Глава 1

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 $\binom{n}{2} = \frac{n(n-1)}{2}$.

```cpp
#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 2

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 3

Фебруар 2021.

Задатак: Сегменти

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. Временска сложеност алгоритма треба да буде $O(n \log n)$, а просторна $O(n)$.

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

Излаз: На стандардни излаз исписати тражени број сегмената.

Пример

Улаз	*Излаз*
`5 3` `2 0 3 2 1`	`5`

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

Решење

Задатак ефикасно можемо решити ако приметимо да се збир сваког сегмента низа може изразити као разлика два збира префикса (ако је p_k збир првих k елемената низа v, тада је $v_j + \ldots + v_{i-1}$ једнако $p_i - p_j$.

За свако i од 1 до n одређујемо број сегмената чији је збир већи или једнак од t који се завршавају на позицији $i - 1$. То радимо тако што пронађемо најмање j за које је $p_i - p_j < t$ – за све вредности j мање од те граничне вредности важи да је збир довољан, док је од те вредности j надаље збир премали (монотоност важи на основу чињенице да је низ префиксних збирова растући, јер су бројеви полазног низа позитивни). Ту граничну вредност j можемо ефикасно пронаћи бинарном претрагом тако што пронађемо прву вредност p_j такву да је $p_j > p_i - t$.

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

using namespace std;

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

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

```
  // izracunavamo sume prefiksa niza
  // ovaj niz je sortiran jer su elementi niza v pozitivni
  vector<int> p(n + 1, 0);
  partial_sum(v.begin(), v.end(), p.begin() + 1);
  // vazi da je zbir segmenta v[j] + ... + v[i-1] = p[i] - p[j]

  // broj segmenata
  int count = 0;
  for(int i = 1; i < p.size(); i++) {
    // odredjujemo broj segmenata koji se zavrsavaju na poziciji i-1

    // binarnom pretragom pronalazimo prvi element takav da je p[j] > p[i] - t
    // tj. takav da je v[j] + ... + v[i-1] = p[i] - p[j] < t
    int j = distance(p.begin(), upper_bound(p.begin(), p.end(), p[i] - t));
    count += i - j;
  }

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

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.

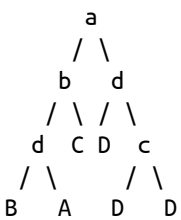
Излаз: На стандардни излаз за сваки лист стабла у засебном реду исписати ниску која представља пут од корена стабла до тог листа.

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
```

```cpp
#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 инсприше 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;
}
```