

# diverta 2019 Programming Contest 2 解説

satashun, yuma000, E869120, square1001

2019/06/15

*For International Readers: English editorial starts on page 9.*

## A: Ball Distribution

最大値と最小値の差が大きくなるように  $N$  個のボールを  $K$  人に配る問題です。

まず、 $K = 1$  人の場合は、 $N$  個を 1 人に配るしかないので差は 0 です。

以下では  $K \geq 2$  とします。

それぞれの人に 1 つも渡さなくてもよい場合は、明らかに 1 人に  $N$  個渡し、他の人には 1 つも渡さないのが最適で、差は  $N$  となります。

今回は全員に 1 個以上配る必要がありますが、先に全員に 1 個ずつ配った上で残りの  $N - K$  個を自由に配ることを考えると、答えは  $N - K$  です。

実装例を以下に示します。

---

```
1 #include <bits/stdc++.h>
2 using namespace std;
3
4 int main() {
5     int N, K; cin >> N >> K;
6     if (K == 1) {
7         cout << 0 << endl;
8     } else {
9         cout << N - K << endl;
10    }
11    return 0;
12 }
```

---

## B: Picking Up

いきなり最小値を求めるのは難しいので、まず  $p, q$  が既に指定されている場合のコストの最小値を考えてみましょう。

重要な考察として、あるボールを選ぶ時のコストを 0 にするような取り方を考えた時、異なるボールについて干渉することはありません。

もう少し丁寧に言うと、 $(a-p, b-p)$  と  $(a, b)$  の両方にボールが存在する場合に  $(a-p, b-p)$  から  $(a, b)$  へ有向辺を張ったグラフを考えます。どの頂点についても入次数と出次数は高々 1 で、直線状の連結成分が複数ある形になっています。この時、連結成分ごとに端のボールから順に拾っていくことでコスト  $N - \text{辺の数}$  が達成でき、これが最小です。

$p, q$  を決めた場合に辺の数がコストを決めることがわかったので、辺の数が最大になるように  $p, q$  を選ぶと良いことがわかります。

$p, q$  として考える必要がある組は、 $i \neq j$  について  $(x_i - x_j, y_i - y_j)$  に現れるもののみです。(そうでない場合、辺の数は 0 です)

よって  $\mathcal{O}(N^2)$  通り  $(p, q)$  の候補を試し、実際に  $i \neq j$  について  $(x_i - x_j, y_i - y_j) = (p, q)$  となる  $(i, j)$  が何組あるか求めることで、 $\mathcal{O}(N^4)$  でこの問題が解けました。実際に  $(p, q)$  として試さずとも、 $(x_i - x_j, y_i - y_j)$  で最も多いものが何個あるかを map などを用いて計算することで  $\mathcal{O}(N^2 \log N)$  にもなります。

## C: Successive Subtraction

書かれている整数を直接追うのではなく、次のように言い換えます。

初め  $A_1, A_2, \dots, A_N$  のそれぞれが要素となった集合が  $N$  個あり、要素の符号は全て  $+$  です。 $N - 1$  回、集合を 2 つ選び、2 つ目の要素の符号を全て反転して、集合を併合します。最後に残った 1 つの集合の要素の和の最大値を求めてください。

このように考えると、最後に  $+$  が  $i$  個、 $-$  が  $N - i$  個残っている時、 $A$  の大きい方から  $i$  個が  $+$ 、残りが  $-$  となるように操作を行うことが最適であることがわかります。

よって、 $+$  と  $-$  が何個ずつある状態にすることができるかが問題ですが、実は  $+$  のみまたは  $-$  のみしかない状態以外は実現可能です。要素数 2 のものは、 $A_i - A_j$  の形しかないので符号が異なります。要素数  $n \geq 3$  の集合を得るとき、最後に要素数  $k$  の集合と  $n - k$  の集合を併合しますが、 $k$  または  $n - k$  が 2 以上でありどちらかの集合に両方の符号が存在するので、併合した集合にも両方の符号が存在します。

そうでない場合、例えば以下のようにして構成することができます。 $A$  の要素をソートしておいて、0 以上の要素の個数を  $P$  個、負の要素の個数を  $Q$  個とします。(全ての符号を等しくすることはできないので、 $P = 0$  のときは  $P = 1, Q = N - 1, Q = 0$  のときは  $P = N - 1, Q = 1$  とします)

$A_1$  を  $x$  として選び続き、 $A_{Q+1}, A_{Q+2}, \dots, A_{N-1}$  を  $y$  として選びます。次に、 $A_N$  から  $A_1, A_2, \dots, A_Q$  を引けば良いです。

## D. Squirrel Merchant(writer : yuma000)

ドングリを失って金属を得ることを「金属を買う」、ドングリを得て金属を失うことを「金属を売る」と表現することになります。まず、直大君の行動ですが、次のように言い換えられます。

1.  $N$  個のドングリを持って巣から出る。
2. 取引所  $A$  で、金属を買い、それら全てを取引所  $B$  で売る。
3. 取引所  $B$  で、金属を買い、それら全てを取引所  $A$  で売る。
4. 巣に戻る。

なぜならば、取引所で金属を売った後にそれを同じ取引所ですぐに買い戻してもドングリの数は変わらないからです。また明らかに、2. と 3. では最大数のドングリが残るような行動を取ればよいです。

まず、2. について考えます。これは、

- 最大容量を  $N$
- 品物の重みを (それぞれの金属 1 グラムの取引所  $A$  での値段)
- 品物の価値を (それぞれの金属 1 グラムの取引所  $B$  での値段) - (それぞれの金属 1 グラムの取引所  $B$  での値段)

としたナップサック問題を解けばよいです。

これによって得られたドングリの最大数を  $M(\leq N * \max(g_2, s_2, b_2))$  として、3. についても同様に、

- 最大容量を  $M$
- 品物の重みを (それぞれの金属 1 グラムの取引所  $B$  での値段)
- 品物の価値を (それぞれの金属 1 グラムの取引所  $A$  での値段) - (それぞれの金属 1 グラムの取引所  $A$  での値段)

としたナップサック問題を解けばよいです。

$a$  が品物の個数、 $b$  がナップサックの容量とした時、動的計画法を使えば計算量  $O(ab)$  で解けるので、この問題は計算量  $O(M) = O(N * \max(g_2, s_2, b_2))$  で解けます。

### おまけ

二回目のナップサック問題の解き方を工夫することで  $O(N + (\max(g_2, s_2, b_2))^2)$  で解くこともできます。

## E: Balanced Piles

まず、このままでは考えにくいのでマスを手前から見たとして次のように問題を言い換えます。

1 列にマスが並んでおり、初めマス 0 に  $1, 2, \dots, N$  の駒がある。最も左の駒を 1 つ選んで、最も右の駒から  $D$  以内まで右に動かす。全ての駒がマス  $H$  にあるように動かす方法は何通りあるか。

駒が 1 回でも置かれるマスの個数を固定すると、隣同士のマス同士の距離が  $D$  以下であればよく、それらを全て通る動かし方の個数は同じであることがわかります。実はこのような動かし方は、0 と  $H$  以外に  $k$  マス使う時  $N! * (\sum_{i=1}^N i!)^k$  通りであることがわかります。

0 マスの時:  $N$  個の駒を順番に動かすので  $N!$  通り

$k+1$  マスの時:  $k$  マスの時の経路を任意に 1 つ取り、初めマス 0 にある駒を順にマス  $1 \leq a_1 \leq a_2, \dots, \leq a_N$  に動かしたとします。後ろから順に  $i$  個をマス 0 から来たときのみそれ以外はマス  $-1$  から直接来たと考えると  $k+1$  マスの時の経路に対応し、 $k$  マスの時の動かし方の  $\sum_{i=1}^N i!$  倍あります。

よって  $dp[i]$  := マス  $i$  までを考慮してマス  $i$  に止まる駒があるときの場合の数 とすると、累積和を用いて高速化することができ、時間計算量は  $O(N + H)$  でこの問題が解けました。

## F: Diverta City

この問題の解法にはいくつかあるかもしれませんが、そのうちの 1 つを説明していきたいと思います。この解法だと、 $N = 10$  で最大のハミルトン経路の長さが 96,755,758,040 になる答えあるいは、少し工夫すると 83,907,014,282 の答えが求まります。

### ★ STEP 1: 最初のステップ - 2 進数を利用して「復元可能に」する

ハミルトン経路の長さを全部異なる値にするというよりかは、「ハミルトン経路の長さが入力されたときに、ハミルトン経路を復元できるような道路の長さの組み合わせ」を作ることを考えます。

道路は  $\frac{N(N-1)}{2}$  個あります。それらの長さを、1, 2, 4, 8, 16, ... と 2 倍ずつにしていくと、2 進数を用いて答えを復元することができます。例えば、「長さ 52 のハミルトン経路」の場合、 $4 + 16 + 32 = 52$  なので、長さ 4, 16, 32 の道路を使っていると分かります。

その場合、ハミルトン経路の長さの最大はおおよそ  $2^{\frac{N(N-1)}{2}}$  になります。つまり、 $N = 10$  だと  $3 \times 10^{13}$  くらいです。

### ★ STEP 2: ハミルトン経路の性質を利用しよう!

ハミルトン経路では、それぞれの頂点は多くても 2 個の辺にしか使われません。これがこの問題を解くキーポイントになります!

そこで、道路を  $K$  色に塗ることを考えます。ただし、各色の道路だけを取り出したときに、「どの道路にも接続している街」があるようにします。別の言葉で言うと、道路の形はウニのようになっています。(グラフ理論の言葉で言うと「スターグラフ」)

スターグラフに分けて考えると、実はこの問題はかなり効率的な答えを求めることができます。

### ★ STEP 3: スターグラフの場合に効率的に解く

スターグラフに含まれる道路の数を  $L$  個とし、道路の長さを  $a_1, a_2, \dots, a_L$  とします。3 本以上の道路を通ることはないので、 $a_i (1 \leq i \leq L)$  と  $a_i + a_j (1 \leq i < j \leq L)$  がすべて異なるような道路の長さの組み合わせを見つければよいです。

当然、1, 2, 4, 8, 16, ...,  $2^{L-1}$  という答えは条件を満たします。ただ、もっと効率的な解を簡単な方法で求めることができます。ここでは、「貪欲法」を使ってほぼ最適な解を求める方法を説明します。次のようなアルゴリズムです。

- $a$  を最初空の数列とする
- 以下の操作を  $n$  回繰り返す
- 値  $x$  を数列  $a$  に追加しても条件を満たすような、最小の整数  $x$  を選び、数列  $a$  の末尾に追加する

この数列は、1, 2, 4, 7, 12, 20, 29, 38, 52, 73, 94, 127, 151, ... と比較的ゆっくりペースで増えていきます。そうすると、2 進数を用いた解法よりも効率的な答えが求まります。

### ★ STEP 4: スターグラフの部分問題を「まとめ上げる」

STEP 2 で、スターグラフになるように道路を色で塗り分けることを考えました。 $i$  色目の道路の長さを

$a_1, a_2, \dots, a_L$  とすると、ハミルトン経路上の  $i$  色目の道路の長さとしてありうるものは  $0, a_j, a_j + a_k$  のみです。このうち最大の値を  $S_i$  として、 $i$  色目の道路の長さ  $D_i$  は  $0 \leq D_i < S_i + 1$  を満たすことになります。

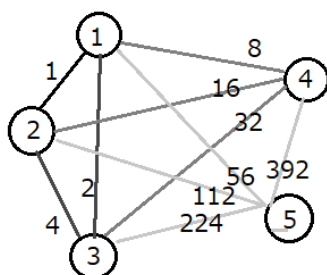
この条件だけを見ると、数列  $D$  は高々  $(S_1 + 1) \times (S_2 + 1) \times (S_3 + 1) \times \dots \times (S_K + 1)$  通りになり、 $i$  色目の道路の長さを「スターグラフについて計算する時点での長さから  $(S_1 + 1) \times (S_2 + 1) \times \dots \times (S_{i-1} + 1)$  倍」することによって、復元可能になります (アイデアは先ほどの 2 進数と似ています)

さて、実際に  $N = 5$  で試してみましよう。色 1 で道路 1-2 を塗り、色 2 で道路 1-3, 2-3 を塗り、色 3 で道路 1-4, 2-4, 3-4 を塗り、色 4 で道路 1-5, 2-5, 3-5, 4-5 を塗ることを考えます。(そのように、 $N - 1, \dots, 2, 1$  本の道路になるように塗り分けます) そのとき、次のようになります。

色の種類	スターグラフの長さ	道路の長さ	$S_i$
色 1	1	1	1
色 2	1, 2	2, 4	3
色 3	1, 2, 4	8, 16, 32	6
色 4	1, 2, 4, 7	56, 112, 224, 392	11

よって、 $N = 5$  のとき最大の辺の長さが 392 であるような答えを見つけることができました。 $N = 10$  のときは、最大のハミルトン経路の長さは 136, 369, 260, 626 になりますので、制限の  $10^{11}$  まで少しだけ改善が必要です。

なお、 $N = 5$  の場合のグラフの例は以下のようになります。



★ STEP 5: 定数倍改善のための「貪欲法」

STEP 4 はよく見ると「街の数  $N - 1$  の場合で解いて、そこから新たに街  $N$  を作って  $N - 1$  個の道路の長さを決める」というようになっています。実は、街の数  $N - 1$  の場合のハミルトン経路の長さの合計の最大が  $M$  のとき、新たな道路は  $(M + 1) \times 1, (M + 1) \times 2, (M + 1) \times 4, (M + 1) \times 7, (M + 1) \times 12, \dots$  としても復元可能なのです。当然、STEP 4 のアルゴリズムよりは効率的、すなわち  $M \leq (S_1 + 1) \times (S_2 + 1) \times \dots \times (S_{N-1} + 1)$  が成り立ちます。このようにして、再帰的に解くことができるのです。

$M$  を探すのにかかる計算量は  $O((N + 1)!)$  ですので、答えを求めるのにかかる計算量は  $O((N + 1)!)$  であり、実行時間制限に余裕で間に合います。また、このアルゴリズムで求めた  $N = 10$  のときの答えの最大のハミルトン経路の長さは 96, 755, 758, 040 になり、正解できます。

★ STEP 6: さらなる改善

みなさんもお分かりだと思いますが、先ほどの 96, 755, 758, 040 という最大のハミルトン経路の長さは最適

ではありません。さらに良い道路の長さの組み合わせがあります。

例えば、先ほどのアルゴリズムだと、 $N = 3$  のときの辺の長さが  $1, 2, 4$  となっています。これを  $1, 2, 3$  にあらかじめ決めておくと、これ以降の道路の長さも短くなります。「 $1 + 2$  と  $3$  が同じなので、区別できなくなるのではないか？」と思うかもしれませんが、それ以外の  $\frac{N(N-1)}{2} - 3$  個の辺がすべて復元できるので、ハミルトン経路に含まれる辺の個数がちょうど  $N - 1$  であることを利用して最後は区別できます。そうすると、最大のハミルトン経路の長さ  $83,907,014,282$  が達成できます。



# diverta 2019 Programming Contest 2 Editorial

satashun, yuma000, E869120, square1001

2019/06/15

## A: Ball Distribution

If there is only  $K = 1$  person, we have no choice but to give all balls to that person, so the difference is 0.

Otherwise, the difference will be maximized when we first give one ball to each person, then give the remaining  $N - K$  balls to one person, so the maximum possible difference is  $N - K$ .

Sample implementation in C++ follows:

---

```
1 #include <bits/stdc++.h>
2 using namespace std;
3
4 int main() {
5     int N, K; cin >> N >> K;
6     if (K == 1) {
7         cout << 0 << endl;
8     } else {
9         cout << N - K << endl;
10    }
11    return 0;
12 }
```

---

## B: Picking Up

First, let us consider the minimum total cost when the values of  $p$  and  $q$  are already specified.

Consider a directed graph where the vertices are the balls and there is a directed edge from a dot  $(a - p, b - p)$  to a dot  $(a, b)$ . The indegrees and outdegrees of the vertices are at most one, and each connected components in the graph is a linear chain of vertices. By picking the balls from the starting end in each connected component, we can achieve the minimum possible total cost:  $N$  - (the number of edges).

Now, our objective is to choose  $p$  and  $q$  so that the number of edges in the graph will be maximized. The only choices of  $(p, q)$  worth considering are the ones that occur as  $(x_i - x_j, y_i - y_j)$  for some  $i$  and  $j$  ( $i \neq j$ ) (for other choices of  $(p, q)$ , the number of edges will be 0). Thus, we can try all  $\mathcal{O}(N^2)$  candidates and count the number of pairs  $(i, j)$  ( $i \neq j$ ) such that  $(x_i - x_j, y_i - y_j) = (p, q)$  for each candidate to obtain an  $\mathcal{O}(N^4)$  solution. (Actually, we do not need to count the pairs  $(i, j)$  for each choice of  $(p, q)$ , and instead we can just examine the distribution of  $(x_i - x_j, y_i - y_j)$  and choose the most frequent value, in  $\mathcal{O}(N^2 \log N)$  time with `map` in C++ or similar data structure.

## C: Successive Subtraction

We will rephrase the problem as follows:

*We have  $N$  multisets. The  $i$ -th multiset has one element  $A_i$ , with a sign  $+$  attached. We will perform the following operation  $N - 1$  times: choose two multisets, invert the sign of each element in the second multiset and merge them. Find the maximum possible sum of the elements in the final multiset remaining.*

We can see that, if the final multiset contains  $i$  elements with  $+$  and  $N - i$  elements with  $-$ , it is optimal to perform operations so that the  $i$  greatest numbers among  $A_1, A_2, \dots, A_N$  get  $+$ , and the remaining numbers get  $-$ .

The problem now is how many numbers can get  $+$  and how many can get  $-$ . The conclusion is: almost all states are achievable, with two exceptions where all numbers get  $+$  and where all numbers get  $-$ . Let us prove it. A multiset of size 2 during the operations always has the form  $A_i - A_j$ , where one of the two numbers gets  $+$  and the other gets  $-$ . For a multiset of size  $n \geq 3$  during the operations, we must have merged two multisets of size  $k$  and  $n - k$  to obtain it. Since at least one of  $k$  and  $n - k$  is 2 or greater, at least one of the two multisets contained both a number with  $+$  and a number with  $-$ , so the merged set also contains both.

Thus, if  $A$  contains both non-negative numbers and negative numbers, we can give  $+$  to all the non-negative numbers and  $-$  to all the negative numbers. If all the numbers in  $A$  are non-negative (negative), we should compromise and give  $- (+)$  to a minimum (maximum) number and  $+$  ( $-$ ) to the others.

This can be implemented as follows. First, we sort the sequence  $A$  in ascending order. Assume there are  $P$  non-negative numbers and  $Q$  negative numbers. (If  $P = 0$ , consider the greatest number as non-negative and assume  $P = 1$ . If  $Q = 0$ , similarly assume  $Q = 1$ ). We subtract  $A_{Q+1}, A_{Q+2}, \dots, A_{N-1}$  from  $A_1$ , then subtract the result of this and  $A_2, \dots, A_Q$  from  $A_N$ .

## D. Squirrel Merchant(writer : yuma000)

Let us say we “buy” metals when we trade acorns for metals, and “sell” metals when we trade metals for acorns. Chokudai’s plan is equivalent to the following:

1. Get out of the nest with  $N$  acorns in his hands.
2. Buy metals in Exchange  $A$  and sell all of them in Exchange  $B$ .
3. Buy metals in Exchange  $B$  and sell all of them in Exchange  $A$ .
4. Go back to the nest.

This is because we can immediately buy back metals that we sold without loss of acorns. Obviously, we should end each of the steps 2. and 3. with the maximum possible number of acorns.

Let us first consider step 2. This is equivalent to the knapsack problem, where

- The maximum weight capacity is  $N$
- The weight of an item (each kind of metal) is (the price of the metal per gram in Exchange  $A$ )
- The value of an item is (the price of the metal per gram in Exchange  $B$ ) – (the price of the metal per gram in Exchange  $A$ )

Let  $M$  be the answer to this problem, which is at most  $N * \max(g_2, s_2, b_2)$ . Step 3. is again equivalent to the knapsack problem, where

- The maximum weight capacity is  $M$
- The weight of an item (each kind of metal) is (the price of the metal per gram in Exchange  $B$ )
- The value of an item is (the price of the metal per gram in Exchange  $A$ ) – (the price of the metal per gram in Exchange  $B$ )

Each of these knapsack problems can be solved by dynamic programming with time complexity  $O(ab)$ , where  $a$  is the number of items and  $b$  is the maximum weight capacity. Thus, we solved the problem with time complexity  $O(M) = O(N * \max(g_2, s_2, b_2))$ .

### Bonus

We can also solve the problem in  $O(N + (\max(g_2, s_2, b_2))^2)$  time, by solving the second knapsack problem more efficiently.

## E: Balanced Piles

First, consider the case  $D = 1$ .

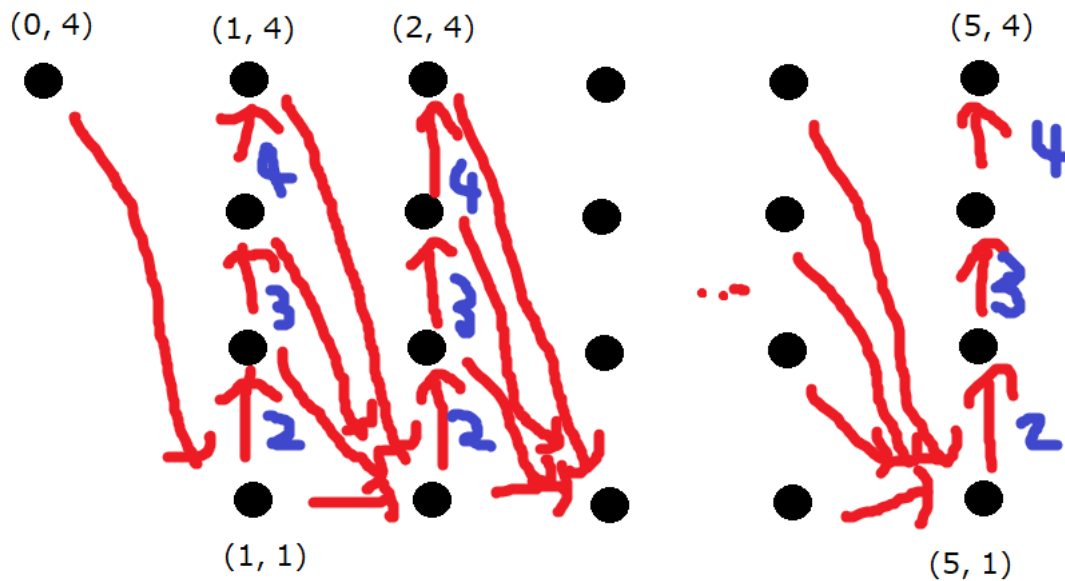
Let  $dp[x][y]$  be the number of ways to add blocks, when the current status is the following:

- Currently, the height of the highest pile is  $x$ , and there are  $y$  piles of height  $x$ . - Additionally, for each set of piles of the same height, the "order" is predefined. For example, if there are two piles of height  $h$ , when those two piles become the shortest pile, there is an additional constraint and we must choose a specific pile in the next operation.

How to compute this  $dp$  array?

- Clearly, if  $(x, y) = (H, N)$ ,  $dp[x][y] = 1$ . - Otherwise, from  $dp[x][y]$ , there are one transition to  $dp[x + 1][1]$  unless  $x = H$  (change the height of the pile we choose next to  $x + 1$ ), and there are  $y + 1$  transitions to  $dp[x][y + 1]$  unless  $y = N$  (change the height of the pile we choose next to  $x$ ; in this case there are  $y + 1$  ways to "insert" the new pile to the order of  $y$  piles).

Therefore, the answer  $dp[0][N]$  corresponds to the number of paths in the following graph:



(when  $N = 4, D = 5$ )

Of course,  $dp[0][N]$  has an extra factor of  $N!$  because in the original problem the "order" of  $N$  piles

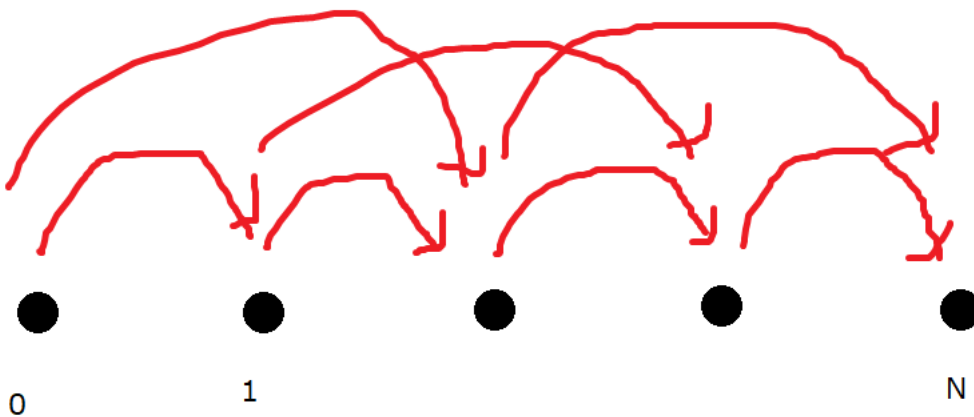
is not defined. However, we also have an extra "order" in  $dp[H][N]$ , and these two factors cancel each other and we can ignore them.

Thus, the answer is  $dp[0][N] = (1! + 2! + \dots + N!)^{(H-1)}N!$  when  $D = 1$ .


How to solve the general case?

Let  $0 = h_0 < h_1 < \dots < h_K = H$  be an array that satisfies  $h_{i+1} - h_i \leq D$  for each  $i$ . Then, the answer to the original problem with an additional constraint that "the set of heights that appear during the operations must be  $\{h_0, \dots, h_K\}$ " is  $(1! + 2! + \dots + N!)^{(K-1)}N!$ .

Thus, the answer to the original problem is the number of paths from 0 to  $N$  in the following graph, times  $N!/(1! + 2! + \dots + N!)$ :



weight of each  is  $1! + 2! + \dots + N!$

there is a  from  $i$  to  $j$  iff  $i < j$  and  $j - i \leq D$

This can be computed in linear time.

## F: Diverta City

We will show a way to inductively construct a solution with the maximum total length of a Hamiltonian path being 96,755,758,040 for the case  $N = 10$ .

If  $N = 2$ , we connect the two towns with a road of length 1 and we have a solution. Now, assume that we have a solution  $G$  for the case  $N = i$  ( $i \geq 2$ ), and we will construct a solution for the case  $N = i + 1$ .

Let  $M$  be the maximum total length of a Hamiltonian path in  $G$ . Add a new town  $i + 1$  to  $G$ , and connect Town  $j$  ( $1 \leq j \leq i$ ) and Town  $i + 1$  with a road of length  $(M + 1) \times a_j$ , where  $a = \{1, 2, 4, 7, 12, 20, 29, 38, 52, 73\}$ . The elements of  $a$  are chosen so that the values  $a_i$  ( $1 \leq i \leq 10$ ) and  $a_i + a_j$  ( $1 \leq i < j \leq 10$ ) are all distinct. This makes the lengths of all Hamiltonian paths with  $i + 1$  towns distinct, since any Hamiltonian path traverses only one or two of the edges added. The time complexity of this approach is  $O((N + 1)!)$ .