qA. Email Aliases

題目簡述：

給你一些長度不超過 100 的 email， 每個 email 只會且一定有一個 @ 字元將 email 切成前後兩段：

- 前半段非空，只由大小寫字母及字元 +. 組成，首字元為字母
- 後半段非空，只由大小寫字母及字元 . 組成，首尾字元皆為字母，無連續的 .

現在要將等價的 email 組再一起，然後輸出有幾組跟每組的個數還有 email。

比較有以下規則：

- 忽略字母大小寫
- 若後半段等價於 bmail.com 則其前半段忽略所有 . 以及第一個 +(含+自己) 到 @ 之間的所有字元

解法：

實作題，每筆資料複製兩份，一份原本的用於輸出，一份標準化後的用於比較。標準化就先把所有大寫轉小寫，再判斷 bmail.com 。開一個 map 用標準化後的字串作 grouping。

My AC Code：

source code: http://codeforces.com/contest/589/submission/14224428

```
#include <cstdio>
#include <cstring>
#include <map>
#include <vector>
using namespace std;

struct myCmp{
	bool operator()(const char* const a,const char* const b) const{
		return strcmp(a,b)<0;
	}
};

void bmailize(char *str){
	bool login_part = true;
	for(int i=0,j=0 ; true ; ++i , ++j){
		while( login_part && str[j]=='.' ) ++j;
		str[i] = str[j];
		if( str[j]=='@' ) login_part = false;
		if( str[j]=='\0' ) break;
	}
	for(int i=0,j=0 ; true ; ++i , ++j){
		if( str[j]=='+' )
			while( str[j]!='@' )
```

```
                ++j;
            str[i] = str[j];
            if( str[j]=='\0' ) break;
        }
}
void nomalize(char *str){
        for(int i=0 ; str[i]!='\0' ; ++i)
            if( str[i]>='A' && str[i]<='Z' )
                str[i] = str[i]-'A'+'a';

        bool bmail = false;
        for(int i=0 ; str[i]!='\0' ; ++i)
            if( str[i]=='@' ){
                bmail = strcmp(&str[i] , "@bmail.com")==0;
                break;
            }

        if( bmail )
            bmailize(str);
}

char ori[20004][104];
char nor[20004][104];

int main(){
        int n;
        scanf("%d",&n);
        for(int i=0 ; i<n; ++i){
            scanf("%s",ori[i]);
            strcpy(nor[i] , ori[i]);
        }

        for(int i=0 ; i<n ; ++i)
            nomalize(nor[i]);

        map<char*,int,myCmp> dic;
        vector< vector<int> > ans;
        for(int i=0 ; i<n ; ++i){
            auto it = dic.find(nor[i]);
            if( it==dic.end() ){
                dic.insert(make_pair(nor[i] , ans.size()));
                ans.push_back(vector<int>(1,i));
            }
            else
                ans[it->second].push_back(i);
        }
```

```
        printf("%d\n",(int)ans.size());
        for(auto &v : ans){
                printf("%d",(int)v.size());
                for(auto &w : v)
                        printf(" %s",ori[w]);
                puts("");
        }
        return 0;
}
```

qB. Layer Cake

題目簡述：

每個蛋糕模型高度都為一，給定 n≤4000 個蛋糕模型及各個的常寬 ≤1e6，同樣常寬的模型可以疊再一起。目標做一個體積最大的蛋糕，模型可以垂直切或水平裁切或90度旋轉(長變寬;寬變長)，讓它跟其他大小一樣的模型疊在一起。

解法：

為了簡化問題令長 L 總是大於寬 W。答案的長寬必定來自輸入的邊長，因為若否，則可將答案擴張到遇到下一長的邊長得到一更好答案。

如此可想到最簡單的作法是枚舉所有輸入的長 L 寬 W 然後統計有 cnt 個模型可用， n^3 找 l*r*cnt 的最大值。理所當然這樣會 TLE。

但若我們枚舉的長寬是從大到小排好的話，因為較長的可以切成較短，反之無法。就可以利用枚舉時會邊長會遞減的特性少掉一層統計 cnt 的迴圈。也就是說，若模型都依 L 從長排到短，則枚舉 L 時，只要在意那些 L 比自己大(也就是陣列 index 比自己小)的模型就好，將這些在意的模型再依 W 長到短排，就可以再枚舉 W 時，利用排序位置瞬間知道有多少模型長≥L，寬≥W。

依實作在 n^2 或 $n^2 \log n$ 內完成都是可以接受的。

My AC Code：

source code: http://codeforces.com/contest/589/submission/14294143

```
#include <cstdio>
#include <vector>
#include <algorithm>
using namespace std;

struct Cake{
	int len , wid;
	int id , id1 , id2;
} cake[4004];

int main(){
	int n;
	scanf("%d",&n);
	for(int i=0,a,b ; i<n ; ++i){
		scanf("%d%d",&a,&b);
		cake[i].len = max(a,b);
		cake[i].wid = min(a,b);
		cake[i].id = i;
	}

	vector<int> id1s , id2s;
	for(int i=0 ; i<n ; ++i){
```

```
        id1s.emplace_back(i);
        id2s.emplace_back(i);
    }
    sort(id1s.begin() , id1s.end() , [](const int &l,const int &r){
        return cake[l].len > cake[r].len ||
                    cake[l].len==cake[r].len && cake[l].id < cake[r].id;
    });
    sort(id2s.begin() , id2s.end() , [](const int &l,const int &r){
        return cake[l].wid > cake[r].wid ||
                    cake[l].wid==cake[r].wid && cake[l].id < cake[r].id;
    });
    for(int i=0 ; i<n ; ++i){
        cake[ id1s[i] ].id1 = i;
        cake[ id2s[i] ].id2 = i;
    }

    long long ansArea = 0LL;
    int ansL , ansW;
    for(int i=0 ; i<n ; ++i){
        long long nowL = cake[ id1s[i] ].len;
        int to = cake[ id1s[i] ].id2;
        int cnt = 0;
        for(int j=0 ; j<to ; ++j){
            if( cake[ id2s[j] ].id1 > i ) continue;
            ++cnt;
        }
        for(int j=to ; j<n ; ++j){
            if( cake[ id2s[j] ].id1 > i ) continue;
            ++cnt;
            long long nowW = cake[ id2s[j] ].wid;
            if( nowL*nowW*cnt > ansArea ){
                ansArea = nowL*nowW*cnt;
                ansL = nowL;
                ansW = nowW;
            }
        }
    }
    printf("%lld\n%d %d\n",ansArea,ansL,ansW);
    return 0;
}
```

qD. Boulevard

題目簡述：

有 2≤n≤1000 個人在林蔭大道上散步，他們每個人會在 t_i 時刻出現在 s_i 上然後以速率 1 朝 f_i 前進，到達 f_i 後瞬間消失，t_i s_i f_i 皆為整數且 $1 \le t_i$ s_i $f_i \le 1e6$，s_i != f_i 。
若同一時刻兩人出現再同一點上，則他們會打招呼。求每個人會跟多少不同的人打招呼。在出現或消失的點上打招呼是可以的。

解法：

n^2 統計每個人跟幾個人相遇。判斷是否相遇我用的方法是：
用出現時間跟走的距離判段兩人出現在大道的時間有無交集，若無則不可能相遇；若有，則算出交集的時間內，兩人分別從拿裡走到拿裡，設分別從 s1 走到 f1，s2 走到 f2。若走的方向相同，因為每個人速率一樣，所以只有當 s1==s2 時才有可能打到招呼，否則他們的永遠相距一段距離。若走的方向不一樣，則兩區間 [min(s1,f1) , max(s1,f1)] ， [min(s2,f2) , max(s2,f2)] 有交集代表有相遇。

My AC Code：

source code: http://codeforces.com/contest/589/submission/13916774

```
#include <cstdio>
#include <cmath>
#include <vector>
using namespace std;

struct Person{
	int t , e , s , f;
	int way;
};
Person person[1004];
int ans[1004];

bool meet(const Person &i,const Person &j){
	if( i.e < j.t ) return false;
	if( j.e < i.t ) return false;
	int t = max(i.t , j.t);
	int e = min(i.e , j.e);
	int s1 = i.s + i.way*(t-i.t);
	int f1 = i.s + i.way*(e-i.t);
	int s2 = j.s + j.way*(t-j.t);
	int f2 = j.s + j.way*(e-j.t);
	if( i.way * j.way > 0 )
		return s1==s2;
	if( s1 > f1 ) swap(s1,f1);
	if( s2 > f2 ) swap(s2,f2);
```

```
        if( f1 < s2 ) return false;
        if( f2 < s1 ) return false;
        return true;
}

int main(){
        int n;
        scanf("%d",&n);
        for(int i=0,t,s,f ; i<n ; ++i){
                scanf("%d%d%d",&person[i].t , &person[i].s , &person[i].f);
                person[i].e = person[i].t + abs(person[i].s - person[i].f);
                person[i].way = (person[i].f > person[i].s)? 1 : -1;
        }
        for(int i=0 ; i<n ; ++i)
                for(int j=i+1 ; j<n ; ++j)
                        if( meet(person[i] , person[j]) )
                                ++ans[i] , ++ans[j];
        for(int i=0 ; i<n ; ++i)
                printf("%d ",ans[i]);
        puts("");
        return 0;
}
```

qF. Gourmet and Banquet

題目簡述：

一個宴會上有 n≤100 道菜，每道菜只會在時間 [a_i , b_i] 出現，其中 0≤a<b≤10000，以整數秒為單位時間。

某美食家吃菜的原則是：

- 一個時間只能吃一到菜
- 每道菜都要吃到
- 花在吃每道菜的時間要一樣(都是整數秒)

再符合上述條件下，求花在吃的總時間最大化

解法：

答案存在的區間為 0~出現時間最短的那到菜，且每道菜能夠吃 x!=0 秒鐘必定可以吃 x-1 秒鐘，存在單調性，故可對答案區間進行二分搜。

現在問題變成，如何知道「每道菜吃 x 秒鐘是否可行」。

運用貪婪策略，先將所有菜依其消失的時間做排序，依序分配吃的時間。分配的策略為，從這道菜出現的時間往結束的時間走，一遇到沒有再吃菜的時間立刻安排吃這道菜。直到這道菜分配夠了 x 秒鐘換分配下一道菜，或無法分配 x 秒鐘 return false 。

為何可以這樣分配呢？

首先，結束時間比我早的都已分配完畢可以不用在意，如此要在意的只剩結束時間晚於等於我的。

對於比我早出現的菜，不管我怎麼分配都會佔用他 x 秒鐘所以對他來說我分配的方式沒差。

對於比我晚出現的菜，因為他們的結束時間都晚於等於我，故我從我一出現的時間開始分配給自己，佔用到其他還沒分配到時間的菜的時間一定最少，式最好的分配時間方式。

因為菜只有 100 道，每道菜可能花費 10000 去找可以用的時間，所以總花費時間為：二分搜時間 * 100 * 10000

My AC Code：

source code: http://codeforces.com/contest/589/submission/14294543

```
#include <cstdio>
#include <cstring>
#include <vector>
#include <algorithm>
using namespace std;

struct event{
	int from,to;
	event(const int &from,const int &to)
```

```
        :from(from), to(to){}
};
vector<event> es;
bool occupy[10004];

bool min_each(int ts){
        memset(occupy , 0 , sizeof(occupy));
        for(auto &e : es){
                int cnt = ts;
                for(int i=e.from ; i<=e.to && cnt ; ++i)
                        if( !occupy[i] ){
                                occupy[i] = true;
                                --cnt;
                        }
                if( cnt ) return false;
        }
        return true;
}

int main(){
        int n;
        scanf("%d",&n);
        int lb = 0 , rb = 0;
        for(int i=0,a,b ; i<n ; ++i){
                scanf("%d%d",&a,&b);
                rb = max(rb , b-a);
                es.emplace_back( event(a , b-1) );
        }
        sort(es.begin() , es.end() , [](const event &l,const event &r){
                return l.to < r.to;
        });

        while( lb<rb ){
                int c = (lb+rb+1)>>1;
                if( min_each(c) ) lb = c;
                else rb = c-1;
        }
        printf("%d\n",lb*n);
}
```

qG. Hiring

題目簡述：

有 n ≤ 2e5 位應徵者，公司開放 m ≤ 2e5 個工作天每天 1 ≤ t ≤ 1e6 單位時間給他們完成作品，每位應徵者在每一天的一開始需要 0 ≤ d ≤ 1e6 單位時間準備才能開始算工作進度，共需要 1 ≤ r ≤ 1e6 單位時間才能完成其所有工作進度。應徵者之間是相互獨立的。

求輸出每位應徵者最快可以在第幾個工作天完成作品，若無法輸出 0

解法：

對於一位應徵者來說，那些開放時數小於等於其準備時間的工作天其實跟沒有開(進度 0)是一樣的，計算 n 位應徵者每位花 m 去檢查最早甚麼時候完成進度會 TLE。但只要我們能快速知道到 x 天為止，所有開放時數 t 大於其準備時數 d 的「天數」和「時數總和」，就可以用：「時數總和」 - d * 「天數」 知道到 x 天為止其消耗的工作進度，這樣即可以二分搜知道每位應徵者最早完成的時間。

要完成上述有條現限制的前綴和 query 太困難。繞點路，我們可以先將應徵者用準備時間，工作天用開放時數從大到小排序。然後依序拿出來應徵者，將所有比現在這位應徵者準備時間還要大的工作天都加到前最綴和裏面，如此每位使用者看到的前綴和都是只有開放時數比自己準備時間還要大的。

前綴和需要更新跟查詢，可用 fenwick tree 實作。

My AC Code：

source code: http://codeforces.com/contest/589/submission/14295585

```
#include <cstdio>
#include <vector>
#include <algorithm>
using namespace std;

struct work{
	int t,day;
	work(const int &t,const int &day)
	:t(t), day(day){}
};
struct person{
	int d,r,id;
	person(const int &d,const int &r,const int &id)
	:d(d), r(r), id(id){}
};

int ans[200004];

long long fenWorktime[200004];
```

```cpp
long long fenWorkdays[200004];
void add(long long *a,int id,long long val){
	while( id<200004 ){
		a[id] += val;
		id += -id & id;
	}
}
long long query(long long *a,int id){
	long long sum = 0LL;
	while( id>0 ){
		sum += a[id];
		id -= -id & id;
	}
	return sum;
}

int main(){
	int n,m;
	scanf("%d%d",&n,&m);
	vector<work> wk;
	vector<person> cd;
	for(int i=1,t ; i<=m ; ++i){
		scanf("%d",&t);
		wk.emplace_back(work(t,i));
	}
	for(int i=0,d,r ; i<n ; ++i){
		scanf("%d%d",&d,&r);
		cd.emplace_back(person(d,r,i));
	}

	sort(wk.begin() , wk.end() , [](const work &l,const work &r){
		return l.t > r.t;
	});
	sort(cd.begin() , cd.end() , [](const person &l,const person &r){
		return l.d > r.d;
	});

	int id = 0;
	for(auto &nowPerson : cd){
		while( id < wk.size() && wk[id].t > nowPerson.d ){
			add(fenWorktime, wk[id].day, wk[id].t);
			add(fenWorkdays, wk[id].day, 1);
			++id;
		}

		int lb = 1 , rb = m+1;
```

```
            while( lb<rb ){
                int c = (lb+rb)>>1;
                long long myT = query(fenWorktime,c) -
                                query(fenWorkdays,c) * nowPerson.d;
                if( myT >= nowPerson.r ) rb = c;
                else lb = c+1;
            }
            ans[nowPerson.id] = (lb==m+1)? 0 : lb;
        }
        for(int i=0 ; i<n ; ++i)
            printf("%d ",ans[i]);
        puts("");
        return 0;
}
```

qH. Tourist Guide

題目簡述：

有 1≤ n ≤50000 個點，0≤ m ≤50000 條邊，1≤ k ≤n 個點是「著名城市」。現要規劃一路徑集合，使得：

- 每條路徑的頭尾都是不同的「著名城市」
- 每個「著名城市」只能當一條路徑的端點(頭或尾)
- 每條路徑不共用邊(但可共用點)

「著名城市」雖然只能當一條路徑的端點，但可以被很多條路徑經過。

讓路徑集合最大化，輸出數量及每條路徑。

解法：

先只考慮最多可有幾條路徑。

讓 dfs 樹的每個節點告知其父母「你的後代裡是否有一條通往著名城市的路徑」。

對於每個節點，一知道有兩條往下走可通往著名城市的路徑加把他們組在一起變成題目規定頭尾兩端都是著名城市的路徑，若節點本身是著名程式，則先幫自己配對好再幫別人處理。最後有多一條沒人用(或自己沒配到)才能告訴父母，「有一條給你用喔，幫它配一下」

本題還要輸出每條路徑，可以讓 dfs 遞迴將存放半條路徑的 vector 傳給父母。

可以留意的是，vector 可以用 move 或 swap 在 O(1) 做 vector 內容物的所有權轉移；而不是花 O(size) 的時間複製一份。這個實作細節在本題可以讓執行時間差 10 倍。

My AC Code：

source code: http://codeforces.com/contest/589/submission/14540061

```
#include <cstdio>
#include <vector>
#include <cstring>
#include <algorithm>
using namespace std;

vector<int> g[50004];
bool remarkale[50004];
bool visited[50004];

struct route {
	vector<int> a, b;
	route()
	:a(), b(){}
};
vector<route> ans;

vector<int> dfs(int nowAt,int parent) {
	visited[nowAt] = true;
```

```
        vector<int> last;
        if( remarkale[nowAt] )
                last.emplace_back(nowAt);
        for(auto &v : g[nowAt]) {
                if( v == parent ) continue;
                if( visited[v] ) continue;
                vector<int> nowR = move(dfs(v, nowAt));
                if( nowR.empty() ) continue;
                nowR.emplace_back(nowAt);
                if( last.size() ) {
                        ans.emplace_back(route());
                        ans.back().a = move(last);
                        ans.back().b = move(nowR);
                }
                else last.swap(nowR);
        }
        return last;
}

int main() {
        int n, m, k;
        scanf("%d%d%d", &n ,&m ,&k);
        for(int i=0,a,b ; i<m ; ++i) {
                scanf("%d%d", &a, &b);
                g[a].emplace_back(b);
                g[b].emplace_back(a);
        }
        for(int i=0,a ; i<k ; ++i) {
                scanf("%d", &a);
                remarkale[a] = true;
        }
        for(int i=1 ; i<=n ; ++i)
                if( !visited[i] )
                        dfs(i,-1);
        printf("%d\n",(int)ans.size());
        for(auto &r: ans) {
                printf("%d",(int)r.a.size()+(int)r.b.size()-2);
                for(int i=0 ; i<r.a.size() ; ++i)
                        printf(" %d",r.a[i]);
                for(int i=r.b.size()-2 ; i>=0 ; --i)
                        printf(" %d",r.b[i]);
                puts("");
        }
        return 0;
}
```

ql. Lottery

題目簡述：

有 n 顆球 k 個人，保證 n 可以被 k 整除 。每顆球都有一個介於 1 到 k 的編號，問至少要幫幾顆求重新圖色才能讓每個編號對應到的球數量一樣多。

解法：

統計每個編號有幾個球，超過 n/k 的部份都是必須要重圖的。

My AC Code：

source code: http://codeforces.com/contest/589/submission/14223448

```
#include <cstdio>
using namespace std;

int cnt[104];

int main(){
	int n,k;
	scanf("%d%d",&n,&k);
	int avg = n/k , ans = 0;
	for(int i=0,c ; i<n ; ++i){
		scanf("%d",&c);
		if( ++cnt[c] > avg )
			++ans;
	}
	printf("%d\n",ans);
	return 0;
}
```

qJ. Cleaner Robot

題目簡述：

有一檯掃地機器，在打掃一個 w * h (1 ≤ w,h ≤ 10)個方格組成的長方形房間。有些格子會被傢俱占著，可以不用打掃。機器人打掃的演算法如下：

1. 把站著的這格清乾淨
2. 若面對的方向的下一格不是傢俱也不是牆壁，則走到該方向下一格，回到 1.
3. 順時鐘轉 90 度回到 2.

給定機器人的初始位置及面對的方向，問總共可以掃多少格

解法：

照著他的演算法實作，然後統計每個格子被造訪的次數，若被造訪超過四次，代表從這格出去的每個方向至少都走過一次，可以不用走了，繼續走來走去也一定都是那幾格。統計哪些格子被走到。

My AC Code：

source code: http://codeforces.com/contest/589/submission/14223707

```
#include <cstdio>
using namespace std;

char grid[14][14];
int tryOutCnt[14][14];

int main(){
	int w,h;
	scanf("%d%d",&h,&w);
	for(int i=0 ; i<h ; ++i)
		scanf("%s",grid[i]);

	int r=-1 , c=-1;
	for(int i=0 ; i<h && r==-1 ; ++i)
		for(int j=0 ; j<w ; ++j)
			if( grid[i][j]!='.' && grid[i][j]!='*' ){
				r = i , c = j;
				break;
			}
	bool forward = false;
	while( tryOutCnt[r][c]<=4 ){
		++tryOutCnt[r][c];
		int nr , nc;
		char rotate;
		switch( grid[r][c] ){
			case 'U':
```

```
                        nr = r-1 , nc = c;
                        rotate = 'R';
                        break;
                case 'R':
                        nr = r , nc = c+1;
                        rotate = 'D';
                        break;
                case 'D':
                        nr = r+1 , nc = c;
                        rotate = 'L';
                        break;
                default:
                        nr = r , nc = c-1;
                        rotate = 'U';
        }
        if( nr>=h || nr<0 || nc>=w || nc<0 || grid[nr][nc]=='*' )
                grid[r][c] = rotate;
        else{
                grid[nr][nc] = grid[r][c];
                grid[r][c] = 'y';
                r = nr , c = nc;
        }
    }

    int ans = 0;
    for(int i=0 ; i<h ; ++i)
        for(int j=0 ; j<w ; ++j)
            if( grid[i][j] != '*' && grid[i][j]!= '.' )
                ++ans;
    printf("%d\n",ans);
    return 0;
}
```

qK. Task processing

題目簡述：

有 1≤ n ≤1e5 個工作，每個工作要執行 l_i 秒，會在 t_i 秒被加入待執行執行 queue， 1≤ l_i ≤1e5， 0≤ t_i ≤1e5。若在時間 T，queue 非空也沒有其他工作在執行，系統會從 queue 中選出 l_i - $(T-t_i)^2$ 最小的工作，若最小的有多個則選 index 最小的，並執行 l_i 秒把他做完。

求每個工作各在甚麼時間被執行完。

解法：

觀察評估優先度的式子，平方項 T 跟 t_i 差愈大愈可能被先拿出來執行，若 queue 中 t_i 最小的為 t_{min}，則其 $(T-t_{min})^2$ 會最大，對於所有 queue 中 $(T-t_i)^2$ 跟 $(T-t_{min})^2$ 差超過 1e5 的可以完全不用在意，因為不管他的 l_i 多小都不可能選到他。所以每次找工作來執行時，從被加入時間早的工作開始搜，一發現 $(T-t_{min})^2$ - $(T-t_i)^2$ > 1e5 就不用繼續找了。

不難發現 $T-t_{min}$ =1000 時：

$1000^2 - x^2 > 1e5$， $x \le 1000$

$x > 948$

要在意的 t_i 數量不會超過 100 個了，當 $T-t_{min}$ 愈大要在意的 t_i 會愈少。

但若每次找工作還是從 queue 中一個一個拿起來檢查，會變成 n^2 。所以關鍵是把 t_i 一樣的全部存在同一 vector 裏用 l_i 跟 index 排好序。這樣就真的可以在 $T-t_{min}$ 大於 1000 時，掃過少於 100 個時間點即可。

My AC Code：

source code: http://codeforces.com/contest/589/submission/14537670

```
#include <cstdio>
#include <vector>
#include <algorithm>
using namespace std;

struct task{
	int id, l;
	task(const int &id,const int &l)
	:id(id), l(l) {}
};
vector<task> t[100004];
long long ans[100004];

inline long long eval(long long T,long long id) {
	return t[id].back().l - (T-id)*(T-id);
}
inline void process(long long &nowT,int id) {
	if( nowT<=id )
		nowT = id + t[id].back().l;
```

```
        else
                nowT += t[id].back().l;
        ans[ t[id].back().id ] = nowT;
        t[id].pop_back();
}

int main() {
        int n;
        scanf("%d",&n);
        for(int i=0,li,ti ; i<n ; ++i) {
                scanf("%d%d",&li,&ti);
                t[ti].emplace_back(task(i,li));
        }
        for(int i=0 ; i<100004 ; ++i)
                sort(t[i].begin(), t[i].end(), [](const task &l,const task &r){
                        return l.l > r.l ||
                                        l.l == r.l && l.id > r.id;
                });
        int id = 0;
        long long nowT = 0;
        for(int i=0 ; i<n ; ++i) {
                while( t[id].empty() ) ++id;
                if( nowT<id ) {
                        process(nowT, id);
                        continue;
                }
                long long diff2 = (nowT-id)*(nowT-id);
                int minid = id;
                int minval = eval(nowT, id);
                int to = min(100000LL, nowT);
                for(int j=id+1,diff=1 ; j<=to &&
                                        diff2-(nowT-j)*(nowT-j)<=100004 ; ++j) {
                        if( t[j].empty() ) continue;
                        int val = eval(nowT, j);
                        if( val<minval ||
                                 val==minval && t[j].back().id<t[minid].back().id ) {
                                minval = val;
                                minid = j;
                        }
                }
                process(nowT, minid);
        }
        for(int i=0 ; i<n ; ++i)
                printf("%lld ",ans[i]);
        puts("");
```

```
        return 0;
}
```