Друштво математичара Србије
Фондација Петља

Такмичења из програмирања за ученике основних школа

2019. година

Садржај

Глава 1

Сезона 2019/2020.

1.1 Први круг квалификација

Задатак: Књига

Поставка: Књига има N поглавља. Никола је првог дана прочитао A поглавља, а другог дана два поглавља више него првог. Написати програм који учитава целе позитивне бројеве N, A, и исписује колико поглавља је остало Николи да прочита.

Улаз: Са стандардног улаза се у првом реду уноси број N ($1 \leq N \leq 99$), а у другом реду број A ($1 \leq A \leq 99$). Улазни подаци су такви да преостали број поглавља никад није негативан.

Излаз: На стандардни излаз исписати један број, број поглавља која Никола још није прочитао.

Пример 1

Улаз	*Излаз*
15	9
2	

Пример 2

Улаз	*Излаз*
20	0
9	

Решење: Решење је сасвим једноставно. Ако је Никола првог дана прочитао A поглавља а другог дана $A+2$ поглавља, то значи да му остаје да прочита још $N-A-(A+2)$ поглавља.

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

int main()
{
    int n, a;
    cin >> n >> a;
    cout << n - a - (a+2) << endl;
    return 0;
```

}

Задатак: Сијалица

Поставка: Стубови уличне расвете су нумерисани редом по улицама. У свакој улици има по N стубова, тако да су стубови у првој улици нумерисани $1, 2, \ldots N$, у другој су бројеви стубова $N + 1, N + 2, \ldots 2N$ итд. Мирко је добио задатак да у свакој другој улици замени сијалицу на сваком трећем стубу. Када је дошао до стуба са бројем A, Мирко се забројао. Написати програм који учитава целе позитивне бројеве N и A и одговара на питање да ли на том стубу треба заменити сијалицу.

Улаз: Са стандардног улаза се у првом реду уноси број N $(1 \leq N \leq 1000)$, а у другом реду број A $(1 \leq A \leq 1000)$.

Излаз: На стандардни излаз исписати само реч da ili ne.

Пример 1		**Пример 2**	
Улаз	*Излаз*	*Улаз*	*Излаз*
20	da	100	ne
75		300	

Решење: Када би се стубови и улице бројали од 0, тада би редни број улице могао да се добије као $A \operatorname{div} N$, а редни број стуба у улици као $A \bmod N$ (где су са div и $\bmod$ означени целобројни количник, тј. остатак при дељењу два цела броја). Међутим, пошто се у задатку броји од 1, учитану ознаку на стубу треба смањити за 1, а добијене резултате за редни број улице и стуба у улици повећати за 1. То значи да редни број улице треба рачунати као $(A - 1) \operatorname{div} N + 1$, а редни број стуба у улици као $(A - 1) \bmod N + 1$.

Након овог израчунавања потребно је још само проверити да ли је редни борј улице дељив са 2 и редни број стуба у улици дељив са 3.

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

int main()
{
    int n, a;
    cin >> n >> a;
    int ulica = (a-1) / n + 1;
    int brUUlici = (a-1) % n + 1;
    if ((ulica % 2 == 0) and (brUUlici % 3 == 0))
        cout << "da" << endl;
    else
        cout << "ne" << endl;
    return 0;
}
```

Задатак: Запета

Поставка: Дат је низ ASCII карактера, међу којима се појављује тачно један знак "," (запета) и на самом крају тачно један знак "." (тачка). Сви остали знаци, ако их има, су слова енглеске абецеде, цифре, заграде и размаци и они могу да се понављају. Написати програм који учитава дати низ карактера, а на стандардни излаз исписује текст у коме су део до запете и део од запете разменили места, док тачка остаје на крају текста.

Улаз: На стандардном улазу се у једном реду налази низ од највише 100 ASCII карактера.

Излаз: На стандардни излаз исписати измењени низ карактера.

Пример 1

Улаз	*Излаз*
`Da si hteo, mogao si.`	`mogao si,Da si hteo.`

Пример 2

Улаз	*Излаз*
`,a b c.`	`a b c,.`

Пример 3

Улаз	*Излаз*
`x,.`	`,x.`

Решење: У овом задатку је најпогодније да се прочита цео ред улаза као један стринг (без обзира на то да ли стринг садржи размаке). Након учитавања можемо да издвојимо део A од почетка стринга до запете (не укључујући запету) а затим и део B од запете до тачке (не укључујући запету и тачку).

У језику C++ позицију карактера `c` унутар стринга `s` можемо одредити позивом `s.find(c)`, док део стринга `s` који почиње на позицији `p` и има `d` карактера можемо одредити позивом `s.substr(p, d)`.

После оваквог издвајања остаје још само да испишемо делове у траженом редоследу (део B, запета, део A и на крају тачка).

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

int main()
{
    string s;
    getline(cin, s);
    int pozZapete = s.find(',');
    int pozTacke = s.find('.');
    string a = s.substr(0, pozZapete);
    string b = s.substr(pozZapete + 1, pozTacke - pozZapete - 1);
    cout << b << ”,” << a << ”.” << endl;
    return 0;
}
```

Задатак: Непливачи

Поставка: Деда Раде жели да упише својих четворо унука непливача у школу пливања. Инструктор му је рекао да је остао само један слободан термин, за који могу да се пријаве само деца виша од 110 сантиметара. Деда Раде не жели да раздваја унуке, па ће их уписати у школу пливања само ако сви испуњавају услов. Написати програм који одређује да ли деда Раде већ сада може да упише свих четворо унука у школу пливања.

Улаз: Са стандардног улаза се уносе четири цела позитивна броја не већа од 180, сваки у посебном реду – висине деда Радетових унука.

Излаз: На стандардни излаз исписати реч SVI ако је свако од четворо деце више од 110cm, а реч NIKO ако бар једно дете није више од 110 cm.

Пример 1

Улаз	*Излаз*
131	SVI
111	
128	
117	

Пример 2

Улаз	*Излаз*
131	NIKO
110	
128	
117	

Решење: Након учитавања потребно је само проверити да ли су сва 4 учитана броја већа од 110. Најједноставнији начин је употреба сложеног логичког услова.

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

int main()
{
    int a, b, c, d;
    cin >> a >> b >> c >> d;
    if (a > 110 and b > 110 and c > 110 and d > 110)
        cout << "SVI" << endl;
    else
        cout << "NIKO" << endl;
    return 0;
}
```

Задатак: Сусрет

Поставка: Новак и Јагош су стари другари и они се често без договарања сачекују на свом омиљеном месту у неко уобичајено време, па ако први дочека другог онда проведу неко време дружећи се. Када Новак стигне први, он чека Јагоша 10 минута, а Јагош (ако он стигне први) чека Новака 15 минута.

Написати програм који за дата времена данашњег појављивања Новака и Јагоша одговара на питање да ли су се састали.

Улаз: Учитавају се четири цела броја, NS, NM, JS, JM редом, сваки у посебном реду стандардног улаза. Бројеви NS, NM представљају сат и минут Новаковог доласка, а JS, JM представљају сат и минут Јагошевог доласка. Подаци ће предста-

вљати исправно записана времена (то јест, важиће $0 \leq NS \leq 23$, $0 \leq NM \leq 59$, $0 \leq JS \leq 23$, $0 \leq JM \leq 59$).

Излаз: На стандардни излаз исписати само реч da ili ne.

Пример 1		**Пример 2**		**Пример 3**		**Пример 4**	
Улаз	*Излаз*	*Улаз*	*Излаз*	*Улаз*	*Излаз*	*Улаз*	*Излаз*
10	da	21	ne	0	da	0	ne
15		15		15		0	
10		21		0		23	
25		26		0		59	

Решење: Задатак се једноставније решава ако се после учитавања прво израчуна број минута од почетка дана до Николиног и Јагошевог доласка: $N = N_{sat} \cdot 60 + N_{min}$, и $J = J_{sat} \cdot 60 + J_{min}$.

Уїнежїена їранања

Решавање задатка сада можемо да довршимо тако што најпре проверимо ко је први стигао на место састанка. Ако је то Новак, онда проверавамо да је Јагош дошао највише 10 минута након Новака, а ако је Јагош први стигао, онда проверавамо да ли је Новак дошао највише 15 минута након Јагоша.

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

int main()
{
  int nsat, nmin, jsat, jmin;
  cin >> nsat >> nmin >> jsat >> jmin;
  int n = nsat*60 + nmin;
  int j = jsat*60 + jmin;

  if (n<j) // ako je Novak dosao pre
    if (j <= n+10)
      cout << "da" << endl;
    else
      cout << "ne" << endl;
  else // ako je Jagos dosao pre (ili su dosli istovremeno)
    if (n <= j+15)
      cout << "da" << endl;
    else
      cout << "ne" << endl;
  return 0;
}
```

Сложени услов

Можемо и да формирамо сложени услов, који директно одговара на питање да ли је дошло до сусрета. Да би одговор био потврдан, треба да важи $N \leq J \leq N + 10$ или $J \leq N \leq j + 15$, у противном одговор је одречан.

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

int main()
{
    int nsat, nmin, jsat, jmin;
    cin >> nsat >> nmin >> jsat >> jmin;
    int n = nsat*60 + nmin;
    int j = jsat*60 + jmin;

    if ((n <= j && j <= n+10) || (j <= n && n <= j+15))
      cout << "da" << endl;
    else
      cout << "ne" << endl;

    return 0;
}
```

Задатак: Јаја

Поставка: У сваку кутију за јаја може да стане K јаја. Када Јоца пакује јаја, он користи најмањи број кутија у које јаја могу да стану, али пошто је сујеверан, он никада не оставља у кутији (позитиван) број јаја дељив са 3.

Написати програм који одређује колико кутија треба Јоци да би спаковао J јаја.

Улаз: у првом реду стандардног улаза је број K, који је увек 10 или 15. У другом реду је цео број J, такав да је $0 \leq J \leq 100$.

Излаз: На стандардни излаз исписати један цео број, број кутија које ће Јоца употребити.

Пример 1		**Пример 2**	
Улаз	*Излаз*	*Улаз*	*Излаз*
10	2	15	1
18		10	

Решење: Прво што можемо да приметимо је да у кутије од 15 јаја можемо да ставимо највише 14 јаја. Према томе, оваквим кутијама можемо одмах након учитавања да смањимо величину за 1 и поједноставимо даље разматрање.

Погледајмо шта би се догодило када би се кутије пуниле редом до краја. Тада би број потребних кутија био $\lceil \frac{J}{K} \rceil$, где је J број јаја која треба спаковати, а K број јаја која стају у кутију.

Пошто након почетне исправке величина кутије K више није дељива са 3, то услов дељивости може да утиче само на последњу употребљену кутију. Зато још треба посебно испитати ситуацију када је број јаја у последњој употребљеној кутији дељив са 3. Таква сиутација може бити разрешена или узимањем нове кутије (само ако је неопходно), или пребацивањем једног или више јаја из претходно попуњених кутија у последњу. Додавање јаја у последњу кутију није могуће само у следећа два случаја:

(1) када нема ниједне претходно попуњене кутије из које бисмо "позајмили" јаја (то јест када је $J \leq K$).

(2) када у последњој кутији има места само за још једно јаје (то јест када је $J \bmod K = K - 1$). У овом случају додавање није могуће зато што би пребацивањем једног јајета из било које попуњене кутије у тој претходно попуњеној кутији остао број јаја дељив са 3.

У свим осталим случајевима се из претпоследње у последњу кутију могу пребацити бар једно или два јаја. На тај начин, број јаја у последњој кутији у сваком случају више неће бити дељив са 3, а од два могућа пребацивања (једно или два јаја) једно од тих пребацивања мора одговарати и претпоследњој кутији, тако да ни у њој број јаја не буде дељив са 3. Према томе, у свим случајевима осим два издвојена раније, број кутија се не мора повећавати.

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

int main()
{
    int velKutije, brJaja;
    cin >> velKutije >> brJaja;
    if (velKutije % 3 == 0) velKutije--;

    int brKutija = (brJaja + velKutije - 1) / velKutije;
    int uPoslednjoj = brJaja % velKutije;

    // ako je u poslednjoj kutiji nezgodan broj
    if (uPoslednjoj > 0 && uPoslednjoj % 3 == 0)
      // ako nemamo odakle u nju da dodamo (poslednja kutija je jedina)
      // ili ako bi se dodavanjem pokvarila prethodna kutija
      if (brKutija == 1 || uPoslednjoj + 1 == velKutije)
        brKutija++;  // onda ce trebati kutija vise

    cout << brKutija << endl;
    return 0;
}
```

Задатак: Огрлица

Поставка: Сара чува перле у N кутијица нумерисаних бројевима од 1 до N. Она увек прави огрлицу тако што из сваке од N кутијица редом по бројевима узме по једну перлу и у истом редоследу наниже перле на конац.

Написати програм који одређује на колико начина Сара може да изабере перле за следећу огрлицу.

Улаз: у првом реду стандардног улаза је цео позитиван број N, број кутијица, не већи од 10. У сваком од следећих N редова је по један цео једноцифрен број, број перли у одговарајућој кутијици по реду.

Излаз: На стандардни излаз исписати један цео број, број начина на које Сара може да направи следећу огрлицу.

Пример 1

Улаз	*Излаз*
5 3 3 2 5 1	90

Пример 2

Улаз	*Излаз*
3 7 0 2	0

Решење: Пошто се перле могу комбиновати свака са сваком, број начина да се наниже огрлица је једнак производу броја перли из појединих кутија. Дакле, потребно је само помножити дате бројеве перли и исписати производ тих бројева.

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

int main()
{
    int n, p;
    cin >> n;
    p = 1;
    for (int i = 0; i < n; i++)
    {
        cin >> a;
        p *= a;
    }
    cout << p << endl;
    return 0;
}
```

Задатак: Секција

Поставка: У школи је велико интересовање за програмерску секцију, на којој ће се правити рачунарске игре. Наставник је поставио услов да на секцију могу да иду само они ученици који имају 5 из информатике и бар 4 из математике.

Написати програм који одређује колико ученика може да се пријави за секцију.

Улаз: у првом реду број N, број ученика заинтересованих за секцију, природан број не већи од 200. У сваком од следећих N редова низ оцена једног од заинтересованих ученика из свих 11 предмета редом, без размака. Оцена из математике је шеста, а из информатике девета у низу од 11 цифара.

Излаз: На стандардни излаз исписати један цео број, број ученика који испуњавају услов.

Пример 1

Улаз	*Излаз*
4	2
55555355455	
55555455455	
55555455555	
22222522522	

Пример 2

Улаз	*Излаз*
2	0
55555555455	
55555355555	

Решење: Најједноставније је да учитавамо сваки ред као стринг од 11 карактера (цифара). Потребно је да пребројимо оне стрингове код којих је шеста цифра слева (бројећи од 1) већа од 3, а девета слева једнака 5.

Након пребројавања исписујемо вредност бројача.

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

int main()
{
    int n, br;
    cin >> n;
    string s;
    br = 0;
    for (int i = 0; i < n; i++)
    {
        cin >> s;
        if (s[5] - '0' > 3 && s[8] - '0' == 5)
            br++;
    }
    cout << br << endl;
    return 0;
}
```

Задатак: Статистике33

Поставка: Чувени кошаркаш Дабло Трипловић има толико добре статистике, да се већ помало сумња да су његови успеси добро пребројани.

Написати програм који одређује колико пута је Трипловић постигао такозвани трипл–дабл.

Улаз: На стандардноим улазу се у првом реду налази цео број N, $(1 \leq N \leq 100)$, број утакмица које је Трипловић одиграо у сезони. У сваком од следећих N редова по 3 цела броја раздвојена по једним размаком: број поена, скокова и асистенција редом (ови бројеви нису већи од 1000).

Излаз: На стандардни излаз исписати један број, број утакмица на којима је Трипловић постигао трипл–дабл.

Напомена

За потребе овог задатка трипл–дабл дефинишемо као најмање двоцифрен учинак (10 или више) у све три категорије које се прате. Другим речима: сва три броја већа од 9 у једном реду улаза.

Пример

Улаз	*Излаз*
3 20 9 7 127 12 11 11 14 12	2

Решење: Потребно је пребројати редове улаза у којима су сва три броја већа од 9. Увешћемо бројач који пре петље постављамо на 0, а затим у петљи учитавамо по три броја, прверавамо да ли су сва три већа од 9, и ако јесу - увећавамо бројач.

По завршетку петље исписујемо вредност бројача.

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

int main()
{
    int n, brTriplDabl = 0, a, b, c;
    cin >> n;
    for (int utak = 0; utak < n; utak++)
    {
        cin >> a >> b >> c;
        if (a > 9 && b > 9 && c > 9)
            brTriplDabl ++;
    }
    cout << brTriplDabl << endl;
    return 0;
}
```

Задатак: Статистике35

Поставка: Чувени кошаркаш Дабло Трипловић има толико добре статистике, да се већ помало сумња да су његови успеси добро пребројани.

Написати програм који одређује колико пута је Трипловић постигао такозвани трипл–дабл.

Улаз: Са стандардног улаза се у првом реду уноси цео број N $(1 \leq N \leq 100)$, број утакмица које је Трипловић одиграо у сезони. У сваком од следећих N редова је по 5 целих бројева раздвојених по једним размаком: број поена, скокова, асистенција, блокада и украдених лопти редом (ови бројеви нису већи од 1000).

Излаз: На стандардни излаз исписати један цео број, број утакмица на којима је Трипловић постигао трипл–дабл.

Напомена

За потребе овог задатка трипл–дабл дефинишемо као најмање двоцифрен учинак (10 или више) у бар три од пет категорија које се прате. Другим речима: бар три броја већа од 9 у једном реду улаза.

Пример

Улаз	*Излаз*
3 20 9 7 2 1 127 12 11 3 0 0 10 11 14 12	2

Решење:

Анализа

У овом задатку треба пребројати редове улаза у којима су бар три од пет бројева већи од 9. Да бисмо установили да ли неки ред улаза треба бројати, треба у сваком реду пребројати елементе који су већи од 9. То значи да ћемо у овом задатку применити технику пребројавања "на два нивоа": глобално бројимо редове који испуњавају услов, а у сваком реду бројимо елементе веће од 9.

Формираћемо двоструку петљу: спољна петља ће ићи по утакмицама (тј. редовима улаза), а унутрашња по категоријама које се прате на свакој утакмици. Бројач трипл-даблова иницијализујемо пре спољне петље, а бројач двоцифрених учинака пре унутрашње.

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

int main()
{
    int n, brTriplDabl = 0;
    int a[5];
    cin >> n;
    for (int utak = 0; utak < n; utak++)
    {
        cin >> a[0] >> a[1] >> a[2] >> a[3] >> a[4];
        int brDvoc = 0;
        for (int kateg = 0; kateg < n; kateg++)
            if (a[kateg] > 0)
                brDvoc++;

        if (brDvoc >= 3)
            brTriplDabl++;
    }
    cout << brTriplDabl << endl;
    return 0;
}
```

Задатак: Близанци

Поставка: Марија и Петар су близанци и желимо да свакоме од њих двоје купимо по једно одело као поклон за рођендан, али тако да се цене та два поклона што мање разликују (при томе није битно чији поклон ће бити скупљи).

Написати програм који учитава цене свих женских одела и свих мушких одела, а одређује и исписује најмању разлику између цена женског и мушког одела.

Улаз: Опис улаза: са стандардног улаза се учитава:

- у првом реду број мушких одела m ($1 \leq m \leq 50000$),
- у другом реду m целих бројева (цели бројеви између 1 и $2 \cdot 10^9$ раздвојени по једним размаком) - цене мушких одела
- у трећем реду број женских одела z ($1 \leq z \leq 50000$)
- и у четвртом реду z целих бројева (цели бројеви између 1 и $2 \cdot 10^9$ раздвојени по једним размаком) - цене женских одела.

Излаз: На стандардни излаз исписати најмању вредност разлике цена мушког и женског одела.

Пример

Улаз	*Излаз*
5 4680 2120 7940 11530 17820 4 850 13420 5770 6300	1090

Објашњење

Најмања разлика се постиже када се купе одела чије су цене 4680 и 5770 динара.

Решење:

Анализа

Наивно решење

Један могући приступ је да одредимо разлику (прецизније, апсолутну вредност разлике) у цени између сваког мушког и сваког женског одела, па од тих разлика нађемо најмању. Овакво решење ће дати тачан резултат у мањим примерима, али на већим примерима се неће извршити на време.

```
#include <iostream>
#include <vector>
#include <cmath>
#include <limits>
using namespace std;

int main() {
  ios_base::sync_with_stdio(false);
  int n1; cin >> n1;
```

```
  vector<int> a1(n1);
  for (int i = 0; i < n1; i++)
    cin >> a1[i];
  int n2; cin >> n2;
  vector<int> a2(n2);
  for (int i = 0; i < n2; i++)
    cin >> a2[i];

  int minRazlika = numeric_limits<int>::max();
  for (int i1 = 0; i1 < n1; i1++)
    for (int i2 = 0; i2 < n2; i2++)
      minRazlika = min(minRazlika, abs(a1[i1] - a2[i2]));

  cout << minRazlika << endl;

  return 0;
}
```

Упоредни пролаз кроз уређене низове

Ефикаснији приступ је да се низови цена најпре сортирају, а да се затим истовремено пролази кроз оба низа, рачунајући разлику текућих елемената и напредујући у оном низу у којем је цена тренутно мања. Успут се, наравно, по потреби ажурира најмања забележена разлика. Када се стигне до краја било којег низа, поступак је завршен и најмања забележена разлика је тада и укупно најмања.

Заиста, пошто су низови сортирани, када се упореде почетни елементи из оба низа, онај који је мањи од њих нема потребе упоређивати са осталим елементима низа коме он не припада, јер ће разлика моћи бити само већа (јер је тај низ сортиран). Тај елемент онда можемо избацити из даљег разматрања тако што ћемо у низу у ком се он налази прећи на следећи елемент. У специјалном случају када су почетни елементи оба низа једнаки, разлика је једнака нули, што је најмања могућа разлика, па нема потребе вршити даљу анализу.

На пример, нека су након сортирања вредности једнаке следећим.

```
1 14 28 33 45
8 21 22 41 56 68
```

- Прво поредимо елементе 1 и 8. Разлика је 7. Разлика између броја 1 и свих даљих бројева у доњем низу је већа од 7, па број 1 не морамо више анализирати.
- Након тога поредимо бројеве 14 и 8 и добијамо разлику 6. Разлика између броја 8 и свих бројева иза 14 је већа, па сада ни 8 не морамо више анализирати.
- Поредимо сада бројеве 14 и 21, разлика је 7, а 14 не морамо више да анализирамо.
- И разлика између 28 и 21 је 7, а број 21 не морамо више да анализирамо.
- Разлика између 28 и 22 је 6, а 22 не морамо да анализирамо даље.
- Разлика између 28 и 41 је 13, а 28 не морамо да анализирамо даље.

- Разлика између 33 и 41 је 8, а 33 не морамо да анализирамо даље.
- Разлика између 45 и 41 је 4, а 41 не морамо да анализирамо даље.
- Разлика између 45 и 56 је 11, а 45 не морамо да анализирамо даље. Пошто нема више елемената у горњем низу, поступак се завршава.

Можемо закључити да је најмања могућа разлика једнака 4 (за бројеве 41 и 45).

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <limits>

using namespace std;

int main() {
  ios_base::sync_with_stdio(false);
  int n1; cin >> n1;
  vector<int> a1(n1);
  for (int i = 0; i < n1; i++)
    cin >> a1[i];
  int n2; cin >> n2;
  vector<int> a2(n2);
  for (int i = 0; i < n2; i++)
    cin >> a2[i];

  sort(begin(a1), end(a1));
  sort(begin(a2), end(a2));
  int i1 = 0, i2 = 0;

  int minRazlika = numeric_limits<int>::max();
  while (i1 < n1 && i2 < n2)
    if (a1[i1] <= a2[i2]) {
      minRazlika = min(minRazlika, a2[i2] - a1[i1]);
      i1++;
    } else {
      minRazlika = min(minRazlika, a1[i1] - a2[i2]);
      i2++;
    }

  cout << minRazlika << endl;

  return 0;
}
```

Задатак: Поклони за родитеље

Поставка: Ненад жели да са путовања донесе поклоне својим родитељима. Пошто путује нискотарифном авио-компанијом и не жели да плаћа додатне трошкове, ограничена је маса коју може да понесе у свом ранцу. Оцу ће купити поклон у једној, а мајци у другој продавници. Написати програм који на основу познатих маса и цена свих поклона у обе продавнице помаже Ненаду да сваком од родитеља купи по један поклон, тако да поклони збирно буду што вреднији и да заједно не прелазе дато ограничење масе.

Улаз: са стандардног улаза се уносе следећи подаци: У првом реду број N_1 ($1 \leq N_1 \leq 10^5$), број производа у првој продавници. У сваком од наредних N_1 редова по два цела броја, сваки између 1 и 10^9, који представљају масу и затим цену једног производа из прве продавнице. У следећем реду број N_2 ($1 \leq N_2 \leq 10^5$), број производа у другој продавници. У сваком од наредних N_2 редова по два цела броја, сваки између 1 и 10^9, који представљају масу и затим цену једног производа из друге продавнице.

У следећем и последњем реду цео број између 1 и $2 \cdot 10^9$, највећа дозвољена маса за пар поклона заједно.

Излаз: На стандардни излаз исписати највећи могући збир цена првог и другог поклона које је могуће понети.

Пример

Улаз	*Излаз*
3 2 4 3 6 4 5 3 2 3 3 2 4 5 6	9

Решење:

Наивно решење

Можемо да проверимо сваки пар предмета (у коме је један предмет из једне, а други из друге продавнице). Ово решење је квадратне сложености, па може да донесе поене само за мале примере.

```
#include <iostream>
#include <vector>
#include <utility>
using namespace std;

typedef pair<int, int> Proizvod;
vector<Proizvod> ucitajProizvode() {
  int n;
  cin >> n;
```

```
  vector<pair<int, int>> proizvodi(n);
  for (int i = 0; i < n; i++) {
    int cena, tezina;
    cin >> cena >> tezina;
    proizvodi[i] = make_pair(cena, tezina);
  }
  return proizvodi;
}

int main() {
  vector<Proizvod> proizvodi1 = ucitajProizvode();
  vector<Proizvod> proizvodi2 = ucitajProizvode();
  int maksTezina;
  cin >> maksTezina;

  int maksCena = 0;
  for (const auto& p1 : proizvodi1) {
    if (p1.first >= maksTezina) continue;
    for (const auto& p2 : proizvodi2)
      if (p1.first + p2.first <= maksTezina &&
          p1.second + p2.second > maksCena)
        maksCena = p1.second + p2.second;
  }
  cout << maksCena << endl;
}
```

Нагађање

Такмичар који не уме да реши задатак, може да покуша да "упеца" неки број поена тако што напише једноставнији програм, који у суштини нагађа резултат. Нагађање може бити мање или више успешно.

Један могући покушај је да се нађе највећа цена из сваке продавнице и да се испише збир тих највећих цена.

Овај покушај нагађања се може комбиновати са наивним (неефикасним) решењем, тако да за довољно мале дужине низова задатак решавамо егзактно квадратним алгоритмом, а за велике примере покушамо да погодимо решење (што обично не даје добре резултате).

```
#include <iostream>

using namespace std;

int main() {
  int n1;
  cin >> n1;
  int maks1 = 0;
  for (int i = 0; i < n1; i++) {
    int tezina, cena;
```

```
    cin >> tezina >> cena;
    if (cena > maks1)
      maks1 = cena;
  }
  int n2;
  cin >> n2;
  int maks2 = 0;
  for (int i = 0; i < n2; i++) {
    int tezina, cena;
    cin >> tezina >> cena;
    if (cena > maks2)
      maks2 = cena;
  }

  cout << maks1 + maks2 << endl;

  return 0;
}
```

Сортирање и максимуми префикса

Решење које претендује на максималан број поена треба да буде ефикасније од квадратног. У ту сврху увек је корисно да подаци буду на неки начин сортирани. Питање је: како сортирати?

Ако оба низа уредимо тако да масе расту, можемо у једном низу да кренемо од почетка (од најлакшег предмета) а у другом од краја. На тај начин брзо долазимо до парова предмета који су у збиру испод лимита масе, а при томе најближи том лимиту. Ипак, ово није довољно да се задатак реши ефикасно, јер масе и цене не расту заједно, па предмет мање масе може бити вреднији. То значи да при прегледању свих парова који су кандидати за најбољи пар не бисмо избегли квадратно време.

Уређивање по цени није боље, јер бисмо тада за фиксиран предмет из једне продавнице морали међу онима из друге (који у збиру са првим) поправљу укупну вредност, да тражимо оне који се уклапају по маси и опет спадамо на квадратно време.

Оно што би нам овде помогло (након сортирања оба низа предмета по маси) је да можемо за сваки предмет P из једног низа да брзо одговоримо колико вреди највреднији предмет тог низа, чија маса није већа од масе предмета P. Такве одговоре можемо да припремимо након сортирања а пре испитивања парова, тако што фромирамо још један низ у коме ће се ти одговори наћи. Конкретније, можемо у једном пролазу кроз низ сортиран по маси да за сваки префикс тог низа израчунамо цену највреднијег предмета у префиксу. Након тога првобитна идеја о истовременом проласку кроз један низ од почетка а кроз други од краја доводи до ефикасног решења.

Прикажимо рад овог алгоритма на једном примеру. Нека су цене и тежине мушких поклона дате у левој, а женских у десној колони, при чему смо поклоне сортирали по тежинама и нека је капацитет ранца једнак 20.

```
4   3              5   6          6
7   9              9   4          6
```

```
12  4                10  12        12
16  11               14  9         12
19  2                18  7         12
```

Покушавамо да за први мушки поклон (то је (4, 3)) пронађемо скуп поклона који се могу са њим упарити. Пошто је други низ сортиран, то ће бити неки поклони који се налазе на почетку тог низа. Крећемо од краја, елиминишемо последњи женски поклон јер је збир 4 + 18 већи од 20 и проналазимо да је последњи женски поклон који се може упарити са првим мушким онај који има масу 14. Потребно је пронаћи најскупљи женски поклон који има масу мању или једнаку 14. У томе нам помаже последња колона у којој су записани максимуми префикса. За поклон чија је маса 14 можемо очитати вредност 12, што значи да постоји неки женски поклон чија је маса мања или једнака 14 који има вредност 12 (то је поклон (10, 12)). Дакле, ако купимо први мушки поклон, тада је највећа цена коју можемо постићи једнака 3 + 12 = 15.

Прелазимо на други мушки поклон (то је (7, 9)) и одређујемо женске поклоне са којима се он може комбиновати. Веома важна напомена је то да се поклони који се нису могли укомбиновати са претходним мушким поклонима, не могу укомбиновати ни са текућим (јер је он још тежи од претходних). Дакле, довољно је само међу оним поклонима који су се могли укомбиновати са претходним поклоном наћи оне који се могу укомбиновати са текућим. Пошто је низ женских поклона сортиран по тежини, анализирамо и евентуално елиминишемо елементе са његовог краја (тј. краја оног дела низа где смо се претходно зауставили). Поклон масе 14 се не може комбиновати са поклоном масе 7 (јер им је укупна маса 21 већа од носивости ранца 20). Најтежи женски поклон који се може упарити са оним мушким масе 7 је онај чија је маса 10. Поново на основу низа максимума префикса очитавамо да је највреднији женски поклон чија маса не прелази 10 онај чија је вредност 12 (то је опет (10, 12)), па се његовим комбиновањем са мушким поклоном (7, 9) добија вредност 12 + 9 = 21, што је боље од претходне.

Прелазимо на следећи мушки поклон (то је (12, 4)), елиминишемо женске поклоне (10, 12) и (9, 4) јер се они не могу комбиновати са мушким поклоном масе 12 и закључујемо да је једини женски поклон који се може комбиновати са (12, 4) поклон (5, 6). Тиме добијамо вредност 4+6 = 10, што је лошије од претходне.

Преласком на следећи мушки поклон (то је (16, 11)), елиминишемо и женски поклон (5, 6) и закључујемо да се ни један мушки поклон који је тежак 16 или више не може укомбиновати ни са једним женским поклоном.

Дакле, оптимално упаривање је упаривање поклона (7, 9) и (10, 12).

```
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;


typedef pair<int, int> Proizvod;
```

```
vector<Proizvod> ucitajProizvode() {
  int n;
  cin >> n;
  vector<pair<int, int>> proizvodi(n);
  for (int i = 0; i < n; i++) {
    int cena, tezina;
    cin >> cena >> tezina;
    proizvodi[i] = make_pair(cena, tezina);
  }
  return proizvodi;
}

int main() {
  vector<Proizvod> proizvodi1 = ucitajProizvode();
  vector<Proizvod> proizvodi2 = ucitajProizvode();
  int maksTezina;
  cin >> maksTezina;

  sort(begin(proizvodi1), end(proizvodi1));
  sort(begin(proizvodi2), end(proizvodi2));
  int n1 = proizvodi1.size(), n2 = proizvodi2.size();

  vector<int> maksCenaDo2(n2 + 1);
  maksCenaDo2[0] = 0;
  for (int i = 1; i <= n2; i++)
    maksCenaDo2[i] = max(maksCenaDo2[i-1], proizvodi2[i-1].second);

  int maksCena = 0;
  int i = 0, j = n2-1;
  while (i < n1 && proizvodi1[i].first < maksTezina) {
    while (j >= 0 && proizvodi1[i].first + proizvodi2[j].first > maksTezina)
      j--;
    if (j >= 0)
      maksCena = max(maksCena, proizvodi1[i].second + maksCenaDo2[j+1]);
    i++;
  }
  cout << maksCena << endl;

  return 0;
}
```

Сортирање, максимуми префикса и бинарна претрага

Претходна идеја се може мало и изменити. Довољно је да сортирамо само други низ и израчунамо максимуме његових префикса, као у претходном решењу. Након тога можемо за сваки предмет неуређеног првог низа да бинарном претрагом нађемо предмет највеће масе из другог низа, који се може понети заједно са првим, а онда (користећи максимуме префикса другог низа) да нађемо и вредност највреднијег предмета у

другом низу, који се може понети заједно са текућим предметом из првог низа.

```
#include <iostream>
#include <utility>
#include <algorithm>
using namespace std;

typedef pair<int, int> Proizvod;

vector<Proizvod> ucitajProizvode() {
  int n;
  cin >> n;
  vector<pair<int, int>> proizvodi(n);
  for (int i = 0; i < n; i++) {
    int cena, tezina;
    cin >> cena >> tezina;
    proizvodi[i] = make_pair(cena, tezina);
  }
  return proizvodi;
}

int main() {
  // ucitavamo podatke o proizvodima i maksimalnu tezinu koja moze da
  // stane u ranac
  vector<Proizvod> proizvodi1 = ucitajProizvode();
  vector<Proizvod> proizvodi2 = ucitajProizvode();
  int maksTezina;
  cin >> maksTezina;

  // sortiramo proizvode iz druge prodavnice po tezini
  sort(begin(proizvodi2), end(proizvodi2));

  // za svaku poziciju u sortiranom nizu proizvoda iz druge prodavnice
  // odredjujemo maksimalnu cenu proizvoda striktno pre te pozicije
  vector<int> maksCenaDo(proizvodi2.size() + 1);
  maksCenaDo[0] = 0;
  for (size_t i = 1; i <= proizvodi2.size(); i++)
    maksCenaDo[i] = max(maksCenaDo[i-1], proizvodi2[i-1].second);

  // maksimalni zbir cena dva proizvoda
  int maksCena = 0;
  // analiziramo jedan po jedan proizvod iz prve prodavnice
  for (const auto& p1 : proizvodi1) {
    // odredjujemo najvecu poziciju pre koje se svi proizvodi iz druge
    // pozicije mogu staviti u ranac sa tekucim proizvodom iz prve
    // prodavnice
    auto p2Granica = upper_bound(begin(proizvodi2), end(proizvodi2),
                                 maksTezina - p1.first,
```

```
                                [](int tezina, const Proizvod& pj) {
                                  return tezina < pj.first;
                                });
    int pj = distance(begin(proizvodi2), p2Granica);
    // ako ima bar neki takav proizvod
    if (pj > 0)
      // kombinujemo tekuci proizvod iz prve prodavnice sa najskupljim
      // proizvodom iz druge pre te pozicije (on se sigurno moze
      // iskombinovati sa tekucim proizvodom iz prve prodavnice)
      maksCena = max(maksCena, p1.second + maksCenaDo[pj]);
  }

  // ispisujemo pronadjeni zbir
  cout << maksCena << endl;

  return 0;
}
```