Разбор задачи «C. Предпраздничная суета»

Автор задачи и разбора — К. Абакумов

Решение на 30 баллов

Чтобы найти минимальное время выхода из магазина, необходимо определить, в какие кассы лучше всего становиться в очередь. Из-за небольших ограничений в 1-ой группе тестов можно использовать перебор. Рассмотрим все возможные комбинации из K касс (нам выгодно использовать максимально допустимое количество). Для каждого набора уже несложно выяснить искомое время: очередной тортик выгоднее всего пробить через ту кассу, которая на данный момент освобождается раньше. Если таких несколько, то можно выбрать любую.

Ответом будет минимальное из полученных чисел.

Решение на 60 баллов

Так как школьников в этом случае не меньше чем касс, то достаточно распределить тортики по всем кассам. Идея проста: добавлять тортик в очередь той кассы, которая на данный момент освобождается раньше.

Рассмотрим массив a из N элементов, где $a[i]$ равно времени, в которое освободится i-я касса на данный момент. Изначально $a[i] = A_i + B_i$. Чтобы решение проходило большие тесты второй группы, оганизуем на a кучу, позволяющую брать минимум за $O(1)$ и изменять элементы за $O(\log N)$. Добавляя текущий тортик в очередь минимальной кассы (можно выбрать любую, если таких несколько), необходимо увеличить соответсвующее время. После добавления по одному P тортиков максимальное из $a[i]$ и будет ответом.

Засчет использования кучи это решение работает за $O(P \log N)$, что допустимо при ограничениях $N, P \leq 100\,000$.

Заметим, что если описанный алгоритм применять ко всем комбинациям K касс из N, то он бы успешно работал не только на второй группе тестов (так как в этом случае конфигурация только одна), но и на первой благодаря небольшим ограничениям.

Правильное решение

Основная идея решения на 100 баллов — двоичный поиск по ответу.

Предположим, что прошло ровно t единиц времени. Научимся определять, смогут ли школьники приобести к этому моменту нужные P товаров.

Сначала для каждой кассы подсчитаем количество тортиков, которое можно купить в ней по прошествии t единиц времени (вся эта процедура займет $O(N)$ операций). Далее отсортируем подсчетом кассы по найденному количеству товаров. Эта сортировка работает за время, пропорциональное максимальному возможному из сортируемых значений. Однако, в нашем случае не имеет смысла рассматривать числа, большие $P - 1$. Действительно, если в какой-то из касс можно купить не менее P тортиков, то за время t школьники гаранированно успеют. Поэтому сортировка будет работать за $O(P)$). Осталось выбрать K максимальных значений, просуммировать их, и если полученная сумма будет больше или равна P, то школьники успеют купить все тортики к моменту времени t, иначе — нет. Суммарная сложность этой проверки равна $O(N + P)$.

Для решения задачи остается найти минимальное время, при котором школьники успеют приобрести P тортиков, методом бинарного поиска. Такое решение работает за $O((N+P)\log T)$, где T — максимально возможный ответ.

Возможные ошибки при реализации:

- Использование LongInt вместо Int64
- Деление на 0: надо учесть, что время обработки одного товара может быть равно 0

Ниже приведен текст решения на Delphi:

```
{$APPTYPE CONSOLE}
Const
  MaxN = 100000;
  MaxP = 100000;
  MaxK = 100000;

type
  Line = record //Одна касса:
    w:Integer;  //  w - сколько надо ждать до своей очереди
    t:Integer;  //  t - время обработки одного товара
  end;

var
  lines: array[1..MaxN] of Line;
  sort: array[0..MaxP] of Integer; //массив для сортировки
  n, k, p, i: Integer;
  t, l, r, s: Int64;

function check(t: int64):boolean;
var
  i, S : Integer;
  X : Int64;
begin
  FillChar (sort, SizeOf(sort), 0);
  S := 0;
  //Сортировка
  for i := 1 to n do begin
    if (lines[i].t = 0) and (lines[i].w > t) then begin
      check:=true;
      Exit;
    end else
      if lines[i].t = 0 then Continue;
    x := (t-lines[i].w) div lines[i].t;
    if x<0 then Continue;
    if x>=p then begin
      check:=true;
      Exit;
    end;
```

```
    Inc(sort[x]);
  end;

  //Подсчет суммы
  x := k;
  i := 100000;
  while (x > 0) and (i > 0) do begin
    if sort[i] > 0 then begin
      Dec(sort[i]);
      Inc(s,i);
      Dec(x);
    end else Dec(i);
  end;

  if s >= p then check:=True
  else check:=false
end;

begin
  reset(input,'c.in');
  rewrite(output,'c.out');
  read(n);
  for i := 1 to n do begin
    read(lines[i].t, t, lines[i].w);
    Inc(lines[i].w, t);
  end;
  Read(k, p);
  if k > n then k := n;
  l := 0;
  r := 1000000000000;

  //Поиск времени T
  for i := 2 to n do begin
    t := Int64(lines[i].w) + int64(lines[i].t)*p;
    if t < r then r := t;
  end;

  //Двоичный поиск
  while r - l > 1 do begin
    t := (l + r) div 2;
    if check(t) then
      r := t
    else
      l := t+1;
  end;
  if check(l) then writeln(l) else writeln(r);
  close(input); close(output)
end.
```