Wieże Hanoi

Prezentacja problemu

Zwykła reprezentacja

Wieża Hanoi (zwana także Wieżą Brahmy lub Wieżą Lucasa, a czasem mnożona jako Wieże) to gra matematyczna lub łamigłówka. Składa się z trzech prętów i kilku dysków o różnych rozmiarach, które można wsunąć na dowolny słupek. Układanka zaczyna się od dysków w uporządkowanym stosie w rosnącej kolejności wielkości na jednym słupie, najmniejszym u góry, tworząc w ten sposób stożkowy kształt.

Celem układanki jest przeniesienie całego stosu na inny słupek, przestrzegając następujących prostych zasad:

1.Jednocześnie można przenosić tylko jeden dysk.

2.Każdy ruch polega na wzięciu górnego dysku z jednego ze stosów i umieszczeniu go na

innym stosie lub na pustym słupku.

3.Nie można umieszczać większego dysku na mniejszym dysku.

Z 3 dyskami łamigłówkę można rozwiązać w 7 ruchach. Minimalna liczba ruchów wymagana do rozwiązania zagadki Wieża Hanoi wynosi 2n - 1, gdzie n to liczba dysków.

Prezentacja problemu z czterema słupkami i więcej

Wygląda to tak samo jak zwykłe wieże hanoi tylko z zamiast trzech słupków mamy cztery albo więcej i N dysków leżących rosnąco na jednym z słupków; problemem jest

przeniesienie wszystkich tych dysków na inny słupek, przesuwając tylko jeden dysk na raz,

i nigdy nie wkładając dysk na mniejszy. Dudeney stwierdza również, że to

należy wykonać minimalną liczbą ruchów i zapytać, jaka jest ta liczba.

Rozwiązanie dla 3 słupków

Rozwiązanie rekurencyjne

Algorytm rekurencyjny składa się z następujących kroków:

1. przenieś (rekurencyjnie) n-1krążków ze słupka A na słupek B posługując się słupkiem C,
2. przenieś jeden krążek ze słupka A na słupek C,
3. przenieś (rekurencyjnie) n-1krążków ze słupka B na słupek C posługując się słupkiem A.

(2^(n/(k-2))-1) ilość przenoszeń krążków.

### Rozwiązanie iteracyjne

### Algorytm iteracyjny składa się z następujących kroków:

1. przenieś najmniejszy krążek na kolejny (\*) słupek,
2. wykonaj jedyny możliwy do wykonania ruch, nie zmieniając położenia krążka najmniejszego,
3. powtarzaj punkty 1 i 2, aż do odpowiedniego ułożenia wszystkich krążków.

(\*) Kolejny słupek wyznaczamy w zależności od liczby krążków. Jeśli liczba krążków jest parzysta, kolejnym słupkiem jest ten po prawej stronie (gdy dojdziemy do słupka C w następnym ruchu używamy słupka A). Natomiast jeśli liczba krążków jest nieparzysta, kolejnym słupkiem jest ten po lewej stronie (gdy dojdziemy do słupka A w następnym ruchu używamy słupka C).

Kilka znanych dopiero rozwiązań dla n ilości słupków

Chociaż wersja z trzema słupkami ma proste, rekurencyjne rozwiązanie, od dawna znane, optymalne rozwiązanie problemu Wieży Hanoi z czterema słupkami (zwanych puzzle Reve) zostało zweryfikowane dopiero w 2014 r. przez Bouscha.

Jednak w przypadku czterech lub więcej kołków algorytm Frame – Stewart, odkryty niezależnie przez Frame i Stewart, jest znany bez dowodu optymalności od 1941 r.

Optymalność tej (i innych) metod podała Demontis dopiero w 2018 r.

Algorytm Frame – Stewart

1.Niech n będzie liczbą dysków.

2.Niech r będzie liczbą kołków.

3.Zdefiniuj T (n, r) jako minimalną liczbę ruchów wymaganych do przeniesienia n dysków za pomocą r słupków.

Algorytm można opisać rekurencyjnie:

1.W przypadku niektórych k, 1 <= k < n, przenieś górne k dysków na jeden słupek inny niż słupek początkowy lub docelowy, biorąc T (k, r) kroków.

2.Bez zakłócania słupka, który teraz zawiera najwyższe dyski k, przenieś pozostałe dyski n - k do słupka docelowego, używając tylko pozostałych kołków r-1, biorąc T(n-k, r-1) kroków.

3.Na koniec przenieś górne dyski k do kołka docelowego, wykonując T (k, r) kroków.

Cały proces wymaga ruchów 2T (k, r) + T (n-k, r-1). Dlatego należy wybrać liczbę k, dla której ta ilość jest minimalna. W przypadku 4 kołków optymalny k jest równy n - {\ left \ floor} pierwiastka kwadratowego {2n + 1} { \ right \ rceil} + 1. Na przykład, dla 15 dysków i 4 kołków będzie 129 kroków, które uzyskuje się dla powyższej wartości k.

Ten algorytm (z powyższym wyborem dla k) zakłada się, że jest optymalny dla dowolnej liczby kołków; jego liczba ruchów wynosi 2^(Θ (n^(1 / (r-2)))) (dla ustalonego r).

Zaproponowany przeze mnie algorytm (C++)

#include<iostream>

#include<stdexcept>

using namespace std;

void Haanoi(int nDisk, int from, int to, int buf, int &step) //nDisk llosc dysków, from

{ początkowy słupek, to docelowy

if (nDisk!=0) słupek, buf pomocniczy słupek,

{ step licznik kroków

Haanoi(nDisk-1, from, buf, to, step);

++step; cout << from << " -> " << to << endl;

Haanoi(nDisk-1, buf, to, from, step);

}

}

int HHanoi(int nDisk, int kPeg) // nDisk ilość dysków, kPeg ilość słupków

{

//sprawdzianie dannych na wyjątki

if(nDisk>0 && kPeg==0) throw invalid\_argument( "received negative value" );

if(nDisk>1 && kPeg==1) throw invalid\_argument( "received negative value" );

if(nDisk>1 && kPeg==2) throw invalid\_argument( "received negative value" );

if(nDisk==0 || kPeg==1) return 0;

if(kPeg==2) {cout << 1 << " -> " << 2 << endl; return 1;}

int step=0; // licznik kroków

int n=nDisk, l, m, j=2\*(kPeg-2)-1; // j ilość wywołań funkcji Haanoi

if(nDisk<=kPeg-2)

{

for(int i=1; i<nDisk; i++)

{Haanoi(1, 1, i+1, i+2, step);}

Haanoi(1, 1, kPeg, kPeg-1, step);

for(int i=nDisk; i>1; i--)

{Haanoi(1, i, kPeg, 1, step);}

}

else

{

while(n%(kPeg-2)!=0) n--; // wyszukiwanie wielokrotności liczby n

l=n/(kPeg-2); // l ilość krążków w stopkach

m=nDisk-(kPeg-3)\*l; // m ilość krążków w najniższej stopce

for(int i=1; i<=j/2; i++) // przesuwanie powyższych stopek

{Haanoi(l, 1, i+1, i+2, step);} //krążków na pośrednie słupki

Haanoi(m, 1, kPeg, kPeg-1, step);

for(int i=2; i<=j/2+1; i++) // przesuwanie stopek z

{Haanoi(l, kPeg-i, kPeg, 1, step);} //pośrednich słupków na

//docelowy

}

return step;

}

int main()

{

int n=-1, k=-1;

while(n<0 || k<0)

{

cout << “Liczba dysków: “; cin >> n;

cout << “Liczba słupków: “; cin >> k;

}

cout << “liczba kroków: “ << HHanoi(n, k);

return 0;

}

Opis algorytmu

n - liczba krążków; k - liczba słupków

Najpierw sprawdzimy k i n na wyjątki.

Jeżeli **n>k**, to:

n dzielimy na k -2, 2 zostawiamy dla przeniesienia (jako pomocniczy)

Jeżeli n nie jest wielokrotnością k-2 odejmujemy od n resztę dzielenia.

Wyższą kupkę przenosimy na kolejny słupek, korzystając z 3 słupka jako pomocniczego, za pomocą f-cji Haanoi; kolejną kupkę przenosimy na 3 słupek, korzystając z 4 jako pomocniczego… i t.d.

Najniższą kupkę przenosimy na celowy słupek, przedostatni słupek pomocniczy.

Przenosimy w odwrotnej do początkowej kolejności wszyscy pozostałe kupki na celowy słupek.

Jeżeli **n<=k**, to ilość wywołań Haanoi będzie inna. Ponieważ nie zastosujemy wszystkich słupków; przenosimy po jednemu n-1 razy na osobne słupki, największy krążek na docelowy słupek, i w odwrotnej kolejności n-1 pozostałych krążków.

Liczba kroków dla k słupków i n krążków

Dla n<=k-2

2(n-1)+1

(n-1) ilość wywołań f-cji Haanoi w pętlach for

\*każde wywołanie Haanoi robie 1 krok. Czyli sumujemy tylko ilość wejść do Haanoi.

Dla n>k

2((2(k-2)-1)/2)2^(N/(k-2))+2^(n-(k-3)(N/(k-2))), gdzie N=n-n%(k-2)

2((2(k-2)-1)/2) ilość wywołań f-cji Haanoi w pętlach for

2^(N/(k-2)) ilość kroków f-cji Haanoi w pętlach for

2^(n-(k-3)(N/(k-2))) ilość kroków f-cji Haanoi w jej wywołaniu mięzy pętlami for

Ilość kroków dla różnej ilości dysków i słupków

n: 1 k: 3 step: 1

n: 2 k: 3 step: 3

n: 3 k: 3 step: 7

n: 4 k: 3 step: 15

n: 5 k: 3 step: 31

n: 6 k: 3 step: 63

n: 7 k: 3 step: 127

n: 8 k: 3 step: 255

n: 9 k: 3 step: 511

n: 10 k: 3 step: 1023

n: 1 k: 4 step: 1

n: 2 k: 4 step: 3

n: 3 k: 4 step: 5

n: 4 k: 4 step: 9

n: 5 k: 4 step: 13

n: 6 k: 4 step: 21

n: 7 k: 4 step: 29

n: 8 k: 4 step: 45

n: 9 k: 4 step: 61

n: 10 k: 4 step: 93

n: 1 k: 5 step: 1

n: 2 k: 5 step: 3

n: 3 k: 5 step: 5

n: 4 k: 5 step: 7

n: 5 k: 5 step: 11

n: 6 k: 5 step: 15

n: 7 k: 5 step: 19

n: 8 k: 5 step: 27

n: 9 k: 5 step: 35

n: 10 k: 5 step: 43

n: 1 k: 6 step: 1

n: 2 k: 6 step: 3

n: 3 k: 6 step: 5

n: 4 k: 6 step: 7

n: 5 k: 6 step: 9

n: 6 k: 6 step: 13

n: 7 k: 6 step: 21

n: 8 k: 6 step: 21

n: 9 k: 6 step: 25

n: 10 k: 6 step: 33

n: 1 k: 7 step: 1

n: 2 k: 7 step: 3

n: 3 k: 7 step: 5

n: 4 k: 7 step: 7

n: 5 k: 7 step: 9

n: 6 k: 7 step: 11

n: 7 k: 7 step: 15

n: 8 k: 7 step: 23

n: 9 k: 7 step: 39

n: 10 k: 7 step: 27

n: 1 k: 8 step: 1

n: 2 k: 8 step: 3

n: 3 k: 8 step: 5

n: 4 k: 8 step: 7

n: 5 k: 8 step: 9

n: 6 k: 8 step: 11

n: 7 k: 8 step: 13

n: 8 k: 8 step: 17

n: 9 k: 8 step: 25

n: 10 k: 8 step: 41

n: 1 k: 9 step: 1

n: 2 k: 9 step: 3

n: 3 k: 9 step: 5

n: 4 k: 9 step: 7

n: 5 k: 9 step: 9

n: 6 k: 9 step: 11

n: 7 k: 9 step: 13

n: 8 k: 9 step: 15

n: 9 k: 9 step: 19

n: 10 k: 9 step: 27

n: 1 k: 10 step: 1

n: 2 k: 10 step: 3

n: 3 k: 10 step: 5

n: 4 k: 10 step: 7

n: 5 k: 10 step: 9

n: 6 k: 10 step: 11

n: 7 k: 10 step: 13

n: 8 k: 10 step: 15

n: 9 k: 10 step: 17

n: 10 k: 10 step: 21