Ведерников Николай Викторович,
Замятин Евгений Игоревич,
Шовкопляс Григорий Филиппович,
Ульянцев Владимир Игоревич

ЗАДАЧА «ШТУРМ»

Этой статьей мы продолжаем цикл публикаций олимпиадных задач по информатике для школьников. Решение таких задач и изучение разборов поможет Вам повысить уровень практических навыков программирования и подготовиться к олимпиадам по информатике.

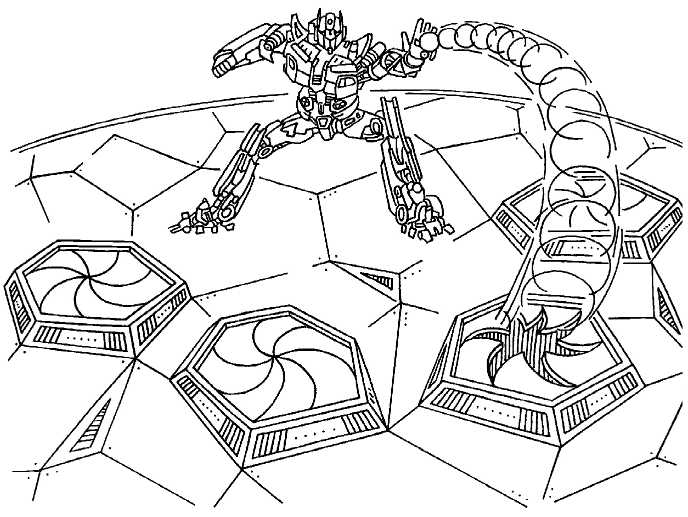
В этой статье рассматривается задача «Штурм», которая предлагалась на Пятой личной интернет-олимпиаде по программированию в 2013–2014 учебном году. Материалы этой олимпиады можно найти на сайте http://neerc.ifmo.ru/school/io/.

УСЛОВИЕ ЗАДАЧИ

Автоботы отважились на штурм базы десептиконов. Из-за эффекта внезапности множество десептиконов полегло на месте, а остальные начали судорожно прятаться в бункеры, которые начали закрываться. Оптимус Прайм, возглавлявший штурм, хочет добить как можно больше десептиконов. Для этого он решил использовать новую разработку автоботов – бомбы «Антибункер».

Принцип работы бомбы заключается в том, что ее можно кинуть внутрь бункера, а когда тот закроется – взорвать, и тогда она убьет всех, кто был внутри, и не покалечит союзников. Единственное ее неудобство заключается в том, что она приводится в действие вручную. То есть Оптимусу, после того как он кинет бомбу, придется ждать закрытия бункера, чтобы взорвать всех внутри, и лишь потом он сможет двигаться дальше.

Оптимус может проехать мимо каждого бункера ровно один раз, посетив их в порядке возрастания номеров. К счастью, он знает, на какой минуте закроется каждый из бункеров, а также количество десептиконов



в каждом из бункеров. Между бункерами Оптимус перемещается очень быстро – перемещение между любой парой бункеров занимает у него ровно одну минуту. Однако, если он проехал бункер, он уже не сможет вернуться обратно.

Оптимус Прайм просит вас помочь ему рассчитать, какое наибольшее количество врагов он сможет убить и в какие бункеры нужно бросить бомбы для этого.

Формат входного файла

В первой строке дано число n ($1 \le n \le 10^5$) – количество бункеров. Далее идут две строки по n целых чисел. В первой строке содержатся числа a_i ($1 \le a_i \le 10^9$) – количество десептиконов в бункере i. Во второй строке содержатся числа t_i ($1 \le t_i \le 10^5$) – время закрытия бункера i в минутах.

Формат выходного файла

В первой строке выходного файла выведите одно число – наибольшее возможное количество врагов, которые будут повержены Оптимусом. Во второй строке выведите количество бункеров, которое он должен взорвать. В третьей строке выведите через пробел номера бункеров, которые будут взорваны. Номера должны следовать в порядке возрастания. Заметим, что момент закрытия каждого взорванного бункера должен быть больше, чем момент закрытия бункера, взорванного перед ним.

Примеры входных и выходных данных

assault.in	assault.out
3 1 2 1 1 2 3	4 3 1 2 3
5 4 1 5 9 3 9 7 9 8 8	10 2 2 4

РАЗБОР ЗАДАЧИ

Чтобы решить данную задачу, нужно выбрать такую подпоследовательность бункеров с возрастающими номерами, чтобы время закрытия каждого следующего было больше, чем время закрытия предыдущего, а кроме того, сумма десептиконов по всем выбранным бункерам была максимальна.

Для решения этой задачи будем использовать *метод динамического программирования* [1]. Суть этого метода – разбиение сложной задачи на более простые подзадачи. Для решения задач этим методом необходимо определить, как и в методе математической индукции, три вещи: *состояние*, *переход* и *базу*.

Для каждого бункера с номером k посчитаем, какое наибольшее число десептиконов d_k можно убить, если закончить штурм на данном бункере. Это и будет состоянием алгоритма динамического программирования. База алгоритма тривиальна: если мы не уничтожили ни один бункер, то мы не убили ни одного врага, и ответ будет равен нулю.

Теперь определим переход динамического программирования. Пусть мы сейчас рассматриваем k-й бункер и посчитали состояния для всех бункеров, у которых номер меньше k. Тогда $d_k = \max(d_i + a_k)$ для всех $i < k$, таких что $t_i < t_k$. После вычисления всех d_k ответ на задачу будет равен максимуму из состояний по всем бункерам.

Если вычислять d_k при помощи линейного поиска по всем бункерам d_i, то время работы программы будет составлять $O(n^2)$, что при ограничениях, заданных в условии, не уложится в ограничение по времени работы программы. Для того чтобы решение укладывалось по времени, ускорим поиск указанного максимума по предыдущим значениям d_i. Для этого воспользуемся деревом Фенвика [2].

В настоящей статье мы не станем описывать принцип работы используемой структуры. В листинге 1 приведены реализации на языке *Pascal* функций поиска максимума `(get)` и обновления `(sets)` для дерева Фенвика.

Построим дерево Фенвика для моментов времени: для минуты t мы будем хранить максимальное число десептиконов f_t (`fenv[i]` в листинге 1), которое мы сможем убить за t минут. Изначально проинициализируем значения f_t нулями и рассмотрим бункеры в порядке возрастания номеров. Во время рассмотрения очередного бункера i сделаем запрос `get(t[i])` в дерево Фен-

Листинг 1. Реализация функций для дерева Фенвика

```
function get(i: longint): int64;
var
  res: int64;
begin
  res := -1;
  while (i >= 1) do begin
    res := max(res, fenv[i]);
    i := i and (i - 1);
  end;
  get := res;
end;
procedure sets(i: longint; v: int64);
begin
  while (i <= MAXTIME) do begin
    fenv[i] := max(fenv[i], v);
    i := 2 * i - (i and (i - 1));
  end;
end;
```

вика на текущий максимум взорванных врагов до времени закрытия текущего бункера. После этого обновим ячейку `t[i] + 1` дерева – перемещение между бункерами по условию занимает минуту. В листинге 2 приведен код для вычисления d_k.

Теперь восстановим последовательность взорванных бункеров. Для этого воспользуемся вычисленными значениями d_k. Рассмотрим значения с конца, а как только встретим бункер, который мог помочь получить максимальный ответ, добавим его в искомую подпоследовательность. В листинге 3 показано, как восстановить искомую подпоследовательность бункеров.

Листинг 2. Вычисление ответа при помощи дерева Фенвика

```
  maxkill := -1;
  for i := 1 to n do begin
    res := get(t[i]);
    dp[i] := res + a[i];
    sets(t[i] + 1, dp[i]);
    maxkill := max(maxkill, dp[i]);
  end;
```

Листинг 3. Восстановление подпоследовательности

```
prev := MAXTIME + 1;
si := 0;
for i := n downto 1 do begin
     if (maxkill <> l[i]) or (t[i] > prev) then
          continue;
     inc(si);
     ans[si] := i;
     dec(maxkill, a[i]);
     prev := t[i];
end;
```

Константа MAXTIME соответствует максимальному моменту времени, по условию задачи MAXTIME = 10^5. Заменив линейный пробег по всем предыдущим значениям d_i на запрос к дереву Фенвика, мы произвели необходимое улучшение время решения с $O(n^2)$ до $O(n \log(\texttt{MAXTIME}))$. Изучение асимптотики времени работы функций дерева Фенвика оставим читателю в качестве дополнительного упражнения.

Литература

1. *Кормен Т., Лейзерсон Ч., Ривест Р., Штайн К.* Алгоритмы: построение и анализ. 2-е изд. М.: «Вильямс». 2006.

2. *Fenwick P.* A new data structure for cumulative frequency tables / Software: Practice and Experience. 24 (3): 327–336. 1994.

Ведерников Николай Викторович,
студент четвертого курса кафедры «Компьютерные технологии» НИУ ИТМО, член жюри Интернет-олимпиад по информатике,

Замятин Евгений Игоревич,
студент первого курса кафедры «Компьютерные технологии» НИУ ИТМО, член жюри Интернет-олимпиад по информатике,

Шовкопляс Григорий Филиппович,
студент первого курса кафедры «Компьютерные технологии» НИУ ИТМО, член жюри Интернет-олимпиад по информатике,

Ульянцев Владимир Игоревич,
аспирант кафедры «Компьютерные технологии» НИУ ИТМО, член жюри Интернет-олимпиад по информатике.