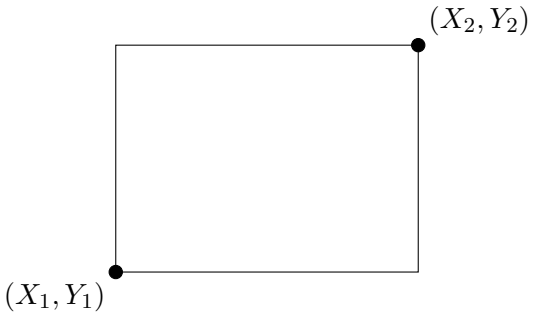
Decembrska tekma 2023
Rešitve nalog

1 Prva skupina

Naloge za prvo skupino ste reševali učenci, ki ste z nami prvo leto. Obsegale so vse tematike, ki smo jih do tekme obdelali.

1.1 Pravokotniki

Cilj naloge je bil, da izračunamo obseg in ploščino pravokotnika, podanega s koordinatami spodnjega levega in zgornjega desnega oglišča. Če si podatke iz vhoda skiciramo, dobimo nekaj takega:



Iz skice hitro vidimo, kako lahko izračunamo dolžine stranic. Dolžina zgornje in spodnje stranice je dana s predpisom $X_2 - X_1$, dolžina leve in desne pa s predpisom $Y_2 - Y_1$. V nalogi je bilo tudi zagotovljeno, da je $X_1 < X_2$ in $Y_1 < Y_2$ (oziroma rečeno drugače, da res dobimo koordinate spodnje leve in zgornje desne točke). Od tu naprej moramo le uporabiti formuli za ploščino in obseg pravokotnika, ki jih poznamo iz šolskega pouka matematike, in sicer

$$\text{obseg} = 2(X_2 - X_1) + 2(Y_2 - Y_1), \qquad \text{plošcina} = (X_2 - X_1)\dot{(}Y_2 - Y_1).$$

Koda celotne rešitve je napisana spodaj. Za dodatno vajo lahko premisliš, kako spremeniš kodo tako, da bo izračunala obseg in ploščino pravokotnika, ki je podan z zgornjim levim in spodnjim desnim oglišcem.

```c
#include <stdio.h>

int main() {
    int X1, X2, Y1, Y2;
    scanf("%d%d%d%d", &X1, &Y1, &X2, &Y2);
    int ploscina = 2 * (X2 - X1) + 2 * (Y2 - Y1);
    int obseg = (X2 - X1) * (Y2 - Y1);
    printf("%d %d\n", ploscina, obseg);
    return 0;
}
```

1.2 Poreden do božiča

Jošt bo darilo dobil samo, če bo priden na dan božiča, torej (po navodilu naloge) na šesti dan pred koncem leta. Poreden je vsak m-ti dan v letu, kjer je m zadnja števka v številu dni v letu N. Z nekaj računanja opazimo, da je Jošt poreden m-ti dan, $2m$-ti dan, $3m$-ti dan, itd. Zanima nas torej, če je število $N - 6$ večkratnik števila m; v tem primeru bo Jošt poreden na božič, v nasprotnem pa bo na božič priden, in bo dobil darilo. K sreči imamo na voljo posebno računsko operacijo, ki izračuna ostanek pri deljenju enega števila z drugim, v tem primeru ostanek pri deljenju $N - 6$ z m. Če bo ostanek enak 0, bo Jošt na božič poreden in ne bo dobil darila.

Preden lahko izračunamo ta ostanek, moramo poiskati število m, t.j. zadnjo števko v številu N. Matematika pove, da je ta števka ravno enaka ostanku pri deljenju števila N s številom 10. Edini posebni primer se pojavi, če je zadnja števka enaka 0; v tem primeru ne bomo mogli izračunati ostanka pri deljenju $N-6$ z m, ker deliti (in računati ostanka pri deljenju) z 0 ne smemo. V navodilo piše, da v tem primeru predpostavimo, da je Jošt celo leto priden, torej tudi na božič.

```c
#include <stdio.h>

int main() {
    int N;
    scanf("%d", &N);
    int m = N % 10;
    if (m == 0) {
        // Jošt je celo leto priden, torej je priden tudi za božič
        // in zato dobi darilo
        printf("DARILO\n");
    } else if ((N-6) % m == 0) {
        // Jošt je na božič poreden
        printf("NI DARILA\n");
    } else {
        // Jošt je nekatere dni poreden, vendar za božič ni,
        // torej prejme darilo
```

```
        printf("DARILO\n");
    }
    return 0;
}
```

Za dodatno vajo poskusi napisati program, ki podobno določi, če bo Jošt na božič priden ali ne, le da je m v tem primeru *prva* števka števila N, in ne zadnja.

1.3 Škrat Štefan

V nalogi moramo prešteti, koliko od danih N daril je oglatih (ta so označena z 1), in koliko je okroglih (ta so označena z 2). V prvi vrstici vhoda se nahaja število N, s katerim si lahko pomagamo, da vemo, koliko vrstic moramo prebrati. Hranili si bomo števec c, ki bo hranil število oglatih daril, videnih do sedaj. Ko bomo na koncu pregledali vsa darila, bo v c shranjeno število vseh oglatih daril. Ker se pojavljata samo dva tipa daril, lahko število okroglih daril izračunamo z razliko $N - c$.

```
#include <stdio.h>

int main() {
    int N, c=0;
    scanf("%d", &N);
    for (int i = 0; i < N; i++) {
        int darilo;
        scanf("%d", &darilo);
        if (darilo == 1) {
            c = c + 1;
        }
    }
    printf("%d\n%d\n", c, N-c);
    return 0;
}
```

Za dodatno vajo razmisli, kako lahko spremeniš to rešitev, da šteje število daril *treh* tipov, kjer tretji tip predstavimo s številko 3.

1.4 Kvartopirci

Vprašanje iz besedila naloge lahko prevedemo na vprašanje, kateri od nizov “AAAAB” ali “BAAAA” se pojavi v vsakem testnem primeru. Ker je vsak niz obrnjen bodisi pravilno bodisi napačno, je dovolj, da preverimo samo, če vsebuje podniz “AAAAB”, v nasprotnem primeru namreč vemo, da je niz obrnjen narobe.

To preverimo tako, da se z zanko zapeljemo od začetka niza do vključno pete črke od zadaj in preverimo, če se na tem mestu začne podniz “AAAAB”. Najlažji način za preverjanje slednjega je, da v pogojnem stavku skupaj nanizamo pet pogojev, vsak izmed katerih preveri eno črko.

```c
#include <stdio.h>

int main() {
    int n, k;
    scanf("%d%d", &n, &k);
    char zapisek[101];

    for (int i = 0; i < n; i++) {
        scanf("%s", zapisek);
        int pravilen = 0;
        for (int igra = 0; igra < k - 4; igra++) {
            if (zapisek[igra] == 'A' and
                zapisek[igra + 1] == 'A' and
                zapisek[igra + 2] == 'A' and
                zapisek[igra + 3] == 'A' and
                zapisek[igra + 4] == 'B') {
                pravilen = 1;
                break;
            }
        }

        if (pravilen == 1) {
            printf("pravilen\n");
        } else {
            printf("obrnjen\n");
        }
    }

    return 0;
}
```

2 Druga skupina

Naloge za drugo skupino ste reševali učenci, ki ste z nami drugo leto. Večino nalog lahko rešimo na več kot en način, zato nobena naloga ni specifično zahtevala znanj, ki smo jih pridobili letos; ste pa si s temi znanji lahko pomagali. Glede na nizko število rešitev so bile naloge malo pretežke.

2.1 Brinina domača naloga

Cilj naloge je, da za dan a poiščemo taki različni števili x in y, večji od a, katerih največji skupni delitelj ni 1, in katerih vsota je najmanjša možna. Tovrstnih nalog se najlažje

lotimo tako, da si na roke napišemo nekaj primerov. Rešimo nalogo za a med 1 in 10:

$$\begin{aligned}
a = 1 &: \quad x = 2, y = 4 \\
a = 2 &: \quad x = 2, y = 4 \\
a = 3 &: \quad x = 3, y = 6 \\
a = 4 &: \quad x = 4, y = 6 \\
a = 5 &: \quad x = 6, y = 8 \\
a = 6 &: \quad x = 6, y = 8 \\
a = 7 &: \quad x = 8, y = 10 \\
a = 8 &: \quad x = 8, y = 10 \\
a = 9 &: \quad x = 9, y = 12 \\
a = 10 &: \quad x = 10, y = 12
\end{aligned}$$

Glede na te rezultate se nam porodi ideja, da je smiselno ločeno obravnavati, če je število a liho ali sodo. Če je sodo, lahko za x uporabimo kar a, za y pa $a + 2$. To si lahko tudi dokažemo; recimo, da je optimalna rešitev x', y'. Če je $x' \geq a + 1$, tedaj mora veljati $y' > x'$, torej $y' \geq a + 2$. Tedaj je vsota $x' + y' \geq 2a + 3$, ampak za zgoraj definirana $x = a$ in $y = a + 2$ dobimo boljšo rešitev $x + y = 2a + 2$.

Kaj pa, če je a liho? Ena možnost je, da uporabimo $x = a+1$ in $y = a+3$, ter dobimo podoben rezultat kot v sodem primeru zgoraj. Ker je a liho, sta tako $a+1$ in $a+3$ sodi, torej deljivi z 2, zato je njun največji skupni delitelj vsaj 2. V zgornjem primeru pa ta konstrukcija deluje le za $a = 5$ in $a = 7$; za števili $a = 3$ in $a = 9$ dobimo drug rezultat. Obe ti števili sta deljivi s 3, in res, če v takem primeru uporabimo formuli $x = a$ in $y = a + 3$, dobimo boljši rezultat, ker je $2a + 3 < 2a + 4$. Za ostala liha števila pa ta formula ne deluje, ker ni nujno, da imata a in $a + 3$ kakšen skupni praštevilski faktor (oziroma skupni delitelj, večji od 1).

S temi tremi formulami lahko zapišemo program, ki izračuna odgovor. Pri tem moramo le biti pozorni na omejitve; velja $1 \leq a \leq 10^{18}$, torej moramo za števila uporabiti `long long`.

```
#include <stdio.h>
typedef long long ll;

int main() {
    int n;
    scanf("%d", &n);
    while (n-->0) {
        ll a;
        scanf("%lld", &a);
        if (a % 2 == 0) printf("%lld %lld\n", a, a+2);
        else if (a % 3 == 0) printf("%lld %lld\n", a, a+3);
        else printf("%lld %lld\n", a+1, a+3);
    }
```

```
    return 0;
}
```

2.2 Zlatolaskina šifra

V nalogi poskušamo ugotoviti Zlatolaskino prvotno sporočilo, ki ga je zakodirala z opisanim postopkom. V postopku kodiranja vsako črko v prvotnem sporočilu raztegnemo na M znakov tako, da med zaporednimi črkami v besedilu dodamo pike. Da pa to sporočilo ni samo M-krat daljše od originalnega, ciklično premikamo črke na indeksih, večjih od N, za N mest nazaj. Postopek ponavljamo, dokler nimajo vsi znaki indeksa, manjšega od N. Končno zakodirano sporočilo dobimo tako, da iz vsakega stolpca vzamemo eno črko, če v tem stolpcu je kakšna črka; sicer si črko izmislimo.

Poglejmo si primer kodiranja za $M = 5$ in $N = 7$. Zašifrirati želimo sporočilo `NAPOMOC`. Prvo dodamo $M - 1 = 4$ pike med vsak par črk, da dobimo razširjeno sporočilo `N....A....P....O....M....O....C`. Vsak znak od sedmega naprej zamikamo levo, dokler jih vseh ne poravnamo v N stolpcev, s čimer dobimo sledeč seznam:

```
N....A.
...P...
.O....M
....O..
..C
```

Sedaj iz vsakega stolpca preberemo eno črko (k sreči je v vsakem stolpcu ravno ena črka), in jih zapišemo v končno sporočilo `NOCPOAM`.

Cilj naloge je, da iz zakodiranega sporočila rekonstruiramo originalno za različne vrednosti M. To storimo tako, da preberemo vsako M-to črko (ker smo pri kodiranju zaporedne črke zamaknili za M), dokler ne pridemo nazaj do začetka; v besedilu naloge je zagotovljeno, da bo to zaustavitveni pogoj za dešifriranje. Prestavitev iz konca seznama na začetek tako obravnavamo kot en korak, kar lahko v programu dosežemo tako, da spremenljivko, ki nakazuje trenutno pozicijo, računamo po modulu N, t.j. dolžini seznama.

```
#include <stdio.h>
#include <string.h>

char zakodirano[1000001];

int main() {
    int q, m, len;
    scanf("%s %d", zakodirano, &q);
    len = strlen(zakodirano);
    while (q--) {
        // zanka se izvede q-krat -- vsakič obravnavamo en možen m
        scanf("%d", &m);
```

```
            printf("%c", zakodirano[0]);
            // Izpisujemo vsako m-to črko. Indeks računamo po modulu len, saj
            // želimo iz konca skočiti nazaj na začetek.
            for (int i = m % len; i != 0; i = (i + m) % len) {
                printf("%c", zakodirano[i]);
            }
            printf("\n");
        }
        return 0;
}
```

Premislimo še, kako lahko pri pošiljanju takega sporočila zagotovimo, da izberemo primerna N in M, da bo pri kodiranju vsak stolpec imel največ eno črko. Denimo, da želimo poslati sporočilo dolžine L. Opazimo lahko, da če vzamemo vsako M-to črko (ciklično) izmed N črk, bo $v(N, M)$-ta črka ravno tista, kjer bomo z M-to črko ponovno prišli na začetek (kjer $v(N, M)$ pomeni najmanjši skupni večkratnik). Ker koristimo vsako M-to črko izmed $v(N, M)$ črk to pomeni, da z izbranima N in M zakodiramo natanko $v(N, M)/M$ črk. Od tod sledi, da mora biti $L = v(N, M)/M = N/D(N, M)$.

Iz zadnjega člena vidimo, da si lahko vedno izberemo $N = L$ in nek M, ki je tuj N (torej tak M, da je $D(N, M) = 1$). Poljuben tak par N in M lahko še množimo s poljubnim naravnim številom, saj je $kN/D(kN, kM) = kN/(kD(N, M)) = N/D(N, M)$. Torej so pari kN in kM za $LN = D(N, M)$ in poljuben k iz naravnih števil edine možnosti.

2.3 Učinkovita dostava

V nalogi iščemo par nizov, ki imata najdaljšo skupno predpono. S pomočjo ocene časovne zahtevnosti ugotovimo, da lahko preverimo vsak par nizov posebej. Vsi nizi so dolgi dvajset znakov, za njihovo število pa velja $N \leq 10000$, torej je $O(20N^2) = O(N^2)$ rešitev, ki preveri vsak par nizov, popolnoma sprejemljiva.

Na koncu moramo izpisati niza, urejena leksikografsko. Ker sta niza gotovo iste dolžine (20), lahko to preprosto preverimo s funkcijo `strcmp`, ki vrne pozitivno število, če je prva črka, kjer se niza razlikujeta, večja v drugem nizu. Če ni, vrstni red nizov obrnemo.

```
#include <stdio.h>
#include <string.h>

char nizi[10002][22];

// funkcija izračuna dolžino najdaljše skupne predpone
// nizov na indeksih niz1 in niz2
int dolzina_predpone(int niz1, int niz2) {
    for (int i = 0; i < 20; i++) {
        if (nizi[niz1][i] != nizi[niz2][i]) return i;
    }
```

```
    return 20;
}

int main() {
    int N;
    scanf("%d", &N);
    for (int i = 0; i < N; i++) {
        scanf("%s", nizi[i]);
    }
    int naj_dolzina = 0;
    int naj_i, naj_j;
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            if (i == j) continue;
            int dolzina = dolzina_predpone(i, j);
            if (dolzina > naj_dolzina) {
                naj_dolzina = dolzina;
                naj_i = i;
                naj_j = j;
            }
        }
    }

    if (strcmp(nizi[naj_i], nizi[naj_j]) > 0) {
        int t = naj_i;
        naj_i = naj_j;
        naj_j = t;
    }
    printf("%s\n%s\n", nizi[naj_i], nizi[naj_j]);
    return 0;
}
```

Za dodatno vajo razmisli, kako lahko nalogo rešiš z višjimi omejitvami (recimo $N \leq 10^5$).

2.4 Pot ob žici

Cilj naloge je, da na krožni poti izračunamo najdaljšo razdaljo med dvema točkama, ki je daljša od K. Za izračun poti med dvema točkama porabimo $O(N)$ časa, torej za izračun dolžin poti med vsakim parom potrebujemo $O(N^3)$ operacij. Čas lahko spravimo na $O(N^2)$, če opazimo, da lahko pri izračunu razdalje od točke i do točke j uporabimo ravnokar izračunano razdaljo od točke i do točke $j-1$, ki ji le dodamo razdaljo med točkama $j-1$ in j. Pri premiku od točke i torej dobimo naraščajoče zaporedje razdalj, ki jim pravimo tudi *kumulativne vsote*.

V testnem primeru za $i = 1$ zaporedje kumulativnih vsot izgleda tako:

$$150, 470, 1140, 1360, 1470, 1520, 1720.$$

Če zaporedje nadaljujemo še v en krog, dobimo

$$150, 470, 1140, 1360, 1470, 1520, 1720, 1870, 2190, 2860, 3080, 3190, 3240, 3440.$$

Recimo, da v tem podaljšanem zaporedju pozabimo na prve tri člene in odštejemo razdaljo med prvo točko in četrto (1140). Dobimo

$$220, 330, 380, 580, 730, 1050, 1720, 1940, 2050, 2100, 2300,$$

kar je ravno zaporedje kumulativnih vsot, ki se začne pri $i = 4$. Med različnimi zaporedji lahko torej prehajamo tako, da začnemo drugje in od členov odštejemo pravilno razdaljo. Povedano drugače: razdaljo med četrto in sedmo točko lahko izračunamo tako, da od razdalje med prvo in sedmo točko odštejemo razdaljo med prvo in četrto točko.

Potrebujemo torej le eno zaporedje kumulativnih vsot; najlažje je, da vzamemo kar kumulativno vsoto od začetne točke 1. Če začnemo na točki i, iščemo prvo točko, kjer je kumulativna vsota večja ali enaka $d(1, i) + K$ (razdaljo med a in b označimo z $d(a, b)$). Ker je zaporedje kumulativnih vsot naraščajoče, lahko tako točko iščemo z bisekcijo. Edina težava se pojavi, če bomo morali na poti preiti točko 1, oziroma če je $d(1, i) + K$ večji od vsote števil $r_{i,i+1}$. V takem primeru prvo skočimo do točke 1, in od tam z bisekcijo iščemo manjše število (ker smo neko razdaljo že prepotovali).

```
#include <bits/stdc++.h>
using namespace std;

typedef long long ll;

// Podatki
int arr[100005];

// Skoraj kumulativne vsote; kum[i] je razdalja od 0 do i po definirani poti
ll kum[100005];

int main() {
    int N, K;
    kum[0] = 0;
    scanf("%d%d", &N, &K);
    for (int i = 0; i < N; i++) {
        scanf("%d", &arr[i]);
        if (i > 0)
            kum[i] = kum[i-1] + arr[i-1];
    }
```

```
int best_start, best_end;
ll best_dist = kum[N-1] + arr[N-1] + 100;  // = inf

// Za vsako postajo pogledamo, kje bi morali končati, če bi začeli pri njej
for (int i = 0; i < N; i++) {
    // Ni nujno, da bomo lahko od te postaje do (N-1)-te zbrali dovoljšnjo
    // razdaljo, zato se bomo morda morali sprehoditi do konca, in potem od
    // začetka naprej

    int end_idx;
    ll dist;

    // Če ni treba ciklati, odgovor samo poiščemo z bisekcijo
    if (kum[N-1] - kum[i] >= K) {
        end_idx = lower_bound(kum+i, kum+N, kum[i]+K) - kum;
        dist = kum[end_idx] - kum[i];
    }

    // Če moramo ciklati, vemo, da moramo nujno iti do konca, in nato še za
    // neko razdaljo od začetka
    else {
        ll ostanek = K - (kum[N-1] - kum[i]) - arr[N-1];

        // Če je dovolj že, da gremo do prve točke, bo ostanek negativen
        if (ostanek <= 0) {
            end_idx = 0;
            dist = K - ostanek;
            // kolikor smo v negativnem, toliko smo dodatno prehodili
        } else {
            // moramo prehoditi še nekaj razdalje
            // tu lahko iščemo do kum+i+1, ker je zagotovljeno, da je celoten
            // krog dovolj velik
            end_idx = lower_bound(kum, kum+i+1, ostanek) - kum;
            dist = kum[end_idx] + kum[N-1] - kum[i] + arr[N-1];
        }

    }

    if (dist < best_dist) {
        best_start = i;
        best_end = end_idx;
        best_dist = dist;
    }
```

```
    }
    printf("%d %d\n", best_start+1, best_end+1);
    return 0;
}
```

Za dodatno vajo razmisli, kako lahko nalogo rešiš brez biseckije, v času $O(N)$.

3 Tretja skupina

Tretja skupina je na tekmi imela dve nalogi iz druge skupine (Brinina domača naloga in pot ob žici), ter eno dodatno nalogo, številska igra.

3.1 Številska igra

V nalogi so podana števila n, a, in b. Cilj je, da s tremi definiranimi operacijami v čim manj korakih transformiramo število a v število b, brez da bi pri tem prekoračili število n. Dovoljene operacije so

- številu prištej 1,
- število deliš z enim od njegovih deliteljev,
- število množiš z enim od njegovih deliteljev.

Omejitve so $a, b, n \leq 10^5$, torej si lahko privoščimo $O(n \log n)$ ali $O(n\sqrt{n})$ operacij.

Števila med 1 in n si lahko predstavljamo kot vozlišča grafa, kjer dovoljene operacije predstavljajo (usmerjene) povezave. Tako v grafu obstajajo povezave $3 \to 4$, $8 \to 4$, $8 \to 32$, itd. Transformacija, ki v najmanj korakih spremeni a v b, bo ustrezala najkrajši poti med vozliščema a in b v tem grafu, to pot pa lahko poiščemo z iskanjem v širino. Če pri generiranju povezav uporabimo trik z iskanjem deliteljev do korena, bo celoten algoritem tekel v $O(n\sqrt{n})$, kar je ena od zgoraj dobljenih možnosti.

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;

ll n, a, b;
bool seen[100004];

void bfs() {
    queue<pair<ll, int>> q;
    q.push({a, 0});
    while (!q.empty()) {
        auto[t, dist] = q.front();
        q.pop();
```

```
        if (t > n || seen[t]) continue;
        seen[t] = true;

        if (t == b) {
            printf("%d\n", dist);
            return;
        }

        // operacija +1
        q.push({t+1, dist+1});

        // množenje in deljenje z delitelji
        for (ll d = 1; d*d <= t; d++) {
            if (t % d != 0) continue;
            q.push({t*d, dist+1});
            q.push({t/d, dist+1});

            // v vsaki iteraciji zanke najdemo dva delitelja
            ll d2 = t / d;
            q.push({t*d2, dist+1});
            q.push({t/d2, dist+1});
        }
    }
}

int main() {
    scanf("%lld%lld%lld", &n, &a, &b);
    bfs();
    return 0;
}
```