USA COMPUTING OLYMPIAD

Mixing Milk

Russ Cox

Since we're acquiring things that are all of the same size (in this case, units of milk), a greedy solution will suffice: we sort the farmers by price, and then buy milk from the farmers with the lowest prices, always completely exhausting one farmer's supply before moving on to the next one.

To do this, we read the input into Farmer structures, sort the array by price, and then walk the array, buying milk until we've got all the milk we want.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <assert.h>

#define MAXFARMER 5000

typedef struct Farmer Farmer;
struct Farmer {
        int p;  /* price per gallon */
        int a;  /* amount to sell */
};

int
farmcmp(const void *va, const void *vb)
{
        return ((Farmer*)va)->p - ((Farmer*)vb)->p;
}

int nfarmer;
Farmer farmer[MAXFARMER];

void
main(void)
{
        FILE *fin, *fout;
        int i, n, a, p;

        fin = fopen("milk.in", "r");
        fout = fopen("milk.out", "w");

        assert(fin != NULL && fout != NULL);

        fscanf(fin, "%d %d", &n, &nfarmer);
        for(i=0; i<nfarmer; i++)
                fscanf(fin, "%d %d", &farmer[i].p, &farmer[i].a);

        qsort(farmer, nfarmer, sizeof(farmer[0]), farmcmp);

        p = 0;
        for(i=0; i<nfarmer && n > 0; i++) {
                /* take as much as possible from farmer[i], up to amount n */
                a = farmer[i].a;
                if(a > n)
                        a = n;
                p += a*farmer[i].p;
                n -= a;
        }

        fprintf(fout, "%d\n", p);
        exit(0);
}
```

Ran Pang of Canada writes:

Here is a program that solves the problem in linear time (with respect to the maximum price, and number of farmers, since we have to read in the data anyway), while I think the qsort used by the solution would consume O(n log n), where n is the number of farmers.

```
#include<stdio.h>

#define MAXPRICE 1001

int amount_for_price[MAXPRICE]={0};
int N, M;

int Cal(void);
int Read(void);

int main(void) {
    Read();
    Cal();
    return 0;
}

int Cal(void) {
    int i;
    int price_total=0;
    int milk_total=0;
    for(i=0;i<MAXPRICE;i++) {
        if(amount_for_price[i]) {
            if(milk_total+amount_for_price[i]<N) {
                price_total+=(i*amount_for_price[i]);
                milk_total+=amount_for_price[i];
            }
            else {
                int amount_needed = N-milk_total;
                price_total+=(i*amount_needed);
                break;
            }
        }
    }
    {
        FILE* out=fopen("milk.out","w");
        fprintf(out,"%d\n",price_total);
        fclose(out);
    }
    return 0;
}

int Read(void) {
    FILE* in = fopen("milk.in","r");
    int i, price, amount;
    fscanf(in,"%d %d",&N,&M);
    for(i=0;i<M;i++) {
        fscanf(in, "%d %d", &(price), &(amount));
        amount_for_price[price]+=amount;
    }
    fclose(in);
    return 0;
}
```

\f2Here is another solution from SVK's Adam Okruhlica\fP

It is unnecessary to sort the prices with quicksort in O(n.lg.n) time, because there is an upper limit of the a single price ($1000) and we know that all prices are integral. We can sort this array with count sort. We establish a 'box' for each of the available prices (0..1000). We save the input to an array. Then we iterate through each farmer and we memoize his index in the (0..1000) array on index equivalent to the price he offers us. Hence there can be more farmers offering the same price we put them in a linked list. Finally we iterate the array from 0 to 1000 andpick the farmers' indexes from the linked lists. It's pretty easy to implement, and the time complexity is O(n).

```
program milk;

type pList = ^List;
      List = record
                farmer:longint;
                next:pList;
              end;
      HeadList = record
                   head:pList;
                   tail:pList;
                 end;

var fIn,fOut:text;
    sofar,i,x,want,cnt,a,b:longint;
    sorted,cost,amount:array[1..5010] of longint;
    csort:array[0..1010] of HeadList;

    t:pList;

begin
    assign(fIn,'milk.in');reset(fIn);
    assign(fOut,'milk.out'); rewrite(fOut);

    readln(fIn,want,cnt);
    for i:=1 to cnt do readln(fIn,cost[i],amount[i]);

    for i:=0 to 1000 do begin
         new(csort[i].head);
         csort[i].tail:=csort[i].head;
         csort[i].head^.farmer:=-1;
    end;

    {Cast indexes into the array}
    for i:=1 to cnt do begin

       t:=csort[cost[i]].tail;
       if t^.farmer = -1 then t^.farmer:=i;
       new(t^.next);
       t^.next^.farmer:=-1;
       csort[cost[i]].tail:=t^.next;
    end;

    {Pick indexes}
    x:=1;
    for i:=0 to 1000 do begin
        t:=csort[i].head;
        while t^.farmer > 0 do begin
          sorted[x]:=t^.farmer;
          inc(x);
          t:=t^.next;
        end;
    end;

    sofar:=0;
    for i:=1 to cnt do begin
      if want < amount[sorted[i]] then begin
        inc(sofar,want*cost[sorted[i]]);
        want:=0; break;
      end

      else inc(sofar,amount[sorted[i]]*cost[sorted[i]]);
      dec(want,amount[sorted[i]]);
    end;

    writeln(fOut,sofar);
    close(fOut);
end.
```

Dwayne Crooks writes: Do we really need the linked list that SVK's Adam Okruhlica uses in his solution, I don't think so. Here is a better solution that uses essentially the same idea as Adam's solution but does away with the linked list. Takes O(max(MAXP,M)) time, where MAXP=1000 and M<=5000 is the input size. *Editors note: Dwayne should be using long long integers (64 bit) instead of int's in order to avoid overflow.*

```
#include <iostream>
#include <fstream>

#define MAXP 1000

using namespace std;

int main() {
    ifstream in("milk.in");
    ofstream out("milk.out");

    int N, M;
    int P[MAXP+1];

    in >> N >> M;
    for (int i = 0; i <= MAXP; i++) P[i]=0;
    for (int i = 0; i < M; i++) {
        int price, amt;
        in >> price >> amt;

         // we can add amounts that cost the same price
        // since x gallons costing c cents and
        //          y gollons costing c cents
        // is the same as
        //      x+y gallons costing c cents
        P[price] += amt;
    }

    // greedy choice: take as much of the item that
    // has the least price per gallon
    int res = 0;
    for (int p = 0; p<=MAXP && N>0; p++) {
        if (P[p]>0) {
            res+=p*(N<P[p]?N:P[p]);
            N-=P[p];
        }
    }
    out << res << endl;

    in.close();
    out.close();

    return 0;
}
```

As the final word, Bulgaria's Miroslav Paskov has distilled all the best ideas into a simple solution:

```
#include <fstream>
#define MAXPRICE 1001
using namespace std;

int main() {
    ifstream fin ("milk.in");
    ofstream fout ("milk.out");
    unsigned int i, needed, price, paid, farmers, amount, milk[MAXPRICE][2];
    paid = 0;
    fin>>needed>>farmers;
    for(i = 0;i<farmers;i++){
        fin>>price>>amount;
        milk[price][0] += amount;
    }
    for(i = 0; i<MAXPRICE && needed;i++){
        if(needed> = milk[i][0]) {
            needed -= milk[i][0];
            paid += milk[i][0] * i;
        } else if(milk[i][0]>0) {
            paid += i*needed;
            needed = 0;
        }
    }
    fout << paid << endl;
    return 0;
}
```