Примерни решения на домашно № 4

Задача 1: Нека Σ е азбука. *Палиндром над* Σ е всеки $x \in \Sigma^+$, който се чете по същия начин отляво надясно и отдясно наляво. Примери за палиндроми над $\{a, b\}$ са $aabbaa$, b и bab. Примери за не-палидроми са ab, aab и $bbbbba$.

Задачата е: по даден $x \in \Sigma^+$, да се намери минимален **брой** k на палиндроми y_1, y_2, ..., y_k, такива че $x = y_1 y_2 \cdots y_k$. Предложете колкото е възможно по-ефикасен алгоритъм, който при вход x връща минималния брой на палиндроми, чиято конкатенация се явява x.

Решение: Задачата далечно прилича на задачата за оптимално умножение на верига от матрици. Нека $x = x_1 x_2 \cdots x_n$. Нека $M(i, j)$ е минималният брой палиндроми-фактори на $x_i \cdots x_j$, за $1 \leqslant i \leqslant j \leqslant n$. Очевидно $M(i, i) = 1$, а отговорът е $M(1, n)$. Ето една рекурсивна декомпозиция:

$$
\begin{aligned}
&M(i,i) = 1, \text{ ако } 1 \leqslant i \leqslant n \\
&M(i,j) = 1, \text{ ако } 1 \leqslant i < j \leqslant n \text{ и } x_i \cdots x_j \text{ е палиндром} \\
&M(i,j) = \min\{M(i,k) + M(k+1,j)\} \,|\, 1 \leqslant k \leqslant j-1\}, \text{ ако } 1 \leqslant i < j \leqslant n \text{ и } x_i \cdots x_j \text{ не е палиндром}
\end{aligned}
$$

Коректността е очевидна: наистина, ако $x_i \cdots x_j$ не е палиндром, той винаги има факторизация на палиндроми, ако ще да са еднобуквени, и оптималната му факторизация на палиндроми може да се представи като $w'w''$, като w' и w'' са непразни негови фактори, а оптималната цена за x е просто сумата от техните оптимални цени.

Тази рекурсивна декомпозиция може лесно да се преведе в псевдокод на алгоритъм по схемата Динамично Програмиране, който ползва триъгълен масив. Псевдокодът е почти същият като на матричното умножение, с незначителните разлики, че

1. първо се проверява дали $x_i \cdots x_j$ е палиндром, което става тривиално във време $O(|x_i \cdots x_j|)$
2. нищо не се добавя към $M(i, k) + M(k + 1, j)$.

Алгоритъмът връща $M[1, n]$.

Сложността по време е $\Theta(n^3)$ със същите съображения като при матричното умножение. Проверката дали $x_i \cdots x_j$ е палиндром не добавя нищо към общата асимптотична сложност, понеже така или иначе намирането на минимума става във време $\Theta(|x_i \cdots x_j|)$. Сложността по памет е $\Theta(n^2)$ със същите съображения като при матричното умножение.

Задача 2: Представете си шахматна дъска $n \times n$. Квадратче $(1, 1)$ е горе вляво, а квадратче (n, n) е долу вдясно. Върху дъската има една царица и няма други фигури. В тази игра царицата се мести по правилата на шаха, но с едно важно ограничение: тя **не може да отива надясно** и **не може да отива надолу**. Пояснение: тя не може да се сложи в квадратче, което е по-ниско или вдясно.

В началото царицата се намира в произволно квадратче. Двама играчи А и Б се редуват, като всеки играч мести царицата някъде, ако това (да бъде преместена) е възможно, и след това играе другият/другата. Първо играе А. Губи играчът, който не може да премести царицата, когато е на ход.

Предложете ефикасен алгоритъм, който при дадено начално квадратче (i, j) на царицата изчислява дали А печели или А губи, при допускането, че и А, и Б играят оптимално. Накратко обосновете коректността и сложността по време на Вашия алгоритъм.

Решение: Разбиваме квадратчетата на печеливши и губещи, по отношение на А:

1. квадратче (i, j) е губещо, ако
   (а) $(i, j) = (1, 1)$, защото по тези правила царицата не може да мръдне от $(1, 1)$,
   (б) или $(i, j) \neq (1, 1)$ и всички квадратчета, достижими от (i, j) по правилата, са печеливши;
2. квадратче (i, j) е печелившо, ако $(i, j) \neq (1, 1)$ и съществува квадратче, достижимо от (i, j) по правилата, което е губещо.

Можем да решим задачата, конструирайки булев масив $A[1, \ldots, n][1, \ldots, n]$, като

$$A[i][j] = \begin{cases} 0, & \text{ако квадратче } (i, j) \text{ е губещо за А} \\ 1, & \text{ако квадратче } (i, j) \text{ е печелившо за А} \end{cases}$$

Очевидно $(1, 1)$ е губещо, така че $A[1][1] = 0$. Също така е очевидно, че A е симетричен спрямо главния диагонал, така че е достатъчно да сметнем $A[i][j]$ за $1 \leqslant i \leqslant j \leqslant n$ и после да присвоим $A[j][i] \leftarrow A[i][j]$ за $1 \leqslant i < j \leqslant n$. Разполагайки с A, за всяко запитване от вида (i, j) отговаряме ДА, ако $A[i][j] = 1$, и НЕ, в противен случай.

Ключовото нещо е порядъкът на изчисляване на стойностите $A[i][j]$. Дъното на рекурсията е $A[1][1] \leftarrow 0$. За всяка валидна (i, j), различна от $(1, 1)$, $A[i][j] \leftarrow 1$ тстк царицата, намираща се в (i, j), може да бъде сложена губещо квадратче с един ход, който ход трябва да е съобразен с ограничението, че царицата не може да отиде в квадратче (i', j'), където $i' > i$ или $j' > j$. Порядъкът на изчисляване може да е по редове отгоре надолу, а в рамките на даден ред, по колони отляво надясно.

Следният код на C решава задачата в смисъл, че създава коректен масив A. Незначителна подробност е адресирането от 0 до $n-1$, а не от 1 до n. Също така е без значение запълването на ред 0, колона 0 и главния диагонал—без $A[0][0]$—с единици като част от дъното на рекурсията.

```c
#include <stdio.h>

void printit(int n, int A[][n]) {
  int i, j;

  for (i = 0; i < n; i ++) {
    printf("\n");
    for (j = 0; j < n; j ++) {
      printf("%2d ", A[i][j]);
    }
  }

  printf("\n");
}


```

```
int main() {
  int n, i, j, k, foundzero;

  printf("\n n = ");
  scanf("%d", &n);

  int A[n][n];

  for (i = 0; i < n; i ++) {
    for (j = 0; j < n; j ++) {
      A[i][j] = -1;
    }
  }

  A[0][0] = 0;
  for (i = 1; i < n; i ++) {
    A[0][i] = 1;
    A[i][0] = 1;
    A[i][i] = 1;
  }

  for (i = 2; i < n; i ++) {
    for (j = 1; j < i; j ++) {
      if (A[i][j] < 0) {
  foundzero = 0;
  k = j-1;
  while (k > 0 && ! foundzero) {
    if (A[i][k] == 0)
      foundzero = 1;
    k --;
  }
  k = i-1;
  while (k > 0 && ! foundzero) {
    if (A[k][j] == 0)
      foundzero = 1;
    k --;
  }
  k = 1;
  while (i - k > 0 && j - k > 0 && ! foundzero) {
    if (A[i-k][j-k] == 0)
      foundzero = 1;
    k ++;
  }
  if (foundzero) {
    A[i][j] = 1;
    A[j][i] = 1;
  }
  else {
    A[i][j] = 0;
    A[j][i] = 0;
  }
      }
    }
  }

```

```
72    printit(n, A);
73
74    return 0;
75
76 }
```

wythoff.c

Сложността по време е $O(n^3)$, тъй като за всяко квадратче (i, j) извършваме работа, линейна в $\max(i, j)$.

Тази задача е известна като Wythoff's game. Обикновено правилата казват, че царицата се движи надолу и наляво, така че квадратчето, от което няма продължение—и поради това е губещо за този, който е на ход—е долу вляво. Интересен факт е, че губещите квадратчета са сравнително малко (което можете да видите, пускайки програмата) и подредбата им е подчинена на прости правила, което дава възможност за доста по-ефикасен алгоритъм от предложеното решение (в който са намесени числата на Фибоначи). Това е извън обхвата на темата за Динамично Програмиране, така че за целите на домашното, алгоритъм, подобен на предложения, е добро решение.

Също така си заслужава да се отбележи, че играта с монотонно (по хоризонтал и вертикал) местене на царица е еквивалентна на игра, при която двама играчи се редуват да вземат от две купчини камъчета: или само от едната, или само от другата, или и от двете, но еднакъв брой, като печели този, който вземе последен.

Задача 3: Разгледайте задачата Плаващи изпити. Дадени са n изпита $e_1, e_2, \ldots, e_n$. Всеки изпит e_i има продължителност ℓ_i минути, възможно най-ранно начало s_i и най-късно време на приключване t_i, като s_i и t_i са естествени числа – ако искате, интерпретирайте ги като брой минути след някакъв начален момент, който е един и същи за всички изпити. Изпълнено е $\ell_i \leqslant t_i - s_i$, иначе изпит e_i не може да се проведе.

Примерно, $\ell_i = 100$, $s_i = 45$ и $t_i = 200$, което означава, че e_i трае 100 минути и трябва да се проведе между 45-тата и 200-минута включително, спрямо някакъв начален момент.

Задачата е за разпознаване. Пита се дали съществува начин да бъдат проведени всички изпити при дадените ограничения, но в само една зала? Щом ще са в една зала, в даден момент не може да се провежда повече от един изпит.

Докажете, че Плаващи изпити е **NP**-пълна с редукция от 2-Partition.

Решение: Първо ще покажем, че Плаващи изпити $\in$ **NP**. А именно, че недетерминирана машина на Turing M, която е верификатор, може в полиномиално време да приеме (кодиране на) всеки Да-екземпляр на задачата. M отгатва функция $b : \{1, \ldots, n\} \to \mathbb{N}$, която има смисъл на "$e_i$ започва в момент $b(i)$, за всяко i". След това в полиномиално време M проверява, че за всяко i:

$$s_i \leqslant b(i)$$
$$b(i) + \ell_i \leqslant t_i$$
$$b(j) + \ell_j \leqslant b_i \text{ или } b(i) + \ell_i \leqslant b_j, \text{ за всяко } j \neq i$$

Сега ще покажем, че 2-Partition $\leqslant_p$ Плаващи изпити. Нека $\phi = \langle s(a_1), \ldots, s(a_n) \rangle$ е екземпляр на 2-Partition. Конструираме екземпляр $\psi = \langle e_1, \ldots, e_{n+1} \rangle$ на Плаващи изпити по следния начин.

1. Нека $S = \sum_{i=1}^{n} s(a_i)$. Първо, за $1 \leqslant i \leqslant n$, правим изпит e_i, като $s_i = 0$, $t_i = S + 1$ и $\ell_i = s(a_i)$.

2. Второ, правим още един изпит e_{n+1}, като $s_{n+1} = \left\lceil \frac{S}{2} \right\rceil$, $t_{n+1} = \left\lceil \frac{S+1}{2} \right\rceil$ и $\ell_{n+1} = 1$.

Конструкцията очевидно може да се направи в полиномиално време.

Първото ключово наблюдение е, че ако S не е четно число, полученият екземпляр на Плаващи изпити е Не-екземпляр, защото тогава $\left\lceil \frac{S}{2} \right\rceil = \left\lceil \frac{S+1}{2} \right\rceil$, така че $t_{n+1} - s_{n+1} = 0$ и изпитът e_{n+1} няма как да се проведе.

Да допуснем, че S е четно. Второто ключово наблюдение е, за изпит e_{n+1} няма никаква гъвкавост: тъй като $\ell_{n+1} = t_{n+1} - s_{n+1}$, той трябва да започне точно в $s_{n+1} = \left\lfloor \frac{S}{2} \right\rfloor$ и да приключи точно в $t_{n+1} = \left\lfloor \frac{S}{2} \right\rfloor + 1$. По този начин e_{n+1} "разсича" времевия интервал $[0, S + 1]$ на два подинтервала: $I_1 = \left[0, \frac{S}{2}\right]$ и $I_2 = \left[\frac{S}{2} + 1, S + 1\right]$. Нещо повече, за да може да се проведат изпитите $e_1, \ldots, e_n$, всеки от тях трябва да е или в I_1, или в I_2. Тъй като всеки от изпитите $e_1, \ldots, e_n$ има една и съща рамка във времето—от 0 до $S + 1$—не се притесняваме, че той (изпитът) може да излезе от рамката си, независимо от това, дали решим да го сложим в I_1 или I_2.

Третото ключово наблюдение е, че ако има начин да се проведат изпитите, не може да има никакви "луфтове" между тях, тъй като тяхната сумарна продължителност е равна на дължината на времевия интервал $[0, S + 1]$. И понеже e_{n+1} има продължителност единица, останалите изпити трябва точно да запълнят двата интервала, без луфтове и без припокриване. Ерго, изпитите може да се проведат тстк S е четно и $\{e_1, \ldots, e_n\}$ да може да се разбие на две подмножества, едното от които има сумарна продължителност $\frac{S}{2}$, а другото има сумарна продължителност $S + 1 - (\frac{S}{2} + 1) = \frac{S}{2}$.

Доказателството за коректност е лесно. В едната посока, нека ϕ е Да-екземпляр на 2-Partition. Това означава, че S е четно и съществува $A' \subset A$, такова че $\sum_{a \in A'} s(a) = \sum_{a \in A \setminus A'} s(a)$, където $A = \{a_1, \ldots, a_n\}$. Но тогава изпитите, съответни на елементите на A', имат сумарна продължителност $\frac{S}{2}$ и може да се сложат в I_1, а тези, които съответстват на елементите на $A \setminus A'$, може да се сложат в I_2. Показахме, че конструираният ψ е Да-пример на Плаващи изпити.

В другата посока, нека конструираният ψ е Да-пример на Плаващи изпити. Тогава S е четно и има начин част от изпитите да се сложат в I_1, а останалите, в I_2. Но тогава сумата от размерите на елементите на A, съответни на изпитите в I_1, е точно $\frac{S}{2}$, и останалите елементи на A също имат сума от размерите $\frac{S}{2}$. Но тогава ϕ е Да-екземпляр на 2-Partition.