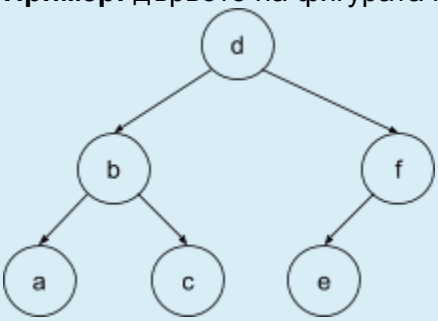
Задача 1. Даден е символен низ **s**. Казваме, че двоичното дърво със символи по върховете **t** представя **s** при следните условия:

- **s** е празният низ и **t** е празното дърво; или
- Ако **s** = $\mathbf{s}_0..\mathbf{s}_{k-1}$, а **m** = **⌊k / 2⌋** (долна цяла част), то коренът на **t** съдържа символа $\mathbf{s}_m$, лявото поддърво на **t**представя низа $\mathbf{s}_0..\mathbf{s}_{m-1}$, а дясното поддърво на **t** представя низа $\mathbf{s}_{m+1}..\mathbf{s}_{k-1}$.

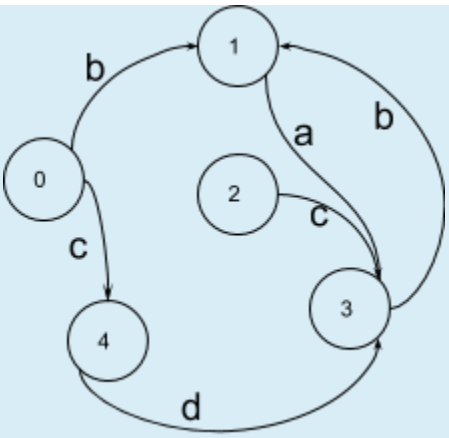
Нека е дадена следната структура, описваща възел в двоично дърво:
```
struct Node { char data; Node *left, *right; };
```
Да се реализира функция
```
[подходящ тип] buildTree ([подходящ тип] s)
```
която построява дърво, представящо низа **s**, и да връща указател към корена му. Дървото да бъде построено в динамичната памет (heap).

Пример: дървото на фигурата представя низа `"abcdef"`.



<u>Упътване:</u> Можете да ползвате метода `std::string::substr(pos, count)`, който връща подниз, започващ със символа с индекс `pos` и с `count` на брой елемента. Ако пропуснете параметъра `count`, ще получавите суфикс на дадения низ от позиция `pos` до края на низа.

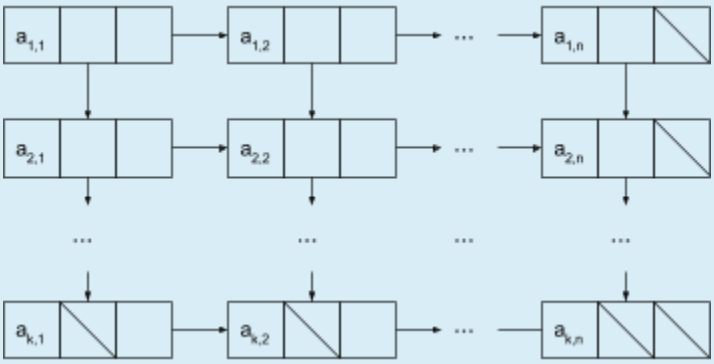
Задача 2. Нека е даден ориентиран граф < **V** = {0, .., **N** - 1}, **E** ⊆ **V** × **V** × {‘a’,..., ‘z’} > с малки латински букви като етикети по дъгите. Графът е представен с матрица на съседство:
```
char G[N][N];
```
където `G[i][j]` е етикет на дъгата от върха **i** до върха **j** в **G** или ‘\0’, ако дъга няма. Казваме, че думата **w** = $\mathbf{w}_1...\mathbf{w}_k \in$ {‘a’,..,’z’}* може да бъде прочетена между върховете **i** и **j** на графа, ако в него има път между **i** и **j** от последователни дъги с етикети $\mathbf{w}_1,..,\mathbf{w}_k$.
Да се реализира функция
```
std::string mostCommon (char G[100][100], int N, int k);
```
която намира най-често срещаната (или една измежду най-често срещаните) **k**-буквени думи, които могат да се прочетат между всички различни двойки върхове в **G**, или празният низ, ако в графа не може да се прочете нито една **k**-буквена дума.



Пример: При **k**=3, за графа на фигурата това е думата "bab", която може да се прочете по два различни начина, между двойките върхове (3,1) и (0,1). Пример за други думи, които се срещат по-рядко, са "cdb" и "aba".

*<u>Упътване:</u> Можете да ползвате `std::map m<std::string, int>`, с ключове - различните думи и стойности — броят на срещанията им. Методът `m.count(w)` ще ви върне колко пъти низът **w** се среща като ключ в m (нула или едно), а с помощта на `++m[w]`, можете да увеличите вече записания брояч за **w**.*

Задача 3. Матрица от цели числа се представя чрез свързана верига от кутии от вида `struct Cell { int data; Cell *right, *down; };`
Където указателят `right` сочи към следващата клетка в реда, а указателят `down` сочи към следващата клетка в колоната, както е показано на фигурата по-долу:



Да се реализира функция,
`[подходящ тип] copySubmatrix([подходящ тип] M, int n)`
която по дадена матрица **M** с поне **n** реда и поне **n** колони, зададена с указател към най-горна най-лява клетка, връща копие на такава нейна подматрица с размери **n** × **n**, в която сумата на елементите е най-голяма измежду всички такива подматрици. Върнатата подматрица да е нова структура, създадена в динамичната памет (heap).

Пример

```
вход:
1 2 3 4
5 2 7 4
8 7 3 1
3 6 5 4
n = 2
резултат:
8 7
3 6
```

Задача 4. Даден е вектор **v** от цели числа. Казваме, че двоичното дърво с числа по върховете **t** представя **v** при следните условия:

- **v** е празният вектор и **t** е празното дърво; или
- Ако **v** = $\mathbf{v}_0,..,\mathbf{v}_{k-1}$, а **m** = **⌊k / 2⌋** (долна цяла част), то коренът на **t** съдържа числото $\mathbf{v}_m$, лявото поддърво на **t**представя вектора $\mathbf{v}_0,..,\mathbf{v}_{m-1}$, а дясното поддърво на **t** представя вектора $\mathbf{v}_{m+1},..,\mathbf{v}_{k-1}$.
  Забележка: ако **k** = 1, то левия подвектор считаме за празен.

Нека е дадена следната структура, описваща възел в двоично дърво:

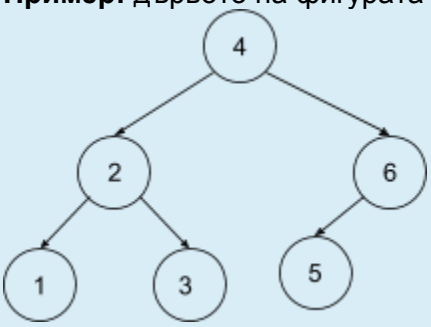
```
struct Node { int data; Node *left, *right; };
```

Да се реализира функция

```
[подходящ тип] buildTree ([подходящ тип] v);
```

която построява дърво, представящо вектора **v** и връща указател към корена му.

Пример: дървото на фигурата представя вектора 1, 2, 3, 4, 5, 6.



*<u>Упътване:</u> Ако **v** е вектор, то с помощта на следния конструктор*
`std::vector<int> L(v.begin(), v.begin() + count)`
*ще получите първите `count` елемента от **v**, а с*
`std::vector<int> R (v.begin() + start, v.end()),`
*ще получите суфикса на **v**, започващ от елемента с индекс `start`.*

Задача 5. Нека е даден ориентиран граф < **V** = {0, .., **N** - 1}, **E** ⊆ **V** × **V** × ℕ> с цели положителни числа като етикети по дъгите. Нека графът е представен с матрица на съседство:

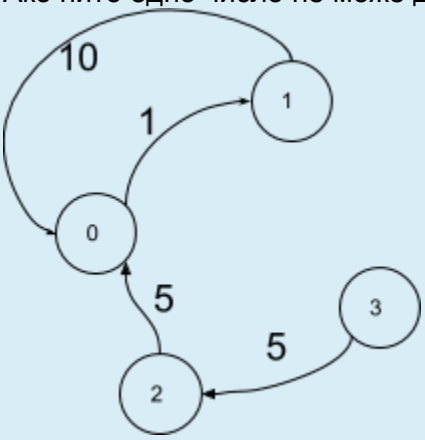
```
int G[N][N];
```

където `G[i][j]` е етикетът на дъгата от върха **i** до върха **j** в **G** или 0, ако дъга няма. Казваме, че числото **x** може да се образува между върховете **i** и **j** на графа, ако в него има път между **i** и **j** от последователни дъги с етикети $\mathbf{e}_1,..,\mathbf{e}_k$, такива че $\mathbf{x} = \mathbf{e}_1 + \ldots + \mathbf{e}_k$.

Да се дефинира функция

```
int mostCommon (int G[100][100], int N, int M);
```

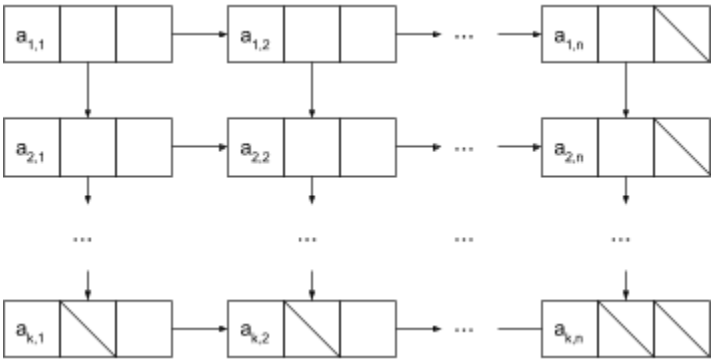
която намира най-често срещаното число (или едно измежду най-често срещаните числа) **x** ≤ **M**, които могат да се образуват между всички различни двойки върхове в **G**. Ако нито едно число не може да се образува в графа, функцията да връща 0.



Пример: При **M** = 20, за графа на фигурата това е числото 11 (= 5 + 5 + 1, = 10 + 1, = 1 + 10), което може да се образува по 3 начина. Пример за друго число, което се образува по-рядко, е 10 (= 5 + 5, = 10).

*Упътване: Можете да ползвате `std::map m<int, int>`, с ключове различните образувани числа и стойности — броят на срещанията им. Методът `m.count(x)` ще ви даде колко пъти **x** се среща като ключ в **m** (нула или едно), а с помощта на `++m[x]`, можете да увеличите вече записания брояч за **x**.*

Задача 6. Матрица от цели числа се представя чрез свързана верига от кутии от вида `struct Cell { int data; Cell *right, *down; };`
Където указателят `right` сочи към следващата клетка в реда, а указателят `down` сочи към следващата клетка в колоната, както е показано на фигурата по-долу:



Да се реализира функция,
`void deleteNegative([подходящ тип] M)`
която по дадена матрица M, зададена с указател към най-горна най-лява клетка, изтрива всички стълбове, в които има отрицателно число. Приемете, че в матрицата има поне един стълб, в който има само неотрицателни числа. Паметта за изтритите клетки да бъде освободена с оператора `delete`.

Пример:
вход:

```
1 -2 3  1
5  2 7 -4
8 -7 3  1
3  0 5  4
```

изход:

```
1 3

5 7

8 3

3 5
```

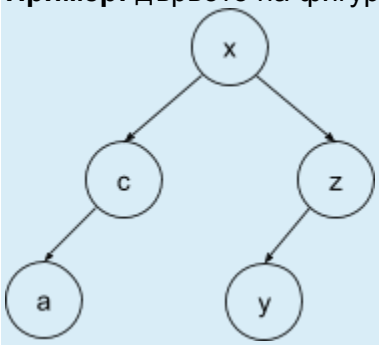
Задача 7. Даден е символен низ **s**. Казваме, че двоичното дърво със символи по върховете **t** представя **s** при следните условия:

- **s** е празният низ и **t** е празното дърво; или
- Ако **s** = $\mathbf{s}_0 \ldots \mathbf{s}_{k-1}$, а **m** = **⌊k / 2⌋** (долна цяла част), то:
  - коренът на **t** съдържа символа $\mathbf{s}_m$,
  - лявото поддърво на **t** представя низа **L**, състоящ се от всички букви на **s**, които са лексикографски по-малки от $\mathbf{s}_m$, в реда, в който са в **s**.
  - дясното поддърво на **t** представя низа **R**, състоящ се от всички букви на **s**, които са по-големи лексикографски от $\mathbf{s}_m$, в реда, в който са в **s**.

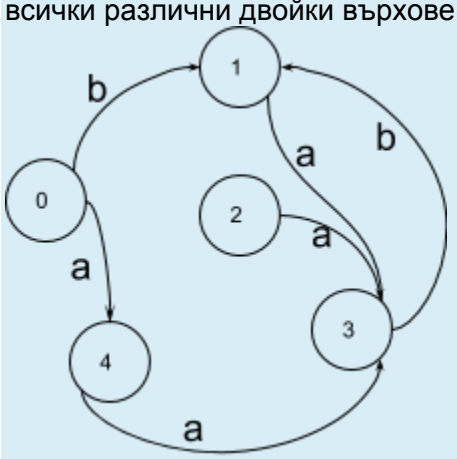
Пример: при **s** = "zazxxxcyc", то **L** = "acc", **R** = "zzy".

Ако е дадена следната структура, описваща възел в двоично дърво:
```
struct Node { char data; Node *left, *right; };
```
Да се реализира функция
```
[подходящ тип] buildTree ([подходящ тип] s);
```
която построява дърво, представящо низа **s** и да връща указател към корена му.

Пример: дървото на фигурата представя низа "zazxxxcyc".



Задача 8. Нека е даден ориентиран граф < **V** = {0, .., **N** - 1}, **E** ⊆ **V** × **V** × {'a',..., 'z'} > с малки латински букви като етикети по дъгите. Графът е представен с матрица на съседство:
```
char G[N][N];
```
където `G[i][j]` е етикет на дъгата от върха **i** до върха **j** в **G** или '\0', ако дъга няма. Казваме, че думата **w** = $\mathbf{w_1...w_k} \in$ {'a',..,'z'}* може да бъде прочетена между върховете **i** и **j** на графа, ако в него има път между **i** и **j** от последователни дъги с етикети $\mathbf{w_1,..,w_k}$.
Да се реализира функция
```
int countUnique (char G[100][100], int N, int k);
```
която намира броя на различните **k**-буквени думи, които могат да се прочетат между всички различни двойки върхове в **G**.



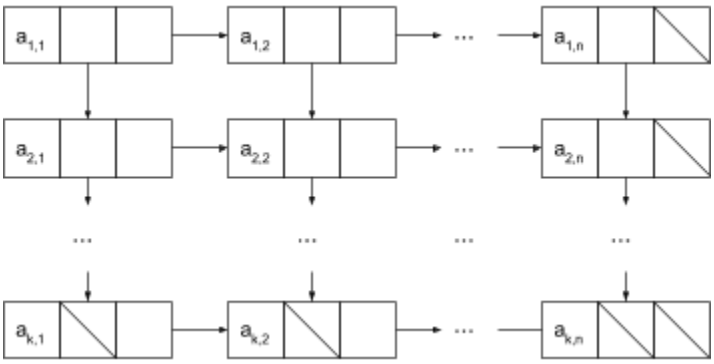
Пример: При **k** = 3, за графа на фигурата това са думите "aab", "aba" и "bab", чиито брой е 3.

<u>Упътване:</u> Можете да ползвате класа `std::set<std::string>`, представящ множество. Основните методи, които са необходими, са `insert` и `size`.

Задача 9. Матрица от цели числа се представя чрез свързана верига от кутии от вида

```
struct Cell { int data; Cell *right, *down; };
```

Където указателят `right` сочи към следващата клетка в реда, а указателят `down` сочи към следващата клетка в колоната, както е показано на фигурата по-долу:



Десен диагонал, наричаме последователност от клетки на матрицата започваща от първия ред на матрицата и завършваща в последния ред, в която всяка следваща клетка се намира диагонално долу и вдясно от предишната.
Да се реализира функция,
`void deleteDiagonal([подходящ тип] M)`
която по дадена матрица M с размерности **m** × **n** с 2 ≤ **m** ≤ **n**, зададена с указател към най-горна най-лява клетка, изтрива този десен диагонал, чиято сума на елементите е максимална. Паметта за изтритите клетки да бъде освободена с оператора `delete`.

Пример:

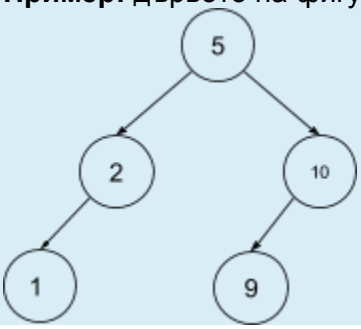
```
1 2 3 1 5        1 3 1 5
5 2 6 2 6   →    5 2 2 6
8 7 3 1 3        8 7 3 3
```

Задача 10. Даден е вектор **v** от цели числа. Казваме, че двоичното дърво с числа по върховете **t** представя **v** при следните условия:

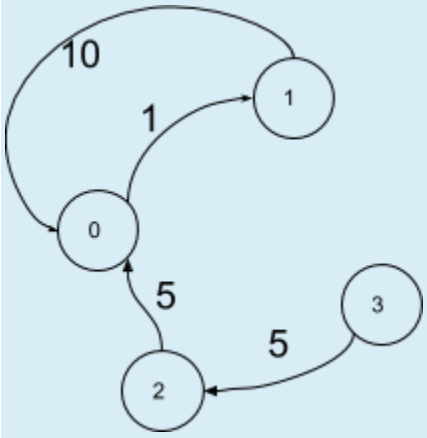
- **v** е празният вектор и **t** е празното дърво; или
- Ако **v** = $\mathbf{v}_0 \ldots \mathbf{v}_{k-1}$, а **m** = **⌊k / 2⌋** (долна цяла част), то:
  - коренът на **t** съдържа числото $\mathbf{v}_m$,
  - лявото поддърво на **t** представя вектора **L**, състоящ се от всички елементи на **v**, които са по-малки от $\mathbf{v}_m$, в реда, в който са във **v**.
  - дясното поддърво на **t** представя вектора **R**, състоящ се от всички елементи на **v**, които са по-големи от $\mathbf{v}_m$, в реда, в който са във **v**.
    Пример: При **v** = { 10, 1, 10, 5, 5, 5, 2, 9, 2 }, то **L** = { 1, 2, 2 }, **R** = { 10, 10, 9 }.

Нека е дадена следната структура, описваща възел в двоично дърво:
```
struct Node { int data; Node *left, *right; };
```
Да се реализира функция:
```
[подходящ тип] buildTree ([подходящ тип] v);
```
която построява дърво, представящо вектора **v**, и връща указател към корена му.

Пример: дървото на фигурата представя вектора {10, 1, 10, 5, 5, 5, 2, 9, 2}.



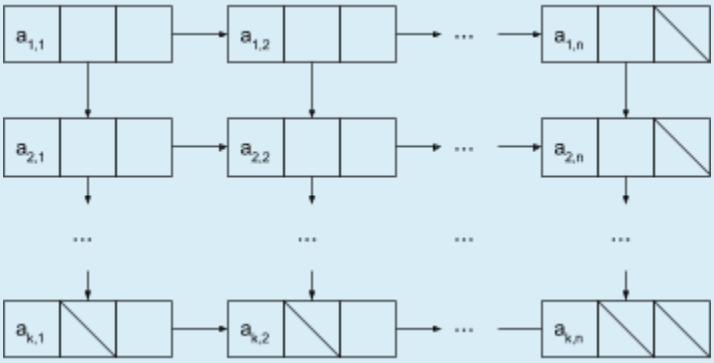
Задача 11. Нека е даден ориентиран граф < **V** = {0, .., **N** - 1}, **E** ⊆ **V** × **V** × ℕ> с цели положителни числа като етикети по дъгите. Нека графът е представен с матрица на съседство:
```
int G[N][N];
```
където `G[i][j]` е етикетът на дъгата от върха **i** до върха **j** в **G** или 0, ако дъга няма. Казваме, че числото x може да се образува между върховете **i** и **j** на графа, ако в него има път между **i** и **j** от последователни дъги с етикети $\mathbf{e}_1,..,\mathbf{e}_k$, такива че **x** = $\mathbf{e}_1$ + … + $\mathbf{e}_k$.
Да се дефинира функция
```
int countUnique (int G[100][100], int N, int M);
```
която намира броя на различните числа **x** ≤ **M**, които могат да се образуват между всички различни двойки върхове в **G**.



Пример: При **M** = 15, за графа на фигурата, това са числата 1, 5, 6, 10, 11 и 12, чиито брой е 6.

Упътване: Можете да ползвате класа `std::set<int>`, преставящ множество. Основните методи, които са необходими, са: `insert` и `size`.

Задача 12. Матрица от цели числа се представя чрез свързана верига от кутии от вида `struct Cell { int data; Cell *right, *down; };`
Където указателят `right` сочи към следващата клетка в реда, а указателят `down` сочи към следващата клетка в колоната, както е показано на фигурата по-долу:



Да се реализира функция,
`[подходящ тип] buildAdjungate([подходящ тип] M)`
която по дадена матрица M с поне два реда и поне две колони, зададена с указател към най-горна най-лява клетка, конструира нова матрица, която се получава от дадената чрез пропускане на реда и стълба, съответстващи на максималния елемент в матрицата. При няколко максимални елемента, се избира първия отгоре надолу и отляво надясно. Върнатата матрица да е нова структура, създадена в динамичната памет (heap).

Пример:

```
1 2 3 1 5          1 2 3 5
5 2 6 9 6   →      8 7 9 9
8 7 9 1 9
```