## Проблем 5. ХорДужи

Перица је на папиру нацртао $n$ хоризонталних дужи. Затим је тај папир дао свом другу Јовици и задао му задатак да преброји све дужи. Међутим, ако се неке дужи преклапају, Јовица неће приметити да су то различите дужи, већ ће их посматрати као једну дуж. Две дужи се преклапају ако имају бар једну заједничку тачку.

Колико дужи ће Јовица да преброји?

**Улаз.** (Улазни подаци се учитавају на стандардног улаза) Први ред стандардног улаза садржи један природан број $n$ ($1 \leq n \leq 100.000$), број дужи. У следећих $n$ редова, налазе се по три цела броја $y_i$ , $xa_i$ , $xb_i$ ($-1.000.000.000 \leq y_i, xa_i, xb_i \leq 1.000.000.000$) који означавају да крајеви $i$-те дужи имају координате $(xa_i, y_i)$ и $(xb_i, y_i)$.

**Излаз.** (Излазне подаци се исписују на стандардни излаз) У први и једини ред стандардног излаза исписати колико дужи види Јовица.

**Пример 1.**

| horduzi.in | horduzi.out |
|---|---|
| 7<br>7 5 8<br>3 8 4<br>4 1 2<br>3 5 2<br>3 3 10<br>7 2 5<br>7 12 9 | 4 |

**Ограничења.** Временско ограничење: 1$s$; Меморијско ограничење: 16$MB$

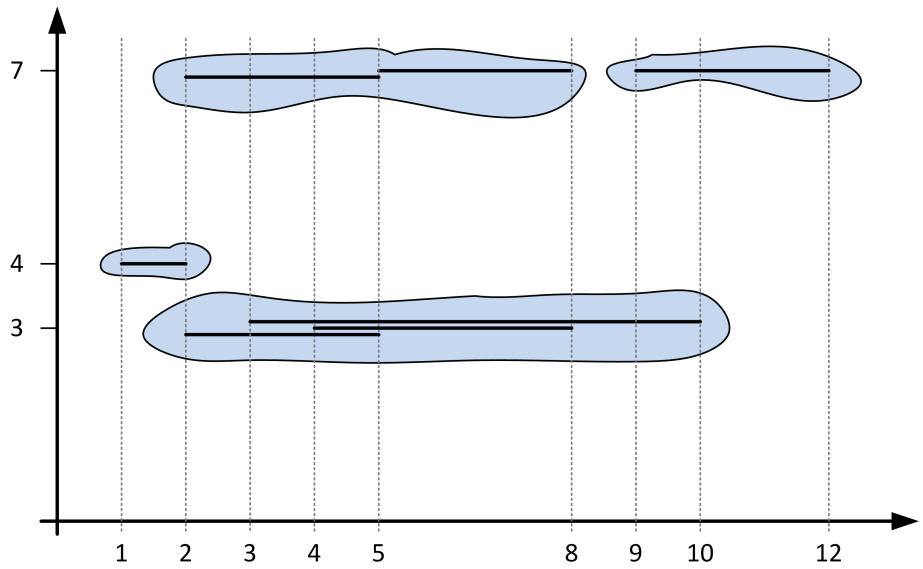
---

**Решење и анализа.** Идеја овог проблема је позната и постоји доста варијанти исте (неке од њих ћемо напоменути касније). Међутим, на квалификацијама се овај проблем показао као солидно чврст орах - просечан број бодова био је свега око 18, док око две трећине такмичара нема позитиван број бодова на њему.

Анализу проблема ћемо започети анализом датог примера (што је увек добар почетак при решавању). На почетку напоменимо да границе дужи у улазу нису сортиране ($xa$ не мора бити мање од $xb$). У даљем делу анализе претпостављамо да важи поредак $xa < xb$, тј. да смо ове вредности заменили на самом улазу уколико поредак није био добар. Пример можемо приказати графички као на слици 1, при чему ово није баш реална ситуација - одређене дужи су мало спуштене или подигнуте како би се приметила места преклапања. Решење, односно дужи које Јовица види су означена у плавим облацима.

Ради лакшег описа алгоритма, скуп дужи које Јовица види као једну називамо групом. Прво што можемо приметити, а што се дало и наслутити, јесте да се дужи са различитом $y$ координатом никада не могу наћи у истој групи односно преклопити у некој тачки. Дакле, за свако $y$ проблем можемо посматрати независно. Ово је битна и јако лепа чињеница јер смо на основу ње проблем свели на дужи које се налазе на "истој висини". Како наћи дужи које

су на истој висини? Тривијално решење, које наравно није добро, је да за сваку вредност $y$ координате тражимо дужи на њој. Међутим, бољи приступ је да се дужи сортирају по $y$ координатама. Ова идеја се сама намеће, јер на тај начин добијамо да се све дужи на истом нивоу (иста $y$ координата) налазе једна уз другу. Кокретно, скуп дужи са истог нивоа представља подниз сортираног низа.




Слика 1. Графички приказ примера "са папира".

Сортирањем дужи из примера добија се низ:

$$(3,2,8),(3,3,10),(3,4,8),(4,1,2),(7,2,5),(7,5,8),(7,9,12)$$

где примећујемо да су дужи са $y = 3$ координатом од прве до треће, да је дуж са индексом 4 једина са $y = 4$ координатом, док су дужи на нивоу 7 од пете до седме. Сваки од ових скупова дужи на истом нивоу посматрамо посебно. Овим смо добили нови подпроблем:

*за дате дужи на x-оси (односно истом нивоу), наћи број дужи које Јовица види*

Дакле, сада можемо посматрати дужи са крајевима $(a_1, b_1), (a_2, b_2), \ldots, (a_m, b_m)$. Посматрајмо дужи $(a_i, b_i)$ и $(a_j, b_j)$ и претпоставимо да оне припадају једно групи дужи. Ако се оне секу, добијамо да је дуж коју оне граде, односно њихова унија, заправо $(\min\{a_i, a_j\}, \max\{b_i, b_j\})$. Примера ради, дужи $(3, 5)$ и $(4, 6)$ које се секу граде дуж $(3, 6)$. Како је потребно наћи ове "ланце" дужи које се преклапају или додирују, некако сортирање опет виси у ваздуху као кључни корак. Идеју методе коју ћемо овде изнети је теже објаснити овако "с неба па у ребра". Зато ћемо одмах почети са описом методе а на крају продискутовати о идеји.

Дефинишимо нови низ $x$ и паралелно са њим низ *open* на следећи начин:

$$x[2k+1] = a_k,\ open[2k+1] = 1$$

$$x[2k] = b_k,\ open[2k] = -1$$

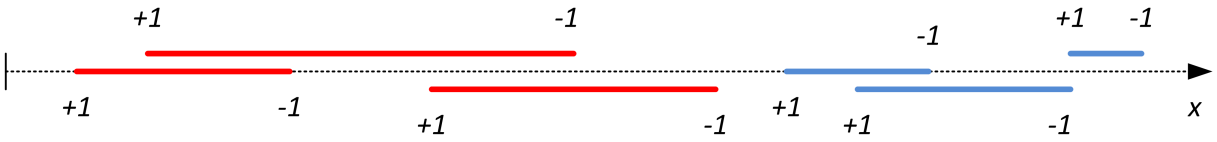
Другим речима, у низ $x$ редом ређамо крајеве дужи а у низу *open* плус и минус један. Као што видимо 1 означава почетак дужи а $-1$ њен крај. Сада долази на реду сортирање. Наиме,

сортирамо низ $x$ у неопадајући поредак, а паралелно са њим мењамо и низ *open* (ово можемо посматрати и као структуру која има $x$ координату и параметар *open* док се сортирање врши упоређивањем само атрибута $x$). Специјалан случај је када се две дужи секу у једној тачки односно крај прве је почетак друге. Тада у низу $x$ имамо два исте вредности али једној одговара $+1$ а другој $-1$ у низу *open*. Зато уводимо правило сотирања да уколико су $x$ координате једнаке тада се сортира по *open* али тако да прво иде $+1$ а затим $-1$ (другим речима нерастуће по *open*). Касније ћемо видети зашто је ово битно (односно шта се мења уколико се сортира само по $x$ координати).

Шта смо добили на овај начин? Ако се крећемо по низу $x$ ми се заправо крећемо по крајевима датих дужи али у неопадајућем поретку. Крајеве дужи ћемо ословљавати са почетак и крај, респективно. Уколико наиђемо на почетак неке дужи, тада све дужи на чији почетак наиђемо пре затварања ове дужи, припадају истој групи. Дакле уколико постоји нека дуж која још увек није затворена (нисмо још увек наишли на њен крај) тада све дужи на које наилазимо припадају истој групи као и та не затворена дуж. У нашем случају није битно које дужи припадају којим групама, тако да последњу реченицу можемо превести како: крај неке дужи је и крај групе ако и само ако су све дужи пре ње затворене.

Ово нас наводи на следећи алгоритам:

- Иницијализујмо број група и број отворених дужи на нула, односно $numGroups = 0$ и $numOpen = 0$
- Редом се крећемо по низу $x$, а самим тим и по низу *open*. Означимо тренутни индекс у овим низовима са $k$.
  - Уколико смо наишли на почетак неке дужи, односно $open[k] = 1$, повећавамо број отворених дужи за 1 тј. $numOpen = numOpen + 1$.
  - Уколико смо наишли на крај неке дужи, односно $open[k] = -1$, број отворених дужи смањујемо за један тј. $numOpen = numOpen - 1$. Ако је ово смањивање довело до тога да је $numOpen = 0$, тада смо затворили текућу групу и укупан број група повећавамо: $numGropus = numGroups + 1$.




Слика 2. Пример за вредности *open* низа. Након сортирања добија се низ $open = (1, 1, -1, 1, -1, -1, 1, 1, -1, 1, -1, -1)$.

Зашто смо на почетку изабрали баш вредности $+1$ и $-1$ за елементе низа *open*? Логичније би било да смо за то искористили неки *boolean* низ (с обзиром да су нам потребне само две вредности) чиме штедимо меморију. Међутим, шта примећујемо у опису алгоритма: када је $open[k] = 1$ број отворених дужи се повећава за један, а када је негативан тада се смањује за један. Сада видимо да за ово не морамо разликовати случајеве, једноставно можемо записати само $numOpen = numOpen + open[k]$. Ово је свакако елегантније од гранања.

Овде налазимо и одговор на питање сортирања: зашто сортирамо по $x$, а уколико су једнаки онда по *open*? Наиме, уколико би неко затварање било пре отварања, тада би *numOpen* могао да постане нула и пре затварања групе. Примера ради, уколико имамо само две дужи $(1, 2)$ и $(2, 3)$, тада би имали да је (у овом погрешном резоновању): $x = (1, 2, 3, 4)$ и $open = (1, -1, 1, -1)$.

Сада би имали да *numOpen* узима вредности $1, 0, 1, 0$, редом. Како два пута узима вредност нула, добили би да је укупан број група 2, а заправо је само један. Зато низ *open* мора бити сортиран као: $open = (1, 1, -1, -1)$.

Вратимо се на наш почетни проблем. Да ли ми морамо посматрати све скупове дужи са истим $y$ координатама засебно? Не морамо, јер ће свакако након завршетка дужи из једног нивоа, број отворених дужи бити нула, тако да можемо директно наставити са другим скупом дужи. Како је тај други скуп дужи одмах у наставку низа (подсетимо се да су дужи сортиране прво по $y$) једноставно ова подела по скуповима се имплицитно решава сортирањем.

Када сву ову причу спојимо, добијамо да је алгоритам који решава овај проблем доста једноставан за имплементирање (иако то на први поглед не делује тако). Наиме, дефинишимо структуру са атрибутима $(x, y, open)$. За сваку дуж са улаза, у низ $p$ ових струкутра убацујемо $(xa_i, y_i, 1)$ и $(xb_i, y_i, -1)$. Затим сортирамо овај низ прво по $y$, за једнаке $y$ по $x$ у неопадајући поредак, а за једнаке и $x$ и $y$ по *open* (монотоност при сортирању по $y$ није битна). Након тога се крећемо по низу и стално инкрементирамо вредност *numOpen* за вредност *open* из структуре и бројимо колико пута узима вредност нула.

Како дефинисати критеријум сортирања? Наиме, као што смо видели, низ чији су елементи описане структуре треба сортирати по три критеријума. Означимо са $A$ и $B$ две инстанце ове структуре. Тада критеријум сортирања можемо описати као:

$$(A.y < B.y) \vee ((A.y == B.y) \wedge (A.x < B.x)) \vee ((A.y == B.y) \wedge (A.x == B.x) \wedge (A.open > B.open))$$

Главни део имплементације (без учитавања тј. инизијализације низа $p$) се може искодирати у само петнаестак линија кода:

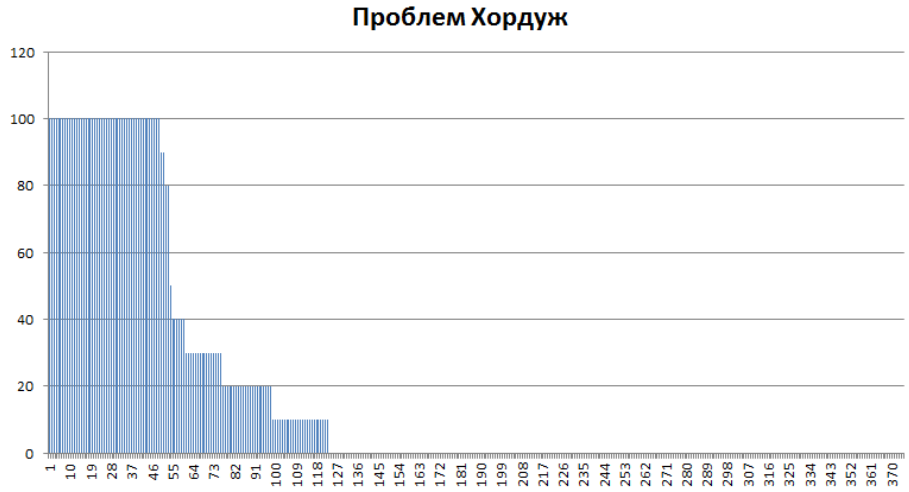
```
int compare (const void * a, const void * b)
{
    POINT A = *(POINT *)a;
    POINT B = *(POINT *)b;
    if ((A.y > B.y) || ((A.y == B.y) && (A.x > B.x)) || ((A.y == B.y) && (A.x == B.x) && (A.open < B.open)))
        return 1;
    return -1;
}

int solve()
{
    int toReturn = 0;

    qsort (p, 2 * n, sizeof(POINT), compare);

    int numOpen = 0;
    for (int i = 0; i < 2 * n; i++)
    {
        numOpen = numOpen + p [i].open;
        if (numOpen == 0)
            toReturn++;
    }

    return toReturn;
}
```

Сложеност изложеног алгоритма је $O(n \log n)$. Заиста, након сортирања потребан је само један пролазак кроз низ. Меморијска сложеност је линеарна по $n$.

**Напомена.** Постоје многе варијанте овог проблема. Неке од њих су (проблеми су изложени у једној димензији):

- за дате дужи на $x$-оси израчунати укупну дужину које оне прекривају;

- за дате дужи на $x$-оси наћи тачку која припада највећем броју дужи (решење ове варијанте је максимална вредност коју узима *numOpen*);
- за дате дужи на $x$-оси наћи оне дужи које припадају истој групи као и дата дуж $d$;




Слика 3. График броја освојених бодова свих такмичара.