目录

abc480 1
E.Product-Bounded Arithmetic Sequences 1
F.Cake Cutting 4

abc480

E.Product-Bounded Arithmetic Sequences

题目描述

一个由正整数组成的序列 $A = \left(A_1, ..., A_{|A|}\right)$ 被称为「好序列」，当且仅当它满足以下所有条件：

- $|A| \geq 3$；
- A 是等差数列，即对所有 $1 \leq i \leq |A| - 1$，都有 $A_{i+1} - A_i = A_2 - A_1$；
- $A_2 - A_1 > 0$。

求所有元素乘积不超过 N 的好序列的数量。

可以证明，在题目约束下答案不超过 10^{18}。

约束条件

- $1 \leq N \leq 10^{18}$
- N 是整数。

输入格式

输入由标准输入给出，格式如下：

```
1 N
```

输出格式

输出答案。

样例

1.in / 1.out

```
1 120
```

```
1 18
```

乘积不超过 24 的好序列有 4 个：$(1, 2, 3), (1, 2, 3, 4), (1, 3, 5), (2, 3, 4)$。

2.in / 2.out

```
1 1000000000000000000
```

```
1 1051148114007
```

题解

自己想到给我想爽了。

思考一下，数列最少是三项，由于乘积小于等于 10^{18}，也就是最大值小于 10^6。

这意味着什么？如果数列是三项，数列的起始值（当然也可以是末尾值）是可以枚举的！

而很显然，首项固定的情况下，数列的乘积随公差单调递增，我们可以二分公差。数列项数增加的时候，范围只会越来越小。

```cpp
#include <stdio.h>
typedef long long i64;
typedef __int128_t i128;

i64 n, res;

inline i64 mulmx(int c, int m, int k) {
    i128 res = 1;
    for (int i = 1; i <= c; i++) {
        res *= m + i128((i - 1)) * k;
        if (res > n)
            return n + 1;
    }
    return res;
}

inline i64 pow(i64 x, int c) {
    i128 ans = 1;
    for (int i = 1; i <= c; i++) {
        ans *= x;
        if (ans > n)
            return n + 1;
    }
    return ans;
}

i64 nsq(i64 x, int c) {
    i64 l = 1, r = x;
    while (r - l > 1) {
        i64 mid = (l + r) >> 1;
        if (pow(mid, c) > x)
            r = mid - 1;
        else
            l = mid;
    }

    if (pow(r, c) > x)
        return l;
    else
        return r;
}

int main() {

    scanf("%lld", &n);

    for (int pw = 3; pw <= 20; pw++) {

```

```
49          i64 ns = nsq(n, pw);
50          if (mulmx(pw, 1, 1) > n) {
51              break;
52          }
53
54          for (int m = 1; m <= ns; m++) {
55              if (mulmx(pw, m, 1) > n) {
56                  break;
57              }
58
59              i64 l = 1, r = 1;
60              while (mulmx(pw, m, r) <= n)
61                  r <<= 1;
62
63              while (r - l > 1) {
64                  i64 mid = (l + r) >> 1;
65
66                  if (mulmx(pw, m, mid) > n)
67                      r = mid - 1;
68                  else
69                      l = mid;
70              }
71
72              i64 ans = -1;
73              if (mulmx(pw, m, r) > n)
74                  ans = l;
75              else
76                  ans = r;
77
78
79              res += ans;
80          }
81      }
82
83      printf("%lld", res);
84      return 0;
85  }
```

以上的一些地方可以优化：

- 为了避免连乘溢出，可以转化为除法。

标程真的很短……我的算法多了一些东西，具体地：

- 二分多判断了一次区间，其实可以直接返回合法的那边。
- 使用二分进行开方运算，多余的溢出检测（改成除就好了）
- 枚举了数列个数作为外层循环，其实可以像标程一样在里面二分（不过这点其实无所谓）

标程的做法相当于枚举首项，然后使用 NN 作为估计值来判断能不能继续算。当 NN 为 0，代表即使公差为 1 也不能得到答案，自然不用算下去。

这个做法相当巧妙，应该学习。

```cpp
#include<bits/stdc++.h>
using namespace std;
using ll=long long;

int main(){
  ll N;
  cin >> N;

  auto calc=[&](int a, int n){
    int ok=0,ng=1e9;
    while(ng-ok>1){
      int mid=(ok+ng)/2;
      ll s=N;
      for(int i=0;i<n;i++)s/=a+i*mid;
      if(s)ok=mid;
      else ng=mid;
    }
    return ok;
  };

  ll ans=0;
  for(int a=1;a<=1e6;a++){
    int n=3;
    ll NN=N/a/(a+1)/(a+2);
    while(NN){
      ans+=calc(a,n);
      NN/=a+n;
      n++;
    }
  }
  cout << ans << endl;
}
```

F.Cake Cutting

题目描述

有 N 个蛋糕，第 i 个蛋糕的大小为 A_i。你要把这些蛋糕分给 M 个人。

设 S 为所有蛋糕的总大小，即 $A_1 + ... + A_N$。这里保证 S 是 M 的倍数。

你可以以任意顺序执行任意次（也可以不执行）以下操作：

- 选择一个蛋糕，以任意方式将其切成 2 块（大小不必为整数）。

操作结束后，把蛋糕分给 M 个人。求使得每个人分到的蛋糕总大小都为 $\frac{S}{M}$ 所需的最少操作次数。

更形式化的表述如下：

- 给定序列 $A = (A_1, ..., A_N)$ 与正整数 M，令 $S = \sum_i A_i$。
- 每次操作：选择整数 i（$1 \le i \le |A|$）与实数 x（$0 < x < A_i$），删除 A 的第 i 个元素，并在末尾追加 x 和 $A_i - x$，序列长度变为 $|A| + 1$。

- 求使得存在集合 $\mathcal{P}$ 满足以下条件的最少操作次数：$\mathcal{P}$ 是下标集合 $\{1,2,...,|A|\}$ 的一个划分（每个元素都是下标集合的子集、两两交集为空、并集为全集），且对每个 $X \in \mathcal{P}$ 都有 $\sum_{i \in X} A_i = \frac{S}{M}$。

约束条件

- $1 \leq N \leq 20$
- $1 \leq M \leq 100$
- $1 \leq A_i \leq 10^9$
- $A_1 + ... + A_N$ 是 M 的倍数。
- 所有输入均为整数。

输入格式

输入由标准输入给出，格式如下：

```
N M
A_1 A_2 ... A_N
```

输出格式

输出答案。

样例

1.in / 1.out

```
2 3
7 23
```

```
2
```

最少通过 2 次操作达成，过程如下：

- 把大小为 23 的蛋糕切成 10 和 13，蛋糕大小变为 $(7,10,13)$。
- 把大小为 13 的蛋糕切成 10 和 3，蛋糕大小变为 $(7,10,10,3)$。
- 把大小为 7 和 3 的蛋糕分给一个人，大小为 10 的蛋糕分给另一个人，剩下大小为 10 的蛋糕分给最后一个人。

2.in / 2.out

```
4 2
1234 123 12 1
```

```
1
```

最少通过 1 次操作达成，过程如下：

- 把大小为 1234 的蛋糕切成 685 和 549，蛋糕大小变为 $(123,12,1,685,549)$。
- 把大小为 $123,12,1,549$ 的蛋糕分给一个人，大小为 685 的蛋糕分给另一个人。

题解

一定要看数据范围呀！$N \leq 20, M \leq 100$，肯定有一种让我们尝试所有枚举方案的意思。

如果用纯粹的暴力，形式化的说，那我们就需要枚举一个排列 $P = (1,2,3,...,n)$，从左至右加入蛋糕，记 $T = \frac{S}{M}$ 如果大小 $> T$ 我们就切一刀，$= T$ 我们就直接分一块。

很容易说明当 P 枚举完全排列后，此问题和原问题是等价的。

但是在 $N \leq 20$ 的情况下，$N!$ 的复杂度是不可接受的，然后我们就想每次枚举必须从头枚举排列吗？不见得，因为 $P_1 = (1, 2, 4, 5, ..., n), P_2 = (1, 2, 5, 4, ..., n)$ 显然都是从 $(1, 2)$ 完成后的结果出发。

那最后我们就想到了状压 dp。具体地，令 $A \in [0, 2^n]$ 的二进制数代表我们所选的集合，f_A 代表已经选择完集合 A 后的答案，s_A 代表选择完集合 A 后，最后一个块的蛋糕体积。（容易说明，s_A 不随枚举顺序变化，所以在哪里处理都是无所谓的。）

然后，记我们现在要选择第 i 个数加入集合 S，记 $I = 2^i, i \in [0, n)$ 如果 $s_{S \cup I} > T$，我们需要在这里切一刀，如果 $s_{S \cup I} \leq T$ 就不用切，此时如果 $s_{S \cup I} = T$ 我们就清空这个块。

实现以及部分转移思路和标程有不同，我认为我这个更好理解。

```cpp
#include <stdio.h>
const int maxn = 20;
const int L = (1 << maxn) + 7;

typedef long long i64;
i64 f[L], s[L], A[maxn + 7];
int n, m;
i64 sum, k;

inline i64 min(i64 a, i64 b) { return a < b ? a : b; }

int main() {
    scanf("%d%d", &n, &m);

    for (int i = 0; i < (1 << n); i++)
        f[i] = (1ll << 62);
    f[0] = 0;

    for (int i = 0; i < n; i++)
        scanf("%lld", &s[1 << i]), sum += s[1 << i];
    k = sum / m;

    for (int i = 0; i < n; i++)
        f[1 << i] = (s[1 << i] - 1) / k, s[1 << i] %= k;

    for (int i = 1; i < (1 << n); i++)
        for (int j = 0; j < n; j++)
            if ((i & (1 << j)) == 0) {
                s[i | (1 << j)] = s[i] + s[(1 << j)];
                if (s[i] + s[1 << j] > k) {
                    s[i | (1 << j)] -= k;
                    f[i | (1 << j)] =
                        min(f[i | (1 << j)], f[i] + f[1 << j] + 1);
                } else {
                    if (s[i] + s[1 << j] == k)
                        s[i | (1 << j)] = 0;
                    f[i | (1 << j)] = min(f[i | (1 << j)], f[i] + f[1 << j]);
                }
```

```
            }

    printf("%lld\n", f[(1 << n) - 1]);

    return 0;
}
```