Как пользоваться библиотекой

ПРИМЕЧАНИЕ: Вам не обязательно писать программу с нуля. Можно взять за основу приложенный пример (про сборку примера см. ниже).

Функция parallelTree

Библиотека ParallelTree предоставляет функцию `parallelTree`, которая обходит дерево вариантов и возвращает наилучший рекорд.

```
std::unique_ptr<Record> parallelTree(
    std::unique_ptr<Node> root,
    const Record& initialRecord,
    uint32_t threads=0
);
```

где:

- `root` - указатель на корень дерева вариантов;
- `initialRecord` - начальное значение рекорда;
- `threads` - число потоков (если `threads=0`, то используется число потоков, равное количеству ядер).

Классы `Node` и `Record` абстрактные. Чтобы воспользоваться функцией `parallelTree`, необходимо унаследоваться от них и переопределить методы.

Класс Record

Класс `Record` должен хранить текущее значение рекорда и реализовывать 2 метода:

- `bool betterThan(const Record& other) const` - возвращает `true`, если текущее значение рекорда лучше, чем `other`. Для задач минимизации "лучше" означает меньше, а для задач максимизации - больше.
- `std::unique_ptr<Record> clone() const` - возвращает копию текущего рекорда, обернутую в `std::unique_ptr`

```
class ExampleRecord : public Record
{
public:
    // Значение рекорда
    uint32_t value;

    bool betterThan(const Record& other) const override
    {
        // Преобразуем Record& в ExampleRecord&
        const ExampleRecord& otherCast = static_cast<const ExampleRecord&>(other);
        // Сравниваем значение. Здесь подразумевается задача максимизации
```

```
        return value > otherCast.value;
    }

    std::unique_ptr<Record> clone() const override
    {
        return std::make_unique<ExampleRecord>(*this);
    }
};
```

Класс Node

Класс `Node` должен хранить узел дерева вариантов и реализовывать 2 метода:

- `std::vector< std::unique_ptr<Node> > process(Record& record)` - обрабатывает текущий узел дерева вариантов и возвращает вектор его потомков. Принимает ссылку на текущий рекорд. `record` можно изменять без синхронизации (без мютексов, без атомарных операций, и т.п.).
- `bool hasHigherPriority(const Node& other) const` - возвращает `true`, если приоритет текущего узла больше, чем у other. Узлы с большим приоритетом обрабатываются раньше. Например, для обхода в глубину приоритет у узлов на большем уровне дерева должен быть больше, а для обхода в ширину - меньше.

```
class ExampleNode : public Node
{
public:
    std::vector< std::unique_ptr<Node> > process(Record& record) override
    {
        ExampleRecord& recordCast = static_cast<ExampleRecord&>(record);
        // ...
    }

    bool hasHigherPriority(const Node& other) const override
    {
        const ExampleNode& otherCast = static_cast<const ExampleNode&>(other);
        // ...
    }
};
```

Подробнее см. в примере.

Сброка примера

Visual Studio

Откройте Visual_Studio/ParallelTreeExample/ParallelTreeExample.sln и запустите пример. Собранная библиотека уже вложена в решение и решение уже настроено так, чтобы ее использовать.

GCC/MinGW

В папке gcc_mingw находится проект cmake с тем же примером, а также туда вложена собранная библиотека для Windows и Linux. Есть 2 варианта:

1. Использовать среду разработки, которая поддерживает проект cmake (Atom, QtCreator и т.д.)
2. Собрать вручную:

```
# Создаем папку для сборки
mkdir build
# Заходим в папку
cd build
# Генерируем makefile
cmake ..
# Собираем пример
cmake --build .
# Запускаем пример
./parallel_tree_example
```