USING DYNAMIC PROGRAMMING ALGORITHM TECHNIQUE FOR SOLVING PROBLEMS OF OLYMPIAD IN INFORMATICS

Nicolai FALICO, Mihail KULEV

Universitatea Tehnică a Moldovei

Abstract: *Each year, during 5 last years, authors organized the National Technical Olympiad in Informatics at Technical University of Moldova for high school students by proposing for solving different Olympiad problems. Some of these problems needed for their solution algorithm of dynamic programming [1, 2]. In this paper the solutions of two problems taken from site ACM.TIMUS.RU for Olympiad held in 2012 have been considered and codes of corresponding programs in C language have been presented.*

Key words: *Dynamic programming, problems of Olympiad in Informatics, programs in C language.*

1. Conditions of the problems (http://acm.timus.ru)

The first problem: 1005. Stone Pile.

You have a number of stones with known weights w1, …, wn (1≤n≤20 and, 1 ≤ wi ≤ 100000). Write a program that will rearrange the stones into two piles such that weight difference between the piles is minimal.

The second problem: 1009. K-based Numbers.

Let's consider K-based numbers, containing exactly N digits. We define a number to be valid if its K-based notation doesn't contain two successive zeros. For example:
* 1010230 is a valid 7-digit number;
* 1000198 is not a valid number;
* 0001235 is not a 7-digit number, it is a 4-digit number.

Given two numbers N and K, you are to calculate an amount of valid K based numbers, containing N digits. You may assume that 2 ≤ K ≤ 10; N ≥ 2; N + K ≤ 18.

2. Solutions of the problems

Dynamic programming is a method for solving complex problems by breaking them down into simpler subproblems. In [1, 2] the good introduction in dynamic programming is given. In general, to solve a problem, we need to solve different parts of the problem (subproblems), and then combine the solutions of the subproblems to reach an overall solution.

The first problem (Stone pile) - is an important problem in the theory of algorithms and cryptography. The problem is NP-complete and can also be regarded as a certain special case of the knapsack problem. The computational complexity of the problem depends on two parameters - the number n of elements of the set, and weights of the stones w1, …, wn. As n is small, the exhaustive search is acceptable. But the weights of the stones w1, …, wn are also small, and because of that it is better to use dynamic programming. At first we find the sum of weights of all the stones w1+ …+ wn=W. Then we reduce this problem to a 0-1 knapsack problem - determine the number of each item to include in a collection so that the total weight is less than or equal to a given limit = W/2 and the total value is as large as possible – assuming that weight is equal to a value. Then for each w ≤ W/2, define m[w] to be the maximum value that can be attained with total weight less than or equal to w. Then m[W/2] is the solution to the problem.

Observe that m[w] has the following properties:

m[0]=0 (the sum of zero items, i.e., the summation of the empty set);
m[w]= max(wi+m[w-wi] : wi<=W/2).

The idea used is that the solution for a knapsack is the same as the value of one correct item plus the solution for a knapsack with smaller capacity, specifically one with the capacity reduced by the weight of that chosen item. Here the maximum of the empty set is taken to be zero. Tabulating the results from m[0] up through m[W/2] gives the solution. Since the calculation of each m[w] involves examining n items, and there are W/2 values of m[w] to calculate, the running time of the dynamic programming solution is O(nW).

The accepted cod in C language is presented below:

```
#include<stdio.h>
#include<math.h>
#define M 20000002
int m[M]={1};
int main()
{
    int n,i,w, wi[21]={0},W=0;
            scanf("%d",&n);
    for(i=1;i<=n;i++)
        {
            scanf("%d",&wi[i]);
            W+=wi[i];
    }
    for(i=1;i<=n;i++)
            for(w=W/2;w;w--)
            if(w>=wi[i] && m[w-wi[i]])
                    m[w]=1;
    w=W/2;
    while(m[w]==0) w--;
    printf("%d",W-w*2);
    return 0;
}
```

It is interesting that due to a very simple restriction for n ($1 \le n \le 20$) we can solve this problem with brute force without using dynamic programming. The accepted cod is presented below:

```
#include<stdio.h>
#include<math.h>
int main( )
{
  long
a[20],s=0,s1,t=1l,j=0,min=2000000000;
    int n,i;
    scanf("%d",&n);
    for(i=0;i<n;i++)
        {
            scanf("%ld",&a[i]);
            s+=a[i];
    }
    t<<=n;
    for(j=0;j<t;j++)
        {
     s1=0;
     for(i=0;i<n;i++)
       if(j&(1<<i)) s1+=a[i];
       if(labs(s-2*s1)<min)      min=labs(s-
2*s1);
     }
    printf("%ld",min);
       return 0;
}
```

The second problem (K-based Numbers) is more complicated. The solution approached the construction of N digit numbers from N-1 digit numbers in the typical way of constructing numbers: multiply N-1 digit number by base K, and add the new digits to the units' position. We can be sure that we're not missing any numbers. We must consider numbers that end in zero separately from those that don't.

Let Z(n) - number of valid numbers ending in zero and NZ(n) - number of valid numbers not ending in zero. It is evident that $Z(2) = k - 1$ and $NZ(2) = (k - 1) * k - (k - 1)$ (total number of valid 2-digit numbers, minus those that end in 0). Then we have the recurrence formulae:

$Z(n) = NZ(n - 1)$,

$NZ(n) = (Z(n-1) + NZ(n-1)) * (k - 1)$.

Compute Z(n) and NZ(n) iteratively and then output F(n) at the end $F(n) = Z(n) + NZ(n)$.

The accepted cod of this problem is presented below. In this cod the recurrence:

$F(n)=(N-1)*(F(n-1)+F(n-2))$,

have been used.

```
#include<stdio.h>

long a[20]={1};

int k;

long rec(int x)
{
     if(a[x])
       return a[x];
     else
       return   a[x]=(k-1)*(rec(x-1)+rec(x-
2));
}
int main()
{
    int n;
    scanf("%d%d",&n,&k);
    a[1]=k-1;
    rec(n);
    printf("%ld",rec(n));
         return 0;
}
```

Bibliography

1. КОРМЕН, Т., ЛЕЙЗЕРСОН, Ч., РИВЕСТ, Р. Алгоритмы: построение и анализ. М., МЦНМО, 2001, 960 с.

2. ОКУЛОВ С. Программирование в алгоритмах. М., БИНОМ, 2002, 341 с.