List of all Topics thats I believe needs revision, if you feel like you are proficient enough in a certain topic and would like to use your time in harder topics, feel free.
Also start arranging your precode if you haven't already, this is gonna be your precode for your entire contest-life. Cheers ^^

Topics:

MORASS problem list dise, ar ki lage? http://codeforces.com/topic/55548

https://docs.google.com/document/d/1_dc3Ifg7Gg1LxhiqMMmE9UbTsXpdRiYh4pKILYG2eA4/edit
// https://discuss.codechef.com/questions/18752/what-are-the-must-known-algorithms-for-online-programming-contests
** https://codeforces.com/blog/entry/23054
geometry https://drive.google.com/drive/folders/0B5OUKDqO5e7RYWFDTWF4NWtla3M zobayer vai (du) er
http://geomalgorithms.com/index.html
https://codeforces.com/blog/entry/48122
https://www.dropbox.com/s/5vsi3amdnbv5qo1/Translated%20Geometry%20Set.pdf?dl=0
Shishir did first 10;
Rosen Discrete Math er NT,Prob,Combi Obosshoi,oboshhoi,oboshhoi pura pora lagbe
BACS camp notebook http://cpps.bacsbd.org/notebook/view-note/notebook
Huge problem for segu, BIT, dp http://praveendhinwacoding.blogspot.com/2013/06/700-problems-to-understand-you-complete.html

- **Some Basics:**
  - Bitwise Operation (Zobayers blog)
- **Number Theory:**
  - Sources: geeks , cf , codechef , e-max **, forthright48 **
  - Prime detection → usually sqrt(n) → see all prime testing methods
  - Optimized prime detection:

```
def isprime(n):
"""Returns True if n is prime."""
 if n == 2:
  return True
 if n == 3:
  return True
 if n % 2 == 0:
  return False
 if n % 3 == 0:
  return False
i = 5 , w = 2
while i * i <= n:
 if n % i == 0:
  return False
 i += w
 w = 6 - w
 return True
```

  It's a variant of the classic O(sqrt(N)) algorithm. It uses the fact that a prime (except 2 and 3) is of form 6k - 1 or 6k + 1 and looks only at divisors of this form. //jumping by 6 , that's all it's doing
  - Prime factorize → sqrt(n)
  - Prime factorize → log(n) : mainly in sieve, in place of keeping 0/1 keep the smallest prime factor, and then run this while (x != 1)

```
{
 ret.push_back(spf[x]);
 x = x / spf[x];
}
```

- Prime sieve O(nloglog(n)):
  - Can half memory by not marking the evens
  - Or even more reduction by keeping 32 bits in an int for mark values of numbers (bitwise sieve)
  - Yarins sieve (didn't learn)
- Segmented sieve (finding primes from a to b) **DIDN'T IMPLEMENT**
- Divisors sieve O($n\log_e(n)$) / O(nln(n)):
  - for(int i=1;i<=n;i++) for(int j=i;j<=n;j+=i) divisors[j].pb(i)
  - No. of divisors of 1 from 1 to n is n/i . n*(1+½+⅓+¼+...1/n)=nlogn .
- SOD, NOD -- loj-article
- Euler Theorem
  - a^phi(m) ≡ 1 (mod m)
- Fermat Theorem
  - a^(p-1) ≡ 1 (mod p) for p is prime
- Euler totient function (sieve phi)
- Loop phi
  - For all primes of n, do n-=n/p
- Modular inverse (a,m)/(a,p)=1 *must coprime*
  - a^(p-2) % p for prime
  - a^phi(m)-1 % m for non-prime
- Big mod : O(logn) , Big Sum
- Powerful tricks with calculation modulo --- https://bit.ly/2npas6h
- Extended gcd
- Modular inverse using egcd
- Linear diophantine -- emaxx
- Linear congruence
- CRT https://nrich.maths.org/5466
- CRT, Lucas, wilsons https://www.spoj.com/problems/DCEPC13D/
- ${}^{n}C_{r}$ for large numbers ? - see this
- gcd/lcm (with and without recursion)
- Huge mod notebook-see tutorial
- Discrete Logarithm/ Shanks Baby step Giant step (loj)
- Extras:
  - Pythagorean triplets
  - Fermat last theorem
  - Perfect number
  - Mersenne prime
  - Fermat decomposition to find deeply ingrained prime pairs.

- **Gaussian Elimination, gauss jordan, inverse finding with gauss jordan**
  - Finding determinant
  - After performing the row operations in a set of equations, the new system we find is equivalent to the original system, i.e the solutions concur at the same points, but physically the systems are not the same
  - Can also be used to solve the equation modulo any number p. (eqn)%p
  - https://cp-algorithms.com/linear_algebra/linear-system-gauss.html **read it**

- (loj 1151 **needs gaussian+expected value, fact is in this problem, normal dp doesn't work because dp states are recursive, so we use gaussian elimination to solve for all dp states.)
- Loj 1272 https://math.stackexchange.com/questions/48682/maximization-with-xor-operator , chef , spoj
- Rank of a matrix
  - The maximum number of linearly independent vectors in a matrix is equal to the number of non-zero rows in its row echelon matrix. Therefore, to find the rank of a matrix, we simply transform the matrix to its row echelon form and count the number of non-zero rows.
  - the row rank of *A* = the column rank of *A*
  - $\text{rank}(A_{m\times n}) \leq \min(m, n)$
- Loj 1288 - solution blog
- Hackerearth problem **didn't solve yet**
- Xor gauss (gauss in GF(2))** from here
  - As others have noted, all the usual methods of solving systems of linear equations (such as Gaussian elimination) in the field of real numbers work just as well in the finite field of integers modulo 2, also known as GF(2).
  - In this field, addition corresponds to the XOR operation, while multiplication corresponds to AND (as it does in the reals, if the operands are restricted to 0 and 1). As both 0 and 1 are their own additive inverses in GF(2) (since $0 \oplus 0 = 1 \oplus 1 = 0$), subtraction is *also* equivalent to XOR, while division is trivial (dividing by 1 does nothing, dividing by 0 is undefined).

- **Mat expo**
  - Must be **linear** recurrence, and recurrence relations coefficient cannot depend on **'n'**
  - Complexity is $O(d^3 \log n)$ d^3 for matrix multiplication of a dxd matrix, and long big mod exponentiation.
  - If base matrix has 0 to k values, then to find nth term we do M^(n-k) * base_matrix
  - Zobayer Blog must basic
  - Explained HackerEarth
  - If the modulo MOD in problem is such, that MOD^2 fits into long long, then you can reduce number of modulo operations to $O(n^2)$ (in case of multiplying two n x n matrices A and B). → just reducing the number of mod operations.
    - ```
      // Since A and B are taken modulo MOD, the product A[i][k] * B[k][j] is
          // not more than MOD * MOD.
          tmp += A[i][k] * 1ll * B[k][j];
          while(tmp >= MOD2) // Taking modulo MOD2 is easy, because we can do it by subtraction
           tmp -= MOD2;
          }
        result[i][j] = tmp % MOD; // One % operation per resulting element
      ```
  - If you problem requires answering many queries of raising the same matrix M to some power, you can precalculate powers of two of M to get things done faster.
  - Fushars Blog, odd-even-cf , Fushar Part-2
  - Loj-1132 summing up powers - wow!
  - Repetition of fibonacci/tribonacci etc f(for huge n)%m Algorithmic toolbox problem, medium tutorial
    - For f(n)%m the period lengths are pisano periods(wolfram)
    - Period always starts with 01, so we iterate fibonacci, till we find 01 again, and the length is the n till that point.
    - ```
      long long get_pisano_period(long long m) {
          long long a = 0, b = 1, c = a + b;
          for (int i = 0; i < m * m; i++) {
      ```

```
            c = (a + b) % m;
            a = b;
            b = c;
            if (a == 0 && b == 1) return i + 1;
        }
    }
```

- **Inclusion Exclusion:**
  - Loj 1124 https://reponroy.wordpress.com/2015/12/23/lightoj-1124-cricket-ranking/ IE principle to find solution to x1+x2+x3=n with upper bounds on the variables
  - Loj 1144 : http://codeforces.com/blog/entry/47412 : m by n grid, me at 0,0 all other points have enemies, how many minimum ray(dont stop on shooting one) needed? So number of unique rays=no. Of such points (x,y) such that x and y are co prime, because all other points are (x*i,y*i) form. Then observe that (m/k) occurs (n/k) times in 1 to n iteration, so ans+=(m/k)*(n/k)*sign ; sign: if no. of prime is odd, plus, else minus from IE; also, any number with prime power>1 should not be considered , because each of its value will have already been calculated, when we took (m/k)*(n/k); like $2^2.3$ was already calculated while calculating 6=2.3 → (2)+(3)-(6); turns out, this is the mobius function 1,-1,0 values for signs.
- **FFT blog , CF**
  - To multiply two polynomials is its main job
  - Normally two n-degree polynomial multiplication will take n^2 , FFT does it in nlogn.
  - Any mathematical transformation is such that it is invertible, and the transformed stuffs main property doesn't change.
  - Precision chaile NT transform(working with primitive roots) lagbe
- **Probability and expected value:**
  - https://www.topcoder.com/community/competitive-programming/tutorials/understanding-probabilities/
  - https://www.codechef.com/wiki/tutorial-expectation
  - the **expected value** of a random variable, intuitively, is the long-run average value of repetitions of the experiment it represents.Less roughly, the law of large numbers states that the arithmetic mean of the values almost surely converges to the expected value as the number of repetitions approaches infinity.
  - E[x1+x2] = E[x1] + E[x2] linearity of expectation.
  - the expected number of coin flips for getting N consecutive heads is $(2^{N+1} - 2)$.
  - What is the expected number of coin flips for getting a head,P(head)=p? / No. Of interviews needed to hire someone, P(hiring)=p / expected No. of dice throws to get 4, P(4)=p → Bernoulli trial(binary outcome) : ANS: 1/p
  - If the probability of a success in a bernoulli trial is p then the expected number of successes in n trials is n*p.
  - If probability of success in a bernoulli trial is p, then the expected number of trials to guarantee N successes is N/p.
  - If x are the outcomes and p are the corresponding probabilities, then E(X)= $\sum_{1}^{n} x * p$, where $\sum p = 1$.
  - Birthday paradox: geeks_explained → given P(same birthday), whats the minimum number of people in a room. → direct formula for loj 1104

$$n \approx \sqrt{2 \times 365 ln\left(\frac{1}{1-p(same)}\right)}$$

  - http://data.faridani.me/expected-number-of-coin-tosses-to-get-one-tail/
  - Loj 1248 recursion: $E(x1) = 1$, because on first move, always one new face will reveal, $E(x2) = (n-1)/n + (1/n)(1 + E(x2))$, i.e probability to get a new face is $(n-1)/n$ + probability of getting the old face is 1/n and if we get an old face, we wasted a move (+1) and still need to find the expected no. of moves to get 2

new faces E(x2) , so (1+E(x2)); ans= sum all E(x) from x=1 to n; If turned into a (harmonic) series, it is $n \sum_{i=1}^{n} 1/i \rightarrow$ n*H_n , where H_n is the n-th Harmonic Number → although couldn't find a direct formula, maybe there isn't one.

- Loj 1267, SEE-hint or can be solved by dp
- Loj 1287 SEE ** probability dp
- Loj 1321 SEE nice
- Loj 1364, pera khaisi , see code
- Loj 1395 SEE also coupon collectors, a little diff.
- Loj 1342 SEE - boshells blog Coupon collectors problem & idea of averaging, Coupon collectors: The key to solving the problem is understanding that it takes very little time to collect the first few coupons. On the other hand, it takes a long time to collect the last few coupons. OR dp solution OR, see code, equation simplified.
- Binomial Distribution http://onlinestatbook.com/2/probability/binomial.html
- Poisson Distribution : The Poisson distribution is a limiting case of the Binomial distribution when the number of trials becomes very large and the probability of success is small.

- **Combinatorics:**
  - nCr(n,r) = nCr(n−1,r)+nCr(n−1,r−1) pascals triangle:

```
def nCr(n, r):
  if r==1: return n
  if n==r: return 1
  return nCr(n-1, r) + nCr(n-1, r-1)
```

```
ncr[0][0]=1;  int lim=10;
for(int i=1;i<=lim;i++)
    for(int j=0;j<=lim; j++){
    if(j>i) ncr[i][j]=0;
    Else if(j==i || j==0) ncr[i][j]=1;
    Else ncr[i][j] = ncr[i-1][j-1] +  ncr[i-1][j]; }
```

  - # of trailing zero in factorials (p=5, no need to count #of 2, cuz #2>#5 always), or #of any prime p → floor(n/p)+floor(n/p^2)+floor(n/p^3)+...til the term is zero(p^x>n)
  - # of digits in factorial (loj 1045) → floor($\log_{10}$n!)+1 → floor($\log_{10}$1+$\log_{10}$2+$\log_{10}$2…+$\log_{10}$n)+1
  - Multiple ways to find nCr:
    - n!/(n-r)!*r! : so in numerator, keep an array of numbers remaining after dividing by max(r! , (n-r)!), that is numbers from max(r! , (n-r)!)+1 to n then divide each number of numerator by min(r! , (n-r)!). Till denominator id 1;
    - First reduce the numerator like above, then loop from max(r! , (n-r)!)+1 to n and 1 to min(r! , (n-r)!), and each time multiply a number of nume. and deno. And then divide both by gcd.
    - If nCr % p : if p is prime and p>n then n!%p*inv(r!)*inv((n-r)!) because p will be coprime
    - If p<n (works for any p, works better for p<n cuz otherwise we cant store p*p nCr precalc) the lucas theorem: mCn = $\prod_{i=0}^{k} (miCni)$ %p where mi and ni are p base representation of m and n. We can save p*p size nCr array as all mi,ni<p.
    - For similar case as above(p<n but still quite big such that n/(p^2)=0): another approach is to turn
      - n! / (n-r)!*r!
      - => a.p^x / b.p^y * c.p^z
      - => a/bc * p^{x-y-z}
      - Here n! = a*p^K , we need to find (a mod p) = k! (n%p)! ((p-1)!)^K = k!(n%p)! (-1)^K by wilsons theorem and p^k is separated
      - K! (cuz there will be 1p.2p.3p….kp where k=(n/p)), and the rest of the stuff where %p becomed 1.2...p k times, so ((p-1)!)^K and those remaining at the end will be (n%p)!
      - So ncr = a/bc * p^{x-y-z} ; if x-y-z>=1 , ncr=0 ; and it will always be >=0 , never negative, so if it is 0, we simply calculate a/bc % p and now, p is coprime will a,b,c ; so we can calculate normally.

        - If k becomes > p , then same problem arises again, p cant be coprime , i.e when n/(p^2) >= 1, this solution won't work.
    - If nCr % M and M is not prime and square free : can express n! As $a.p1^x.p2^y\ldots$ so nCr =a/bc $.p1^x.p2^y\ldots$ => we can calculate a/bc % M now, as they are now coprime, and the $p1^x.p2^y\ldots$ can be done by bigmod...
    - If nCr % M and M is not prime, but M is square free number → go ahead with CRT ; what if the prime factors p1,p2 etc are less than n and/or r ; apply lucas for each of the primes, then CRT
    - Even if prime factors are not square free, can be solved by CRT : like if we have p^a , this is pair wise coprime with other primes or prime powers
    - In CRT, if we have equations with mod p, p^2 , p^3 plus other primes, we could say , oh well they are not pairwise coprime, solution doesnt exist, BUT, actually we could only take the equation with p^highest and discard other p^anything, then the system has solution, and it is the answer, these systems are equivalent.
  - Derangement number → # of permutation with no element in its own position
  - Rencontres Number → # of permutation with exactly k fixed points
  - Stirling number of 2nd kind → # of ways to divide n labelled objects into k unlabelled non empty groups.
  - Striling number of the 1st kind http://www-history.mcs.st-andrews.ac.uk/Miscellaneous/StirlingBell/stirling1.html → The number of ways to permute a list of *n* items into *k* cycles. Kinda like how many ways can you form k groups with each unique cycles(direction different makes cycle different, like 123 and 231 are same, but 132 isn't same)
  - Inclusion Exclusion: these problems are sometimes $O(2^n)$, if we need to keep track of which object we have/have not taken, we may use bitmask here, also fastens the code.
  - Catalan Number TomDavis PDF

$$C_n = \binom{2n}{n} - \binom{2n}{n+1} = \frac{1}{n+1}\binom{2n}{n} \quad \text{for } n \geq 0,$$

$$C_n = \frac{1}{n+1}\binom{2n}{n} = \frac{(2n)!}{(n+1)!\,n!} = \prod_{k=2}^{n} \frac{n+k}{k} \quad \text{for } n \geq 0.$$

- **String Stuffs:**
  - Hashing : **link -> also problems
    - $\text{hash}(S) = (\sum_{i=0}^{n-1} Si * pi) \% MOD$
    - a hash should depend on length and also on the order of the characters
    - We will try to use p,MODas primes and pp greater than number of distinct elements in our language.
    - Another good method is to store hash modulo two primes MOD1 and MOD2 which surely results in lesser collisions. Also, we should keep MOD such that MOD∗MOD doesn't result into overflow in C/C++.
    - 2d matrix hash uva 11019
  - KMP (Loj problems)
    - O(n) n is size of text
    - Lps is longest **proper** prefix suffix, which is size of pattern also called the 'prefix funxtion' (pi)
    - Longest palindrome from end using kmp : strNew= str # str(rev) then find lps of last char of strNew, thats is the length of the longest palindrome in str from right.
    - TCoder Rabin Karp and kmp tuto
    - Emaxx**
      - Search s(len=n) in t: find lps of s#t, for every element of t, if lps(i)=n, theres a substring in here=s
      - at the position i ends the prefix of length π[i], the prefix of length π[π[i]−1], the prefix π[π[π[i]−1]−1], and so on, until the index becomes zero.
      - ```
        vector<int> ans(n + 1);
        for (int i = 0; i < n; i++)
            ans[pi[i]]++;
        for (int i = n-1; i > 0; i--)
            ans[pi[i-1]] += ans[i];
        for (int i = 0; i <= n; i++)
            ans[i]++;
        ```
        ```

            - Here for each value of the prefix function we first count how many times it occurs in the array π(lps), and then compute the final answers: if we know that the length prefix i appears exactly ans[i]times, then this number must be added to the number of occurrences of its longest suffix that is also a prefix. At the end we need to add 11 to each result, since we also need to count the original prefixes also. This is the number of occurrences of each prefix in the same string.
            - To find occurance of all prefix of s in t: we create the string s+#+t and compute its prefix function. The only differences to the above task is, that we are only interested in the prefix values that relate to the string t, i.e. π[i] for i≥n+1. With those values we can perform the exact same computations as in the above task.
            - Given a string s of length n. We want to compute the number of different substrings appearing in it.So let k be the current number of different substrings in s, and we add the character c to the end of s. Obviously some new substrings ending in c will appear. We want to count these new substrings that didn't appear before.We take the string t=s+c and reverse it. Now the task is transformed into computing how many prefixes there are that don't appear anywhere else. If we compute the maximal value of the prefix function π_{max} of the reversed string t, then the longest prefix that appears in ss is π_{max} long. Clearly also all prefixes of smaller length appear in it.Therefore the number of new substrings appearing when we add a new character c is $|s|+1-\pi_{max}$.So for each character appended we can compute the number of new substrings in O(n)times, which gives a time complexity of $O(n^2)$ in total.It is worth noting, that we can also compute the number of different substrings by appending the characters at the beginning, or by deleting characters from the beginning or the end.
            - Finding period of a string: Let us compute the prefix function for s. Using the last value of it we define the value k=n−π[n−1]. We will show, that if k divides n, then k will be the answer, otherwise there doesn't exists an effective compression and the answer is n. Also here
            - 2d kmp uva 11019 **didnt do** with kmp, did with 2d hashing. see idea
        - Suffix array (Loj problems) Universal PDF
        - Aho Corasick
            - Original paper : download
            - Cf tuto, animation aho, emaxx
- **Backtracking:**
    - NQueen Problem
    - Bitwise NQueen (https://github.com/johnbhurst/nqueens , http://jgpettibone.github.io/bitwise-n-queens/)
    - All topsorts
- **Dynamic Programming:**
    - List of types of DP ShakilsBlog
    - 0/1 knapsack: maximize/minimize cost(C) with sum of weight<=W. Each time we take maximum of cost[i]+rec(pos+1,cap-weight[i]) and rec(pos+1,cap) ;
    - Coin change: Set of coins given, each coin can be used infinite number of time, need to make a required amount(W). Number of ways? rec(i,amount-cost[i])+rec(i+1,amount) ; Notice the call to (i) again in the first rec, thats cuz each coin can be used infinite number of times. If number of coins restriction given, run a loop till 'limit' of each coin.
    - LIS: code in 'precode'. **Unsolved** এখন একটি প্রশ্ন: একটা sequence এর কয়টি LIS আছে সেটা বের করতে বলা হলে কি করবে? চিন্তা করে জানাও :-)।
    - bit mask dp when to use? বিটমাস্ক লাগবে আমাদের তখনই যখন আগের স্টেটে কোন কোন জিনিস/ডিজিট/নোড ইত্যাদি ব্যবহার করা হয়েছে সে তথ্যটি আমার বর্তমান স্টেটে লাগবে।
    - LCS, EDIT distance:
        - Lexicographically minimum lcs (loj 1110 -- string lcs)

        - Longest Common Increasing Subsequence (chef)
        - # of distinct lcs between two strings (loj 1157)
    - MCM(matrix chain multiplication)
        - Total scalar multiplication to multiply m by n and n by q matrices is m*n*q
        - We need to find the minimum number of scalar multiplications to multiply the matrices A*B*C (or more) from all possible ways example -- by doing (A*B)*C or A*(B*C)?
        - Dp[beg][end] = min(rec(beg,i)+rec(i+1,end))+i.rowsize*i.colsize*end.colsize //for all i : beg <= i < end
        - Base case? if(b>=e) return 0;
        - Space O(n^2) and time O(n^3) → n^2 dp and loop of n = n^3
- **Geometry:** topics and links
    - Triangle, circle, sphere etc area, volume, surface area formula, angle formulas, triangle trigonometric formulas, for area and angles.(done)
    - Given three sides of triangle
        - Radius of circumcircle
        - Of innercircle
        - Length of three medians
        - Acute/obtuse/right? For the angle A, a^2 = b^2 + c^2. = Hoile right, < hoile acute, > hoile obtuse.
    - To find angle of a triangle, if we use asin, we don't know if the answer is x or 180-x , there are other possible values but none are applicable for triangle, as any angle of triangle wouldn't be more than 180, so its **better we use acos**, which was distinct values from 0 to 180 degrees.
    - asin , acos , sqrt haves when given double values after some computation, slight errors like -0.000001 for sqrt , or 1.00001 for asin may give RE, so make own functions, and do abs() in sqrt, and check for <=1 for asin.
    - Double equality a==b is abs(a-b)<=eps , a<=b is a<=b+eps
    - Line: if we use y=mx+c then problem are 1. m is floating, 2. No way to express vertical lines(x=5) 3. c calculation required. Better we use ax+by=c
        - no floating point problems
        - Easily plug third point to check sides or if on line
        - Are two given points on the same side of line? If both pos/both neg, they on same side, else not. If 0, its on the line itself.
    - Another way to express line is parametric form.
        - A line passing through vectors A and B is A+t(B-A) = the set of points (ax+t*bx-t*ax,ay+t*by-t*ay)
        - If t is [0,1] its a line segment.
        - If t is [0,inf] its a ray.
        - If t is [-inf,inf] its a line.
        - With t as ½ or ⅓ we can find the mid-point or one-third point.
        - Checking if a point C is on it? Solve t for A+t*(B-A) = C, now we will get two equations, one for x, one for y, see it both give same t.
        - Are two points on the same side? Find value of det [(xa,ya,1),(xb,yb,1),(xc,yc,1)] if det is 0, C is online, if pos, ABC are ccw, else cw.
    - Bisection
    - Ternary Search : For integer points Ternary search will have to stop when**(r−l)<3**, because in that case we can no longer select m1 and m2 to be different from each other as well as from l and r, and this can cause infinite iterating. Once (r−l)<3, the remaining pool of candidate points (l,l+1,…,r) needs to be checked to find the point which produces the maximum value f(x). For double points: **(r-l) > eps** where eps is the error limit
    - Stanford Geo-Lib
    - Basic Geometry Topcoder 1-3 series TC problem list
    - Line segment intersection (Geeks)

- Point in polygon (ics.com)(with implementation and corner case - http://alienryderflex.com/polygon/)
  - If a ray from the point to infinity(randomly large number) crosses odd #of edges, then it is inside. Handle on edge cases separately. Geeks → has bug, fixed in here
  - **Rather do pii extreme(p.x+inf,p.y+inf+1) so that no integer points fall in this line even, avoiding the whole falling in the vertices case**
- Pick's Theorem link : Area of polygon = B/2 + I - 1 deikho, ki ki ase
- Any line segment intersection in a set (True/False) hor/ver and any type line (hackerearth tutorial, also TC tuto) could not **succesfully implement, wa in hackerearths own solution(hor/ver), another not done**
  - Find all intersection in Horizontal-vertical segment set: O(nlogn + k) where k is the #of intersection
  - Find all intersection in any line segment set O(nlogn + klogn)
  - Both run slow when #of intersection (k) is high (n^2)
  - All line segment intersection tuto - 2 algo
- Line Sweep (topcoder tutorial, hackerearth tutorial)
  - All line segment intersection (see above)
    - hor/ver
    - Any type segment
  - Closest Pair of points
  - Rectangle Union **have code, see and implement**
  - Convex Hull
  - Manhattan minimum spanning tree **not implemented dont understand even**
- P1 x P2 = P1 is clockwise from P2 wrt (0,0) if cross product is <0 ; anticlockwise if >0 ; else collinear if 0.
- # of integer points in a line segment is $GCD(x_1-x_2, y_2-y_1)+1$ (including both end points) or -1 (excluding both end points)
- Minimum radius circle to cover all points TC solution - n^2 over all points to form circles with (i,j) pair on two sides and check if this circle encloses all points, then run n^3 to all points and find the unique circle with them and see if the circle encloses all points, minimum radii of all these circles.
- Spherical coordinates(lat,long,alt)/(phi,theta,rho) to cartesian : (A minor difference: altitude is usually measured from the surface of the sphere; rho is measured from the center -- to convert, just add/subtract the radius of the sphere.)
  - double x = sin(lng/180*PI)*cos(lat/180PI)*alt;
  - double y = cos(lng/180PI)*cos(lat/180PI)*alt;
  - double z = sin(lat/180PI)*alt;
- Say that you have two vectors originating at the origin, S and R.Then, the closest point to the origin is R if $|R|^2 + |R-S|^2 \leq |S|^2$ and it is S if $|S|^2 + |R-S|^2 \leq |R|^2$ → used for point to line segment shortest distance measurement
- Union of line segments geeks
- Angular Sweep (Maximum points that can be enclosed in a circle of given radius) geeks **DIDNT DO**
- Find minimum radius such that atleast k point lie inside the circle of center(0,0) ; find distance of all points from (0,0) ; sort ; the k-th smallest distance is the answer.
- Minimum cost polygon triangulation O(n^3) dp geeks
- Find number of diagonals in n sided convex polygon ; ${}^nC_2$-n = n(n-3)/2
- Two divide and conquer methods of convex hull finding geeks1 , geeks2-quickhull
- Finding upper and lower tangent of two convex hulls geeks
- Misc:
  - Radius of circle from width and height of arc : http://www.mathopenref.com/arcradiusderive.html

- **Graph Theory:**
  - Following Shafaet + prog. Contest book:

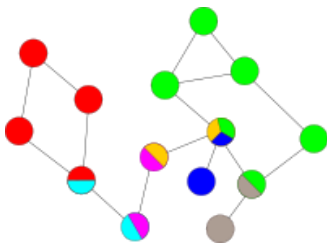
        - **হ্যান্ডশেকিং লেমা** একটা জিনিস আছে যেটা বলে একটা বিজোড় ডিগ্রীর নোডের সংখ্যা সবসময় জোড় হয়
        - একটা গ্রাফের **ডিগ্রীগুলোর যোগফল** হবে **এজসংখ্যার দ্বিগুণ**
        - গ্রাফ **আনডিরেক্টেড** হলে **ম্যাট্রিক্সটি সিমেট্রিক** হয়ে যায়
        - To keep track of path, in bfs/dfs traversal : previous[u]=v , while traversing
    - Bicoloring (bfs)
        - If a graph has odd length cycle, its **not** bicolorable.
    - Dfs:
        - In normal recursive method, worst case, there can be “v recursions” in compiler stack, sometimes this may result in stack overflow. To avoid it, we have to implement dfs by using ‘stl stack’.
        - (implemetation in prog conts book) → also here (https://www.geeksforgeeks.org/iterative-depth-first-traversal/)
        - যদি কোনো সময় একটি গ্রে(running node) নোড থেকে আরেকটি গ্রে নোডে যেতে চেষ্টা করে তাহলে সে এজটি **ব্যাকএজ** এবং গ্রাফে অবশ্যই সাইকেল আছে. For undirected graph going to any visited node(gray/black i.e. running or operation over) is a back-edge.
        - আর যখন আমরা স্বাভাবিক ভাবে গ্রে থেকে সাদা নোডে(unvisited) যাচ্ছি তখন সে এজগুলোকে বলা হয় **ট্রি এজ**।
        - শুধুমাত্র ট্রি এজ গুলো রেখে বাকি এজগুলো মুছে দিলে যে গ্রাফটা থাকে তাকে বলা হয় **ডিএফএস ট্রি**
    - Bfs/Dfs problem :
        - 2 L(empty), 3L(empty) and 8L(full) glasses. Can you separate 4L (dfs enough), in how many minimum steps (bfs). 3d queue, with state of each glass, and all possible operations are edges.
        - Shortest path with 0/1 cost: 0 does **NOT** mean no edge, the cost is zero. No need to go to dijkstra, use a dequeue. Similar bfs function, **0-cost** edges pushed to **front**, **1-cost** edge pushed to **back.** Always **pop** from **front.** Rest is bfs like, keep a dist array etc.
    - Prims / Kruskal
        - Second most minimum spanning tree: find mst, find E more mst by removing each edge, and find the minimum
    - Topsort (bfs)
        - Algo:
            - Keep indegree
            - Queue ‘0’ indegree nodes
            - Run queue, visit and keep reducing indegree
            - When some nodes’ indegree is ‘0’ , push it.
            - The order followed is the ‘required topsort’
        - If after queue is empty some nodes remain (with non zero indegree) it means graph has cycle.
        - To find all possible topsort need to use “backtracking”.
        - There's problem to find lexicographically minimum topsort.
    - Topsort(dfs)
        - Topsort not possible if there is a **directed cycle** in the graph, can be used to detect cycle too. If the current node goes to a gray node then cycle, if goes to a black node, do nothing, if goes to a white node, go.
        - Keep start and end time of each node
        - যে নোডটি সবার আগে আসবে তার finishing time অবশ্যই সবথেকে বেশি হবে,কারণ প্রথম নোডের উপর নির্ভরশীল সব নোড ঘুরে আসার পরে সে নোডের finishing time assign করা হয়। So **sort** by **finishing time**, that's your topsort.
        - *** OR SIMPLY :- while returning from a node, push it to a array/stack. This will have the reverse order of our topsort.
        - Problem: min #of dominos to hit, to drop all dominos.:
            - Topsort
            - From topsort, one by one run dfs for unvisited , count of dfs is the count of dominoes to knock down
    - Dijkstra Algo
        - Path relaxation : if dis[u]+cost[u][v] < dis[v] : dis[v] = dis[u]+cost[u][v]

        - Mark visited when popping from priority_queue , cuz its shortest distance has now been found and is fixed.
        - Works only on **NON NEGATIVE** edge weight.
        - https://www.quora.com/Can-you-add-a-large-constant-to-a-weighted-graph-with-negative-edges-so-that-all-the-weights-become-positive-then-run-Dijkstras-algorithm-starting-at-node-S-and-return-to-the-shortest-path-found-to-node-T-in-order-to-still-get-the-shortest-path (because then the cost becomes path length dependent and so gives wrong answers)
        - Second shortest path: **notice**, when we update with the minimum distance, the previous value was the second shortest distance :v
    - Bellman ford
        - If graph has **negative cycle**, need to use Bellman-ford (to at least detect the cycle)
        - Relax all the edges n-1 times, in any order (**shuffle** the edge list to be safe)
        - i-th time we relax all the edges, we get the shortest paths of all nodes using at least(can be mpre too) i edges. **But** even shorter length may get updated, depending on the ordering of edges. Like i-th relaxation may update a node with shortest path using >i edges.
        - Can break the n-1 loop, when no more updates are occuring.
        - If the graph realxes n-th time, then theres a negative cycle connected with the source, shortest paths can't be found.
        - Q : which nodes are in the negative cycle? (Related to scc)
        - ftp://ftp.informatik.uni-stuttgart.de/pub/library/ncstrl.ustuttgart_fi/TR-2010-05/TR-2010-05.pdf (please take time to read it and summarize here)
        - Loj 1108 :
            - A negative weight cycle will not be changed by reverse graph. But as the problems wants the node of the negative weight cycle as well as all the nodes that can lead me to that cycle . The 2nd part of the problem is easier if we do it in reverse graph . Because in reverse graph that the node reachable form negative cycle are the nodes that can lead me to the negative cycle in real graph .
            - Another way → my solution involves splitting the original graph in its strongly connected components (with Tarjan's algorithm), then running Bellman Ford on every SCC, and finally doing DFS to find all vertices that can reach a SCC with a negative cycle.
            - Build a reverse graph (you don't really need the original graph)
            - Apply the Bellman Ford algorithm over the reverse graph (depending on your implementation, you might need to connect everything to a dummy node or not)
            - Every time BF relaxes some vertex in the last iteration, run a DFS/BFS from that vertex(need to dfs from vertex that is part of the negative cycle, in reverse graph, only cycles vertices will relaxed only), because every node that is reachable from it is part of the solution.
            - That's it. Print the answer in order.
            - *** soln'
    - Stable marriage problem (loj) :
        - Results in stable marriage, but not optimal, optimal only for those who send the request.
        - Can be solved by modification of bipartite matching also.
    - Floyd Warshall (**all pair** shortest path):
        - Need to keep both (i,j) and (j,i) values for bi-directional.
        - for k from 1 to |V|
            for i from 1 to |V|
              for j from 1 to |V|
                if matrix[i][j] > matrix[i][k] + matrix[k][j]
                  matrix[i][j] ← matrix[i][k] + matrix[k][j]
        - No, **can't put k-loop inside,** look at these cases :

        - i - k - j → shortest path is nodes 1 - 5 - 3 - 2
        - i - j - k → shortest path is nodes 5 - 4 - 3 - 2
    - Works for negative edges
    - If any of the (i,i) nodes has neg, value (originally was 0) then graph has negative cycle.
    - **PATH** ধরো আমাদের একটা ম্যাট্রিক্স আছে **next[][]**। এখন next[i][j] দিয়ে আমরা বুঝি i থেকে j তে যেতে হলে পরবর্তী যএ যেতে হবে সেই নোডটা।তাহলে একদম শুরু-তে সব i,j এর জন্য next[i][j] = j হবে। কারণ শুরু-তে কোন "মাঝের নোড" নেই এবং তখনও শর্টেস্ট পাথ বের করা শেষ হয় নি।এখন আমরা যখন matrix[i][j] আপডেট করবো লুপের সেটার মানে হলো মাঝে একটা নোড k ব্যবহার করে আমরা যাবো। লক্ষ্য কর আমরা কিন্তু মূল গ্রাফে সরাসরি এজ দিয়ে i থেকে k তে নাও যেতে পারি, আমরা শুধু জানি i,j নোড দুটোর মাঝে k আছে যেখানে আমাদের যেতে হবে j তে যাবার আগে। i থেকে k তে যাবার পথে পরবর্তি যে নোডে যেতে হবে সেটা রাখা আছে next[i][k] তে! তাহলে next[i][j] = next[i][k] হয়ে যাবে।
    - for k from 1 to |V|
        for i from 1 to |V|
          for j from 1 to |V|
            if matrix[i][k] + matrix[k][j] < matrix[i][j] then
              matrix[i][j] ← matrix[i][k] + matrix[k][j]
              next[i][j] ← next[i][k]

      findPath(i, j)
        path = [i]
        while i ≠ j
          i ← next[i][j]
          path.append(i)
        return path
    - **Transitive Closure :**
        - matrix[i][j] = 1 যদি i থেকে j তে সরাসরি এজ থাকে
        - matrix[i][j] = 0 যদি এজ না থাকে
        - এখন আমরা এমন একটা ম্যাট্রিক্স তৈরি করতে চাই যেটা দেখে বলে দেয়া যাবে i থেকে j তে এক বা একাধিক এজ ব্যবহার করে যাওয়া যায় কিনা। আমরা চাইলে উপরের মত করে o এর জায়গায় ইনফিনিটি দিয়ে শর্টেস্ট পাথ বের করে কাজটা করতে পারতাম। কিন্তু এক্ষেত্রে "OR" আর "AND" অপারেশন ব্যবহার আরো **দ্রুত** কাজটা করা যায়। এখন আপডেটের শর্তটা হয়ে যাবে এরকম:
        - **matrix[i][j] = matrix[i][j] || (matrix[i][k] && matrix[k][j])**
        - এটার মানে matrix[i][j] তে তখনই ১ বসবে যখন হয় "matrix[i][j] তে ১ আছে" অথবা "matrix[i][k] এবং matrix[k][j]" দুটোতেই ১ আছে। তারমানে হয় সরাসরি যেতে হবে অথবা মাঝে একটা নোড k ব্যবহার করে যেতে হবে।
- Articulation Point:
    - if deletion of a node increases #of component
    - Cut node, articulation node, critical point
    - Only for undirected graph
    - If u is **root** and no_of_children_of_root>1 , u is articulation point
    - If **not root** : if none of the elements in subtree(u) have a backedge to ancestor(u), then u is articulation point.
    - Low[u] → min (all disc[] of nodes in subtree(u) and all nodes to which subtree(u) has a backedge)
    - যদি d[u]<=low[v] হয়, তাহলেই শুধুমাত্র u একটা আর্টিকুলেশন পয়েন্ট হবে।
    - দুটি নোডের মধ্যে একাধিক এজ থাকলে অবশ্য এটা কাজ করবে না। তখন কি করতে হবে সেটা চিন্তা করা তোমার কাজ!
    - Problem: #of components each articulation points divides the graph into:
        - Suppose there are 'd' components in the whole graph (d>1 for disconnected graph) = #of dfs done
        - For normal vertex: (#of times low[v]>=d[u] is true + 1) + (d - 1)
        - For root : #of children + (d - 1)

- Articulation Bridge:
  - if deletion of an edge increases #of component
  - Cut edge, critical edge
  - ব্রিজ আর আর্টিকুলেশন পয়েন্টের সুডোকোডের পার্থক্য খালি এক জায়গায় : d[u]<=low[v] এর জায়গায় d[u]<low[v] লিখতে হবে।
  - Unique Path problem: q queries: is there a path from u to v?
    - There will be multiple paths if atleast one edge in the pair of paths is different.
    - Path will be unique is every edge in the path is a 'bridge'
    - 1st work: remove all edges other than the bridges.
    - Now are u and v connected? yes = path is unique : no = no unique path.
- SCC (Kosaraju - 2 dfs):
  - In directed graph
  - In a scc, we can go from every node to every other node.
  - If we consider each scc in a graph as a 'big-node' , then the graph with the 'big-nodes' is a DAG. Therefore, this graph can have a topsort. If n1 'big-node' is before n2 in the topsort, then the finishing times of all the nodes of the scc → n1 is greater than(thus earlier in topsort) than n2.
  - **Note** that the scc cycle remains the same in the transpose graph too.
  - Now if we run dfs from the topsorted order in transpose graph, we will get a new SCC in each dfs (because of edge reversal, going anywhere other than the cycle becomes impossible and going to previous scc also not possible cuz we would have visited that first anyways)
  - So algo: 1. Topsort 2. Run dfs in topsort order 3. Each dfs is a scc
- SCC (Tarjan - 1 dfs):
  - See code.
  - Problem: minimum number of edges to add so that whole graph is a SCC:
    - Turn graph to DAG by 1) finding scc 2) considering each scc a node
    - max(#node with indegree==0,#node with outdegree==0) is the answer
- Biconnected component:
  - A **biconnected** undirected graph is a connected graph that is not disconnected by deleting any single **vertex** (and its incident edges).
  - A **biconnected** directed graph is one such that for any two vertices *v* and *w* there are two directed paths from *v* to *w* which have no vertices in common other than *v* and *w*.
  - We will work with *undirected* now: A graph can be divided into some 'biconnected components/subgraphs' where every edge is in exactly 1 component, and the vertices can be common to multiple components, these shared vertices are the cut vertices really.



  - Conversely - If we are asked to find components such that removing and **edge** doesn't disconnect it, well that's actually about finding and removing all the bridges in the graph and running dfs-s to find all the components.
  - A single edge can be a biconnected component, but finding components means finding the maximal components in a graph.
  - Algo: Idea is to store visited edges in a stack while DFS on a graph and keep looking for Articulation Points(highlighted in above figure). As soon as an Articulation Point u is found, all edges visited while DFS from node u onwards will form one biconnected component. When DFS completes for one connected component, all edges present in stack will form a biconnected component. If there is no Articulation Point in graph, then graph is biconnected and so there will be one biconnected component which is the graph itself. See code.
  - **Block cut vertex tree:** a tree formed by considering each block or biconnected component as a node, and also the cut vertices as separate nodes is called block cut vertex tree.

        - So for every “connected undirected graph” we can get a “block cut vertex tree” . Quite useful this tree is.
    - Euler Trail:
        - an **Eulerian trail** (or **Eulerian path**) is a trail in a finite graph which visits every edge exactly once. Similarly, an **Eulerian circuit** or **cycle** is an Eulerian trail which starts and ends on the same vertex.
        - A graph is called Eulerian if it has an Eulerian Cycle and called Semi-Eulerian if it has an Eulerian Path
        - An *undirected* graph has Eulerian cycle if following two conditions are true.
            - All vertices with non-zero degree are connected. We don’t care about vertices with zero degree because they don’t belong to Eulerian Cycle or Path (we only consider all edges).
            - All vertices have even degree.
        - An *undirected* graph has Eulerian Path if following two conditions are true.
            - a) Same as condition (a) for Eulerian Cycle
            - b) If zero or two vertices have odd degree and all other vertices have even degree. Note that only one vertex with odd degree is not possible in an undirected graph (sum of all degrees is always even in an undirected graph)
        - A *directed* graph will have an Euler circuit if it is strongly connected and indegree=outdegree।
        - A *directed* graph will have an Euler path if it is strongly connected and one node have indegree=outdegree–1, one node has indegree-1=outdegree and for rest of the nodes indegree=outdegree.
        - Fleurys Algo to find the path/circuit ;
            - আমরা যখন ডিরেক্টেড গ্রাফ নিয়ে কাজ করবো তখন ‘underlying undirected graph’ থেকে ব্রিজ বের করবো।
            - এখন ধরে নিলাম শুরুর নোডটি হলো u(any for circuit, undirected path=odd degree node, directed path=node with in=outdegree-1)। এবার আমরা u এর অ্যাডজেসেন্ট যেকোনো একটি নোড v নিব এমন ভাবে যেন u–v এজটি মুছে দিলে গ্রাফটি ডিসকানেন্টেড না হয়ে যায় যায়, অর্থাৎ u–v যেন ব্রিজ না হয়। এরপর u–v এজটা মুছে দিয়ে আমরা পরের নোড vনোড থেকে আবার একই কাজ করবো। কিন্তু এমন হতে পারে যে এমন কোনো এজ u–v নেই যেটা একটি ব্রিজ নয়, সেক্ষেত্রে ব্রিজ ধরেই এগিয়ে যাবো মুছে । এভাবে যতক্ষণ না সবগুলো এজ মুছে ফেলা হয় ততক্ষণ আগাতে থাকবো। সবগুলো এজ মুছে ফেলা হয়ে গেলে আমরা অয়লার সার্কিট বা পাথও খুজে পাবো।
            - মূল ধারণাটা হলো একপাশের সবগুলো এজ ঘুরে না আসার আগেই ব্রিজ ভেঙে না দেয়া। তুমি যদি একবার ব্রিজ ভেঙে ফেলো তাহলে গ্রাফটি ডিসকানেক্টেড হয়ে যাবে, তুমি ব্রিজের অন্যপাশের নোডে আর ফিরে আসতে পারবে না। তাই আমরা বুদ্ধি করে আগে সব নন-ব্রিজ ঘুরে আসবো, যখন দেখবো আর সেরকম এজ নাই তখনই শুধু ব্রিজ ভাঙবো।প্রতিবার এজ মুছে দেয়ার পর ব্রিজ খুজে বের করতে হয়(in the new graph)।
        - Heirholzer Algo:
            - এই অ্যালগরিদম মূলত অয়লার সার্কিট বের করার জন্য কিন্তু একটু বুদ্ধি খাটিয়ে অয়লার পাথও বের করা যায় .
            - **অয়লার সার্কিট গ্রাফ থেকে একটি সাইকেলের সবগুলো এজ মুছে দিলে বাকি যে কানেক্টেড গ্রাফটি থাকবে সেটা নতুন আরেকটি অয়লার সার্কিট হবে.** কারণ যখন তুমি সাইকেলের এজগুলো মুছে দিচ্ছ তখন সাইকেলের অন্তর্ভুক্ত সবগুলো নোডের ইনডিগ্রি এবং আউটডিগ্রি ১ কমছে। আমাদের কাজ হবে একটা করে সাইকেল ডিটেক্ট করা এবং সাইকেলের এজগুলোকে মুছে ফেল.
            - tour_stack = empty stack
              find_circuit(u):
                for all edges u->v in G.adjacentEdges(v) do:
                    remove u->v
                    find_circuit(v)
                end for
                tour_stack.add(u)
                Return
              The stack will contain the circuit
            - অয়লার পাথ কিভাবে বের করা যায়? খুব সহজ, প্রথম নোড থেকে শেষ নোডে একটি ডামি এজ যোগ করে সার্কিট খুজে বের করলেই পাথও পেয়ে যাবে।
            - Complexity is O(E) , works on undirected and directed graph.
            - https://www.geeksforgeeks.org/eulerian-path-undirected-graph/
            - http://iampandiyan.blogspot.com/2013/10/c-program-to-find-euler-path-or-euler.html

- Chinese postman problem http://lbv-pc.blogspot.com/2012/09/jogging-trails.html-Loj 1086 -- minimimum path covering each edge ATLEAST (not exactly) once.

- Tree Diameter:
  - Dfs from any node as root. If x is the farthest from root, then farthest from x is y(suppose). x-y is the diameter of the tree.
  - *এই অ্যালগোরিদমটা জেনারেল গ্রাফে কাজ করবেনা, শুধু ট্রি এর ক্ষেত্রে করবে। জেনারেল গ্রাফে লংগেস্ট পাথ বের করার প্রবলেম এন-পি হার্ড, আর ডায়ামিটার বের করার প্রবলেম লংগেস্ট পাথ প্রবলেমেরই স্পেশাল কেস।*
- Longest path problem:
  - From a source, with unweighted edges
  - Wecan obviously make any path 'infinitely long' by cycling.
  - We can say → find longest path from sources, without visiting any node twice
  - Or we say → how to find the longest simple path
  - Hamilton path problem says find a the longest path from a node, of length n-1, hamilton path is NP complete, i.e. no polynomial solution
  - If we could solve longest path problem we could also solve hamiltonian path problem.
  - It is np-hard problem, with backtracking (exponential time) we can find it.
- Longest path in DAG:
  - **Did Not read yet**
  - Find topsort; go serially to do max(dist[v],dist[u]+w);
- Max flow(Ford-Fulkerson):
  - TP-maxflow:
    - In final-flow: incoming==outgoing for all nodes except source and destination
    - The algorithm (known as the Ford-Fulkerson method) is guaranteed to terminate: due to the capacities and flows of the edges being integers and the path-capacity being positive.
    - As a side note, the algorithm isn't guaranteed to even terminate if the capacities are irrationals.
    - A cut in a flow network is simply a partition of the vertices in two sets, let's call them A and B, in such a way that the source vertex is in A and the sink is in B. The capacity of a cut is the sum of the capacities of the edges that go from a vertex in A to a vertex in B. The flow of the cut is the difference of the flows that go from A to B (the sum of the flows along the edges that have the starting point in A and the ending point in B), respectively from B to A, which is exactly the value of the flow in the network, due to the entering flow equals leaving flow – property, which is true for every vertex other than the source and the sink.
    - 'A' set → every vertex that is reachable by a path that starts from the source and consists of non-full forward edges and of non-empty backward edges.
    - **max-flow min-cut theorem** states that the value of the maximum flow through the network is exactly the value of the minimum cut of the network. Minimum cut = the sum of capacities of the cut vertices(ie. given a weighted directed graph, remove a minimum-weighted set of edges in such a way that a given node is unreachable from another given node.).
    - To find augmented path: dfs-->poor ; bfs → a little better as finds shortest augmented path ; pfs → We assign as a priority to each vertex the minimum capacity of a path (in the residual network) from the source to that vertex. We process vertices in a greedy manner, as in Dijkstra's algorithm, in decreasing order of priorities, so as to find the maximum flow containing augmented.
    - Problems: You are given the in and out degrees of the vertices of a directed graph. Your task is to find the edges (assuming that no edge can appear more than once)
      - We can compute the number M of edges by summing the out-degrees or the in-degrees of the vertices. If these numbers are not equal, clearly there is no graph that could be built.First, build a network that has 2 (in/out) vertices for each initial vertex. Now draw an

edge from every out vertex to every in vertex. Next, add a super-source and draw an edge from it to every out-vertex. Add a super-sink and draw an edge from every in vertex to it. for each edge drawn from the super-source we assign a capacity equal to the out-degree of the vertex it points to. As there may be only one arc from a vertex to another, we assign a 1 capacity to each of the edges that go from the outs to the ins. As you can guess, the capacities of the edges that enter the super-sink will be equal to the in-degrees of the vertices. If the maximum flow in this network equals M – the number of edges, we have a solution, and for each edge between the out and in vertices that has a flow along it (which is maximum 1, as the capacity is 1) we can draw an edge between corresponding vertices in our graph. Note that both x-y and y-x edges may appear in the solution.

- Bottlneck → lowest capacity in a path.
- গ্রাফের প্রতিটা এজ (u,v) এর ক্যাপাসিটি হবে এজ টার residual ক্যাপাসিটির সমান।
- প্রতি এজ (u,v) এর জন্য উল্টা এজ (v,u) এর residual ক্যাপাসিটি হবে (u,v) এজ এ ফ্লো এর সমান।
- উল্টো দিকে ফ্লো পাঠানোর মানে হলো মূল ফ্লো টাকে বাতিল করে দেয়া
- যতক্ষণ সম্ভব আমরা residual graph এ একটা Augmented path খুজে বের করবো এবং সেই পথে ফ্লো পাঠিয়ে দিবো!
- A-B unirected এজের ক্যাপাসিটি ১০ হলে Cf[A][B] = Cf[B][A]=১০ হবে।তবে এক্ষেত্রে কোন নোডে ফ্লো কত হচ্ছে সেটা বের করতে হলে তোমাকে আরেকটু বুদ্ধি খাটাতে হবে। proof of work. We can either just use these two edges, or make two more edges for each of these to act as residual edge.
- Multiple source/sink: super source and sink, connected with *directed* edges and weight=inf
- Node capacity: turn -0- node, into -0--0- two nodes, with middle edge having the node capacity.
- Edge disjoint path: paths that don't have any edge in common, how many such paths? #of flow with all edge capacity=1;
- *Uva 10806*: Edges have weights, minimizing the sum of edges in disjoint paths.

- Min-Cut:
  - Partition of the vertices into two disjoint sets is a cut, and the minimizing the sum of weights on the cut edges is min-cut.
  - Minimum cut = Maximum flow and the two sets: run a ***bfs*** from source, so visited are one set, unvisited are another set.
  - A minimum-cut with the minimum number of edges
    - Notice what happens if we **multiply each edge** capacity with a constant T. Clearly, the value of the maximum flow is multiplied by T, thus the value of the minimum cut is T times bigger than the original. A minimum cut in the original network is a minimum cut in the modified one as well.
    - **Add to each edge:** Is a minimum cut in the original network a minimum cut in this one? The answer is no. The value of the cut will be T(multiplied each edge by T then added 1) times the original value of the cut, plus the number of edges in it. Thus, a non-minimum cut in the first place could become minimum if it contains just a few edges.We can fix this by choosing T large enough to neutralize the difference in the number of edges between cuts in the network. T=one more than the number of edges in the original network. So we just find the min-cut in this new network to solve the problem correctly.
  - How many minimum edges to remove , so that source and sink are disconnected? -- undirected, unweighted graph. ans=# of flow with capacity=1, because that's the #of edge disjoint path, so if we remove one edge from each path, source and sink will be disconnected.
  - **Unsolved Q.** Which edges to remove?
  - No source/sink, just how many minimum edges to remove to make the graph disconnected? (n^2 by taking each pair as source and sink, but there's a better way!) → Because the graph should be disconnected, there **must** be another vertex unreachable from it. So it suffices to treat vertex 1 as the source and iterate through every other vertex and treat it as the sink.

        - What if instead of edges we now have to remove a minimum number of vertices to disconnect the graph? Now we are asked for a different min-cut, composed of vertices. Convert the vertex to an edge by adding a 1-capacity arc from the in-vertex to the out-vertex. Now for each two vertices we must solve the sub-problem of minimally separating them. So, just like before take each pair of vertices and treat the out-vertex of one of them as the source and the in-vertex of the other one as the sink and take the lowest value of the maximum flow. This time we can't improve in the quadratic number of steps needed, because the first vertex may be in an optimum solution and by always considering it as the source we lose such a case. Normal edge weights=inf?
    - Max flow (Dinic): see
        - Use bfs to see if flow can we sent and simultaneously construct the level graph
        - Send multiple flow to the level graph, in every flow, levels of path nodes should be 0, 1, 2… (in order) from s to t until blocking flow is reached.
        - A flow is **Blocking Flow** if no more flow can be sent using level graph.
    - Max flow (Push rebel): Faster than others for dense graph (See, did not read yet)
    - BMP with maxflow:
        - A **matching or independent edge set** in a graph is a set of edges without common vertices(no two endpoints share the same node).A **maximum matching** (also known as maximum-cardinality matching) is a matching that contains the largest possible number of edges
        - A graph with 2 vertex sets and every edge connects the two sets is called **bipartite** graph. (i.e. there is no edge connecting vertices from the same set.)
        - we will draw an edge from the super-source to each of the vertices in set A and from each vertex in set B to the super-sink. In the end, each unit of flow will be equivalent to a match between a vertex in A and a vertex in B, so each edge will be assigned a capacity of 1.(If we would have assigned a capacity larger than 1 to an edge from the super-source, we could have assigned more than one job to an employee(A). Likewise, if we would have assigned a capacity larger than 1 to an edge going to the super-sink, we could have assigned the same job to more than one employee(B).)
        - It is easy to find out whether a vertex in set B is matched with a vertex x in set A as well. We look at each edge connecting x to a vertex in set B, and if the flow is positive along one of them, there exists a match. O(VE).
        - we could drop the 2-dimensional array that stored the residual network and replace it with two one-dimensional arrays: one of them stores the match in set B (or a sentinel value if it doesn't exist) for each element of set A, while the other is the other way around. Also, notice that each augmenting path has capacity 1, as it contributes with just a unit of flow. Each element of set A can be the first (well, the second, after the super-source) in an augmenting path at most once, so we can just iterate through each of them and try to find a match in set B. If an augmenting path exists, we follow it. This might lead to de-matching other elements along the way, but because we are following an augmenting path, no element will eventually remain unmatched in the process. (Literally the BPM algo)
        - Problem See-parking-last-problem maximum bipartite matching with minimizing edge cost (run binary search over the minimum cost, or follow PFS described in the link)
    - Minimum Vertex cover:
        - Its a NP-hard problem, but solvable with dp/flow for tree(no cycle) with n-1 edges.
        - Minimum number of guards to put in nodes in order to cover all edges.
        - A--B is an edge, so if we put a guard in A, ans=1+min(put guard in B,dont put guard in B) ; if we dont put guard in A, MUST put guard in all its adjacent nodes, ans=guard in B; final answer is min (guard in A, no guard in A)
        - In a **bipartite** graph **maximum matching = minimum vertex cover** (Konigs Theorem)
        - Its complexity is better than doing BPM(for tree ofcourse), but for other graphs it cant, but BPM can.

        - From maxflow sense: in this bipartite graph, suppose there are Left nodes(L) and right node(R), and after maxflow we find nodes in set S and set T , L has mode nodes in S and some in T(found by bfs from S), so does R. So the nodes in minimum vertex cover are $\mathbf{L_T \cup R_S}$.
        - From BPM sense: in mincut we would have run a bfs from S, we would go along edges to L with remaining capacity>0, i.e actually those nodes in L which **dont** have any matching, so insert these 'unmatched' nodes in a queue for bfs. And we we would have gone from L to R if capacity>0, so the same we will do in this bfs. I,e from L we will go to all its **'unmatched'** nodes in R. From R, we go to nodes that **have** a matching in L(represents the residual edges in mincut).
    - Maximum Independent Set:
        - Opposite of Vertex cover: Select some nodes so that they dont have any edges in common. Also NP-hard for normal graph. Solvable in Bipartite graph.
        - Max Independent set = {All nodes} - {set of vertices in Minimum Vertex cover}
        - Because atleast one of the nodes of all edges is in 'min vertex cover'. So the remaining nodes can not ever have any edge in common, and thats the maximum number of it.
    - Hopcroft Karp BPM
    - Weighted BPM (Hungarian Algo) : Could run min cost max flow problem (if asked to maximize the cost of edges, just multiply by -1). Another way is Hungarian , $O(V^3)$, topcoder article read.
    - Min cost Max flow:
        - The **minimum-cost flow problem** is to find the cheapest possible way of sending a certain amount of flow through a flow network.
        - supporting edges with negative costs.
        - Variation: A variation of this problem is to find a flow which is maximum, but has the lowest cost among the maximum flow solutions. This could be called a **minimum-cost maximum-flow** problem and is useful for finding minimum cost maximum matchings.
        - Basic idea: use bellmanford / Dijkstra instead of bfs in flow algo
        - SEE this-hacker-earth **TODO**
- **HLD (heavy light decomposition):** a graph DS
        - https://blog.anudeep2011.com/heavy-light-decomposition/
        - List of problems in the blog
- **DSU on tree:** a graph DS
    - http://codeforces.com/blog/entry/44351
- **Data Strcuture:**
    - Segment Tree, node update, range update(lazy propagation)
    - GSS(1-5) Spoj , kgss (first and second maximum) , BRCKTS (see this)
    - Segment tree problems https://jaskamalkainth.github.io/Segment-tree-Problems/ , http://codewarriers.blogspot.com/2016/02/segment-trees-and-lazy-propagation.html
    - Mos Square Root decomposition
    - Treap:
        - QMAX3VN, GSS6
- **Centroid Decomposition:** a graph DS
    - Petr-Blog , TanujKhattar , Medium(detailed***)
    - Gaurav sen video and correction video
    - https://www.youtube.com/watch?v=2izuGA8T8IE Problems list in video description
    - Given a tree with N nodes, there exists a vertex whose removal partitions the tree into components, each with at most N/2 nodes. (i.e. For any given tree, the centroid always exists)
    - height of the centroid tree is log(N)

- Consider any two arbitrary vertices A and B and the path between them (in the original tree) can be broken down into A–>C and C–>B where C is LCA of A and B in the centroid tree.
- Time to build the tree = n logn
- Query Time = logn (lca find) + (logn)^2 (A to lca) + (logn)^2 (B to lca) (another logn for finding distance from A to lca in original tree)
- Update is also (logn)^2
- Problems CF-E , Qtree 5, Qtree 4

- **Game Theory:**
  - *My notes have detailed*
  - Shafaet done
  - Nim game variation: http://www.suhendry.net/blog/?p=1612 - good(y) done
  - http://codeforces.com/blog/entry/44651 - staircase nim done, problem not done
  - **NEEDTO** https://blog.plover.com/math/sprague-grundy.html leonard boshell says its good, so its good\
  - http://www.gabrielnivasch.org/fun/combinatorial-games **DO**
  - Normal win/lose
  - Nim, Index k nim, misere nim and many more variations
  - Sprague Grundy theorem !fuchka! >>Damned: Loj 1229 editorial <<
  - Green HackenBush : 1. Colon principle 2. Fusion Principle (search and solve more)
  - Red Blue HackenBush https://discuss.codechef.com/questions/29027/hackenbush **SOLVE**
  - https://www.youtube.com/playlist?list=PLMCXHnjXnTnuolrTKzZkTMGmQNEP3NaBa
  - http://www.geeksforgeeks.org/introduction-to-combinatorial-game-theory/
  - https://www.math.ucla.edu/~tom/Game_Theory/Contents.html
  - Combinatorial Games by Matej Antol
  - https://www.topcoder.com/community/data-science/data-science-tutorials/algorithm-games/
- **Stir Your Brain:**
  - Loj 1189

Did You Know?

- The use of **__builtin_popcount()**
- $\mathbf{Log}_b$ x = $\mathbf{Log}_a$ x/$\mathbf{Log}_a$ b (calculate log in any base)
- C++ string in printf myString.c_str()
- cin/cout → speedup → ios_base::sync_with_stdio(false);
  https://stackoverflow.com/questions/31162367/significance-of-ios-basesync-with-stdiofalse-cin-tienull
- Don't use "\n" with cin.tie(false). Will give WA
- Try avoiding unordered_map, its amortized complexity is O(1) but it may turn to O(n), will be a disaster then.
- C string to long int (in any base) conversion :
  - /* **strtol** example */

```
#include <stdio.h>      /* printf */
#include <stdlib.h>     /* strtol */

int main ()
{
  char szNumbers[] = "2001 60c0c0 -110111010011010010000 0x6fffff";
  char * pEnd;
  long int li1, li2, li3, li4;
  li1 = strtol (szNumbers,&pEnd,10);
```

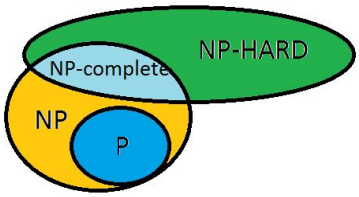
```
 li2 = strtol (pEnd,&pEnd,16);
 li3 = strtol (pEnd,&pEnd,2);
 li4 = strtol (pEnd,NULL,0);
 printf ("The decimal equivalents are: %ld, %ld, %ld and %ld.\n", li1, li2, li3, li4);
 return 0;
}
```

    - The decimal equivalents are: 2001, 6340800, -3624224 and 7340031
    - The function first discards as many whitespace characters as necessary until the first non-whitespace character is found. Then, starting from this character, takes as many characters as possible that are valid following a syntax that depends on the base parameter, and interprets them as a numerical value. Finally, a pointer to the first character following the integer representation in str is stored in the object pointed by endptr.
    - If the value of base is zero, the syntax expected is similar to that of integer constants, which is formed by a succession of:
        - An optional sign character (+ or -)
        - An optional prefix indicating octal or hexadecimal base ("0" or "0x"/"0X" respectively)
        - A sequence of decimal digits (if no base prefix was specified) or either octal or hexadecimal digits if a specific prefix is present
    - If the base value is between 2 and 36, the format expected for the integral number is a succession of any of the valid digits and/or letters needed to represent integers of the specified radix (starting from '0' and up to 'z'/'Z' for radix 36). The sequence may optionally be preceded by a sign (either + or -) and, if base is 16, an optional "0x" or "0X"prefix.

    - If the first sequence of non-whitespace characters in str is not a valid integral number as defined above, or if no such sequence exists because either str is empty or it contains only whitespace characters, no conversion is performed.
- Forgot about these? %s , %[ABCD] , %[A-Z] , %[^789]
- If we run through a 2d array for(row)for(col), its is at least twice as fast , than running it for(col)for(row) : because of cpu caching
- //But by default we don't have hash function for pairs so we'll use hash function defined in Boost library unordered_map<pair<int, int>, int,boost::hash<pair<int, int> > > Mymap;

Complexities:

- Types of complexities:
    - **P(Polynomial) problem:** P বা পলিনমিয়াল ক্যাটাগরি প্রবলেমকে ডিটারমিনিস্টিক টুরিং মেশিন (our computers) দিয়ে পলিনমিয়াল টাইমে সলভ করা যায়।
    - **NP(Non-deterministic Polynomial) problem:** NP প্রবলেমকে ডিটারমিনিস্টিক টুরিং মেশিন দিয়ে পলিনমিয়াল টাইমে ভেরিফাই করা যায়। কিন্তু পলিনমিয়াল টাইমে সলভ করা যায় কি যায় না সেটা প্রমাণ করা সম্ভব হয়নি। P হলো NP এর একটা সাবসেট।কারণ সব P প্রবলেমকে পলিনমিয়াল টাইমে ভেরিফাই করা যায় যেটা NP ক্যাটাগরির একমাত্র শর্ত।আমরা জানিনা P=NP নাকি P!=NP। এটা কম্পিউটার সায়েন্সের সবথেকে বিখ্যাত একটি ওপেন প্রবলেম।



P!=NP ধরে নিয়ে ভ্যান ডায়াগ্রাম

    - **NP-hard:** NP কে পলিনমিয়াল টাইমে সলভ করা যায়না, ভেরিফাই করা যায়। **NP hard** কে পলিনমিয়াল টাইমে সলভ করা যায়না, ভেরিফাই করা যেতেও পারে, নাও যেতে পারে। its atleast as hard as NP problem, coonvertible to NP problems.
    - **NP complete:** একটা প্রবলেম NP-complete হবে কেবল যদি প্রবলেমটা NP-hard হয় এবং সেটা NP ক্যাটাগরীতেও পড়ে। **NP complete** প্রবলেমকে পলিনমিয়াল টাইমে অন্য যেকোনো NP প্রবলেমে কনভার্ট করা যায় এবং এই প্রবলেমের সার্টিফিকেটকে(an example) পলিনমিয়াল টাইমে ভেরিফাইও করা যায়।
    - ২-স্যাটকে (Satisfiability problem of 2 variables) পলিনমিয়াল টাইমে সলভ করা যায়, k>2 হলে পলিনমিয়াল সলিউশন পাওয়া যায়না। Its NP-complete problem.
- Bfs/Dfs - O(V+E) because traversed each node once and each edge once
- PRIMS is ElogE, can be reduced to ElogV (was told to think). Prims is good for dense graph.
- Krushkal is ElogE. Krushkal is good for sparse graph.

- Bfs like topsort (v+e)
- Dijkstra :
    - O((e+v)*logv) for binary heap (cuz deletion is O(logn))
    - reduced to O(vlogv + e) for fibonacci heap (cuz deletion is O(1))
    - With **priority queue** eloge, but why dont we use **set(**also there won't be any node repetition, we could delete same node if present already in pq, and insert the 'new valued' same node**)** ? : we could reduce to elogv, we don't usually because internally set is red-black tree which is complex and pq is a heap. Red-black trees and heaps have both 'logn' for all operations but '**constant factor**' of RBtree is high.
    - But is m ~= n^2 then set will work better.
- Bellman ford: O(VE) cuz relaxing the edge list v times.
- Floyd Warshall :
    - time complexity (n^3) ; space complexity (n^2)
    - Running dijkstra n times might work (all pair shortest) but two problems
        - Negative edge not handled
        - If m~=n^2 then nmlogm becomes n^3logn^2 , so might as well use n^3 floyd.
- DSU:
    - With path compression : logn
    - Additionally with union rank : O(inverse ackermann function)
- Hierholzer euler circuit/path finding : O(E)
- Flow-Ford Fulkerson - Edmond Karp (bfs) : O(VE^2)
- Flow-Dinic : O(EV^2). BFS to construct level graph O(E) time. Sending multiple more flows until a blocking flow (dfs) is reached takes O(VE) time. So O(E + VE)=O(VE). The outer loop runs at-most O(V) time. In each iteration, we construct new level graph. It can be proved that the number of levels increase at least by one in every iteration . So the outer loop runs at most O(V) times. Therefore overall time complexity is O(EV^2). Wiki.
- BPM with flow: The number of augmenting paths is limited by min(|A|,|B|), making the running time O(N·M), where N is the number of vertices, and M the number of edges in the graph.
- BPM recursion: O(VE)
- Hopcroft Karp Algo for BPM: O($V\sqrt{E}$)
- MCM : Space O(n^2) and time O(n^3) → n^2 dp and loop of n = n^3
- Gaussian elimination and gauss jordan O(n^3)
- Divisor sieve is O(nln(n)):
    - Divisor sieve from 1 to n: in inner loop runs for n/i for each 1 to n, so n/1 + n/2 + … = n(1+½+⅓+...) = n*H_n → appx=n*(area under 1/x)=n*integration from 1 to n of (1/x) = n*ln(n).
- KMP : O(text)
- Aho Corasick: *O(n + m + z)* Let **n** be the length of text, **m** be the total number characters in all patterns, **z** is total number of occurrences of patterns in text.

Some links:

- List of **ALL** topics, for guidance https://www.geeksforgeeks.org/how-to-prepare-for-acm-icpc/
- Code **library** https://zobayer2009.wordpress.com/code/
- Prime, totient etc related http://www.doc.ic.ac.uk/~mrh/330tutor/index.html
- SHAFAET http://www.shafaetsplanet.com/planetcoding
- Geometry basics (with code) http://geomalgorithms.com/
- Math theoretical (esp geo) http://www.mathopenref.com/
- Topic wise problem with solve (good for revising) https://f0rth3r3c0rd.wordpress.com/

- Most basic topics explained (lesser alternate to the book) http://www.personal.kent.edu/~rmuhamma/Algorithms/algorithm.html
- Yet another code **library** https://github.com/forthright48/code-library/tree/master/code_templates
- **Cmp function** http://fusharblog.com/3-ways-to-define-comparison-functions-in-cpp/
- https://discuss.codechef.com/questions/18752/what-are-the-must-known-algorithms-for-online-programming-contests
- Some topics http://shakilcompetitiveprogramming.blogspot.com/
- **MASTER** http://codeforces.com/blog/entry/23054
- **TOPCODER DP** https://www.topcoder.com/community/data-science/data-science-tutorials/dynamic-programming-from-novice-to-advanced/
- Other TP https://www.topcoder.com/community/data-science/data-science-tutorials/
- Collection of **LOTS** of problems http://www.gottfriedville.net/mathprob/index.htm

SEE:

- https://harunscorner.wordpress.com/2013/10/06/spoj-beads-solution/ see this problem, suffix array to **Duval**
- মনে করো একজন পোস্টম্যান প্রতিদিন সকালে পোস্টঅফিস থেকে সাইকেল নিয়ে বের হয়ে বিভিন্ন রাস্তায় চিঠি দিয়ে আবার পোস্টঅফিসে ফিরে আসে। কোন পথে গেলে তার সবথেকে কম কষ্ট করতে হবে? যদি রাস্তাগুলোকে একটি গ্রাফ চিন্তা করা হয় এবং গ্রাফটিতে অয়লার সার্কিট থাকে তাহলে এক রাস্তায় কখনোই দুইবার যেতে হবে না। কিন্তু বাস্তবে বেশিভাগ ক্ষেত্রেই সেটা সম্ভব না। সেক্ষেত্রে এক রাস্তা একাধিকবার ব্যবহার করলেও চেষ্টা করা হয় মোট দূরত্ব কমিয়ে আনার। এই প্রবলেমের একটি নাম আছে, এটাকে বলে **চাইনিজ পোস্টম্যান প্রবলেম**। এটা একটু অ্যাডভান্সড লেভেলের প্রবলেম যেটা সলভ করতে ওয়েটেড বাইপারটাইট ম্যাচিং বা মিন-কস্ট-ম্যাক্স-ফ্লো জানা লাগে। তোমার আগ্রহ থাকলে এই নিয়ে ঘাটাঘাটি করতে পারো।
- It's possible to generalize Euler tours to the case of mixed graphs, which contain both directed and undirected edges. See article UVa_10735 for a description of one possible algorithm for such graphs.
- See all ways to test prime
- Yarins sieve
- Segmented sieve