naive running time is O(mn),because we need to run (n-1) operations per loop ,and try m edges, so we need something to improve the present algorithm to make it faster in case we calculate the distance of a big map, so we add to HEAP datastructure , which is kind of trees and wee can extract from each key and we can bubble up or bubble down
   The feature of HEAP data structure:
   a.extract-min by swapping up the last leaf bubbling down
   b.insert via bubbling up
   c.elements in heap is smaller than it's sons
   Here is the improved pseudo code:
   -for each edge (w,v)
   -if v $\in$ V-X (in heap)
   -delete v from heap
   -recpmpute key(v)=min
   -re-insert v into heap



The running time of this algorithm is now o(m$\log n$),which is much more faster

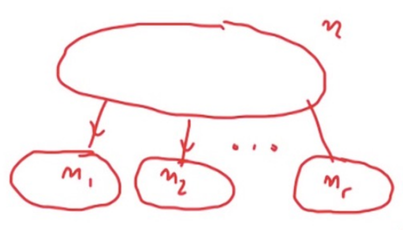
6.3 the record of my Posibility and Statistics

1. Today I learned about the COUNTING, number choices n1,n2,n3......,divided into repitation allowed and forbidden
   We use die roll example to explain the multiplication method ,which is literally fundamental

2. Then I learned about choice : C^kn=$\frac{n!}{k!(n-k)!}$

3. Then I learned about the BINOMICAL PROBABILITIES , Let's take coin toss as an example:
   p(particular)=$p^{numofhead}(1-p)^{numoftail}$
   p(k heads)=$p^k(1-p)^{m-k}C^{khead}$n

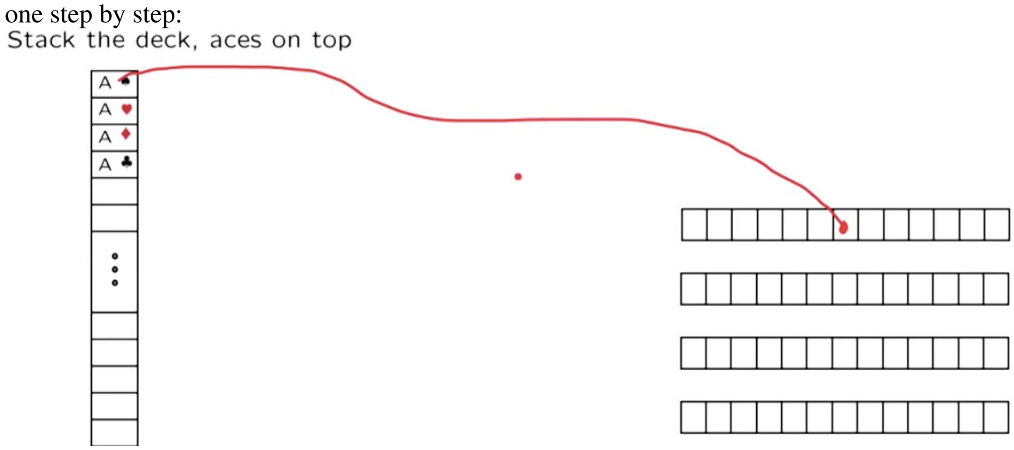
4. Then I learned about partiton:

   The formula is:
   choices=$\frac{n!}{m1!m2!......mr!}$

5. Then I met an application of how to divide the play card for each of the player, ensuring each of them has an ACE:
   There are totallt two ways, one is the traditional way of counting and partition ,I would like to talk about the second here:
   We put the four ACE in a stack and then pull them out ,and calculate each probability one

by one step by step:



6.4 summary

Cheer up next week!

7 Sunday

7.1 current state

Today is for fun.

8 questions

1. Frankly speaking,though I understand some basic data structures ,namely stack or queue,but I have not figured the features of HEAP structure, is there any book to recommand for a total greenhand to learn about the fundamental datastructures?