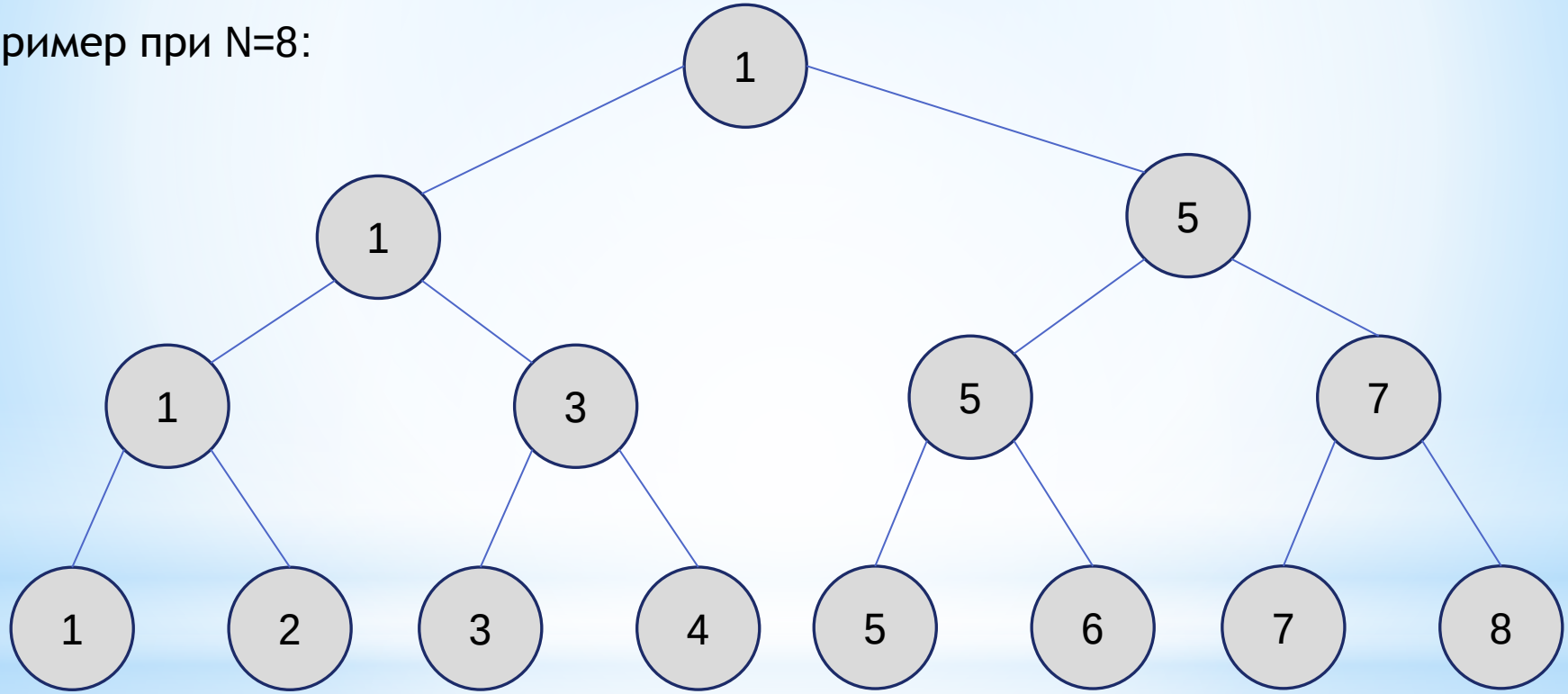
Вологодская область
I Областная олимпиада школьников по информатике
2016-2017 учебный год
9-10 классы
Заключительный тур

Разбор задач

Задача 1 – Игра на вылет

Частный случай: N – степень двойки.
Для удобства будем считать, что матчи разбиты на туры. В каждом туре команды делятся на пары и играют друг с другом, половина команд выбывает.

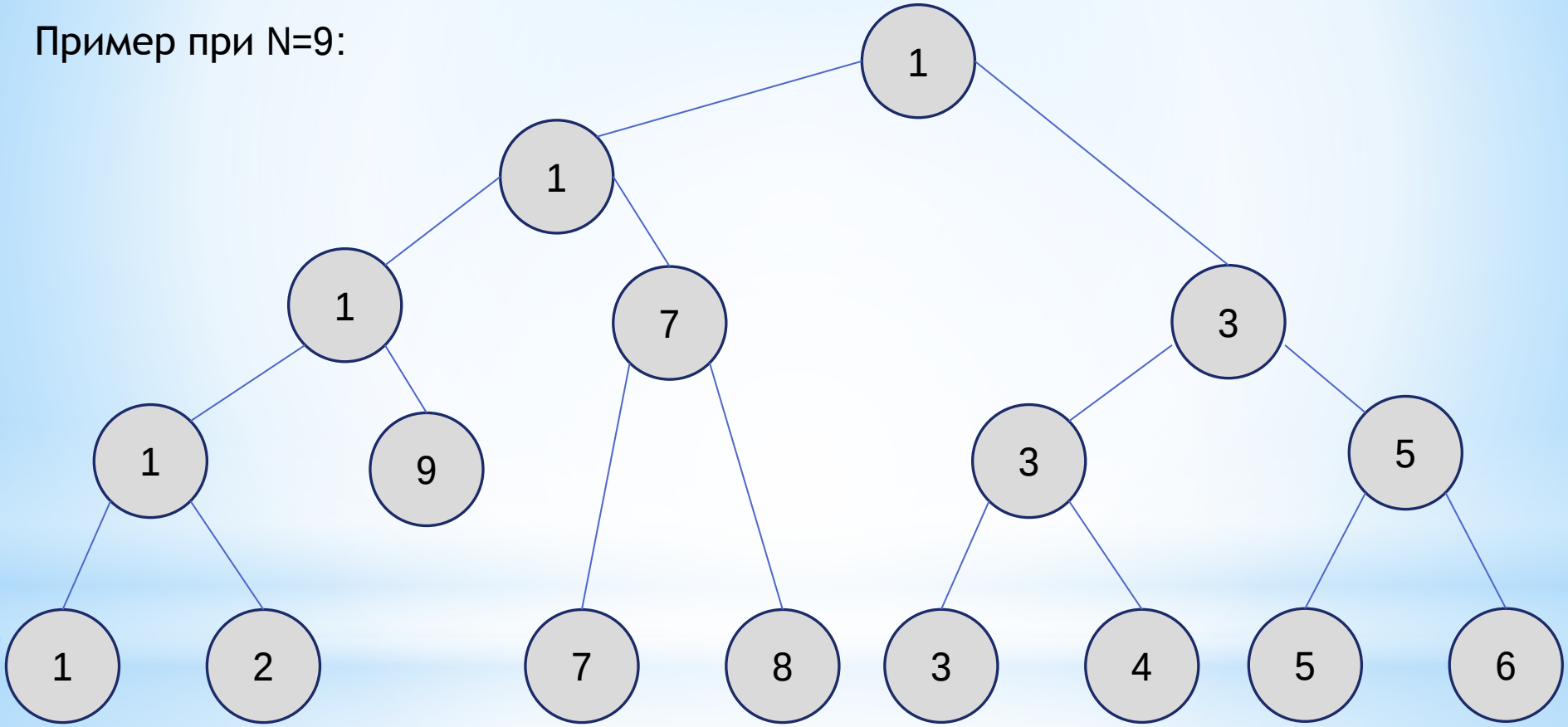
Пример при N=8:



Ответ – число туров, то есть высота дерева, то есть $\log_2(n)$

Общий случай: N – любое число.
В первом туре команды делятся на пары. Если N – нечётное, то одна команда сразу проходит во второй тур, в котором она обязательно сыграет (и будем считать, что проиграет). Пусть также первая команда всегда выигрывает.

Пример при N=9:



Ответ – число туров, то есть высота дерева, то есть $\lceil \log_2(n) \rceil$ (где символы [ ] означают округление в большую сторону)

Ответы к задаче: 3 6 8 20 31

Задача 2 – Сумма

Для решения нужно знать формулу суммы арифметической прогрессии:

$$S = (x_1 + x_n) * n / 2,$$

где x_1 и x_n – первый и последний член последовательности, n – кол-во членов.

Проблема: как найти x_1, x_n, n без оператора if?

Решение

$$x_1 = A/2*2+1$$

*Например, 4/2*2+1=5, 5/2*2+1=5*

$$x_n = (B-1)/2*2+1$$

*Например, (7–1)/2*2+1=7, (8–1)/2*2+1=7*

$$n = (x_n - x_1 + 2) / 2$$

Подставив в верхнюю формулу и сократив, получим ответ:

$$(A/2*2 + (B-1)/2*2+2) * ((B-1)/2-A/2+1) / 2$$

Возможны и другие верные решения, например:

```
((A+1-A%2) + (B-1+B%2)) * (((B-1+B%2) - (A+1-A%2)) / 2 + 1) / 2
```

Задача 3 – Пилообразные числа

Подзадача 1: перебрать все N-разрядные числа, разбить каждое на цифры и проверить пилообразность

Подзадача 2: преподсчёт. Перебрать все 12-разрядные пилообразные числа можно менее чем за минуту (на Intel Core i5), используя перебор с возвратом):

```
const UP = 1; DOWN = 2;
procedure backtrack(prev, n, dir: integer);
var
  d: integer;
begin
  if dir = UP then begin
    for d := prev + 1 to 9 do
      if n > 1 then
        backtrack(d, n - 1, DOWN)
      else inc (ans);
  end else begin
    for d := 0 to prev - 1 do
      if n > 1 then
        backtrack(d, n - 1, UP)
      else
        inc(ans);
  end;
end;
```

На проверку посылается программа с заранее сосчитанными ответами

Подзадача 3: динамическое программирование.

Создадим две матрицы g и l размерности $1..N$ x $0..9$, где:

- $g[i][j]$ – это количество пилообразных последовательностей длины i, которые кончаются цифрой j, а предпоследняя цифра больше последней.

- $l[i][j]$ – это количество пилообразных последовательностей длины i, которые кончаются цифрой j, а предпоследняя цифра меньше последней.

Правила перехода от i к (i+1):

Для всех пар цифр k и j, где $k > j$:

$$l[i + 1][k] \mathrel{+}= g[i][j]$$

Для всех пар цифр k и j, где $k < j$:

$$g[i + 1][k] \mathrel{+}= l[i][j]$$

Текст программы:

```
NMAX = 18;
var
  g, l: array[1..NMAX, 0..9] of Int64;
  n, i, j, k: integer;
  ans: int64;
begin
   // обнуление переменных g, l, ans...
  read(n);
  for i := 1 to 9 do begin
    g[1][i] := 1;
    l[1][i] := 1;
  end;
  for i := 1 to n - 1 do
    for j := 0 to 9 do begin
      for k := j + 1 to 9 do
        l[i + 1][k] := l[i + 1][k] + g[i][j];
      for k := 0 to j - 1 do
        g[i + 1][k] := g[i + 1][k] + l[i][j];
    end;
  for j := 0 to 9 do
    ans := ans + l[n][j] + g[n][j];
  writeln(ans);
end.
```

Задача 4 – Лабиринт

Задача решается методом поиска в ширину (bfs) одновременно от начальных положений всех игроков.

Структуры данных:

- матрица *field* – игровое поле;
- трёхмерный массив *visited:* для каждой клетки и каждого игрока отмечаем, что игрок в этой клетке был;
- матрица *playersCount:* для каждой клетки храним количество игроков, которые эту клетку уже посетили.
- очередь *queue: сюда помещаем записи с четырьмя полями*:
  - координаты клетки *x* и *y*,
  - номер игрока *player*,
  - длина кратчайшего пути до этой клетки *pathLength*.

```
type Point = record
    x, y, player, pathLength: smallint;
end;

var
  field: array[1..NMAX] of String[NMAX];
  visited: array[1..PMAX, 1..NMAX, 1..NMAX] of boolean;
  playersCount: array[1..NMAX, 1..NMAX] of smallint;
  queue: array[1..NMAX * NMAX * PMAX] of Point;
```

Алгоритм:

Заносим в очередь начальные координаты всех игроков.

В массиве *visited* отмечаем, что эти клетки ими посещены.

В матрице *playersCount* ставим в эти клетки по единичке.

Повторяем в цикле следующие действия:

Извлекаем из очереди очередной элемент. Пытаемся сходить влево, вправо, вверх и вниз. Если в соответствующей клетке нет стены, и мы там ещё не бывали (смотрим в *visited*), то делаем следующее:

- заносим её координаты в очередь (увеличивая поле *pathLength* на 1),
- в массиве *visited* отмечаем посещённость клетки,
- в матрице *playersCount* увеличиваем число игроков для этой клетки на 1.

Процесс закончится, когда в матрице *playersCount* впервые появится число >= *K* (или очередь опустеет – тогда решения нет)

Оценка количества действий:

Поскольку каждая тройка <*y*, *x*, *игрок*> может быть добавлена в очередь только один раз, то количество итераций не превышает *M*·*N*·*U* (где *U* – количество игроков на поле).

При максимальных ограничениях задачи это будет 200·200·100= 4 000 000.

Подзадача 1: возможны различные менее эффективные решения – например, двоичным поиском подбирать ответ и запускать bfs по очереди от каждого игрока. В зависимости от реализации такие решения могут набрать баллы, близкие к макс.

Текст программы:

```
  readln(nY, nX, needPlayers);
  for y := 1 to nY do
    readln(field[y]);
  player := 0; qRight := 0;
  for y := 1 to nY do
    for x := 1 to nX do
      if field[y][x] = '*' then begin
        inc(player);
        p.x := x; p.y := y; p.player := player; p.pathLength := 0;
        inc(qRight);
        queue[qRight] := p;
        playersCount[y][x] := 1;
        visited[player][y][x] := true;
      end;
  qLeft := 1;
  while (qLeft <= qRight) do begin
    p := queue[qLeft];
    inc(qLeft);
    for dy := -1 to 1 do
      for dx := -1 to 1 do
        if ((dy<>0)xor(dx<>0)) and (p.y+dy>=1)
          and (p.x+dx>=1) and (p.y+dy<=nY) and (p.x+dx<=nX) then
          if (field[p.y+dy][p.x+dx] <> '#') and (not visited[p.player][p.y+dy][p.x+dx])
          then begin
            pNext.y := p.y + dy;
            pNext.x := p.x + dx;
            pNext.player := p.player;
            pNext.pathLength := p.pathLength + 1;
            inc(playersCount[pNext.y][pNext.x]);
            if (playersCount[pNext.y][pNext.x] >= needPlayers) then begin
              writeln(pNext.pathLength);
              exit;
            end;
            visited[pNext.player][pNext.y][pNext.x] := true;
            inc(qRight);
            queue[qRight] := pNext;
          end;
  end;
  writeln(-1);
end.
```

Задача 5 – Олимпиады

Подзадача 1: полный перебор всех подмножеств

Подзадача 2 и 3: вначале ищем решение среди однотуровых олимпиад. Затем ищем решение среди двухтуровых олимпиад, которые не пересекаются с однотуровыми.
Утверждение: такое решение оптимально.
Доказательство. Пусть нам дано оптимальное решение. Пусть в него входит какая-то двухтуровая олимпиада, и она пересекается с однотуровой (которая в решение не входит). Поменяем эти две олимпиады местами (первую уберём из решения, вторую включим). Очевидно, решение по-прежнему оптимально. Повторяем, пока в решении есть двухтуровые олимпиады, пересекающиеся с однотуровыми.

Вопросы:
1). Как найти максимальное кол-во однотуровых олимпиад?
Ответ: это количество различных дней в году, в которые проводятся однотуровые олимпиады

2). Как найти максимальное кол-во двухтуровых олимпиад?
Рассмотрим решения отдельно для подзадач 2 и 3.

Подзадача 2: все двухтуровые олимпиады без пропусков.

Решение - жадный алгоритм:

Отсортируем все двухтуровые олимпиады по дню проведения. Первую олимпиаду включаем в решение, а пересекающиеся с ней – отбрасываем. Далее действуем аналогично, пока олимпиады не кончатся.

Утверждение:

данный алгоритм даёт оптимальное решение

Доказательство.

Пусть есть оптимальное решение, куда самая ранняя олимпиада не вошла. Уберём из этого решения первую олимпиаду и поставим вместо неё самую раннюю. Решение корректно и по-прежнему оптимально, то есть наш жадный выбор не закрывает путь к оптимальному решению.

Отбросив все олимпиады, которые пересекаются с самой ранней, мы пришли к точно такой же задаче, что и исходная. Тогда по индукции алгоритм верен.

Подзадача 3: имеются двухтуровые олимпиады с промежуточным днём

Решение: используем метод динамического программирования

Структуры данных:

Массивы для входных данных:

- *t*1[*i*] = *true*, если имеется однотуровая олимпиада, идущая в день i,
- *ta*[*i*] = *true*, если имеется двухтуровая олимпиада без промежутка между турами, у которой второй тур идёт в день i,
- *tb*[*i*] = *true*, если имеется двухтуровая олимпиада с промежутком между турами, у которой второй тур идёт в день i,

Массивы для метода динамического программирования:

- *ca*[*i*] – максимальное количество двухтуровых олимпиад, в которых можно поучаствовать в дни 1..*i*,
- *cb*[*i*] – максимальное количество двухтуровых олимпиад, в которых можно поучаствовать в дни 1..*i* при условии, что в день (*i*−1) мы в олимпиадах не участвуем

Правила перехода от меньших подзадач к б*о*льшим:

1. Можно в *i*-й день вообще ни в какой олимпиаде не участвовать. Тогда:

```
ca[i] := ca[i - 1];
cb[i] := ca[i - 2];
```

2. Можно в *i*-й день поучаствовать во втором дне олимпиады без промежуточного дня между турами (если такая в этот день проводится). Для этого нужно, чтобы предыдущий день был свободен:

```
if ta[i] and (ca[i - 2] + 1 > ca[i]) then
    ca[i] := ca[i - 2] + 1;
```

3. Можно в *i*-й день поучаствовать во втором дне олимпиады с промежуточным днём между турами (если такая в этот день проводится). Для этого нужно, чтобы день (*i*−2) был свободен:

```
if tb[i] then begin
    ca[i] := max(ca[i], max(ca[i - 3] + 1, cb[i - 1] + 1));
    cb[i] := max(cb[i], ca[i - 3] + 1);
end;
```

Текст программы:

```
for i := 1 to n do begin
    read(t, d1);
    if t = 1 then t1[d1] := true
    else begin
      read(d2);
      if d1 + 1 = d2 then ta[d2] := true else tb[d2] := true
    end;
  end;
  ans := 0;
  for i := 1 to DMAX do
    if t1[i] then begin
      inc(ans);
      ta[i] := false;
      ta[i + 1] := false;
      tb[i] := false;
      tb[i + 2] := false;
    end;
  if ta[2] then ca[2] := 1;
  for i := 3 to DMAX + 2 do begin
    // если этот день пропустить
    ca[i] := ca[i - 1];
    cb[i] := ca[i - 2];
    // если поучаствовать в олимпиаде без промежуточного дня
    if ta[i] and (ca[i - 2] + 1 > ca[i]) then
      ca[i] := ca[i - 2] + 1;
    // если поучаствовать в олимпиаде со свободным днём
    if tb[i] then begin
      ca[i] := max(ca[i], ca[i - 3] + 1);
      ca[i] := max(ca[i], cb[i - 1] + 1);
      cb[i] := max(cb[i], ca[i - 3] + 1);
    end;
  end;
  ans := ans + max(ca[DMAX + 2], cb[DMAX + 2]);
  writeln(ans);
end.
```