СОФИЙСКИ УНИВЕРСИТЕТ
“СВ. КЛИМЕНТ ОХРИДСКИ”

ФАКУЛТЕТ ПО МАТЕМАТИКА
И ИНФОРМАТИКА

ДЪРЖАВЕН ИЗПИТ

ЗА ПОЛУЧАВАНЕ НА ОКС “БАКАЛАВЪР ПО КОМПЮТЪРНИ НАУКИ”

ЧАСТ I (ПРАКТИЧЕСКИ ЗАДАЧИ)

Драги абсолвенти:

- Попълнете факултетния си номер в горния десен ъгъл на всички листове.
- Пишете само на предоставените листове, без да ги разкопчавате.
- Решението на една задача трябва да бъде на същия лист, на който е и нейното условие (т.е. може да пишете отпред и отзад на листа със задачата, но не и на лист на друга задача).
- Ако имате нужда от допълнителен лист, можете да поискате от квесторите.
- На един лист не може да има едновременно и чернова, и белова.
- Черновите трябва да се маркират, като най-отгоре на листа напишете “ЧЕРНОВА”.
- Ако решението на една задача не се побира на нейния лист, трябва да поискате нов бял лист от квесторите. Той трябва да се защипе с телбод към листа със задачата.
- Всеки от допълнителните листове (белова или чернова) трябва да се надпише най-отгоре с вашия факултетен номер.
- Черновите също се предават и се защипват в края на работата.
- Времето за работа по изпита е 3 часа.

Изпитната комисия ви пожелава успешна работа!

Задача 1.

Задачата да се реши на езика C++. В подточките, в които се изисква да се посочи какво ще се изведе, точки се дават само ако отговорът напълно съвпада с това, което извежда съответният код.

1) Дадена е дефиницията:

```
void print(int* arr, int size) {
  for (int i = 0; i < size; ++i)
    cout << arr[i] << ";";
  cout << "\n";
}
```

Какво ще изведе на стандартния изход даденият по-долу фрагмент? Всеки отделен ред от изхода да се попълни в съответното поле.

```
const int size = 5;
int arr[] = { 5, 10, -1, 5, 6 };
for (int i = 0; i < size; ++i) {
  for (int j = 1; j < size; ++j) {
    if (arr[j-1] < arr[j]) {
      swap(arr[j-1], arr[j]);
    }
  }
  print(arr, size);
}
```

На ред 1 извежда: _____
На ред 2 извежда: _____
На ред 3 извежда: _____
На ред 4 извежда: _____
На ред 5 извежда: _____

2) Какво трябва да се попълни на мястото на коментара в `toUpper` така, че ако на функцията се подаде малка латинска буква, тя да връща съответната ѝ главна, а всички други символи да връща непроменени?

```
char toUpper(char c) {
  return /* какво трябва да има тук? */ ;
}
```

А) `(c >= 'a' || c <= 'z') ? (c - 'a' + 'A') : c`
Б) `(c >= 'a' || c <= 'z') ? (c + 'A') : c`
В) `(c >= 'a' || c <= 'z') ? (c - 'a') : c`
Г) `(c >= 'a' && c <= 'z') ? (c - 'a' + 'A') : c`[1]
Д) `(c >= 'a' && c <= 'z') ? (c + 'A') : c`[1]
Е) `(c >= 'a' && c <= 'z') ? (c - 'a') : c`[1]

3) Да се попълнят празните места в дефинициите на функциите така, че `isOdd` да връща истина тогава и само тогава, когато подаденото ѝ число е нечетно, а `isEven` — когато то е четно.

```
bool isEven(unsigned int);
bool isOdd(unsigned int n) {
  if (n == 0) return ________;
  if (n == 1) return ________;
  return isEven(________);
}
bool isEven(unsigned int n) {
  if (n == 0) return ________;
  if (n == 1) return ________;
  return isOdd(_________);
}
```

4) Под всеки от дадените по-долу фрагменти да се посочи какво ще изведе той на стандартния изход.

```
for (int i = 0; i < 9; ++i)
  cout << (1 << i) << ";";
```

_____

```
switch(5 % 2) {
case 0:
  std::cout << "0";
case 1:
  std::cout << "1";
default:
  std::cout << "x";
}
```

_____

```
for (int i = 0; i < 5; ++i) {
  if ( ! (i - 3))
    continue;
  std::cout << i;
}
```

_____

```
int arr[] = { 1, 2, 3, 4, 0 };
*(arr + 2) *= 10;
for (int* p = arr; *p; p++)
  std::cout << *p << ";";
```

_____

[1] В дадената на изпита тема на тези отговори беше допусната печатна грешка, която тук е отстранена.

Критерии за оценяване

В подточките, в които се изисква да се посочи какво ще се изведе, точки се дават само ако отговорът напълно съвпада с това, което извежда съответният код. Ако изходът от реда не съвпада с това, което е посочено в отговора, той се оценява с 0 т. Сумата от точките се закръгля до цяло число.

По-конкретно за съответните подточки:

1) Всеки напълно коректно посочен ред носи 0,4 т. (общо 2 т.). Изходът е както следва:

а) На ред 1 извежда: `10;5;5;6;-1;`
б) На ред 2 извежда: `10;5;6;5;-1;`
в) На ред 3 извежда: `10;6;5;5;-1;`
г) На ред 4 извежда: `10;6;5;5;-1;`
д) На ред 5 извежда: `10;6;5;5;-1;`

2) Носи 1 т. ако е посочен коректният отговор и 0 т. в противен случай. Коректният отговор е:

```
(c >= 'a' && c <= 'z') ? (c-'a'+'A') : c
```

3) Всяко напълно коректно попълнено място носи 0,5 т. (общо 3 т.). Ако в някой отговор вместо булевите литерали `true` и `false` са използвани 0 или 1, точките за отговора се намаляват наполовина. Примерно решение:

```
bool isEven(unsigned int);
bool isOdd(unsigned int n) {
  if (n == 0) return false;
  if (n == 1) return true;
  return isEven(n - 1);
}
bool isEven(unsigned int n) {
  if (n == 0) return true;
  if (n == 1) return false;
  return isOdd(n - 1);
}
```

4) Всеки напълно коректно отговорен фрагмент носи 1 т. (общо 4 т.). Изходът от фрагментите е посочен по-долу:

```
for (int i = 0; i < 9; ++i)
  cout << (1 << i) << ";";
```

```
1;2;4;8;16;32;64;128;256;
```

```
switch(5 % 2) {
case 0:
  std::cout << "0";
case 1:
  std::cout << "1";
default:
  std::cout << "x";
}
```

```
1x
```

```
for (int i = 0; i < 5; ++i) {
  if ( ! (i - 3))
    continue;
  std::cout << i;
}
```

```
0124
```

```
int arr[] = { 1, 2, 3, 4, 0 };
*(arr + 2) *= 10;
for (int* p = arr; *p; p++)
  std::cout << *p << ";";
```

```
1;2;30;4;
```

Задача 2.

Задачата да се реши на езика C++. Дадени са дефинициите:

```
class A {
public:
  A()   { cout << "A()\n"; }
  A(A&) { cout << "A(A&)\n"; }
  virtual ~A() { cout << "~A()\n"; }
  A& operator=(A&) {
    cout << "op=(A&)\n";
    return *this;
  }
};

class B : public A {
public:
  B()   { cout << "B()\n"; }
  B(B&) { cout << "B(B&)\n"; }
  virtual ~B() { cout << "~B()\n"; }
  B& operator=(B&) {
    cout << "op=(B&)\n";
    return *this;
  }
};

void f(A b) { cout << "f(A)\n"; }
```

Под всеки от редовете на дадения вдясно програмен фрагмент да се посочи какво ще се изведе в резултат от неговото изпълнение. (Между редовете нарочно е оставено повече място, за да може да попълните отговора си) Ако смятате, че някой ред няма да изведе нищо, напишете "не извежда нищо". Ако смятате, че някой от редовете ще предизвика грешка, напишете "грешка" и обяснете каква е тя и защо възниква.

За коректни се считат отговорите, които напълно съответстват на това, което ще се случи за съответния ред. Текст "грешка" без обяснение носи нула точки.

```
B d;
```

```
B copy = d;
```

```
A b = d;
```

```
A& ref = d;
```

```
B arr[2];
```

```
f(d);
```

```
A* p = new B(d);
```

```
delete p;
```

```
d = d;
```

```
ref = d;
```

Критерии за оценяване

- Всеки ред, за който напълно коректно е посочено какво ще се изведе/случи, носи 1 т.
- Ако изходът от реда не съвпада с това, което е посочено в отговора, той се оценява с 0 т.
- Текст "Грешка" без обяснение към него се оценява с 0 т.

Изходът от редовете е посочен по-долу:

```
Изход от "B d":
  A()
  B()

Изход от "B copy = d":
  A()
  B(B&)

Изход от "A b = d":
  A(A&)

Изход от "A& ref = d":
  не извежда нищо

Изход от "B arr[2]":
  A()
  B()
  A()
  B()

Изход от "f(d)":
  A(A&)
  f(A)
  ~A()

Изход от "A* p = new B(d)":
  A()
  B(B&)

Изход от "delete p":
  ~B()
  ~A()

Изход от "d = d":
  op=(B&)

Изход от "ref = d":
  op=(A&)
```

Задача 3.

Задачата да се реши на езика C++.

В тази задача едносвързан списък ще представяме чрез указател към първата му кутия, а кутиите му – чрез структури от вида:

```
struct node {
  int data;   // числов елемент
  node* next; // следваща кутия в списъка
              // или nullptr, ако няма
  node(int data, node* next = nullptr)
    : data(data), next(next)
  {}
};
```

1) Да се довърши кодът на функциите (в подчертаните места) така, че `sortListAscending` да сортира в нарастващ ред елементите на списъка `lst`. Ако списъкът е празен, функцията да не прави нищо.

```
node* min(node* lst)
{
  node* result = lst;
  for (; lst; lst = lst->next) {
    if (result->data ____  _________________)
      result = lst;
  }
  return result;
}

void sortListAscending(node* lst)
{
  while (___________________) {
    swap(___________________, min(lst)->data);
    lst = ___________________;
  }
}
```

2) Разгледайте кода на функцията `mystery`:

```
void mystery(node* lst)
{
  while (lst && lst->next) {
    lst->data += lst->next->data;
    lst->data /= 2;
    node* tmp = lst->next;
    lst = lst->next = lst->next->next;
    delete tmp;
  }
}
```

Под дадения долу фрагмент да се напише точно какво ще изведе на стандартния изход.

```
node* e = new node(56);
node* d = new node(20, e);
node* c = new node(10, d);
node* b = new node(4, c);
node* a = new node(2, b);
mystery(a);
while (a) {
  std::cout << "[" << a->data << "] -> ";
  a = a->next;
}
std::cout << "NULL";
```

---

3) Възлите на двоично дърво представяме чрез структури от вида:

```
struct tnode {
  int data;      // числов елемент
  tnode* left;   // ляв наследник
  tnode* right;  // десен наследник
  // указателите са nullptr, ако възелът
  // няма съответния наследник
};
```

Да се попълнят празните места в кода на функцията `toArray`. Тя получава корена `t` на двоично наредено дърво. Функцията да копира елементите му в масива `out` така, че да са сортирани в нарастващ ред. Това да става чрез обхождане ляво-корен-дясно (inorder). За целите на задачата допускаме, че в `out` има достатъчно място.

```
int* toArray(tnode* t, int* out)
{
  if (_____________) {
    return _____________;
  }
  else {
    int* p = toArray(_____________, out);
    *p = t->data;
    return toArray(_____________, _____________);
  }
}
```

Критерии за оценяване

1) и **3)** Всяко напълно коректно попълнено празно място носи 0,7 т. (общо 3,5 т.)

- Точки се дават само за напълно коректно посочени отговори.
- Ако написаното не е синтактично коректно или е различно от коректния отговор, се дават 0 т.

2) Носи 3 точки.

- Точките се дават само за напълно коректен отговор.
- Ако посочената в отговора поредица от числа не съвпада напълно с тази в отговора се дават 0 точки.
- Ако в отговора липсват ограждащите квадратни скоби точките се намаляват с 0,5.
- Ако в отговора липсват стрелките, точките се намаляват с 0,5.
- Ако в отговора липсва NULL, точките се намаляват с 0,5.

Сумата от точките се закръгля до цяло число.

Примерно решение на задачата

1)

```
node* min(node* lst)
{
  node* result = lst;
  for (; lst; lst = lst->next) {
    if (result->data > lst->data)
      result = lst;
  }
  return result;
}

void sortListAscending(node* lst)
{
  while (lst) {
    swap(lst->data, min(lst)->data);
    lst = lst->next;
  }
}
```

2) `[3] -> [15] -> [56] -> NULL`

3)

```
int* toArray(tnode* t, int* out)
{
  if (!t) {
    return out;
  }
  else {
    int* p = toArray(t->left, out);
    *p = t->data;
    return toArray(t->right, p + 1);
  }
}
```

Задача 4. *Задачата да се реши на един от езиците Scheme или Haskell.*

Група приятели отиват на екскурзия и се редуват да плащат общите сметки. Когато се прибират, установяват, че всеки дължи на някого дребна сума пари. Всяко задължение се представя с кортеж от три елемента: длъжник (низ), сума (цяло положително число) и получател (низ). За Scheme под "кортеж" се има предвид просто списък. "Верига" наричаме последователност от **различни** хора, в която всеки дължи на предишния една и съща сума. "Кръг" наричаме затворена верига, т.е. верига, в която първият дължи на последния същата сума. Приятелите се сещат, че ако в задълженията им има верига, то може последният човек в нея да се издължи директно на първия, а ако има кръг, то всички задължения в него се погасяват. Да се попълнят празните полета по-долу така, че:

а) функцията `maxByLength` да намира най-дълъг списък в списъка от списъци `ls`;
б) рекурсивната функция `maxChain` да намира най-дълга верига, завършваща със списъка `chain`, сума `amount` и краен получател (т.е. първи елемент) `final` при даден списък от задължения `dues`;
в) функцията `maxCircle` да намира най-дълъг кръг при даден списък от задължения `dues`.

За всички функции, ако има няколко различни отговора с еднаква дължина, не е от значение кой от тях ще бъде върнат като резултат, а ако няма нито един отговор, да се връща празният списък.

Упътване: *могат да се използват наготово всички функции в R5RS за Scheme и в Prelude за Haskell.*

Haskell

```
maxByLength ls = foldr ____________________________________________________ [] ls
maxChain chain@(last:rest) amount final dues
 | ____________________________ = chain
 | otherwise   = maxByLength [ maxChain ____________________________ amount final dues  |
                              ____________________________ <- dues, amount == _____________,
                              last == _____________, ____________________________ ]
maxCircle dues = maxByLength [ maxChain _____________ _____________ _____________ dues |
                               ____________________________ <- dues ]
```

Пример:

```
dues = [("Georgi",10,"Boyan"),("Boyan",15,"Sia"),("Sia",15,"Maria"),("Maria",10,"Georgi"),
        ("Maria",10,"Petar"),("Petar",10,"Georgi"),("Boyan",10,"Maria")]
maxChain ["Boyan"] 15 "Maria" dues ⟶ ["Maria","Sia","Boyan"]
maxCircle dues ⟶ ["Boyan","Georgi","Petar","Maria"]
```

Scheme

```
(define (maxByLength ls)
  (foldr ____________________________________________________ '() ls))
(define (maxChain chain amount final dues)
  (let ((last (car chain)) (rest (cdr chain)))
    (if ____________________________ chain
        (maxByLength (map (lambda (t) (maxChain ____________________________ amount final dues))
                          (filter (lambda (t) (and (= amount _____________)
                                                   (eqv? last _____________)
                                                   ____________________________)) dues))))))
(define (maxCircle dues)
  (maxByLength (map (lambda (t)
                      (maxChain _____________ _____________ _____________ dues) dues))))
```

Пример:

```
(define dues '(("Georgi" 10 "Boyan") ("Boyan" 15 "Sia") ("Sia" 15 "Maria") ("Maria" 10 "Georgi")
               ("Maria" 10 "Petar") ("Petar" 10 "Georgi") ("Boyan" 10 "Maria")))
(maxChain '("Boyan") 15 "Maria" dues) ⟶ ("Maria" "Sia" "Boyan")
(maxCircle dues) ⟶ ("Boyan" "Georgi" "Petar" "Maria")
```

Примерни решения

Haskell

```
maxByLength ls = foldr (\x r -> if length x > length r then x else r) [] ls

maxChain chain@(last:rest) amount final dues
 | last == final = chain
 | otherwise     = maxByLength [ maxChain (receiver:chain) amount final dues  |
                                 (giver, due, receiver) <- dues, amount == due,
                                 giver == last, not (receiver `elem` rest) ]

maxCircle dues = maxByLength [ maxChain [receiver] due giver dues |
                               (giver, due, receiver) <- dues ]
```

Scheme

```
(define (maxByLength ls)
  (foldr (lambda (x r) (if (> (length x) (length r)) x r)) '() ls))

(define (maxChain chain amount final dues)
  (let ((last (car chain)) (rest (cdr chain)))
    (if (eqv? last final) chain
        (maxByLength (map (lambda (t)
                            (maxChain (cons (caddr t) chain) amount final dues))
                          (filter (lambda (t) (and (= amount (cadr t))
                                                   (eqv? last (car t))
                                                   (not (memv (caddr t) rest)))) dues))))))

(define (maxCircle dues)
  (maxByLength (map (lambda (t)
                      (maxChain (list (caddr t)) (cadr t) (car t) dues)) dues)))
```

Критерии за оценяване

Ако е писано и по двата езика, се взима по-високият резултат. Сумата от точките се закръгля нагоре до цяло число.

Haskell

По ред на празните слотове:

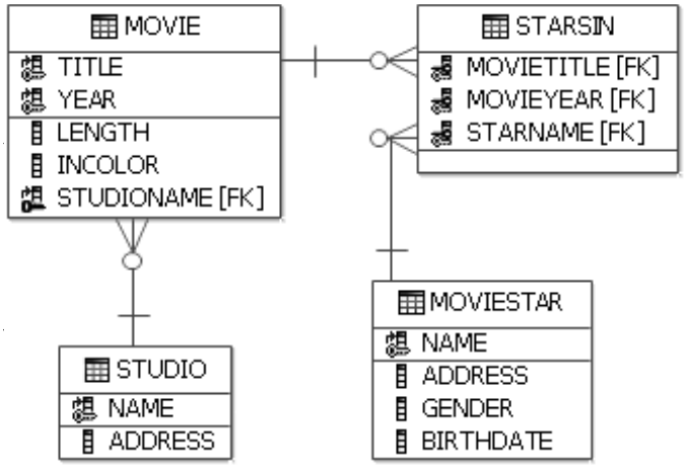
1. **3 т.** за функцията `\x r -> if length x > length r then x else r` или еквивалентна на нея, която връща този от параметрите, който е с по-голяма (или не по-малка) дължина;
2. **1 т.** за сравнението `last == final` или симетричното му;
3. **1 т.** за конструкцията `(receiver:chain)` или еквивалентна, където `receiver` е третият елемент от слот 4, не се дават точки при синтактична грешка в слот 4;
4. не се дават или отнемат точки, служи само за именуване на променливите, които се използват в другите слотове;
5. **0,5 т.** за попълнен вторият елемент от слот 4, не се дават точки при синтактична грешка в слот 4;
6. **0,5 т.** за попълнен първият елемент от слот 4, не се дават точки при синтактична грешка в слот 4;
7. **2 т.** за проверката `not (receiver `elem` rest)` или еквивалентна проверка за избягване на зациклянето, където `receiver` е третият елемент от слот 4, не се дават точки при синтактична грешка в слот 4;
8. **1 т.** за конструкцията `[receiver]` или еквивалентна, където `receiver` е третият елемент от слот 11, не се дават точки при синтактична грешка в слот 11;
9. **0,5 т.** за попълнен вторият елемент от слот 11, не се дават точки при синтактична грешка в слот 11;
10. **0,5 т.** за попълнен първият елемент от слот 11, не се дават точки при синтактична грешка в слот 11;
11. не се дават или отнемат точки, служи само за именуване на променливите, които се използват в другите слотове.

Scheme

По ред на празните слотове:

1. **3 т.** за функцията `(lambda (x r) (if (> (length x) (length r)) x r))` или еквивалентна на нея, която връща този от параметрите, който е с по-голяма (или не по-малка) дължина;
2. **1 т.** за сравнението `(eqv? last final)` или симетричното му, позволено е използването и на `equal?`, **0,5 т.** при използване на `=` или `eq?`;
3. **1 т.** за конструкцията `(cons (caddr t) chain)` или еквивалентна;
4. **0,5 т.** за конструкцията `(cadr t)` или еквивалентна;
5. **0,5 т.** за конструкцията `(car t)` или еквивалентна;
6. **2 т.** за проверката `(not (memv (caddr t) rest)` или еквивалентна проверка за избягване на зациклянето, позволено е използването и на `member`, **1 т.** при използване на `memq`;
7. **1 т.** за конструкцията `(list (caddr t))` или еквивалентна;
8. **0,5 т.** за конструкцията `(cadr t)` или еквивалентна;
9. **0,5 т.** за конструкцията `(car t)` или еквивалентна.

Задача 5. Дадена е базата от данни `Movies`, в която се съхранява информация за филми, филмови студиа, които ги произвеждат, както и актьорите, които участват в тях.



Таблицата `Studio` съдържа информация за филмови студиа:

- `name` — име, първичен ключ
- `address` — адрес;

Таблицата `Movie` съдържа информация за филми. Атрибутите `title` и `year` заедно формират първичния ключ.

- `title` — заглавие
- `year` — година, в която е заснет филмът
- `length` — дължина в минути
- `incolor` — `'Y'` за цветен филм и `'N'` за чернобял
- `studioname` — име на студио, външен ключ към `Studio.name`;

Таблицата `MovieStar` съдържа информация за филмови звезди:

- `name` — име, първичен ключ
- `address` — адрес
- `gender` — пол, `'M'` за мъж (актьор) и `'F'` за жена (актриса)
- `birthdate` — рождена дата.

Таблицата `StarsIn` съдържа информация за участието на филмовите звезди във филмите. Трите атрибута заедно формират първичния ключ. Атрибутите `movietitle` и `movieyear` образуват външен ключ към `Movie`.

- `movietitle` — заглавие на филма
- `movieyear` — година на заснемане на филма
- `starname` — име на филмовата звезда, външен ключ към `MovieStar.name`.

1) Да се попълнят празните места в следната заявка така, че тя да извежда името на студиото на филма 'The Usual Suspects' и заглавията на всички филми на същото студио:

```
SELECT s.name, m.title
FROM movie ____ JOIN studio s ON m.studioname ____________________
WHERE s.name ____ (SELECT _________________
                   FROM   _________________
                   WHERE title = 'The Usual Suspects' AND year = 1995);
```

2) Да се посочи коя от следните заявки извежда имената на филмовите звезди, за които няма информация в кои филми са играли:

```
А) SELECT DISTINCT starname
   FROM starsin
   GROUP BY starname
   HAVING COUNT(*) = 0;
```

```
Б) SELECT ms.name, si.movietitle
    FROM moviestar ms
    LEFT JOIN starsin si
           ON ms.name=si.starname
    WHERE si.movietitle IS NULL;
```

```
В) SELECT name
   FROM starsin
   JOIN moviestar ON starname = name
   GROUP BY name
   HAVING COUNT(name) = 0;
```

```
Г) SELECT name
   FROM moviestar
   WHERE NOT EXISTS (SELECT starname
                     FROM starsin);
```

Критерии за оценяване

а) Общо 5 т., по ред на празните слотове:
- `m` или `AS m` — **1 т.**
- `=s.name` или `= name` — **1 т.**
- `=` или `IN` — **1 т.**
- `studioname` — **1 т.**
- `movie` — **1 т.**

б) Единственият верен отговор е Б). Посочването на този отговор носи 5 т., а посочването на грешен отговор или комбинация от отговори носи 0 т.

Заявката, описана в този отговор, коректно извежда името на филмовите звезди, за които няма информация в кои филми са играли, съгласно условието на тази подточка. Фактът, че в допълнение на исканата в условието информация заявката извежда също и втора колона с име `movietitle` и стойности `NULL`, не променя коректността на заявката относно зададеното така условие.

Заявките, посочени във всички останали отговори, не отговарят на условието на подточката. В частност, заявките в отговор А) и отговор В) винаги връщат празно множество, независимо дали има или не филмови звезди, за които няма информация в кои филми са играли, както би трябвало да е по условие. Заявката в отговор Г) не отговаря на условието, понеже в случая, в който таблицата `starsin` не е празна, не се извежда името на нито една филмова звезда, вместо да се изведат имената на тези, за които няма информация в кои филми са играли, както би трябвало да е по условие. Това поведение е съгласно стандарта за езика SQL и не зависи от конкретната реализация в система за управление на бази от данни.

Задача 6. Дадени са плочки с размери 1×1 в четири цвята, в неограничено количество от всеки цвят. Дадени са и плочки с размери 1×2 в пет цвята, в неограничено количество от всеки цвят. Ако n е цяло положително число, по колко различни начина може да бъде покрит напълно, без припокриване, правоъгълник с размери $1 \times n$ с дадените плочки?

Отговорът носи точки само ако е придружен от коректна обосновка.

Примерно решение: Нека S_n е броят на начините да покрием правоъгълник $1 \times n$ по описания начин.

Ако $n = 1$, има точно 4 начина за покриване: може да се ползва само плочка 1×1 в точно един от четирите цвята. Заключаваме, че $S_1 = 4$.

Ако $n = 2$, има точно 21 начина:

- ако се ползва плочка 1×1, налага се да се ползва още една плочка 1×1, като има $4 \cdot 4 = 16$ начина за слагане на двете в линейна наредба;
- ако се ползва плочка 1×2, друга плочка не се ползва, а тази 1×2 плочка е в точно един от пет цвята.

Заключаваме, че $S_2 = 16 + 5 = 21$.

Ако $n > 2$:

- може да започнем с плочка 1×1, като има точно четири начина да изберем цвета ѝ, и след това покриваме правоъгълник с размери $1 \times (n-1)$;
- може да започнем с плочка 1×2, като има точно пет начина да изберем цвета ѝ, и след това покриваме правоъгълник с размери $1 \times (n-2)$.

Тези начала са несъвместими, така че, съгласно комбинаторния принцип на събирането, $S_n = 4S_{n-1} + 5S_{n-2}$.

Изведохме следното рекурентно уравнение:

$$S_n = \begin{cases} 4, & \text{ако } n = 1 \\ 21, & \text{ако } n = 2 \\ 4S_{n-1} + 5S_{n-2}, & \text{ако } n > 2 \end{cases}$$

Да решим уравнението. Характеристичното уравнение е

$$x^2 - 4x - 5 = 0$$

с корени 5 и -1. Общото решение е

$$S_n = A5^n + B(-1)^n$$

за някакви константи A и B, които ще намерим от началните условия.

$$A5^1 + B(-1)^1 = 4$$
$$A5^2 + B(-1)^2 = 21$$

Тоест,

$$5A - B = 4$$
$$25A + B = 21$$

Оттук $30A = 25$, тоест $A = \frac{5}{6}$. Тогава $B = \frac{1}{6}$. Решението е

$$S_n = \frac{1}{6}\left(5^{n+1} + (-1)^n\right)$$

□

Схема за оценяване Най-естествено е задачата да се реши, като първо се състави рекурентно уравнение и после се реши уравнението.

За съставяне на коректно рекурентно уравнение се дават 6 точки. За коректно общо решение на уравнението се дава още 1 точка. За намиране без грешки на константите в общото решение се дават още 3 точки. И така, при решение по тази схема без грешки се дават 10 точки на задачата.

Ако рекурентното уравнение е вярно и общото решение е вярно и само е допусната дребна грешка в сметките за намиране на константите, дават се 9 точки на задачата – дребната грешка води до отнемане на 1 точка.

Ако рекурентното уравнение е грешно, оценката на задачата е 0 точки.

Ако рекурентното уравнение е вярно, но общото решение е грешно, оценката на задачата е 6 точки.

Ако рекурентното уравнение е вярно и общото решение е вярно, но нататък се продължава по грешен път, задачата се оценява със 7 точки.

Ако решението е по друга схема, 10 точки се дават на пълно вярно решение, което дава формула, която като алгоритъм има сложност, сходна с тази на $\frac{1}{6}\left(5^{n+1}+(-1)^n\right)$. Ако решението е вярно, но крайният резултат е формула със сумирания или други действия, която формула като алгоритъм има сложност, значително надвишаваща сложността на $\frac{1}{6}\left(5^{n+1}+(-1)^n\right)$, отнемат се точки в зависимост от сложността на алгоритъма.

Ако решението е невярно дори за една стойност на n, задачата се оценява с 0 точки.

Задача 7. За думи α и β над азбука $\Sigma = \{a, b, c\}$, означаваме

$$\alpha \preceq \beta \Leftrightarrow (\exists \gamma \in \Sigma^\star)[\alpha \cdot \gamma = \beta].$$

За произволен език $L \subseteq \Sigma^\star$, означаваме

$$\texttt{Pref}(L) = \{\alpha \in \Sigma^\star \mid (\exists \beta \in L)[\alpha \preceq \beta]\}.$$

Вярно ли е, че за всеки език $L \subseteq \Sigma^\star$:

а) ако L е регулярен, то $\texttt{Pref}(L)$ е регулярен ?

б) ако L не е регулярен, то $\texttt{Pref}(L)$ не е регулярен ?

Обосновете отговорите си като приложите доказателства!

Примерно решение:

Доказателство.

а) Нека L е произволен регулярен език. Ще покажем, че $\texttt{Pref}(L)$ също е регулярен.

Да фиксираме детерминиран краен автомат $\mathcal{A} = (\Sigma, Q, \delta, q_0, F)$ с език $L(\mathcal{A}) = L$. Нека $Q_{co-acc} = \{q \in Q \mid \exists \gamma \in \Sigma^*(\delta^*(q, \gamma) \in F)\}$ и

$$\mathcal{A}_{pref} = (\Sigma, Q, \delta, q_0, Q_{co-acc}).$$

От теоремата на Клини е достатъчно да докажем, че $L(\mathcal{A}_{pref}) = \texttt{Pref}(L)$.

Първо, нека $\alpha \in \texttt{Pref}(L)$. Това означава, че $\alpha \preceq \beta$, за някоя дума $\beta = \alpha \cdot \gamma \in \mathcal{L}(\mathcal{A})$. Тогава $\delta^\star(q_0, \alpha \cdot \gamma) \in F$, откъдето $p = \delta^\star(q_0, \alpha)$ е добре дефинирано и $\delta^\star(p, \gamma) \in F$. Последното показва, че $p \in Q_{co-acc}$, откъдето $\delta^\star(q_0, \alpha) \in Q_{co-acc}$ и $\alpha \in L(\mathcal{A}_{pref})$.

Второ, нека $\alpha \in \mathcal{L}(\mathcal{A}_{pref})$. Тогава $\delta^\star(q_0, \alpha) = p \in Q_{co-acc}$ и по дефиницията на Q_{co-acc} получаваме, че има дума γ, за която $\delta^\star(p, \gamma) \in F$. Сега е ясно, че $\delta^\star(q_0, \alpha \cdot \gamma) = \delta^\star(p, \gamma) \in F$, тоест $\alpha \cdot \gamma \in L$. Следователно $\alpha \in \texttt{Pref}(L)$.

б) Ще докажем, че условието б) не е изпълнено. За тази цел, ще посочим нерегулярен език L, за който $\texttt{Pref}(L)$ е регулярен.

Да изберем езика $L = \{a^n b^m \mid n \leq m\}$. Ще видим, че L не е регулярен като покажем, че съществуват безкрайно много класове на еквивалентност на релацията на Майхил-Нероуд $\approx_L$, където

$$\alpha \approx_L \beta \Leftrightarrow (\forall \gamma \in \Sigma^\star)[\alpha\gamma \in L \iff \beta\gamma \in L].$$

Достатъчно е да докажем, че за всеки две естествени числа ℓ и k

$$\ell < k \implies a^\ell \not\approx_L a^k.$$

Това лесно се съобразява, защото за $\gamma = b^\ell$, $a^\ell\gamma \in L$, но $a^k\gamma \notin L$.

Сега ще докажем, че $\texttt{Pref}(L) = \{a\}^\star \cdot \{b\}^\star$, който е регулярен език.

Нека $\alpha \in \texttt{Pref}(L)$. Това означава, че $\alpha \preceq a^n b^k$ за някои $k \geq n$. Ясно е, че тогава $\alpha \in \{a\}^\star \cdot \{b\}^\star$.

Обратно, ако $\alpha \in \{a\}^\star \cdot \{b\}^\star$, то $\alpha = a^n b^k$ за някои $n, k \in \mathbb{N}$, откъдето $\alpha \preceq (a^n b^k) \cdot b^n = a^n b^{n+k} \in L$, тоест $\alpha \in \texttt{Pref}(L)$.

□

Критерии за оценяване:

а) 3 точки, от които:

- 1 точка – за конструкцията на $\mathcal{A}_{pref}$;
- 1 точка – за доказателство, че $\texttt{Pref}(L) \subseteq L(\mathcal{A}_{pref})$;
- 1 точка – за доказателство, че $L(\mathcal{A}_{pref}) \subseteq \texttt{Pref}(L)$.

б) 7 точки, от които:

- 1 точка – за избора на езика L. Тази точка се дава, само ако решението заслужава поне още една точка от долните (в частност алтернативната схема);
- 3 точки – за доказателство, че L не е регулярен, от които:
  - 1 точка – за деклариране на безброй много две по две нееквивалентни думи;
  - 2 точки – за доказателство, че те наистина са две по две нееквивалентни.
- 3 точки – за доказателство, че $\texttt{Pref}(L)$ е регулярен, от които:
  - 1 точка – за деклариране, че $\texttt{Pref}(L) = \{a\}^\star \cdot \{b\}^\star$;
  - 1 точка – за доказателство, че $\texttt{Pref}(L) \subseteq \{a\}^\star \cdot \{b\}^\star$;
  - 1 точка – за доказателство, че $\{a\}^\star \cdot \{b\}^\star \subseteq \texttt{Pref}(L)$;

Алтернативна схема за а).: При доказателства на а). използващи рекурсивната дефиниция за регулярен език:

- 1 точка – за доказателство $\texttt{Pref}(L_1 \cup L_2) = \texttt{Pref}(L_1) \cup \texttt{Pref}(L_2)$;
- 1 точка – за доказателство $\texttt{Pref}(L_1 \cdot L_2) = \texttt{Pref}(L_1) \cup L_1 \cdot \texttt{Pref}(L_2)$;
- 1 точка – за доказателство $\texttt{Pref}(L_1^\star) = L_1^\star \texttt{Pref}(L_1)$.
- -1 точка – ако решението е изпълнило горните, но липсва някой от базовите случаи $\texttt{Pref}(\{a\}) = \{\varepsilon, a\}$ и/или $\texttt{Pref}(\emptyset) = \emptyset$. (Доколкото, $\emptyset^\star = \{\varepsilon\}$, тези два случая са достатъчни.)

Алтернативна схема за б).: При доказателства в б)., че L не е регулярен, използващи лема за разрастването:

- 1 точка – за коректен избор на w с проверка, че $w \in L$ и $|w| \geq n_0$, където предварително n_0 е въведено като константата от лемата за разрастването за езика L, например $w = a^{n_0} b^{n_0}$;
- 2 точки – за достигането до противоречие.
- Решения, които не формулират и/или използват по грешен начин лемата не получават точки.

Забележка: Точките от алтернативната и основната схема за оценяване не са адитивни. Ако са приложими повече от една схема, работата се оценява с максимума по съответната схема.

Задача 8. Да се пресметне определеният интеграл

$$\int_0^1 \sqrt{x}\,\ln\sqrt{1+x}\,dx.$$

Примерно решение

В интеграла правим субституцията $x = t^2$, $t \geq 0$ Използваме, че $\ln \sqrt{1+x} = \frac{1}{2} \ln(1+x)$. Така получаваме

$$\int_0^1 \sqrt{x} \ln \sqrt{1+x}\, dx = \int_0^1 t^2 \ln(1+t^2)\, dt.$$

След това внасяме t^2 под знака на диференциала и интегрираме по части:

$$\begin{aligned}\int_0^1 t^2 \ln(1+t^2)\, dt &= \frac{1}{3} \int_0^1 \ln(1+t^2)\, d(t^3) \\ &= \frac{1}{3} \left(t^3 \ln(1+t^2) \Big|_0^1 - \int_0^1 t^3 d \ln(1+t^2) \right) \\ &= \frac{\ln 2}{3} - \frac{2}{3} \int_0^1 \frac{t^4}{1+t^2}\, dt.\end{aligned}$$

Използваме разлагането

$$\frac{t^4}{1+t^2} = \frac{t^4-1}{t^2+1} + \frac{1}{1+t^2} = t^2 - 1 + \frac{1}{1+t^2}.$$

Така получаваме

$$\begin{aligned}\int_0^1 t^2 \ln(1+t^2)\, dt &= \frac{\ln 2}{3} - \frac{2}{3} \int_0^1 t^2\, dt + \frac{2}{3} \int_0^1 dt - \frac{2}{3} \int_0^1 \frac{dt}{1+t^2} \\ &= \frac{\ln 2}{3} - \frac{2}{3} \frac{t^3}{3} \Big|_0^1 + \frac{2}{3} - \frac{2}{3} \operatorname{arctg} t \Big|_0^1 \\ &= \frac{\ln 2}{3} - \frac{2}{9} + \frac{2}{3} - \frac{2}{3} \left(\frac{\pi}{4} - 0 \right) \\ &= \frac{\ln 2}{3} + \frac{4}{9} - \frac{\pi}{6}.\end{aligned}$$

Критерии за оценяване

- субституция $x = t^2$: 2 т.,
- интегриране по части: 3 т.,
- разлагане на $\frac{t^4}{1+t^2}$ във вид удобен за интегриране (в сума на елементарни дроби): 2 т.
- пресмятане на $\int_0^1 (t^2 - 1)\, dt$: 1 т.,
- пресмятане на $\int_0^1 \frac{1}{1+t^2}\, dt$: 1 т.,
- получаване на окончателния отговор: 1 т.

Чернова