Задача №1

Стандартная библиотека содержит std::bitset и специализацию шаблона std::vector<bool> для эффективного хранения и обработки большого объема данных для битов и bool. Наша задача реализовать контейнер для компактного хранения значения множества тритов. Трит аналог бита в [Троичной системе счисления](https://ru.wikipedia.org/wiki/%D0%A2%D1%80%D0%BE%D0%B8%D1%87%D0%BD%D0%B0%D1%8F_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0_%D1%81%D1%87%D0%B8%D1%81%D0%BB%D0%B5%D0%BD%D0%B8%D1%8F). Для симметричной троичной системы его можно определить как {False, Unknown, True} или {-1, 0, 1} (Логики Клини и Приста соответственно). Таблицы истинности для операций NOT, AND, OR даны на странице: [Троичная логика](https://ru.wikipedia.org/wiki/%D0%A2%D1%80%D0%BE%D0%B8%D1%87%D0%BD%D0%B0%D1%8F_%D0%BB%D0%BE%D0%B3%D0%B8%D0%BA%D0%B0)

(F: FALSE, U: UNKNOWN, T: TRUE)

|  |  |
| --- | --- |
| ***A*** | **¬ *A*** |
| **F** | T |
| **U** | U |
| **T** | F |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ***A* ∧ *B*** | | ***B*** | | |
| **F** | **U** | **T** |
| ***A*** | **F** | F | F | F |
| **U** | F | U | U |
| **T** | F | U | T |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ***A* ∨ *B*** | | ***B*** | | |
| **F** | **U** | **T** |
| ***A*** | **F** | F | U | T |
| **U** | U | U | T |
| **T** | T | T | T |

Для хранения одного трита достаточно 2 битов. Поэтому контейнеры из std неэффективно расходуют память для хранения тритов. Наш контейнер должен реализовать динамическое управление массивом типа uint[] для хранения тритов. Код должен учитывать что uint может быть разным на разных платформах. При обращении к неустановленным тритам чтение должно возвращать значение Unknown, а запись Unknown не приводить к выделению памяти для хранения данных. То есть:

*enum Trit{False, Unknown, True};*

……….

*//резерв памяти для хранения 1000 тритов*

*TritSet set(1000);*

*// length of internal array*

*size\_t allocLength = set.capacity();*

*assert(allocLength >= 1000\*2 / 8 / sizeof(uint) );*

*// 1000\*2 - min bits count*

*// 1000\*2 / 8 - min bytes count*

*// 1000\*2 / 8 / sizeof(uint) - min uint[] size*

*//не выделяет никакой памяти*

*set[1000’000’000] = Unknown;*

*assert(allocLength == set.capacity());*

*// false, but no exception or memory allocation*

*if(set[2000’000’0`00]==True){}*

*assert(allocLength == set.capacity());*

*//выделение памяти*

*set[1000’000’000] = True;*

*assert(allocLength < set.capacity());*

*//no memory operations*

*allocLength = set.capacity();*

*set[1000’000’000] = Unknown;*

*set[1000’000] = False;*

*assert(allocLength == set.capacity());*

*//освобождение памяти до начального значения или*

*//до значения необходимого для хранения последнего установленного трита*

*//в данном случае для трита 1000’000*

*set.shrink();*

*assert(allocLength > set.capacity());*

Реализовать перегрузку операций AND, OR, NOT с расширением результата для хранения необходимых данных. То есть

*TritSet setA(1000);*

*TritSet setB(2000);*

*TritSet setC = setA & setB;*

*assert(setC.capacity() == setB.capacity());*

Дополнительно реализовать методы:

*//число установленных в данное значение тритов*

*//для трита Unknown - число значений Unknown до последнего установленного трита*

*size\_t cardinality(Trit value);*

*//аналогично но сразу для всех типов тритов*

*std::unordered\_map< Trit, int, std::hash<int> > cardinality();*

*// забыть содержимое от lastIndex и дальше*

*void trim(size\_t lastIndex);*

*// logical length - индекс последнего не Unknown трита +1*

*size\_t length();*

Для проверки корректности работы необходимо покрыть unit test-ами все публичные методы и операторы.

В качестве библиотеки для тестирования использовать Google Test Framework (<https://ru.wikipedia.org/wiki/Google_C%2B%2B_Testing_Framework>)

<http://www.ibm.com/developerworks/aix/library/au-googletestingframework.html>