Task 1: LunchBox

Author: Ken Sung

Problem

- Input:
  - N lunch boxes,
  - m schools asking for k_1, k_2, …, k_m lunch boxes.
- Rule: We give either 0 or k_i lunch boxes to the i^{th} school.
- Output: maximum number of schools that can receive lunch boxes.

N=4

Maximum No. of schools: 2

Solution

- Greedy algorithm: Distribute the lunch boxes to the school requesting smaller number of lunch boxes first.
  - Sort the schools from small to big. O(m log m) time
  - Distribute the lunch box O(N) time
  - Report the number of schools

N=4

Maximum No. of schools: 2

Task 2: Fabric

Author: Mark Theng

Problem

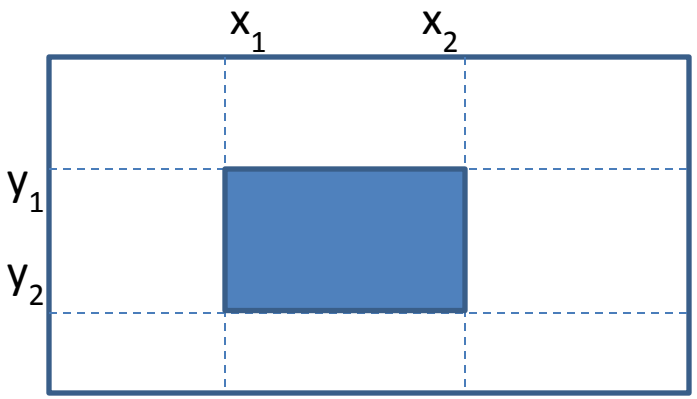
- Input:
  - A N×M cloth with some hole (hole is represented by 1)
  - An area threshold K
- Output:
  - The number of ways to cut a rectangle such that the area is at least K and there is no hole

- Example: 2×4 cloth, K=3
  - Two rectangles of size 3
  - One rectangle of size 4
  - Total: 3 rectangles of size 3!

Subtask 1

- Input: a N×M cloth with no hole, K=1.

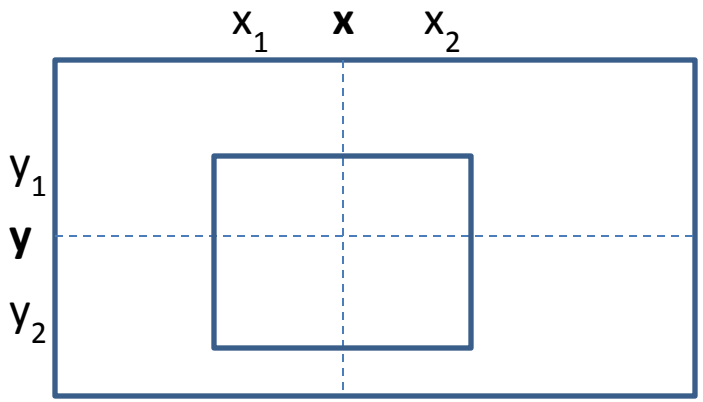
- Hence, any rectangle is correct.
- The total number of rectangles is $\binom{N+1}{2}\binom{M+1}{2}$.



Subtask 2

- Input: a N×M cloth with one hole at (x,y), K=1.

- The total number of rectangles is $\binom{N+1}{2}\binom{M+1}{2}$.
- The number of rectangles covering (x,y) is $x(M-x+1)y(N-y+1)$.
- Hence, the answer is $\binom{N+1}{2}\binom{M+1}{2} - x(M-x+1)y(N-y+1)$.



Subtask 3

- Input requirement: $0<N,M\leq 50$

- C=0;
- For every $1 \leq x_1 \leq x_2 \leq N$,
  - For every $1 \leq y_1 \leq y_2 \leq M$,
    - Let $S = \sum_{i=x_1}^{x_2} \sum_{j=y_1}^{y_2} s_{i,j}$;
  - If (S==0 && $(x_2-x_1+1)*(y_2-y_1+1)\geq K$) C++;
- Report C;

$O(N^3M^3)$ time.

				y_1				y_2		
	1	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
x_1	0	1	0	0	1	0	0	1	0	0
	0	0	0	0	0	0	0	1	0	0
x_2	0	0	1	0	0	0	0	0	0	1
	0	0	0	0	0	0	0	0	0	0

Subtask 4

- Input requirement: $0<N,M\leq 500$

- C=0;
- For x_1=1 to N,
  - Let A[y]=0 for y=1, …, M;
  - For x_2=x_1 to N,
    - Let $A[y] = A[y] + s_{x_2,y}$ for y=1,…,M;
    - Let z_1, z_2, …, z_q be the length of consecutive zeroes in A[1..M];
    - $\Delta = \left\lceil \frac{K}{x_2 - x_1 + 1} \right\rceil$; $C = C + \binom{z_1 + 2 - \Delta}{2} + \cdots + \binom{z_q + 2 - \Delta}{2}$;
- Report C;

$O(N^2M)$ time.

	1	0	1	0	0	0	0	0	0	0
x_1	0	1	0	0	1	0	0	1	0	0
	0	0	0	0	0	0	0	1	0	0
x_2	0	0	1	0	0	0	0	0	0	1
	0	0	0	0	0	0	0	0	0	0

A	0	1	1	0	1	0	0	2	0	1

$\binom{1+2-1}{2}=1$ $\binom{1+2-1}{2}=1$ $\binom{2+2-1}{2}=3$ $\binom{1+2-1}{2}=1$

$K = 2$

$\Delta = 1$

Subtask 5

- Input requirement: $0<N,M\leq 2000$ and $K=1$.

- Denote $T_{x,y}$ be the number of rectangles whose lowest right corner is (x,y).
- The blue cells (called limiters) are the first '1' just above the y^{th} row.
- The staircase shape green region indicates the feasible region for all rectangles whose lowest right corner is (x,y).
- $T_{x,y}$ equals the number of green cells.

	1	2	3	4	5	6	7	8	9	10
1	1	0	1	0	0	1	0	0	0	0
2	0	1	0	0	1	0	0	1	0	0
3	0	0	0	0	0	0	0	1	0	0
4	0	0	1	0	0	0	0	0	0	1
5	0	0	0	0	0	0	0	0	0	0

The shape for (7,5) = $(3,4)^3$, $(5,2)^6$, $(6,1)^4$, $(7,0)^5$

- For example, $T_{5,7} = 3+6+4+5 = 18$.

Subtask 5

- Given the shape of (x,y), we can compute the shape of (x+1,y).
- Each step must remove $O(N)$ limiters and add one to the list:

	1	2	3	4	5	6	7	8	9	10
1	1	0	1	0	0	1	0	0	0	0
2	0	1	0	0	1	0	0	1	0	0
3	0	0	0	0	0	0	0	1	0	0
4	0	0	1	0	0	0	0	0	0	1
5	0	0	0	0	0	0	0	0	0	0

The shape for (7,5) = $(3,4)^3$, $(5,2)^6$, $(6,1)^4$, $(7,0)^5$

$T_{7,5}$=3+6+4+5=18

Remove $(5,2)^6$, $(6,1)^4$, $(7,0)^5$
Insert $(8,3)^{10}$

	1	2	3	4	5	6	7	8	9	10
1	1	0	1	0	0	1	0	0	0	0
2	0	1	0	0	1	0	0	1	0	0
3	0	0	0	0	0	0	0	1	0	0
4	0	0	1	0	0	0	0	0	0	1
5	0	0	0	0	0	0	0	0	0	0

The shape for (8,5) = $(4,3)^3$, $(8,3)^{10}$

$T_{8,5}$= 3+10=13

Subtask 5

- C=0;
- For y = 1 to M
  - Update the limiters L[i] (the first '1' just above the y^{th} row) in O(N) time;
  - Set S be empty stack;
  - For x = 1 to N
    - Update the shape S by removing a list of limiters (j, L[j]) such that L[j] <L[x] and insert (x, L[x]) into S;
    - Compute $T_{x,y}$ from $T_{x-1,y}$;
    - C = C + $T_{x,y}$;
- Report C;

- For the y^{th} row, the time required to update the shape S and compute $T_{x,y}$ is O (N). There are M rows. The time complexity is O(NM).

Subtask 6

- Input requirement: 0<N,M≤2000.

- We use the same algorithm in subtask 5.
- Moreover, we need to mask all rectangles of size at most K.
- For example, for (7,5), if K=5, we need to mask 10 entries (gary entries).
- We don't describe the detail due to time limit.

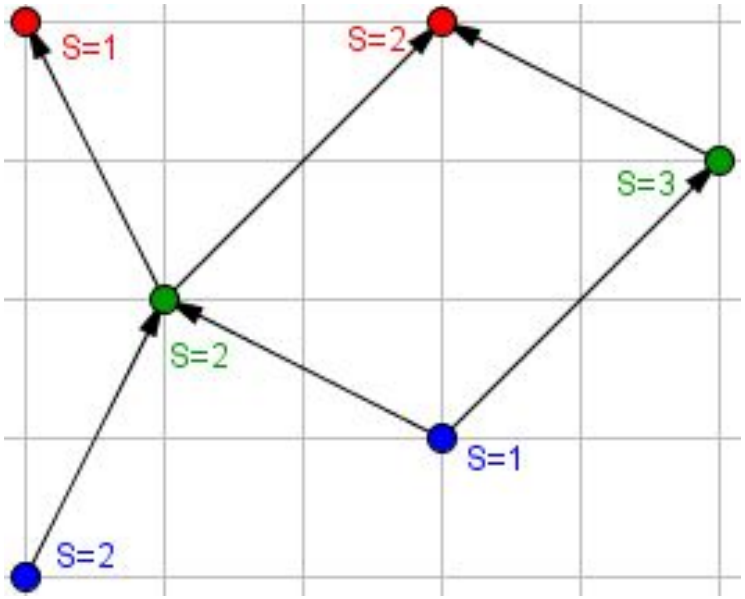
	1	2	3	4	5	6	7	8	9	10
1	1	0	1	0	0	1	0	0	0	0
2	0	1	0	0	1	0	0	1	0	0
3	0	0	0	0	0	0	0	1	0	0
4	0	0	1	0	0	0	0	0	0	1
5	0	0	0	0	0	0	0	0	0	0

Task 3: Rock Climbing

Author: Gan Wei Liang

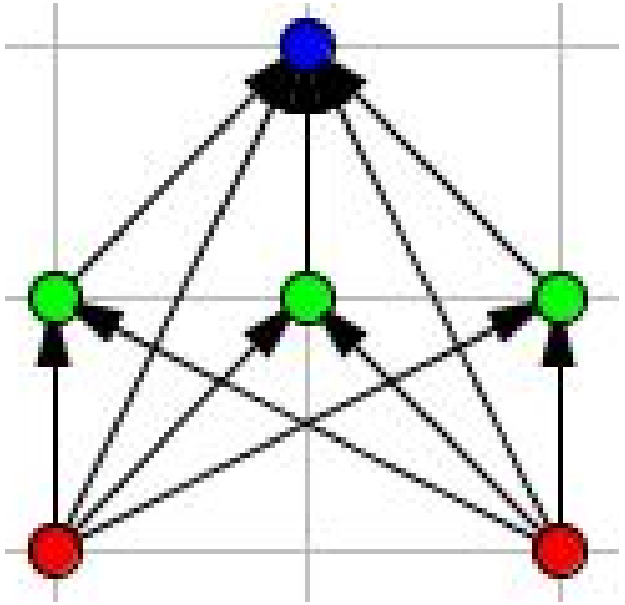
Abridged Problem Statement

- $N \leq 500$ rocks, each at (X_i, Y_i) and has slip rating of S_i
- Can climb from rock i to rock j if $Y_i < Y_j$ and $\max(|X_i - X_j|, |Y_i - Y_j|) \leq \max(S_i, S_j)$
- Find the smallest K such that for any set of K rocks, there is a directed path from some rock in the set to another
- K is hard to find so find K' instead
- K' = size of largest set of vertices such that no two vertices in the set are connected by a directed path
- It is easy to see that $K = K' + 1$
- In the right example, $K' = 2$
- Thus $K = 3$



Subtask 1

- $S_i = 2 \times 10^6$ for all i
- There is an edge from i to j as long as $Y_i < Y_j$
- Can only pick vertices with the same Y
- Count number of vertices with each value of Y using an array
- $K' =$ maximum number of vertices with the same Y
- For the right example, $K' = 3$
- Thus, $K = 4$



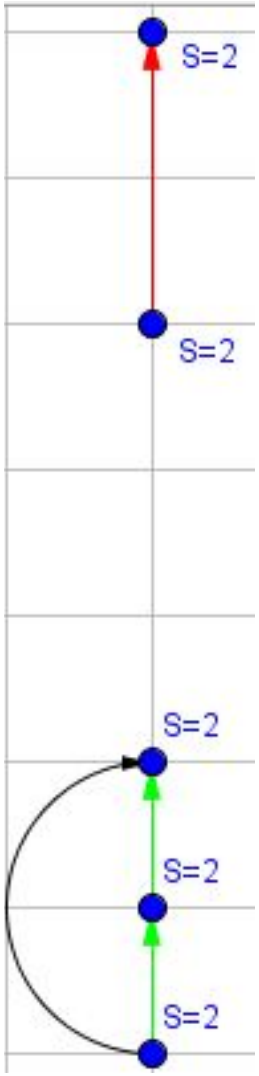
Subtask 2

- $1 \leq N \leq 20$
- N is small so check all $2^N \leq 2^{20} = 1048576$ possible subset of vertices
- For each subset, check every pair of vertices to see if there is a path from one to the other
- Use any search algorithm like DFS/BFS to search for path
- To make it faster, precompute for each pair of vertices whether there is a path from one to another and store in an array

Subtask 3

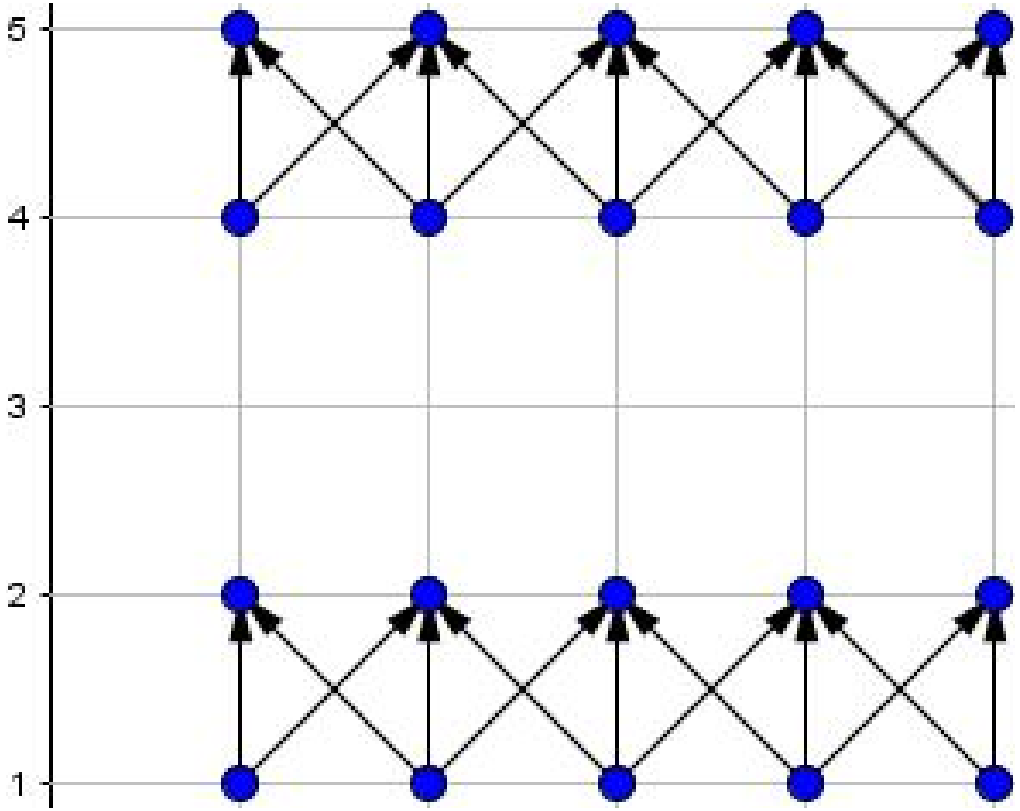
- $X_i = 0, S_i = S_j$ for all i and j
- For each vertex, check if there is an edge to the next highest vertex
- These edges will form disjoint paths, pick one vertex from each path
- $K' =$ number of such paths

  For the case on the right, $K' = 2$



Subtask 4

- $S_i = 1, Y_i$ is not a multiple of 3 for all i
- Note that this graph is bipartite

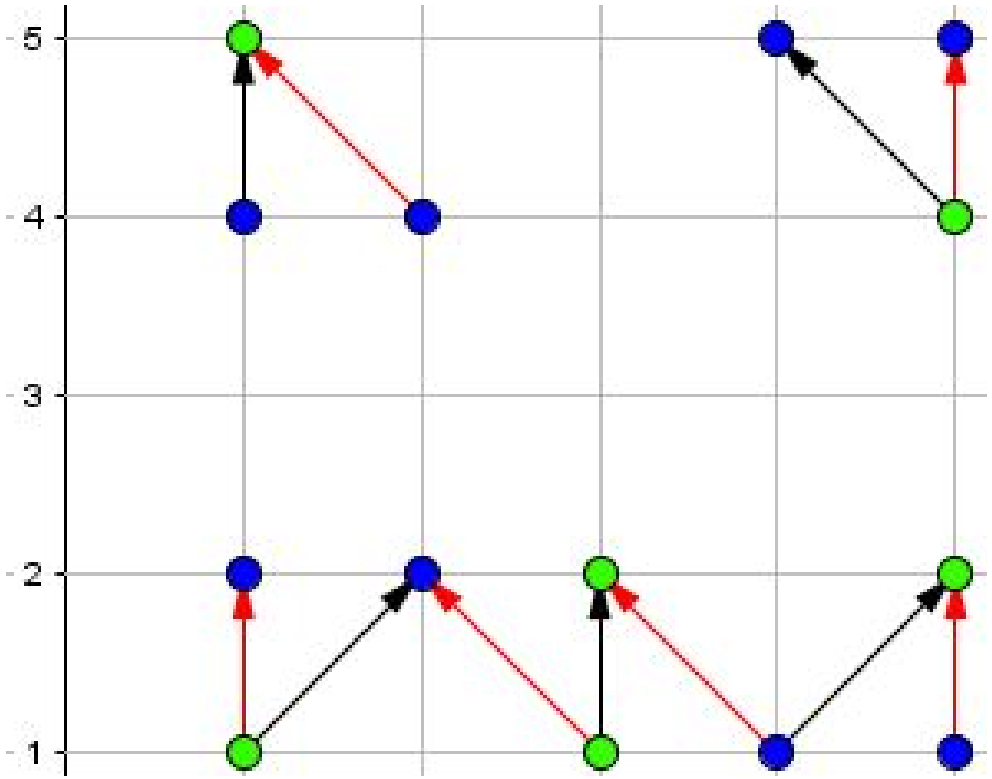


Algorithm

- Since all paths are of length 1, the answer is the maximum size of a set of vertices such that no 2 are connected by an edge
- => Maximum Independent Set (MIS) problem
- Size of MIS = N − Size of Minimum Vertex Cover (MVC)
- As the graph is bipartite, Size of MVC = Size of Maximum Cardinality Bipartite Matching (MCBM)
- Use $O(N^3)$ Augmenting Path Algorithm to find maximum matching of size M
- Answer is $N - M$

Example

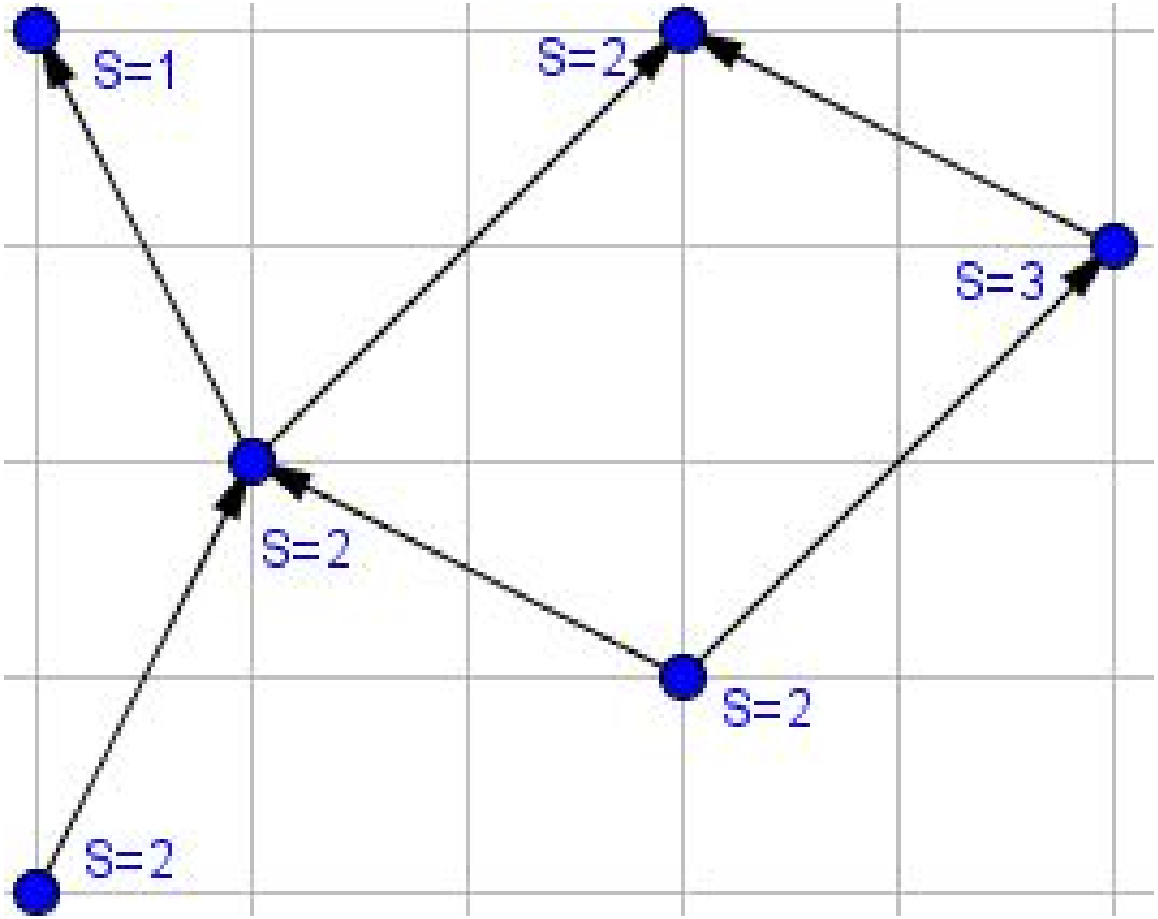
- Red edges form MCBM
- Green vertices form MVC
- Blue vertices form MIS
- For this case $K' = 6$
- Thus, $K = 7$



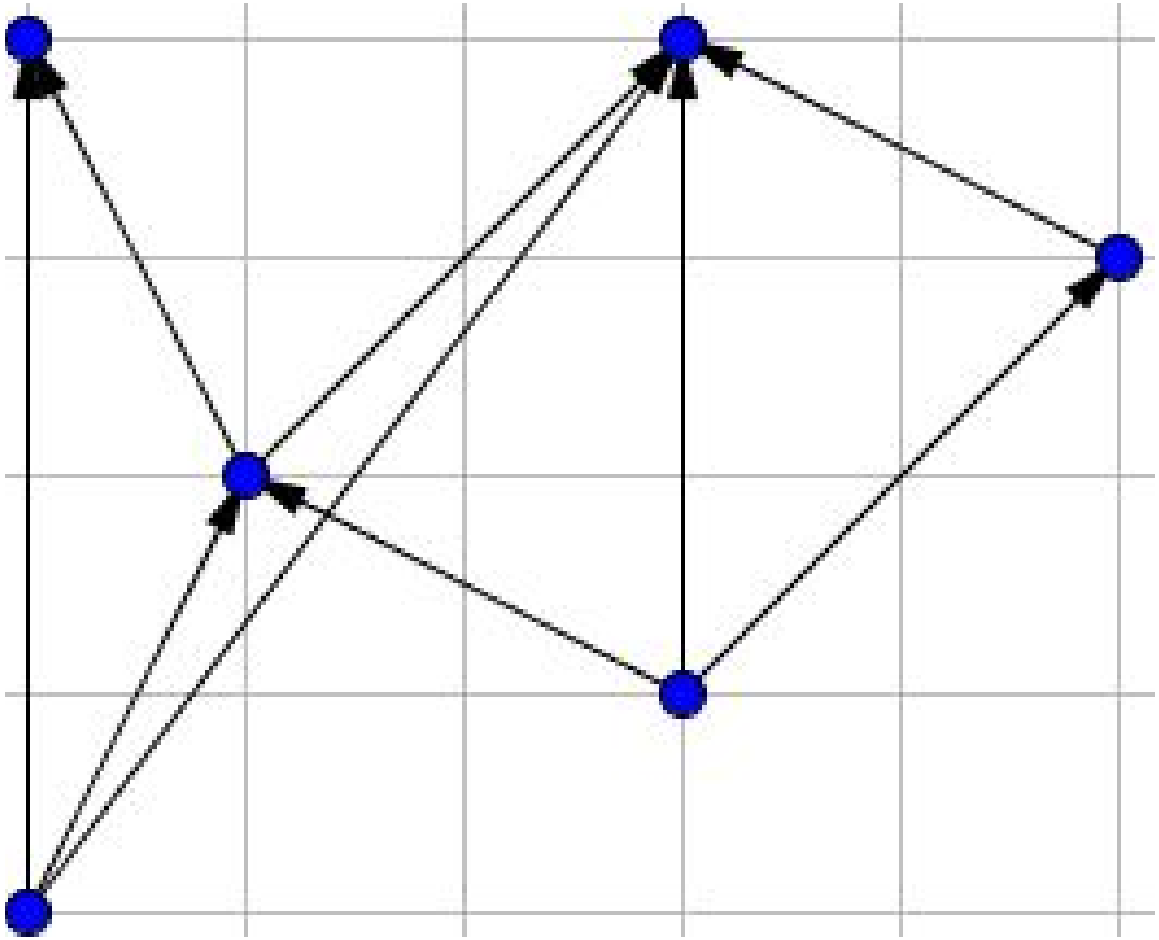
Subtask 5

- No other restrictions
- Construct a new graph where there is an edge from i to j if there is a path from i to j in the original graph
- To do this, set all the edge weights to 0 and run Floyd-Warshall algorithm
- Add an edge in the new graph from i to j if the shortest distance from i to j is 0
- Alternatively, can also run BFS/DFS from every vertex
- This is known as transitive closure

Example



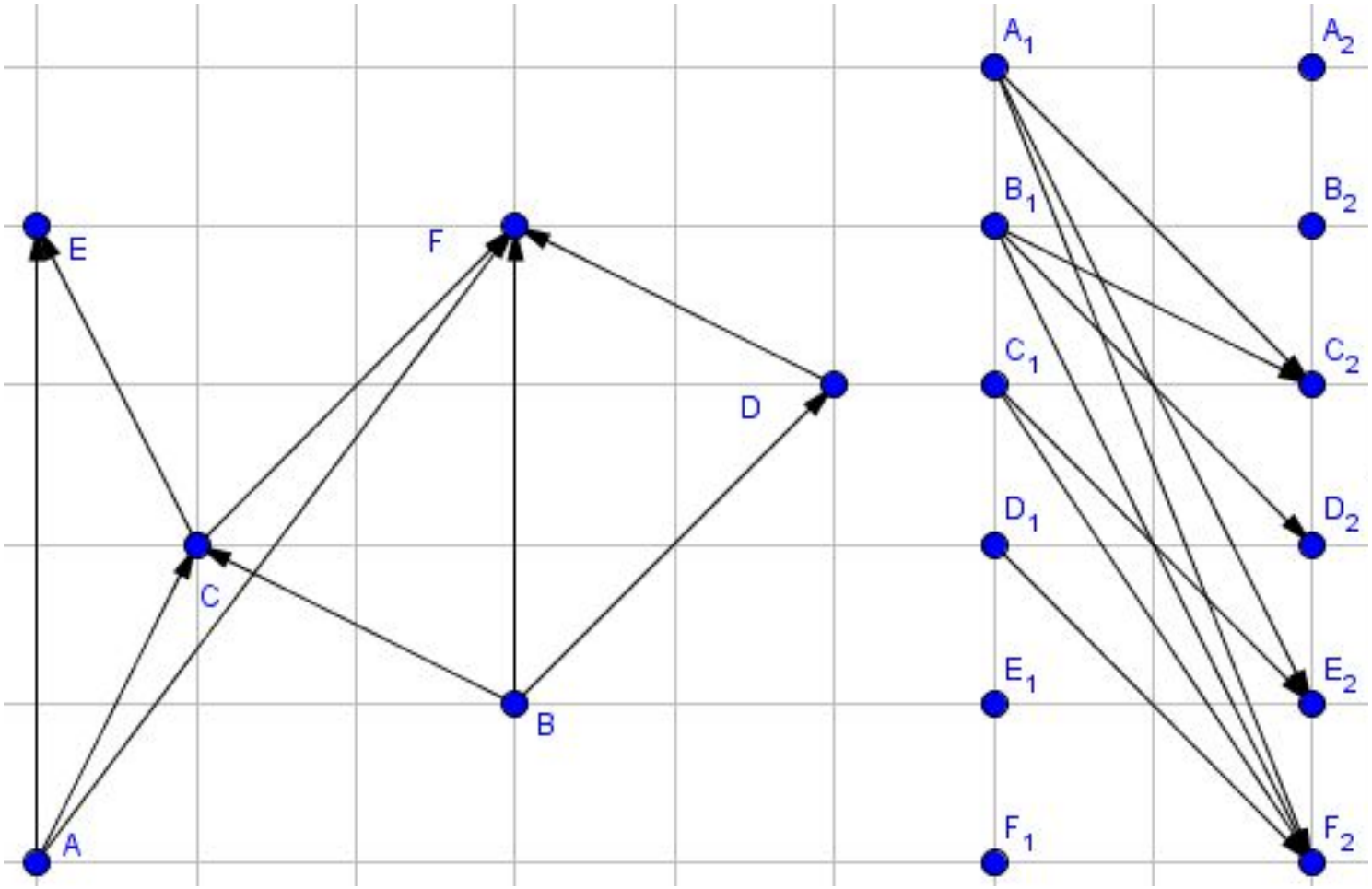
Transitive Closure



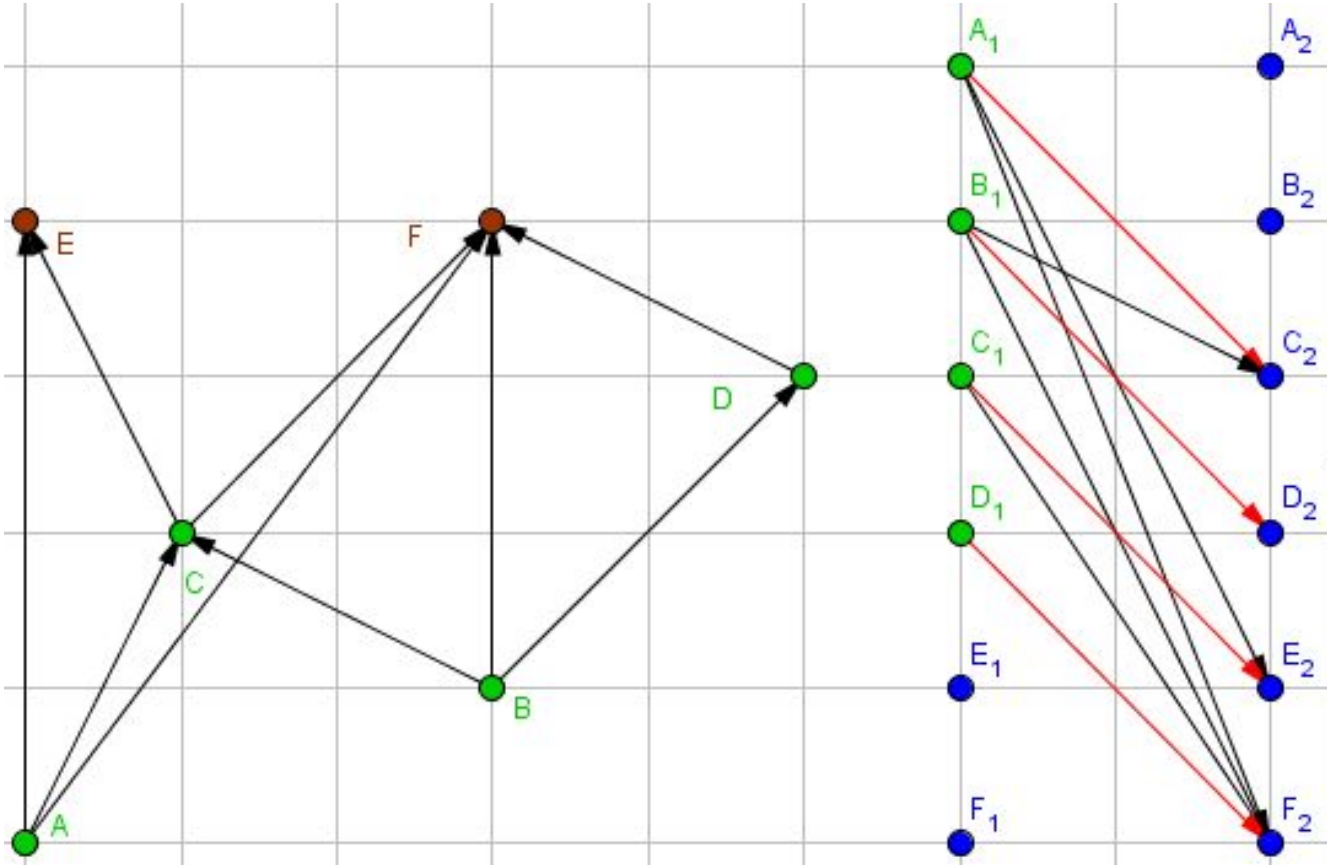
Algorithm

- In the new graph, an edge from i to j means there is a path from i to j so can't pick both
- => Maximum Independent Set (MIS) problem
- Construct a new graph with 2 copies of each vertex, i and i', add an edge from i to j' in the new graph for each edge from i to j in the original graph
- This graph is bipartite so Size of MVC = Size of MCBM
- It can be proven that the MVC in this bipartite graph will not contain both i and i'
- Thus, it equivalent to an MVC in the original graph

Bipartite Graph



Bipartite Matching



$$K' = 2 \Rightarrow K = 3$$

Conclusion

1. Perform transitive closure using Floyd-Warshall Algorithm
2. Construct bipartite graph with 2 copies of each vertex
3. Perform MCBM using augmenting path algorithm to get matching of size M
4. If $M = 0$, answer is -1, else $K = N - M + 1$

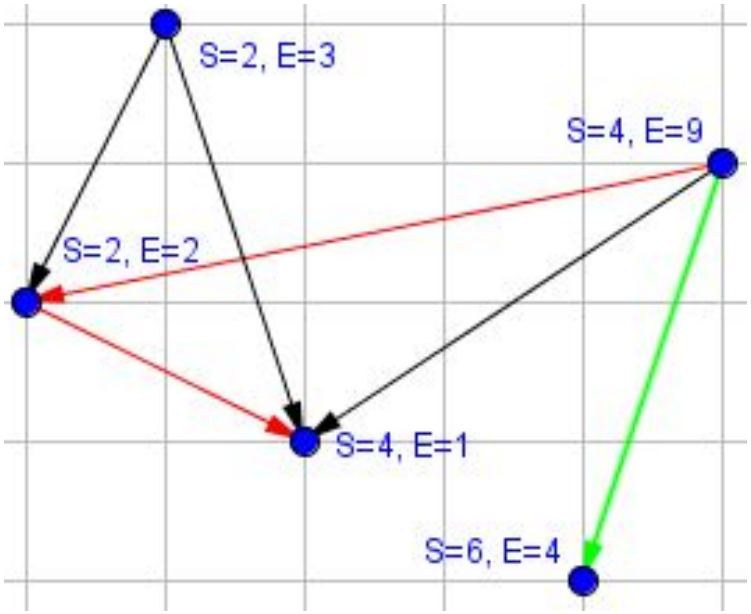
- Details of terms like MIS, MVC and MCBM can be found in Steven Halim's book "Competitive Programming 3"
- Proof of correctness of this algorithm is left as an exercise

Task 4: Panda Ski

Author: Gan Wei Liang

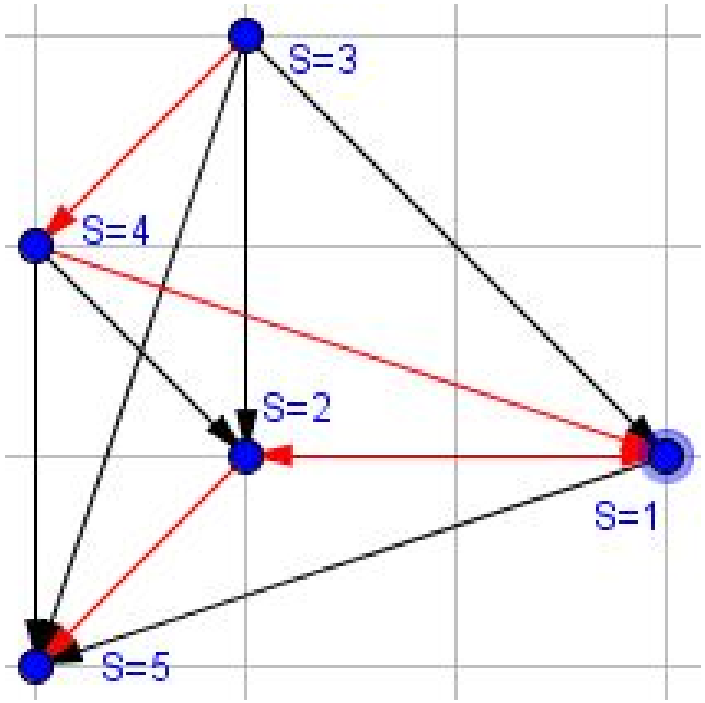
Abridged Problem Statement

- There are $N \leq 200000$ gates, each gate is at coordinate (X_i, Y_i) and has score S_i and easiness E_i
- You can go from gate i to gate j if $Y_i \geq Y_j$ and $\max(|X_i - X_j|, |Y_i - Y_j|) \leq E_i$
- Total score of a path is the sum of scores of the gates passed, each gate only counts once
- Find maximum possible total score of a path
- For the example, answer is 10



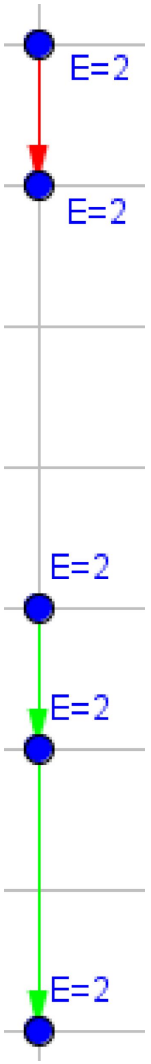
Subtask 1

- $E_i = 200000$ for all i
- You can go from gate i to gate j as long as $Y_i \geq Y_j$
- Start from the gate with the highest Y_i
- Go through all the gates
  in order of non-increasing Y_i
- Answer is the sum of scores
  of all the gates
- For the example, answer is 15



Subtask 2

- $X_i = 0, E_i = E_j$ for all i and j
- For each gate, try to pass through as many gates as possible by going to the next gate below
- This will form disjoint paths
- Answer is the maximum total score across all paths



Subtask 3

- $1 \leq N \leq 300, Y_i \neq Y_j$ for all $i \neq j$
- Construct a graph where the gates are vertices and edges from i to j are of weight S_j. Graph is acyclic because can go downhill but cannot go uphill
- Use Floyd-Warshall's Algorithm to calculate the longest distance $d(i, j)$ between every pair of gates i and j in $O(N^3)$
- The maximum score of path from i to j is $S_i + d(i, j)$
- Thus, answer is the maximum value of $S_i + d(i, j)$ across all pairs of gates i and j

Subtask 4

- $1 \leq N \leq 2000, Y_i \neq Y_j$ for all $i \neq j$
- Add a source vertex of score 0 that connects to all the gates and run Dijkstra's Algorithm
- The answer is the length of the longest path
- Alternatively, let $dp(i)$ be the maximum score of a path starting at gate i
- $dp(i) = S_i + \max(dp(j))$ for all j such that you can go from gate i to gate j
- Process the gates in increasing height to find $dp(i)$
- The answer is the maximum $dp(i)$ for all gates i

Subtask 5

- $1 \leq N \leq 50000, E_i = E_j, Y_i \neq Y_j$ for all $i \neq j$
- $N^2 \leq 2.5 \times 10^9$ is too large for the previous solution
- You can go from gate i to gate j if $Y_i \geq Y_j$ and $\max(|X_i - X_j|, |Y_i - Y_j|) \leq E_i$

  → ($X_i - E_i \leq X_j \leq X_i + E_i$ and $Y_i - E_i \leq Y_j \leq Y_i$)
- For this subtask, all the gates have the same easiness rating. Let this value be E
- When processing the gates in increasing height, if a gate j has $Y_j < Y_i - E$ then it does not need to be

Algorithm

Sort gates i in increasing Y_i

Add the gates in order to queue q, lowest gate at the front

for $i = 1\ to\ N$

    while y-coordinate of q.front $< Y_i - E$

        gate $j = q$.front

        if $maxy[X_j] = Y_j$, update $maxdp[X_j] = 0$

        pop j from q

    $dp[i] = S_i +$ max $(maxdp[X], X_i - E \leq X \leq X_i + E)$

    update $maxdp[X_i] = dp[i]$

print $\max(dp[i], 1 \leq i \leq N)$

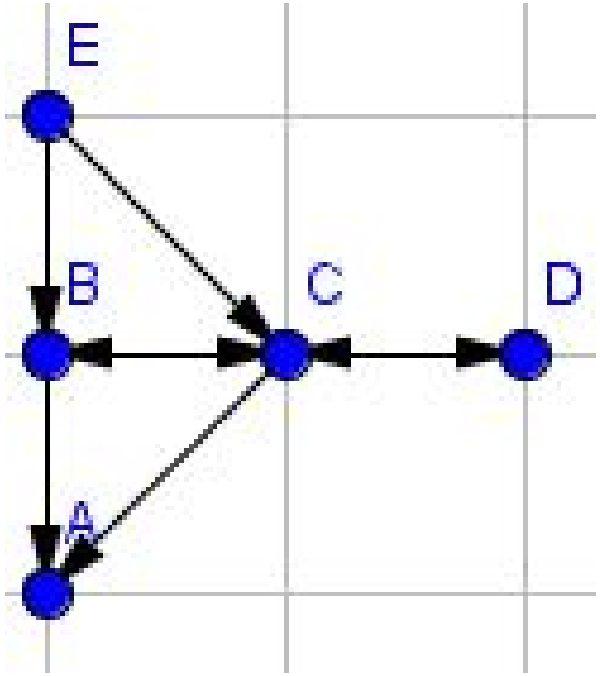
- By constructing a segment tree for $maxdp$ the range queries can be answered in $O(\log N)$ so the complexity is $O(Nlog\ N)$

Subtask 6

- Same as subtask 5 except Y_i is not unique
- It is possible to go back and forth between two gates on the same row, so it may be better to go through the same gate multiple times for the best score
- For the example on the right,

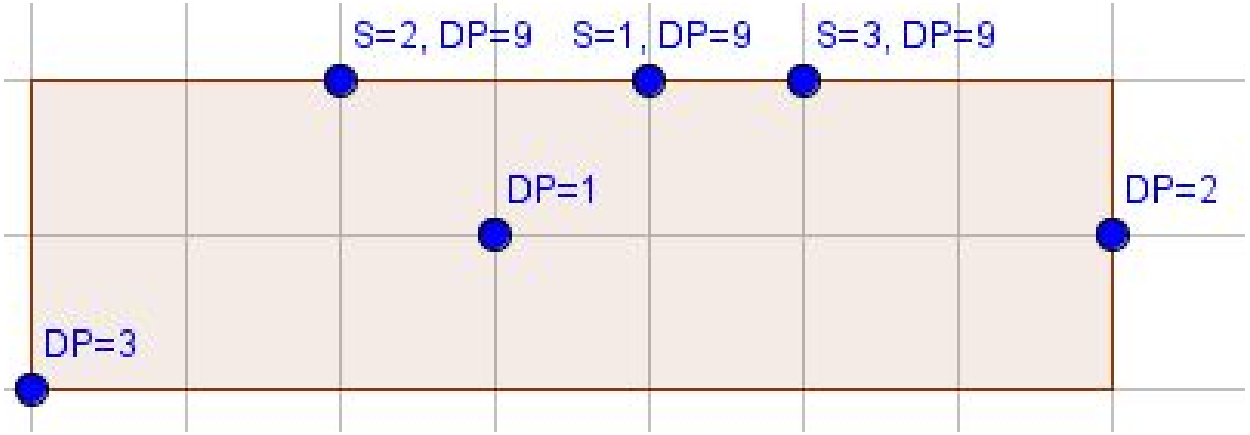
  A best path is E→C→D→C→B→A

  since gate scores are positive.
- There is no best path that visits each gate only once



Connected gates

- A set of gates are connected if they are the same height and every pair of neighbouring gates are at most E units apart
- In this case, $E = 2$ and the top 3 gates are connected
- DP value is the sum of the scores of the 3 gates plus the maximum DP value in the combined range,



Algorithm

Sort gates i in non-decreasing Y_i and increasing X_i for the same Y_i

for $i = 1\ to\ N$

while y-coordinate of q.front $< Y_i - E$

gate $j = q$.front

if $maxy[X_j] = Y_j$, update $maxdp[X_j] = 0$

pop j from q

set $j = i$, increment j until $Y_j \neq Y_{j+1}$ or $X_{j+1} - X_j > E$

$maxv = \max(maxdp[X], X_i - E \leq X \leq X_j + E)$

for $k = i\ to\ j$

update $dp[k] = \text{sum}(S_v, i \leq v \leq j) + maxv$

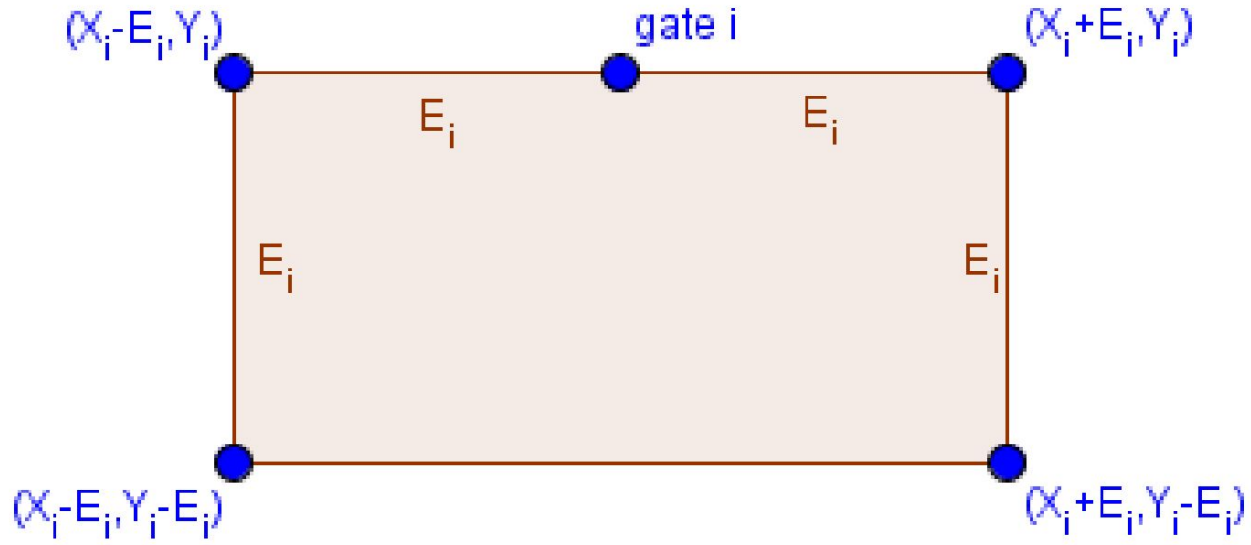
update $maxdp[X_k] = dp[k]$

skip for loop to gate $j + 1$

print $\max(dp[i], 1 \leq i \leq N)$

Subtask 7

- Same as subtask 5 but E_i may not be equal to E_j so the previous solution does not work
- Condition to go from gate i to gate j is
- $(X_i - E_i \leq X_j \leq X_i + E_i$ and $Y_i - E_i \leq Y_j \leq Y_i)$
- Thus, gate j must lie in the rectangle below



2D Segment Tree

- Use a discretised 2-dimensional segment tree, which can query a rectangular range in $O(\log^2 N)$ time

for $i = 1\ to\ N$

$dp[i] = S_i + \max(f(X, Y), X_i - E_i \leq X \leq X_i + E_i$ and $Y_i - E_i \leq Y \leq Y_i)$

Update point $f(X_i, Y_i) = dp[i]$

print $\max(dp[i], 1 \leq i \leq N)$

- Runtime is approximately $O(N \log^2 N)$

Subtask 8

- Same as subtask 7 except $1 \leq N \leq 200000$
- 2D segment tree with discretisation will exceed the limits for this question as N is larger
- Construct a modified 1D segment tree where each node represents a range of x-coordinates
- The root node represents the range $[\min(X_i), \max(X_i)]$ across all gates i
- In each node, add a node stack that can store pairs of (y-coordinate, value)

Algorithm

Sort gates i in increasing Y_i

for $i = 1\ to\ N$

query the range $[X_i - E_i, X_i + E_i]$ on the segment tree

for each node traversed whose range is in the above range

binary search the node stack to find (Y, V) with the smallest $Y \geq Y_i - E_i$

update $dp[i] = \max(dp[i], S_i + V)$

for each node in the segment tree whose range contains X_i

while top of node stack (Y, V) satisfies $V \leq dp[i]$

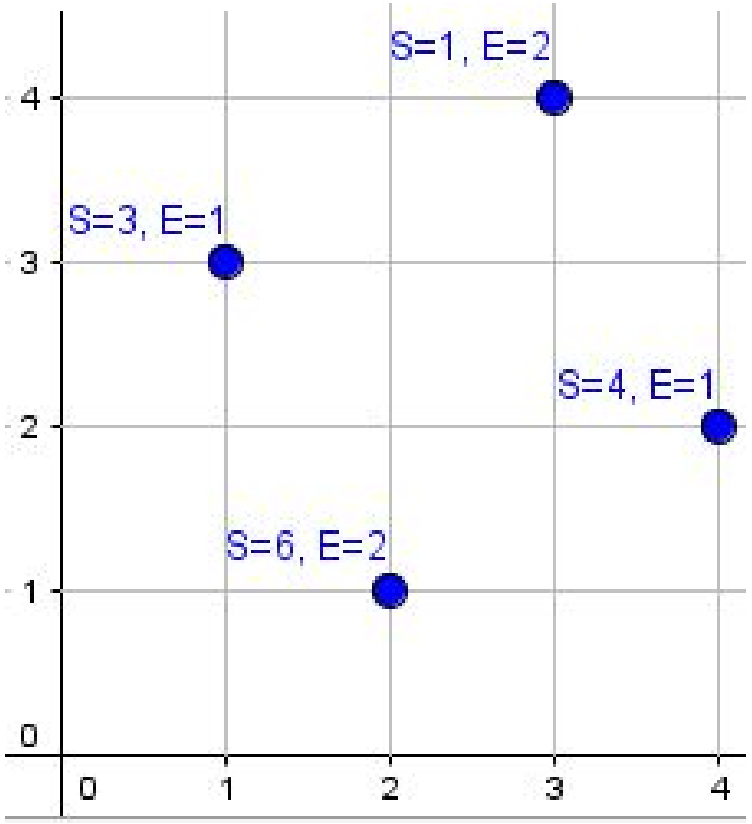
pop (Y, V) from node stack

push $(Y_i, dp[i])$ onto node stack

print $\max(dp[i], 1 \leq i \leq N)$

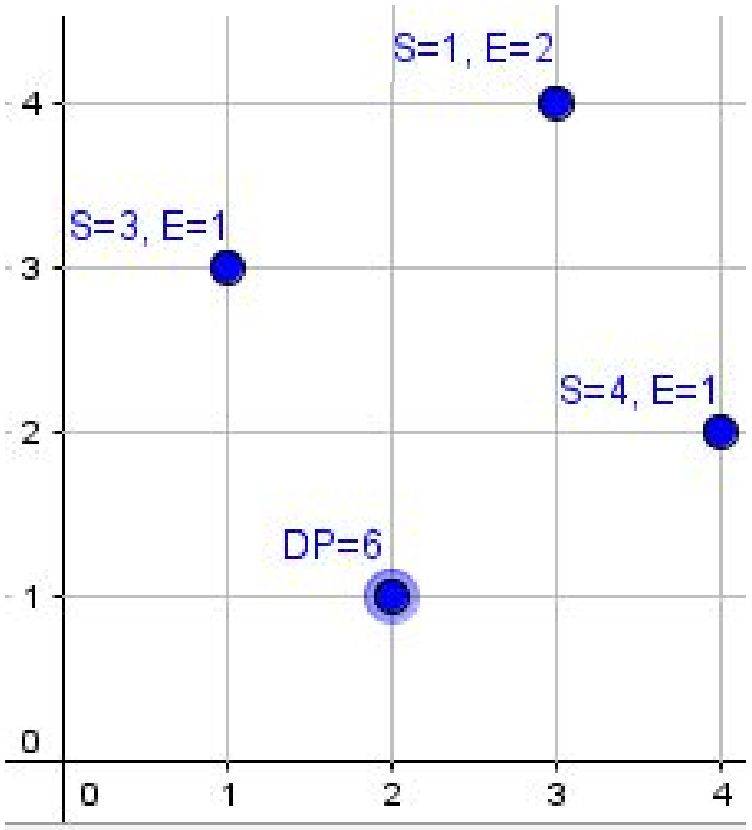
Example

<table>
<tr><td colspan="4">Range: [1,4]</td></tr>
<tr><td colspan="2">Range: [1,2]</td><td colspan="2">Range: [3,4]</td></tr>
<tr><td>Range: [1,1]</td><td>Range: [2,2]</td><td>Range: [3,3]</td><td>Range: [4,4]</td></tr>
</table>



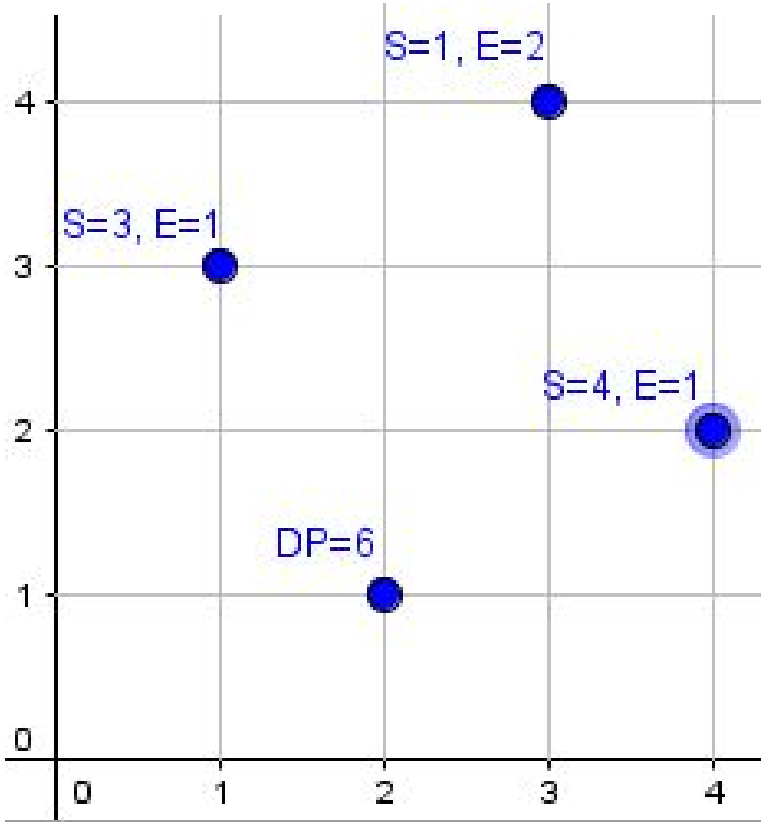
Example

[(1,6)]			
[(1,6)]		[]	
[]	[(1,6)]	[]	[]



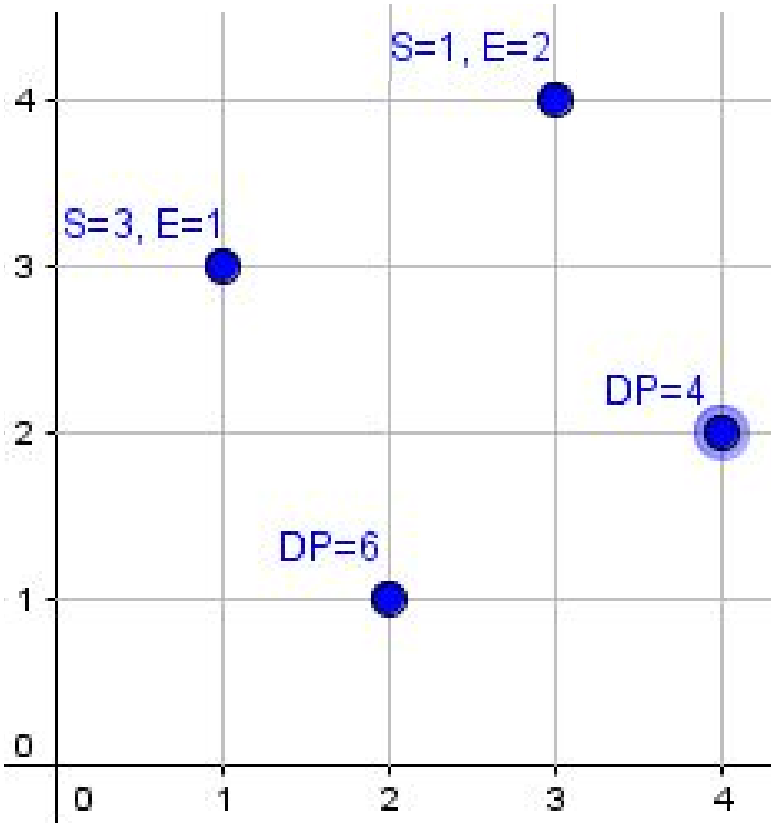
Example

[(1,6)]			
[(1,6)]		[]	
[]	[(1,6)]	[]	[]



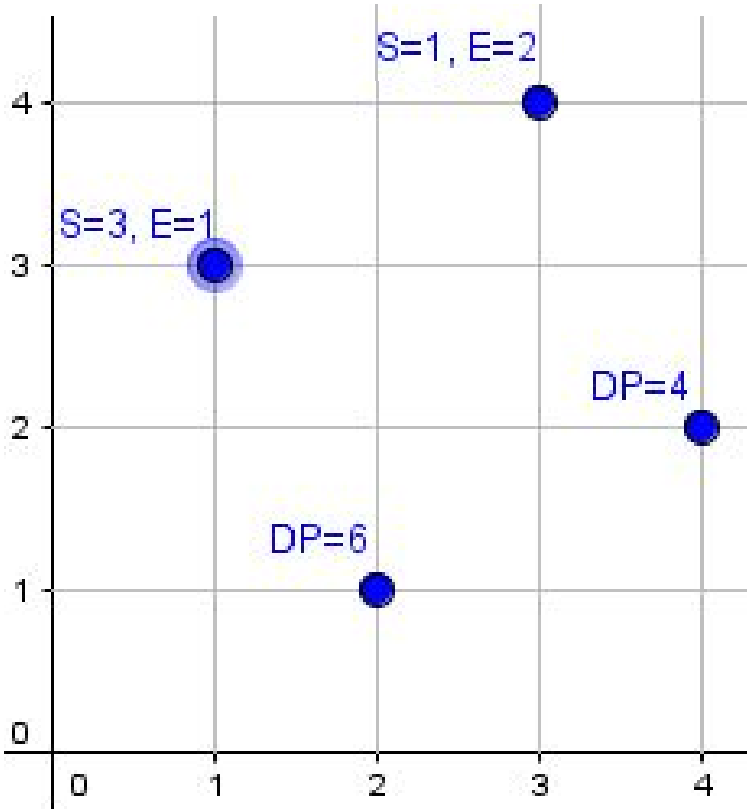
Example

[(1,6),(2,4)]			
[(1,6)]		[(2,4)]	
[]	[(1,6)]	[]	[(2,4)]



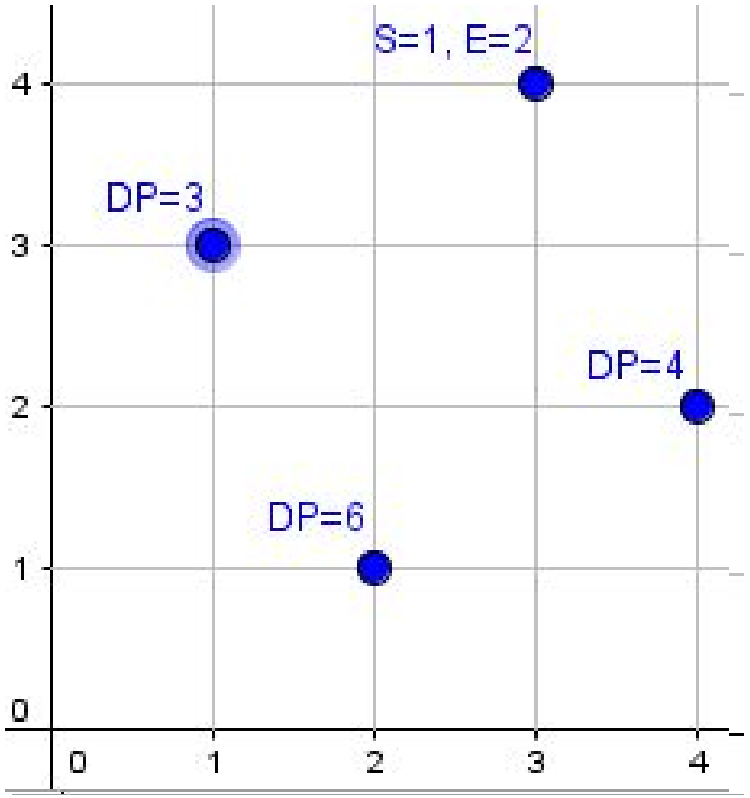
Example

[(1,6),(2,4)]			
[(1,6)]		[(2,4)]	
[]	[(1,6)]	[]	[(2,4)]



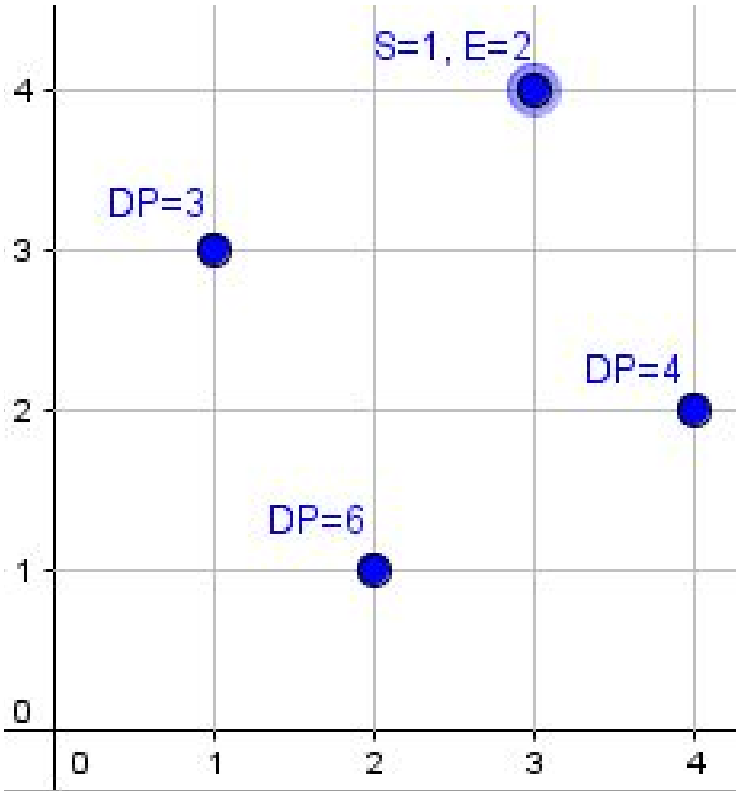
Example

<table>
<tr><td colspan="4">[(1,6),(2,4),(3,3)]</td></tr>
<tr><td colspan="2">[(1,6),(3,3)]</td><td colspan="2">[(2,4)]</td></tr>
<tr><td>[(3,3)]</td><td>[(1,6)]</td><td>[]</td><td>[(2,4)]</td></tr>
</table>



Example

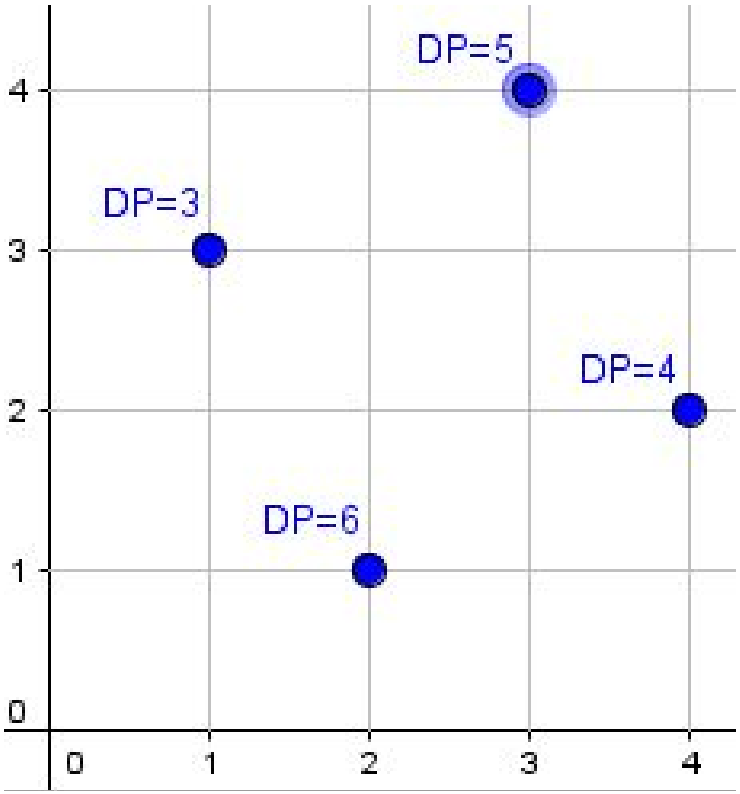
<table>
<tr><td colspan="4">[(1,6),(2,4),(3,3)]</td></tr>
<tr><td colspan="2">[(1,6),(3,3)]</td><td colspan="2">[(2,4)]</td></tr>
<tr><td>[(3,3)]</td><td>[(1,6)]</td><td>[]</td><td>[(2,4)]</td></tr>
</table>



Example

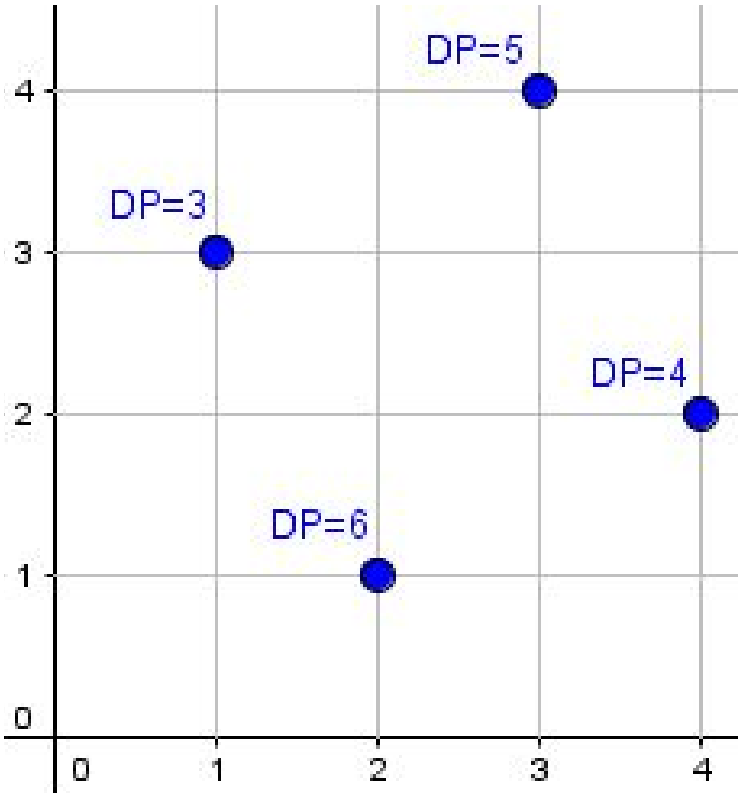
[(1,6),~~(2,4),(3,3)~~,(4,5)]	
[(1,6),(3,3)]	[(2,4),(4,5)]

[(3,3)]	[(1,6)]	[(4,5)]	[(2,4)]



Example

<table>
<tr><td colspan="4">[(1,6),(4,5)]</td></tr>
<tr><td colspan="2">[(1,6),(3,3)]</td><td colspan="2">[(2,4),(4,5)]</td></tr>
<tr><td>[(3,3)]</td><td>[(1,6)]</td><td>[(4,5)]</td><td>[(2,4)]</td></tr>
</table>



Time Complexity

- Let $\max(X_i) - \min(X_i) = W$, each update is stored in at most $\log W$ nodes so the total number of updates stored is $N \log W$
- Each update is pushed and popped at most once so overall complexity for updates is $O(N \log W)$
- Each query on the segment tree requires a $\log N$ binary search on $\log W$ nodes
- So the overall complexity for the queries $O(N \log W \log N)$

Conclusion

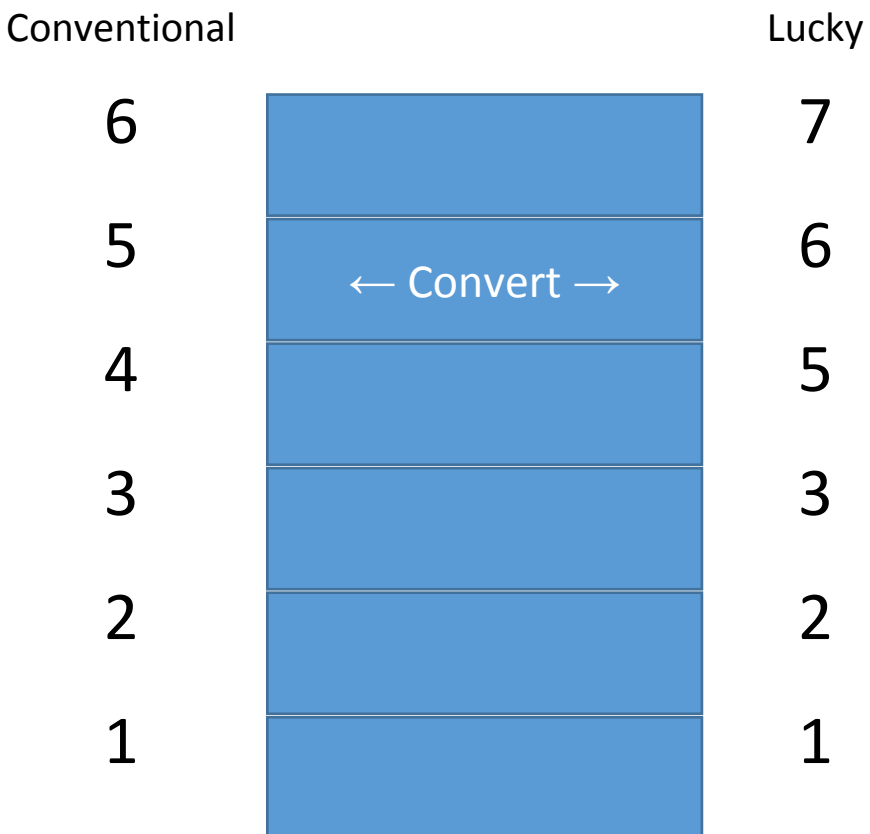
- The solution presented in subtask 8 will not earn full credit since it cannot handle the case of connected gates
- Need to combine with solution from subtask 6 to get the full credit
- Due to time constraints, details of how and why the algorithms work have been omitted
- Proof of methods and theorems used have also been omitted, this shall be left as an exercise

Task 5: Unlucky floors

Author: Ranald Lam

Problem Summary

- Conventional numbering schemes are positive integers
- Lucky numbering scheme omits numbers with ‘4’ and ‘13’



Subtask 1: Print answer

Subtask	Marks	N	T_i	X_i
1	5	$0 < N \leq 50$	$T_i = 1$ or 2	$0 < X_i \leq 25$

- Floor numbers are up to 25
- Answer for up to 20 is provided in the statement
- Manually solve the remaining 5
- Code using “Answer array” or if-else statements

Subtask 2: String Processing

Subtask	Marks	N	T_i	X_i
1	5	$0 < N \leq 50$	$T_i = 1$ or 2	$0 < X_i \leq 25$
2	12	$0 < N \leq 50$	$T_i = 1$ or 2	$0 < X_i \leq 100,000$

- We define unlucky numbers as those that contain either a ‘4’ or ‘13’
- To check if a number is unlucky:
  - Convert the number into individual digits (using sprintf/stringstream/base conversion)
  - Loop through each digit to check for presence of ‘4’
  - Loop through each consecutive pair of digits to check for presence of ’13’
- Loop through from 1 onwards, count the number of unlucky numbers to calculate the answer

Subtask 2: String Processing

```
bool unlucky(long long x) {
    char str[20];
    sprintf(str, "%lld", x);
    for (int i = 0, l = strlen(str); i < l; i++) {
        if (str[i] == '4') return 1;
        if (i == 0) continue;
        if (str[i-1] == '1' && str[i] == '3') return 1;
    }
    return 0;
}
```

Subtask 3: String Processing + Counter

Subtask	Marks	N	T_i	X_i
1	5	$0 < N \leq 50$	$T_i = 1$ or 2	$0 < X_i \leq 25$
2	12	$0 < N \leq 50$	$T_i = 1$ or 2	$0 < X_i \leq 100,000$
3	18	$0 < N \leq 100,000$	$T_i = 1$ or 2	$0 < X_i \leq 100,000$

- Maintain 2 counters: 1 for conventional, 1 for lucky

```
for (int c = 1, l = 1; c <= 100000; c++, l++) {
    while (unlucky(l)) l++;
    lucky_to_conv[l] = c;
    conv_to_lucky[c] = l;
}
```

Subtask 4 Method 1: Manual / Pattern Recognition

Subtask	Marks	N	T_i	X_i
4	11	$0 < N \leq 100,000$	$T_i = 1$	$X_i = 10^K - 1$ where $1 \leq K \leq 16$

- Find a formula then compute by hand manually

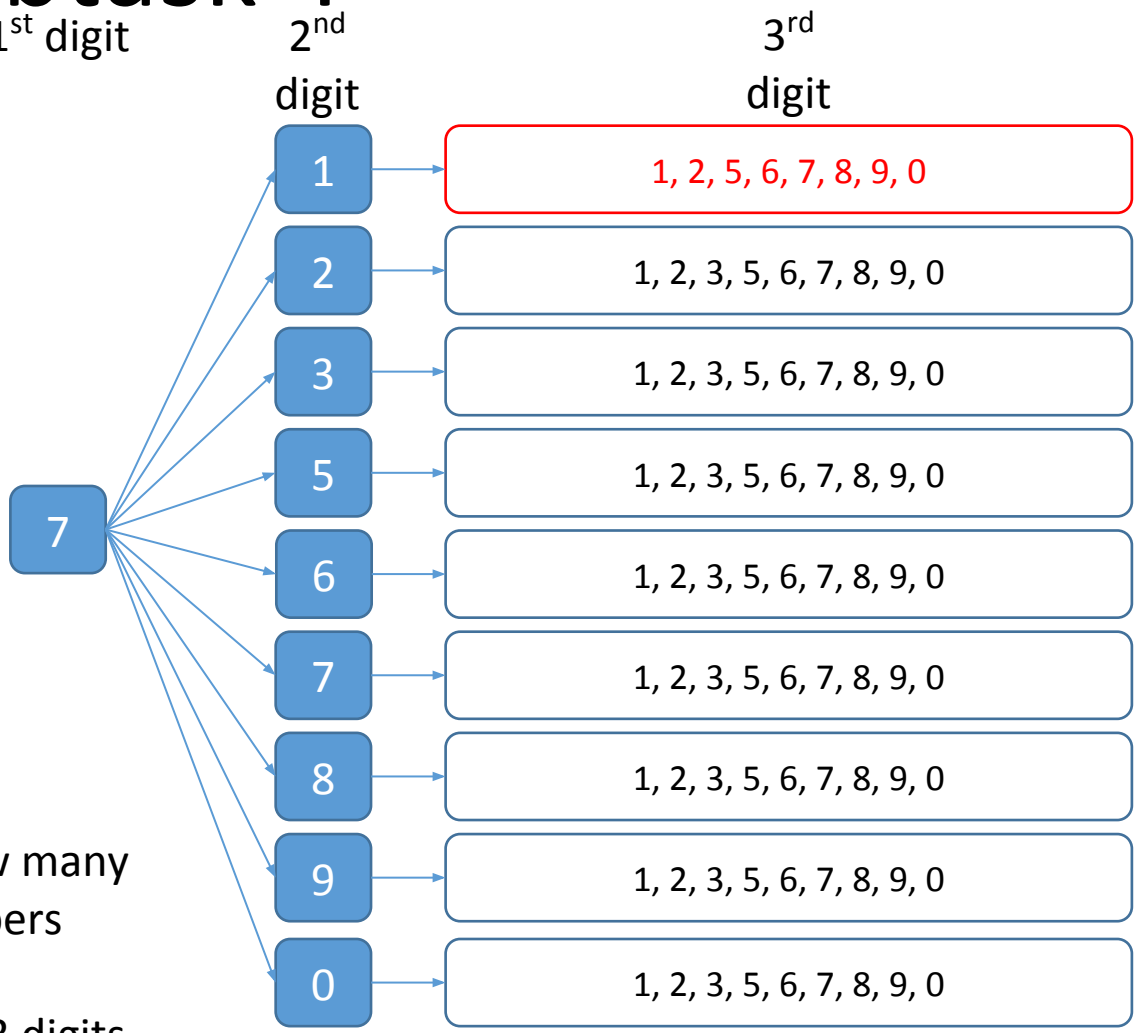
k	10^k-1	f(k)	
1	9	8	
2	99	79	
3	999	710	(79-8)*10
4	9999	6318	(710-79)*10+8
5	99999	56159	(6318-710)*10+79
6	999999	499120	(56159-6318) *10+710
k	10^k-1	f(k) =	(f(k-1) – f(k-2))*10 + f(k-3)

Subtask 4 Method 2: Dynamic Programming

Subtask	Marks	N	T_i	X_i
4	11	$0 < N \leq 100,000$	$T_i = 1$	$X_i = 10^K - 1$ where $1 \leq K \leq 16$

- Use dynamic programming to count number of lucky numbers for each digit
- $dp(p, d)$: no. of lucky numbers with ***p*** digits and ***d*** as first digit

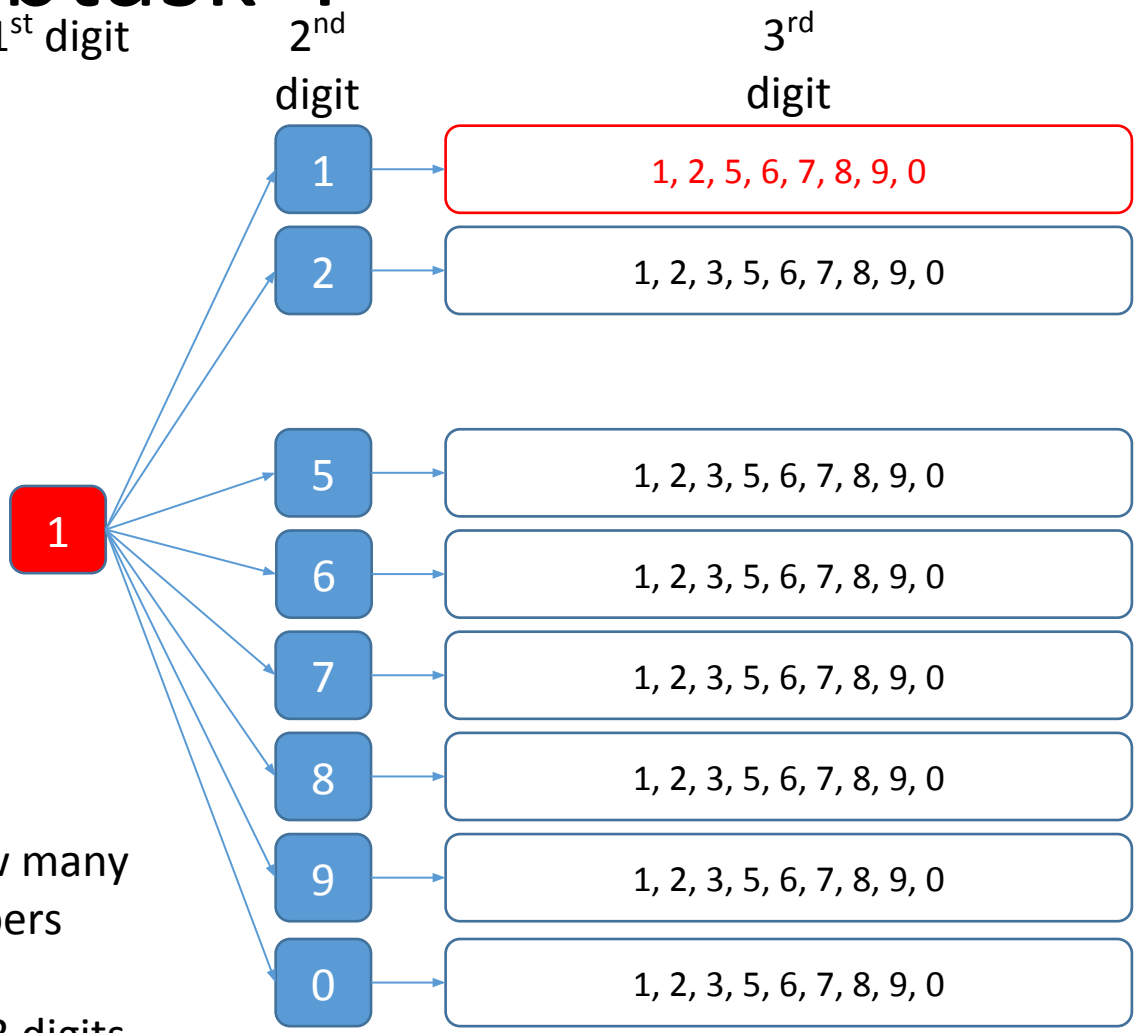
Subtask 4



Counts how many lucky numbers with:

- exactly 3 digits
- starting with **7**

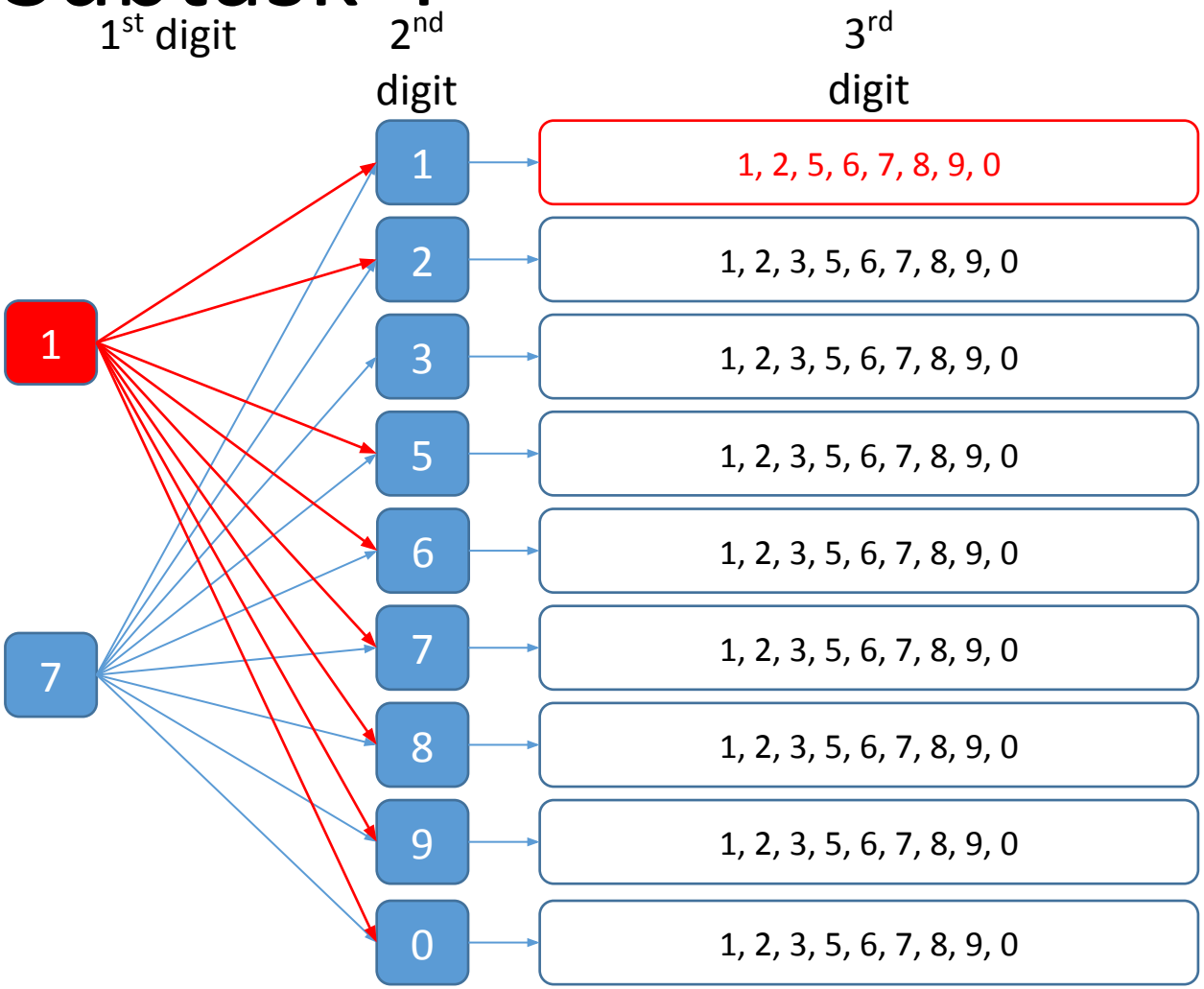
Subtask 4



Counts how many lucky numbers with:

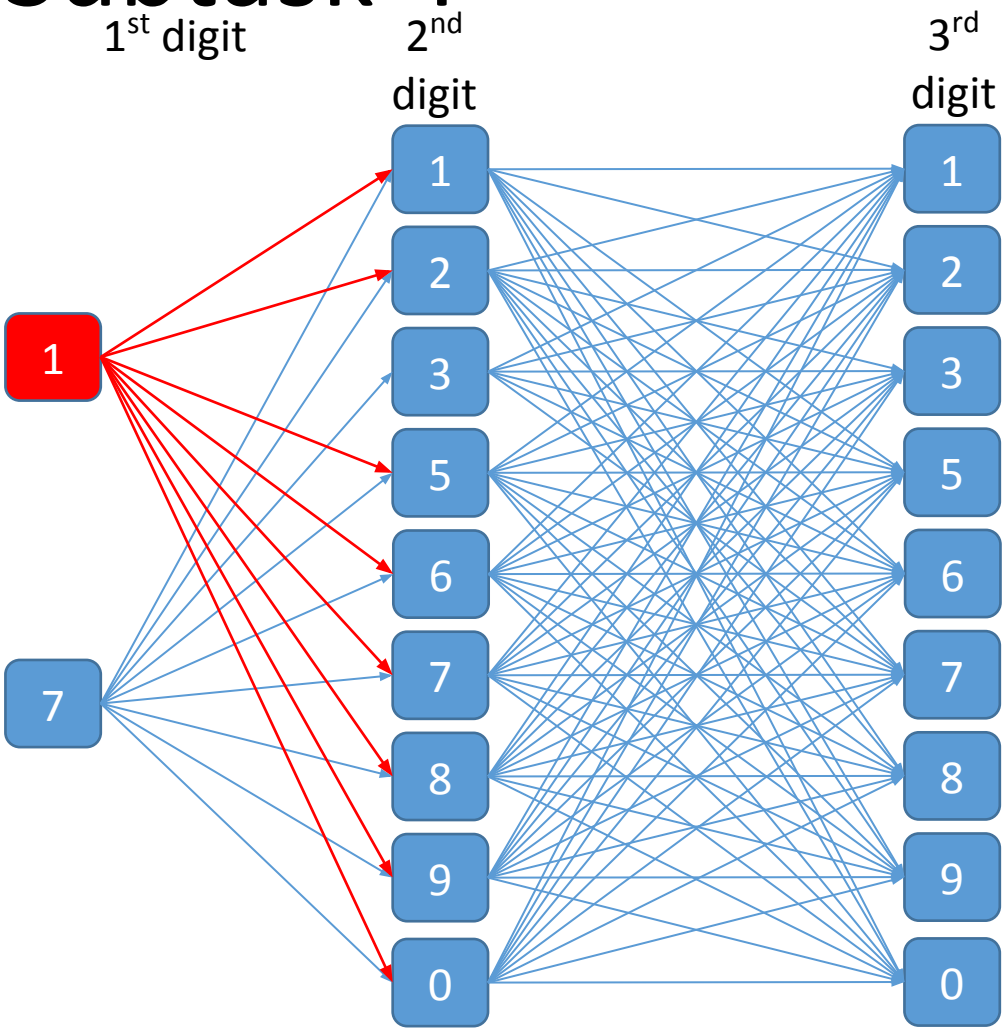
- exactly 3 digits
- starting with **1**

Subtask 4



Overlapping subproblems!

Subtask 4



Overlapping subproblems!

Subtask 4:
Dynamic Programming

- Take the rocks as vertices

- Draw a directed edge from vertex i to j if you can climb from rock i to j directly

- You can climb from rock i to j through a series of rocks if there is a directed path from rock i to j

Subtask 4:
Dynamic Programming

- Find the smallest value of K such tha
  set of K vertices, there exists a direct
  from some vertex in the set to anoth

- If K' is the size of the largest set of ve
  such that no two vertices in the set a
  connected by a directed path, we can
  that $K = K' + 1$. Let us find K' instea

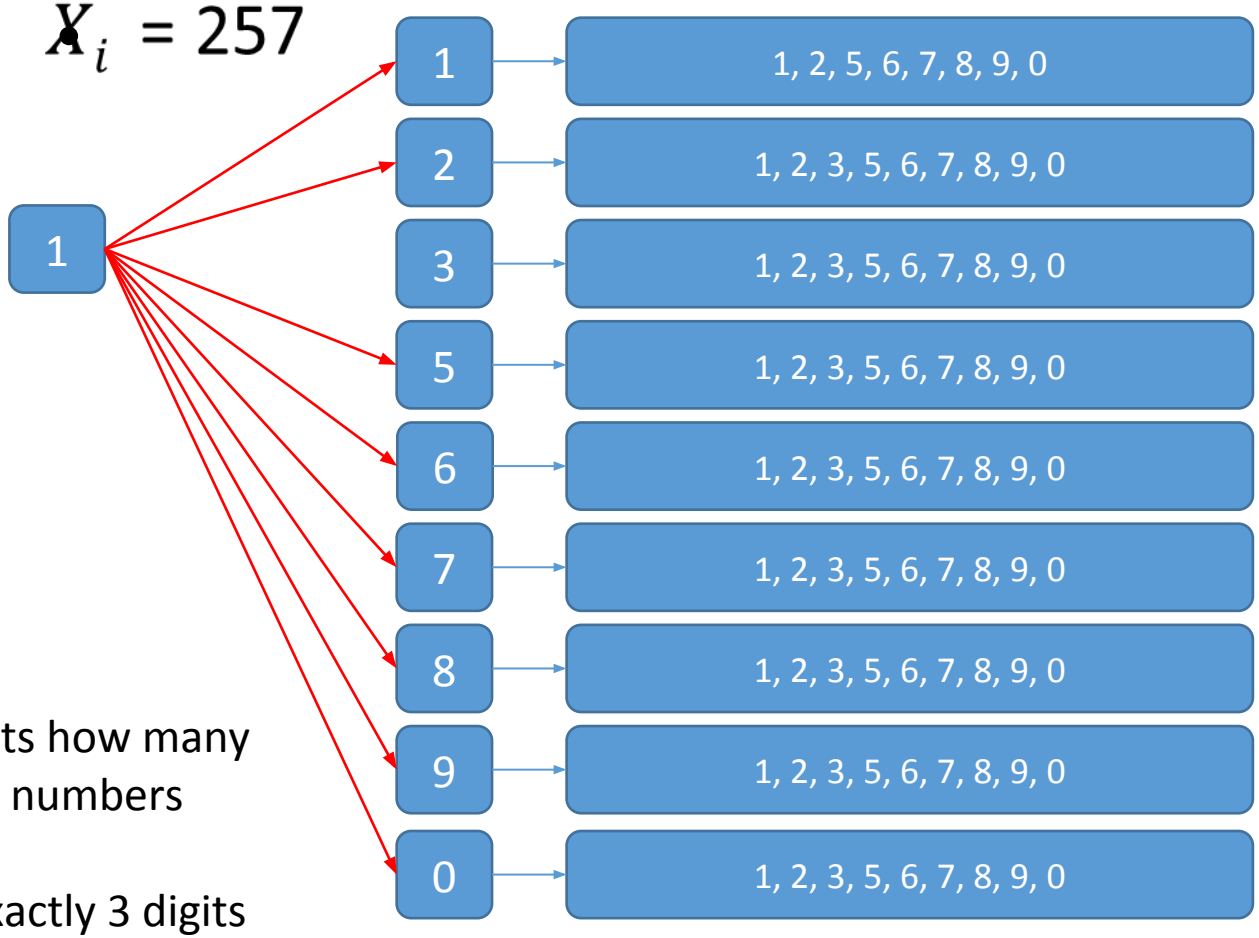
Subtask 5

Subtask	Marks	N	T_i	X_i
4	11	$0 < N \leq 100,000$	$T_i = 1$	$X_i = 10^K - 1$ where $1 \leq K \leq 16$
5	37	$0 < N \leq 100,000$	$T_i = 1$	$0 < X_i \leq 10^{16}$

- We need a more efficient way of counting lucky numbers
- What if we break it down into digits?
  - If the current digit is 7, what are the possible next digits?
    - 0, 1, 2, 3, 5, 6, 7, 8, 9
  - If the current digit is 1, what are the possible next digits?
    - 0, 1, 2, 5, 6, 7, 8, 9
  - If the current digit is 4, what are the possible next digits?
    - None! (No lucky number can be formed)

Subtask 5

$X_i = 257$



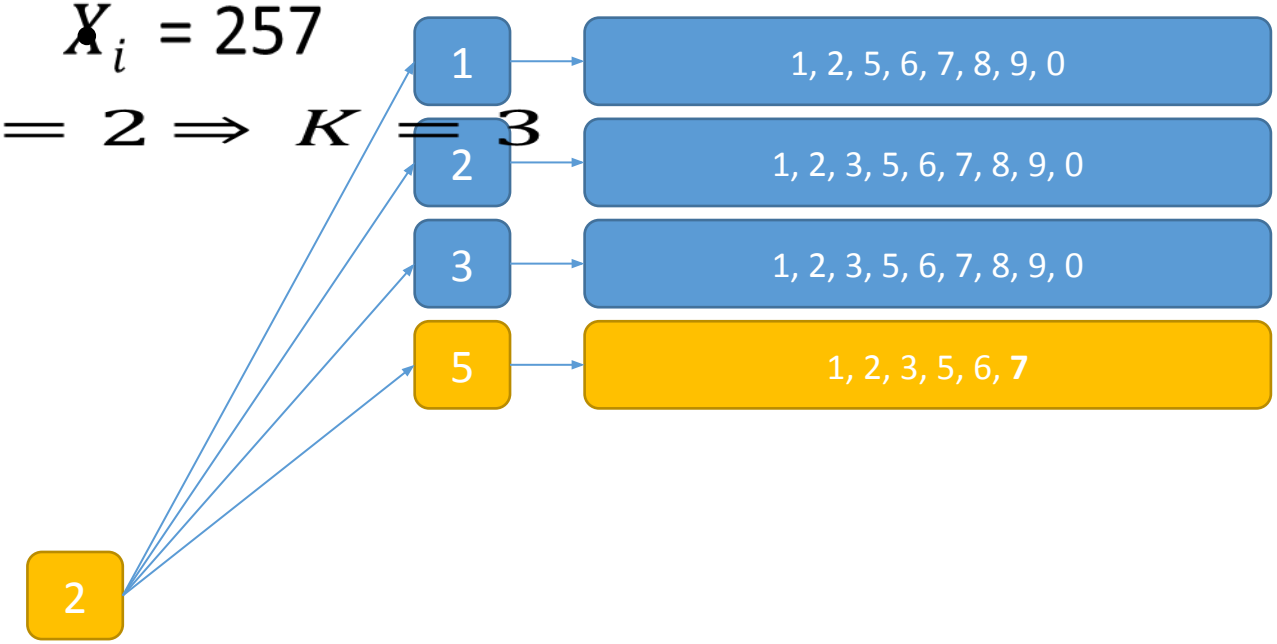
Counts how many lucky numbers with:

- exactly 3 digits
- starting with 1

Subtask 5

$X_i = 257$

$K' = 2 \Rightarrow K = 3$



Counts how many lucky numbers with:

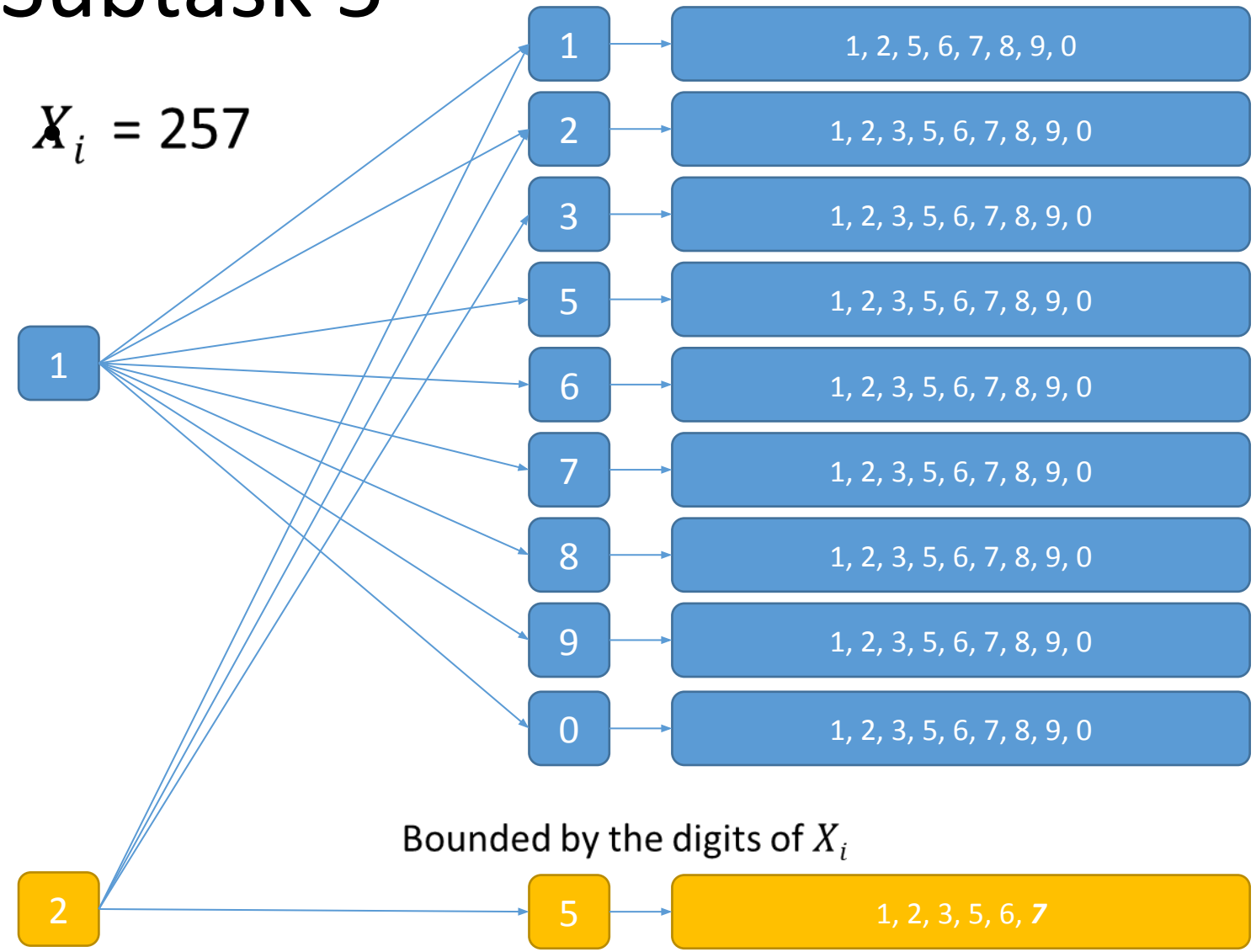
- exactly 3 digits
- starting with **2**
- Not more than X_i

Counts how many lucky numbers with:

- exactly 3 digits
- starting with **25**
- Not more than X_i

Subtask 5

$X_i = 257$



Subtask 5: Dynamic Programming

- Input: a N×M cloth with no hole, K=1.

- Hence, any rectangle is correct.
- The total number of rectangles is $\binom{N+1}{2}\binom{M+1}{2}$.

$dp(p, d, 0)$

$dp(p, d, 1)$

Subtask 5: Dynamic Programming

- Input requirement: $0<N,M\leq 500$
- C=0;
- For x_1=1 to N,
  - Let A[y]=0 for y=1, …, M;
  - For x_2=x_1 to N,
    - Let $A[y] = A[y] + s_{x_2,y}$ for y=1,…,M;
    - Let z_1, z_2, …, z_q be the length of consecutive zeroes in A[1..M];
    - $\Delta = \left\lceil \frac{K}{x_2 - x_1 + 1} \right\rceil$; $C = C + \binom{z_1 + 2 - \Delta}{2} + \cdots + \binom{z_q + 2 - \Delta}{2}$;
- Report C;

Subtask 5: Dynamic Programming

- $\binom{1+2-1}{2} = 1$

$dp(p, d, 0)$

Subtask 5: Dynamic Programming

State

- $dp(p, d, isBound)$
  - no. of lucky numbers with ***p*** digits and ***d*** as first digit
  - $isBound$ indicates whether the **next digit is bounded by X_i**

Transitions (Yellow Boxes) $dp(p, d, 1)$

Let n = $p - 1^{th}\ digit\ of\ Xi$ (next digit)

- $dp(p, d, \mathbf{1})$=
  - if **d** is 1
    - $\sum_i dp(p-1, i, 0)$ where $i \in \{0, 1, 2, 5, 6, 7, 8, 9\}\ and\ i < n +$
    - $dp(p-1, n, 1)$ if $n \in \{0, 1, 2, 5, 6, 7, 8, 9\}$
  - if **d** is not 1
    - $\sum_i dp(p-1, i, 0)$ where $i \in \{0, 1, 2, 3, 5, 6, 7, 8, 9\}, and\ i < p - 1^{th}\ digit\ of\ Xi +$
    - $dp(p-1, n, 1)$ if $n \in \{0, 1, 2, 3, 5, 6, 7, 8, 9\}$

Subtask 5: Dynamic Programming

- Given the shape of (x,y), we can compute the shape of (x+1,y).
- Each step must remove $O(N)$ limiters and add one to the list:

Subtask 6: Binary Search the Answer

Subtask	Marks	N	T_i	X_i
6	17	$0 < N \leq 100,000$	$T_i = 1$ or 2	$0 < X_i \leq 10^{16}$

- For $T_i = 1$
  - Use Subtask 5's solution
- For $T_i = 2$
  - **Binary Search the Answer**
    - Guess a certain floor in the lucky numbering scheme
    - Convert it to conventional to *check*
    - If guess is too high, the answer must be lower
    - If guess is too low, the answer must be higher
  - Verify using Subtask 5's solution (but with inputs up to $7.7*10^{17}$)
    - 18 digits total

Subtask 6

- Subtask 5's solution might not be fast enough for Subtask 6
  - Current Runtime: O(N * states * avg. transition * $\log_2 10^{18}$)
  - Estimated iterations: 100000 * (18 * 10 * 2) * 10 * 60 = **2.16×10^{10} iterations**

- Constant time optimisations *might* be required
  - Reduction of DP states
  - Reduction of DP transition
  - Reduce recalculation of DP states

Subtask 6

Observation 1

- $dp(p, d, 0)$ states do not need to be recomputed for every X_i
  - "Blue" states are not bounded by X_i
- Only $\log_{10} X_i + 1$ states to calculate per X_i
  - "Yellow" states
- $dp(p, d, 0)$ states can be pre-computed at the start of the program

Subtask 6

- $K' = 2 \Rightarrow K = 3$

Subtask 6

Observation 2

- Transition for $dp(p, d, 1)$ states can be reduced to a linear combination of
  - $dp(p-1, 0, 0)$
  - $dp(p-1, 1, 0)$
  - $dp(p-1, 0, 1)$
  - $dp(p-1, 1, 1)$
- Number of states can be reduced
  - p from 1 to 18
  - d = 0 or 1
  - isBound = 0 or 1

Subtask 6

- Constant time optimisation reduces runtime to the following
  - Runtime: O(N * states * avg. transition * $\log_2 10^{18}$)
  - Estimated iterations: 100000 * 18 * 4 * 60 = **4.32 x 10^8 iterations**

- Disclaimer
  - It is possible to obtain full marks for this problem via other means of reducing constant time