Садржај

Глава 1

Јун 2020.

Задатак: Светиљка

У једној улици налази се n кућа. Потребно је поставити једну светиљку тако да што више кућа буде обасјано. Јачина светиљке је d, што значи да ће бити обасјане само оне куће које су на удаљености мањој или једнакој d од светиљке (и лево и десно од ње). Улица је паралелна x-оси и свака кућа је одређена својом x координатом. Написати програм који реализује алгоритам за одређивање највећег броја кућа које се могу обасјати постављањем једне светиљке на улицу.

Улаз: Са стандардног улаза се учитавају број кућа n ($1 \leq n \leq 10^6$) и n целих бројева из интервала $[-10^9, 10^9]$ који представљају x координате кућа. Затим се учитава позитиван цео број d ($1 \leq d \leq 5 \cdot 10^8$) који представља јачину светиљке.

Излаз: На стандардни излаз исписати један цео број који представља максималан број кућа које могу бити обасјане једном светиљком.

Пример

Улаз	*Излаз*
6 29 -11 15 13 -68 -4 13	4

Објашњење

Постављањем светиљке у тачку са координатом 2 обасјавају се куће на координатама -11, 15, 13 и -4.

Решење

Груба сила

За сваку кућу x_i можемо израчунати број кућа које би биле осветљене када би она била та која је последња осветљена од свих кућа. То можемо израчунати тако што пребројимо све куће које су лево од ње, на растојању мањем од $2d$ у односу на њу (светиљка која је постављена у тачку $x_i - d$ осветљава интервал $[x_i - 2d, x_i]$). Пребројавање тих кућа можемо извршити грубом силом.

Анализа сложености. Алгоритам анализира n кућа и за сваку кућу x_i од њих поново анализира свих n кућа проверавајући да ли су лево од куће x_i и да ли су на растојању мањем од $2d$ у односу на њу (што се врши у константној сложености), па му је укупна сложеност $O(n^2)$.

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

using namespace std;

int main() {
  ios_base::sync_with_stdio(false);
```

```
  int n;
  cin >> n;

  vector<int> x(n);
  for(int i = 0; i < n; i++)
    cin >> x[i];

  int d;
  cin >> d;

  int maks = 0;
  for (int i = 0; i < n; i++) {
    int broj = 0;
    for (int j = 0; j < n; j++)
      if (x[j] <= x[i] && x[i] - x[j] <= 2*d)
        broj++;
    maks = max(maks, broj);
  }

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

Два показивача

Пошто светиљка осветљава d јединица на лево и d јединица на десно, све куће које су у интервалу ширине $2d$ могу бити осветљене истом светиљком. За сваку кућу x_i у сортираном низу кућа проверавамо да ли би могла да буде најдешња кућа која је осветљена светиљком (та светиљка може бити постављена у тачку $x_i - d$ и она осветљава интервал $[x_i - 2d, x_i]$). Под претпоставком да јесте, број осветљених кућа одређујемо тако што одредимо све куће које су лево од ње на растојању највише $2d$ од ње. Довољно је да знамо прву такву кућу x_j у сортираном низу кућа. Тада светиљка постављена у тачку $x_i - d$ осветљава $i - j + 1$ кућа (кућу x_i, x_j и све куће између њих).

Када са једне куће x_i пређемо на следећу $x_{i'} = x_{i+1}$, да бисмо одредили њој одговарајућу кућу $x_{j'}$, не морамо анализирати све куће из почетка, већ можемо искористити то што знамо да је x_j прва кућа која је на растојању мањем од или једнаком $2d$ у односу на кућу x_i. Ако она на растојању мањем од или једнаком $2d$ и у односу на светиљку x_{i+1}, тада смо сигурни да је то прва таква кућа (јер је, ако постоји, кућа x_{j-1} је на растојању већем од $2d$ у односу на кућу x_i, а растојање до куће x_{i+1} је још веће). Ако није, тада настављамо да анализирамо куће x_{j+1}, x_{j+2} итд. све док не дођемо до прве куће која је на растојању мањем или једнаком од $2d$ у односу на x_{i+1}.

Алгоритам, дакле, можемо имплементирати техником два показивача. Левим показивачем i обилазимо једну по једну кућу слева надесно, претпостављајући да је x_i последња осветљена кућа здесна. Десни показивач указује на прву кућу x_j слева која је на растојању мањем или једнаком од $2d$ у односу на x_i. Иницијално можемо поставити i и j на нулу и у петљи од 0 до n анализирати све куће x_i. Када год се i повећа, повећавамо и j све док је x_j на превеликом растојању у односу на x_i. Када одредимо прву кућу x_j која је на растојању мањем или једнаком од $2d$ у односу на x_i, тада израчуавамо $i - j + 1$ и ажурирамо максималан број кућа, ако је тај број већи од дотадашњег максимума.

Пример. Прикажимо рад алгоритма на једном примеру. Нека су положаји кућа након сортирања $-68, -11, -4, 13, 15, 29$ и нека је домет светиљке $d = 13$.

- Ако је последња осветљена кућа $x_i = x_0 = -68$, тада је она једина осветљена тј. $x_j = x_0 = -68$, па је $i - j + 1 = 1$.
- Ако је последња осветљена кућа $x_i = x_1 = -11$, тада је она једина осветљена тј. $x_j = x_1 = -11$ (j се увећава на 1, јер је растојање између кућа -68 и -11 веће од $2d = 26$), па је $i - j + 1 = 1$.
- Ако је последња осветљена кућа $x_i = x_2 = -4$, тада је прва кућа која је осветљена $x_j = x_1 = -11$ (j се не увећава, јер је растојање између кућа -11 и -4 мање од $2d = 26$), па је $i - j + 1 = 2$.
- Ако је последња осветљена кућа $x_i = x_3 = 13$, тада је прва кућа која је осветљена $x_j = x_1 = -11$ (j се не увећава, јер је растојање између кућа -11 и 13 мање од $2d = 26$), па је $i - j + 1 = 3$.

- Ако је последња осветљена кућа $x_i = x_4 = 15$, тада је прва кућа која је осветљена $x_j = x_1 = -11$ (j се не увећава, јер је растојање између кућа -11 и 15 једнако $2d = 26$), па је $i - j + 1 = 4$.
- Ако је последња осветљена кућа $x_i = x_5 = 29$, тада је прва кућа која је осветљена $x_j = x_3 = 13$ (j се прво увећава на 2, јер је растојање између кућа -11 и 29 веће од $2d = 26$, па се опет увећава на 3, јер је растојање између кућа -4 и 29 веће од $2d = 26$, након чега остаје 3, јер је растојање између кућа 13 и 29 мање од $2d = 26$), па је $i - j + 1 = 3$.

Максимална вредност $i - j + 1$ једнака је 4..

Анализа сложености. Иницијално сортирање кућа захтева време $O(n \log n)$. Након тога обилазак вршимо техником два показивача који се крећу у истом смеру и могу начинити највише $2n$ корака, при чему се у сваком кораку врши константан број операција, па је сложеност $O(n)$.

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

using namespace std;

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

  vector<int> x(n);
  for(int i = 0; i < n; i++)
    cin >> x[i];

  int d;
  cin >> d;

  sort(x.begin(), x.end());

  int maks = 0, j = 0;
  for (int i = 0; i < n; i++) {
    while(x[j] < x[i] - 2*d)
      j++;
    maks = max(maks, i - j + 1);
  }

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

Задатак: Зли учитељ

Зли учитељ Нави одлучио је да се освети својим ученицима за ометање часа. Следећи домаћи задатак који им буде задао радиће се у паровима. Учитељ за сваког ученика зна колико сати му је потребно да уради свој део задатка, а време потребно пару ученика да уради задатак је збир времена та два ученика. Учитељ жели да одреди колико највише сати може да зада за израду домаћег задатка (зато што жели да прикрије своје зле намере и да делује фер) тако да бар један пар ученика не може да стигне да уради задатак, без обзира на то како се ученици поделе у парове.

Улаз: Са стандардног улаза се учитава паран број ученика n ($1 \leq n \leq 10^6$). Затим се учитава n природних бројева који представљају бројеве дана потребних за израду задатка сваког ученика.

Излаз: На стандардни излаз исписати један природан број који представља број дана који учитељ треба да постави као рок за израду задатка.

Пример 1

Улаз	*Излаз*
6 5 7 2 6 4 6	10

Пример 2

Улаз	*Излаз*
6 2 4 1 16 7 11	16

Решење

Потребно је одредити упаривање ученика такво да је највеће време израде домаћег задатка најмање могуће. Другим речима, у низу бројева, потребно је пронаћи оно упаривање које даје најмањи могући (у односу на сва упаривања) максимални збир пара елемената (у односу на све парове). Учитељ може ученицима да остави рок који је за један мањи од тог минималног максималног збира и биће сигуран да неће сви ученици моћи да на време заврше домаћи (и то ће бити најдужи такав рок).

Минимални максимални збир пара можемо одредити грамзивим алгоритмом који упарује најспоријег и најбржег ученика и тиме своди проблем на проблем мање димензије (који се рекурзивно решава на исти начин). Када се исти принцип примени на преостале ученике, закључујемо да је један начин да добијемо оптимално упаривање да се други најбржи упари са другим најспоријим, трећи најбржи са трећим најспоријим и тако даље. Овим смо у потпуности одредили упаривање које може дати минималну максималну разлику и њену вредност одређујемо испитивањем збирова времена свих парова ученика.

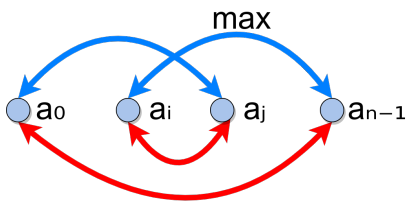
Имплементација се може направити итеративно. Прво сортирамо времена потребна да се уради домаћи неопадајуће. Након тога за свако $0 \leq i < \frac{n}{2}$ одређујемо збир $a_i + a_{n-1-i}$ и проналазимо максимум тих збирова.

```
int minimalniMaksimalniZbir(const vector<int>& a) {
  // pravimo sortiranu kopiju svih vremena
  auto as = a;
  sort(begin(as), end(as));
  // broj vremena
  int n = as.size();
  // minimalni maksimalni zbir se dobija u uparivanju u kojem je prvi
  // ucenik uparen sa poslednjim, drugi sa pretposlednjim itd.
  // odredjujemo maksimalni zbir nekog para u tom uparivanju
  int m = 0;
  for (int i = 0; i < n / 2; i++)
    m = max(m, as[i] + as[n - 1 - i]);

  return m;
}
```

Доказ коректности. Нека је низ $a_0, \ldots, a_{n-1}$ неопадајуће сортиран низ времена потребних да се уради домаћи. Тврдимо да постоји упаривање у ком се постиже минимални максимални збир, а у коме су најбржи и најспоријег ученик упарени.

Размотримо упаривање у коме је најспорији ученик a_{n-1} упарен са неким учеником a_i, али који није најбржи ($i > 0$). Тада је најбржи ученик a_0 упарен са неким учеником a_j, али који није најспорији ($j < n - 1$). Ако сада направимо размену тако да упаримо ученике a_0 и a_{n-1} (најбржег и најспоријег) и ученике a_i и a_j, максимум се може само смањити. Остали парови остају непромењени, па је потребно посматрати само упаривања ова 4 ученика.



Слика 1.1: Стање пре размене и након размене. Збир $a_i + a_{n-1}$ је већи или једнак од $a_0 + a_j$, $a_0 + a_{n-1}$ и $a_i + a_j$, тако да се након размене максимум не може повећати

Важи да је $\max(a_0 + a_j, a_i + a_{n-1}) = a_i + a_{n-1}$, јер је $a_0 \leq a_i$ и $a_j \leq a_{n-1}$. Пошто је $a_0 \leq a_i$, важи да је $a_0 + a_{n-1} \leq a_i + a_{n-1}$, а пошто је $a_j \leq a_{n-1}$, важи да је $a_i + a_j \leq a_i + a_{n-1}$. Дакле, оба броја

$a_0 + a_{n-1}$ и $a_i + a_j$ су мања или једнака од вредности $a_i + a_{n-1} = \max(a_0 + a_j, a_i + a_{n-1})$, па важи $\max(a_0 + a_{n-1}, a_i + a_j) \leq \max(a_0 + a_j, a_i + a_{n-1})$.

Дакле, ако се у неком оптималном упаривању изврши размена тако да се први и последњи елемент упаре, упаривање остаје оптимално (максимум се не може смањити, јер је полазно упаривање оптимално, а на основу доказаног не може се ни повећати). Према томе, постоји оптимално упаривање у коме су најбржи и најспорији ученик упарени.

Анализа сложености. Након сортирања које се врши у времену $O(n \log n)$, упаривање се одређује у времену $O(n)$.

Задатак: Мала игра

Задат је низ поља дужине n. У неким пољима је уписан по један позитиван цео број, док су остала поља празна. Потребно је попунити сва празна поља позитивним целим бројевима тако да је збир бројева у свака суседна два поља потпун квадрат. Написати програм који реализује алгоритам за одређивање лексикографски најмањег таквог решења, уколико решење постоји.

Улаз: Са стандардног улаза се учитавају дужина низа n ($1 \leq n \leq ?$) и максимални број који се може уписати у било које поље m ($1 \leq m \leq ?$). Затим се учитава n бројева уписаних у поља, при чему 0 означава празно поље.

Излаз: На стандардни излаз исписати лексикографски најмање решење игре уколико постоји, односно -1 уколико не постоји.

Пример

Улаз	*Излаз*
5 20 0 11 0 0 7	5 11 14 2 7

Решење

```
#include <iostream>
#include <vector>
#include <cmath>

using namespace std;

bool potpun_kvadrat(int x) {
  int koren = (int)round(sqrt(x));
  return koren * koren == x;
}

bool resi(vector<int>& v, int i, int m) {
  if (i == v.size())
    return true;

  if (v[i] > 0) {
    if (potpun_kvadrat(v[i] + v[i-1]))
      return resi(v, i + 1, m);
    else
      return false;
  }

  for (int j = (int)(sqrt(v[i-1])) + 1; j * j - v[i-1] <= m; j++) {
    v[i] = j * j - v[i-1];
    if (resi(v, i + 1, m))
      return true;
  }
  v[i] = 0;
```

```
  return false;
}

bool resi(vector<int>& v, int m) {
  if (v[0] > 0)
    return resi(v, 1, m);

  for (int j = 1; j <= m; j++) {
    v[0] = j;
    if (resi(v, 1, m))
      return true;
  }
  return false;

}

int main() {
  int n, m;
  cin >> n >> m;

  vector<int> v(n);
  for(int i = 0; i < n; i++)
    cin >> v[i];

  if (resi(v, m))
    for(int x : v)
      cout << x << ' ';
  else
    cout << -1;
  cout << '\n';

  return 0;
}
```

Глава 2

Септембар 2020.

Задатак: XOR

Задат је низ различитих неозначених целих бројева дужине n и неозначен цео број t. Потребно је одредити колико постоји парова бројева у низу таквих да је њихова ексклузивна дисјункција једнака управо броју t. Написати програм који реализује алгоритам за одређивање траженог броја парова. Временска сложеност алгоритма треба да буде $O(n \log n)$, а просторна $O(n)$.

За рачунање ексклузивне дисјункције у C++ користити бинарни оператор ^.

Улаз: Са стандардног улаза се учитавају дужина низа n ($n \leq 10^5$). Затим се учитава n неозначених целих бројева мањих од 2^30 који представљају елементе низа. На крају се учитава и број t ($0 < t < 2^30$).

Излаз: На стандардни излаз исписати један број који представља тражени број парова.

Пример

Улаз	*Излаз*
5 1 4 5 2 6 3	2

Објашњење

Постоје 2 таква пара: $12 = 3$ и $56 = 3$.

Решење

Груба сила

Задатак се може једноставно решити грубом силом, тако што се испитају сви парови, израчуна вредност њихове екслузивне дисјунције и то упореди са жељеним бројем t.

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

using namespace std;

int main() {
  int n;
  cin >> n;

  vector<int> v(n);
  for(int i = 0; i < n; i++)
    cin >> v[i];

  int t;
```

```
  cin >> t;

  int broj = 0;
  for (int i = 0; i < n; i++)
    for (int j = i + 1; j < n; j++)
      if ((v[i] ^ v[j]) == t) {
        broj++;
        break;
      }

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

Бинарна претрага

Обежелимо операцију ексклузивне дисјункције са $\oplus$. Лако се утврђује да је то асоцијативна операција и да за свако x важи $x \oplus x = 0$ и $0 \oplus x = x$. Зато, ако важи $x \oplus y = t$, важи и $x \oplus (x \oplus y) = x \oplus t$. Важи да је $x \oplus (x \oplus y) = (x \oplus x) \oplus y = 0 \oplus y = y$. Зато из $x \oplus y = t$ следи да је $y = x \oplus t$, што значи да за свако x у низу морамо проверити да ли низ садржи и вредност $x \oplus t$. Пошто је t различито од нуле, вредности x и $x \oplus t$ ће увек бити различите.

Проналажење одговарајућих парова ефикасно можемо урадити ако низ сортирамо и затим на њега применимо бинарну претрагу. Пошто ће се сваки пар пронаћи два пута (једном за први, а једном за други елемент пара), укупан број пронађених парова треба на крају поделити са 2.

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

using namespace std;

int main() {
  int n;
  cin >> n;

  vector<int> v(n);
  for(int i = 0; i < n; i++)
    cin >> v[i];

  int t;
  cin >> t;

  sort(v.begin(), v.end());

  int broj = 0;
  for(int i = 0; i < n; i++)
    if (binary_search(v.begin(), v.end(), t ^ v[i]))
      broj++;

  cout << broj / 2 << endl;
  return 0;
}
```

Задатак: Лексикографски наредна шетња

Шетња је низ бројева такав да је први елемент једнак 0 и да се свака два суседна елемента низа разликују за највише 1. Дата је шетња дужине n. Написати програм који реализује алгоритам за одређивање лексикографски наредне шетње. Временска и просторна сложеност алгоритма треба да буду $O(n)$.

Улаз: Са стандардног улаза се учитава број n ($n \leq 10^6$). Након тога се учитава n бројева који представљају елементе шетње.

Излаз: На стандардни излаз исписати n бројева који представљају наредну шетњу. Уколико таква шетња не постоји, исписати -1.

Пример

Улаз	*Излаз*
6 0 1 0 -1 -2 -1	0 1 0 -1 -1 -2

Решење

Алгоритам подсећа на алгоритам одређивања наредне варијације у лексикографском редоследу.

Анализирамо елемент по елемент низа, здесна налево и тражимо први елемент који је могуће повећати за један. Да би то могло да се уради мора да важи $a_j + 1 - a_{j-1} \leq 1$. Ако такав елемент не постоји (што ће се догодити у случају да је шетња облика $0, 1, 2, \ldots, n-1$), тада не постоји наредна варијација. У супротном увећавамо пронађени елемент a_j за један, а након њега правимо шетњу која се састоји од што мањих бројева (да би била лексикографски најмања), тако што иза a_j постављамо редом елементе $a_j - 1$, $a_j - 2$, ….

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

using namespace std;

bool naredna_setnja(vector<int>& a) {
  int n = a.size();
  int j = n-1;
  while (j > 0 && a[j] + 1 - a[j-1] > 1)
    j--;
  if (j == 0)
    return false;
  a[j]++;
  for (int k = j+1; k < n; k++)
    a[k] = a[k-1] - 1;
  return true;
}

int main() {
  ios_base::sync_with_stdio(false);

  int n;
  cin >> n;
  vector<int> a(n);
  for (int i = 0; i < n; i++)
    cin >> a[i];

  if (naredna_setnja(a)) {
    for (int i = 0; i < n; i++)
      cout << a[i] << " ";
    cout << endl;
  } else
    cout << "-1" << endl;

  return 0;
}
```

Задатак: Једнобојан квадрат

Марко је желео да купи једнобојан тепих у облику квадрата, али како су у радњи имали само шарене тепихе у облику правоугаоника одлучио је да купи један такав и да из њега исече један једнобојан квадратни део. Он жели да га исече тако да површина добијеног тепиха буде што већа. Тепих је задат матрицом димензија $n \times m$ при чему је у свако поље матрице уписан један број који представља боју тог поља. Написати програм који реализује алгоритам за одређивање дужине странице највећег једнобојног квадрата који се може пронаћи на датом тепиху. Временска и просторна сложеност алгоритма треба да буду $O(n \cdot m)$.

Улаз: Са стандардног улаза се учитавају бројеви n и m ($n, m \leq 1000$). Затим се учитава матрица бројева величине $n \times m$ при чему је сваки елемент матрице неозначен цео број мањи или једнак 10^9.

Излаз: На стандардни излаз исписати један број који представља дужину странице највећег једнобојног квадрата.

Пример

Улаз	*Излаз*
3 4 1 5 3 0 8 3 3 4 2 3 3 4	2

Објашњење: Највећи једнобојан квадрат је боје 3 и његова страница је дужине 2.

Решење

Опис главног решења

У овом блоку се описује главно решење задатка.

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

using namespace std;

int main() {
  int n, m;
  cin >> n >> m;

  vector< vector<int> > mat(n, vector<int>(m));
  for(int i = 0; i < n; i++)
    for(int j = 0; j < m; j++)
      cin >> mat[i][j];

  vector<vector<int>> dp(n, vector<int>(m));
  for (int i = 0; i < n; i++)
    dp[i][0] = 1;
  for (int j = 0; j < m; j++)
    dp[0][j] = 1;
  for (int i = 1; i < n; i++)
    for (int j = 1; j < m; j++)
      if (mat[i-1][j]   == mat[i][j] &&
          mat[i][j-1]   == mat[i][j] &&
          mat[i-1][j-1] == mat[i][j])
        dp[i][j] = 1 + min({dp[i-1][j], dp[i][j-1], dp[i-1][j-1]});
      else
        dp[i][j] = 1;

  int maxd = 0;
  for(int i = 0; i < n; i++)
```

```
    for(int j = 0; j < m; j++)
      maxd = max(maxd, dp[i][j]);

  cout << maxd << '\n';
  return 0;
}
```

Глава 3

Октобар 2020.

Задатак: Кајмак

Алекса жели да отпутује један дан до Златибора да купи кајмак. Он толико воли кајмак да ће сваки дан током свог боравка на Златибору купити сав кајмак са пијаце. Пошто путује аутомобилом у који може да смести строго мање од t килограма кајмака, на Златибору ће остати до оног дана када би куповина кајмака препунила ауто. Написати програм који одређује највећи броја килограма кајмака који Алекса може да купи.

Улаз: Са стандардног улаза се учитавају број n ($n \leq 10^6$) и након тога се учитава n бројева који представљају колико ће килограма кајмака бити на пијаци ког дана. Збир тих бројева није већи од 10^9. На крају се учитава број t ($t \leq 10^9$), који представља носивост аутомобила.

Излаз: На стандардни излаз исписати један број који представља колико највише килограма кајмака Алекса може да донесе са Златибора.

Пример

Улаз	*Излаз*
6 4 2 3 2 10 1 9	7

Решење

Груба сила

Задатак грубом силом решавамо тако што за сваку почетни дан одређујемо количину кајмака која се може купити ако би летовање кренуло баш тог дана. То радимо тако што сабирамо све количине кајмака, док се не достигне вредност t.

Анализа сложености. Сложеност овог решења је $O(n^2)$.

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

using namespace std;

int main() {
  ios_base::sync_with_stdio(false);

  int n;
  cin >> n;

  vector<int> v(n);
  for(int i = 0; i < n; i++)
    cin >> v[i];
```

```
  int t;
  cin >> t;

  int max_s = 0;
  for(int i = 0; i < n; i++) {
    int s = 0;
    for(int j = i; j < n; j++) {
      s += v[j];
      if (s < t)
        max_s = max(max_s, s);
      else
        break;
    }
  }

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

Два показивача

Задатак можемо решити ефикасно техником два показивача. Одржаваћемо текући распон дана $[j, i]$ и укупну количину кајмака у тим данима. Иницијално ћемо посматрати интервал $[0, 0]$. Док год је укупна количина кајмака у тим данима строго мања од t помераћемо десни показивач i и ажурираћемо максимум ако је то потребно. Када количина постане већа или једнака t, помераћемо леви показивач j све док се количина не спусти испод t или док j не достигне i.

Овим се врши значајно одсецање, јер се за многе потенцијалне почетке j не одређује која је максимална укупна количина кајмака била када би тај j био први дан. То је безбедно урадити, јер смо сигурни да се максимум не може достићи за тако елиминисане почетке. Наиме, у ситуацији када померамо j, знамо да је збир елемената из интервала $[j, i]$ већи или једнак t, док је збир елемената из интервала $[j, i-1]$ строго мањи од t (у супротном не би био померен десни показивач i). Када се j помери на позицију $+1$ збир свих елемената из интервала $[j+1, i]$ може бити и строго мањи, али и већи или једнак од t. Ако је строго мањи, тада га упоређујемо са максимумом и ажурирамо максимум ако је то потребно. Ако је већи или једнак, то значи да се максимум кајмака кренувши од дана $j+1$ постиже закључно са неким даном који је или $i-1$ или неки раније. Међутим, тај интервал је подинтервал интервала $[j, i-1]$, па му збир не може бити већи од збира елемената интервала $[j, i-1]$ који је раније обрађен. Стога тај збир не треба прецизно одређивати и може се одмах прећи на наредну вредност j.

Анализа сложености. Пошто се оба показивача померају само у једном смеру и пошто се у сваком кораку врши само константан број операција сложеност овог решења је $O(n)$.

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

using namespace std;

int main() {
  ios_base::sync_with_stdio(false);

  int n;
  cin >> n;

  vector<int> v(n);
  for(int i = 0; i < n; i++)
    cin >> v[i];

  int t;
  cin >> t;
```

```
  int s = 0, j = 0, maks = 0;
  for(int i = 0; i < n; i++) {
    s += v[i];
    while(j <= i && s >= t)
      s -= v[j++];
    maks = max(maks, s);
  }

  cout << maks << '\n';
  return 0;
}
```

Префиксни збирови и два показивача

Задатак се ефикасно може решити тако што се израчунају збирови свих префикса низа. Ако је Z_i збир првих i елемената низа, тада се збир елемената на позицијама из интервала $[i, j]$ може израчунати као $Z_{j+1} - Z_i$. Дакле, потребно је пронаћи највећу разлику два елемента низа која је строго мања од t. То је могуће ефикасно урадити техником два показивача. Пошто су елементи низа позитивни, низ збирова префикса је сортиран растући. Одржавамо два показивача i и j које иницијализујемо на 0. Када је разлика $Z_j - Z_i$ (а то је збир елемената интервала $[i, j-1]$ већи од или једнак t, померамо показивач i (то се не може догодити када је $i = j$, тј. када је интервал $[i, j-1]$ празан). У супротном ажурирамо максимум ако је то потребно и проширујемо интервал померајући десни показивач j.

Анализа сложености. Низ збирова префикса се формира у времену $O(n)$. Пошто се оба показивача померају само у једном смеру и пошто се у сваком кораку врши само константан број операција сложеност овог решења је $O(n)$.

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

using namespace std;

int main() {
  ios_base::sync_with_stdio(false);

  int n;
  cin >> n;

  vector<int> ps(n+1);
  ps[0] = 0;
  for(int i = 1; i <= n; i++) {
    int x;
    cin >> x;
    ps[i] = ps[i-1] + x;
  }

  int t;
  cin >> t;

  int maks = 0, i = 0, j = 0;
  while (j <= n) {
    if (ps[j] - ps[i] >= t) {
      i++;
    } else {
      maks = max(maks, ps[j] - ps[i]);
      j++;
    }
  }
```

```
  cout << maks << '\n';
  return 0;
}
```

Префиксни збирови и бинарна претрага

Задатак можемо решити тако што се за сваки леви крај i одреди највећи десни крај j такав да је збир елемената у интервалу $[i, j]$ строго мањи од t. Ако се формира низ збирова префикса (који је растући сортиран, јер су количине кајмака позитивне), задатак се своди на то да се за свако i одреди највеће вредности $j \geq i$ тако да је $z_j - z_j < t$, што се може урадити бинарном претрагом и проналажењем најмање вредности $j \geq i$ такве да је $z_j - z_i \geq t$ тј. да је $z_j \geq z_i + t$.

Анализа сложености. Низ збирова префикса се формира у времену $O(n)$. Након тога се врши n бинарних претрага, свака у сложености $O(\log n)$, па је укупна сложеност $O(n \log n)$.

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

using namespace std;

int main() {
  ios_base::sync_with_stdio(false);

  int n;
  cin >> n;

  vector<int> ps(n+1);
  ps[0] = 0;
  for(int i = 1; i <= n; i++) {
    int x;
    cin >> x;
    ps[i] = ps[i-1] + x;
  }

  int t;
  cin >> t;

  int maks = 0;
  for (int i = 0; i <= n; i++) {
    int psj = *prev(lower_bound(next(begin(ps), i), end(ps), ps[i] + t));
    maks = max(maks, psj - ps[i]);
  }

  cout << maks << '\n';
  return 0;
}
```

Задатак: Кретање низ матрицу

Дата је матрица целих бројева димензије $n \times m$. Могуће је започети кретање из било ког поља прве врсте матрице и завршити га у било ком пољу последње врсте. У сваком кораку је могуће прећи само на поље доле, доле-десно или доле-лево у односу на тренутно. Написати програм који реализује алгоритам за одређивање максималног збира бројева у пољима на путу који се може остварити током једне шетње. Временска и просторна сложеност алгоритма треба да буду $O(nm)$.

Улаз: Са стандардног улаза се учитавају бројеви n и m ($n, m \leq 1000$). Након тога се учитава матрица целих бројева димензије $n \times m$. Бројеви су мањи од 10^8.

Излаз: На стандардни излаз исписати један број који представља тражени максимални збир.

Пример

Улаз	*Излаз*
4 3 1 2 3 2 -4 2 3 0 -1 1 2 3	9

Објашњење

Највећи збир се остварује кретањем редом кроз друго поље прве врсте, прво поље друге врсте, прво поље треће врсте и коначно друго поље четврте врсте.

Решење

Задатак решавамо динамичким програмирањем.

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

using namespace std;

int main() {
  ios_base::sync_with_stdio(false);
  int n, m;
  cin >> n >> m;

  vector<vector<int>> mat(n, vector<int>(m));
  for (int i = 0; i < n; i++)
    for (int j = 0; j < m; j++)
      cin >> mat[i][j];

  vector<vector<int>> dp(mat);
  for (int i = 1; i < n; i++)
    for (int j = 0; j < m; j++) {
      int mp = dp[i - 1][j];
      if (j > 0)
        mp = max(mp, dp[i - 1][j - 1]);
      if (j < m - 1)
        mp = max(mp, dp[i - 1][j + 1]);
      dp[i][j] = mp + mat[i][j];
    }

  int res = *max_element(begin(dp[n - 1]), end(dp[n - 1]));
  cout << res << endl;

  return 0;
}
```

Задатак: Распоред са најмањим максималним збиром три узастопна

Ана и Бобан играју игру. Игра се игра са n папирића и на сваком је написан по један број. Ана почиње партију тако што папириће поређа у круг у произвољном редоследу. Бобан затим бира нека три узастопна папирића и рачуна збир бројева написаних на њима. Бобан осваја број бодова једнак том збиру и циљ му је да одабере три папирића тако да освоји што више бодова. Са друге стране Ана распоређује папириће тако да Бобан може да оствари што мање бодова. Написати програм који реализује алгоритам за одређивање броја бодова које ће Бобан освојити уколико оба играча играју савршено. Временска сложеност алгоритма треба да буде $O((n-1)!)$, а просторна $O(n)$.

Улаз: Са стандардног улаза се учитава број n ($n \leq 12$). Затим се учитава n различитих позитивних бројева поређаних у неопадајући поредак који представљају бројеве написане на папирићима. Бројеви нису већи од 10^8.

Излаз: На стандардни излаз исписати један број који представља максималан број бодова које ће Бобан освојити.

Пример

Улаз	*Излаз*
5 2 3 4 5 6	13

Објашњење

Ана може да распореди папириће на следећи начин: $[2, 6, 4, 3, 5]$. Бобан тада може на два начина да оствари збир 13, као $5 + 2 + 6$ и као $6 + 4 + 3$. Не постоји распоред у ком Бобан може максимално да скупи мање од 13 бодова.

Решење

Испитивање свих пермутација

Решење грубом силом подразумева да се испитају све пермутације и да се за сваку пермутацију испитају све тројке узастопних елемената.

Анализа сложености. Сложеност овог решења је $O(n \cdot n!)$.

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

using namespace std;

int main() {
  int n;
  cin >> n;

  vector<int> v(n);
  for(int i = 0; i < n; i++)
    cin >> v[i];

  int res = v[n - 1] + v[n - 2] + v[n - 3];
  do {
    int maxsum = 0;
    for (int i = 0; i < n; i++)
      maxsum = max(maxsum, v[i] + v[(i + 1) % n] + v[(i + 2) % n]);
    res = min(res, maxsum);
  } while (next_permutation(v.begin(), v.end()));

  cout << res << '\n';
  return 0;
}
```

Оптимизовано испитивање свих пермутација

Размотримо све пермутације 4 елемента.

```
1 2 3 4       2 1 3 4       3 1 2 4       4 1 2 3
1 2 4 3       2 1 4 3       3 1 4 2       4 1 3 2
1 3 2 4       2 3 1 4       3 2 1 4       4 2 1 3
1 3 4 2       2 3 4 1       3 2 4 1       4 2 3 1
1 4 2 3       2 4 1 3       3 4 1 2       4 3 1 2
1 4 3 2       2 4 3 1       3 4 2 1       4 3 2 1
```

Иако су све ове пермутације различите, сваки кружни распоред папирића је представљен са 4 различите пермутације. На пример, пермутације `1 2 3 4`, затим `2 3 4 1`, затим `3 4 1 2` и на крају `4 1 2 3` све представљају исти распоред папирића. Време се може уштедети ако се ово избегне и ако се за сваки распоред анализира само једна пермутација. То на пример увек може бити она која има број 1 на почетку.

Анализа сложености. Сложеност овог решења је $O(n \cdot (n-1)!)$. Да би се смањио константни фактор у имплементацији је пожељно избегавати целобројно дељење.

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

using namespace std;

int main() {
  int n;
  cin >> n;

  vector<int> v(n);
  for(int i = 0; i < n; i++)
    cin >> v[i];

  int res = v[n - 1] + v[n - 2] + v[n - 3];
  do {
    int maxsum = 0;
    for (int i = 0; i < n; i++) {
      int i1 = i + 1;
      if (i1 >= n)
        i1 -= n;
      int i2 = i + 2;
      if (i2 >= n)
        i2 -= n;
      maxsum = max(maxsum, v[i] + v[i1] + v[i2]);
    }
    res = min(res, maxsum);
  } while(next_permutation(v.begin() + 1, v.end()));

  cout << res << '\n';
  return 0;
}
```

Глава 4

Јануар 2021. - група А

Задатак: Не-нула збир

Написати програм који за унети низ целих бројева a дужине n одређује колико постоји парова позиција i и j ($i < j$) за које важи $a_i + a_j \neq 0$. Временска сложеност алгоритма треба да буде $O(n \log n)$, а просторна $O(n)$.

Улаз: Са стандардног улаза се уноси број n ($1 \leq n \leq 100000$), а затим и n целих бројева a_i ($-10^9 \leq a_i \leq -10^9$).

Излаз: На стаднардни излаз исписати тражени број парова. Користити 64-битни тип података.

Пример

Улаз	*Излаз*
4	4
3 4 -3 -3	

Објашњење: У питању су парови $(0, 1)$ $(3 + 4)$, $(1, 2)$ $(4 + (-3))$, $(1, 3)$ $(4 + (-3))$ и $(2, 3)$ $((-3) + (-3))$.

Решење

Уместо да одредимо колико има парова где збир није нула, лакше можемо да одредимо колико има парова где збир јесте нула. За сваки елемент $_i$ важи да је број позиција j пре њега таквих да је $a_i + a_j = 0$ једнак је броју појављивања вредности $-a_i$ у делу низа на позицијама $[0, i)$. Због тога користимо мапу помоћу које бројимо појављивања свих елемената низа. Када израчунамо број парова који дају збир нула, број парова који не дају збир нула можемо добити одузимањем тог броја од укупног броја парова који је једнак $\binom{n}{2} = \frac{n(n-1)}{2}$.

```
#include <iostream>
#include <map>

using namespace std;

int main() {
  int n, x;
  cin >> n;

  map<int, int> bp; // broj pojaviljivanja do sada ucitanih elemenata
  int64_t broj_parova_0 = 0; // broj parova ciji je zbir jednak nuli
  for(int i = 0; i < n; i++) {
    cin >> x;
    broj_parova_0 += bp[-x];
    bp[x]++;
  }
  int64_t ukupan_broj_parova = (int64_t)n * (n - 1) / 2;
  int64_t broj_parova_ne_0 = ukupan_broj_parova - broj_parova_0;
  cout <<  broj_parova_ne_0 << '\n';
```

```
  return 0;
}
```

Задатак: Скраћивање ниске

Над ниском се извршава следећа операција докле год је то могуће: одаберу се два једнака узастопна слова и обришу се из ниске. Написати програм који за унету ниску дужине n одређује ниску која се добија након извршења свих операција. Временска и просторна сложеност алгоритма треба да буду $O(n)$.

Улаз: Са стандардног улаза се уноси ниска састављена од малих слова енглеског алфабета дужине n ($1 \leq n \leq 10^6$).

Излаз: На стандардни излаз исписати резултујућу ниску.

Пример

Улаз	*Излаз*
`abceeccbaaba`	`abca`

Решење

Брисање карактера са почетка или из средине ниске захтева померање осталих карактера, што доводи до неефикасног програма. Уместо тога, могуће је креирати нову ниску обрађујући један по један карактер полазне ниске. Претпоставићемо да смо обрадили првих k карактера ниске и да смо брисањем свих појављивања узастопних једнаких карактера добили ниску t. Ако је ниска t празна или ако је последњи карактер ниске t различит од текућег карактера s_k ниске s (оног на позицији k), карактер s_k треба додати на t, док у супротном (ако је последњи карактер ниске t једнак карактеру s_k) треба уклонити последњи карактер ниске t и тиме (у оба случаја) добијамо резултат обраде првих $k + 1$ карактера ниске s. Приметимо да се ниска t понаша као стек (карактери јој се додају и уклањају са десног краја). Тип `string` у језику C++ подржава методе `empty`, `back`, `push_back` и `pop_back` који се извршавају у сложености $O(1)$, па се тај тип може користити као стек карактера.

Анализа сложености. Пошто су све операције за рад са стеком сложености $O(1)$ и сваки карактер се највише једном може додати и једном може уклонити са стека, сложеност овог алгоритма је $O(n)$. И меморијска сложеност је такође $O(n)$.

```
#include <iostream>
#include <string>

using namespace std;

int main() {
  string str;
  cin >> str;

  string s;
  for(char c : str)
    if(s.empty() || s.back() != c)
      s.push_back(c);
    else
      s.pop_back();

  cout << s << endl;

  return 0;
}
```

Задатак: Степенасти низ

Написати програм који исписује све степенасте низове дужине n у лексикографском поретку. Низ је степенаст ако почиње бројем 1 и сваки наредни елемент низа је или једнак претходном или за један већи од њега.

Улаз: Са стандардног улаза се уноси број n ($1 \leq n \leq 18$).

Излаз: На стандардни излаз исписати све степенасте низове дужине n, сваки у засебном реду.

Пример

Улаз	*Излаз*
3	1 1 1 1 1 2 1 2 2 1 2 3

Решење

Задатак решавамо рекурзивним набрајањем тражених комбинаторних објеката.

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

using namespace std;

void print(vector<int>& v, int i) {
  // ceo niz je popunjen, pa ga ispisujemo
  if (i == v.size()) {
    for(int x : v)
      cout << x << ' ';
    cout << '\n';
    return;
  }

  // niz nije popunjen, pa ga dopunjujemo narednim elementom

  // naredni element je jednak prethodnom
  v[i] = v[i - 1];
  print(v, i + 1);

  // naredni element je od prethodnog veci za jedan
  v[i] = v[i - 1] + 1;
  print(v, i + 1);
}

// stampa sve stepenaste nizove duzine n
void print(int n) {
  vector<int> v(n);
  v[0] = 1;
  print(v, 1);
}

int main() {
  int n;
  cin >> n;
  print(n);

  return 0;
}
```

Задатак: Распоред кућа

Дато је n плацева поређаних у низ и за сваки је позната потрошња струје. Свака кућа је дужине 1 или 2 плаца. Потрошња струје куће одређена је производом њене дужине и највеће потрошње струје плацева на којима се налази. У циљу што већих профита потребно је распоредити куће тако да сви плацеви буду заузети и да

укупна потрошња струје свих кућа буде што већа.

Написати програм који за задати низ плацева дужине n одређује максималну укупну потрошњу струје. Временска и просторна сложеност алгоритма треба да буду $O(n)$.

Улаз: Са стандардног улаза се уноси број n ($2 \leq n \leq 10^6$). Затим се уноси n бројева p_i ($1 \leq p_i \leq 100$) који представљају потрошње струје сваког плаца.

Излаз: На стандардни излаз исписати један број који представља тражену максималну потрошњу.

Пример

Улаз	*Излаз*
5 1 4 2 1 3	16

Објашњење: Најбоље је поставити прво кућу дужине 2, затим кућу дужине 1 и на крају поново кућу дужине 2. Прва кућа има потрошњу 8, друга има потрошњу 2, а трећа има потрошњу 6.

Решење

Задатак можемо решити динамичким програмирањем.

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

using namespace std;

int main() {
  int n;
  cin >> n;

  vector<int> v(n);
  for(int i = 0; i < n; i++)
    cin >> v[i];

  vector<int> dp(n);
  dp[0] = v[0];
  dp[1] = 2 * max(v[0], v[1]);
  for(int i = 2; i < n; i++)
    dp[i] = max(v[i] + dp[i - 1], 2 * max(v[i], v[i - 1]) + dp[i - 2]);

  cout << dp[n - 1] << '\n';
  return 0;
}
```

Глава 5

Јануар 2021. - група Б

Задатак: Број парних

Написати програм који за задати низ целих бројева дужине n одређује колико постоји оних бројева који се у том низу појављују паран број пута. Временска сложеност алгоритма треба да буде $O(n \log n)$, а просторна $O(n)$.

Улаз: Са стандардног улаза се уноси број n ($1 \leq n \leq 10^5$). Затим се уноси n бројева a_i ($-10^9 \leq a_i \leq 10^9$).

Излаз: На стандардни излаз исписати број различитих бројева који се у низу појављују паран број пута.

Пример

Улаз	*Излаз*
9 2 -3 1 2 -3 -3 -3 1 1	2

Објашњење: Бројеви 2 и -3 се једини појављују паран број пута у низу.

Решење

Задатак можемо решити тако што једноставно коришћењем мапе избројимо појављивања сваког броја, а затим у посебном пролазу видимо колико се од тих елемената јављало паран број пута. Задатак можемо решити и у једном пролазу тако што сваки пут пре него што увећамо бројач проверимо да ли је вредност парна или непарна – ако је парна и позитивна, број парних елемената се умањује за 1 (јер се елемент који се раније јављао паран број сада јавља непаран број пута), а ако је непарна, увећава се за један (јер се сада тај елемент јавља паран број пута).

```
#include <iostream>
#include <map>

using namespace std;

int main() {
  int n, x;
  cin >> n;

  map<int, int> m;
  int cnt = 0;
  while(n--) {
    cin >> x;
    if(m[x] > 0)
      if(m[x] % 2 == 0)
        cnt--;
      else
        cnt++;
    m[x]++;
```

```
  }

  cout << cnt << '\n';
  return 0;
}
```

Задатак: Бесконачан скуп

Бесконачан скуп S_n дефинисан је као најмањи скуп за који важи: - $n \in S_n$; - ако је $x \in S_n$, онда је $2x \in S_n$ и $4x - 3 \in S_n$.

Написати програм који за задато n и k исписује најмањих k бројева скупа S_n. Временска сложеност алгоритма треба да буде $O(k \log k)$, а просторна $O(k)$.

Улаз: Са стандардног улаза се уносе бројеви n ($2 \leq n \leq 100$) и k ($1 \leq k \leq 10^5$).

Излаз: На стандардни излаз у једном реду исписати најмањих k елемената скупа S_n, раздвојене размаком.

Пример

Улаз	*Излаз*
`2 5`	`2 4 5 8 10`

Решење

За сваки елемент x у скупу се налазе и елементи $2x$ и $4x-3$. Кренимо, на пример, од елемента 2. Он узрокује додавање елемената 4 и 5. Елемент 4 узрокује додавање елемената 8 и 13, док елемент 5 узрокује додавање елемената 10 и 17. Елемент 8 узрокује додавање елемената 16 и 29 итд. Зато су елементи низа 2, 4, 5, 8, 10, 13 итд.

Одржаваћемо скуп елемената за које смо на основу до сада обрађених елемената утврдили да треба да буду чланови низа. Најмањи елемент тог скупа је сигурно наредни елемент низа. Заиста, сви елементи у скупу ће бити већи од свих до тог тренутка исписаних и обрађених елемената низа. Лако се може доказати да су сви елементи у скупу већи или једнаки од 2, као и да за $x \geq 2$ важи да је $2x > x$ и да је $4x - 3 > x$. Зато ниједан тренутни елемент скупа не може имати наследнике који би били мањи од тренутно најмањег елемента скупа.

На основу овога можемо једноставно направити имплементацију коришћењем ефикасних структура података (скупа или реда са приоритетом). Скуп иницијализујемо на вредност n а затим му у сваком вадимо минимални елемент x и уместо њега додајемо вредности $2x$ и $4x - 3$.

```
#include <iostream>
#include <queue>

using namespace std;

int main() {
  int n, k;
  cin >> n >> k;
  priority_queue<int, vector<int>, greater<int> > pq;
  pq.push(n);
  for (int i = 0; i < k; i++) {
    int x = pq.top();
    pq.pop();
    cout << x << ' ';
    pq.push(2 * x);
    pq.push(4 * x - 3);
  }
  return 0;
}
```

Задатак: Бројеви

Написати програм који исписује у лексикографском поретку све бројеве са n цифара у систему са основом 4 који немају две суседне непарне цифре.

Улаз: Са стандардног улаза се уноси број n ($1 \leq n \leq 12$).

Излаз: На стандардни излаз исписати све тражене бројеве, сваки у засебном реду.

Пример

Улаз	*Излаз*
2	10 12 20 21 22 23 30 32

Решење

Задатак се једноставно решава рекурзивним набрајањем тражених комбинаторних објеката.

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

using namespace std;

void print(vector<int>& v, int i) {
  // niz od n cifara je popunjen, pa ispisujemo broj
  if(i == v.size()) {
    for(int x : v)
      cout << x;
    cout << '\n';
    return;
  }

  // niz nije popunjen do kraja, pa popunjavamo poziciju i, pa
  // rekurzivno popunjavamo ostatak

  // na mesto i stavljamo cifre 0, 1, 2 i 3, pri čemu cifru 0 ne smemo
  // staviti na početnu poziciju
  for(int j = (i == 0); j < 4; j++) {
    v[i] = j;
    // nije dozvoljeno da dve uzastopne cifre budu neparne
    if(i > 0 && v[i] % 2 == 1 && v[i - 1] % 2 == 1)
      continue;
    // rekurzivno popunjavamo ostatak
    print(v, i + 1);
  }
}

// ispisuje sve n-tocifrene brojeve u osnovi 4 koji nemaju dve
// uzastopne neparne cifre
void print(int n) {
  vector<int> v(n);
  print(v, 0);
}

int main() {
```

```
  int n;
  cin >> n;
  print(n);
  return 0;
}
```

Задатак: Партиционисање

Дат је низ дужине n чији су елементи бројеви 1, 2 и 3. Потребно је поделити низ на неколико сегмената тако да је збир елемената у сваком сегменту строго мањи од 4. Написати програм који одређује на колико начина је могуће направити поделу. Временска и просторна сложеност алгоритма треба да буду $O(n)$.

Улаз: Са стандардног улаза се уноси број n ($1 \leq n \leq 10^6$). Затим се уноси n бројева a_i ($1 \leq a_i \leq 3$).

Излаз: На стандардни излаз исписати број могућих партиционисања на описан начин. Како тај број може бити велики, исписати га по модулу 1000000007.

Пример

Улаз	*Излаз*
4	3
1 2 1 3	

Објашњење: Могућа партиционисања су: $1|2|1|3$, $12|1|3$ и $1|21|3$.

Решење

Задатак решавамо динамичким програмирањем навише.

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

#define M 1000000007

using namespace std;

int main() {
  int n;
  cin >> n;

  vector<int> v(n);
  for(int i = 0; i < n; i++)
    cin >> v[i];

  vector<int> dp(n + 1);
  dp[0] = 1;
  for(int i = 1; i <= n; i++) {
    dp[i] = dp[i - 1];
    if(i > 1 && v[i - 1] + v[i - 2] < 4)
      dp[i] = (dp[i] + dp[i - 2]) % M;
    if(i > 2 && v[i - 1] + v[i - 2] + v[i - 3] < 4)
      dp[i] = (dp[i] + dp[i - 3]) % M;
  }

  cout << dp[n] << '\n';
  return 0;
}
```

Глава 6

Фебруар 2021.

Задатак: Број сегмената малог збира

Дат је низ природних бројева a дужине n и један број t. Написати програм који одређује колико постоји сегмената датог низа чији је збир свих елемената строго мањи од t.

Улаз: Са стандардног улаза се учитава број n ($1 \leq n \leq 50000$) и затим се учитава n бројева a_i ($0 \leq a_i \leq 1000$) који представљају елементе низа. На крају се учитава број t ($1 \leq t \leq 10^9$).

Излаз: На стандардни излаз исписати тражени број сегмената.

Пример

Улаз	*Излаз*

`5`
`2 0 3 2 1`
`3`
	`5`

Објашњење

Сегменти са збиром елемената мањим од 3 су: `[2]`, `[2 0]`, `[0]`, `[2]` и `[1]`.

Решење

Задатак: Стабло

Пуно бинарно стабло је бинарно стабло чији сви унутрашњи чворови имају тачно 2 детета. Дат је префиксни обилазак пуног бинарног стабла са n чворова где је у сваком чвору уписано по једно слово, при чему су листови означени великим словом. Написати програм који исписује све путеве од корена до листа тог стабла. Временска сложеност алгоритма треба да буде $O(nh)$, а просторна $O(h)$ где h означава висину стабла.

Улаз: Са стандардног улаза се учитава једна ниска састављена од малих и великих слова енглеског алфабета дужине n ($1 \leq n \leq 10^5$) која представља префиксни обилазак датог стабла.

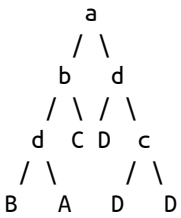
Излаз: На стандардни излаз за сваки лист стабла у засебном реду исписати ниску која представља пут од корена стабла до тог листа.

Пример

Улаз	*Излаз*

`abdBACdDcDD`	`abdB`
	`abdA`
	`abC`
	`adD`
	`adcD`
	`adcD`

Објашњење: Учитано стабло је дато на слици



Решење

Задатак решавамо тако што симулирамо обилазак дрвета у дубину уз помоћ стека.

Обилазимо карактер по карактер ниске која се добија префиксним обиласком и ако је у питању мало слово, додајемо га на стек. Ако је у питању велико слово, тада смо стигли до листа и знамо да се на стеку налази једна путања од корена до листа и можемо је исписати. У том тренутку се претрага враћа назад и скидају се одређени елементи са стека. То су сви они елементи за које знамо да су обрађена оба детета. Зато је на стеку поред слова у чвору потребно додатно памтити и информацију да ли су за тај чвор тренутно обрађена оба детета или нису (у питању је једна логичка вредност). Када чвор стављамо први пут на стек, стављамо га са вредношћу `false`. Када наиђемо на велико слово, скидамо све вредности са стека које имају вредност `true` и ако нам на стеку остане неки елемент који има вредност `false`, мењамо му вредност на `true` пре него што изнад њега додамо нови чвор (његово десно дете).

Пример. Прикажимо како алгоритам ради на примеру `abdBACdDcDD` (вредност `true` је представљена са `T` а вредност `false` са `F`).

stek	ulaz	izlaz
	abdBACdDcDD	
(a, F)	bdBACdDcDD	
(a, F) (b, F)	dBACdDcDD	
(a, F) (b, F) (d, F)	BACdDcDD	
(a, F) (b, F) (d, F) (B, T)	ACdDcDD	abdB
(a, F) (b, F) (d, T) (A, T)	CdDcDD	abdA
(a, F) (b, T) (C, T)	dDcDD	abC
(a, T) (d, F)	DcDD	
(a, T) (d, F) (D, T)	cDD	adD
(a, T) (d, T) (c, F)	DD	
(a, T) (d, T) (c, F) (D, T)	D	adcD
(a, T) (d, T) (c, T) (D, T)		adcD

```
#include <iostream>
#include <string>
#include <vector>

using namespace std;

void print_path(vector<pair<char, bool>>& s) {
  for(auto x : s)
    cout << x.first;
  cout << '\n';
}

int main() {
  string p;
  cin >> p;

  vector<pair<char, bool>> s;
  for(char ch : p)
    if(islower(ch))
      s.push_back({ch, false});
    else {
      s.push_back({ch, true});
```

```
      print_path(s);
      while(!s.empty() && s.back().second)
        s.pop_back();
      if(!s.empty())
        s.back().second = true;
    }
  return 0;
}
```

Задатак: Обилазак низа

Дат је низ природних бројева a дужине n. Играч почиње на елементу са позицијом 0. У једном потезу са позиције i играч може да пређе на једну од позиција $i + a_i$ или $i - a_i$ уколико та позиција постоји. Написати програм који за задати низ исписује све могуће обиласке тог низа у којима играч свако поље посећује тачно једном.

Улаз: Са стандардног улаза се учитава број n ($1 \leq n \leq 100$). Затим се учитава n бројева a_i ($1 \leq a_i \leq n - 1$).

Излаз: На стандардни излаз исписати све обиласке низа у било ком редоследу. Сваки обилазак исписати у засебном реду као низ бројева раздвојених размаком који представљају у ком тренутку је играч стао на коју позицију.

Пример

Улаз	*Излаз*
5	0 2 1 3 4
2 2 1 1 3	0 4 1 2 3

Објашњење: У првом обиласку играч иде редом кроз позиције 0, 2, 1, 3 и 4. У другом обиласку играч иде редом кроз позиције 0, 2, 3, 4 и 1.

Решење

Задатак решавамо рекурзивним набрајањем тражених комбинаторних објеката.

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

using namespace std;

// vrsimo obilazak niza od pozicije i, pri cemu je m trenutni redni
// broj koraka, dok se u nizu v cuva informacija o tome u kom koraku
// je neko polje u nizu poseceno --- vrednost -1 oznacava da jos nije
// poseceno
void search(int i, int m, vector<int>& v, const vector<int>& a) {
  // na poziciju i stavljamo element m cime oznacavamo da je pozicija i
  // posecena u koraku m
  v[i] = m;

  // niz je ceo popunjen, pa ga ispisujemo
  if(m == a.size() - 1) {
    for(int x : v)
      cout << x << ' ';
    cout << '\n';
  }

  // pokusavamo da napravimo a[i] koraka nalevo (to je moguce ako je indeks
  // i-a[i] unutar granica niza i ako je ta pozicija nije do sada posecena)
  if(i - a[i] >= 0 && v[i - a[i]] == -1)
    search(i - a[i], m + 1, v, a);

  // pokusavamo da napravimo a[i] koraka nadesno (to je moguce ako je indeks
```

```
  // i+a[i] unutar granica niza i ako je ta pozicija nije do sada posecena)
  if(i + a[i] < a.size() && v[i + a[i]] == -1)
    search(i + a[i], m + 1, v, a);

  // oznacavamo da pozicija i nije posecena
  v[i] = -1;
}

// obilazak niza a
void search(const vector<int>& a) {
  // broj elemenata niza
  int n = a.size();
  // nijedno polje do sada nije poseceno
  vector<int> v(n, -1);
  // u nultom koraku se nalazimo na polju 0
  search(0, 0, v, a);
}

int main() {
  int n;
  cin >> n;

  vector<int> a(n);
  for(int i = 0; i < n; i++)
    cin >> a[i];

  search(a);
  return 0;
}
```

Задатак: Збир непарних

Написати програм који за дати природни број n одређује на колико се начина он може записати као збир непарних природних бројева. Временска и просторна сложеност алгоритма треба да буду $O(n)$.

Напомена: Решење које ради у временској сложености $O(n^2)$ вреди $5/7.5$ поена.

Улаз: Са стандардног улаза се учитава број n ($1 \leq n \leq 10^5$).

Излаз: На стандардни излаз исписати решење по модулу 1000000007.

Пример

Улаз	*Излаз*
6	8

Објашњење: Могући збирови су

```
1+1+1+1+1+1
1+1+1+3
1+1+3+1
1+3+1+1
1+5
3+1+1+1
3+3
5+1
```

Решење

Нека је $f(n)$ број начина да се број n изрази као збир непарних сабирака. Анализирањем случајева за први сабирак, видимо да он може бити $1, 3, \ldots, k$, где је k последњи непаран број мањи или једнак n. Тако добијамо везу $f(n) = f(n-1)+f(n-3)+\ldots+f(n-k)$. Излаз из ове рекурзије је случај $f(0) = 1$ (јер се нула разлаже

на један начин, као збир празног скупа сабирака). Ово инсиприше решење динамичким програмирањем чија је сложеност квадратна.

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

using namespace std;

int main() {
  int n;
  cin >> n;

  vector<int> dp(n + 1);
  dp[0] = 1;
  for(int i = 1; i <= n; i++)
    for(int j = 1; j <= i; j += 2)
      dp[i] = (dp[i] + dp[i - j]) % 1000000007;

  cout << dp[n] << '\n';
  return 0;
}
```

Можемо приметити да тражени број разлагања заправо представља елементе чувеног Фибоначијевог низа, па их можемо израчунати и у линеарној сложености.

```
#include <iostream>

using namespace std;

int main() {
  int n;
  cin >> n;

  pair<int, int> dp = {0, 1};
  for(int i = 2; i <= n; i++)
    dp = {dp.second, (dp.first + dp.second) % 1000000007};

  cout << dp.second << '\n';
  return 0;
}
```

Глава 7

Јун 2022. - група А

Задатак: Сажимање бекства из лавиринта

Робот Карел је побегао из лавиринта у ком је био заробљен. Лавиринт се састоји из више квадратних поља и свако поље је суседно са највише 4 друга поља (лево, десно, горе и доле). Познато је да између свака два поља у лавиринту постоји тачно један пут (низ различитих суседних поља) који их повезује.

Робот је тражећи пут прилично лутао (више пута се враћао на исто поље). Написати програм који за познате кораке које је робот правио одређује најкраћи редослед корака који воде од његове почетне позиције до излаза из лавиринта. Један корак подразумева прелазак са тренутног поља на једно од суседних.

Улаз: Са стандардног улаза се уноси ниска S која представља кораке које је робот правио. Сваки карактер ниске је једно слово које одговара једном кораку (`L` - лево, `R` - десно, `U` - горе, `D` - доле). Број корака није већи од 10^5.

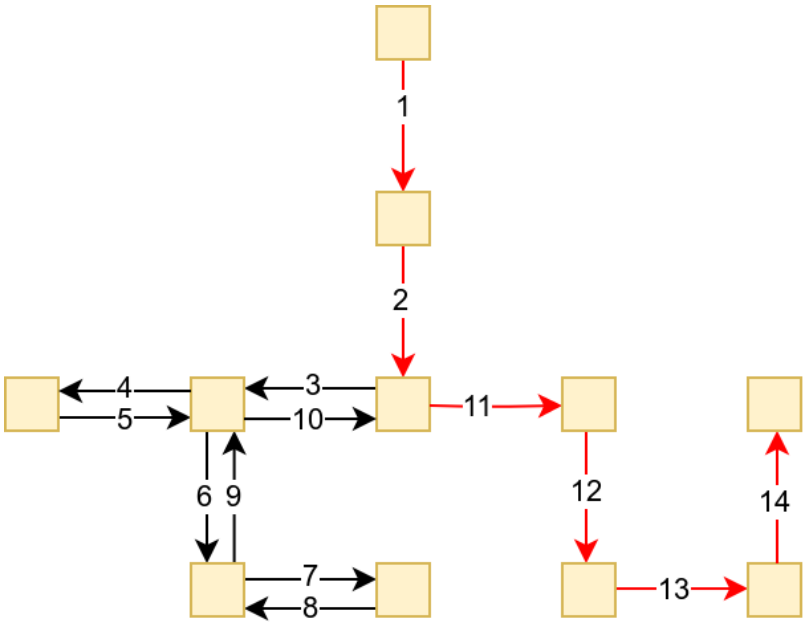
Излаз: На стандардни излаз исписати најкраћи низ корака који води од почетне позиције робота до излаза из лавиринта.

Пример

Улаз	*Излаз*
`DDLLRDRLURRDRU`	`DDRDRU`

Објашњење

Путања којом је робот прошао је приказана на слици. Путања која се добије након сажимања тј. уклањања непотребних корака приказана је црвеним стрелицама.



Слика 7.1: Путања излаза из лавиринта

Решење

Непотребни кораци настају тако што робот крене у једном смеру и затим се врати у супротном. Јасно је да се такви кораци могу слободно избацити.

Пример. Прикажимо елиминисање сувишних корака на примеру ниске `DDLLRDRLURRDRU`.

- Сувшни су кораци 4 и 5 `LR`, па затим кораци 7 и 8 `RL`. Њиховим избацивањем се добија путања `DDLDURRDRU`.
- Сада се и у овој путањи налазе сувшни кораци – то су кораци `DU` (они су били на местима 6 и 9 у оригиналној путањи). Њиховим елиминисањем добија се путања `DDLRRDRU`.
- Ова ниска такође садржи сувишне кораке `LR` (они су били на местима 3 и 10 у оригиналној путањи). Њиховим елминисањем добија се путања `DDRDRU`.
- Пошто добијена путања нема сувишних корака и она представља тражени резултат.

Задатак се зато своди на то да се итеративно проналазе узастопни парови слова који означавају супротне смерове и да се елиминишу из ниске, све док их има. Ово се може имплементирати ефикасније ако се употреби стек. Читамо ниску редом и карактере постављамо на стек. Пре постављања карактера проверавамо да ли се можда на врху стека налази њему супротан карактер и ако се налази, скидамо га (уместо да текући карактер поставимо на стек). Садржај стека на крају представља тражену сажету путању.

Пошто се на крају очекује испис путање, уместо да користимо `stack`, употребићемо обичан `string`, који методама `back`, `push_back` и `pop_back` пружа интерфејс стека.

```
#include <iostream>
#include <string>

using namespace std;

char suprotan_smer(char c) {
  switch(c) {
  case 'L': return 'R';
  case 'R': return 'L';
  case 'U': return 'D';
  case 'D': return 'U';
  }
  return 0;
```

```
}

int main() {
  string s;
  cin >> s;

  string stek;
  for (char c : s)
    if (!stek.empty() && stek.back() == suprotan_smer(c))
      stek.pop_back();
    else
      stek.push_back(c);

  cout << stek << endl;

  return 0;
}
```

Глава 8

Јун 2022. - група Б

Задатак: Збирови свих малих вредности

Дат је низ целих бројева дужине n и q упита. За сваки упит дат је један цео број x и потребно је одредити збир свих вредности из низа мањих или једнаких од x. Написати програм који обрађује упите.

Улаз: Са стандардног улаза се уноси број n ($n \leq 10^5$), након чега се уноси n бројева из интервала $[-10^9, 10^9]$ који представљају елементе низа. Затим се уноси број q ($q \leq 10^5$), након чега се уноси q бројева који представљају упите.

Напомена: Због великих вредности користити тип 64-битне целе бројеве.

Излаз: На стандардни излаз за сваки упит исписати по један број који представља одговор на упит.

Пример

Улаз	*Излаз*
5 9 2 -1 2 6 3 2 10 -5	3 18 0

Решење

Груба сила

Задатак се може решити грубом силом тако што се за сваки упит изнова саберу сви елементи низа који су мањи од дате границе.

Анализа сложености. Претрага и сабирање свих елемената низа мањих од дате границе се врши једним проласком кроз низ, у сложености $O(n)$. Пошто се то ради за сваки од q упита, укупна сложеност је $O(qn)$.

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

using namespace std;

int main() {
  ios_base::sync_with_stdio(false); cin.tie(0);
  int n;
  cin >> n;

  vector<int> v(n);
  for(int i = 0; i < n; i++)
```

```
    cin >> v[i];

  sort(v.begin(), v.end());

  int q;
  cin >> q;

  for (int i = 0; i < q; i++) {
    int x;
    cin >> x;

    long long zbir = 0;
    for(int j = 0; j < n && v[j] <= x; j++)
      zbir += v[j];

    cout << zbir << '\n';
  }

  return 0;
}
```

Бинарна претрага и префиксни збирови

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

using namespace std;

int main() {
  ios_base::sync_with_stdio(false); cin.tie(0);
  int n;
  cin >> n;

  vector<long long> v(n);
  for(int i = 0; i < n; i++)
    cin >> v[i];

  sort(v.begin(), v.end());

  vector<long long> p(n + 1, 0);
  partial_sum(v.begin(), v.end(), p.begin() + 1);

  int q;
  cin >> q;

  for (int i = 0; i < q; i++) {
    long long x;
    cin >> x;

    int pos = upper_bound(v.begin(), v.end(), x) - v.begin();
    cout << p[pos] << '\n';
  }

  return 0;
}
```

Глава 9

Јул 2021.

Задатак: Два блиска предајника

Постоји n локација на x-оси на које је могуће поставити предајник. На располагању су два предајника са дометом d. Потребно је поставити их тако да буду што више размакнути како би покривеност била што већа, али и да буду на раздаљини највише d како би могли међусобно да комуницирају. Написати програм који одређује максималну раздаљину између предајника. Временска сложеност треба да буде $O(n \log n)$, а просторна $O(n)$.

Улаз: Са стандардног улаза се уноси број n ($n \leq 10^5$) и затим се уноси n целих бројева из интервала $[-10^9, 10^9]$ који представљају координате тачака на x-оси где је могуће сместити предајнике. На крају се уноси број d ($d \leq 10^9$).

Излаз: На стандардни излаз исписати један број који представља тражену раздаљину.

Пример

Улаз	*Излаз*
4 7 3 1 8 3	2

Решење

Анализа

Смернице за алгоритам

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

using namespace std;

int main() {
  int n;
  cin >> n;

  vector<int> v(n);
  for(int i = 0; i < n; i++)
    cin >> v[i];

  int d;
  cin >> d;

  sort(v.begin(), v.end());
```

```
  int j = 0;
  int maxd = 0;
  for (int i = 0; i < n; i++) {
    while (j < i && v[i] - v[j] > d)
      j++;
    maxd = max(v[i] - v[j], maxd);
  }

  cout << maxd << endl;

  return 0;
}
```

Глава 10

Септембар 2021.

Задатак: Симетрична разлика

Написати програм који одређује симетричну разлику два унета скупа. Временска сложеност алгоритма треба да буде $O((m+n)\log(m+n))$, а просторна $O(m+n)$ при чему m и n представљају број елемената сваког скупа.

Улаз: Са стандардног улаза се уноси број m, након чега се уноси m различитих целих бројева. Потом се уноси број n и n различитих целих бројева.

Излаз: На стандардни излаз исписати елементе добијеног скупа, уређене растуће.

Пример

Улаз	*Излаз*
4 5 2 -3 8 3 8 3 2	-3 3 5

Објашњење

Бројеви -3, 3 и 5 се једини јављају у тачно једном од унетих скупова.

Решење

Опис главног решења

У овом блоку се описује главно решење задатка.

```
#include <iostream>
#include <set>

using namespace std;

int main() {
  ios_base::sync_with_stdio(false);

  int n, m, x;

  set<int> A, B, C;

  cin >> n;
  for (int i = 0; i < n; i++) {
    cin >> x;
    A.insert(x);
  }
```

```
  cin >> m;
  for(int i = 0; i < m; i++) {
    cin >> x;
    B.insert(x);
  }

  for (int a : A)
    if (B.find(a) == B.end())
      C.insert(a);

  for (int b : B)
    if(A.find(b) == A.end())
      C.insert(b);

  for (int c : C)
    cout << c << ' ';
  cout << endl;

  return 0;
}
```

Глава 11

Јануар 2022. - група А

Задатак: Магнет

Дат је низ од n гомила металних куглица. Могуће је поставити магнет испред било које гомиле и тада ће све куглице из првих k гомила са десне стране магнета бити привучене до њега. Потребно је поставити магнет тако да укупан пређени пут куглица буде што већи. Написати програм који одређује највећи могући укупан пређени пут. Временска и просторна сложеност треба да буду $O(n)$.

Улаз: Са стандардног улаза се уносе бројеви n ($1 \leq n \leq 10^5$) и k ($1 \leq k \leq n$). Затим се уноси n бројева не већих од 10^5 који представљају величине гомила.

Излаз: На стандардни излаз исписати један број који представља решење задатка.

Напомена: Због великих вредности користити 64-битне типове података.

Пример

Улаз	*Излаз*
5 3 1 4 2 3 2	17

Објашњење

Постављањем магнета испред прве гомиле куглице прелазе пут $1 \cdot 1 + 2 \cdot 4 + 3 \cdot 2 = 15$. Постављањем магнета између прве и друге гомиле куглице прелазе пут $1 \cdot 4 + 2 \cdot 2 + 3 \cdot 3 = 17$. Постављањем магнета између друге и треће гомиле куглице прелазе пут $1 \cdot 2 + 2 \cdot 3 + 3 \cdot 2 = 13$. Постављањем магнета десно од ове позиције само скраћује пређени пут, па није потребно разматрати га.

Решење

Опис главног решења

У овом блоку се описује главно решење задатка.

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

using namespace std;

int main() {
  ios_base::sync_with_stdio(false);

  int n, k;
  cin >> n >> k;

  vector<long long> v(n);
  for(int i = 0; i < n; i++)
    cin >> v[i];
```

```
  // izracunavamo predjeni put i broj kuglica kada je magnet ispred
  // prve gomile
  long long predjenPut = 0;
  long long brojKuglica = 0;
  for(int i = 0; i < k; i++) {
    brojKuglica += v[i];
    predjenPut += (i + 1) * v[i];
  }

  long long maxPredjenPut = predjenPut;
  // pomeramo magnet jednu po jednu gomilu nadesno i inkrementalno
  // azuriramo podatke
  for(int i = k; i < n; i++) {
    predjenPut -= brojKuglica;
    predjenPut += k * v[i];
    maxPredjenPut = max(maxPredjenPut, predjenPut);
    brojKuglica -= v[i - k];
    brojKuglica += v[i];
  }

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

Глава 12

Јануар 2022. - група Б

Задатак: Сузавање круга

У познатој игри *Двонедеља* налази се n играча представљених тачкама на x-оси. Простор за играње одређен је интервалом полупречника r чији је центар у координатном почетку (дакле, $(-r, r)$). Играч који се нађе изван тог интервала (на удаљености од координатног почетка строго већој од r) испада из игре. Сваког минута, полупречник се смањује за d и игра се завршава када полупречник постане строго мањи од 0. Написати програм који за сваког играча одређује колико минута ће бити у игри (могуће је да играч изгуби и на самом почетку игре). Временска и просторна сложеност треба да буду $O(n)$.

Улаз: Са стандардног улаза се уносе бројеви n ($1 \leq n \leq 10^5$), r ($1 \leq r \leq 10^18$) и d ($1 \leq d \leq r$). Након тога се у n редова уноси по један број x који представља координату играча.

Напомена: Због великих вредности користити 64-битне типове података.

Излаз: На стандардни излаз исписати n бројева, по један за сваког играча, који представљају број минута које је играч провео у игри.

Пример

Улаз	*Излаз*
4 8 3	1
7	2
-4	0
-9	3
2	

Решење

Опис главног решења

У овом блоку се описује главно решење задатка.

```
#include <iostream>

using namespace std;

int main() {
  ios_base::sync_with_stdio(false);

  long long n, r, d, x;
  cin >> n >> r >> d;

  for (int i = 0; i < n; i++) {
    cin >> x;
    if (abs(x) > r)
      cout << 0 << '\n';
```

```
    else {
      long long k = (r - abs(x)) / d + 1;
      cout << k << '\n';
    }
  }

  return 0;
}
```

Задатак: Најдужа аритметичка прогресија

Дат је скуп величине n и цео број d. Написати програм који одређује величину највећег подскупа датог скупа таквог да његови елементи чине аритметичку прогресију са размаком d. Временска сложеност треба да буде $O(n \log n)$, а просторна $O(n)$.

Улаз: Са стандардног улаза се уносе бројеви n ($1 \leq n \leq 10^5$) и d ($-10^5 \leq d \leq 10^5, d \neq 0$). Након тога се уноси n бројева из интервала $[-10^9, 10^9]$ који представљају елементе скупа.

Излаз: На стандардни излаз исписати један број који представља величину траженог подскупа.

Пример

Улаз	*Излаз*
8 3 5 8 4 1 9 11 2 7	4

Објашњење

Највећи тражени подскуп је $2, 5, 8, 11$.

Решење

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

using namespace std;

int main() {
  int n, d;
  cin >> n >> d;

  vector<int> v(n);
  for(int i = 0; i < n; i++)
    cin >> v[i];

  if (d < 0)
    d = -d;

  sort(begin(v), end(v));
  map<int, int> duzinaDo;
  int maxDuzina = 1;
  for (int i = 0; i < n; i++)
    if (duzinaDo.find(v[i] - d) != duzinaDo.end()) {
      duzinaDo[v[i]] = duzinaDo[v[i] - d] + 1;
      maxDuzina = max(maxDuzina, duzinaDo[v[i]]);
    } else
      duzinaDo[v[i]] = 1;

  cout << maxDuzina << endl;
```

```
  return 0;
}
```

Глава 13

Фебруар 2022. - група А

Задатак: Рачуни

Аутор: Иван Дрецун

Потребно је симулирати банковни систем за k различитих корисника. Сваки корисник има рачун са почетним стањем 0. Потребно је подржати две врсте операција:

- `upit x` одређује колико постоји корисника чији рачун садржи тачно x динара
- `ime x` додаје x динара на рачун особе са именом `ime` (x може бити и негативно)

Написати програм који пордржава извршавање n оваквих операција.

Улаз: Са стандардног улаза се уносе бројеви n и k. Након тога се у n редова уноси по једна операција.

Излаз: За сваки упит (операцију првог типа) исписати одговор, сваки у засебном реду.

Пример

Улаз	*Излаз*

`6 4`
`marko 2`
`milan 5`
`dragana 4`
`upit 0`
`milan -1`
`upit 4`
	`1`
	`2`

Решење

Задатак се једноставно и лако може решити употребом две мапе тј. речника. Једна се користи да преслика име корисника у износ на његовом рачуну, а друга да преслика износ на рачуну у број рачуна који садрже тај износ.

```
#include <iostream>
#include <map>

using namespace std;

int main() {
  int n, m, x;
  cin >> n >> m;

  map<string, int> racun;
  map<int, int> brPojavljivanja;
  brPojavljivanja[0] = m;

  for (int i = 0; i < n; i++) {
```

```
    string s;
    cin >> s >> x;

    if (s == ”upit”)
      cout << brPojavljivanja[x] << '\n';
    else {
      brPojavljivanja[racun[s]]--;
      racun[s] += x;
      brPojavljivanja[racun[s]]++;
    }
  }

  return 0;
}
```

Задатак: К најближих датом сортирано

Дат је низ од n бројева. Потребно је одредити k бројева у низу најближих броју x. Број a је ближи броју x од броја b ако $|a-x| \leq |b-x|$ или $|a-x| = |b-x|, a < b$. Временска сложеност треба да буде $O(n \log n)$, а просторна $O(n)$.

Улаз: Са стандардног улаза се уносе бројеви $n(1 \leq n \leq 10^6)$, $k(1 \leq k \leq n)$ и $x(1 \leq x \leq 10^9)$. Затим се уноси n бројева који су између 1 и 10^9.

Излаз: На стандардни излаз исписати тражених k бројева сортираних неопадајуће.

Пример

Улаз	*Излаз*
7 4 4 7 4 2 3 1 5 6	2 3 4 5

Решење

Опис главног решења

У овом блоку се описује главно решење задатка.

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

using namespace std;

int k_najblizih(vector<int> &v, int k, int x) {
  int n = v.size();
  sort(v.begin(), v.end());
  int l = 0;
  int r = n-1;
  while (r-l >= k) {
    if (abs(v[l] -x) <= abs(v[r]-x))
      r--;
    else
      l++;
  }

  return l;
}


int main() {
```

```
  ios_base::sync_with_stdio(false);

  int n, k, x;
  cin >> n >> k >> x;

  vector<int> v(n);
  for(int i=0; i<n; i++)
    cin >> v[i];

  int l = k_najblizih(v, k, x);

  for(int i = l; i < l+k; i++)
    cout << v[i] << " ";
  cout << endl;

  return 0;
}
```

Глава 14

Фебруар 2022. - група Б

Задатак: Број парова датог збир упити

Нека је дат цео број t. Написати програм који подржава следеће операције:

- `pisi x` записује број x на таблу,
- `brisi x` брише једно појављивање броја x са табле,
- `upit` одређује колико постоји парова бројева записаних на табли таквих да је њихов збир једнак датом броју t.

Временска сложеност треба да буде $O(n \log n)$, а просторна $O(n)$, где је n број упита.

Улаз: Са стандардног улаза се уносе бројеви n ($1 \leq n \leq 10^5$) и t ($-5 \cdot 10^8 \leq t \leq 5 \cdot 10^8$). Затим се уноси n редова који представљају извршене операције.

Излаз: За сваки упит исписати одговарајуће решење у засебном реду.

Пример

Улаз	*Излаз*
10 6 pisi 4 pisi 2 upit pisi 2 upit pisi 3 pisi 3 upit brisi 4 upit	1 2 3 1

Решење

Опис главног решења

У овом блоку се описује главно решење задатка.

```cpp
#include <iostream>
#include <map>

using namespace std;

int main() {
  ios_base::sync_with_stdio(0);
  cin.tie(0);
```

```
  int n, t, x;
  cin >> n >> t;

  map<int, int> brPojavljivanja;
  int brNacina = 0;
  for (int i = 0; i < n; i++) {
    string s;
    cin >> s;

    if (s == ”pisi”) {
      cin >> x;
      brNacina += brPojavljivanja[t - x];
      brPojavljivanja[x]++;
    } else if (s == ”brisi”) {
      cin >> x;
      brPojavljivanja[x]--;
      brNacina -= brPojavljivanja[t - x];
    } else {
      cout << brNacina << '\n';
    }
  }
  return 0;
}
```

Задатак: Најмања дужина k-точланих сегмената довољног збира

Дат је низ од n бројева и број t. Наћи најмањи број k (ако постоји) тако да за сваких k узастопних бројева у низу важи да им је збир већи или једнак t. Временска сложеност треба да буде $O(n \log n)$, а просторна $O(n)$.

Улаз: Са стандардног улаза се уносе бројеви $n(1 \leq n \leq 10^6)$ $t(1 \leq t \leq 10^18)$, затим n бројева који су између 0 и 10^9.

Излаз: На стандардни излаз исписује се тражено k ако постоји, у супротном исписује се 0.

Пример

Улаз	*Излаз*
6 5 3 1 2 1 5 2	4

Решење

Опис главног решења

У овом блоку се описује главно решење задатка.

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

using namespace std;

int main() {
  ios_base::sync_with_stdio(false);

  int n;
  long long t;
  cin >> n >> t;
  vector<int> a(n);
  for (int i = 0; i < n; i++)
    cin >> a[i];

  long long zbir = a[0];
```

```
  int l = 0, d = 0;
  int mink = 0;
  while (d < n) {
    if (zbir < t) {
      d++;
      zbir += a[d];
      mink = max(mink, d - l + 1);
    } else {
      zbir -= a[l];
      l++;
    }
  }

  if (mink == n+1)
    cout << 0 << endl;
  else
    cout << mink << endl;


  return 0;
}
```