**কনটেস্ট টাইমে Algorithm Selection**

হুম, তুমি এখন কনটেস্টে আছো, হাতে প্রবলেম সেটটা পেয়েছো। প্রবলেমগুলার মধ্যে দিয়ে যাওয়ার সময় পেয়ে গেলে একটা পরিচিত প্রবলেম, তোমাদের মনে হচ্ছে এটা সল্ভ করতে পারবে। প্রবলেম পড়ার পরপরই তুমি এবং তোমার জিনিয়াস টীমমেট দুজন বাঘাবাঘা অ্যালগরিদম দাড় করায়ে ফেললে, তার মধ্যে থেকে সব থেকে সিম্পলটা সিলেক্ট করে কোড করার ট্রাই করতেছো। But there is a catch! তোমার অ্যালগরিদম কি আসলেই efficient??

ভালো কনটেস্ট প্রোগ্রামাররা এমন অ্যালগরিদম সিলেক্টশন বা বানানোর সময় প্রবলেমে দেওয়া ইনপুট দেখে অ্যালগরিদমটার কমপ্লেক্সিটিটা হিসাব করে নেয়। তার যথেষ্ট ভালো কারনও আছে। সাধারনত আমরা কনটেস্ট প্রোগ্রামাররা ধরে নি এখনকার কম্পিউটারগুলা 107 টি instructions মোটামুটি ১ সেকেন্ডে execute করতে পারে। এখন তুমি কনটেস্টে একটা অ্যালগরিদম তৈরী করে ফেললে যেটা কিনা ইনপুট সাইজের সাথে বিবেচনা করলে 108 বা তার বেশি হয়ে যাচ্ছে। কোড সাবমিট করলে জাজ তো তোমাকে TLE verdict দিতে বাধ্য হবেনই। তাই কোন অ্যালগরিদম চিন্তা করার আগে অবশ্যই মাথায় রাখতে হবে worst case scenario এর ব্যাপারটা। প্রবলেমে বলা থাকে সাধারনত যে testcases কতগুলা হবে এবং maximum input size কতগুলা থাকে। Worst case complexity = maximum test cases \* maximum input size হবে (যদিও এই নিয়মেই সব প্রবলেমের কমপ্লেক্সিটি হিসাব করতে যেয়ো না,  সমস্যার বর্ণনায় কিছু তথ্য এক্সট্রা দেয়া থাকতে পারে)।

তো তুমি যখন তোমার worst case complexity পেয়ে গেলে তাইলে এখন খুব সহজেই টীমমেটদের সাথে গল্প করে most efficient এবং most simpler অ্যালগরিদমটি সিলেক্ট করতে পারবে। কমপ্লেক্সিটির কথা বিবেচনা না করে যদি কোড করা শুরু করো তাইলে দেখবে কোড-টোড করা শেষ, কিন্তু TLE খাচ্ছো বারংবার।

ইনপুট সাইজ এবং টেস্টকেইজ দেখে কনটেস্টের সময়ই কিছু সাম্ভাব্য অ্যালগরিদম guess করতে পারবে। যদিও সব সময় এমনটা হবে এমন কোনো গ্যারান্টি নাই। ইনপুট সাইজ দেখে সব থেকে খারাপ complexity এর সেসব অ্যালগরিদম ব্যবহার করতে পারবে ( অথচ যেগুলা ব্যবহার করলেও TLE খাবে না) সেটার একটা টেবিল দিলাম। চেষ্টা করবে যেন ইনপুট সাইজের সাথে এর সমতুল্য বা তার চে আরও efficient অ্যালগরিদম খুজে বের করে তারপর প্রবলেম সলভ করার।

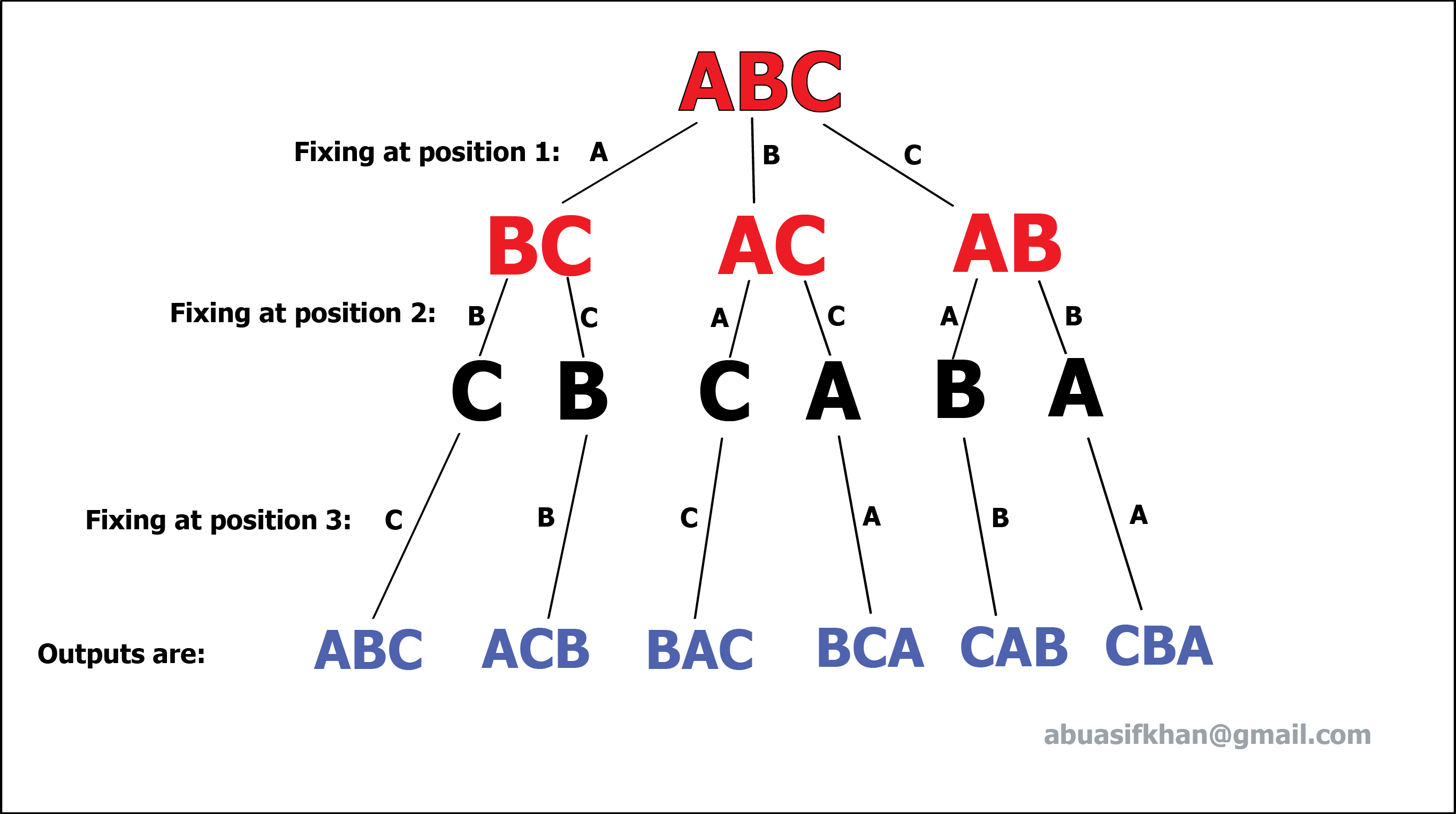
|  |  |  |  |
| --- | --- | --- | --- |
| **Maximum N** | **Maximum complexity can be applied** | **Algorithms** | **Possible data structures to use** |
| 1,000,000,000 and higher | log (n), sqrt (n) | binary search, ternary search, fast exponentiation, euclid algorithm |  |
| 10,000,000 | n, n (log (log (n))), n log(n) | set intersection, Eratosthenes sieve, radix sort, KMP, topological sort, Euler tour, strongly connected components, 2sat | disjoint sets, tries, hash\_map, [rolling hash](http://www.infoarena.ro/blog/rolling-hash)deque |
| 1,000,000 | n log n | sorting, divide and conquer, sweep line, Kruskal, Dijkstra | segment trees, range trees, heaps, treaps, binary indexed trees, suffix arrays |
| 100,000 | n log2 n | divide and conquer | 2d range trees |
| 50,000 | n1.585, n sqrt n | Karatsuba, square root trick | two level tree |
| 1000 – 10,000 | n2 | largest empty rectangle, Dijkstra, Prim (on dense graphs) |  |
| 300-500 | n3 | all pairs shortest paths, largest sum submatrix, naive matrix multiplication, matrix chain multiplication, gaussian elimination, network flow |  |
| 30-50 | n4, n5, n6 |  |  |
| 25 – 40 | 3n/2, 2n/2 | meet in the middle | hash tables (for set intersection) |
| 15 – 24 | 2n | subset enumeration, brute force, dynamic programming with exponential states |  |
| 15 – 20 | n2 2n | dynamic programming with exponential states | bitsets, hash\_map |
| 13-17 | 3n | dynamic programming with exponential states | hash\_map (to store the states) |
| 11 | n! | brute force, backtracking, next\_permutation |  |
| 8 | nn | brute force, cartesian product |  |

Keep coding… :)

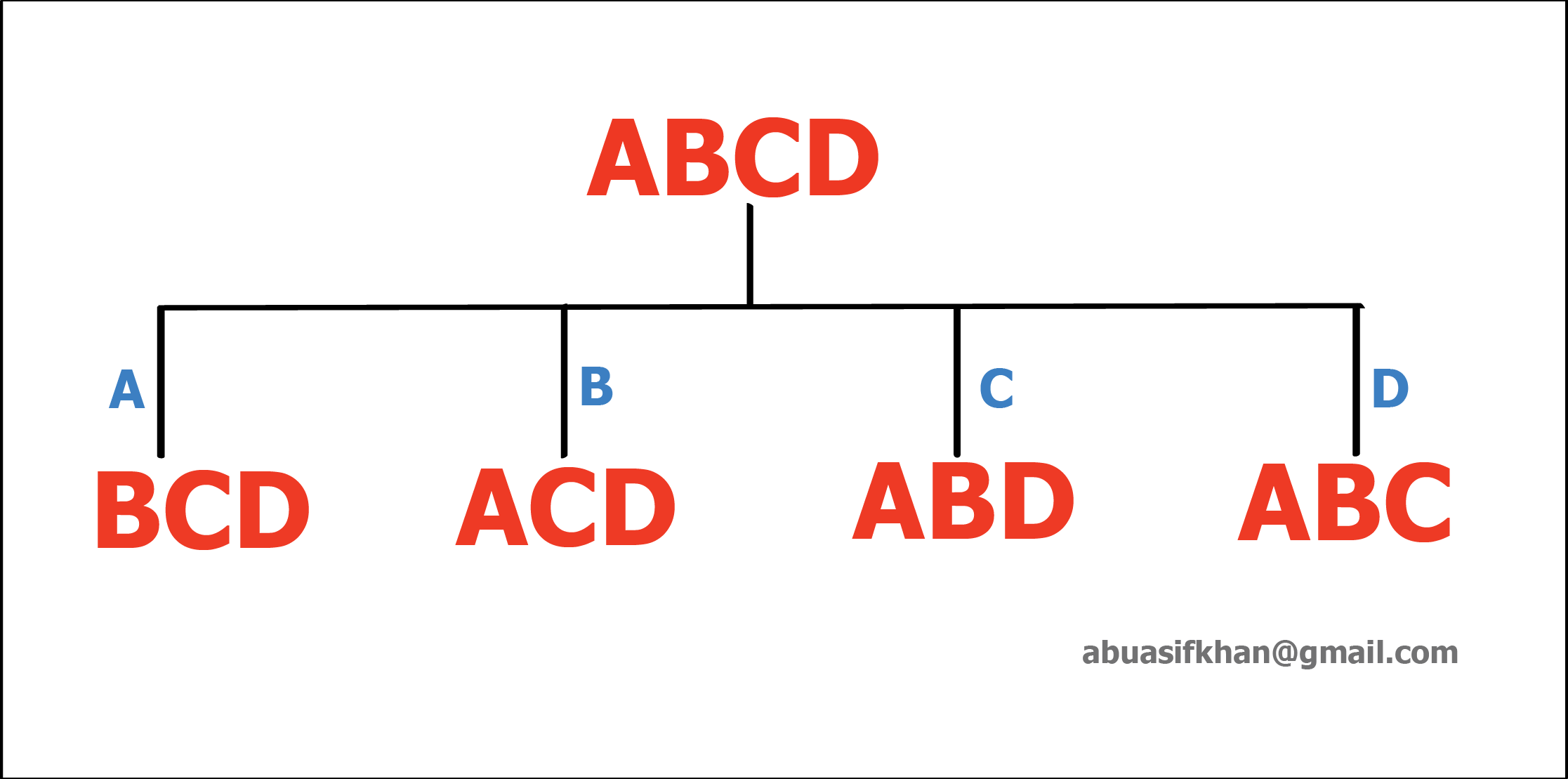
**বিন্যাস করা যাক (পর্ব: ২)**

        গতপর্বে permutation করার বেসিক জিনিসগুলা দেখানোর চেষ্টা করেছিলাম। আজকের পর্বে শেয়ার করবো কিভাবে permutation generator কোড করা যায়। কোডিং করার আগে তো আমাদের অবশ্যই জানতে হবে কিভাবে permutation generate করা হয়। আগে সেটা দেখাই।

**নিজে কিভাবে করবে?**



        উপরের ছবিটার মত ধরে নাও “ABC” এই তিনটা character গুলাকে আমরা বিন্যাস করতে চাই। তো এই বিশাল কর্ম সম্পাদনের জন্য আমাদেরকে  প্রথমে বিভিন্ন পজিশনে বর্ণগুলাকে fixed করে রেখে পরের পজিশনের কথা বিবেচনা করতে হবে। যদি একদম শেষ পজিশনে এসে যাই তাইলে প্রথম থেকে শেষ পর্যন্ত fixed করে রাখা পজিশনগুলার সবগুলা character একসাথে প্রিন্ট করে দিবো। যেমন: প্রথমে আমরা 1 no. পজিশনে ‘A’ fixed রেখেছি, তারপর 2 no. পজিশনে ‘B’, তারপর 3 no. পজিশনে রেখেছি ‘C’. সুতরাং প্রথম বিন্যাস আমরা পাচ্ছি “ABC”. এভাবে পরে 2 no. পজিশনে ‘C’ fixed করেছি। তাইলে 3 no. পজিশনে কেবল আমরা ‘B’ কেই বসাতে পারি। তাই পরের বিন্যাসটা আমরা পাচ্ছি “ACB”. এভাবে ট্রীতে আস্তে আস্তে উপরে লেভেলে যেয়ে যেয়ে এবং বিভিন্ন পজিশনে বিভিন্ন character বসিয়ে আমরা সকল সাম্ভাব্য বিন্যাস generate করতে পারবো।

        যদি 4 টা character দেয়া থাকে বা তার বেশি? তখন কিভাবে করবে? উপরে যেভাবে করেছো সেভাবেই করে যাবে। আচ্ছা জিনিসটা সহজ করার জন্য নিচে “ABCD” characterগুলা থেকে সকল বিন্যাস গঠন করার প্রথম ধাপটা দিলাম।

        1 no. পজিশনে প্রথমে ‘A’ fixed রেখে বাকি characterগুলা অর্থাৎ “BCD” এর বিন্যাসগুলা প্রথম চিত্রের মত করে করে ফেলো। তারপর প্রথম পজিশনে ‘B’ কে fixed রেখে “ACD” characterগুলা আগের মত উপায়ে বিন্যাস করে ফেলো। এভাবে বাকি দুইটা করে ফেললে তুমি মোট 4! = 24 টা সাম্ভাব্য বিন্যাস গঠন করতে পারবে। কাজ সহজ, খাটনি একটু এই আর কি। :P

**প্রোগ্রামে কিভাবে করবে?**

        আচ্ছা এখন million dollar question হলো কিভাবে তুমি এটা প্রোগ্রামে করতে পারবে? যদি একটা সাধারন প্রোগ্রামারকে বলা হয় যে তিন এলিমেন্টের কোন অ্যারের বিন্যাসগুলা খুজতে সে খুব সহজেই ৩ টা for loop ব্যবহার করে সেটা করে ফেলতে পারবে। কিন্তু যদি বলি ৪, ৫, ১০, ২০? তখন কি প্রতিবার অতগুলা লুপ সে তৈরী করতে পারবে? নাহ, এভাবে সব কার্যদ্ধার করা যাবে না। এজন্য আমাদের লাগবে রিকার্সিভ ব্যাকট্রাকিং। এই টেকনিকের বড় সুবিধা হলো আমরা সম্পুর্ন search spaceটা search করতে পারি অত লুপ-টুপ ব্যবহার না করেই। তবে সমস্যাও আছে, search space অনেক বড় হলে এই পদ্ধতি  খুব বেশি সময় অপচয়কারী হয়ে দাড়ায়। এক্সপোনেন্সিয়াল আকারে বাড়তে থাকে বলে খুব বড় ইনপুটের ক্ষেত্রে সমাধান বের করতে কয়েক বছর লাগিয়ে দিতে পারে, যেটা কনটেস্টের কোন প্রবলেমের সমাধান হিসেবে জাজদের মনমত হওয়ার সুযোগ কম, TLE খাওয়া লাগতে পারে। তবে সকল সাম্ভব্য বিন্যাস generate করতে এটাই ব্যবহার করা হয়। এখন কাজের কথায় আসি, আমরা all possible permutation generate করার জন্য একটা ফাংশন তৈরী করবো ব্যাকট্রাকিং এর সাহায্যে যেটাতে আমরা উপরের যে পদ্ধতি বিন্যাস করার জন্য ব্যবহার করেছি সেটাই করবো। নিচে কোডটা দিলাম, ওখানে ব্যাখ্যাও দেয়া আছে একটু-অর্ধেক…

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31 | #include <iostream>  #include <cstdio>  #include <vector>    using namespace std;    vector<char>permuted;  int fixedPos[20]= {0};    ///ব্যাকট্রাক মেথড  void generate(int array[]){      if(permuted.size()==4){    /// যদি সব কয়টা পজিশন ফিক্সড হয়ে যায়          for(int i=0; i<4; i++)              printf("%c",permuted[i]); ///তাইলে প্রিন্ট করতে হবে permuted result          printf("\n");          return;      }      for(int i=0; i<4; i++){          if(fixedPos[i]==0){    /// যদি অ্যারের i-তম পজিশনের characterটা ফিক্সড করা না থাকে              fixedPos[i]=1;        /// তাইলে fixed করে নিলাম              permuted.push\_back(array[i]);    /// ফিক্সড character টা save করে রাখা হলো              generate(array);    /// ট্রীতে নিচের লেভেলে যাওয়ার জন্য রিকার্সিভ কল।              fixedPos[i]=0;        /// i-তম পজিশনে array[i] character টা রেখে কাজ করা শেষ।              permuted.pop\_back();    /// কাজ শেষ তো character টা তো save রাখার আর দরকার নাই          }      }  }  int main(){      int array[] = {'A', 'B', 'C', 'D'};      generate(array);  } |

আপাতত এ পর্যন্তই। পরবর্তি পর্বে n-তম বিন্যাস কিভাবে বের করা যায় সেটা দেখানোর চেষ্টা করবো। আরও ভালো জানতে শাফায়েত ভাইয়ের লেখা [আর্টিকেলটা](http://www.shafaetsplanet.com/planetcoding/?p=1266)পড়তে পারো। আর যা যা শিখলে সেগুলা প্রাকটিস করার জন্য কিছু প্রবলেম দিলাম। Try to solve them: uva

[524 – Prime Ring Problem](http://uva.onlinejudge.org/external/5/524.html)

[10776 – Determine The Combination](http://uva.onlinejudge.org/external/107/10776.html)

[291 – The House of Santa Claus](http://uva.onlinejudge.org/external/2/291.html)

[677 – All walks of length n from the first node](http://uva.onlinejudge.org/external/6/677.html)

Keep coding… :)

**খাতা-কলমে Extended Euclid Method**

আগেরদিন Extended Euclidean Algorithm কিভাবে কাজ করে দেখিয়েছিলাম। এবার দেখাবো এটা খাতা-কলমে কিভাবে বের করা যায়। খুব সহজ একটা ম্যাথ প্রবলেম। অ্যালগরিমটা লেখার আগে এই লেখাটা দেয়া উচিৎ ছিলো। এজন্য অনেকেই খুব ভালো বুঝতে পারে নাই এবং ডিটেইলস জানতে চেয়েছিলো আমার কাছে। যাইহোক, এখন লিখে ফেলতেছি। আগে উদাহরন হিসেবে দিয়েছিলাম **a=120** এবং**b=23. GCD(a, b) = 1.**

তাইলে আমাদের কাজ থাকছে a**x + by = 120x + 23y  = GCD(a,b)=1** সমীকরনটি থেকে **x** এবং **y** এর মান খুজে বের করা using Extended Euclid Method. ইউক্লিড দিয়ে GCD বের করার পদ্ধতিটা আগে দিলাম:

**120 = 5 \* 23 + 5**

**23 = 4 \* 5 + 3**

**5 = 1 \* 3 + 2**

**3 = 1 \* 2 + 1**

এবং এখানেই পদ্ধতিটা থামবে কারন ভাগশেষ এই ধাপেই ১ পেয়েছি।

এখন উপরের সমীকরনগুলা নিচের মত করে লিখতে পারি না কি??  
**120 – 5 \* 23 = 5 …… (1)**

**23 – 4 \* 5 = 3  …… (2)**

**5 – 1 \* 3 = 2  …… (3)**

**3 – 1 \* 2 = 1 …… (4)**

এখন (4) নং সমীকরন থেকে পাচ্ছি:

**1 \* 3 – 1 \* 2 = 1**

**=> 1 \* 3 – 1 \* ( 5 – 1 \* 3 ) = 1      [ (3) নং থেকে পাই ]**

**=> 1 \* 3 – 1 \* 5 + 1 \* 3 = 1**

**=> -1 \* 5 + 2 \* 3 = 1**

**=> – 1 \* 5 + 2 \* ( 1 \* 23 – 4 \* 5 ) = 1          [ (2) নং থেকে পাই]**

**=> -1 \* 5 + 2 \* 23 – 8 \* 5**

**=> 2 \* 23 – 9 \* 5 = 1**

**=> 2 \* 23 – 9 \* ( 1 \*  120 – 5 \* 23 ) = 1     [ (1) নং থেকে পাই]**

**=> 2 \* 23 – 9 \* 120 + 45 \* 23 = 1**

**=> -9 \* 120 + 47 \* 23 = 1 …… (5)**

আমরা (5) নং সমীকরনের সাথে **120x + 23y = 1** এর তুলনা করেই পেয়ে যাই **x** এবং **y** এর মান। অর্থাৎ, **x = -9** এবং **y = 47**.

এখানে যেহেতু GCD এর মান ১ তাই আমরা modular multiplicative inverse খুব সহজেই বের করতে পারবো। এখানে, x এর মান হবে ( a modulo b) এর modular multiplicative inverse এবং y হলো (b modulo a) এর modular multiplicative inverse.

অর্থাৎ **(120 mod 23 ) ≡ -9** এর মানে **120 এর multiplicative inverse modulo 23 হলো -9.** Multiplicative Inverse কে গণিতের ভাষায় এভাবেও লেখা হয়:

**120-1 ≡ -9 (mod 23)**

এবং **(23 mod 120) ≡ 47** একই রকমভাবে এর মানে **23 এর multiplicative inverse modulo 120 হবে 47.**সুতরাং**23-1 ≡ 47 (mod 120).**

একটু চেষ্টা এবং প্রাকটিসের জন্য একটা সমীকরন দিলাম, সমাধান করে ফেলো ফটাফট:

**701x + 322y = 1**

উত্তর সম্পর্কে কমেন্টবারে কথা হবে।  
আর এখন যেহেতু হাতে-কলমে কিভাবে Extended Euclid Method এর মাধ্যমে সমাধান করা যায় শিখেই গেছো তাইলে কোড করা এবং অ্যালগরিদম শেখা দরকার। এজন্য এই [লিংকে](http://www.abuasifkhan.me/2013/07/extended-euclidean-algorithm/) চলে যাও। Keep coding. :)

**Extended Euclidean Algorithm এবং একটুখানি Modular Multiplicative Inverse**

GCD (Greatest Common Divisor) বের করার জন্য হয়তো সবাই ইউক্লিডের পদ্ধতি ব্যবহার করেই দেখেছ। এটা আমার জানা মনে দুটি সংখ্যার GCD বের করার সব থেকে সহজ এবং efficient উপায়। যাহোক Extended Euclidean Algorithm জানার জন্য এই পদ্ধতিটাই কাজে লাগে। এখন দেখাই আগে কিভাবে GCD বের করতে হয় যদিও প্রায় সবার জানা আছে।

**Python Code:**

|  |  |
| --- | --- |
| 1  2  3  4 | def GCD(x, y):      if(y==0):          return x      return GCD(y, x%y) |

**C++ Code:**

Recursive Method:

|  |  |
| --- | --- |
| 1  2  3  4  5 | int GCD(int x, int y)  {    if (y==0) return x;    return GCD(y,x%y);  } |

**Iterative Method:**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | int GCD(int x, int y)  {    while (y != 0)    {      int temp = y;      y = x % y;      x = temp;    }    return x;  } |

এবার কাজে আসি। প্রথমে দেখা দরকার কিভাবে Extended Euclid Algorithm কাজ করে। তার আগে তোমাদেরকে বলবো Extended Euclid Method দিয়ে হাতে-কলমে কিভাবে সমীকরন সমাধান করা যায় সেটা দেখে আসতে। এজন্য এই [লিংকে](http://www.abuasifkhan.me/2013/07/extended-euclid-method/)একটু ঢু মেরে আসো।

কাজে ফিরে আসি। Extended Euclid এর মাধ্যমে সমাধান করার জন্য সমীকরনটা অবশ্যই এমন কাঠামোর হতে হবে: **ax + by = GCD(a, b)**

অর্থাৎ যদি ডান পাশে যে সংখ্যা থাকবে সেটা অবশ্যই a, b এর GCD হইতে হবে। কেবল সেক্ষেত্রেই আমরা x এবং y এর মান বের করতে পারবো Extended Euclidean Algorithm এর মাধ্যমে।

এখন একটা উদাহরন দিলে বুঝবে সুত্রটা কেমন সময় ব্যবহার করা যায়। ধর এক একটা আম এবং কলার মুল্য যথাক্রমে ১৫ এবং ৭ টাকা। একজন মহিলা ৮৫০ টাকা দিয়ে তাইলে কতটা আম এবং কলা কিনতে পারবে? এক্ষেত্রে এই সুত্রটা ব্যবহার করা যেতে পারে। অর্থাৎ 15x+7y=850. তবে এটা কেবল উদাহরন। Extended Euclid Algorithm এর জন্য ডান পাশের ধ্রুবক মানটি অবশ্যই (15, 7) এর গসাগু হইতে হবে।

প্রথমেই জানা দরকার দুটি সংখ্যার ভাগশেষ বের করার সুত্র :

**X % Y = X – floor(X/Y)\*Y**

Extended Euclid এর সুত্রটা এই form এ কাজ করে: **ri = axi + byi  ……(1)**

এখানে r হল রিমাইন্ডার। এখন ভাগশেষ বের করার সুত্রটা যদি এখানে ri বের করার জন্য ব্যবহার কর তাইলে সুত্রটা কিছুটা এমন হয়।

যদি **qi  = floor ( ri-2 / ri-1)** লিখি তাইলে সুত্রটা দাড়ায়:

এখন (1) নং সমীকরন থেকে যদি**r**i-1 এবং**r** i-2 মান উপরের সমীকরনে বসাও তাইলে পাবে:

**ri = (axi-2 + byi-2) – qi (axi-1 + byi-2)**

**=> ri = a (xi-2 – qixi-1) + b (yi-2 – qi yi-2)**

এখানে **xi=(xi-2 – qixi-1)  …… (2) এবং yi= (yi-2 – qi yi-2) ….. (3)**

Initially ri এবং r2 এভাবে পাই:

**r1 = a(1) + b(0) = a      // x=1, y=0**

**r2 = a(0) + b(1) = b     // x=0, y=1**

r1 এবং r2এর মান তো পেয়ে গেলাম এখন শুধু qi , ri , xi , yi এর মানগুলা লুপ ঘুরিয়ে কেবল আপডেট করব এবং a%b==0 হইলে অর্থাৎ r এর মান ০ হইলে লুপের কাজ শেষ করে দিব। এখন একটা উদাহরন দিলে ভাল ভাবে বুঝতে পারবে আরও। ধর, a=23 এবং b=120.

**Initial Step:**

|  |  |
| --- | --- |
| 1  2 | r1 = a(1) + b(0) = 120 x 1 + 23 x 0 = 120 // x=1, y=0       r2 = a(0) + b(1) = 120 x 0 + 23 x 1 = 23   // x=0, y=1 |

**Iterative Steps:**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | Step 1:            a=23, b=120 ,  r3 = 120 % 23 = 5 ,  q3 = 120 / 23 = 5            x3 = 0 – 5\*1 ,  y3 = 1 – 5\*0     /// From (2) and (3)    Step 2:           a=5, b=23 ,  r4 = 23 % 5 = 3 ,  q4 = 23 / 5 = 4           x4 = 1 – 4\*(-5) = 21 ,  y4 = 0 – 4\*1 = -4    Step 3:           a=3, b=5 ,  r5 = 5 % 3 = 2 ,  q5 = 5 / 3 = 1           x5 = -5 – 1\*21 = -26 ,  y5 = 1 – 1 \* (-4) = 5    Step 4:           a=2, b=3 ,  r6 = 3 % 2 = 1 ,  q6 = 3 / 2 = 1          x6 = 21 – 1\*(-26) = 47 ,  y6 = -4 – 1 \* 5 = -9 |

লুপের সমাপ্তি, কারন পরের ধাপে r এর মান 0 হবে।

উপরের উদাহরন থেকে আমরা কিছু তথ্য খুজে পেতে পারি। যেমন যদি x এর মান পজিটিভ হয় তাইলে y নেগেটিভ এবং y এর মান পজিটিভ হলে x নেগেটিভ।

আবার যদি GCD(a, b) = 1 হয় অর্থাৎ যদি a, b co-prime হয়ে থাকে তাইলে x হবে  (a modulo b) এর Modular Multiplicative Inverse, এবং y হবে (b modulo a) এর Modular Multiplicative Inverse.

অর্থাৎ উপরের উদাহরনে -9 হল 120 modulo 23 এর multiplicative inverse এবং 47 হবে 23 modulo 120 এর multiplicative inverse.

Extended Euclid ফাংশনের কোড:

**Iterative Method:**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37 | int Extended\_Euclid(int A, int B, int \*X, int \*Y)  {      int x, y, u, v, m, n, a, b, q, r;      ///\* B = A(0) + B(1) \*/      x = 0;  /// x [i-2]      y = 1;  /// y [i-2]      ///\* A = A(1) + B(0) \*/      u = 1;  /// x[i-1]      v = 0;  /// y[i-1]      a = A;      b = B;        while ( a != 0 )      {          ///\* b = aq + r and 0 <= r < a \*/          q = b / a;            /// GCD function          r = b % a;          b = a;          a = r;           /// /\* r = A(x - uq) + B(y - vq) \*/ ///          m = x - (u \* q); /// m = x[i] = (x - uq)          n = y - (v \* q);  /// n = y[i] = (y - vq)          x = u;  /// updating x[i-1] = x[i-2]          y = v;  /// updating y[i-1] = y[i-2]          u = m;  /// updating x[i] = x[i-1]          v = n;  /// updating y[i] = y[i-1]      }        ///\* Ax + By = gcd(A, B) \*/      \*X = x;      \*Y = y;        return b;  } |

**Recursive Method:**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | int extendedEulid(int a, int b)  {      if(b==0)      {          x=1; y=0; d=a;  /// some extensions          return a;      }        int ret = extendedEulid(b, a%b);   /// GCD function      int x1 = y;         /// some extensions      int y1 = x - (a/b) \*y;      x = x1;      y = y1;        return ret;  } |

**Python Code:**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | def ExtendedEuclid(a, b):      x, xi = 0, 1      y, yi = 1, 0        while b>0:          q = int(a / b)          a, b = b, a % b          x, xi = xi - q\*x, x          y, yi = yi - q\*y, y        return (xi, yi, a) |

যথাসম্ভব ব্যাখ্যা করে লেখার চেষ্টা করেছি। ভুলত্রুটি থাকাটা স্বাভাবিক। সংশোধন অথবা কোন সমস্যা থাকলে কমেন্টে জানাও এবং keep coding… :)

**Chinese Remainder Theorem**

[**Chinese Remainder Theorem**](http://en.wikipedia.org/wiki/Chinese_remainder_theorem) একটা খুবই interesting theorem. প্রথমে বলি এটা কোন কোন প্রবলেমগুলার সাথে deal করতে পারে। এটা জানলেই বুঝবে কেন এটাকে এতটা interesting বলতেছি। আগেই বলে রাখি, এই থিওরেমটি শিখতে এবং প্রোগ্রামে কোড করতে হলে অবশ্যই [**Modular Multiplicative Inverse**](http://en.wikipedia.org/wiki/Modular_multiplicative_inverse) কিভাবে Extended Euclid Method এর সাহায্যে বের করতে হবে সেটা জেনে রাখা দরকার( এজন্য এই [লিংকে](http://www.abuasifkhan.me/2013/07/extended-euclidean-algorithm/)চলে যাও)। যদিও খাতা-কলমে কিভাবে Modular Multiplicative Inverse বের করা যায় সেটা শেখাবো এখানে। তারপরও কোড করার জন্য ইউক্লিডের মেথডটা শিখে রাখার জন্য বলবো সবাইকে।

এখন আমরা জানি যে: **5 (mod 8)** **≡ 5** আবার **13 (mod 8) ≡ 5, 21 (mod 8) ≡ 5**. এখন যদি তোমাকে একটা শর্ত দিয়ে দেয়া হয় যে **z (mod 3)  ≡ 2** হতে হবে। এখানে **z = {5, 13, 21}**, তাইলে শর্তটা দেখা যাচ্ছে মোটামুটি **5** এর জন্য সত্য তবে **13** এবং **21** এর জন্য সত্য হচ্ছে না। অর্থাৎ এখানে একমাত্র **5** ই সঠিক মান যেটা সকল শর্ত পুরন করতেছে। এই ধরনের সমস্যার সমাধান Chinese Reminder Theorem দিতে পারে। অর্থাৎ একাধিক modular condition থাকবে যেটা থেকে এমন সব সঠিক মান বের করতে হবে যা সকল দেয়া শর্ত মেনে চলবে। অবশ্য এক্ষেত্রে উত্তর অসীম সংখ্যক হইতে পারে। সেগুলাও বের করার উপায়ও এই থিওরেমটি দিয়ে দেয়। :)

এখন কাজে আসি, একটা উদাহরন সমাধান করার মাধ্যমে থিওরেমটা শেখার চেষ্টা করি। কিছু শর্ত আছে ধরে নিলাম: **Z = 4 (mod 5)**

**Z = 6 (mod 7)**

**Z = 3 (mod 11)**

এই শর্তগুলা সিদ্ধ করে এমন সব **Z** এর মান আমাদের বের করতে হবে। তো প্রথমে বলে রাখি এখানে রিমাইন্ডারগুলা হবে **b**i এর মান। অর্থাৎ, **b = {5, 7, 11}** এবং **ci**হবে প্রাপ্তমানগুলা অর্থাৎ **c = {4, 6, 3}**. প্রথমে যেটা করতে হবে সেটা হলো **B (big B)** এর মান বের করা। এটার সুত্র হলো:

যার মানে হলো সকল **bi**এর গুণফলগুলা হলো**B (big B)** এর মান। এক্ষেত্রে,

**B = 5x7x11 = 385**

এবার আমাদের কাজ হলো **B**iএর মান বের করা। এটার সূত্র হলো:

**Bi = B ÷ bi**

অর্থাৎ,

**B1 = 385/5 = 77**

**B2 = 385/7 = 55**

**B3 = 385/11 = 35**

চায়নিজ রিমাইন্ডার থিওরেম থেকে Z এর মান বের করার সূত্রটা হলো:

**Z = B1X1c1+ B2X2c2+ B3X3c3+ … + BnXncn**

এই উদাহরনটার ক্ষেত্রে:

**Z = B1X1c1+ B2X2c2+ B3X3c3**

এখানে আমাদের **Bi**এবং **c**i এর মান আগে থেকেই জানা। তবে এখানে**X**iটা আবার কি জিনিস?? হুম, এই কাজেই আমাদের লাগবে Extended Euclid. এটা শেখার জন্য [লিংকে](http://www.abuasifkhan.me/2013/07/extended-euclidean-algorithm/) যাও (যদিও লেখার শুরুতে একবার দিয়েছি লিংকটা)। এখানে আমরা **Bi**এবং **b**i এর modular multiplicative inverse বের করবো। এ দুইটা মানের উপর Extended Euclid চালালে আমরা যে **X** এর মানটা পাই সেটাই এখানে **Xi**এর মান।

এখন

**B1X1 ≡ 1 (mod b1)**

**=> 77 X1 ≡ 1 (mod 5)**

**=> (77-80)**X1 **≡ 1 (mod 5)                   [ 4 এর গুণিতক দ্বারা 5 কে বিয়োগ করে ]**

**=> (-3) X1 ≡ 1 (mod 5)**

**=> (-3) X1≡ 6 (mod 5)                                    [ 1 (mod 5) ≡ 6 (mod 5) ]**

**সুতরাং, X1 = -2**

**আবার,**

**B2 X2 ≡ 1 (mod b2)**

**=> 55 X2 ≡ 1(mod 7)**

**=> (55-56) X**2 **≡ 1 (mod 7)                               [ 55 থেকে 7 এর গুণিতক বিয়োগ করে ]**

**=> (-1) X2 ≡ 1 (mod 7)**

**সুতরাং, X2 = -1**

**এভাবেই, 35 X3 ≡ 1 (mod 11) থেকে পাই, X3 = -5. X এর মানগুলার অনেক হতে পারে, তবে Extended Euclid Method ব্যবহার করলে এই মানগুলাই পাওয়া যায়।**

এখন Chinese Reminder Theorem এর আসল সূত্রটাতে আসি:

**Z = B1X1c1+ B2X2c2+ B3X3c3**

**=> Z = 77x(-2)x4 + 55x(-1)x6 + 35x(-5)x11 = -1471**

যে মানটা পাইলাম সেটা দিয়ে দেয়া শর্তগুলার সবকয়টি সিদ্ধ হবে। সুতরাং এটা একটা উত্তর। তবে আমি আগেই বলেছি অসীম সংখ্যক উত্তর থাকবে এই সমস্যাটার জন্য। তাইলে আমরা সেগুলা কিভাবে বের করবো? খুবই simple, প্রাপ্ত **B (big B)** এর মানের যেকোনো গুণিতক দিয়ে **Z** এর প্রাপ্ত মানের সাথে যোগ অথবা বিয়োগ দিলেই হয়ে গেলো। অর্থাৎ **( 4×385 – (-1471) ) = 69**, এটাও একটি সঠিক মান। এভাবে তুমি যেকোনো লিমিটের জন্য একটা সাম্ভাব্য মান খুজে পেতে পারবে। এখন নিচের উদাহরনটি সমাধান করার ট্রাই করো:

**Z = 3 (mod 8)**

**Z = 1 (mod 9)**

**Z = 4 (mod 11)**

**Lightweight Set of Boolean ওরফে Bitmask**

তোমাদের জানা আছে কম্পিউটারে integer কিছু বিটের সমন্বয়ে স্টোর হয়ে থাকে। আমরা যেমন Boolean অ্যারে ব্যবহার করি বিভিন্ন কাজে, তেমন আমরা একটা integer এর বিটগুলা নিয়ে এই একই কাজ করতে পারি। এটাকে এজন্য বলা হয় Lightweight small Set of Boolean. তবে যখন অ্যারে সাইজ কম থাকে অর্থাৎ ১৬ অথবা ৩২ এর মধ্যে থাকে তখনই আমরা bitwise operation এর সাহায্যে অনেক সহজে এবং efficient way তে integer কে বুলিয়ান অ্যারের মত ব্যবহার করতে পারি । আমরা আজ কিভাবে এই বিটগুলার তথ্য ব্যবহার করে বিভিন্ন মজার মজার অপারেশন করা যায় তা দেখবো। প্রথমে তোমাদের মনে করিয়ে দিলাম যদিও জানা আছে সবার, বুলিয়ান অপারেশনের জন্য “&” মানে গুণ করা, এবং “|” এর অর্থ যোগ করা। (আগেই বলে রাখা উচিৎ এই টিউটোরিয়ালটি পড়ার আগে বুলিয়ান অপারেশনগুলা অর্থাৎ AND, OR, XOR, NOR, এবং bit shifting সম্পর্কে একটু জেনে রাখলে বুঝতে সুবিধা হবে)

[code]1. প্রথমে দেখি bitmask দিয়ে কিভাবে কোন সংখ্যাকে ২ দিয়ে গুণ/ভাগ করা যায়।  
S = 34 (base 10) = 100010 (base 2)  
S = S<<1 = S\*2 = 68 (base 10) = 1000100 (base 2)  
S = S>>2 = S/4 = 17 (base 10) = 10001 (base 2)  
S = S>>1 = S/2 =  8 (base 10) = 1000 (base 2) //লক্ষ্য কর শেষের 1 টা নাই আর[/code]

2. এখন দেখি কিভাবে j-তম বিট অন করা যায়। অর্থাৎ j-তম বিটে 1 যোগ করলেই সেটা অন হয়ে যাবে। এজন্য আমাদের ব্যবহার করতে হবে S |= ( 1<< j) এটা কিভাবে কাজ করে সেটা নিচের উদাহরনটা দেখে বোঝ কিনা দেখ তো।

|  |  |
| --- | --- |
| 1  2  3  4  5 | S = 34 (base 10) = 100010 (base 2)      j = 3, (1<<j)    = 001000 (base 2)   // j এর মান 1 ছিল। left shifting করে আমরা j এর                                           //প্রথম বিটটা ৩ ঘর বামে সরিয়ে দিয়েছি।                     ----------------   ( OR operation অর্থাৎ বিটে ১ যোগ হচ্ছে  )      S = 42 (base 10) = 101010 (base 2)   // S এর মান আপডেট হয়েছে। |

3. এখন যদি চাও j-তম বিটের তথ্য জানতে অর্থাৎ j-তম বিট 1/0 সেটা জানতে চাই। এজন্য আমরা অবশ্যই j ভেরিয়েবলটার j-তম বিটের মান 1 করে নিব। তারপর S ভেরিয়েবলটার সাথে “&” operation করবো। কেন? কারনটা নিচের উদাহরনটা দেখলেই বুঝতে পারবে আশা করি।

|  |  |
| --- | --- |
| 1  2  3  4 | S = 42 (base 10) =  101010 (base 2)      j = 3, (1<<j)    =  001000 // ৩টা বিট শিফট করলাম।                     ---------------- ( AND operation )      T = 8 (base 10)  =  001000 // অর্থাৎ S এর ৩ নং বিটে 1 আছে। |

আরও একটা উদাহরন দিলাম:

|  |  |
| --- | --- |
| 1  2  3  4 | S = 42 (base 10)  =  101010 (base 2)      j = 2, (1<<j)     =  000100 // 1 কে ২ বিট শিফট করলাম                     ---------------- ( AND operation )      T = 0 (base 10)   =  000000 (base 2) // অর্থাৎ j-তম বিটে 0 আছে। |

4. এবার দেখি কিভাবে j-তম বিটের মান clear/turn off করতে হয়। এজন্য আমরা ব্যবহার করি, S &= ~(1<<j). bitwise NOT এর কাজ হল বিটে যে মানটি আছে তার বিপরীত করে দেয়া। তাইলে এক্ষেত্রে j ভেরিয়েবলটির j-তম পজিশনের মান 0 হবে এবং বাকি সকল বিটের মান 1 হবে। এখন যদি “&” অপারেশন চালাই তাইলে S এর j-তম বিট সর্বদা 0 হয়ে যাবে। উদাহরনটা দেখলেই বুঝবে:

|  |  |
| --- | --- |
| 1  2  3  4 | S = 42 (base 10) =  101010 (base 2)       j = 1, ~(1<<j)   =  111101 // এটাই bitwise “NOT” operation.                     ----------------.  ( AND operation )       S = 40 (base 10) =  101000  // S এর j-তম বিটের মান আপডেট হয়েছে |

5. এখন যদি আমরা j-তম বিটের মান toggle অর্থাৎ বিপরীতটা করতে চাইলে কি করবো? এজন্য bitwise XOR operator ব্যবহার করবো। কারন XOR এর মান তখনই 1 যখন দুইটা বিটের মান ভিন্ন হবে। একাজের জন্য আমরা ব্যবহার করবো S ^= (1<<j)

|  |  |
| --- | --- |
| 1  2  3  4 | S = 40 (base 10)  =  101000 (base 2)      j = 2, (1<<j)     =  000100 (base 2)                    ---------------- (XOR operation. True if both bits are different)      S = 44 (base 10)  =  101100 (base 2) // S এর j-তম বিটের মান আপডেট হয়েছে। |

আপাতত এপর্যন্তই। কিছু বুঝতে সমস্যা হইলে কমেন্টে জানাবে অবশ্যই।

**সি++ দিয়ে বিভিন্নরকম conversions**

**প্রোগ্রামিং করার সময় প্রায়ই অনেকরকম কনভার্সনের দরকার পড়ে। সি++ এ অনেকভাবেই এবং অনেক সহজে এসব কনভার্সন করা যায়। যদিও আমার personal favorite stringstream. কিছু উদাহরন দিচ্ছি নিচে, আশা করি ব্যাখ্যাও করা লাগবে না। :)**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30 | ///int to string          stringstream ss;          int a = 1234;          ss<<a;  /// stream in          string s1;          ss>>s1; /// stream out          cout<<s1<<endl;            ///string to int          ss.clear();          string s = "1234"          ss<<s;          int b;          ss>>b;          cout<<b<<endl;            /// char\* to string          ss.clear();          char somechars[10] = "ruet cse";          ss<<somechars;          string somestrings;          ss>>somestrings;          cout<<somestring<<endl;            /// string to char\*          ss.clear();          somestrings = "ruet cse";          ss<<somestrings;          ss>>somechars;          cout<<somechars<<endl; |

আশা করি বুঝতে অসুবিধা হয় নাই। যদি কোনরকম সমস্যা অথবা ভুল থাকে তাইলে কমেন্ট করে জানাবে অবশ্যই।

**Vector STL এ ইনপুট নেয়ার পদ্ধতি**

2D ভেক্টরের জন্য ডিক্লেয়ারেশন একটু ভিন্ন রকম।

|  |  |
| --- | --- |
| 1 | vector<int>array[30]; |

মাথায় একটা চিন্তা আগে আসতে পারে এই 2D অ্যারেতে তো একটা ডাইমেনশন ৩০টি এলিমেন্ট দিয়ে সিমাবদ্ধ করে দেয়া হয়েছে। কিন্তু ভেক্টর তো resizable array. তাইলে কিভাবে লিখতে পারি?

|  |  |
| --- | --- |
| 1 | vector< vector<int> > array; |

একটু সাবধান হওয়া উচিৎ এমনভাবে ডিক্লেয়ার করার সময়। যদি এমন লিখে ফেল “>>” তাইলে কম্পাইলার (সব না) এটাকে রাইট শিফ্ট ধরে নিতে পারে। এজন্য মাঝে একটা ছোট্ট স্পেস দিয়ে রাখা ভাল। অর্থাৎ “> >”. এখন ইনপুট কিভাবে নিব? প্রথমটার জন্য:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | vector<int>array[30];    for(int i=0; i<25; i++)  {      for(int j=0; j<20; j++)      {          int a;          cin>>a;          array[i].push\_back(a);      }  } |

এটা ২৫X২০ সাইজের একটা 2D অ্যারের কাজ করবে। এবার দ্বিতীয়টার জন্য:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | vector<vector<int> >array;    for(int i=0; i<25; i++)  {      vector<int>temp;      for(int j=0; j<20; j++)      {          int a;          cin>>a;          temp.push\_back(a);      }      array.push\_back(temp);  } |

এটাও ২০X২৫ সাইজের একটা অ্যারের কাজ করবে। তবে row এর সাইজ যত ইচ্ছা তত বাড়ানো বা কমানো যাবে। 3D অ্যারের কাজ মনে হয় বোঝানোর দরকার পড়বে না আর। বুঝতে ক‌োনরকম সমস্যা হইলে কমেন্ট করে জানাতে ভুলবে না।

**সি++ এর PRIORITY QUEUE**

**priority queue**হলো সি++ একটা STL যা প্রোগ্রামে ইনপুটগুলাকে সর্টেড অবস্থায় রেখে দেয়। অর্থাৎ এটা এক প্রকার অ্যারে, আরো ভাল ভাবে বলা যায় ভেক্টর যার উপাদানগুলা সর্টেড অবস্থায় থাকে। ডিফল্ট অবস্থায় এটা হাইয়ার টু লোয়ার প্রেসিডেন্সে (MAX priority queue)উপাদানগুলা সর্ট করে রাখে। তবে কাজের ভিন্যতার জন্য লোয়ার টু হাইয়ার (MIN priority queue) প্রেসিডেন্সেও সর্ট করা যায়। তবে এটা ডিফল্ট ভাবে করা থাকে না।

**MAX Priority Queue:**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | #include <iostream>  #include <queue>  using namespace std;  main()  {      priority\_queue<int> pq;      pq.push(1);      pq.push(16);      pq.push(6);      pq.push(13);      while ( !pq.empty() )      {          cout << pq.top() << endl;          pq.pop();      }  } |

এই কোডটা রান করলে আউটপুট হয়:

|  |  |
| --- | --- |
| 1  2  3  4 | 16  13  6  1 |

**MIN Priority Queue:**

মিনিমাম প্রায়োরিটি কিউ কাজ করাতে গেলে একটু কাজ নিজেদের করে নিতে হয়। অর্থাৎ কম্পেয়ারিং। একটা ক্লাস অথবা স্ট্রাকচার তৈরী করতে হয় যা উপাদানগুলাকে মিনিমাম প্রায়োরিটি অনুযায়ী সাজায়ে রাখে। তেমন কঠিন কিছুনা। জাস্ট এতটুকু লিখতে হয় এক্সট্রা:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | struct compare  {    bool operator()(const int& l, const int& r)    {        return l > r;    }  }; |

এবং priority\_queue টাইপ ভেরিয়েবল ডিক্লেয়ার করার সময় লিখতে হয়:

|  |  |
| --- | --- |
| 1 | priority\_queue<int, vector<int>, compare>pq; |

অর্থাৎ কোডটা হবে:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25 | #include <iostream>  #include <queue>  using namespace std;    struct compare  {    bool operator()(const int& l, const int& r)    {        return l > r;    }  };  main ()  {      priority\_queue<int,vector<int>, compare > pq;        pq.push(1);      pq.push(16);      pq.push(6);      pq.push(13);      while ( !pq.empty() )      {          cout << pq.top() << endl;          pq.pop();      }  } |

এই কোডটার আউটপুট হবে:

|  |  |
| --- | --- |
| 1  2  3  4 | 1  6  13  16 |

**Priority Queue on Custom Class:**

প্রায়োরিটি কিউ যদি আমরা কাস্টম ক্লাস অথবা স্ট্রাকচারের সাথে ইমপ্লিমেন্ট করতে চাই তাইলে আরো একটু বেশী কষ্ট করতে হয়। তবে কাজটা অনেক মজার। একটু দেখলেই বুঝতে পারবে।

আগে কোডটা একটু দেখে নাও:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38 | #include <iostream>  #include <queue>    using namespace std;    struct hambaa{        public:      string apprnce;      int age;      hambaa( string c, int a)      {          apprnce = c;          age = a;      }  };  struct compare  {    bool operator()(const hambaa& abc, const hambaa& def)    {        return abc.age > def.age;    }  };    main ()  {      priority\_queue < hambaa , vector<hambaa> , compare > pq;        pq.push(hambaa ("bachhur",1)) ;      pq.push( hambaa ("boro goru",16 ));      pq.push( hambaa ("majhari goru", 6));      pq.push( hambaa ("majhari boro goru", 10));      while ( !pq.empty() )      {          cout << pq.top().apprnce<< " " << pq.top().age << endl;          pq.pop();      }  } |

আউটপুট হবে:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | bacchur 1    majhari goru 6    majhari boro goru 10    boro goru 16 |

এখানে একটা কাস্টম স্টাকচার (hambaa) তৈরী করা হয়েছে। যেখানে ২টা উপাদান আছে। আমরা চাই age অনুযায়ী সর্ট করতে। তাই আমরা compareস্ট্রাকচারে hambaa টাইপের abc এবং def নিয়েছি। এবং তা থেকে age অনুযায়ী রিটার্ন করেছি।  আর খেয়াল করলে দেখতে পারবে main() function এ যা করেছি সব আগের মতই।  
আরও একটা কাজ করতে পারো। Operator Overloading করেও কাজটি অনেক সহজ হয়ে যায়। তাইলে তোমাকে আর এত ঝামেলা করে এইটুকু লেখা লাগবে না।

|  |  |
| --- | --- |
| 1 | priority\_queue < hambaa , vector<hambaa> , compare > pq; |

নিচের কোডে Operator Overloading করে কাজটা করেছি। কোডটা কতটা দেখতে সুন্দর লাগছে দেখতে পারো এখন।

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27 | #include <iostream>  #include <queue>    using namespace std;    struct hambaa  {  string apprnce;  int age;  hambaa(string s, int nmbr){apprnce=s; age=nmbr;}  bool operator < ( const hambaa& p) const { return p.age < age; }  };    main ()  {      priority\_queue <hambaa> pq;      pq.push(hambaa("bachhur",1));      pq.push( hambaa("boro goru",16 ));      pq.push( hambaa("majhari goru", 6));      pq.push(hambaa("majhari boro goru", 10));        while ( !pq.empty() )      {          cout << pq.top().apprnce<< " " << pq.top().age << endl;          pq.pop();      }  } |

এভাবে priority\_queue নিয়ে অনেক মজার মজার কাজ করতে পারবে, সর্টেড লিস্টের জন্য। কিপ কোডিং :)

* **শাফায়েত**

ব্লগটা আজকেই চোখে পড়লো, চমৎকার :)।সরাসরি স্ট্রাকচারের ভিতর অপারেটর ওভারলোডিং করলে লিখতে হয় অনেক কম। যেমন:struct node{ int n,w; bool operator < ( const node& p ) const { return w < p.w; }};ডিক্লেয়ারের সময় শুধু: priority\_queueq;

**Abu Asif Khan Chowdhury**

খুব ভালো একটা প্রশ্ন। প্রথম প্রশ্ন যে, আমরা vector কেন নিচ্ছি। এর জন্য আগে আমি[http://www.cplusplus.com/refer...](http://www.cplusplus.com/reference/queue/priority_queue/) যেতে বলব। এখানে template library এর definition এ বলা আছে:template < class T, class Container = vector<T>, class Compare = less<typename Container::value\_type> > class priority\_queue;অর্থাৎ টেমপ্লেটে ৩টা প্যারামিটার আছে। প্রথম ওই কাস্টম ক্লাসটার নাম, দ্বিতীয় কাস্টম ক্লাসের একটা রিসাইজেবল অ্যারে/vector এবং compare class. যেহেতু আমরা কাস্টম ক্লাসটার অথবা ডেটা টাইপটার জন্য comparison operators overload করি নাই এজন্য আমরা একটা vector এর মধ্যে সব ডেটা রেখে দিচ্ছি। পরে compare class/function এর মাধ্যমে প্রাইওরিটি কিউ তথা হিপে সর্টেট অবস্থায় রাখছি। তোমার প্রশ্ন হতে পারে int, float, char ইত্যাদির ক্ষেত্রে তো আমরা লিখি, priority\_queue <int/float/char> pq. এখানে তো vector নেয়া লাগছে না। তার কারন হলো এসব ডিফল্ট ডেটা টাইপগুলার comparison class C/C++ আগে থেকে overload হয়ে আছে। শেষ উদাহরনে যখন আমরা comparison operator overload করে নিয়েছি তখন আর vector নেওয়া লাগে নাই।তোমার দ্বিতীয় প্রশ্নটার উত্তরও একই সাথে দেয়া হয়ে গেছে। একটু দেখলেই ভুলটা বুঝতে পারবে। আশা করি বোঝাতে পেরেছি। সমস্যা থাকলে জানাবে।( দু:খিত ব্লগটা এখনও HTML friendly করতে পারি নাই। এজন্য কমেন্ট করতে সমস্যা হচ্ছে। খুব শিঘ্রই ঠিক করার চেষ্টা করছি )

**ইমতিয়াজ**

priority\_queue[int, vector[int], compare]pq;[] represents <>এখানে vector দিয়ে কী কাজ হচ্ছে?ভাইয়া এটা এভাবে (<http://ideone.com/K92y2p)> না করে এভাবে করতে চাচ্ছি ([http://ideone.com/mYutDV)যেভাব...](http://ideone.com/mYutDV)%E0%A6%AF%E0%A7%87%E0%A6%AD%E0%A6%BE%E0%A6%AC%E0%A7%87) আমি নরমাল সর্টিং (<http://ideone.com/qKu5am)> করে থাকি।

**Abu Asif Khan Chowdhury**

ধন্যবাদ, ভাইয়া, এটা জানতাম না। এভাবে ওভারলোডিং করলে ডিক্লেয়ারেশন সত্যিই অনেক সহজ হয়ে যায়। ওয়াও! :)