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 $minDif - nodes[i].difParent$.

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).