package main

import "testing"

func FiboRecursive(n uint64) uint64 {

if n < 2 {

return 1

} else {

return FiboRecursive(n-1) + FiboRecursive(n-2)

}

}

func FiboCyclic(n uint64) uint64 {

a, b := uint64(1), uint64(0) // a = f[0], b = f[-1]

for n > 0 {

a, b = a+b, a

n--

}

return a

}

type memo []uint64

func F(n uint64, m memo) uint64 {

if m[n] == 0 {

m[n] = F(n-1, m) + F(n-2, m)

}

return m[n]

}

func FiboMemo(n uint64) uint64 {

m := make([]uint64, n+1, n+1)

m[0], m[1] = 1, 1

return F(n, m)

}

// Названия тестируемых функций должны начинаться на Benchmark,

// за которым идёт название, начинающееся с большой буквы

func BenchmarkFiboRecursive16(b \*testing.B) {

for i := 0; i < b.N; i++ {

FiboRecursive(16)

}

}

func BenchmarkFiboRecursive24(b \*testing.B) {

for i := 0; i < b.N; i++ {

FiboRecursive(24)

}

}

func BenchmarkFiboRecursive32(b \*testing.B) {

for i := 0; i < b.N; i++ {

FiboRecursive(32)

}

}

func BenchmarkFiboRecursive40(b \*testing.B) {

for i := 0; i < b.N; i++ {

FiboRecursive(40)

}

}

func BenchmarkFiboCyclic16(b \*testing.B) {

for i := 0; i < b.N; i++ {

FiboCyclic(16)

}

}

func BenchmarkFiboCyclic24(b \*testing.B) {

for i := 0; i < b.N; i++ {

FiboCyclic(24)

}

}

func BenchmarkFiboCyclic32(b \*testing.B) {

for i := 0; i < b.N; i++ {

FiboCyclic(32)

}

}

func BenchmarkFiboCyclic40(b \*testing.B) {

for i := 0; i < b.N; i++ {

FiboCyclic(40)

}

}

func BenchmarkFiboCyclic60(b \*testing.B) {

for i := 0; i < b.N; i++ {

FiboCyclic(60)

}

}

func BenchmarkFiboCyclic90(b \*testing.B) {

for i := 0; i < b.N; i++ {

FiboCyclic(90)

}

}

func BenchmarkFiboMemo16(b \*testing.B) {

for i := 0; i < b.N; i++ {

FiboMemo(16)

}

}

func BenchmarkFiboMemo24(b \*testing.B) {

for i := 0; i < b.N; i++ {

FiboMemo(24)

}

}

func BenchmarkFiboMemo32(b \*testing.B) {

for i := 0; i < b.N; i++ {

FiboMemo(32)

}

}

func BenchmarkFiboMemo40(b \*testing.B) {

for i := 0; i < b.N; i++ {

FiboMemo(40)

}

}

func BenchmarkFiboMemo60(b \*testing.B) {

for i := 0; i < b.N; i++ {

FiboMemo(60)

}

}

func BenchmarkFiboMemo90(b \*testing.B) {

for i := 0; i < b.N; i++ {

FiboMemo(90)

}

}

benchmark\_Fibonacci\_test.go

package main

// Запуск из командной строки:

// go test -bench . benchmark\_Invest\_test.go

// Имя файла обязательно должно заканиваться на \_test

import (

"fmt"

"os"

"testing"

)

var (

inc [][]int

n, maxsum int

)

func init() {

f, \_ := os.Open("income.dat")

fmt.Fscan(f, &n, &maxsum)

// n - количество вариантов инвестиций (производств)

// sum - максимально возможная суммарная сумма инвестиций

// inc[i][s] - доход, который приносит инвестиция

// s денег в производство #i

inc = append(inc, make([]int, maxsum+1, maxsum+1))

for i := 1; i <= n; i++ {

inc = append(inc, make([]int, maxsum+1, maxsum+1))

inc[i][0] = 0

for s := 1; s <= maxsum; s++ {

fmt.Fscan(f, &inc[i][s])

}

}

}

func MaxIncomeCyclic(n int, sum int) int {

// инвестируем sum денег в проекты 1..n

var max, w int

var res [][]int

res = append(res, make([]int, sum+1, sum+1))

for k := 1; k <= n; k++ {

res = append(res, make([]int, sum+1, sum+1))

max = inc[k][sum]

for s := 1; s <= sum; s++ {

for reminder := 1; reminder <= s; reminder++ {

w = inc[n][s-reminder] + res[k-1][reminder]

if w > max {

max = w

}

}

res[k][s] = max

}

}

return res[n][sum]

}

func MaxIncomeRecursive(n int, sum int) (result int) {

// инвестируем sum денег в проекты 1..n

var w int

if n > 0 {

result = inc[n][sum]

for reminder := 1; reminder <= sum; reminder++ {

w = inc[n][sum-reminder] + MaxIncomeRecursive(n-1, reminder)

if w > result {

result = w

}

}

return result

} else {

return 0

}

}

type memo [][]int

func MaxIncome(n int, sum int, m memo) int {

// Рекурсивная функция с мемоизацией

// инвестируем sum денег в проекты 1..n

var w, result int

if n > 0 && m[n][sum] == 0 {

result = inc[n][sum]

for reminder := 1; reminder <= sum; reminder++ {

w = inc[n][sum-reminder] + MaxIncome(n-1, reminder, m)

if w > result {

result = w

}

}

m[n][sum] = result

}

return m[n][sum]

}

func MaxIncomeMemo(n int, sum int) (result int) {

// Слайс res используется для мемоизации

var res [][]int

// Подготавливаем слайс res, он заполняется нулями.

for i := 0; i <= n; i++ {

res = append(res, make([]int, sum+1, sum+1))

}

// Рекурсивно (с мемоизацией) вычисляем максимальный доход,

// получаемый при инвестировании sum денег в проекты 1..n.

return MaxIncome(n, sum, res)

}

// Названия тестируемых функций должны начинаться на Benchmark,

// за которым идёт название, начинающееся с большой буквы

func BenchmarkInvestRecursive\_5\_10(b \*testing.B) {

for i := 0; i < b.N; i++ {

MaxIncomeRecursive(5, 10)

}

}

func BenchmarkInvestRecursive\_4\_20(b \*testing.B) {

for i := 0; i < b.N; i++ {

MaxIncomeRecursive(4, 20)

}

}

func BenchmarkInvestRecursive\_5\_20(b \*testing.B) {

for i := 0; i < b.N; i++ {

MaxIncomeRecursive(5, 20)

}

}

func BenchmarkInvestCyclic\_5\_10(b \*testing.B) {

for i := 0; i < b.N; i++ {

MaxIncomeCyclic(5, 10)

}

}

func BenchmarkInvestCyclic\_4\_20(b \*testing.B) {

for i := 0; i < b.N; i++ {

MaxIncomeCyclic(4, 20)

}

}

func BenchmarkInvestCyclic\_5\_20(b \*testing.B) {

for i := 0; i < b.N; i++ {

MaxIncomeCyclic(5, 20)

}

}

func BenchmarkInvestMemo\_5\_10(b \*testing.B) {

for i := 0; i < b.N; i++ {

MaxIncomeMemo(5, 10)

}

}

func BenchmarkInvestMemo\_4\_20(b \*testing.B) {

for i := 0; i < b.N; i++ {

MaxIncomeMemo(4, 20)

}

}

func BenchmarkInvestMemo\_5\_20(b \*testing.B) {

for i := 0; i < b.N; i++ {

MaxIncomeMemo(5, 20)

}

}

benchmark\_Invest\_test.go

package main

import "testing"

const mod = 1000000007

func Mult(a, b uint32) uint32 {

return uint32(uint64(a) \* uint64(b) % mod)

}

func PowerCyclic0(a uint32, n uint32) uint32 {

var res uint32 = 1

for ; n > 0; n-- {

res = Mult(res, a)

}

return res

}

func PowerRecursive0(a uint32, n uint32) uint32 {

if n == 0 {

return 1

}

return Mult(a, PowerRecursive0(a, n-1))

}

func PowerRecursive1(a uint32, n uint32) uint32 {

var b uint32

if n == 0 {

return 1

} else {

b = PowerRecursive1(a, n/2) // b = a^(n/2)

b = Mult(b, b)

if n%2 == 0 {

return b

} else {

return Mult(b, a)

}

}

}

func PowerRecursive1a(a uint32, n uint32) uint32 {

var b uint32

if n == 0 {

return 1

} else {

b = Mult (PowerRecursive1a(a, n/2), PowerRecursive1a(a, n/2))

if n%2 == 0 {

return b

} else {

return Mult(b, a)

}

}

}

func PowerRecursive2(a uint32, n uint32) uint32 {

var b uint32

if n == 0 {

return 1

} else {

b = Mult(a, a) // b = a^2

b = PowerRecursive2(b, n/2)

if n%2 == 0 {

return b

} else {

return Mult(b, a)

}

}

}

func PowerCyclic2(a uint32, n uint32) uint32 {

var res uint32 = 1

for n > 0 {

if n%2 == 1 {

res = Mult(res, a)

}

n /= 2

a = Mult(a, a)

}

return res

}

func BenchmarkPowerCyclic0\_6(b \*testing.B) {

for i := 0; i < b.N; i++ {

PowerCyclic0(2020, 1000000)

}

}

func BenchmarkPowerRecursive0\_6(b \*testing.B) {

for i := 0; i < b.N; i++ {

PowerRecursive0(2020, 1000000)

}

}

func BenchmarkPowerRecursive1\_6(b \*testing.B) {

for i := 0; i < b.N; i++ {

PowerRecursive1(2020, 1000000)

}

}

func BenchmarkPowerRecursive1\_9(b \*testing.B) {

for i := 0; i < b.N; i++ {

PowerRecursive1(2020, 1000000000)

}

}

func BenchmarkPowerRecursive1a\_6(b \*testing.B) {

for i := 0; i < b.N; i++ {

PowerRecursive1a(2020, 1000000)

}

}

func BenchmarkPowerRecursive2\_6(b \*testing.B) {

for i := 0; i < b.N; i++ {

PowerRecursive2(2020, 1000000)

}

}

func BenchmarkPowerRecursive2\_9(b \*testing.B) {

for i := 0; i < b.N; i++ {

PowerRecursive2(2020, 1000000000)

}

}

func BenchmarkPowerCyclic2\_6(b \*testing.B) {

for i := 0; i < b.N; i++ {

PowerCyclic2(2020, 1000000)

}

}

func BenchmarkPowerCyclic2\_9(b \*testing.B) {

for i := 0; i < b.N; i++ {

PowerCyclic2(2020, 1000000000)

}

}

benchmark\_Power\_test.go

package main

import "fmt"

type arr [1000000]int64

func UnlimitedRecursion(a arr) {

fmt.Println(a[0])

a[0]++

UnlimitedRecursion(a)

}

func main() {

var c arr

UnlimitedRecursion(c)

}

stack\_overflow.go

package main

import "fmt"

func P(n byte) {

if n>0 {

P(n / 2)

fmt.Printf("%d.", n)

P(n / 3)

}

}

func main() {

P(20)

}

abstractSample.go