Межрегиональная предметная олимпиада КФУ
по предмету «Информатика»
заключительный этап (ответы)
2019-2020 учебный год
9 класс

Задача 1

Для решение задачи применяется метод динамического программирования. Создадим матрицу mas[n][m], в которой для каждой клетки с индексами i и j буем хранить минимальное количество топлива, чтобы добраться с левого верхнего угла в аэродром с индексами i и j. Изначально заполним его очень большими значениями.

Пусть в массиве a[n][m] хранится заданная матрица. Тогда:

Часть кода:

```
mas[0][0]=a[0][0];

for (int i=0;i<n;i++)
{
        for (int j=0; j<m; j++)
        {
                if (a[i][j]!=-1)
                {
                        for (int k=j+1;k <= min(j+a[i][j], m-1); k++)
                                if (a[i][k]!=-1)
                                        mas[i][k]=min(mas[i][k], mas[i][j]+a[i][k]);
                        for (int k=i+1;k <= min(i+a[i][j], n-1); k++)
                                if (a[k][j]!=-1)
                                        mas[i][k]=min(mas[i][k], mas[i][j]+a[k][j]);
                }
        }
}
cout<<mas[n-1][m-1];
```

Задача 2

Для решения задачи необходимо на нечетных строках (если нумерация начинается с 1) выводить числа Фибоначчи слева направо, на четных – справа налево. Для хранения матрицы заведем двумерный массив mas[n+1][n+1].

Часть кода:

```
int f1=0, f2=1, f, mas[9][9];
        int n;
        cin>>n;
        mas[0][0]=0; mas[0][1]=1;
        for(int i=2;i< n; i++)
        {
                f=f1+f2;
                f1=f2;
                f2=f;
```

```
            mas[0][i]=f;
        }

        for(int i=1; i<n; i++)
        {
            if ((i+1)%2==1) // в с++ индексация в массивах начинается с 0. Случай слева напрво
            {
                for (int j=0; j<n; j++)
                {
                    f=f1+f2;
                    f1=f2;
                    f2=f;
                    mas[i][j]=f;
                }
            }
            else
            {
                for (int j=n-1; j>=0; j--) //Случай справа налево
                {
                    f=f1+f2;
                    f1=f2;
                    f2=f;
                    mas[i][j]=f;
                }
            }
        }

        //вывод матрицы ...
```

Задача 3

Задача на разбор арифметического выражения, решаться может по-разному, например, алгоритмом рекурсивный спуск (другой вариант: обратная польская запись). Будем считать, что у нас есть строка s, в которой сохранено выражение. Добавим в конец строки символ решетка ’#’. Пусть переменная j будет хранить текущий индекс. Сначала напишем функцию, которая вычисляет значение выражения до того как не встретит символ не цифра и не + или -, это будет функция expression(). Для этого переберем части строки, разделенные + или -. В качестве элементов могут быть числа или в скобках выражения, которые мы обрабатываем с помощью рекурсивного вызова expression().

```
String s;

int j;

bool error = false;

int addend();

int expression(){

  int k = 1;
```

```
  if (s[j] ==’-’) {
    k = -1; j++;
  }
  int result = k*addend();
  if (error) return 0;
  char c = s[j];
  while (c == ‘+’ || c== ‘-’){
    j++;
    int a = addend();
    if (error) return 0;
    if (c==’-’) a = -a;
    result = result + a;
    c = s[j];
  }
  return result;
}

int addend(){
  int result = 0;
  if (s[j]==’(‘){
    j++;
    result = expression();
    if (s[j]==’)’){ j++;)
    else {error = true; return 0;}
  }
  else result = number();
  return result;
}

int number(){
  int result;
```

```
  while (s[j]>=’0’ && s[j]<=’9’){
    result = result*10 + ((int)s[j] – (int)‘0’);
    j++;
  }
  return result;
}

int main(){
  cin<<s;
  s = s+”#”;
  j = 0;
  int result = expression();
  if (s[j]!=’#’) error = true;
  if (error) cout<<”Error”;
  else cout<<result;
}
```

Задача 4

Задача решается с помощью поиска в глубину (или в ширину). Достаточно запустить поиск в глубину из каждой незанятой точкой. Если на границе связной области фишки одного цвета, то они принадлежат этому цвету, если разного, то никакому. Напишем функцию dfs, которая будет возвращать Б, если принадлежит белым, Ч для черных и ‘?‘ для «ничейных». Будем считать, что матрица сохранена в массиве d.

```
int n = 19;
char d[n][n];
int di[4] = {0, 0, +1, -1};
int dj[4] = {-1, +1, 0, 0};
char dfs(int i, int j){
  d[i][j] = ‘#’;
  char result = ‘-’;
  for (int k =0; k <4; k++){
    int i1 = i+di[k];
```

```
    int j1 = j + dj[k];
    if (i1<0||i1>=n||j1<0||j1>=n ||d[i1][j1]=='#') continue;
    char r ;
    if (d[i1][j1]=='.'){
      char r = dfs(i1,j1);
    }
    if (d[i1][j1]=='Ч' ||d[i1][j1]=='Б' )
      r = d[i1][j1];
    if (r=='?' || result == '?') result = '?';
    else
    if (result !='-' && r!=result) result = '?';
    else result = r;
  }
  return result;
}

void main(){
//чтение массива d
int b = 0, w = 0;
for (int i = 0; i <0; i++)
  for (int j = 0; j < 0; j++)
    if (d[i][j]=='.'){
    char r = dfs(i,j);
    if (r =='Ч') b++;
    if (r =='Б') w++;
   }
cout<<w<<" "<<b;
}
```

Задача 5

Самое простое решение для данной задачи: вывести “2” и n-1 “0”. Пример: пусть n=4. Тогда

$\frac{2000}{2^4} = \frac{2*10*10*10}{2*2*2*2}$, каждая 10 делится на 2, и первая двойка делится на 2.

Следующий вариант решения: возвести 2 в степень n любым стандартным способом, сохранив результат в переменную типа int64, посчитать k=количество цифр в полученном числе, вывести это число 2^n и следом за ним n-k “0”.

Задача 6

Задача сводится к решению задачи о правильности расстановки скобок (и). Их баланс никогда не может быть отрицательным и в конце должен быть равен 0.

```
int n;
cin >> n;
char * s = new char s [n*2+1];
cin >> s;
int bal = 0;
bool ok = true;
for ( int i=0; i < 2*n; i++ )
   if ( s [i] == 'S' )
      bal++;
   else
   {
      bal--;
      if ( bal < 0 )
      {
         ok = false; break;
      }
   }
   if ( ok && bal == 0 )
      cout << "Yes";
   else
      cout << "No";
```

Межрегиональная предметная олимпиада КФУ
по предмету «Информатика»
заключительный этап (ответы)
2019-2020 учебный год
10 класс

Задача 1

Для решение задачи применяется метод динамического программирования. Создадим матрицу mas[n][m], в которой для каждой клетки с индексами i и j буем хранить минимальное количество топлива, чтобы добраться с левого верхнего угла в аэродром с индексами i и j. Изначально заполним его очень большими значениями. Создадим ещё одну матрицу p[n][m], её тоже необходимо заполнить очень большими значениями. В p[i][j] будем хранить минимальное количество аэродромов, в которых Дрон останавливался, чтобы добраться до аэродрома с индексами i и j за минимальное количество топлива.

Пусть в массиве a[n][m] хранится заданная матрица. Тогда:

Часть кода:

```
mas[0][0]=a[0][0];
p[0][0]=1;
for (int i=0;i<n;i++)
{
        for (int j=0; j<m; j++)
        {
                if (a[i][j]!=-1)
                {
                        for (int k=j+1;k <= min(j+a[i][j], m-1); k++)
                                if (a[i][k]!=-1)
                                        if (mas[i][k]<mas[i][j]+a[i][k])
                                        {
                                                mas[i][k]=mas[i][j]+a[i][k];
                                                p[i][k]=p[i][j]+1;
                                        }
                                        else
                                                if (mas[i][k]==mas[i][j]+a[i][k])
                                                {
                                                        p[i][k]=min(p[i][k], p[i][j]+1);
                                                }
                        for (int k=i+1;k <= min(i+a[i][j], n-1); k++)
                                if (a[k][j]!=-1)
                                        if (mas[k][j]<mas[i][j]+a[k][j])
                                        {
                                                mas[k][j]=mas[i][j]+a[k][j];
                                                p[k][j]=p[i][j]+1;
                                        }
                                        else
                                                if (mas[k][j]==mas[i][j]+a[k][j])
                                                {
                                                        p[k][j]=min(p[k][j], p[i][j]+1);
                                                }
```

```
                }
        }
}
cout<<p[n-1][m-1];
```

Задача 2

Для заполнения матрицы числами Фибоначчи в заданном порядке заведём символьную переменную d, которая будет следить, в каком направлении мы сейчас движемся. В матрицу mas[m][m] будем хранить нужную матрицу из чисел Фибоначчи. Изначально в mas[m][m] запишем значения -1. Если в mas[i][j] ещё стоит -1, значит, мы туда ещё ничего не записали.

Варианты значения переменной d:

‘u’ - наверх

‘r’ - напрво

‘l’ - налево

‘b’ -вниз

Часть кода:

```
int f1=0, f2=1, f;

cin>>m;

for(int i=0; i<m; i++)
        for(int j=0;j<m; j++)
                mas[i][j]=-1;
mas[0][0]=0; mas[0][1]=1;
int k=2;
char d='r'; //текущее направление
int i=0, j=2; //индексы для матрицы
while (k<=m*m)
{
        k++;
        f=f1+f2;
        f1=f2;
        f2=f;
        if (d=='r')
        {
                mas[i][j]=f;
                j++;
                if (j==m or a[i][j]!=-1)
                {
                        d='b';
                        i++;
                        j--;
                }
                continue;
        }
        if (d=='b')
        {
```

```
                    mas[i][j]=f;
                    i++;
                    if (i==m or a[i][j]!=-1)
                    {
                        d='l';
                        j--;
                        i--;
                    }
                    continue;
                }
                if (d=='l')
                {
                    mas[i][j]=f;
                    j--;
                    if (j==-1 or mas[i][j]!=-1)
                    {
                        d='u';
                        i--;
                        j++;
                    }
                    continue;
                }
                if (d=='u')
                {
                    mas[i][j]=f;
                    i--;
                    if (mas[i][j]!=-1) // здесь не нужно проверять, не вышли ли мы за пределы, так как самая верхняя строчка уже заполнена
                    {
                        d='r';
                        j++;
                        i++;
                    }
                }
            }

            //вывод матрицы ...
```

Задача 3

Задача на разбор арифметического выражения, решаться может по-разному, например, алгоритмом рекурсивный спуск (другие варианты: обратная польская запись; просто вычислить все что связано знаками * и /, а потом все что +-). Будем считать, что у нас есть строка s, в которой сохранено выражение. Добавим в конец строки символ решетка ’#’. Пусть переменная j будет хранить текущий индекс. Сначала напишем функцию, которая вычисляет значение выражения до того как не встретит символ не цифра и не + или - это будет функция expression(). Для этого переберем части строки разделенные + или -. В качестве элементов могут быть числа или числа связанные * или /, которые мы обрабатываем по аналогии.

```
String s;
```

```
int j;
bool error = false;
int addend();
int expression(){
  int k = 1;
  if (s[j] ==’-’) {
    k = -1; j++;
  }
  int result = k*addend();
  if (error) return 0;
  char c = s[j];
  while (c == ‘+’ || c== ‘-’){
    j++;
    int a = addend();
    if (error) return 0;
    if (c==’-’) a = -a;
    result = result + a;
    c = s[j];
  }
  return result;
}

int addend(){
  int result = number();
  if (error) return 0;
  char c = s[j];
  while (c == ‘*’ || c== ‘/’){
    j++;
    int a = number();
    if (error) return 0;
    if (c==’*’) result = result * a;
```

```
    else result = result / a;
    c = s[j];
  }
  return result;
}

int number(){
  int result;
  while (s[j]>='0' && s[j]<='9'){
    result = result*10 + ((int)s[j] – (int)'0');
    j++;
  }
  return result;
}

int main(){
  cin<<s;
  s = s+"#";
  j = 0;
  int result = expression();
  if (s[j]!='#') error = true;
  if (error) cout<<"Error";
  else cout<<result;
}
```

Задача 4

Задача решается с помощью поиска в глубину (или в ширину). Достаточно запустить поиск в глубину из каждой точкой цвета противника. Если на границе связной области фишки другого цвета, то они окружены. Если есть свободная точка, то тогда группа не окружена. Напишем функцию dfs, которая будет возвращать число фишек в группе или -1, если группа расширяем. Будем считать, что матрица сохранена в массиве d.

```
int n = 19;
```

```
char d[n][n];
int di[4] = {0, 0, +1, -1};
int dj[4] = {-1, +1, 0, 0};
bool is_surrounded = true;
int dfs(int i, int j, char p){
  d[i][j] = '#';
  int result = 1;
  for (int k =0; k <4; k++){
    int i1 = i + di[k];
    int j1 = j + dj[k];
    if (i1<0||i1>=n||j1<0||j1>=n ||d[i1][j1]=='#') continue;
    int r = 0;
    if (d[i1][j1]==p){
      r = dfs(i1,j1);
    }
    else
    if (d[i1][j1]=='.') r = -1;
    else if (d[i1][j1]!=p) {is_surrounded = true;}
    if (r==-1) result = -1;
    else result+=r;
  }
  return result;
}

void main(){
char p;
cin>>p;
if (p=='Ч') p = 'Б'; else p = 'Ч';
//чтение массива d
int answer = 0;
for (int i = 0; i <0; i++)
```

```
  for (int j = 0; j < 0; j++)
    if (d[i][j]==p){
    is_surrounded = false;
    int r = dfs(i,j,p);
    if (r !=-1 && is_surrounded) answer+=r;
   }
cout<<answer;
}
```

Задача 5

Самое простое решение для данной задачи: вывести “2” и n-1 “0”. Пример: пусть n=4. Тогда

$\frac{2000}{2^4} = \frac{2*10*10*10}{2*2*2*2}$, каждая 10 делится на 2, и первая двойка делится на 2.

Следующий вариант решения: возвести 2 в степень n любым стандартным способом, сохранив результат в переменную типа int64, посчитать k=количество цифр в полученном числе, вывести это число 2^n и следом за ним n-k “0”.

Задача 6

Для решения необходимо выполнить данную программу, используя вместо тупика стек (можно моделировать его с помощью обыкновенного массива).

```
int n;
cin >> n;
char * s = new char s [n*2+1];      // программа для тупика
int * v = new int [n];              // вагоны на выходе
int * iv = new int [n];
for ( int i=0; i < n; i++ )   // вагоны на входе
   iv [i] = i+1;
stack <int> st;               // тупик
int j=0, k=0;
for ( int i=0; i < 2*n; i++ )
   if ( s [i] == 'S' )
      st.push (k++);
   else
   {
      v [j++] = st.top ();
      st.pop ();
   }
for ( int i=0; i < n; i++ )
   cout << v [i];
```

Межрегиональная предметная олимпиада КФУ
по предмету «Информатика»
заключительный этап (ответы)
2019-2020 учебный год
11 класс

Задача 1

Для решение задачи применяется метод динамического программирования. Создадим матрицу mas[n][m], в которой для каждой клетки с индексами i и j буем хранить минимальное количество топлива, чтобы добраться с левого верхнего угла в аэродром с индексами i и j. Изначально заполним его очень большими значениями. Создадим ещё трёхмерный массив p[n][m][2]. В p[i][j][0] будем хранить индекс строки аэродрома, из которого прилетел Дрон с минимальным суммарным топливом, а в p[i][j][1] – номер столбца этого аэродрома.

Пусть в массиве a[n][m] хранится заданная матрица. Тогда:

```
mas[0][0]=a[0][0];
        for (int i=0;i<n;i++)
        {
                for (int j=0; j<m; j++)
                {
                        if (a[i][j]!=-1)
                        {
                                for (int k=j+1;k < min(j+a[i][j], m); k++)
                                        if (a[i][k]!=-1)
                                                if (mas[i][k]<mas[i][j]+a[i][k])
                                                {
                                                        mas[i][k]=mas[i][j]+a[i][k];
                                                        p[i][k][0]=i+1;
                                                        p[i][k][1]=j+1;
                                                }

                                for (int k=i+1;k < min(i+a[i][j], n); k++)
                                        if (a[k][j]!=-1)
                                                if (mas[k][j]<mas[i][j]+a[k][j])
                                                {
                                                        mas[k][j]=mas[i][j]+a[k][j];
                                                        p[k][j][0]=i+1;
                                                        p[k][j][1]=j+1;
                                                }
                        }
                }
        }
```

Теперь напишем вспомогательную функцию, которая поможет восстановить пусть с индексами нужных аэродромов. Для простоты напишем её рекурсивно.

```
void path(int p[4][4][2], int i, int j)
{
        if (!(i!=0 and j!=0))
        {
```

```
            path(p, p[i][j][0], p[i][j][1]);
        }
        cout<<i+1<<' '<<j+1<<' ';
}
```

Задача 2

Для заполнения матрицы числами Фибоначчи в заданном порядке заведём символьную переменную d, которая будет следить, в каком направлении мы сейчас движемся. В матрицу mas[m][m] будем хранить нужную матрицу из чисел Фибоначчи. При заполнении важно

Возможные значения d:

‘u’ –поднимаемся по диагонали наверх

‘b’ – спускаемся по диагонали вниз

Часть кода:

```
int f1=0, f2=1, f, mas[9][9];
        int m;
        cin>>m;

        mas[0][0]=0;
        int i=0, j=1; //индексы для матрицы
        char d='b';
        int k=1;
        int l=2; //количество чисел, которые нужно вывести для данной диагонали
        while(k<=m*m)
        {
                if (l<m)
                {
                        if (d=='b')
                        {
                                for(int t=0; t<l;t++)
                                {
                                        k++;
                                        f=f1+f2;
                                        f1=f2;
                                        f2=f;
                                        mas[i][j]=f;
                                        i++;
                                        j--;
                                }
                                d='u';
                                j++;
                                l++;
                                continue;
                        }
                        if (d=='u')
                        {
                                for(int t=0; t<l;t++)
                                {
                                        k++;
```

```
                        f=f1+f2;
                        f1=f2;
                        f2=f;
                        mas[i][j]=f;
                        i--;
                        j++;
                }
                d='b';
                i--;
                l++;
                continue;
        }
}
else
{
        if (d=='b')
        {
                for(int t=0; t<l;t++)
                {
                        k++;
                        f=f1+f2;
                        f1=f2;
                        f2=f;
                        mas[i][j]=f;
                        i++;
                        j--;
                }
                d='u';
                j+=2;
                i++;
                l--;
                continue;
        }
        if (d=='u')
        {
                for(int t=0; t<l;t++)
                {
                        k++;
                        f=f1+f2;
                        f1=f2;
                        f2=f;
                        mas[i][j]=f;
                        i--;
                        j++;
                }
                d='b';
                i+=2;
                j--;
                l--;
                continue;
        }
}
```

```
        }
```

//вывод матрицы ...

Задача 3

Задача на разбор арифметического выражения, решаться может по-разному, например, алгоритмом рекурсивный спуск (другой вариант: обратная польская запись). Будем считать, что у нас есть строка s, в которой сохранено выражение. Добавим в конец строки символ решетка ’#’. Пусть переменная j будет хранить текущий индекс. Сначала напишем функцию, которая вычисляет значение выражения до того, как не встретит символ не цифра и не + или -, это будет функция expression(). Для этого переберем части строки разделенные + или -. В качестве элементов могут быть числа или строка связанная операциями * или /. Ее напишем по аналогии и назовем числа addend(). Уже ее аргументами могут быть числа или целое выражение, заключенное в скобки.

```
String s;
int j;
bool error = false;
int addend();
int expression(){
  int k = 1;
  if (s[j] ==’-’) {
    k = -1; j++;
  }
  int result = k*addend();
  if (error) return 0;
  char c = s[j];
  while (c == ‘+’ || c== ‘-’){
    j++;
    int a = addend();
    if (error) return 0;
    if (c==’-’) a = -a;
    result = result + a;
    c = s[j];
  }
```

```
  return result;
}

int addend(){
  int result = multiplier();
  if (error) return 0;
  char c = s[j];
  while (c == '*' || c== '/'){
    j++;
    int a = multiplier();
    if (error) return 0;
    if (c=='*') result = result * a;
    else result = result / a;
    c = s[j];
  }
  return result;
}
int multiplier(){
  int result = 0;
  if (s[j]=='('){
    j++;
    result = expression();
    if (s[j]==')'){ j++;)
    else {error = true; return 0;}
  }
  else result = number();
  return result;
}

int number(){
  int result;
```

```
  while (s[j]>='0' && s[j]<='9'){
    result = result*10 + ((int)s[j] – (int)'0');
    j++;
  }
  return result;
}

int main(){
  cin<<s;
  s = s+"#";
  j = 0;
  int result = expression();
  if (s[j]!='#') error = true;
  if (error) cout<<"Error";
  else cout<<result;
}
```

Задача 4

Задача решается с помощью поиска в глубину (или в ширину). Достаточно запустить поиск в глубину из каждой точкой цвета противника. Если на границе связной области фишки другого цвета, то они окружены. Если есть свободная точка, то тогда группа не окружена. Напишем функцию dfs, которая будет возвращать число фишек в группе или -1, если группа расширяем. Будем считать, что матрица сохранена в массиве d.

```
int n = 19;
char d[n][n];
int di[4] = {0, 0, +1, -1};
int dj[4] = {-1, +1, 0, 0};
bool is_surrounded = true;
int dfs(int i, int j, char p){
  d[i][j] = '#';
  int result = 1;
  for (int k =0; k <4; k++){
```

```
    int i1 = i + di[k];
    int j1 = j + dj[k];
    if (i1<0||i1>=n||j1<0||j1>=n ||d[i1][j1]==’#’) continue;
    int r = 0;
    if (d[i1][j1]==p){
      r = dfs(i1,j1);
    }
    else
    if (d[i1][j1]==’.’) r = -1;
    else if (d[i1][j1]!=p) {is_surrounded = true;}
    if (r==-1) result = -1;
    else result+=r;
  }
  return result;
}

void main(){
char p;
cin>>p;
if (p==’Ч’) p = ’Б’; else p = ’Ч’;
//чтение массива d
int answer = 0;
for (int i = 0; i <0; i++)
  for (int j = 0; j < 0; j++)
    if (d[i][j]==p){
    is_surrounded = false;
    int r = dfs(i,j,p);
    if (r !=-1 && is_surrounded) answer+=r;
   }
cout<<answer;
}
```

Задача 5

Самое простое решение для данной задачи: вывести “2” и n-1 “0”. Пример: пусть n=4. Тогда

$\frac{2000}{2^4} = \frac{2*10*10*10}{2*2*2*2}$, каждая 10 делится на 2, и первая двойка делится на 2.

Следующий вариант решения: возвести 2 в степень n любым стандартным способом, сохранив результат в переменную типа int64, посчитать k=количество цифр в полученном числе, вывести это число 2^n и следом за ним n-k “0”.

Задача 6

Будем строить программу для тупика последовательно, стараясь получить очередной вагон на выходе. Чтобы вагон с номером V_i попал на выход, он должен находиться либо в вершине стека (последним попал туда), либо должен быть еще где-то на входе. Тогда мы его (и все предыдущие вагоны на входе) последовательно поместим в тупик. Если на каком-то шаге это не удастся сделать, значит, такую программу невозможно построить.

```
int n;
cin >> n;
char * s = new char s [n*2+1];      // искомая программа, если получится
int * v = new int [n];
for ( int i=0; i < n; i++ )   // вагоны на выходе
   cin >> v [i];
int * iv = new int [n];
for ( int i=0; i < n; i++ )   // вагоны на входе
   iv [i] = i+1;
stack <int> st;               // тупик
bool ok = true;
int j=0, k=0;     // указатель на результат и очередной входной вагон
for ( int i=0; i < n; i++ )
{
   if ( ! st.empty () && st.top () == v [i] )  // вагон в верхушке тупика
   {
      s [j++] = 'X'; st.pop ();
   }
   else if ( v [i] >= k )           // вагон где-то на входе
   {
      while ( v [i] >= k )
      {
         st.push (iv [k++]); s [j++] = 'S';
      }
      s [j++] = 'X'; st.pop ();
   }
   else           // вагон где-то в тупике, но не последний!
   {
      ok = false; break;
   }
}
if ( ok )
   cout << s;
else
   cout << "NO";
```