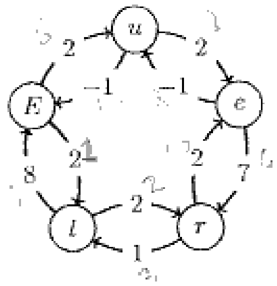
ALGORITMOS Y ESTRUCTURAS DE DATOS III - 1er Parcial

Fecha examen: 04-OCT-2017 / Fecha notas: a determinar

Completar:	Nº Orden: 21		Cant. hojas[1]: 11
	Nota (Nº)	Nota (Letras)	Docente
No completar:	9,50	nueve con 50/100	

1. Utilizar un algoritmo eficiente visto en clase para calcular los caminos mínimos desde el vértice u hacia todos los vértices del digrafo que aparece en la figura. Presentar pseudocódigo del algoritmo y realizar un seguimiento del mismo. Hecho esto, si se ordenan los vértices de acuerdo a su distancia al vértice inicial, podrá leerse el apellido del matemático suizo que descubrió la relación entre la exponenciación para números complejos y las funciones trigonométricas, uno de cuyos casos particulares es la igualdad $e^{\pi i} + 1 = 0$. 2



2. Sea G un grafo de n vértices tal que para todo vértice v se cumple que $d(v) \geq (n-1)/2$. Demostrar que G es conexo. 2

3. Un grafo es d-numerable ($d \in \mathbb{Z}_{>0}$) si y sólo si todos sus vértices tienen grado d y es posible asignar un número entero entre 1 y d a cada vértice de manera tal que para todo vértice sus vecinos tengan números diferentes entre sí. Una d-numeración es la mencionada asignación. Un grafo es d-numerado si y sólo si tiene asociada una d-numeración.

   Sea G un grafo de n vértices.

   (a) Demostrar que si G es d-numerado entonces para cada entero entre 1 y d existen al menos dos vértices adyacentes que tienen asignado ese entero. Deducir que si G es d-numerable entonces $n > 2d$. 0.

   (b) Demostrar que si G es d-numerado entonces para cada entero entre 1 y d hay exactamente n/d vértices que tienen asignado ese entero. Deducir que si G es d-numerable entonces n es múltiplo de d. 0.

   (c) Demostrar que si G es d-numerable entonces n es múltiplo de 2. 0.

4. Un grafo simple se dice 1-árbol si es un árbol con un eje agregado.

   Sea G un 1-árbol de n vértices con pesos asociados a sus ejes. Diseñar un algoritmo de complejidad $O(n)$ que determine el peso de un árbol generador mínimo de G. Mostrar que el algoritmo propuesto es correcto y determinar su complejidad. Justificar. 2

5. En una materia de la facultad están desarrollando un nuevo videojuego para enseñar programación dinámica. El videojuego es una adaptación del clásico Strikers 1945, que por algún motivo se va a llamar Dijkstrers 1930. Cada nivel del juego está dado por una cantidad de disparos disponibles y una matriz. El jugador maneja un avión propio que inicialmente se ubica en la posición inferior izquierda de la matriz. En cada una de las otras posiciones puede haber un avión enemigo. Cada segundo el avión propio se mueve una fila hacia arriba, y el jugador puede elegir que se mantenga en la misma columna, o que simultáneamente se mueva una columna a la izquierda o a la derecha, siempre dentro de los límites de la matriz. Si el avión propio y un avión enemigo ocupan la misma posición, el jugador puede destruir al avión enemigo efectuando un disparo; caso contrario los dos aviones colisionan y se pierde el nivel. Para completar el nivel, el avión propio debe llegar a la fila superior sin haber colisionado. El objetivo es completar el nivel habiendo destruído la máxima cantidad posible de aviones enemigos con los disparos disponibles. Diseñar un algoritmo eficiente que decida si el nivel puede ser completado y en tal caso indique esa cantidad. La entrada del algoritmo es la cantidad $d \geq 1$ de disparos disponibles y la matriz de f filas y c columnas. El algoritmo debe tener complejidad $O(dfc)$. Mostrar que el algoritmo propuesto es correcto y determinar su complejidad. Justificar. En los siguientes ejemplos el avión propio se representa con $\triangle$ y cada avión enemigo con $\triangledown$. 2

$d = 1$

$\triangledown$	$\triangledown$	$\triangledown$
$\triangledown$	$\triangledown$	$\triangledown$
$\triangle$	$\triangledown$	$\triangledown$

imposible

$d = 1$

$\triangledown$	$\triangledown$	
$\triangle$	$\triangledown$	

max = 1

$d = 1$

	$\triangledown$	
	$\triangledown$	
$\triangle$		

max = 1

$d = 314$

		$\triangledown$
	$\triangledown$	
$\triangle$		

max = 2

$d = 271$

	$\triangledown$	
		$\triangledown$
$\triangle$		

max = 1

SUGERENCIA: Definir $f(i, j, k)$ como la máxima cantidad de aviones enemigos destruídos cuando el avión propio está en la posición (i, j) de la matriz y le quedan k disparos disponibles, o $-\infty$ si eso no es posible.

[1] Incluyendo a esta hoja. Entregar esta hoja junto al examen.

1/10
04/10/17

① Como el grafo tiene ejes negativos, utilizaré el algoritmo de Bellman-Ford, de complejidad $O(n.m)$, con $n = \#V$ y $m = \#X$, siendo $G = (V, X)$ el grafo.

<u>Bellman-Ford</u> $(G = (V, X), u)$

$\pi = [\infty \dots \infty]$ // largo "n" ← $\pi[u] = 0$

For $i = 1$ to n

  For $e = (v_1, v_2) \in X$

   L $\pi[v_1] = \min(\pi[v_1], \pi[v_2] + l(e))$

  if no hubo cambios en π; break

devolver π

Falta el vector de predecesores para devolver los caminos

<u>Seguimiento del algoritmo</u>: $\pi = [\text{"E"}, \text{"u"}, \text{"l"}, \text{"e"}, \text{"r"}]$
  1 2 3 4 5

- $i = 1$ $(\pi = [\infty, 0, \infty, \infty, \infty]$

  $e = (u, e) \rightsquigarrow \pi = [\infty, 0, \infty, \underline{2}, \infty]$

  $e = (e, r) \rightsquigarrow \pi = [\infty, 0, \infty, 2, \underline{9}]$

  $e = (u, E) \rightsquigarrow \pi = [\underline{-1}, 0, \infty, 2, 9]$

  $e = (E, l) \rightsquigarrow \pi = [-1, 0, \infty, 2, 9]$

  $e = (e, u) \rightsquigarrow \pi = [-1, 0, \infty, 2, 9]$

  $e = (r, e) \rightsquigarrow \pi = [-1, 0, \infty, 2, 9]$

  $e = (E, u) \rightsquigarrow \pi = [-1, 0, \infty, 2, 9]$

Euler

$e=(\ell,E) \rightsquigarrow \pi = [-1,0,\infty,2,9]$

$e=(\ell,r) \rightsquigarrow \pi = [-1,0,\infty,2,9]$

$e=(r,\ell) \rightsquigarrow \pi = [-1,0,\underline{10},2,9]$

- $i=2$ $(\pi = [-1,0,10,2,9])$

$e=(E,\ell) \rightsquigarrow \pi = [-1,0,\underline{1},2,9]$

$e=(\ell,r) \rightsquigarrow \pi = [-1,0,1,2,\underline{3}]$

$e=(r,\ell) \rightsquigarrow \pi = [-1,0,1,2,3]$

$e=(e,r) \rightsquigarrow \pi = [-1,0,1,2,3]$

$e=(\ell,E) \rightsquigarrow \pi =$ "

$e=(E,u) \rightsquigarrow \pi =$ "

$e=(u,e) \rightsquigarrow \pi =$ "

$e=(e,u) \rightsquigarrow \pi =$ "

$e=(u,E) \rightsquigarrow \pi =$ "

$e=(r,e) \rightsquigarrow \pi =$ "

- $i=3$

// Con $\pi = [-1,0,1,2,3]$ ya vimos que todos estos no cambian nada (solo para ahorrar escribir, el código es igual).

⋮

$e=(E,\ell) \rightsquigarrow \pi = [-1,0,1,2,3]$

$e=(\ell,r) \rightsquigarrow \pi = [-1,0,1,2,3]$

$\Rightarrow$ En este ciclo no hubo cambios $\rightsquigarrow$ BREAK.

$$\boxed{\text{Resultado: } \pi = [-1,0,1,2,3] \quad \begin{array}{ccccc} \uparrow & \uparrow & \uparrow & \uparrow & \uparrow \\ E & u & \ell & e & r \end{array}}$$

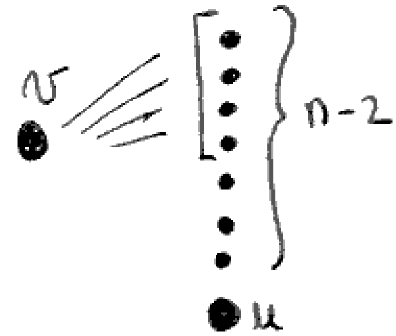
② $G=(V,X)$ grafo de "n" vértices / $\forall v \in V$ se cumple que $d(v) \geq (n-1)/2$. Demostrar que es conexo.

Queremos ver que dados $v, u \in V$ nodos de G, existe un camino entre ellos.

Sean $v, u \in V$. Si $(v,u) \in X$, ya tenemos un camino ✓. Si $(v,u) \notin X$:

- $\exists\, w \in V / (v,w) \in X \wedge (w,u) \in X$
  $\Rightarrow$ Hay un camino: $(v,w), (w,u)$ ✓
- $\nexists\, w \in V / (v,w) \in X \wedge (w,u) \in X$.

Es decir, v y u no comparten vecinos. Esto implica que $d(u) + d(v) \leq n-2$, pues si no habría alguno de los $n-2$ nodos restantes que se conectaría a los dos. Esquema:



$$\Rightarrow d(u) + d(v) \leq n-2 \Leftrightarrow d(u) \leq n-2-d(v)$$
$$\Rightarrow d(u) \leq n-2-(n-1)/2 = 2(n-1)/2 - (n-1)/2 - 1$$
$$= \frac{n-1}{2} - 1 \Rightarrow d(u) \leq \frac{n-1}{2} - 1 \quad \underline{\text{ABSURDO!}}$$

Luego, debe existir un camino entre u, v cualesquiera. ✓

3/10
04/10/17

Nota: 7

③ G con "n" vértices.

a) G d-numerado.

Sea "v" un nodo de G.

Sea $i \in [\![1;d]\!]$ el número que le corresponde a "v" en la d-numeración. Sabemos que v tiene "d" vecinos, todos de distinto número. Si hay que diferenciar d cosas con números en $[\![1;d]\!]$, que son d, alguno debe ser "i".

$\Rightarrow$ Hallamos para cualquier v un vecino con el mismo número. (de número i) ✓

Si por cada $i \in [\![1,d]\!]$ hay al menos 2 nodos con ese número, y se asignan todos, luego debe haber al menos 2.d nodos.

$\Rightarrow$ $\underline{n \geq 2d}$. ✓

<u>Nota importante</u>: para ambas demos supuse trivial el hecho de que $\forall i \in [\![1,d]\!]$ exista un nodo con ese número. Si no hubiera, sería imposible numerar "d" vecinos de un nodo. ✓

Si, ERA IMPORTANTE!

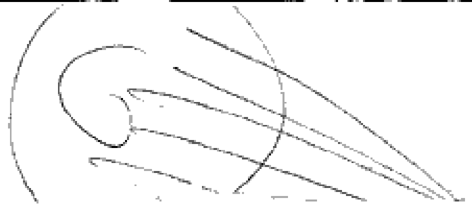
b) Por cada uno de los nodos corresponde un conjunto de vecinos de números $1 \ldots d$.
Como hay n nodos, tendremos "n" conjuntos de este tipo : $\{v_1^i \ldots v_d^i\}$, que corresponden a los d vecinos del nodo "i".

Nótese que un nodo v no puede aparecer en más de "d" de estos ~~conjuntos~~ "n" conjuntos, pues significaría que tiene más de d vecinos. De hecho, aparecerá exactamente d veces: una vez por cada uno de sus vecinos, que tienen cierto conjunto asociado.

Si debemos formar "n" conjuntos, y un nodo con número "i" está solamente en d de ellos, necesitaremos tantos como n/d nodos de número "i", pues debe haber un número "i" en cada conjunto.

Si entonces n no fuera múltiplo de d, habría nodos de número "i" que no estarían en exactamente "d" conjuntos de ~~vecinos~~ vecinos.
· Si estuviera en menos $\Rightarrow$ ABSURDO! No tiene d vecinos
Si estuviera en más $\Rightarrow$ ABSURDO! No tiene d vecinos.

Luego, n debe ser múltiplo de d.



4/10
04/10/17

c) Si "n" fuera impar, como por "b)", "n" es múltiplo de "d", luego "d" tiene que ser impar, por el teorema fundamental del álgebra.
Sabemos que todos los nodos tienen grado "d". Luego, la suma de los grados, que sabemos es el doble de la cantidad de ejes, es:

$$\sum_{v \in V} d(v) = d \cdot n = 2 \cdot m \; ; \text{ siendo } m = \# X$$

Pero si n es impar, d es impar, multiplicados no pueden dar un par "2.m". Luego, "n" debe ser par.

¡Muy bien!

5/10
04/10/17

$\boxed{2}$

④ Cosas que sabemos:

- G es 1-árbol ⇒ G tiene $n-1+1$ ejes
- Si se le saca un eje (el/los adecuados), G se convierte en árbol.
- $m = \#X \in O(n)$. De hecho, $n=m$
  ← Esto lo usaremos mucho en la complejidad

La idea del algoritmo será hallar un árbol cualquiera, y luego calcular los posibles pesos de los árboles intercambiando los ejes del <u>único</u> ciclo que tiene (visto en teórica y práctica; si tuviera más de uno no bastaría con sacarle un eje para tener un árbol), obteniendo así los pesos de todos los AG posibles.

Pseudocódigo

```
O(n)  [ • enÁrbol ← [0 ... 0] // largo "n", para los ejes
      [ • numerar los ejes de 1 a n // O(n)
      [ • ejes ← [e_1, ... e_n] // e_i ∈ X
O(n+n)=O(n) → [ • árbol ← BFS(G) // devuelve la lista de ejes del árbol que forma
O(n)  [ Para e ∈ árbol
      [  | • enÁrbol[e] ← 1 // usando el id asignado
      [ FinPara
O(n)  [ Para i = 1 ... n
      [  | si enÁrbol[i] ≠ 0
      [  |   | • e ← ejes[i] // me guardo el eje que no está en el árbol
      [  | finsi
      [ FinPara
```

↘ sigue...

$O(n)$ [• camino ← caminoEntre(árbol, u, v) // con $e=(u,v)$

$O(n)$ [• e_máx ← máxPeso(camino) // guardo la arista del ciclo de máximo peso

$O(n)$ [• res ← sumaPeso(árbol)

$O(1)$ si $l(e\_máx) > l(e)$ // si hay una arista del ciclo mayor, la mejoro

  • res ← res − $l(e\_máx) + l(e)$

fin

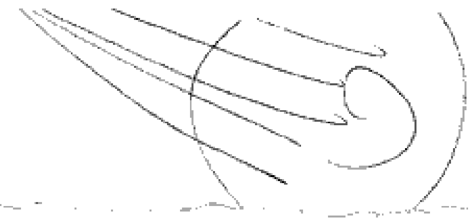
devolver res

fin

⇒ Complejidad total: $O(n)$

Detalle:

- Las inicializaciones y recorridos de arreglos de largo n se hacen ~~cantidades~~ una cantidad acotada de veces.
- BFS, devolviendo la lista de ejes que visitó, es $O(n+m) = O(n+n) = O(n)$.
- El camino entre u y v (que como es un árbol es único) puede calcularse en orden lineal con BFS.
- El máximo peso del camino encontrado se encuentra en $O(n)$, pues son a lo sumo "n" ejes.
- El peso total de los ejes en árbol también es $O(n-1) = O(n)$



6/10

04/10/17

Correctitud:

Sea T el AGM de G.

Sea e la única arista que deja afuera el BFS (sabemos que es única porque G es un árbol con un solo eje de más), y C el único ciclo de G.

Sabemos que T tendrá todos los ejes de G menos uno, que está en el ciclo C. Si la arista que deja afuera estuviera en C, el "árbol" de BFS no sería un árbol, pues tendría un ciclo.

Si el eje de C que no está en T no fuera el de máximo peso entre todos los de C, poner el eje que quedó afuera en T y sacar el máximo de C, rompiendo el único ciclo, nos daría un AG de menor peso, por lo que T no sería AGM. Luego, el peso de T puede escribirse como:

- $\ell(T) = \ell(G) - \ell(e)$

Donde e es la arista de máximo peso en C

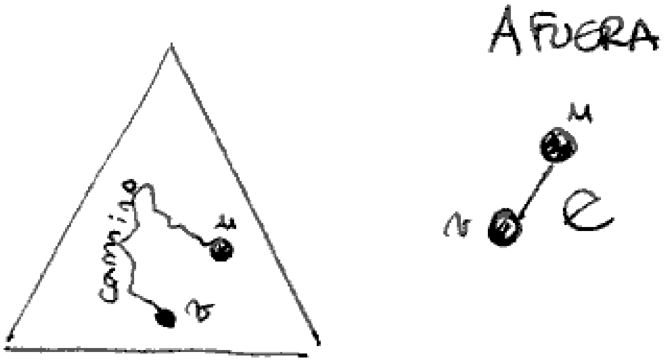
El algoritmo propuesto calcula el peso total del árbol de BFS, y luego verifica si la arista de longitud máxima de C que está en el árbol de BFS es mayor que la que quedó afuera.

Si es así, el AGM es el que resulta de intercambiar la máxima arista del ciclo, que está en el árbol, por la que quedó afuera.

Si la arista que quedó afuera era la máxima del ciclo, el AGM es el encontrado por BFS.

Nótese que para trabajar con las aristas del ciclo C el algoritmo busca el <u>único</u> camino en el árbol de BFS (por ser árbol) entre los extremos del eje que quedó afuera.

Esquema:



<u>ÁRBOL BFS</u>

Nótese que camino + e = C.

Auxiliares

```
▶ CaminoEntre(árbol, u, v) {
    ady ← lista de adyacencia (árbol) // O(m+n) = O(n)
    padres ← [0 . . . . 0] // largo "cant. nodos".
    cola ← {u}
    while ¬ cola.vacía()
        sig ← cola.desencolar
        for w ∈ ady_i(sig)                      ]
            if padres[w] ≠ 0                    ]
                padre[w] ← sig                  ]  O(m)
                cola.encolar(w)                 ]
            fi                                  ]
        endfor
    end while

O(n) [ • iterar en "padres" reconstruyendo el camino
       u-v, empezando desde v y terminando en u.
}
```

Complejidad: puede verse que es muy parecido a un BFS. Itera sobre todas las aristas, inicia un arreglo, y obtiene la lista de adyacencias a partir de la lista de ejes.

Recordando que esto es un árbol y $O(m) = O(n)$, la complejidad queda $O(n)$.

- SumaPeso (ejes) {

  res ← 0

  Para e ∈ ejes

  res ← res + $l(e)$

} ⇒ Comp. $O(|ejes|)$ // en nuestro caso, será "n".

- MáxPeso (ejes) {

  res ← $-\infty$

  Para e ∈ ejes

  | res ← máx(res, $l(e)$)

  finPara

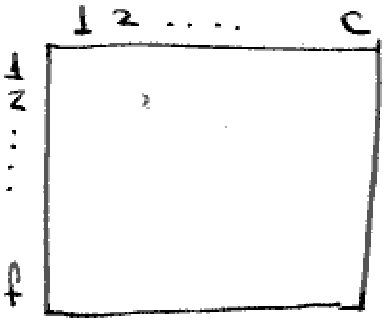
} ⇒ Comp. $O(|ejes|)$ // en nuestro caso, el camino puede tener menos de "n".

(-2,) 1

8/10
04/10/17

2

5. Tomaré la siguiente numeración de la matriz M del juego, que la supondré de booleanos:



IMPORTANTE:

Supongo que no es posible que haya un avión enemigo en (f,1), donde se empieza.

Propongo la siguiente función recursiva:

Supongo que en la función hablas de K y no de "d".

¿qué es M_{ij}?

[illegible]

$$
\bullet\ f(i,j,K) = \begin{cases}
0 & \text{si } i==f \wedge j==1 \\
-\infty & \text{si } (i \neq f \wedge d==0 \wedge M_{ij}) \vee c-j>f-i \\
1+\max\big(f(i+1,j,d-1),\ f(i+1,\min(1,j-1),d-1),\ f(i+1,\max(c,j+1),d-1)\big) & \text{si } i\neq f \wedge d>0 \wedge M_{ij} \\
\max\big(f(i+1,j,d),\ f(i+1,\min(1,j-1),d),\ f(i+1,\max(c,j+1),d)\big) & \text{si } i\neq f \wedge \neg M_{ij}
\end{cases}
$$

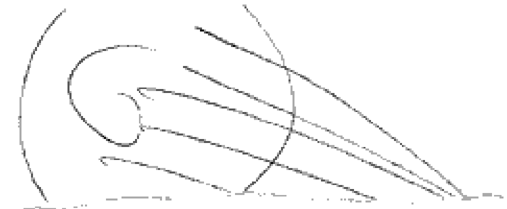
Donde $f(i,j,K)$ indica lo que la SUGERENCIA.

Nota: $c-j>f-i \Rightarrow -\infty$ porque no es posible más a la derecha que para arriba.

```
Dijkstrans (d, M, f, c) {
    D [f][c][d] ← NULL x f×c  // variable global, junto
                                con f y c
    res ← -∞

    For K = 1 to c
    |  res ← máx (res, f(1, K, d))
    endfor
    devolver res
}  // Supongo M, D-s, f y c variables globales.

f(i, j, d) {
    if  Dijd ≠ NULL
        return Dijd
    if  i == f ∧ j == 1
    |   res ← 0
    if  c-j > f-i  ∨ (d == 0 ∧ Mij)
    |   res ← -∞
    if  d > 0 ∧ Mij
    |   res ← 1 + máx ( f(i+1, j, d-1), f(i+1, mín(j-1), d-1),
    |                   f(i+1, máx(c, j+1), d-1) )
    if  ¬Mij
    |   res ← máx ( f(i+1, j, d), f(i+1, mín(j-1), d-1),
    |               f(i+1, máx (c, j+1), d-1) )
    fi
    Dijd ← res
    return res
}
```



9/10
04/10/17

Notas sobre el código:

- La función principal es Dijkstrers.
- Para simplificar el código, supongo globales varias variables; las que no dependen de una llamada en particular.
- La "matriz" D es la que usaré para guardar sub-resultados. Empieza con valores nulos, que sirven para indicar que aún no han sido calculados.

Complejidad:

Se guardan los resultados de la programación dinámica en D para no recalcularlos.
Hay menos de $f \times c \times d$ subproblemas posibles (justamente el tamaño de la "matriz").
Como cada llamada correspondiente a un subproblema tiene complejidad $O(1)$ sin contar la complejidad de sus llamadas, y teniendo en cuenta que cada subproblema se calcula una única vez, la complejidad será:

$$\text{costo subprob} \times \text{cant. subprob} = O(1) \times O(fcd) = \boxed{O(fcd)}$$

Nótese que la función principal toma el máximo de "c" llamadas. El ciclo $O(c)$ no agrega orden de complejidad ya que las llamadas usan, por supuesto, la misma "matriz" D para los subresultados.

Correctitud

(Basada en la definición recursiva de f).

- Proposición: $\forall i,j,d$, $f(i,j,d)$ devuelve la máxima cantidad de aviones enemigos destruidos cuando el avión está en (i,j) y tiene d disparos.

Casos base (inducción en i. Quedó al revés por cómo numeré la matriz)

▶ $i==f \wedge j==1$

↳ recién empieza: 0 ✓

No son casos base, pero lo puse aparte.

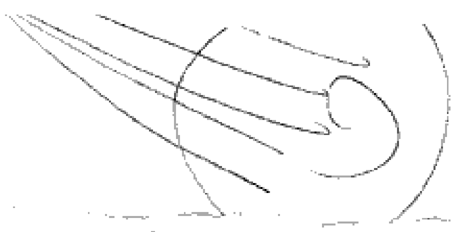
▶ $i \neq f \wedge (d==0 \wedge M_{ij}) \vee (c-j > f-i)$

$i \neq f$
↳ Si $d==0 \wedge M_{ij}$, me mata el enemigo: $-\infty$ ✓
↳ Si $c-j > f-i$, hice trampa y me moví más a la derecha que para arriba: $-\infty$ ✓

Paso Inductivo

(quedó al revés por la numeración de M_{ij})

H.I.: $\forall a,j,d$ con $a > i$, $f(a,j,d)$ devuelve la máxima cantidad de enemigos destruidos cuando el avión



10/10
04/10/17

Si $d>0$ y M_{ij} tiene un enemigo, lo podemos matar. Hasta ese momento, el máximo número de naves matadas será haber matado la de recién, y haber tomado el camino que me llevó allí en el que más naves maté.

Es decir:

$$1+\text{máx}\Big(f(i+1,j,d-1),\; f(i+1,\text{mín}(1,j-1),d-1),\; f(i+1,\text{máx}(c,j+1),d-1)\Big)$$

Pues vine de abajo $(i+1)$ y de cualquiera de las columnas de al lado o la misma, y en ese entonces tenía un disparo menos $(d-1)$.

Si M_{ij} no tiene un enemigo, en este casillero no puedo sumar nada, por lo que la mayor cantidad de naves matadas estará dada por la mayor cantidad dentro de todas las formas posibles que tenía de llegar allí:

$$\text{máx}\Big(f(i+1,j,d),\; f(i+1,\text{mín}(1,j-1),d),\; f(i+1,\text{máx}(j+1,c),d)\Big)$$

Idem antes pero no usé el disparo.

Supongamos que f devuelve la cantidad máxima de matados posibles en la posición i,j con d disparos, y la función principal toma el máximo de f(1,1,d) ... f(1,c,d), estamos obteniendo la máxima cant. de matados posibles cuando la nave llega a la fila 1, que es cuando gana.

Es decir, estamos obteniendo el resultado que se pide.

Nota: supuse que el juego termina cuando la nave llega a la última fila, sin importar la columna.