## Проблем 4: ZiPV™

Временско ограничење: 1 секунда
Меморијско ограничење: 512 MB

## Текст проблема

Још у 1984. години (када Интернет каквим га данас знамо није ни постојао), Грамп (F.T. Grampp) и Морис (R.H. Morris) су дали чувени цитат: "Лако је имати безбедан рачунарски систем. *Само* треба пресећи све везе са другим рачунарима, дозволити само директне жичане терминале између рачунара, ставити машине у забрављену собу, и поставити чувара испред врата.''

Вођени тим цитатом, челници Уједињених Комисијских Држава су до 2061. године учинили да све везе које повезују спољни свет са њиховим машинама не могу да шаљу више од **тридесет 64-битних целих бројева одједном**, уз оглашавање опште узбуне уколико се пошаље више таквих пакета. Међутим, и поред тога су чувени хакери Перица и Давис успели да координираним нападом запоседну две државне машине.

Перица је успео да, на машини коју је запосео, добије приступ неким сликама (величине 100 x 100, само у нијансама сиве - вредност сваког пиксела је цео број између 0 и 255) које би могле да крију скривене шифре за лансирање нуклеарних ракета у себи. Желео би да пошаље слике Давису, међутим због ограничења преносних канала, може послати само тридесет 64-битних целих бројева-тако да су могући губици при слању. Замолио вас је за помоћ.

## Описи функција

Потребно је да имплементирате две функције:

*Encode(In[...][...], Code[...]),*

која претвара улазну слику (представљену као 100 x 100 матрицу 8-битних целих бројева) у низ од тридесет 64-битних целих бројева;

*Decode(Code[...], Out[...][...]),*

која на основу израчунатог низа треба да, што прецизније, реконструише оригиналну слику.

**Једини начин на који ове две функције могу разменити информације је низ** *Code*!

**При попуњавању, низ** *Code* **и матрицу** *Out* **индексирајте од 0!**

## Ограничења

- Све слике ће бити представљене као 100 x 100 матрице 8-битних вредности ([0, 255]);
- **Дужина кода је ограничена на тридесет 64-битних целих бројева**.

## Подзадаци и бодовање

Тест примери су подељени у 4 подзадатка од по 25 поена, који представљају различите врсте слика. За сваку слику, *укупна грешка* се рачуна као *сума квадратних одступања* улазне од излазне слике на свакој позицији:

$$err = \sum_{i=0}^{99}\sum_{j=0}^{99}(In[i][j] - Out[i][j])^2$$

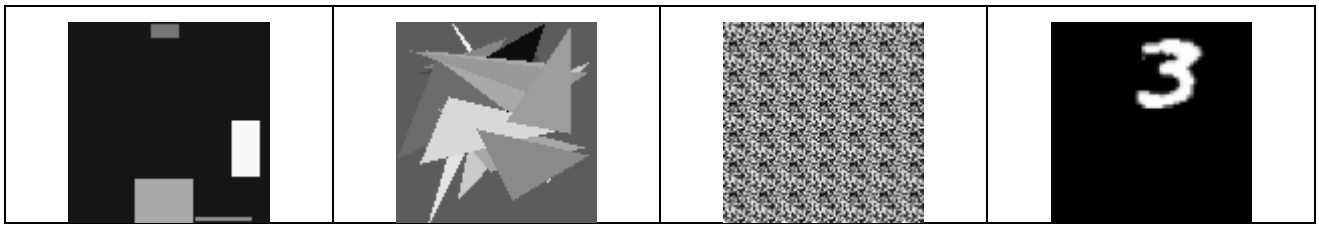
За сваку слику дефинисане су константе *A* и *B* тако да:

- Уколико важи $err \leq A$, освајате 25 поена за ту слику;
- Уколико важи $err \geq B$, освајате 0 поена за ту слику;
- Иначе, освајате $25 \times \frac{B - err}{B - A}$ поена за ту слику.

Број освојених поена на једном подзадатку је *минималан* број поена освојених на свим сликама унутар тог подзадатка.

## Тренинг подаци

Унутар сваког подзадатка, генерисано је **осам** слика на сличан начин. Од тог броја, **три** слике су насумично одабране и **директно су вам доступне у архиви приложеној уз задатак**. Преосталих пет слика ће бити коришћене за тестирање унутар сваког подзадатка. Слике су дате у два формата: у *Bitmap* формату (са екстензијом **.bmp**), као и текстуални фајл (са екстензијом **.in**) који у себи садржи одговарајућу 100 x 100 матрицу вредности пиксела. На слици испод се налази, за сваки подзадатак, по један пример од три која су вам дата у архиви.



## Детаљи имплементације

Потребно је да пошаљете тачно **два** фајла, под називима [**zipv_encode.c**; **zipv_decode.c**], [**zipv_encode.cpp; zipv_decode.cpp**] или [**zipv_encode.pas; zipv_decode.pas**], који имплементирају две горепомењуте функције. Осим тражених функција, ваши фајлови могу садржати и додатне глобалне променљиве, помоћне функције и додатне библиотеке. Међутим, имајте на уму да ће функције *Encode* и *Decode* да се позивају у одвојеним програмима, тако да *нећете моћи да сачувате икакву глобалну променљиву која није константна између позива ове две функције!*

Зависно од програмског језика који користите, ваше функције/процедуре морају бити следећег облика:

| | |
|---|---|
| C/C++ | void Encode(unsigned char **data_in, long long *code);<br>void Decode(long long *code, unsigned char **data_out) |
| Pascal | type matrix = array[0..99] of array[0..99] of byte;<br>procedure Encode(var data_in : matrix; var code : array of int64)<br>procedure Decode(var code : array of int64; var data_out : matrix) |

Уколико радите у C/C++-у, потребно је на почетку фајлова ставити **#include "zipv_encode.h"**, односно **#include "zipv_decode.h"**, а уколико радите у Pascal-у, потребно је на почетку фајла ставити **Unit zipv_encode;**, односно **Unit zipv_decode; (ово је већ додато у фајловима који су вам обезбеђени)**.

## Тестирање и експериментисање

Уз задатак, обезбеђени су вам **"template"** фајлови (**zipv_encode.c, zipv_decode.c, zipv_encode.cpp, zipv_decode.cpp, zipv_encode.pas, zipv_decode.pas**) које можете користити и мењати по потреби. Такође су вам обезбеђени програми (**manager.c, manager.cpp, manager.pas**) који служе да лакше тестирате кодове. Ови програми учитавају са стандардног улаза матрицу од 100 x 100 8-битних целих бројева (између 0 и 255). Затим позивају функцију **Encode** над том матрицом, а након тога позивају функцију **Decode** над добијеним низом, и коначно исписују на стандардни излаз добијену матрицу.

## Проблем 5: Метеори

Временско ограничење: 0.1 секунди
Меморијско ограничење: 256 MB

## Текст проблема

У ери брзих и драматичних дешавања у васиони се десила експлозија огромних размера. Она је изазвала померање огромног броја метеора. Научници покушавају да измоделирају дешавања у васиони. Положаје метеора представљају њиховим координатама у равни. Иако је за очекивати да се метеори крећу у огромном броју различитих праваца, сваки од њих се креће у једном од четири правца: на горе, на доле, удесно или улево. Сви се крећу константном брзином од једног метра по секунди.

За објашњавање догађања, научници су закључили да им треба конвексни омотач за скуп тачака које одговарају позицијама метеора по истеку $10^{1000}$ секунди од експлозије. За то им треба ваша помоћ.

## Описи функција

Потребно је имплементирати функцију

*KonveksniOmotac(N, x*[…]*,y*[…]*,d*[…]*)*

где је *N* - број метеора, а $x, y$ и $d$ низови који описују метеоре: $i$-ти метеор се налази на позицији $(x[i], y[i])$ и, уколико је $d[i] = 0$, он се креће на горе, уколико је $d[i] = 1$, он се креће удесно, уколико је $d[i] = 2$, он се креће на доле, а уколико је $d[i] = 3$ он се креће улево. **Сви низови су индексирани од 0**. Ова функција **мора да врати један цео број** – број темена на конвексном омотачу скупа тачака које представљају позиције метеора у тој далекој будућности. Оне тачке из тог скупа, које нису темена конвексног омотача, већ **се налазе на страницама полигона се не рачунају**.

## Пример:

Нека $N = 6$, $x = [1, 12, 7, 3, 5, 4]$, $y = [1, 2, 10, 4, 7, 8]$ и $d = [0, 2, 1, 2, 3, 0]$. Ови подаци означавају да имамо 6 метеора: први је на позицији $(1,1)$ и иде на горе, други је на позицији $(12,2)$ и иде на доле, трећи је на позицији $(7,10)$ и иде удесно, четврти на позицији (3,4) иде на доле, пети на позицији (5,7) иде улево и шести на позиџи (4,8) иде на горе. Оба метеора која иду на доле ће бити на конвексном омотачу. Метеор који креће са позиције (1,1) ће бити унутар омотача. Метеори који иду удесно и улево (по један) ће бити на конвексном омотачу. Тако је одговор 5.

## Ограничења

- $4 \leq N \leq 100.000$
- За свако $i = \overline{0, N-1}$ важи $-10^9 \leq x[i], y[i] \leq 10^9$ и $d[i] \in \{0, 1, 2, 3\}$
- Све координате су цели бројеви

## Подзадаци и бодовање

- **ПОДЗАДАТАК 1 [9 ПОЕНА]**: $N \leq 1.000$.
- **ПОДЗАДАТАК 2 [24 ПОЕНА]**: Сви метеори се крећу на горе или на доле (али бар по један у сваком од та два смера) тј. $d[i] = 0$ или $d[i] = 2$ за свако $i$.
- **ПОДЗАДАТАК 3 [18 ПОЕНА]**: Координате метеора су бројеви између -5000 и 5000.

- **ПОДЗАДАТАК 4 [49 ПОЕНА]**: Нема додатних ограничења.

## Детаљи имплементације

Потребно је да пошаљете тачно један фајл, под називом **meteori.c**, **meteori.cpp** или **meteori.pas**, који имплементира горе поменуту функцију. Осим тражене функције, ваш фајл може садржати и додатне глобалне променљиве, помоћне функције и додатне библиотеке.

Зависно од програмског језика који користите, ваша функција/процедура мора бити следећег облика:

| | |
|---|---|
| C/C++ | int KonveksniOmotac(int N, int* x, int* y, int* d); |
| Pascal | function KonveksniOmotac(N : longint; var x, y, s : array of longint) : longint; |

Уколико радите у C/C++-у, потребно је на почетку фајла ставити **#include "meteori.h"** а уколико радите у Pascal-у, потребно је на почетку фајла ставити **Unit meteori;** (**ово је већ додато у фајловима који су вам обезбеђени**).

## Тестирање и експериментисање

Уз задатак, обезбеђени су вам "template" фајлови (**meteori.c**, **meteori.cpp**, **meteori.pas**) које можете користити и мењати по потреби. Такође су вам обезбеђени програми (**grader.c**, **grader.cpp**, **grader.pas**) који служе да лакше тестирате кодове. Ови програми учитавају са стандардног улаза следеће податке:

- У првом реду број $N$
- У следећих $N$ редова бројеве $x[i], y[i], d[i]$, редом, раздвојене размаком

а затим позивају вашу функцију **KonveksniOmotac** из одговарајућег фајла (**meteori.c**, **meteori.cpp**, **meteori.pas**) са учитаним параметрима и на крају вредност коју ваша функција враћа исписују на стандардни излаз. Кодове ових програма можете мењати по потреби.

## Проблем 6: Свемир

| | |
|---|---|
| Временско ограничење: | 2 секунде |
| Меморијско ограничење: | 256 MB |

## Текст проблема

Средином 23. века људи са планете Земље су коначно развили технике међупланетарног путовања и кренули са освајањем свемира! Већ за две године су освојили $N$ планета, нумерисали их бројевима од $1$ до $N$ и направили свемирске тунеле између неких од њих! Сви тунели су **двосмерни** и систем тунела је направљен тако да **за сваке две планете постоји јединствен начин да се од једне дође до друге** користећи ове тунеле и неке успутне планете. Може се закључити да тих тунела има тачно $N - 1$ и да заједно са планетама образују структуру коју људи на Земљи називају **стаблом**. Ти тунели су заправо нека врста црних рупа и то нема пуно смисла али оно што је битно је чињеница да се овај задатак дешава у свемиру!

Осим што су у свемиру, ове планете имају и нека друга својства – за сваку планету је позната вредност $C_i$ која представља квалитет свемирских сарми на овим планетама; неке од ових вредности могу бити и негативне што је логично у свемиру. Познати крijумчар свемирских сарми Јурај Маргарин је сковао свемирски план – изабраће две (не нужно различите) планете $A$ и $B$ и трејдоваће сарме на **свим планетама (укључујући и ове две) које се налазе на (сетимо се) једниственом путу између $A$ и $B$.** Јурију је наравно циљ да **збир квалитета трејдованих сарми буде највећи могућ**.

Једини проблем у целој овој свемирској причи је што Маргарина (и све нас) види Свемирска полиција, светлост универзума. Прецизније, постоји тачно $K$ свемирских патрола – за сваку је позната почетна и крајња планета $P_i$ и $Q_i$ ($P_i \neq Q_i$) њиховог патролирања и та патрола обилази **све планете на јединственом путу између планета $P_i$ и $Q_i$ (укључујући и њих)**. Са ведрије стране, Јурај Маргарин има довољно сарми у свемирском штеку да подмити **највише једну свемирску патролу**. Према томе, Јурај мора **изабрати пут са највећим збиром квалитета сарми али такав да се на том путу он може сусрести са највише једном патролом (можда више пута)**. Помозите му!

## Описи функција

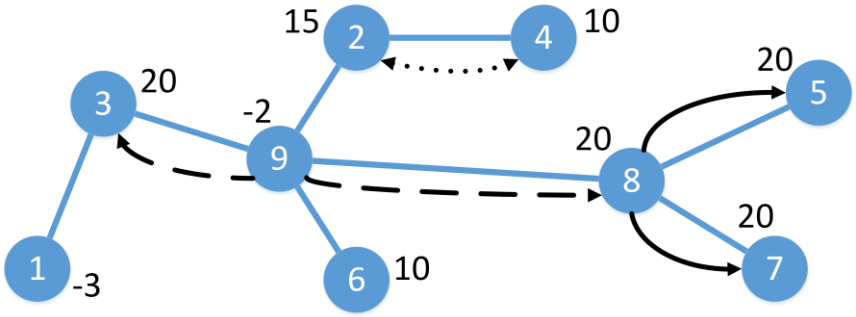
Потребно је да имплементирате функцију

$$SvemirskiPut(N, K, t[\dots], pat[\dots], c[\dots])$$

где је $N$ – број планета, $K$ – број свемирских патрола, $t$ низ дужине $2(N-1)$ који описују тунеле, $pat$ низ дужине $2K$ који описује патроле и $c$ је низ квалитета дужине $N$. У низу $t$ су, редом, наведени парови планета између којих постојоје тунели (постоји тунел између планета $t[0]$ и $t[1]$, између планета $t[2]$ и $t[3]$ и у општем случају између планета $t[2i]$ и $t[2i+1]$ за свако $i = \overline{0, N-2}$). На потпуно исти начин низ $pat$ описује парове планета између којих постоји свемирска патрола. На крају, за свако $i = \overline{0, N-1}$ квалитет сарми на планети $i+1$ је $c[i]$. **Сви низови су индексирани од 0**. Ова функција **мора да врати један цео број** – највећи могући збир квалитета сарми на Маргариновом пут који задовољава горе поменуте услове.

## Пример:

Нека је $N = 9, K = 3, t = [1, 3, 3, 9, 2, 4, 7, 8, 9, 8, 9, 6, 8, 5, 2, 9], pat = [4, 2, 3, 8, 7, 5]$ и $c =$ [-3, 15, 20, 10, 20, 10, 20, 20, -2]. Ови подаци описују систем планета и тунела као на слици: плави кругови представљају планете при чему су њихови редни бројеви унутар кругова а бројеви поред њих представљају квалитет њихових сарми; плаве линије предстаљају тунеле; остале линије представљају патроле – имамо 3 патроле, прва патролише између планета 2 и 4 (обилази обе), друга између планета 3 и 8 (обилази планете 3, 9 и 8) и трећа између планета 7 и 5 (обилази планете 5, 8 и 7). Јурај ће најбоље трејдовати сарме ако изабере планете 3 и 6 – тада подмићује само другу патролу а укупан квалитет сарми на путу од 3 до 6 једнак 20 + (-2) + 10 = 28; боље од тога не може и то је број који ваша функција треба да врати. Приметимо да Јурај нпр. **не може** изабрати пут између планета 3 и 4 или између 3 и 8 јер би се онда могао сусрести са две различите патроле.




## Ограничења

- $2 \leq N \leq 200.000$
- $1 \leq K \leq 200.000$
- За свако $0 \leq i < 2(N-1)$ важи $1 \leq v[i] \leq N$
- За свако $0 \leq i < 2K$ важи $1 \leq pat[i] \leq N$
- За свако $0 \leq i < K$ важи $pat[2i] \neq pat[2i+1]$
- За свако $0 \leq i < N$ важи $-10^9 \leq c[i] \leq 10^9$
- Улазни подаци ће бити такви да ће између сваке две планете постојати јединствен пут
- Улазни подаци ће бити такви да ће решење увек постојати

## Подзадаци и бодовање

- **ПОДЗАДАТАК 1 [15 ПОЕНА]**: $N, K \leq 1.000$
- **ПОДЗАДАТАК 2 [15 ПОЕНА]**: Планете су повезане на следећи начин: $1 - 2 - 3 - \cdots - n$
- **ПОДЗАДАТАК 3 [20 ПОЕНА]**: Све сарме у свемиру су неквалитетне тј. $c[i] < 0$ за свако $i$
- **ПОДЗАДАТАК 4 [20 ПОЕНА]**: Никоје две патроле не пролазе кроз исту планету
- **ПОДЗАДАТАК 5 [30 ПОЕНА]**: Нема додатних ограничења

## Детаљи имплементације

Потребно је да пошаљете тачно један фајл, под називом **svemir.c**, **svemir.cpp** или **svemir.pas**, који имплементира горе поменуту функцију. Осим тражене функције, ваш фајл може садржати и додатне глобалне променљиве, помоћне функције и додатне библиотеке.

Зависно од програмског језика који користите, ваша функција/процедура мора бити следећег облика:

| | |
|---|---|
| C/C++ | long long SvemirskiPut(int N, int K, int* t, int* pat, int* c); |
| Pascal | function SvemirskiPut(N, K : longint; var t, pat, c : array of longint) : int64; |

Уколико радите у C/C++-у, потребно је на почетку фајла ставити **#include "svemir.h"** а уколико радите у Pascal-у, потребно је на почетку фајла ставити **Unit svemir;** (**ово је већ додато у фајловима који су вам обезбеђени**).

## Тестирање и експериментисање

Уз задатак, обезбеђени су вам "template" фајлови (svemir.c, svemir.cpp, svemir.pas) које можете користити и мењати по потреби. Такође су вам обезбеђени програми (grader.c, grader.cpp, grader.pas) који служе да лакше тестирате кодове. Ови програми учитавају са стандардног улаза следеће податке:

- У првом реду бројеве $N$ и $K$, редом
- У наредном реду $2(N-1)$ бројева $t[i]$, раздвојених размаком
- У наредном реду $2K$ бројева $pat[i]$, раздвојених размаком
- У наредном реду $N$ бројева $c[i]$, раздвојених размаком

а затим позивају вашу функцију **SvemirskiPut** из одговарајућег фајла (svemir.c, svemir.cpp, svemir.pas) са учитаним параметрима и на крају вредност коју ваша функција враћа исписују на стандардни излаз. Кодове ових програма можете мењати по потреби.