Central European Olympiad in Informatics

28 July – 4 August 2005 Sárospatak, Hungary
http://ceoi.inf.elte.hu

Ticket Office

We provide an algorithm using greedy method. The algorithm works in three steps.
The first step allocates as many full-price orders as possible. The greedy method processes the orders in decreasing order of the seat number and allocates it if possible. Denote the resulted schedule by $S1$.
The second step modifies the schedule $S1$ to minimize the wastage. It means that it replaces $p1$ with an order p for which $p < p1$ and $p \, mod \, L$ is minimal (where L is the number of seats in the bunch).
The third step fills in the gaps between full-price orders with half-price orders resulting the schedule $S3$.
We prove that $S3$ is an optimal schedule.

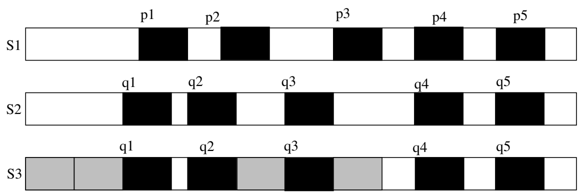

Figure 1:

Let S be an optimal schedule, that is a schedule resulting maximal income. We apply usual method of proving correctness of greedy algorithm. We modify S step by step to obtain $S3$ while the total income remains the same.
Let k be the number of scheduled full-price orders in $S1$. First we prove that there is an optimal schedule with k full-price orders. The number of full-price orders in S is at most k. Assume that the number of full-price orders in S is less then k. If there is an order p_i in $S1$ that does not collide with full-price order in S then p_i collides with at most 2 half-price order in S, therefore removing them from S and including p_i in S the total income does not decrease. See figure 2-a. Repeat this procedure until each order in $S1$ collides with at least one full-price order in S. If the number of full-price orders in S still less than k, then there is a situation shown in figure 2-b. Remove the orders from S that collide with either of the two orders in $S1$ and include the two orders of $S1$ in S.
We may assume that the half-price orders in the schedules are shifted on the left as much as possible. Let p_1 be the first seat

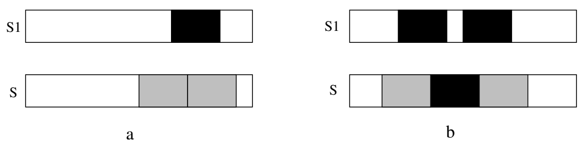

Figure 2:

number of the leftmost full-price order in $S1$, q_1 in $S3$ and r_1 in S. Notice that the full-price order at q_1 must collide with the full-price order at r_1.
If $q_1 < r_1$ then the order at q_1 can not collide with half-price order in S, because in this case the waste would be less than for the first full-price order in $S3$ and the second step of our algorithm would choose it. Therefore we can remove the order at r_1 and include the order at q_1 in S. See figure 3.
Now consider the case $r_1 < q_1$.
Let k be the largest integer such that $a = k \cdot L < r_1$. Moreover, let b the smallest integer such that $a < b$ and the first half-price order in the S end at b. If there is no such half-price order then $b = m$ (m is the largest seat number). See figure 4. It is clear that the number of full-price orders in $S3$ between a and b equals the number of full-price orders in S between a and b. Moreover, if there is a half-price order in S ending at b that there is a half-price order in $S3$ at $a+1$ and conversely. Therefore we can replace the orders of S located between a and b with the orders of $S3$ located between a and b without affecting the total income.
Consider the reduced problem when the available seats are $q_u \ldots m$ and the orders are those that are not scheduled up to b.

Central European Olympiad in Informatics

28 July – 4 August 2005 Sárospatak, Hungary
http://ceoi.inf.elte.hu

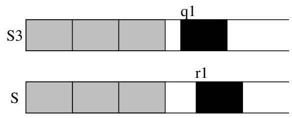

Figure 3:

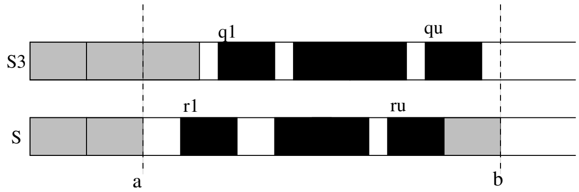

Figure 4:

Complexity:
Time: $O(n+m)$
Memory: $O(n+m)$
Implementation

```
Program ticket;
Const
  MaxN=100000;
  MaxM=30000;
Var
  N,m:longint;
  L:longint;
  Start,R:Array[0..MaxN] of longint;
  S:Array[0..MaxM] of longint;
  Sch:Array[0..MaxM] of longint;
  i,ii,k,x:longint;
  Lend,fn,gap,free:longint;
  Sol:longint;
  OutF:Text;

procedure ReadIn;
var
  InFile:Text;
  i,x:longint;
begin
  assign(InFile,'ticket.in'); reset(InFile);
  readln(InFile,m, L);
  readln(InFile, N);
  for i:=1 to N do begin
    read(InFile, x);
    Start[i]:=x;
    R[i]:=i;
  end {for i};
  Start[0]:=0;
```

```
  Start[n+1]:=m+1;
end {ReadIn};

procedure Sort;
var
  Nox:array[0..MaxM] of longint;
  i,ii,j,x:longint;
begin
  Nox[0]:=0;
  for x:=1 to m do Nox[x]:=0;
  for i:=1 to n do inc(Nox[Start[i]]);
  for x:=1 to m do Nox[x]:=Nox[x]+Nox[x-1];
  for i:=1 to n do begin
    ii:=Start[i];
    j:=Nox[ii];
    R[j]:=i;
    dec(Nox[ii]);
  end;
  R[0]:=0;
  R[n+1]:=n+1;
end {Sort};

Begin {Prog}
  ReadIn;
  Sort;
  fn:=0;
  Lend:=m+1;
{1. step: allocate full-price orders}
  for i:=N downto 1 do begin
    ii:=R[i];
    if Start[ii]+L<=Lend then begin
      inc(fn);
      S[fn]:=ii;
      Lend:=Start[ii];
    end;
  end {for i};

  for i:=1 to fn div 2 do begin
    x:=S[i]; S[i]:=S[fn-i+1];
    S[fn-i+1]:=x;
  end;
{2. step: modify the schedule to minimize the wastage }
  S[fn+1]:=n+1;
  free:=1;
  k:=1;
  gap:=m;
  Sol:=0;
  Lend:=Start[S[1]];
  for i:=1 to n do begin
    if Start[R[i]]>Lend then begin
      Sol:=Sol+(Start[S[k]]-free) div L+1;
```

```
      free:=Start[S[k]]+L;
      inc(k);
      Lend:=Start[S[k]];
      gap:=m;
    end;
    if (free<=Start[R[i]])and((Start[R[i]]-free) mod L<gap) then begin
      gap:=(Start[R[i]]-free) mod L;
      S[k]:=R[i];
    end;
  end {for i};

  Sol:=Sol+(Start[S[fn]]-free) div L;
  Sol:=Sol+(m-(Start[S[fn]])+1) div L;
  if Sol>n then Sol:=n;
  assign(OutF, 'ticket.out'); rewrite(OutF);
  writeln(OutF, Sol+fn);
  writeln(OutF, Sol);
{3. step: fill in the gaps with half-price orders}
  for i:=1 to m do Sch[i]:=0;
  for i:=1 to fn do
    for k:=Start[S[i]] to Start[S[i]]+L-1 do
      Sch[k]:=S[i];

  free:=1; i:=1;
  while i<=n do begin
    if free+L-1>m then break;
    if Sch[Start[R[i]]]=R[i] then begin
      inc(i); continue;
    end;
    k:=Sch[free+L-1];
    if (Sch[free]=0)and(k=0) then begin
       Sch[free]:=R[i];
       free:=free+L;
       inc(i);
    end else
      free:=Start[k]+L;
  end {while i};
  k:=1;
  while k<=m do begin
    if Sch[k]>0 then begin
       writeln(OutF, Sch[k],' ',k);
       k:=k+L;
    end else begin
       inc(k);
    end;
  end;
  close(OutF);
End.
```