Теоретическое решение задачи B.

Задачу будем решать с помощью системы непересекающихся множеств. Номер сундука будем считать определенным элементом множества. Изначально у нас каждый сундук содержится в своем множестве, а затем мы будем их объединять, следуя условию задачи (каждая запись Врунгеля будет соответствовать операции объединения множеств).

Хранить элементы СНМ будем в виде структуры, которая содержит указатель на представителя множества и ранг (для выполнения эвристик), а также разницу в монетах между этим элементом и элементом в множестве с минимальным количеством монет, разницу в монетах между этим элементом и представителем множества, а также флаг, который будет указывать на то, находится ли первый сундук (с нулем монет) в этом конкретном множестве или нет.

Как и в стандартной реализации СНМ у нас будет 3 основных функции: $make\ set$, $find\ set$ и $union\ sets$. Функция $make\ set$ ничем не отличается от стандартной реализации и просто должна создавать новый объект множества и инициализировать его поля. Функция $find\ set$ также почти полностью совпадает с ее стандартной реализацией, за исключением лишь того, что нам на каждом шаге приходится дополнительно поддерживать минимальное расстояние до представителя множества и обновлять его. Последняя функция $union\ sets$ отличается от стандартной реализации достаточно сильно. Опишем подробнее ее.

Пусть $parent$ – представитель $k-$ го множества, в котором содержится элемент $i,\ minDif$ – разница в монетах между $i-$ м элементом $k-$ го множества и элементом этого множества с минимальным количеством монет, а $difParent$ – разница в монетах между $i-$ м элементом и представителем $k-$ го множества. Тогда, если в какой-то момент, при выполнении операции $union\ sets$ получается, что для какого-то элемента $difParent - parent.minDif < 0$, то это означает, что найти минимальное количество монет в сундуках невозможно. Также в этой функции обязательно нужно поддерживать дополнительно хранимые данные в элементе множества (описаны выше). Также стоит заметить, что в случае, если у нас есть запись $M = (i,\ j, d)$, в которой i и j – два определенных сундука, а d – разница в монетах между i и j сундуками и при этом $d < 0$, то мы можем полностью законно поменять знак у d и при этом не забыть сделать $swap(i,\ j)$ (преобразования эквивалентны).

Теперь создадим массив $nodes$ из элементов множества (сделаем это в цикле с помощью функции $make\ set$). А далее будем поочередно считывать новые записи и объединять множества в соответствии с описанными выше правилами. Если в какой-то момент получается, что функция $union\ sets$ вернет $false$ – найти минимальное количество монет, соответствующих записям нельзя, иначе – ответом для $i-$ го элемента массива будет $minDif - nodes[i].difParent.$

Доказательство корректности.

Допустим у нас есть множество A с n элементами, представителем a_{rep}, минимальным элементом a_{min} и другими элементами $a_1,\ a_2,\ \dots,\ a_{n-2}$, а также множество B с m элементами, представителем b_{rep}, минимальным элементом b_{min} и другими элементами $b_1,\ b_2,\ \dots,\ b_{m-2}$. Для каждого элемента хранить также будем $rank -$ верхнюю границу глубины дерева T, в котором содержится этот элемент.

Есть запрос на объединение множеств вида $union(a_i\ \ b_j\ \ d)$, где d – разница между a_i и b_j. Придерживаясь ранговой эвристики, присоединять будем всегда дерево с меньшим рангом к дереву с большим (пусть в данном случае $a_i\text{−>}\ \mathrm{rank}\ < b_i\text{−>}\ \mathrm{rank}$, тогда присоединять будем A к B). При объединении деревьев мы не теряем никаких решений и не получаем новых, поскольку, если б это было не так, то тогда б это означало, что в новом (слитом) дереве $C\ \ c_{min}\ != \min(a_{min},\ b_{min})$, а это означало б, что в множествах A или B есть элемент меньший указанного минимума, что противоречит изначальному утверждению про минимальные элементы в множествах A и B.

Тогда, можно сделать вывод, что мы можем объединять множества и пересчитывать требуемые расстояния без потери решений. Однако, есть одно исключение – случай, когда решений вообще нет. Допустим, у нас есть все те же множества A и B с указанными характеристиками. Тогда имеем два случая, в которых у нас нет решений:

1. Если $a_{rep} = b_{rep}$ и при этом расстояния от a_i до a_{rep} и $b_j + d$ до b_{rep} не совпадают, ведь условие $a_{rep} = b_{rep}$ является признаком того, что элементы a_i и b_j на самом деле находятся в одном множестве, а в случае, если такие расстояния разные получаем противоречие тому, что данные элементы находятся в одном множестве. Соответственно в этом варианте решений нет.

2. Если в одном из сливаемых деревьев есть нулевой элемент Z_0 (самый первый пустой сундук) и расстояние от него до c_{rep}, где c_{rep} – представитель нового дерева, полученного в результате сливания A и B, будет не минимальным. Докажем это. Допустим, у нас есть другой c_i элемент с расстоянием до c_{rep} меньше, чем от Z_0. Тогда, это означает, что монет в c_i сундуке будет меньше, чем монет в Z_0, а это противоречит условию и здравому смыслу, ведь в Z_0 сундуке у нас 0 монет, а в c_i сундуке тогда должно быть меньше нуля монет, но этого не может быть, поскольку в сундуке не может быть отрицательного количества монет. Соответственно имеем, что расстояние между c_{rep} и Z_0 всегда должно быть минимальным – в другом случае решения нет (что и требовалось доказать).

Оценим время работы описанного алгоритма.

1. Функция $make\,set$ будет работать за $O(1)$, поскольку выполняем только константные операции.
2. Функция $find\ set$ будет работать за $O(1)$, поскольку выполняются обе эвристики, а соответственно на один запрос получается $O(\alpha(n))$ в среднем, где $\alpha(n)$ — обратная функция Аккермана, которая растет настолько медленно, что для всех разумных ограничений n она не превосходит 4. Соответственно сложность данной операции можно считать константной.
3. Функция $union\,sets$ аналогично предыдущей будет работать за $O(1)$, поскольку также в ней выполняются все эвристики.

Таким образом, имеем N вызовов функции $make\ set$ (для занесения каждого сундука в свое множество) и M вызовов функции $union\ sets$ (количество записей Врунгеля). Тогда весь алгоритм отработает за $O(N + M)$ по времени.

Оценим требуемую память данного алгоритма. Требуется $O(N)$ для хранения исходного массива с элементами множеств (сундуков).