Greedy Problem নিয়ে এইটা আমার দ্বিতীয় ব্লগ । কেউ প্রথমটা দেখতে চাইলে এই লিঙ্ক পাওয়া যাবে । এইখানে ইচ্ছা আছে কিছু ভাল greedy problem solution নিয়ে কথা বলার ।

The Double HeLiX :::

এই প্রবলেম এ আমাদের দুইটা নাম্বারের সিকুয়েন্স দেওয়া থাকবে । এই সিকুয়েন্স এর মধ্যে কিছু নাম্বার ইন্টারসেকশন থাকতে পারে । আমরা চাইলে যেকোন একটা সিকুয়েন্স ধরে যাইতে পারব । যে যে নাম্বার এর উপর দিয়ে যাব তা এড করতে থাকব । এখন যখনই আমরা কোন ইন্টারসেকশন নাম্বারে আসব । আমরা চাইলে সিকুয়েন্স চ্যাঞ্জ করে অন্য সিকুয়েন্স এর যেখানে ইন্টারসেকশন নাম্বারট আছে ওতে চলে যাইতে পারব , আবার চাইলে নাও যেতে পারি । আমাদের বলতে হবে আমরা হাইস্ট কত এইভাবে পেতে পারি ।

এইখানে সিকুয়েন্স এর লিমিট বলা হইছে <= ১০০০০ । আমরা এখন যদি চাই কোন কোন পয়েন্ট এ তারা ইন্টারসেকশন করছে তাহলে Naive process এ দুইটা ফর লুপ চালাইয়া পেয়ে যাব । রান টাইম O(n * m) (যেখানে 'n' একটা সিকুয়েন্স এর লেন্থ আর 'm' অন্য একটা সিকুয়েন্স এর লেন্থ) । যদি A[] , B[] দুইটা সিকুয়েন্স নাম্বার হয় তাহলে A[n-1] & B[m-1] ইন্টারসেকশন করুক আর না করুক আমরা ধরে নিব এরা ইন্টারসেক্ট করছে । এখন কিছু Observation থেকে আমরা Greedy solution টা develop করতে পারি ।

1. প্রথম এ লক্ষ্য করি কেন এইটা greedy process এ solve হবে । ধরি array A[] এর ইনডেক্স 'i' এবং array B[] এর ইনডেক্স 'j' intersect করছে । তাহলে অবশ্যই যদি A[0] - A[i] এর sum value B[0] - B[j] এর sum value থেকে বড় হয় তাহলে আমাদের সব সময়ই A সিকুয়েন্স থেকে যাত্রা করা লাভ যখন হবে । এবং এইভাবে যদি দেখি আমাদের প্রত্যকটা ইন্টারসেকশন independently এই ভাবে কাজ করতে পারে ।
২। এইখানে সব সময়ই এখন যেখানে আসি (মানে ইন্টারসেকশন ইনডেক্স , যদি কোন ইন্টারসেকশন নাও থেকে থাকে আমরা n-1 , m-1 এর একটা dummy intersection করেছি বলে কম্পেয়ার করতে পারব ।
৩। রান টাইম কি হবে ? আমরা এইখানে যদি A[] array এর ইনডেক্স ফিক্সকড করে ক্যালকুলেশন করি তাহলে m টা ইন্টারসেকশন পেতে পারি , যেখানে আমরা লিনিয়ার ভাবে মারজ করে ভ্যালু চেক করছি । মানে এই চ্যাকিং এবং কম্পায়ের কাজে আমাদের O(n+m) টাইম লাগবে । আমাদের ইন্টারসেকশন পয়েন্ট গুলা বের এর জন্য লাগছে O(n*m) . অর্থাৎ আমরা O(n*m) (কোন কোডে যত অপারেশন হয় এর হাইস্ট যেটাতে টাইম লাগে ঐটাই কোন কোড এর রান টাইম) greedy solution develop করতে পারি ।

কোড

```
const int MX = 1e5 + 100 ; // limit
Long a[MX] , b[MX]  ; // two sequence
int n , m , x[MX] , y[MX] ; // intersection point
typedef long long int Long ;
void solve()
{
       scanf("%d",&n);
       for ( int i = 0 ; i < n ; i++ ) scanf("%d",&a[i]);
       scanf("%d",&m);
       for( int i = 0 ; i < m ; i++ ) scanf("%d",&b[i]);
       int cs = 0 , i = 0 , j = 0 ;
       for ( i = 0 ; i < n ; i++ )
       {
           for ( j = 0 ; j < m ; j++ )
           {
               if( a[i] == b[j] )
               {
```

```
                    x[cs] = i ;
                    y[cs] = j ;
                    cs++;
                    break ;  // ব্রেক করা অনেক দরকারি
                }
            }
        }
        x[cs] = n - 1 ;
        y[cs] = m - 1 ;
        Long ans = 0 ;
        i = 0 , j = 0 ;
        for ( int k = 0 ; k <= cs ; k++ )
        {
            Long sum1 = 0 , sum2 = 0 ;
            for ( ; i <= x[k] ; i++ ) sum1 += a[i];
            for ( ; j <= y[k] ; j++ ) sum2 += b[j];
            ans += max(sum1,sum2); // compare the value , always chose the best value
        }
    printf("%I64d\n",ans);
}
```

view raw **Greedy1.cpp** hosted with ❤ by **GitHub**

Expedition :

এইটা খুব সুন্দর একটা প্রবলেম । priority_queue use কেন অনেক প্রবলেম solve সহজ করে দেয় , তা বুঝা যায় এই প্রবলেমটা করলে । এইখানে বলা হইছে গরু গাড়ি চালাইতে পারে :P ওরা একটা ট্রাক দখল করছে , এই ট্রাক এ করে নিকটবর্তী শহরে যাবে । যার দূরত্ব দেওয়া আছে । কিন্তু যেহেতু তারা ভাল মত গাড়ি চালাতে পারে না তারা গাড়ির ফুয়েল লিক করে ফেলছে , এখন কারেন্ট ফুয়েল দিলে ১ unit যাওয়া যায় । শহর আছে ট্রাকের অবস্থা থেকে d unit দূরে , এবং তাদের গাড়িতে ফুয়েল আছে f unit . এখন আরোও কিছু রিফ্রাইনিং স্টেশনের ইনফরমেশন দেওয়া আছে , এইটা কত দূরে (শহর থেকে গাড়ি থেকে নয় , আমাদের এদের দূরত্ব গাড়ি থেকে প্রথমে বের করে নিতে হবে) এবং কতটুকু ফুয়েল আছে (গরুতে মেলা টাকা পয়সা :p তারা কতটা ফুয়েল করবে এইটা ব্যাপার না) । এই প্রবলেম এ এই ইনফরমেশন গুলা নিয়ে বলা হইছে মিনিমাম কতটা ফুয়েল স্টেশনে থেমে গরুর দল নিকটবর্তী শহরে যাবে বা আদ্যও যাবে কিনা ।

- আচ্ছা স্বাভাবিক ভাবেই আমরা ফাস্ট এ যে যে স্টেশনে আছে তাদের দূরত্ব দিয়ে সর্ট করে নিব । ক্যালকুলেশন এর সুবিধার জন্য আমরা শহরকে এড দিব যেন দেখতে পারি শহরে পৌঁছাতে পারছি কিনা ।
- এইখানে অবশ্যই অবশ্যই যে যে স্টেশনে আমাদের কাছে এখন যা ফুয়েল আছে তা দিয়ে যাওয়া যায় তাদের মধ্যে থেকে ম্যাক্সিমাম যে যে স্টেশনে আছে তাদের থেকে ফুয়েল নিব (যা না নিলেই নয়) । এই কাজটা আমাদের priority_queue অনেক সহজে করে দেয় ।
- আমরা যদি কোন স্টেশনে না যেতে পারি মানে হইল আমাদের পক্ষে শহরে যাওয়া পসিবল না ।

কোড :

```
const int NX = 1e6 + 10 ;
priority_queue < int > pq ; // as input is huge it better not to declear it local
```

```
pii inp[NX];
int d , f , n ;
void solve()
{
    int cs , t = II ;
    for ( cs = 1 ; cs <= t ; cs++ )
    {
       n = II ;
       rep( i , n )
       {
           int x = II , y = II ;
           inp[i] = mp( x , y );
       }
       d = II , f = II ;
       rep ( i , n )
       {
           inp[i].ff =  d - inp[i].ff ; // adjust distance from truck
       }
       inp[n++] = mp( d , 0 );
       sort ( inp , inp + n ); // distance wise sort
       int i , ans  = 0 ;
       while( !pq.empty() ) pq.pop();
       for ( i = 0 ; i < n ; i++ )
       {
           if( inp[i].ff > f )
           {
               while( !pq.empty() && f < inp[i].ff )
               {
                   f += pq.top();
                   pq.pop();
                   ans++;
               }
               if( f < inp[i].ff ) break ; // we are not able to make it so break
           }
           pq.push( inp[i].ss );
       }
       printf("%d\n", i == n ? ans : -1 );


    }


}
```

view rawgreedy2.cpp hosted with ❤ by **GitHub**

এই কোডের রান টাইম হবে O(nlg(n)) .

GERGOVIA :

এই প্রবলেমটা অনেক সহজ একটা প্রবলেম , একই প্রবলেম আমি Uva তেও দেখছি । এমন কি codechef , hacckerrank এর কনটেস্ট এও আসতে দেখছি । তাই এইখানে include করলাম ।

খুবই simple idea এর প্রবলেম । solution টাই হইল 'Think simple" । এই প্রবলেম এ একটা city এর সামগ্রিক অবস্থা তুলে ধরা হইছে । এইখানে একজন salesman বিভিন্ন বাড়িতে ওয়াইন দেয় বা কিনে , যদি ইনপুট '-' হয় তাহলে কিনবে যদি '+' হয় তাহলে কিনে ঐখানে বিক্রি করবেন । এক বাড়ি থেকে অন্য বাড়ি যতটা ওয়াইন সরবরাহ করতে তার মিনিমাম কত unit কাজ করতে হবে ।

এই প্রবলেম এ বলাই আছে চাহিদা এবং যোগান সমান থাকবে । এইটাই আমাদের ক্লু । একটা সাম ভ্যারিএবল নিব । এখন প্রথম বাড়িথেকে শেষ বাড়িতে ক্রমানয়ে যাব সাথে সাথে তাদের যা লাগবে (কেউ কিনবে বা কেউ বিক্রি করবে টা যোগ করতে থাকব) সাথে সাথে আমরা আমাদের ans current sum value এর absolute value add করতে থাকব । এইটাই আমাদের আন্সার হবে ।

আমরা চাইলে এইটা প্রুভ করতে পারি কেন কাজ করে । ধরে নেই শুধু মাত্র দুইটা বাড়ি আছে তাহলে কি হবে 5 -5 তাহলে এক বাড়ি থেকে অন্যবাড়িতে শুধু 5 unit কাজ করলেই হবে । consecutive sum করার ফলে ফাস্ট এ 5 পরের বার ০ এড হবে । এখন যদি আরো একটু বড় কেইজ নেই । 3 2 -5 . এইখানে আমাদের 7 unit কাজ করতে হয় , আমরা যে খান থেকে শুরু করি প্রথম থেকে বা শেষ থেকে consecutive sum করে যেতে থাকলেই আমরা রেজাল্টটা পাই । এই অবজারবেশন থেকে আমরা প্রবলেমটা সল্ভ করতে পারি ।

```
void solve()
{
     int n ;
    while( n = II )
    {
        if( !n ) break ;
        Long sum = 0 , ans = 0 ;
        rep ( i , n )
        {
            Long x = LL ;
            sum += x ;
            ans += abs( sum );
        }
        cout << ans << endl ;
    }


}
```

view rawgreedy3.cpp hosted with ❤ by **GitHub**

এই প্রবলেম Uva এবং codechef এ ছিল । এখন আইডি আমার মনে আসতেছে না । আমি চেস্টা করব আইডি গুলা এড করে দেবার জন্য পরে ।

এই লিখাগুলা লিখার পিছনে অনেকে কস্ট করে কমেন্ট করতেছে , ভাল ভাল কথা লিখতেছে :p যেইটা শুনতে তো ভালই লাগে । তাই লিখা ভাল লাগলে অবশ্যই জানাইতে ভুলবেন না :D আরো কিভাবে লিখাগুলাকে ভাল করা যায় বলবেন । আশারাখি সামনেই তৃতীয় পর্বটা আসবে ।