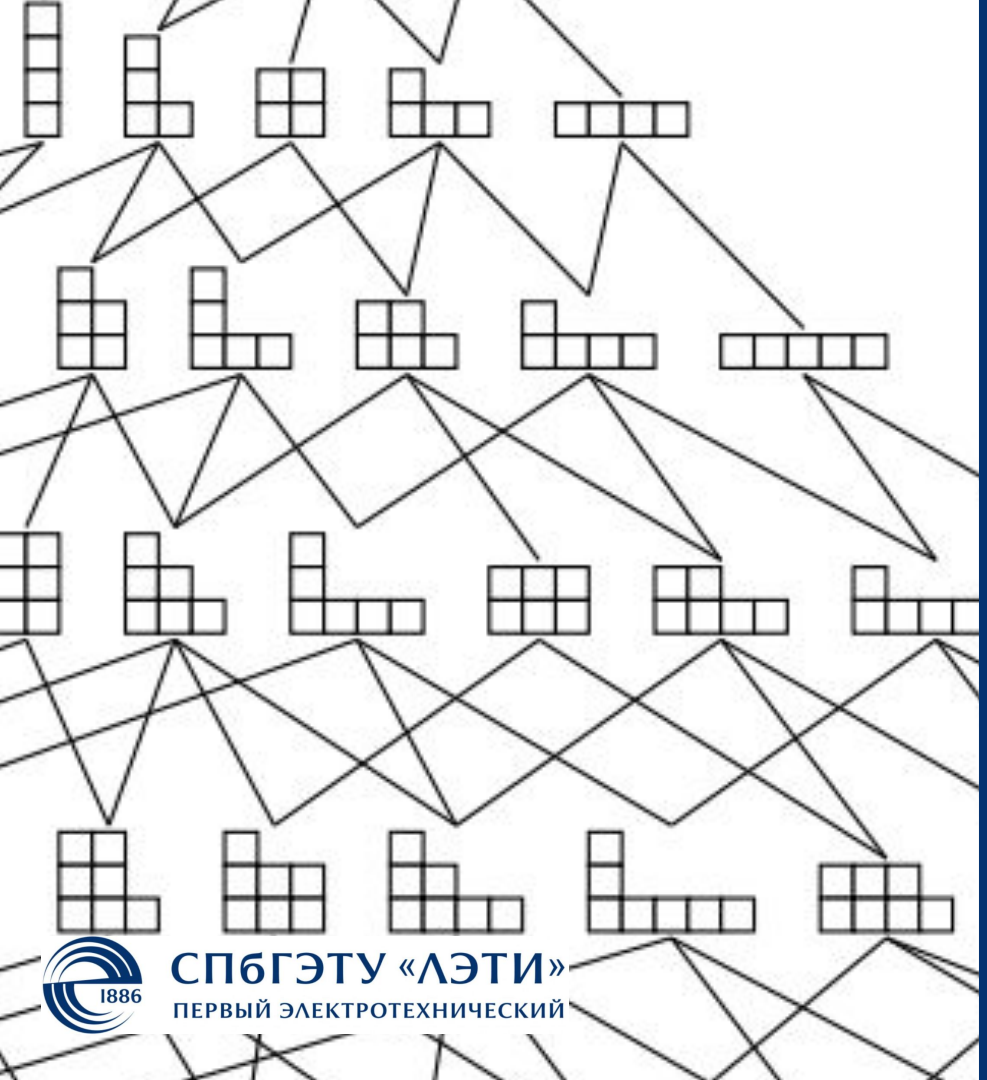
Визуализация начала графа Юнга

Санкт-Петербург
2024

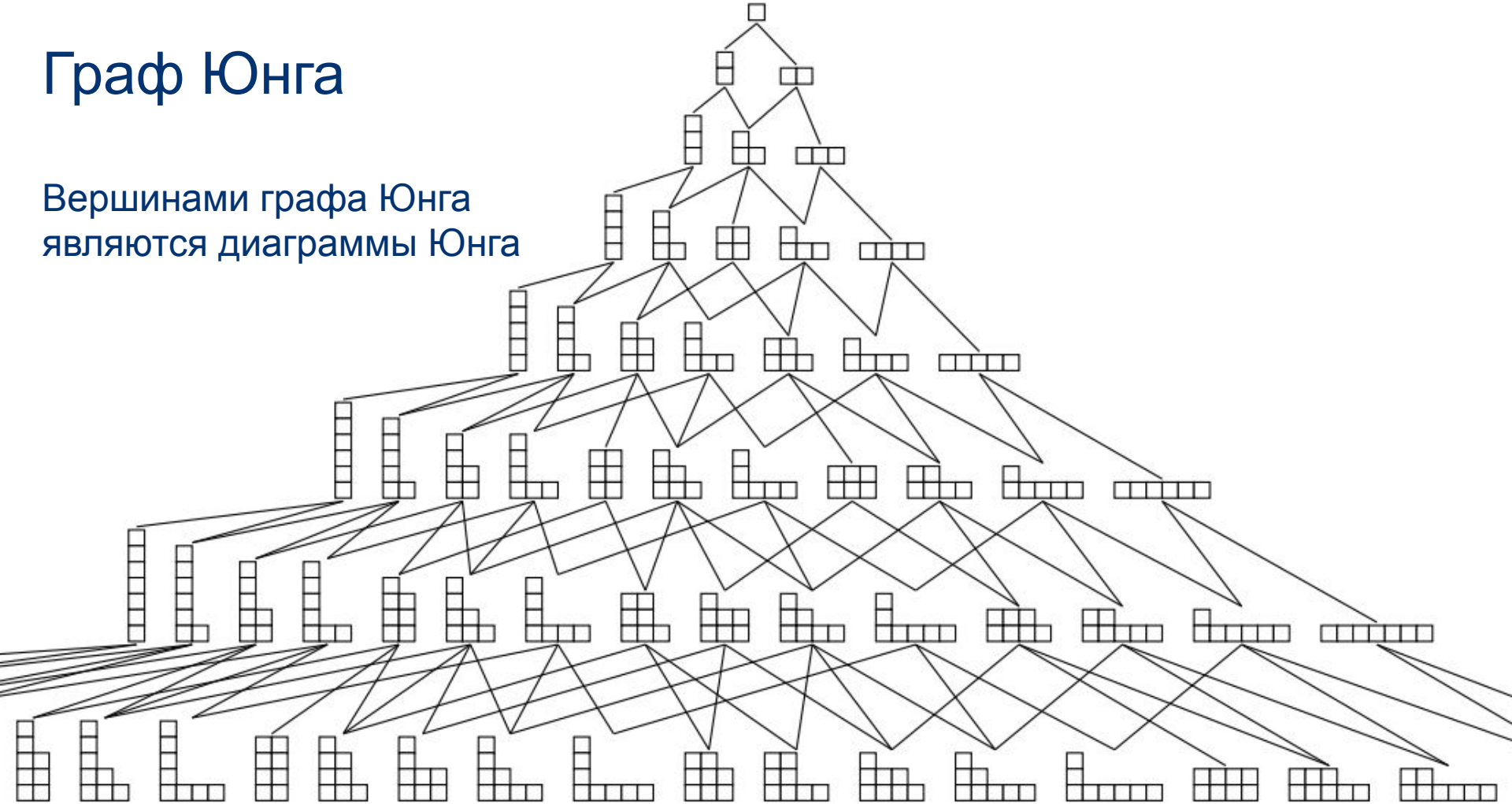


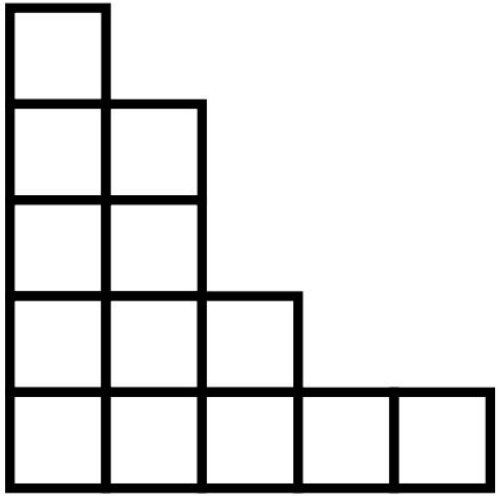
Диаграммы Юнга

— совокупность клеток, выровненных по левому и нижнему краю, соответствующая целочисленному разбиению числа, слагаемые которого упорядочены в порядке невозрастания.

Граф Юнга

Вершинами графа Юнга являются диаграммы Юнга



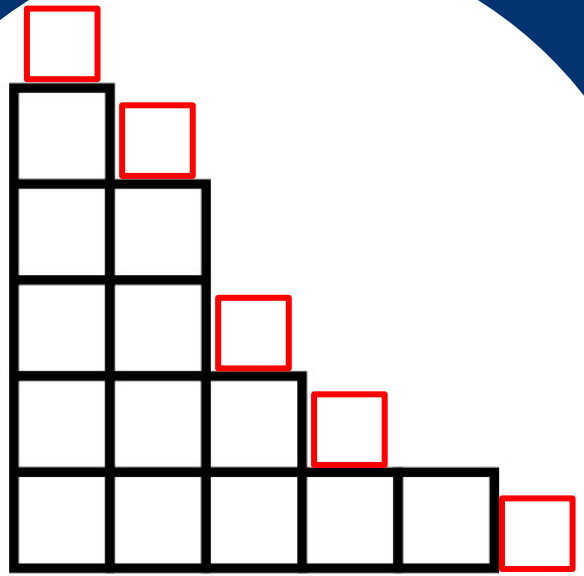


Программная реализация диаграмм Юнга

Диаграмма представлена в памяти как одномерный массив, состоящий из высот ее столбцов.

Диаграмма на рисунке: “542110”

СПбГЭТУ «ЛЭТИ»
ПЕРВЫЙ ЭЛЕКТРОТЕХНИЧЕСКИЙ



Программная реализация диаграмм Юнга

Каждая диаграмма имеет определенный потенциал для получения из нее новых диаграмм.

```
struct Diagram
{
    int * ivals;
    int len;
    int level;
    struct Diagram ** Children;
    int ways;
    int number;
    int CountOfChildren;
    int OutFlag;
    int color;
    int * prime_c;
    int CountOfPrimeChildren;
```

Программа генерации реализована на языке C++. Каждая диаграмма представлена в виде структуры.

```
struct Diagram
{
    int * ivals;
    int len;
    int level;
    struct Diagram ** Children;
    int ways;
    int number;
    int CountOfChildren;
    int OutFlag;
    int color;
    int * prime_c;
    int CountOfPrimeChildren;
```

— массив, хранящий высоты столбцов

```
struct Diagram
{
    int * ivals;
    int len;
    int level;
    struct Diagram ** Children;
    int ways;
    int number;
    int CountOfChildren;
    int OutFlag;
    int color;
    int * prime_c;
    int CountOfPrimeChildren;
```

— длина массива ivals

```
struct Diagram
{
    int * ivals;
    int len;
    int level;
    struct Diagram ** Children;
    int ways;
    int number;
    int CountOfChildren;
    int OutFlag;
    int color;
    int * prime_c;
    int CountOfPrimeChildren;
```

— уровень, на котором находится диаграмма

```
struct Diagram
{
    int * ivals;
    int len;
    int level;
    struct Diagram ** Children;
    int ways;
    int number;
    int CountOfChildren;
    int OutFlag;
    int color;
    int * prime_c;
    int CountOfPrimeChildren;
```

— массив исходящих ребер, хранящий ссылки на диаграммы следующего уровня, которые можно получить из данной диаграммы

```
struct Diagram
{
    int * ivals;
    int len;
    int level;
    struct Diagram ** Children;
    int ways;
    int number;
    int CountOfChildren;
    int OutFlag;
    int color;
    int * prime_c;
    int CountOfPrimeChildren;
```

— количество путей, по которым можно дойти до данной диаграммы из корня графа

```
struct Diagram
{
    int * ivals;
    int len;
    int level;
    struct Diagram ** Children;
    int ways;
    int number;
    int CountOfChild
    int OutFlag;
    int color;
    int * prime_c;
    int CountOfPrimeChildren;
```

— номер появления диаграммы при генерации графа

```
struct Diagram
{
    int * ivals;
    int len;
    int level;
    struct Diagram ** Children;
    int ways;
    int number;
    int CountOfChildren;
    int OutFlag;
    int color;
    int * prime_c;
    int CountOfPrimeChildren;
```

— длина массива исходящих ребер

```
struct Diagram
{
    int * ivals;
    int len;
    int level;
    struct Diagram ** Children;
    int ways;
    int number;
    int CountOfChildren;
    int OutFlag;
    int color;
    int * prime_c;
    int CountOfPrimeChildren;
```

— флаг, обозначающий то, была ли диаграмма отправлена в визуализатор

```
struct Diagram
{
    int * ivals;
    int len;
    int level;
    struct Diagram ** Children;
    int ways;
    int number;
    int CountOfChildren;
    int OutFlag;
    int color;
    int * prime_c;
    int CountOfPrimeChildren;
```

— цвет диаграммы

```
struct Diagram
{
    int * ivals;
    int len;
    int level;
    struct Diagram ** Children;
    int ways;
    int number;
    int CountOfChildren;
    int OutFlag;
    int color;
    int * prime_c;
    int CountOfPrime
```

— массив номеров исходящих ребер, которые ведут к диаграммам, впервые сгенерированным из данной диаграммы

```
struct Diagram
{
    int * ivals;
    int len;
    int level;
    struct Diagram ** Children;
    int ways;
    int number;
    int CountOfChildren;
    int OutFlag;
    int color;
    int * prime_c;
    int CountOfPrimeChildren;
```

— длина массива prime_c

Генерация графа

Генерация происходит аналогично обходу графа в глубину.
Функция AddChildren создает диаграммы, которые можно получить из переданной в качестве аргумента.

```
void AddChildren(int levels, drgm * El, drgm * first, head * Head, 
{
    for (int j = 0; j < El->len; j++)
    {
        if (j == 0 || j == El->len-1 || El->ivals[j-1] > El->ivals[
        {

            drgm * Child;
            Child = new(drgm);

            for (int i = 0; i < El->len; i++)
            {
                Child->ivals[i] = El->ivals[i];
            }

            Child->len = El->len;
            Child->ivals[j]++;

            Child->level = (El->level + 1);
            Child->OutFlag = 0;

            if (Child->ivals[Child->len - 1] != 0)
            {
                Child->ivals[Child->len] = 0;
                Child->len++;
            }

            bool accordance = true;
```

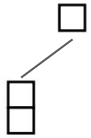
СПбГЭТУ «ЛЭТИ»
ПЕРВЫЙ ЭЛЕКТРОТЕХНИЧЕСКИЙ

Алгоритм генерации

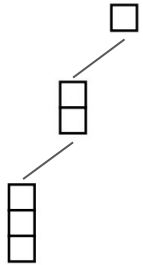
Алгоритм генерации

На первом шаге существует только корневой элемент.
Для него генерируется 1 потомок.

Алгоритм генерации

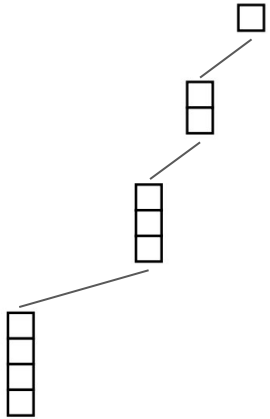


На первом шаге существует только корневой элемент.
Для него генерируется 1 потомок.



Алгоритм генерации

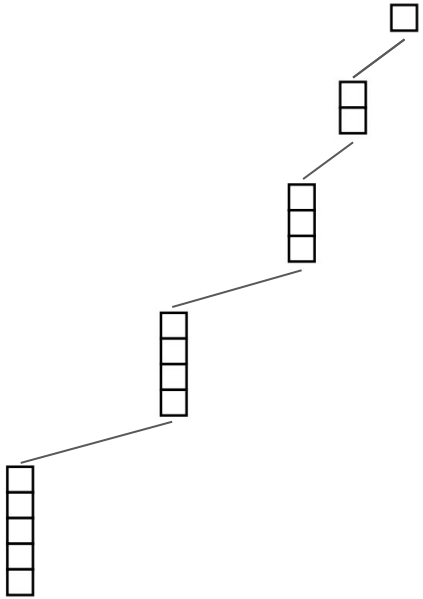
После генерации для новой диаграммы сразу же вызывается AddChildren.
Поэтому на втором шаге мы получим диаграмму с высотами столбцов “30”.



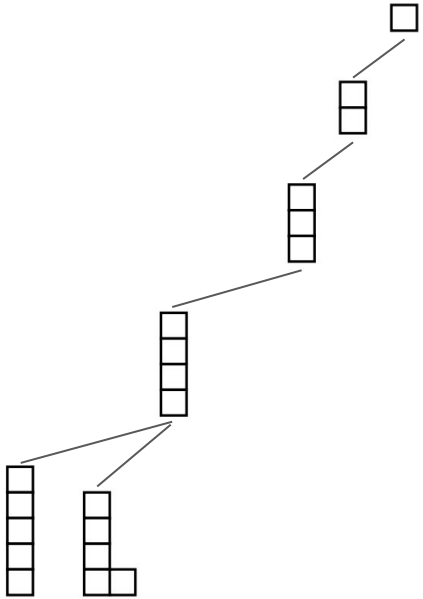
Алгоритм генерации

Далее диаграмму с высотами столбцов “40”

Алгоритм генерации

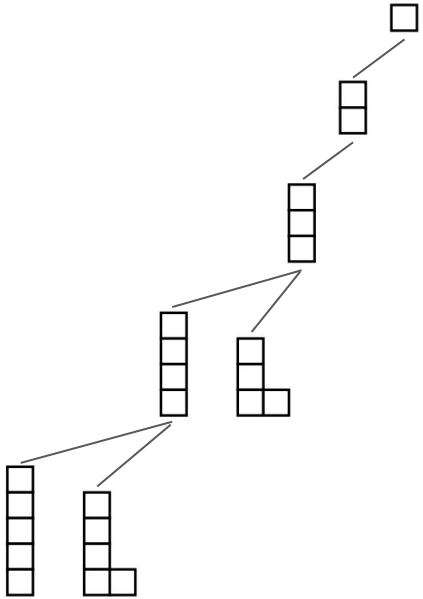


Далее диаграмму с высотами столбцов “40” и “50”.



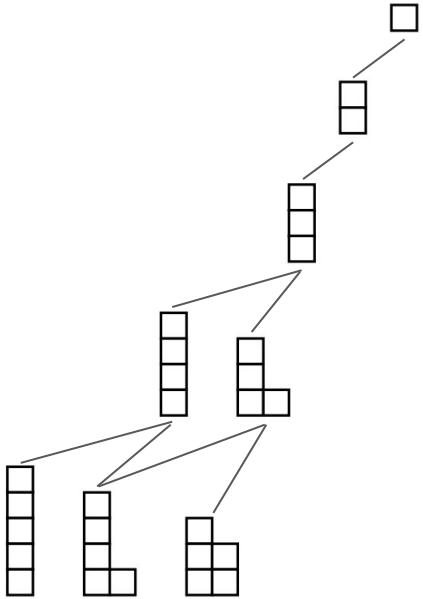
Алгоритм генерации

Установим ограничение в 5 этажей. Тогда для диаграммы “50” функция AddChild не вызывается. Поэтому следующей появится диаграмма, полученная из “40”.



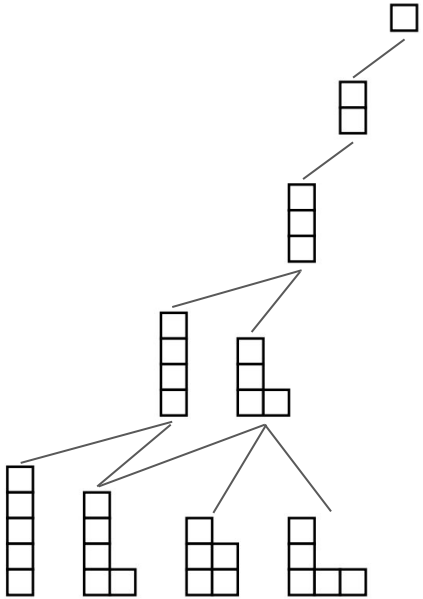
Алгоритм генерации

Так как из “40” мы больше не можем ничего получить, то происходит генерация от диаграммы, добавленной ранее.



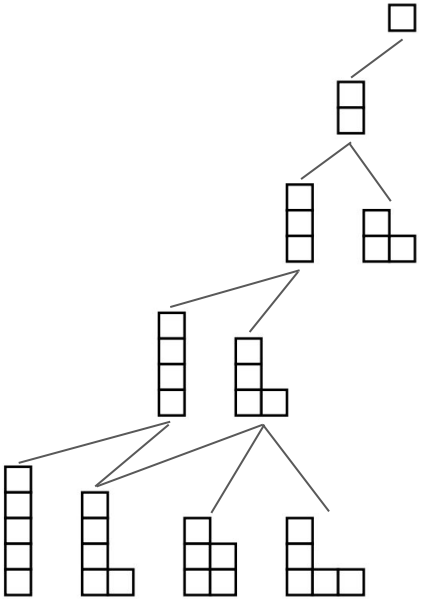
Алгоритм генерации

Функция AddChildren вызывается для добавленной диаграммы “310”.
Так как диаграмма “410” уже была получена, то вместо добавления новой достраивается связь.



Алгоритм генерации

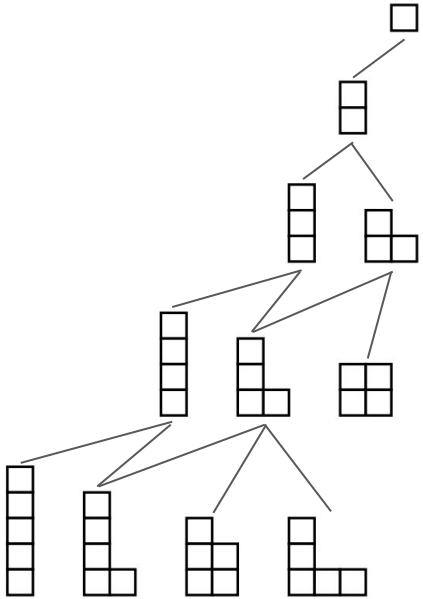
Функция AddChildren вызывается для добавленной диаграммы “310”.



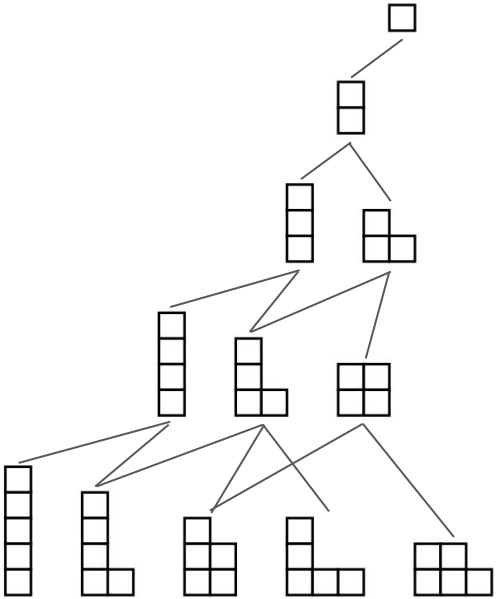
Алгоритм генерации

Из “310” и “30” больше нельзя получить диаграмм, двигаемся дальше.

Алгоритм генерации

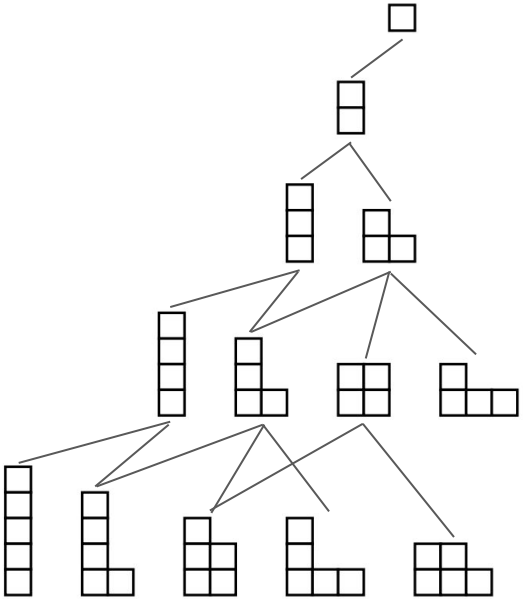


Достраиваем связь от “210” к “310” и генерируем новую диаграмму “220”.



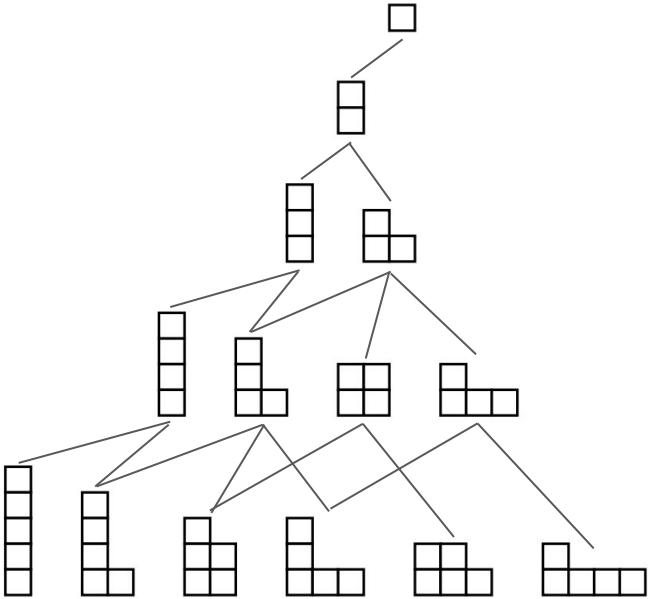
Алгоритм генерации

Генерация продолжается до тех пор, пока хотя бы из одной диаграммы можно сгенерировать новую.



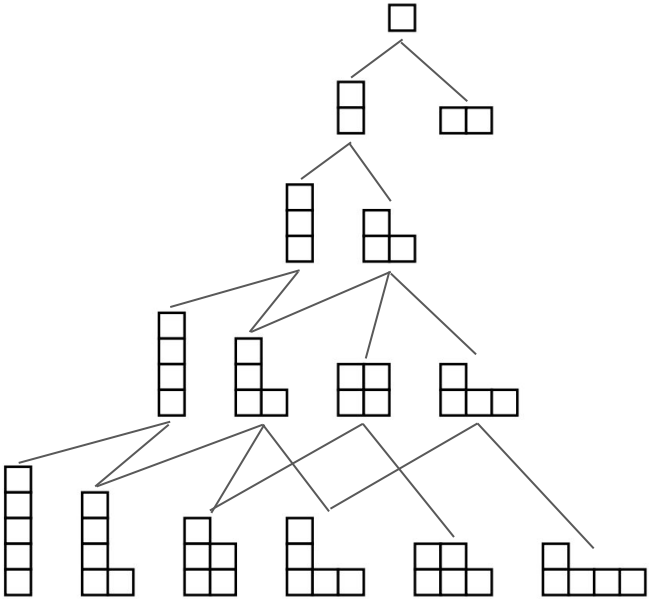
Алгоритм генерации

Генерация продолжается до тех пор, пока хотя бы из одной диаграммы можно сгенерировать новую.



Алгоритм генерации

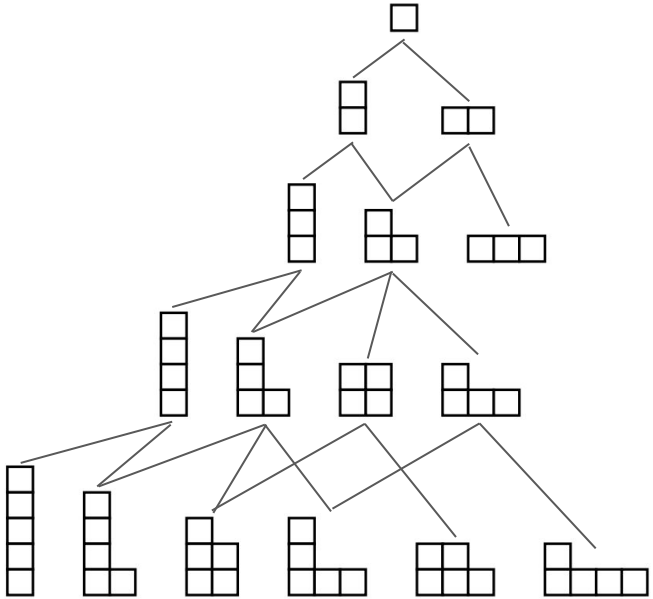
Генерация продолжается до тех пор, пока хотя бы из одной диаграммы можно сгенерировать новую.



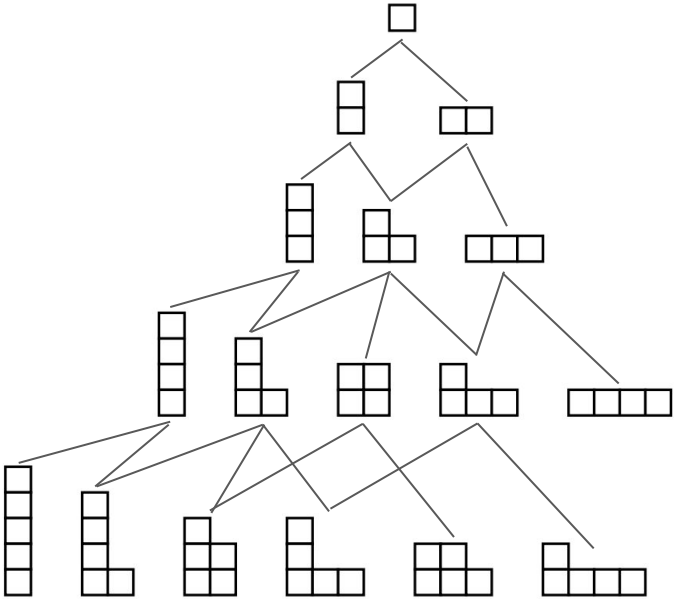
Алгоритм генерации

Генерация продолжается до тех пор, пока хотя бы из одной диаграммы можно сгенерировать новую.

Алгоритм генерации

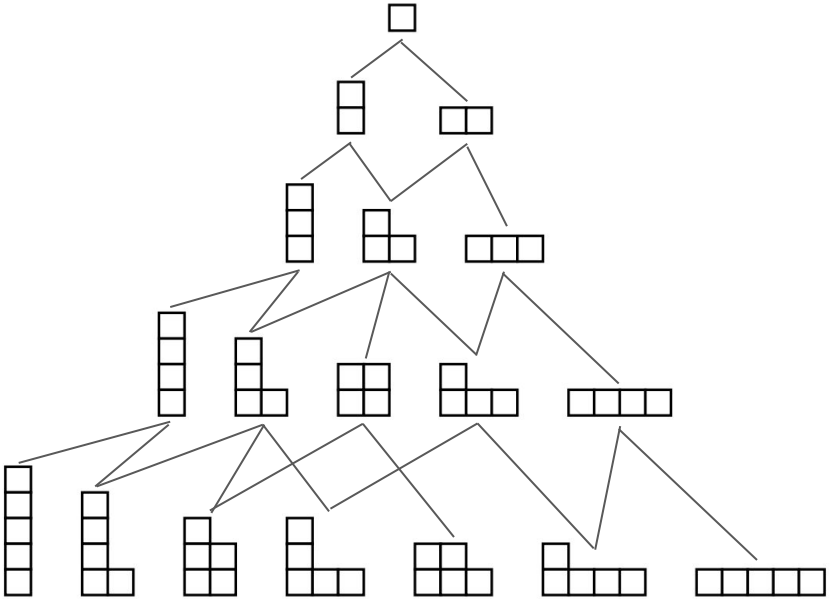


Генерация продолжается до тех пор, пока хотя бы из одной диаграммы можно сгенерировать новую.



Алгоритм генерации

Генерация продолжается до тех пор, пока хотя бы из одной диаграммы можно сгенерировать новую.



Алгоритм генерации

Возможностей для создания новых диаграмм не осталось, значит генерация окончена.

Визуализация

После того, как список диаграмм был сгенерирован, наступает очередь визуализации.

Программа-визуализатор написана на языке программирования Python.

```
for e in range(0, len(parts[b])):
    y += (b + 1 - int(parts[b][e][0])) * 10

    fy = y
    fx = x
    fstart_y = fy
    r k in range(0, len(parts[b][e])):
        for g in range(0, int(parts[b][e][k])):

            if color_necessary:
                graph.add(graph.rect((fx, fy + (10 * g)), (10, 10
            lse:
                graph.add(graph.rect((fx, fy + (10 * g)), (10, 10

            += 10
        f k < len(parts[b][e]) - 1:
            fstart_y = fstart_y + 10 * (int(parts[b][e][k]) - int
            fy = fstart_y

    x_in = x + 5 * len(parts[b][e])
    y_in = y - 2
    x_out = x + 5 * len(parts[b][e])
    y_out = y + 10 * int(parts[b][e][0]) + 2
    ver_coords[str_parts[b][e]] = [x_in, y_in, x_out, y_out]
```

```
'--args0' '00' '--args1' '1$0$#000000' '2$0$#000000'
'1;1$0$#000000' '3$0$#000000' '2;1$1$#000000' '1;1;1
$0$#000000' '4$0$#000000' '3;1$1$#000000' '2;2
$0.666667$#000000' '2;1;1$1$#000000' '1;1;1;1$0$#
000000' '5$0$#000000' '4;1$0.666667$#000000' '3;2
$0.833333$#000000' '3;1;1$1$#000000' '2;2;1$0.833333
$#000000' '2;1;1;1$0.666667$#000000' '1;1;1;1;1$0$#
000000' '--args2' '0110000000000000000'
'0001100000000000000' '0000110000000000000'
'0000011000000000000' '0000001110000000000'
'0000000011000000000' '0000000000110000000'
'0000000000001110000' '0000000000000010100'
'0000000000000001110' '0000000000000000011'
'0000000000000000000' '0000000000000000000'
'0000000000000000000' '0000000000000000000'
'0000000000000000000' '0000000000000000000'
'0000000000000000000'
```

Входные данные

Программа получает данные в виде строки, в которой значения разделены на блоки, на которые указывают флаги вида “--args”.

```
'--args0' '00' '--args1' '1$0$#000000' '2$0$#000000'
'1;1$0$#000000' '3$0$#000000' '2;1$1$#000000' '1;1;1
$0$#000000' '4$0$#000000' '3;1$1$#000000' '2;2
$0.666667$#000000' '2;1;1$1$#000000' '1;1;1;1$0$#
000000' '5$0$#000000' '4;1$0.666667$#000000' '3;2
$0.833333$#000000' '3;1;1$1$#000000' '2;2;1$0.833333
$#000000' '2;1;1;1$0.666667$#000000' '1;1;1;1;1$0$#
000000' '--args2' '0110000000000000000'
'0001100000000000000' '0000110000000000000'
'0000011000000000000' '0000000111000000000'
'0000000011000000000' '0000000000011000000'
'0000000000000111000' '0000000000000010100'
'0000000000000001110' '0000000000000000011'
'0000000000000000000' '0000000000000000000'
'0000000000000000000' '0000000000000000000'
'0000000000000000000' '0000000000000000000'
'0000000000000000000'
```

Входные данные

‘--args0’ свидетельствует о том, что следующий аргумент - системные флаги вида ‘00’
Если первое значение 0 - не нужно отображать ребра, если 1, то они должны быть.
Второе значение отвечает за отображение цвета.

```
'--args0' '00' '--args1' '1$0$#000000' '2$0$#000000'
'1;1$0$#000000' '3$0$#000000' '2;1$1$#000000' '1;1;1
$0$#000000' '4$0$#000000' '3;1$1$#000000' '2;2
$0.666667$#000000' '2;1;1$1$#000000' '1;1;1;1$0$#
000000' '5$0$#000000' '4;1$0.666667$#000000' '3;2
$0.833333$#000000' '3;1;1$1$#000000' '2;2;1$0.833333
$#000000' '2;1;1;1$0.666667$#000000' '1;1;1;1;1$0$#
000000' '--args2' '0110000000000000000'
'0001100000000000000' '0000110000000000000'
'0000011000000000000' '0000001110000000000'
'0000000110000000000' '0000000001100000000'
'0000000000111000000' '0000000000010100000'
'0000000000001110000' '0000000000000011000'
'0000000000000000000' '0000000000000000000'
'0000000000000000000' '0000000000000000000'
'0000000000000000000' '0000000000000000000'
'0000000000000000000'
```

Входные данные

‘--args1’ означает, что дальше идет последовательность аргументов, отвечающих за диаграммы.
Они имеют вид:
‘массив высот столбцов, разделенных “;” $ насыщенность цвета $ код цвета’

```
'--args0' '00' '--args1' '1$0$#000000' '2$0$#000000'
'1;1$0$#000000' '3$0$#000000' '2;1$1$#000000' '1;1;1
$0$#000000' '4$0$#000000' '3;1$1$#000000' '2;2
$0.666667$#000000' '2;1;1$1$#000000' '1;1;1;1$0$#
000000' '5$0$#000000' '4;1$0.666667$#000000' '3;2
$0.833333$#000000' '3;1;1$1$#000000' '2;2;1$0.833333
$#000000' '2;1;1;1$0.666667$#000000' '1;1;1;1;1$0$#
000000' '--args2' '0110000000000000000'
'0001100000000000000' '0000110000000000000'
'0000011000000000000' '0000001110000000000'
'0000000110000000000' '0000000001100000000'
'0000000000111000000' '0000000000010100000'
'0000000000001110000' '0000000000000000011'
'0000000000000000000' '0000000000000000000'
'0000000000000000000' '0000000000000000000'
'0000000000000000000' '0000000000000000000'
'0000000000000000000'
```

Входные данные

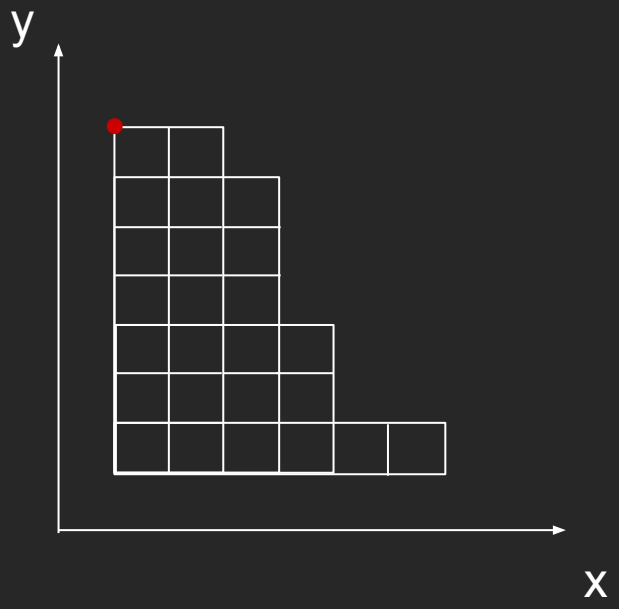
‘--args2’ означает, что следующие аргументы - строки матрицы смежности.

Алгоритм визуализации

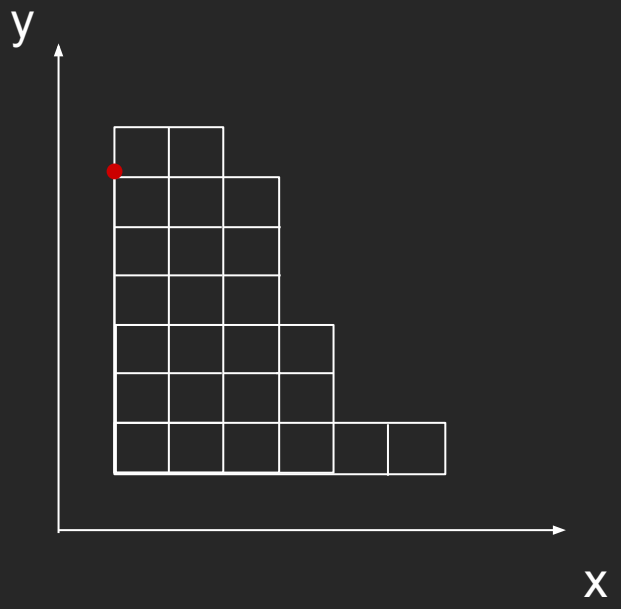
Программа движется по списку аргументов из поля ‘--args1’.
Для отрисовки используется библиотека svgwrite.
На каждом этапе рассчитываются координаты, после чего по ним рисуется квадрат заданных размеров.

```
fy = y
fx = x
fstart_y = fy
for k in range(0, len(parts[b][e])):
    for g in range(0, int(parts[b][e][k])):

        if color_necessary:
            graph.add(graph.rect((fx, fy + (10 * g))
        else:
            graph.add(graph.rect((fx, fy + (10 * g))

    fx += 10
    if k < len(parts[b][e]) - 1:
        fstart_y = fstart_y + 10 * (int(parts[b][e][
        fy = fstart_y

x_in = x + 5 * len(parts[b][e])
y_in = y - 2
x_out = x + 5 * len(parts[b][e])
y_out = y + 10 * int(parts[b][e][0]) + 2
ver_coords[str_parts[b][e]] = [x_in, y_in, x_out, y_
```

Определение координат

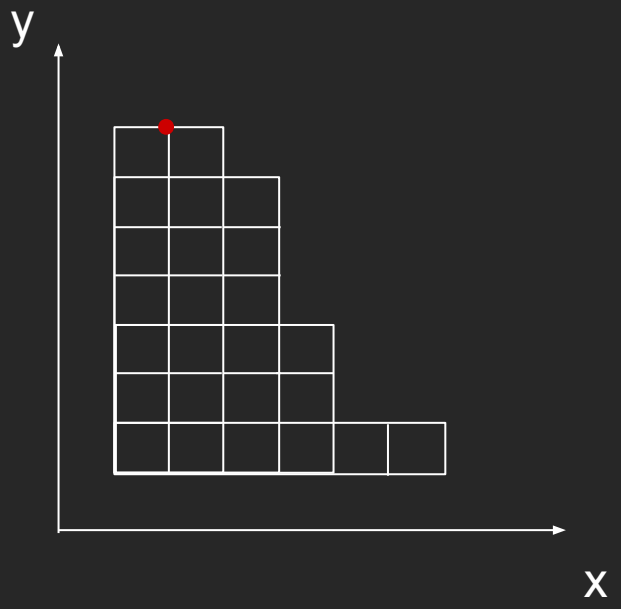


Координаты, с которых начинается отрисовка диаграммы - верхний левый угол.
Они рассчитываются исходя из предыдущей нарисованной диаграммы.



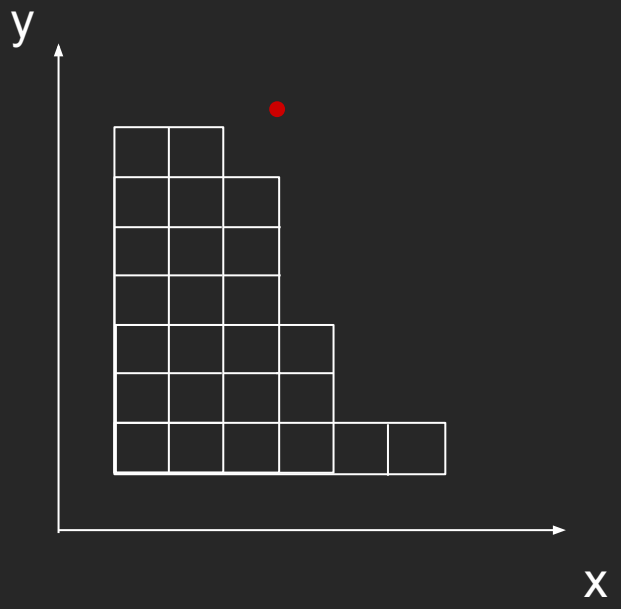
Определение координат

После построения первой клетки из координаты по Y вычитается высота клетки.
После чего строится следующая.
Так происходит до тех пор, пока количество клеток не станет нужным.



Определение координат

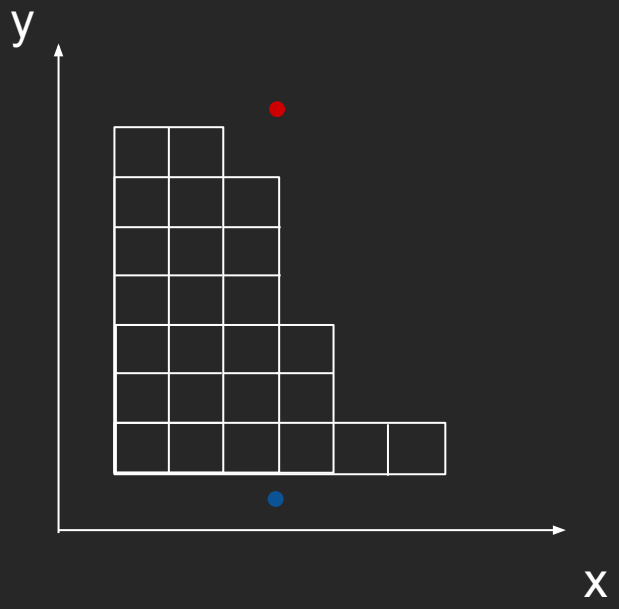
Как только построен первый столбец вычисляются координаты начала следующего.
Для этого по X прибавляется ширина клетки, а по Y количество клеток в столбце, умноженное на их высоту.



Определение координат

После построения диаграммы необходимо определить точки, в которые будут входить и выходить ребра.
Точка входа ребер (красная) располагается выше самой высокой точки диаграммы по оси Y и посередине диаграммы по оси X.

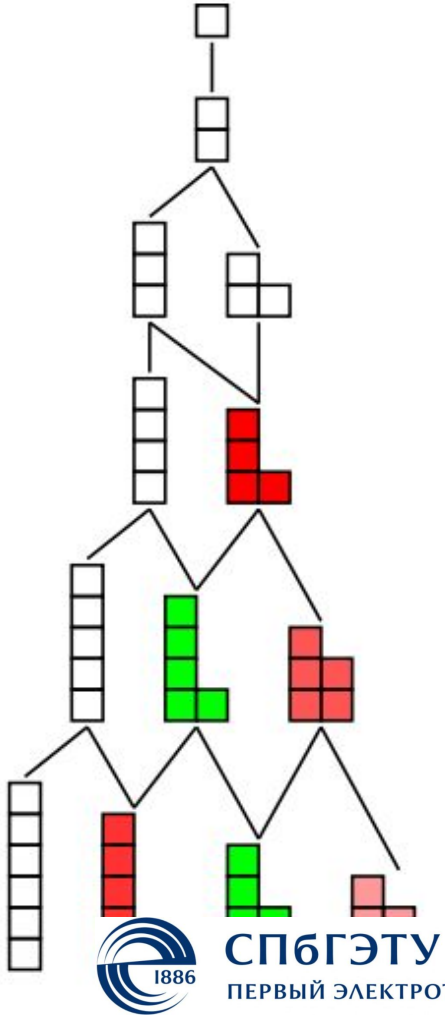
Определение координат



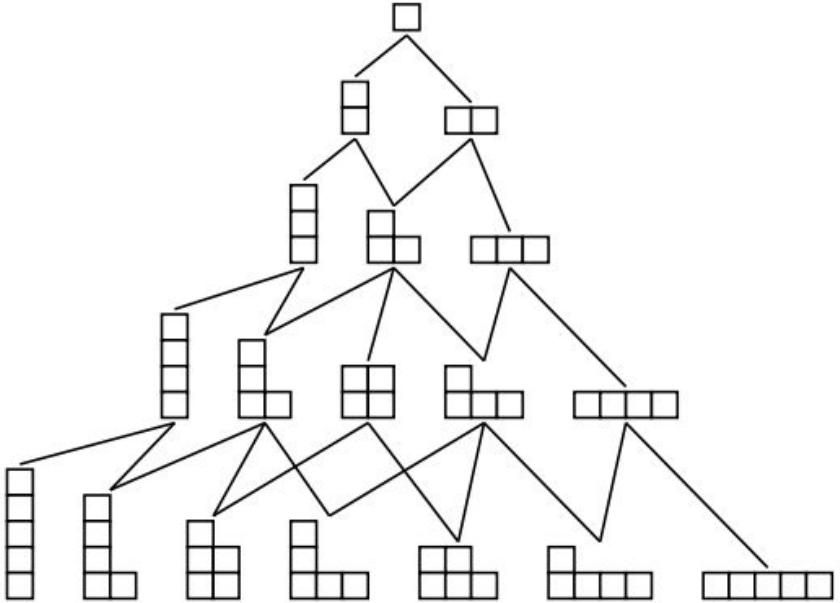
Точка выхода ребер (синяя) располагается ниже самой низко расположенной точки диаграммы по оси Y и посередине диаграммы по оси X.

Дополнительные возможности

Так как визуализатор необходим для исследований графа Юнга, во время которых удобно проводить с графом различные манипуляции, программа оснащена дополнительными возможностями.

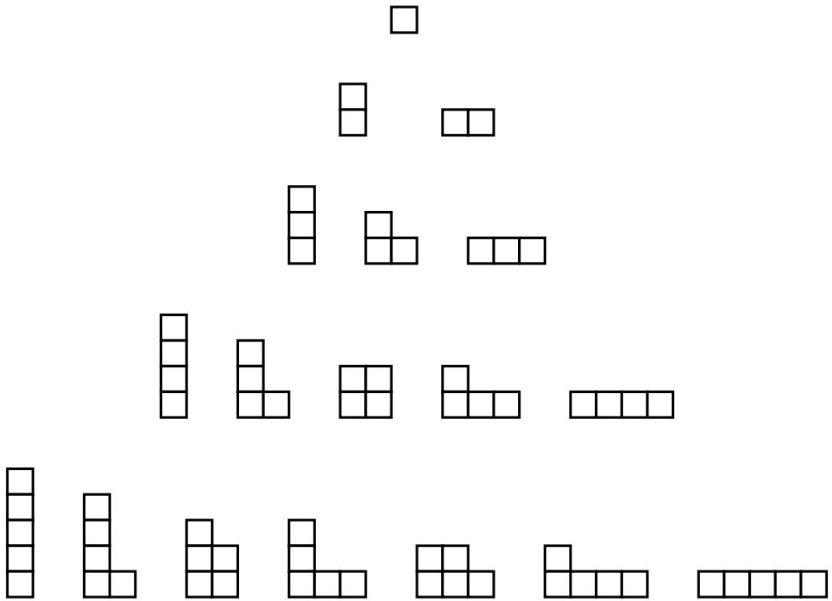


СПбГЭТУ «ЛЭТИ»
ПЕРВЫЙ ЭЛЕКТРОТЕХНИЧЕСКИЙ



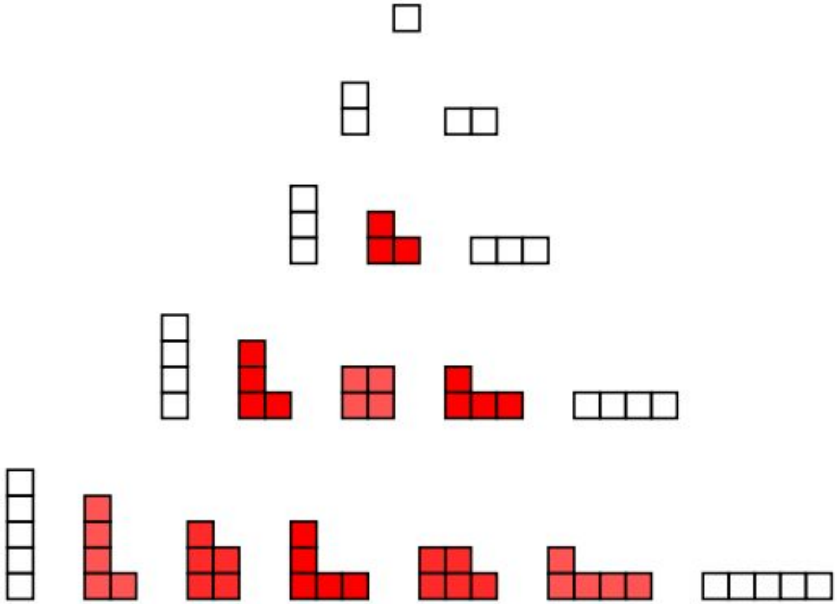
Удаление ребер

Удаление ребер позволяет упростить восприятие графа в тех случаях, когда нет необходимости отслеживать пути.



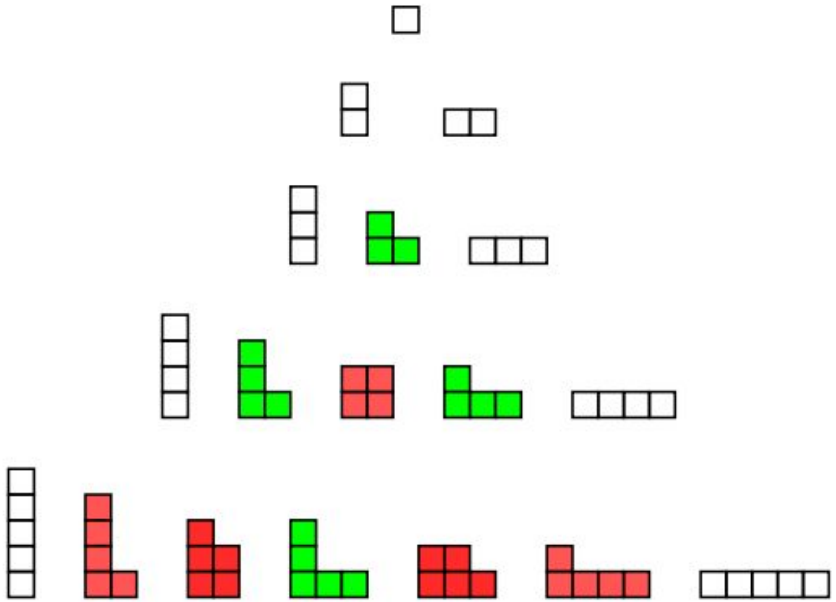
Удаление ребер

Удаление ребер позволяет упростить восприятие графа в тех случаях, когда нет необходимости отслеживать пути.



Подкраска

Подкраска интересующих диаграмм позволяет легче определять диаграммы, соответствующие определенным качествам.



Подкраска

Подкраска интересующих диаграмм позволяет легче определять диаграммы, соответствующие определенным качествам.

Так же есть возможность выбора цвета и добавления второго для выделения.

Размерность диаграмм Юнга

Размерность - количество путей, ведущих от корня графа к данной диаграмме

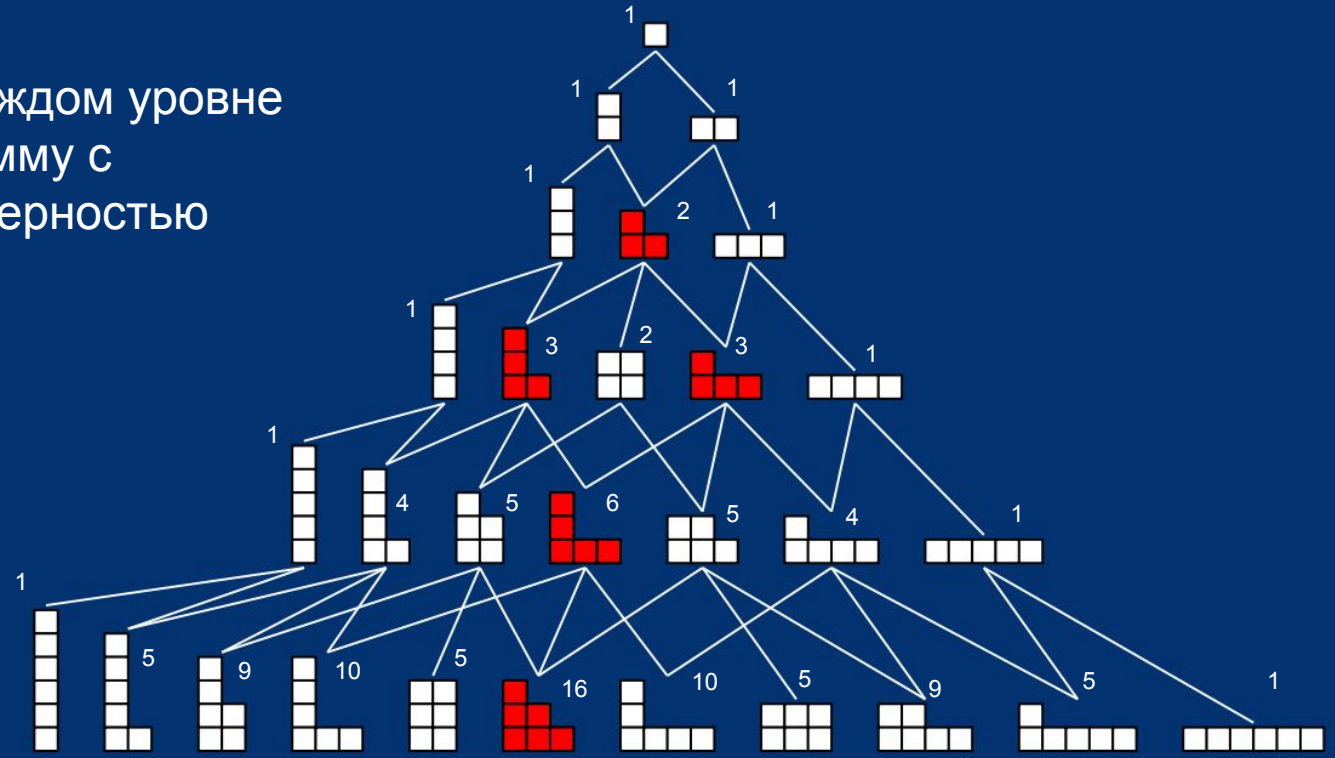
Размерность диаграммы λ(n), размера n (количество клеток в диаграмме) может быть вычислена по формуле крюков:

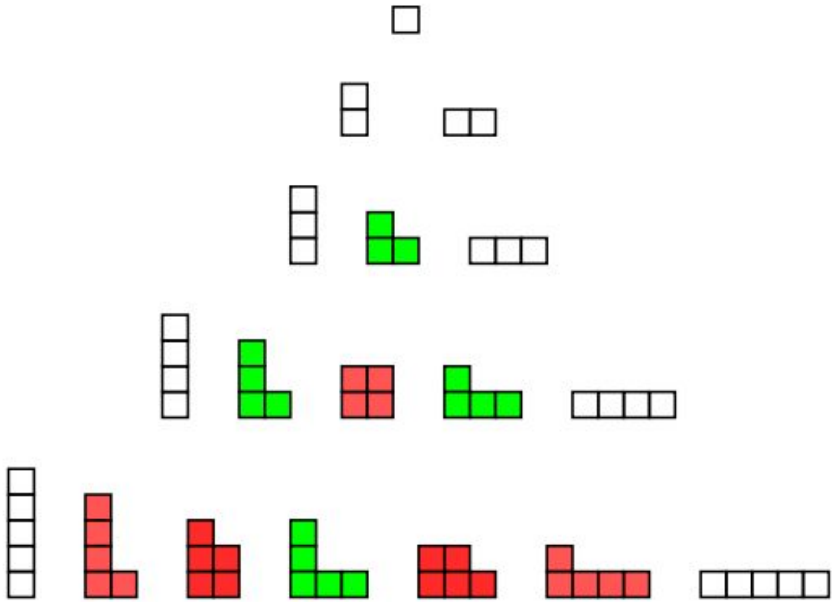
$$\dim(\lambda_n) = \frac{n!}{\prod_{(i,j\in\lambda)} h(i,j)}$$

где h(i, j) - длина крюка с вершиной в клетке (i, j), то есть количество клеток, расположенных выше в том же столбце и правее в той же строке, включая клетку (i, j).

Задача поиска максимальных размерностей

Задача: найти на каждом уровне графа Юнга диаграмму с максимальной размерностью





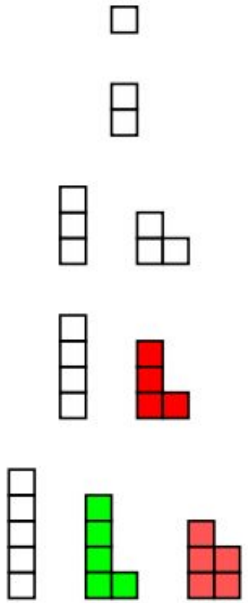
Подкраска

В данном случае цвет указывает на размерность диаграммы - те, которые имеют максимальную размерность на этаже, окрашены в зеленый. Остальные — в оттенки красного: чем ярче, тем больше размерность.

```
//A section for checking conditions for subgraphs. It is necessary to set the

            switch(subgraph_check)
            {
            case 1:
                if (!Shura(Child))
                {
                    delete Child;
                    accordance = false;
                }
                break;
            case 2:
                if (!SmallerTK(Child, K))
                {
                    delete Child;
                    accordance = false;
                }
                break;
            case 3:
                if (!MNrectangle(Child, n, m))
                {
                    delete Child;
                    accordance = false;
                }
                break;
            }

//-------------------------------------------------------------------------------
```

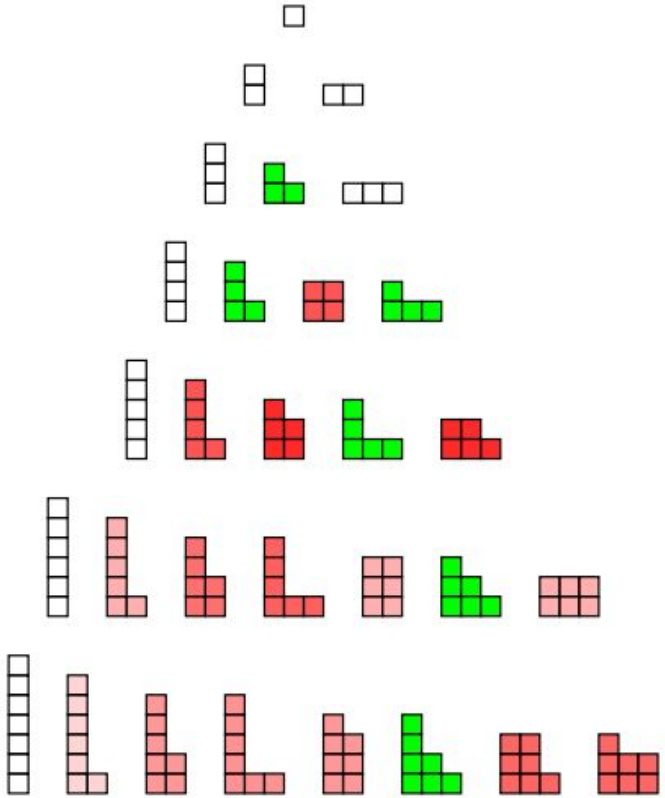
Подграфы

Визуализатор предоставляет возможность легко добавить функционал. Исследователь может быстро прописать функцию для выделения произвольного подграфа.



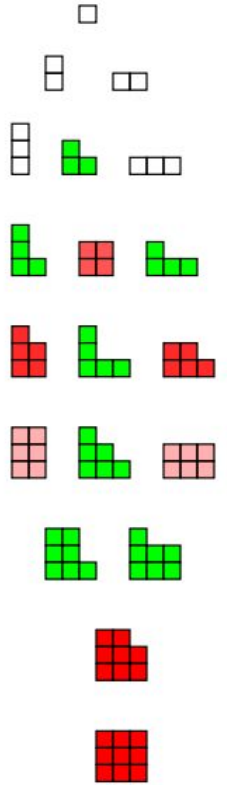
Граф Шура

Граф Шура является подграфом графа Юнга, содержащим только те диаграммы, все столбцы которых имеют разные высоты.
Визуализатор предусмотрен для подобных задач.



<=K подграф

Данный подграф включает в себя диаграммы Юнга, количество столбцов в которых не превышает некоторое К.
На рисунке пример для К = 3

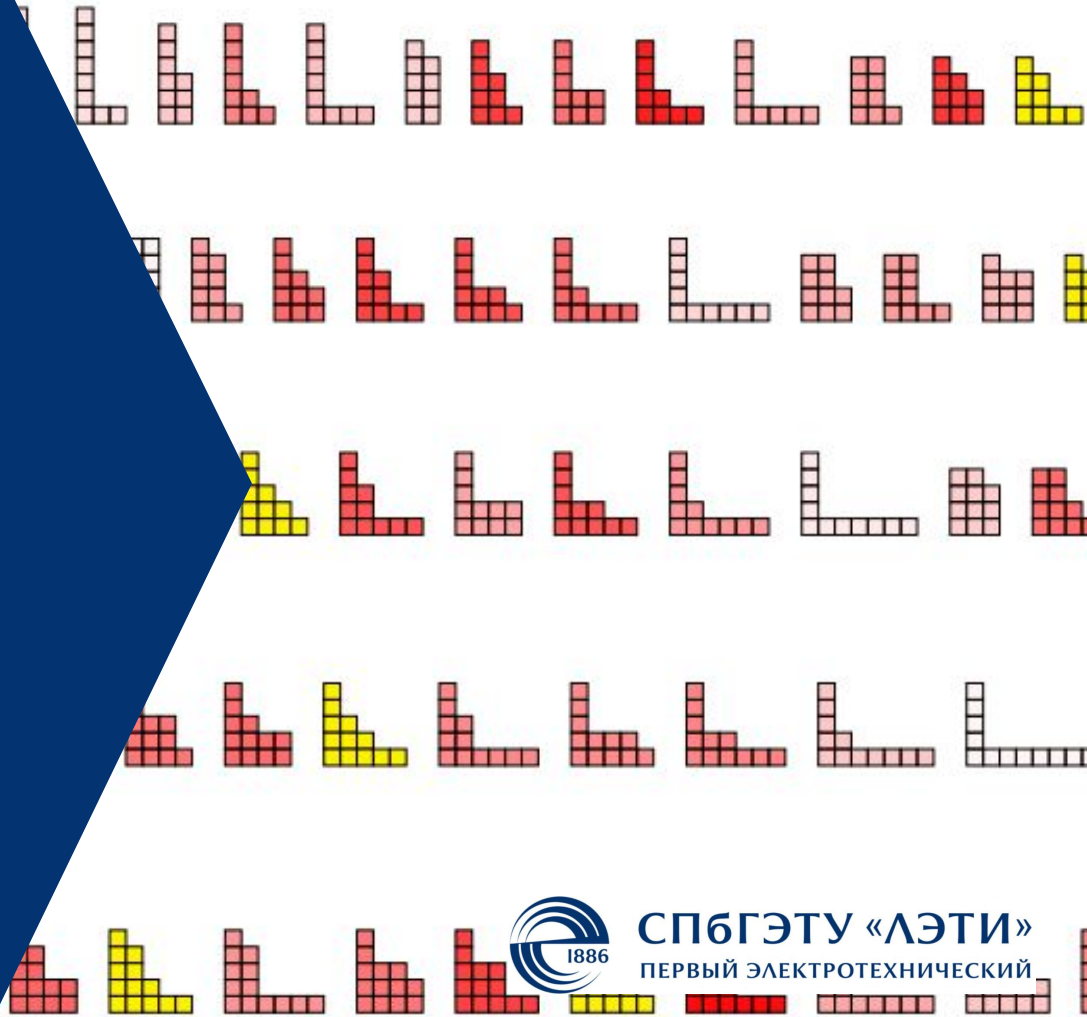


MN прямоугольник

Данный подграф включает в себя диаграммы Юнга, вписанные в прямоугольник со сторонами NxM.
На рисунке N = 3, M = 3.

Заключение

В результате разработан программный комплекс, выполняющий визуализацию начала графа Юнга.
Проект уже может быть использован исследователями и имеет неограниченные возможности для оптимизации и дальнейшего расширения.



СПбГЭТУ «ЛЭТИ»
ПЕРВЫЙ ЭЛЕКТРОТЕХНИЧЕСКИЙ