Presentación práctica de eficiencia

Asignatura: Algorítmica

Rubén Morales Pérez Francisco Javier Morales Piqueras
Bruno Santindrian Manzanedo Ignacio de Loyola Barragan Lozano Francisco Leopoldo Gallego Salido

5 de mayo de 2016

Índice

Minimizando el número de visitas al proveedor

Ordenador usado para la ejecución

Asus N56VJ
Sistema operativo: Linux mint (Rosa)
Memoria: 8GB
Procesador: Inter Core i7-4710HQ x 8
Gráficos: Nvidia geforce 750M
Tipo de SO: 64 bits
Disco: 1TB

Problema

Enunciado

Un granjero necesita disponer siempre de un determinado fertilizante. La cantidad máxima que puede almacenar la consume en r días, y antes de que eso ocurra necesita acudir a una tienda del pueblo para abastecerse. El problema es que dicha tienda tiene un horario de apertura muy irregular (solo abre determinados días). El granjero conoce los días en que abre la tienda, y desea minimizar el número de desplazamientos al pueblo para abastecerse.

Ejercicios

1. Diseñar un algoritmo greedy que determine en qué días debe acudir al pueblo a comprar fertilizante durante un periodo de tiempo determinado (por ejemplo durante el siguiente mes).
2. Demostrar que el algoritmo encuentra siempre la solución óptima.

Diseño del algoritmo

Algoritmo

Aplicamos greedy escogiendo en cada iteración el día más lejano que cumpla estas dos condiciones.

- Que diste menos de r días de la ultima reposición.
- Que la tienda esté abierta.

Representación

- Un vector de booleanos representa los días que abre la tienda.
  Por ejemplo [1, 0, 0, 1, 0, 1] significa que la tienda abre los días 1, 4 y 6.
- Una lista de enteros almacena los días que el granjero tiene que ir a comprar.

Ejemplo

L	M	X	J	V	S	D
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30					

Reserva (4 días) Tienda abierta Visita

Ejemplo

L	M	X	J	V	S	D
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30					

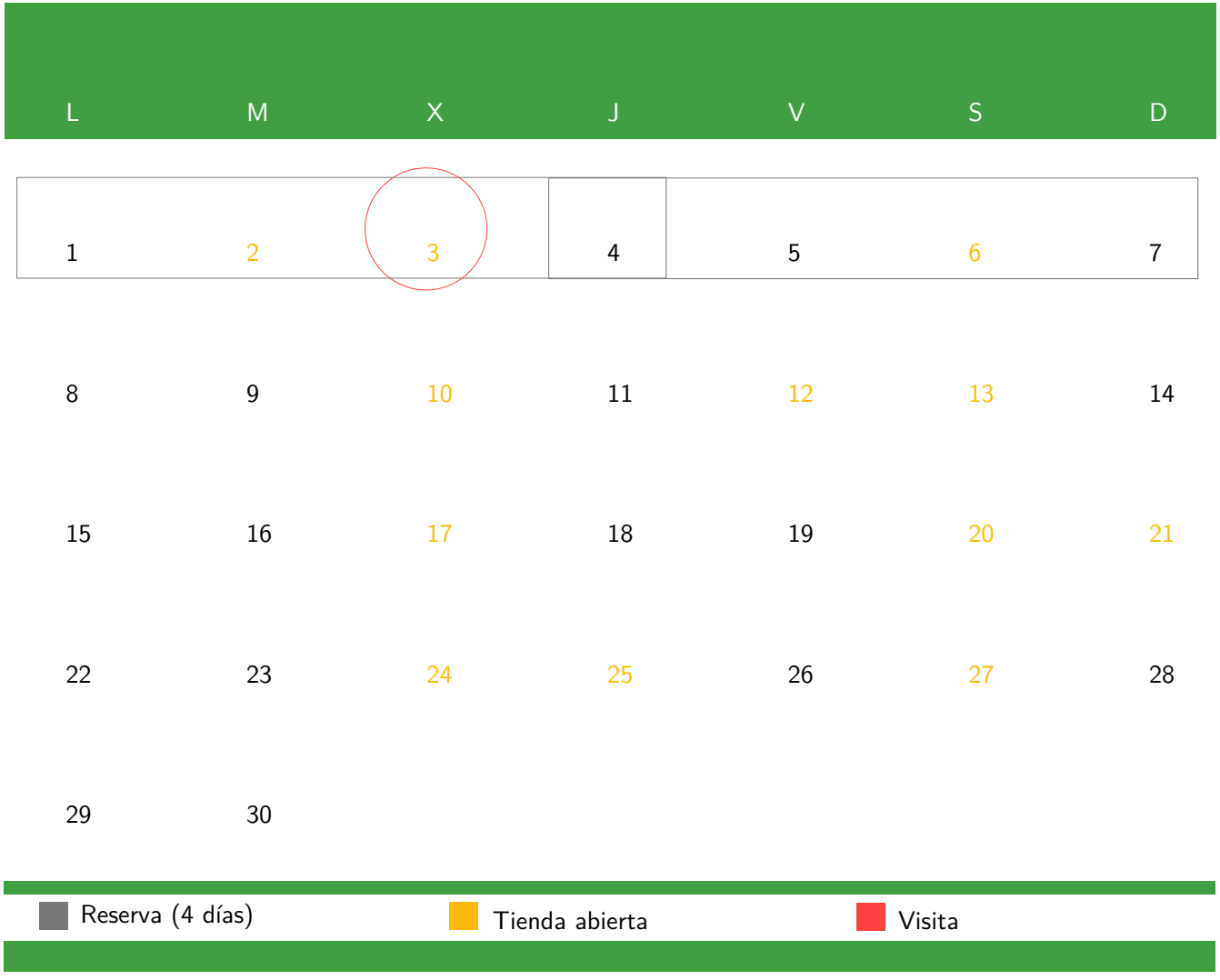
Reserva (4 días) Tienda abierta Visita

Ejemplo

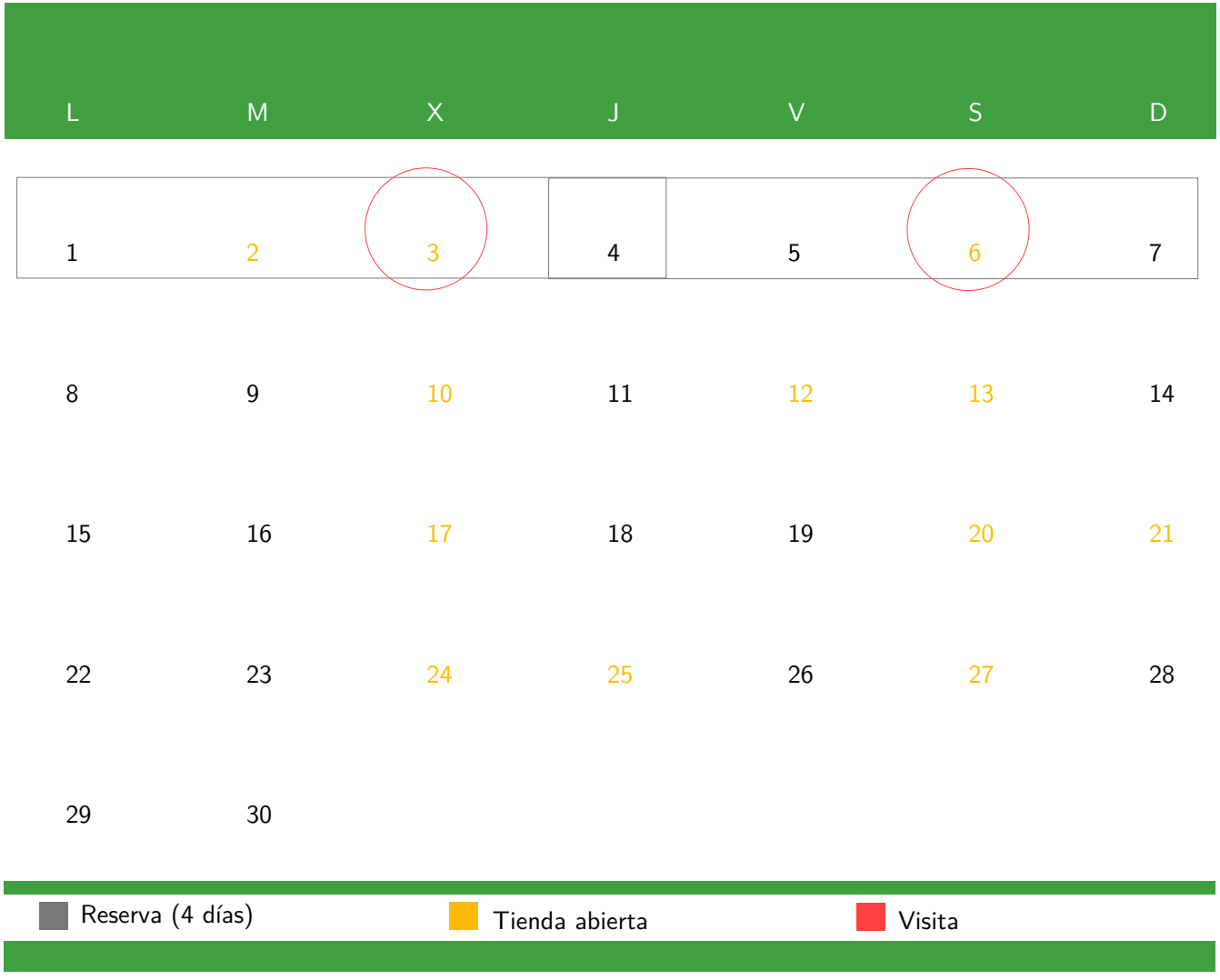
L	M	X	J	V	S	D
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30					

Reserva (4 días) Tienda abierta Visita

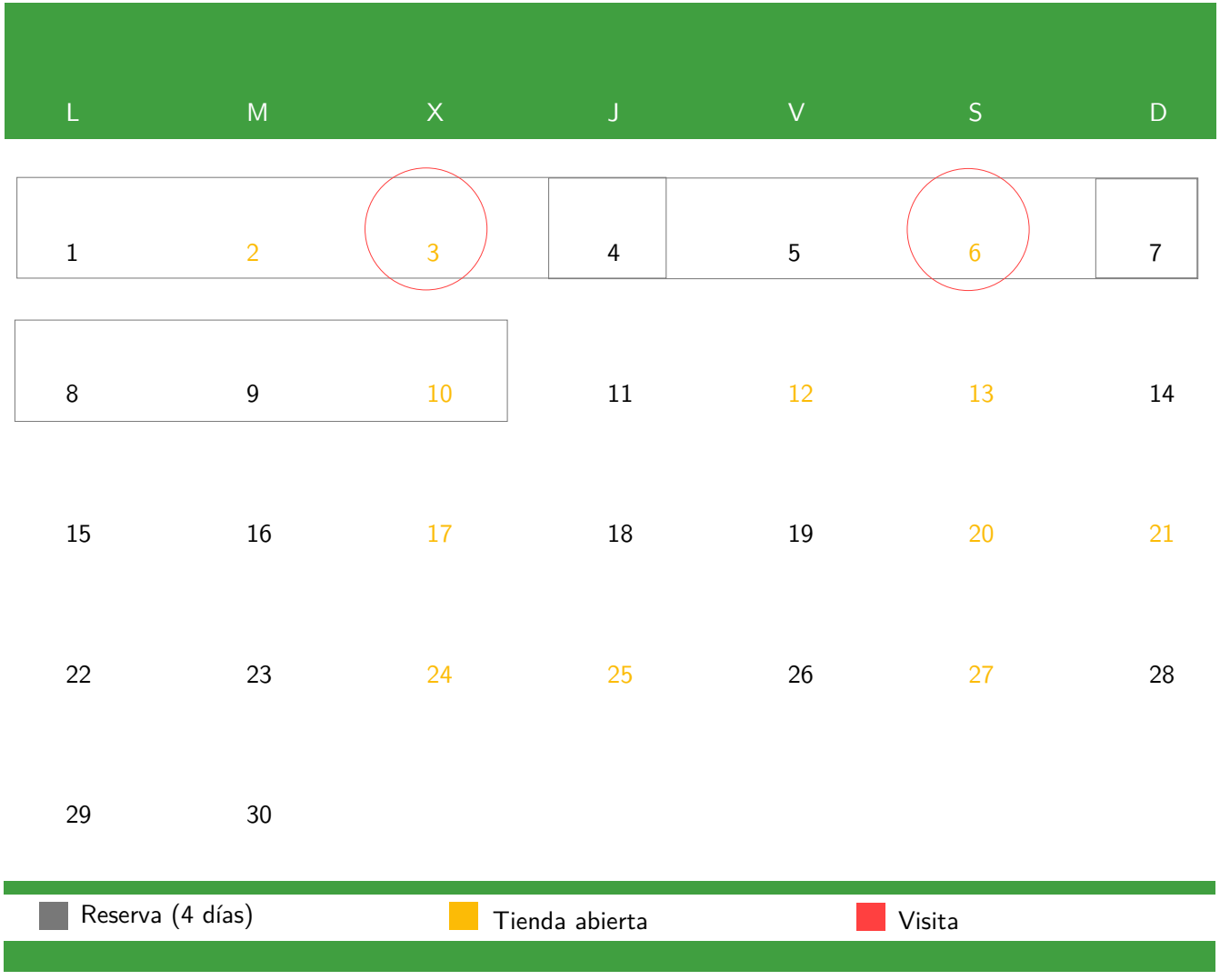
Ejemplo


L M X J V S D
1 2 3 4 5 6 7
8 9 10 11 12 13 14
15 16 17 18 19 20 21
22 23 24 25 26 27 28
29 30
Reserva (4 días)
Tienda abierta
Visita

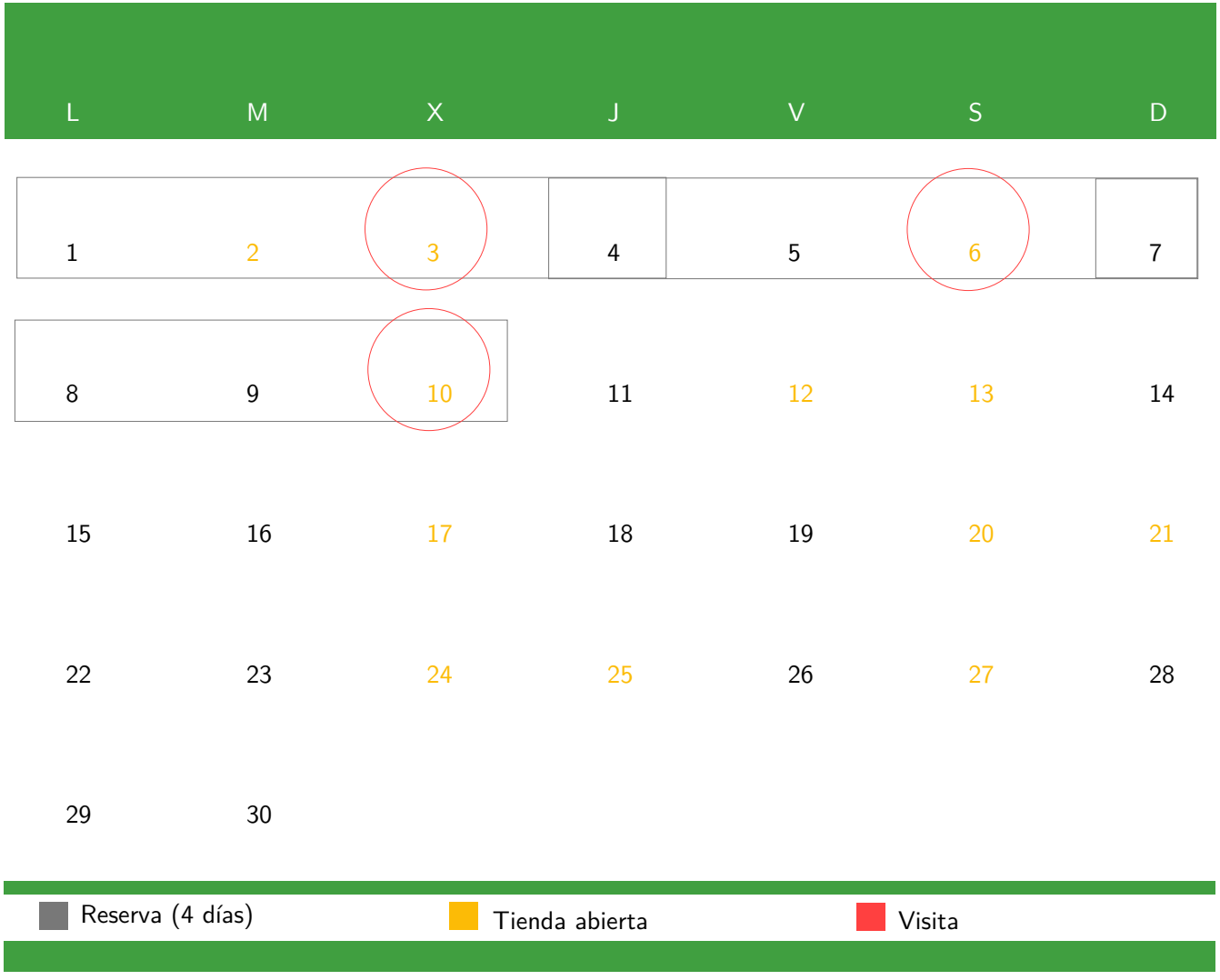
Ejemplo


L M X J V S D
1 2 3 4 5 6 7
8 9 10 11 12 13 14
15 16 17 18 19 20 21
22 23 24 25 26 27 28
29 30
Reserva (4 días)
Tienda abierta
Visita

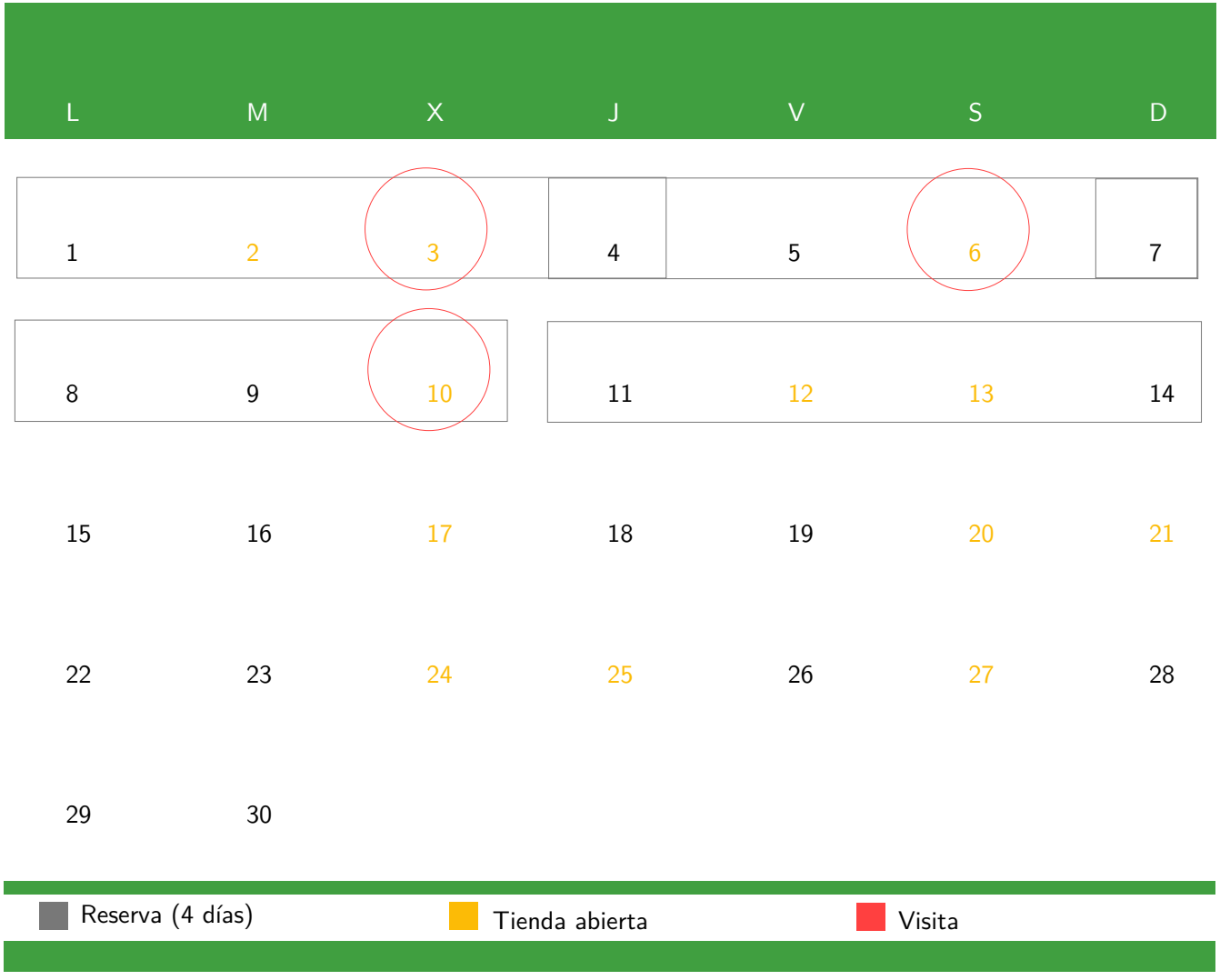
Ejemplo


L
M
X
J
V
S
D
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
Reserva (4 días)
Tienda abierta
Visita

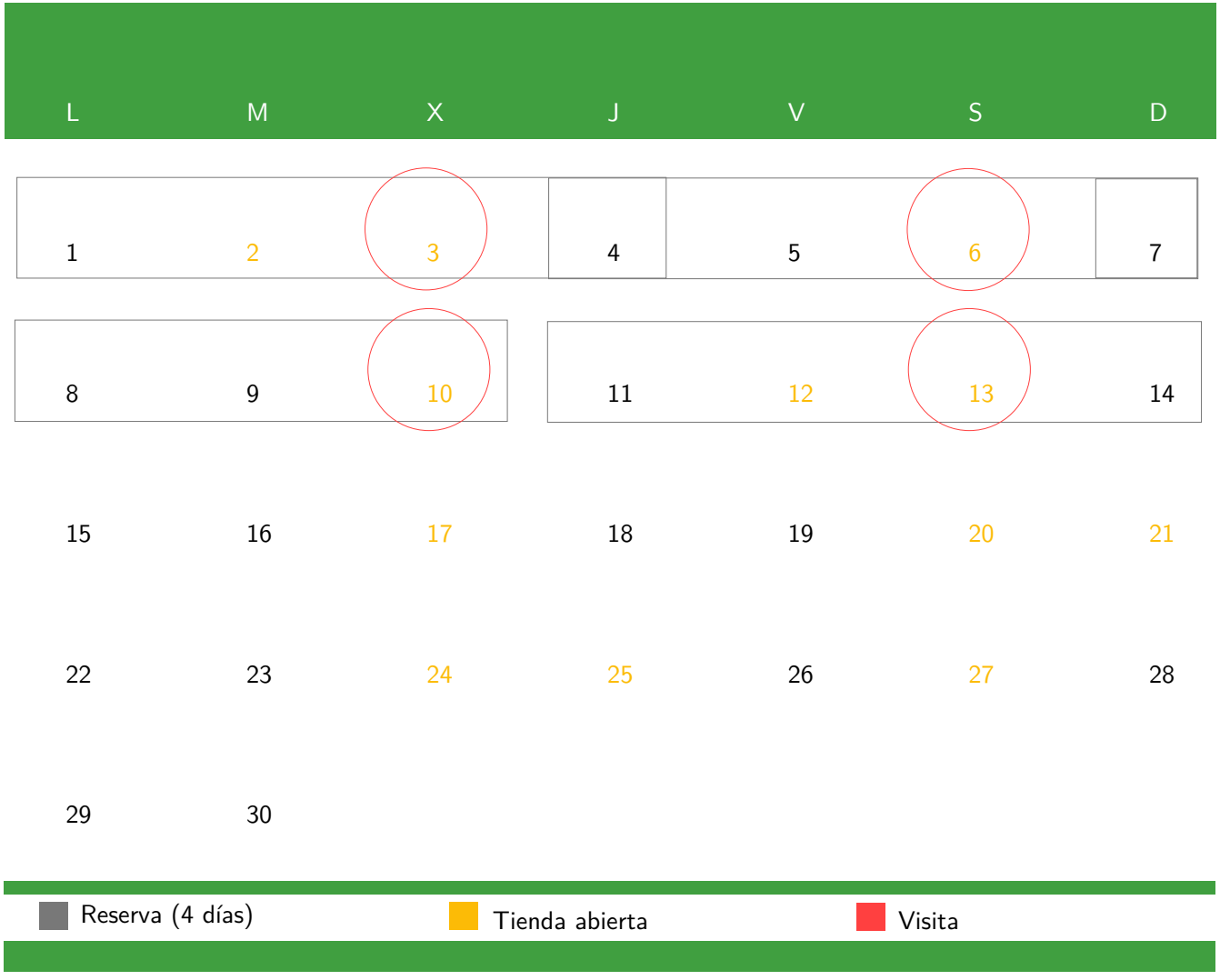
Ejemplo


L
M
X
J
V
S
D
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
Reserva (4 días)
Tienda abierta
Visita

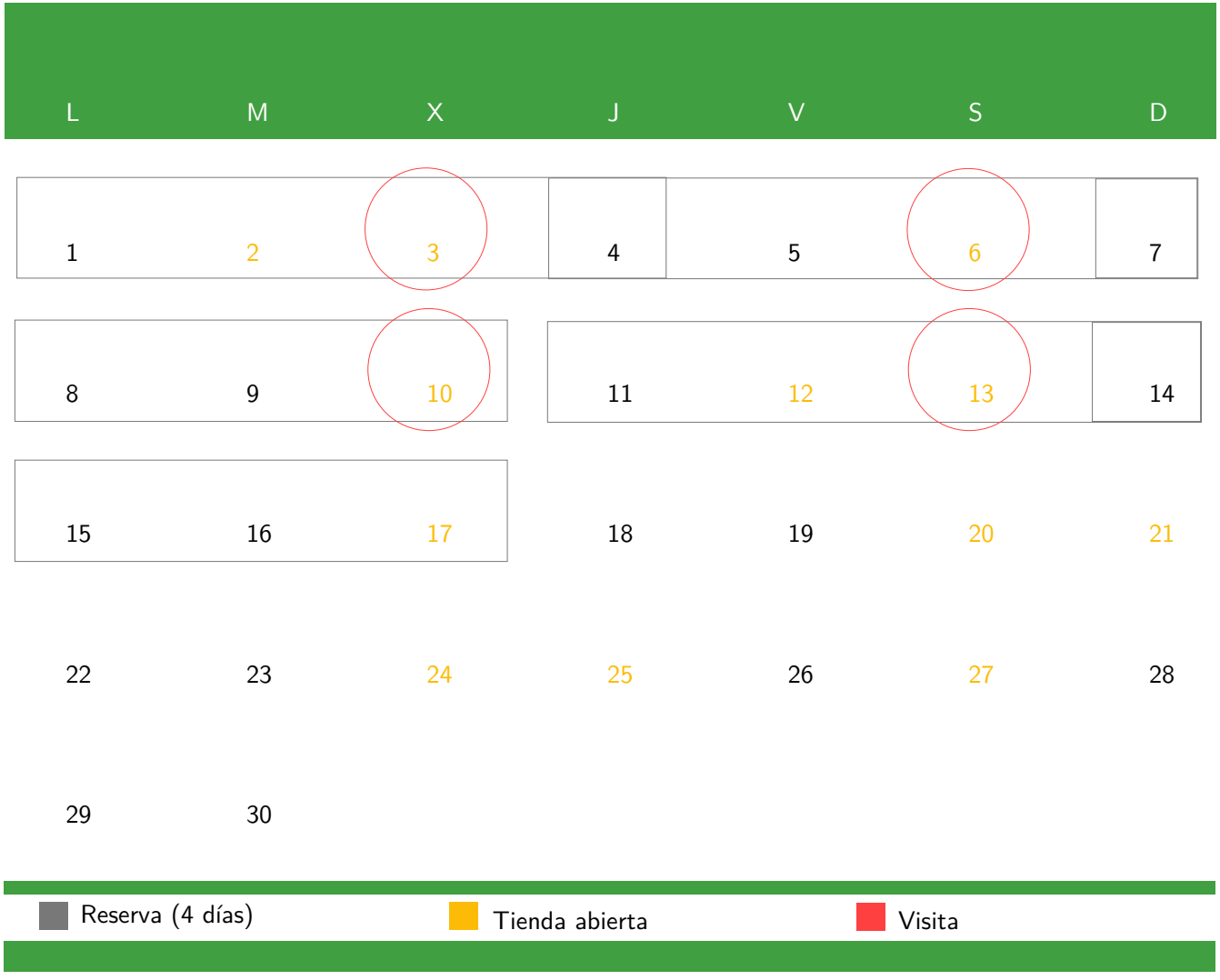
Ejemplo


L
M
X
J
V
S
D
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
Reserva (4 días)
Tienda abierta
Visita

Ejemplo


L
M
X
J
V
S
D
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
Reserva (4 días)
Tienda abierta
Visita

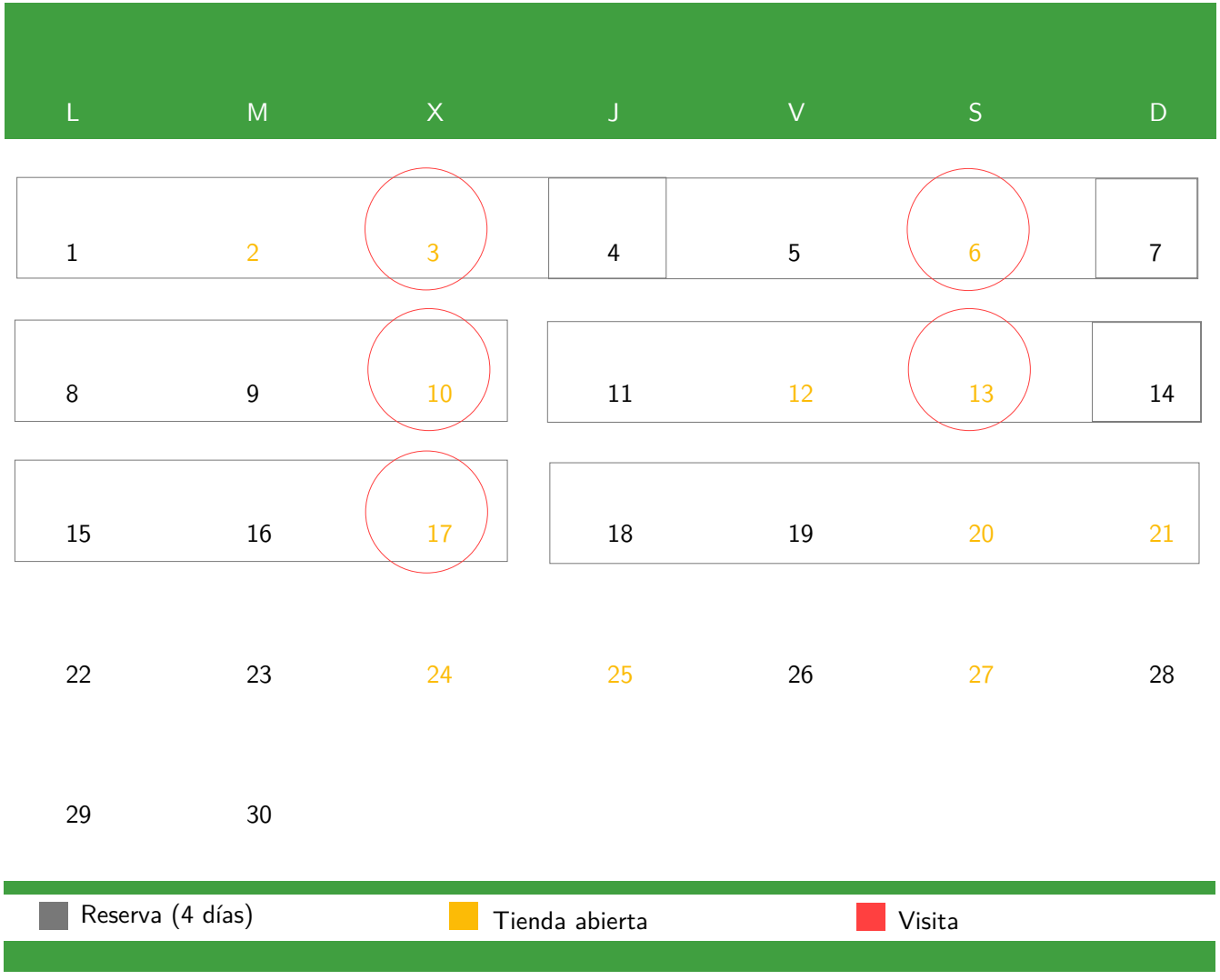
Ejemplo


L
M
X
J
V
S
D
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
Reserva (4 días)
Tienda abierta
Visita

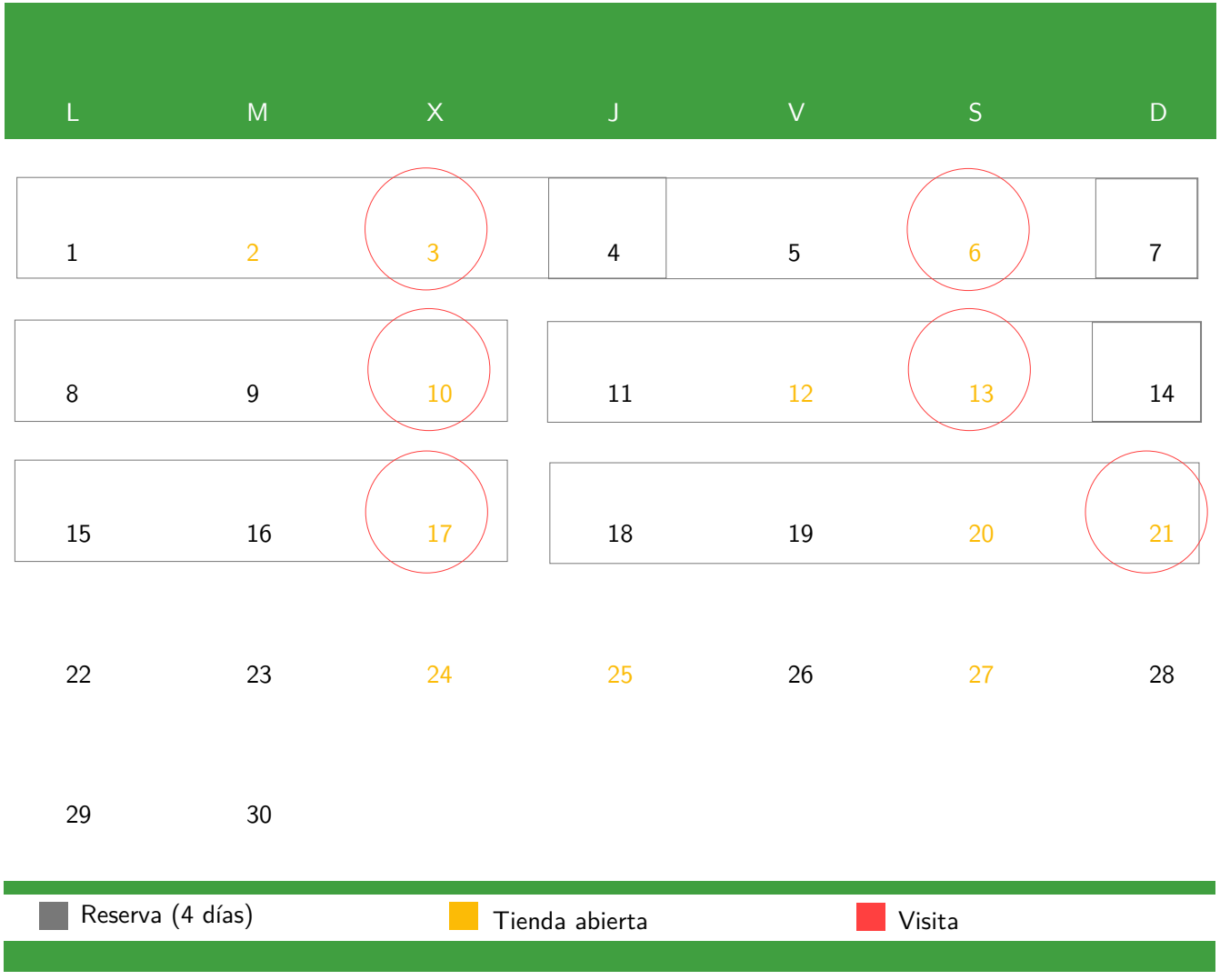
Ejemplo


L
M
X
J
V
S
D
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
Reserva (4 días)
Tienda abierta
Visita

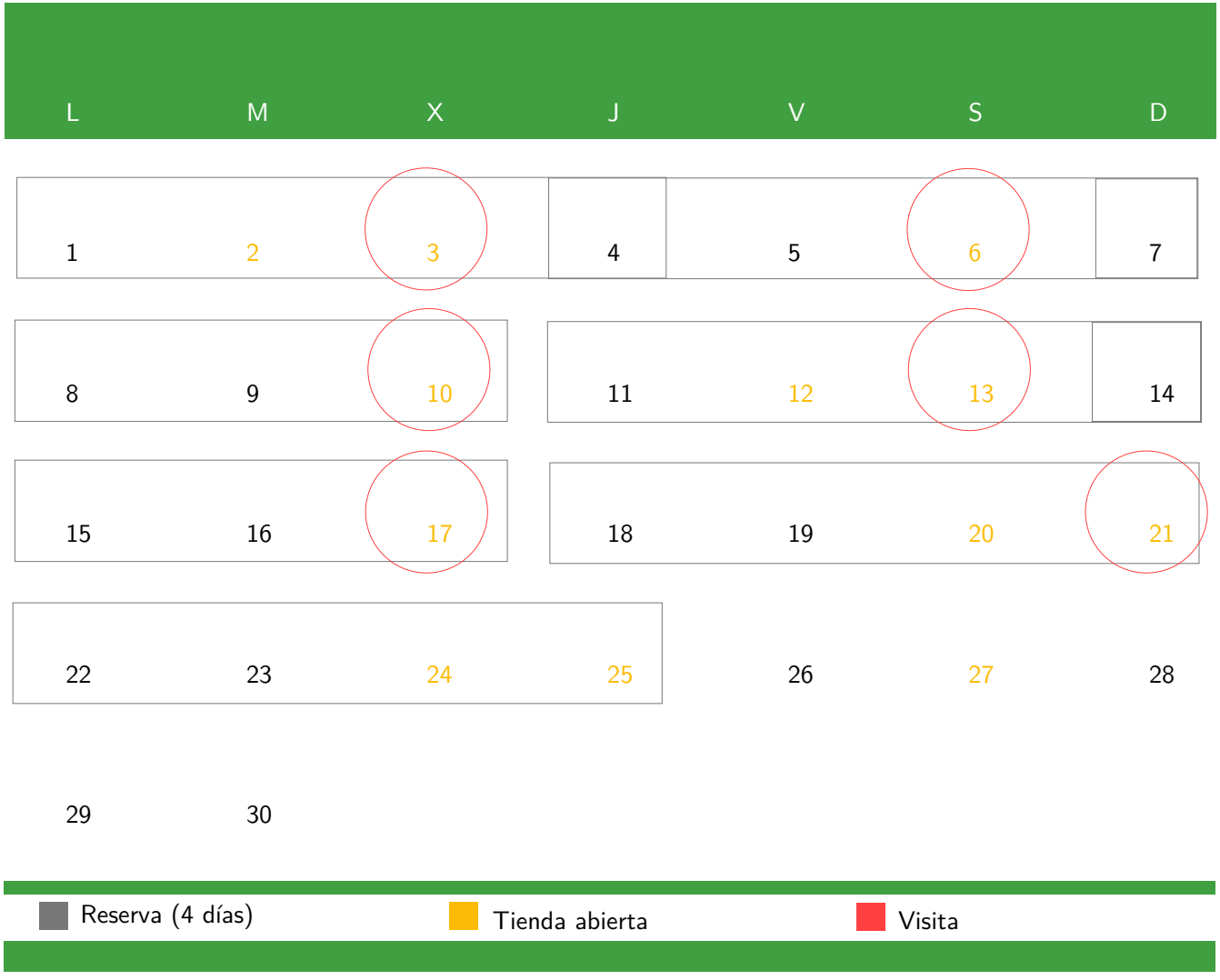
Ejemplo


L
M
X
J
V
S
D
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
Reserva (4 días)
Tienda abierta
Visita

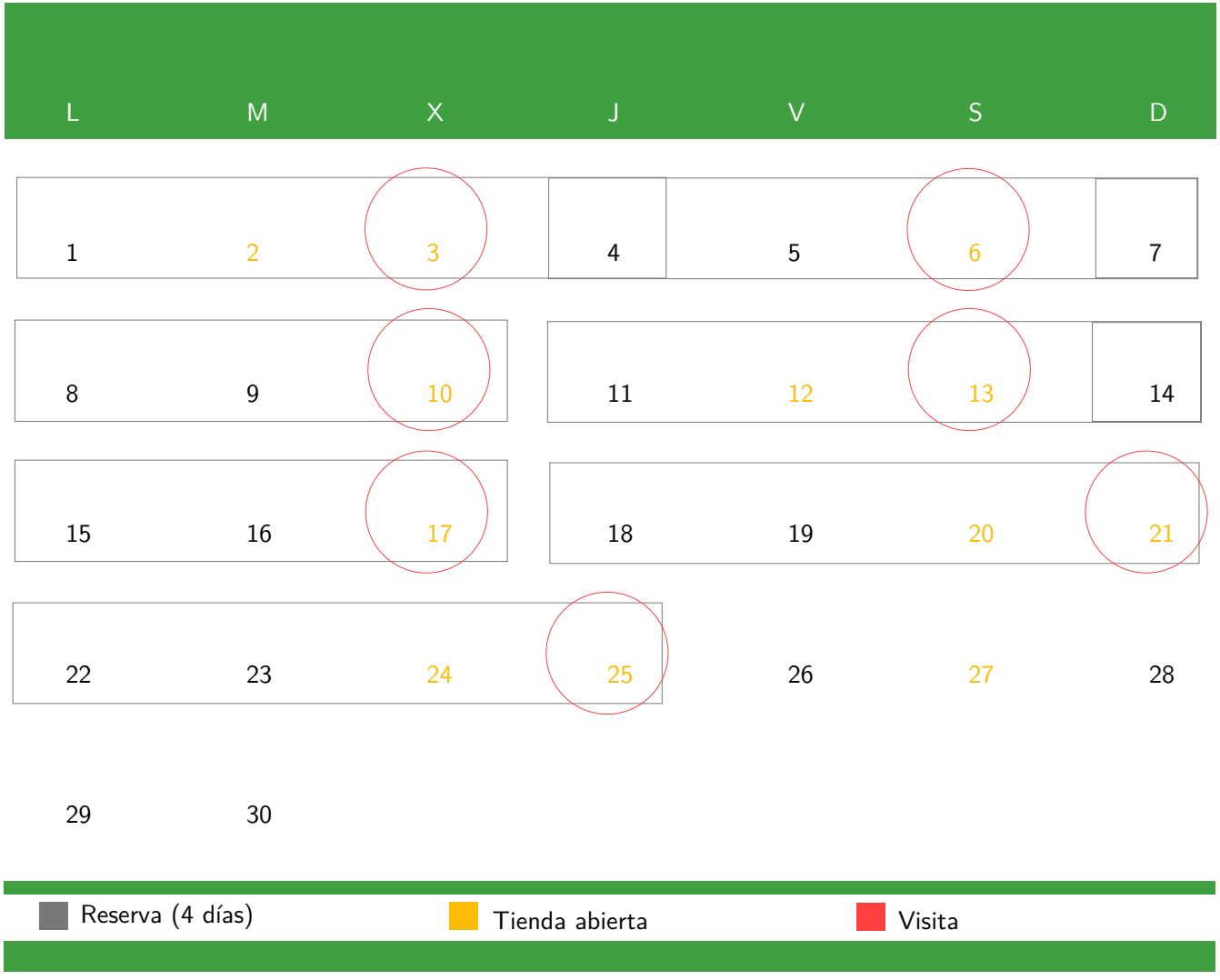
Ejemplo


L
M
X
J
V
S
D
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
Reserva (4 días)
Tienda abierta
Visita

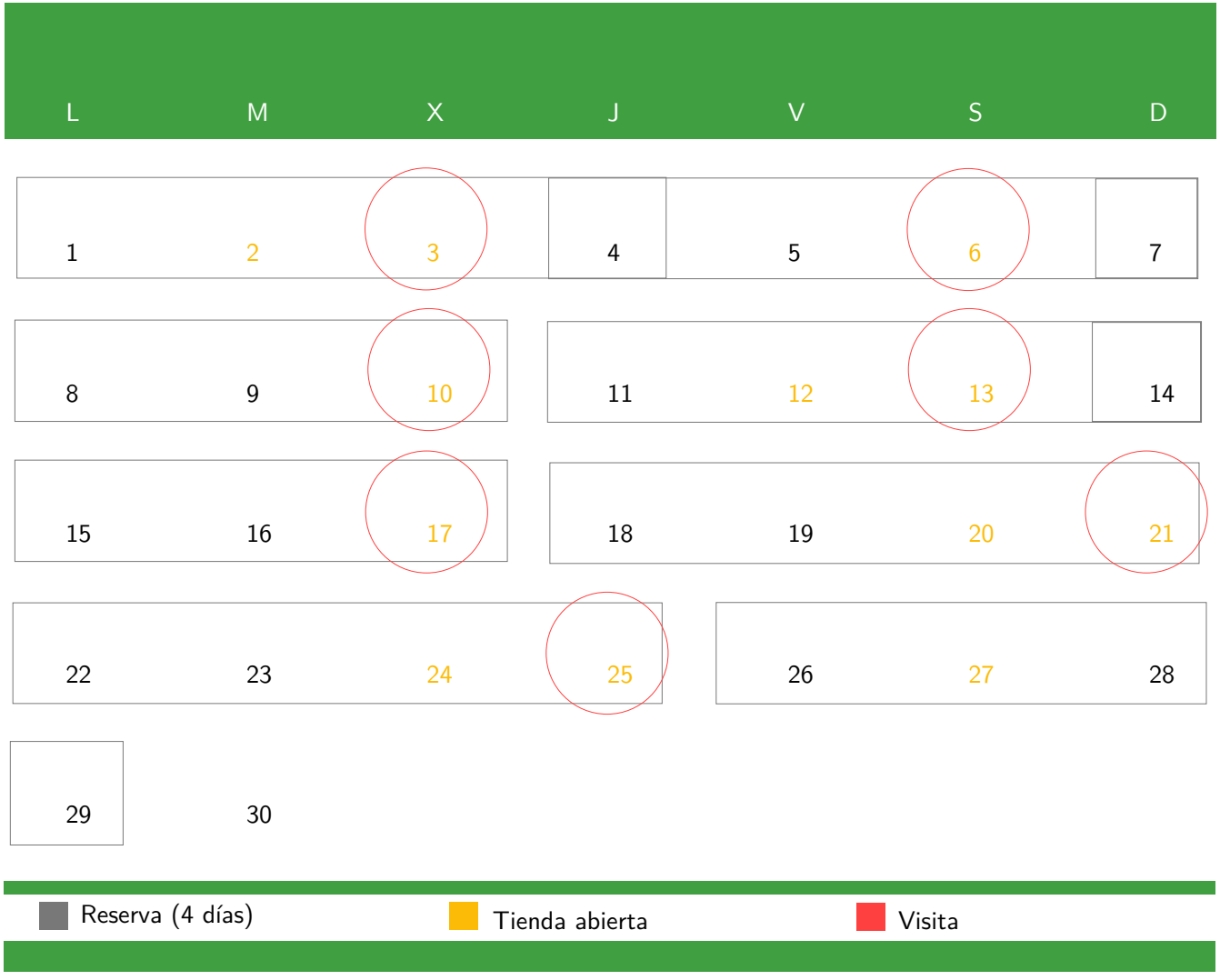
Ejemplo


L
M
X
J
V
S
D
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
Reserva (4 días)
Tienda abierta
Visita

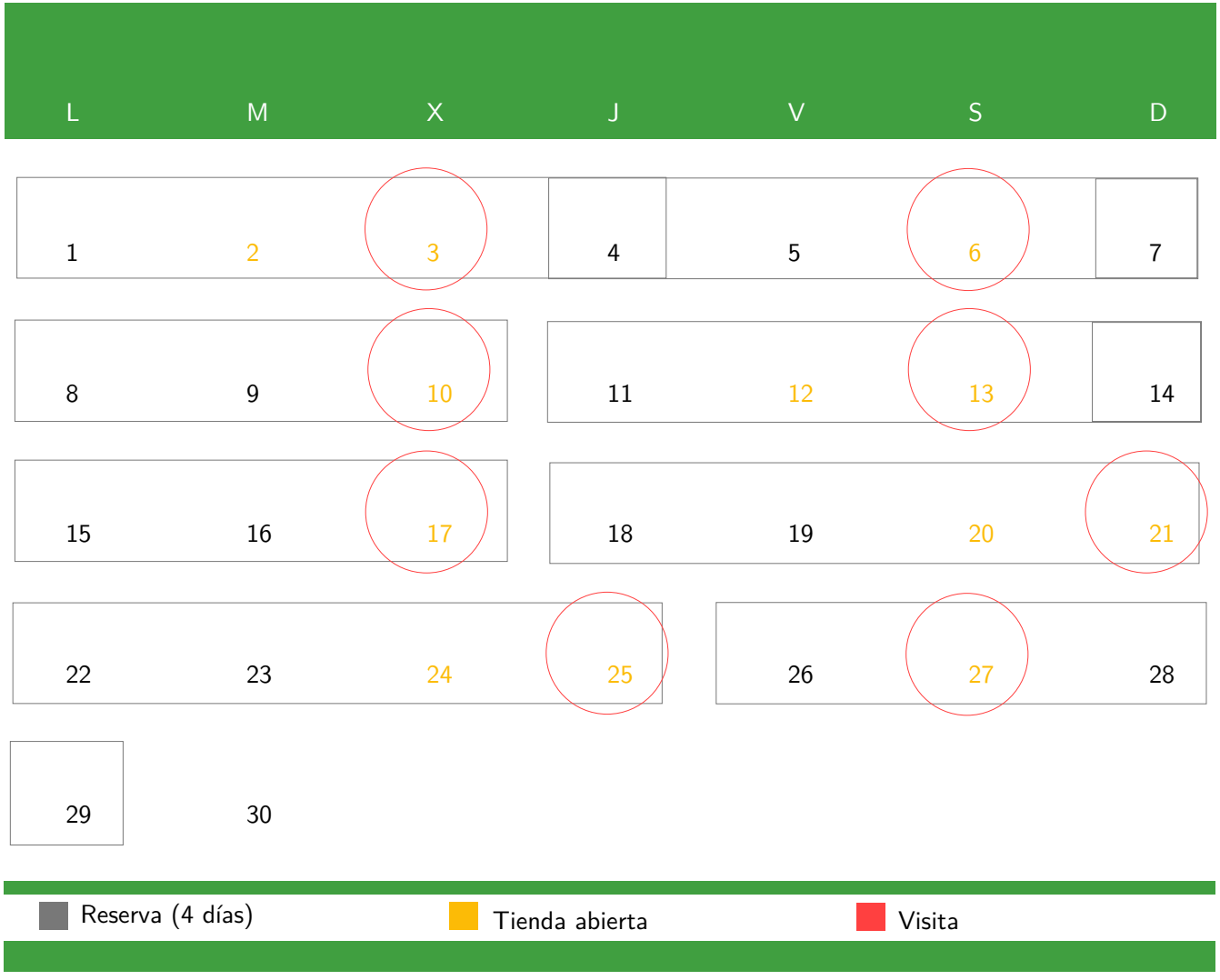
Ejemplo


L
M
X
J
V
S
D
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
Reserva (4 días)
Tienda abierta
Visita

Ejemplo


L M X J V S D
1 2 3 4 5 6 7
8 9 10 11 12 13 14
15 16 17 18 19 20 21
22 23 24 25 26 27 28
29 30
Reserva (4 días)
Tienda abierta
Visita

Ejemplo


L M X J V S D
1 2 3 4 5 6 7
8 9 10 11 12 13 14
15 16 17 18 19 20 21
22 23 24 25 26 27 28
29 30
Reserva (4 días)
Tienda abierta
Visita

Implementación

```
void minimize_visits(bool* days_open, int P, int R,
    list<int>& visit_days){
  for (int i=R-1; i < P; i+=R){
    while (!days_open[i])
      i--;
    visit_days.push_back(i+1); // Contamos los días a
        partir del 1
  }
}
```

Elementos de la demostración

- **p** la duración del periodo (ej: 30 días)
- **r** la duración de la reserva
- **a** vector de tamaño p que representa los días que abre la tienda
  La tienda abre el día n si $a_n = 1$
  La tienda no abre el día n si $a_n = 0$
- **d** el vector que almacena los días de visita
  $d_{i+1} = d_i + k_i \ / \ k_i \in \{x \in \mathbb{N} : 0 < x < r \ y \ a_{d_i+x} = 1\}$

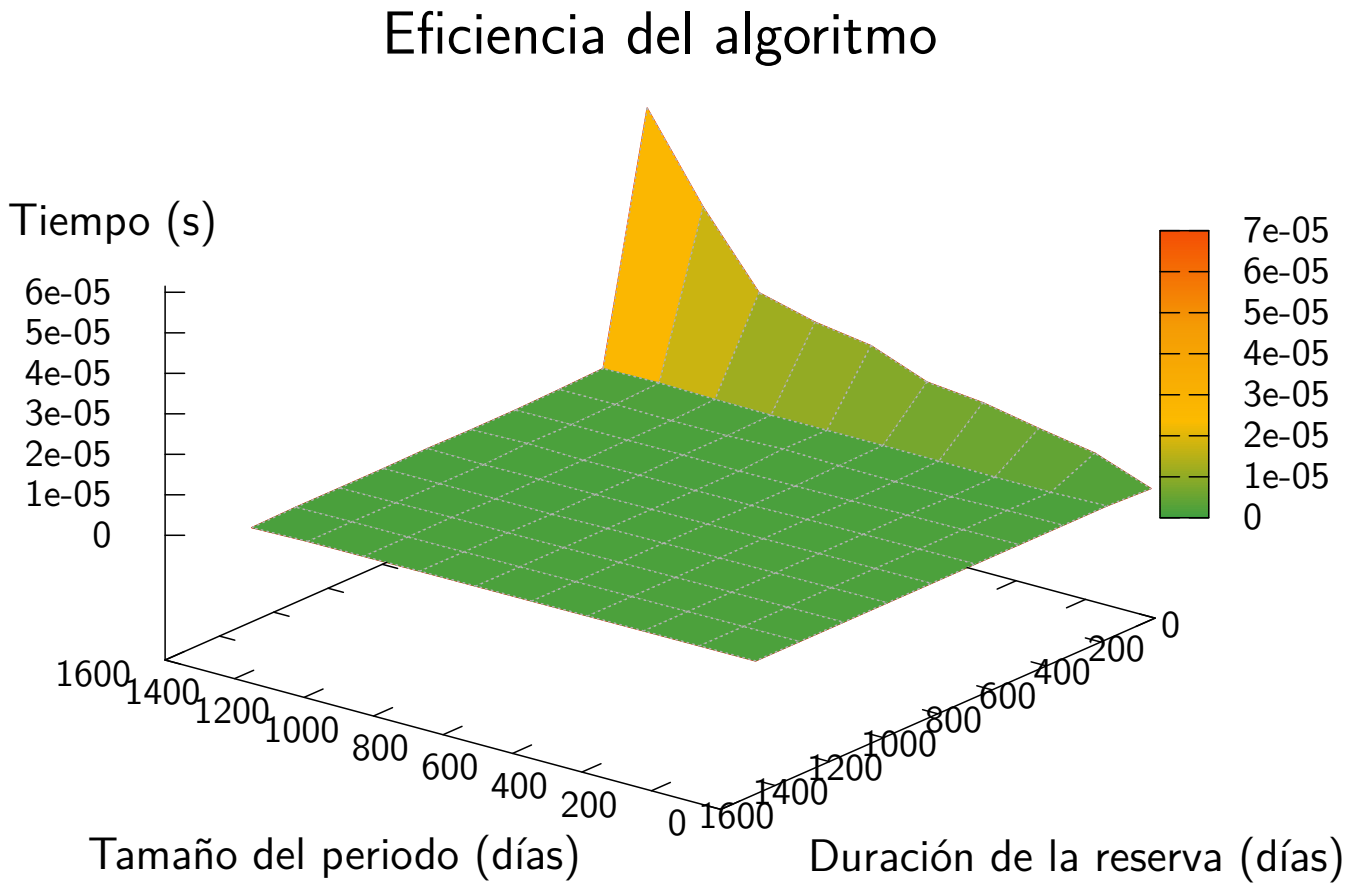
Demostración

Formulamos dos ecuaciones:
$d^h_{i+1} = d^h_i + h_i$ donde $h_i = max\{x \in \mathbb{N} : 0 < x < r \ y \ a_{d_i+x} = 1\}$
Podemos definir $n \in \mathbb{N} \ / \ d^h_n = min\{d^h_i : d^h_i + r > p\}$ como el índice del último día que necesitamos ir a comprar.
$d^t_{i+1} = d^t_i + t_i$ donde $t_i \in \{x \in \mathbb{N} : 0 < x < r \ y \ a_{d_i+x} = 1\} \ / \ t_i < h_i$ para algún $i < n$
Por las definiciones anteriores sabemos que
$\exists m \in \mathbb{N} \ / \ d^t_i \leq d^h_i \ \forall i, \ m \leq i < n \Rightarrow d^t_n \leq d^h_n$
Distinguimos dos casos:

1. Si $p - r < d^t_n \leq d^h_n$ las dos opciones son igual de buenas
2. Si $d^t_n < p - r < d^h_n$ la opción greedy es mejor

De aquí deducimos que utilizar el enfoque greedy en este caso siempre nos da el mejor resultado.

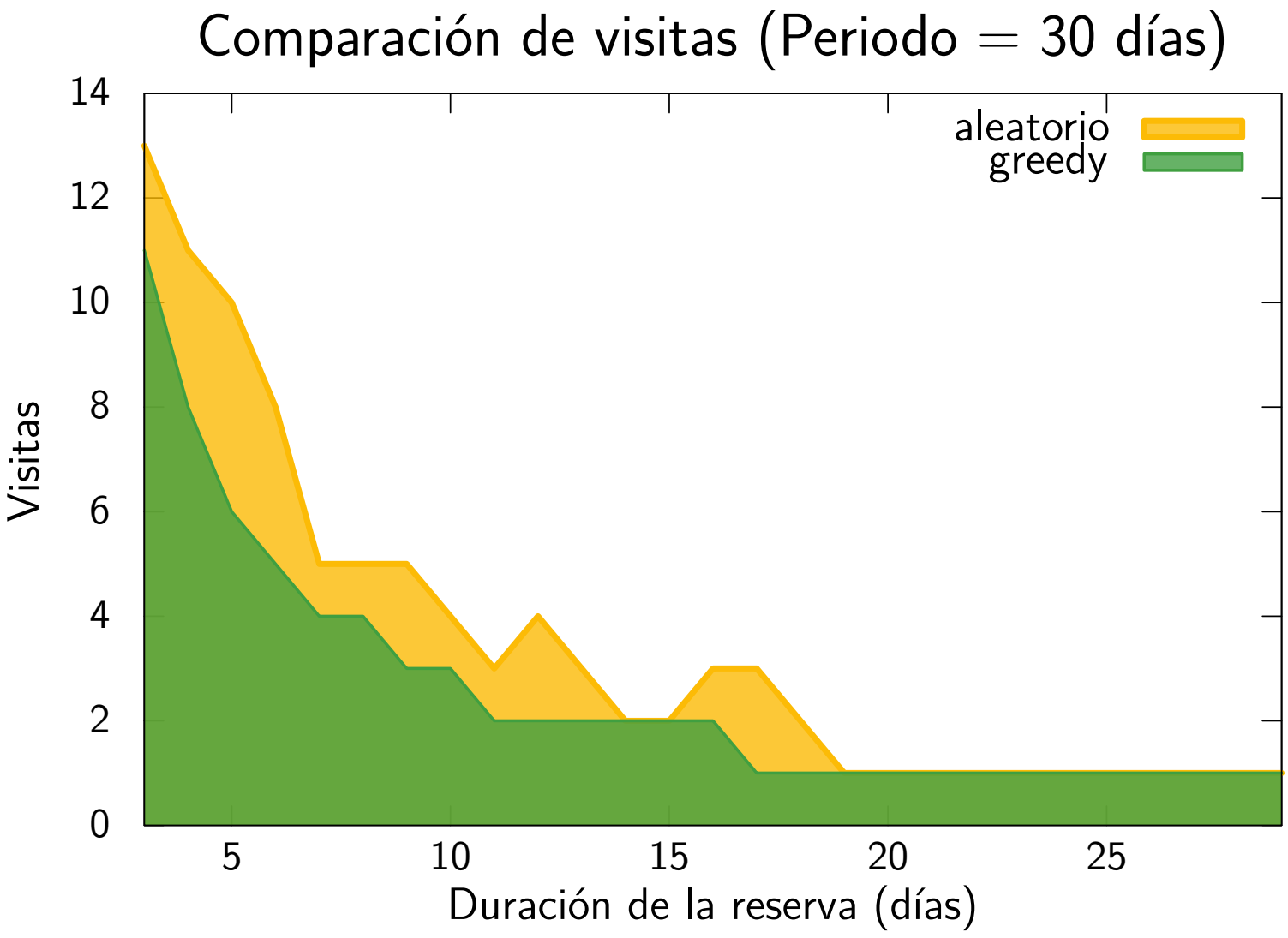
Eficiencia



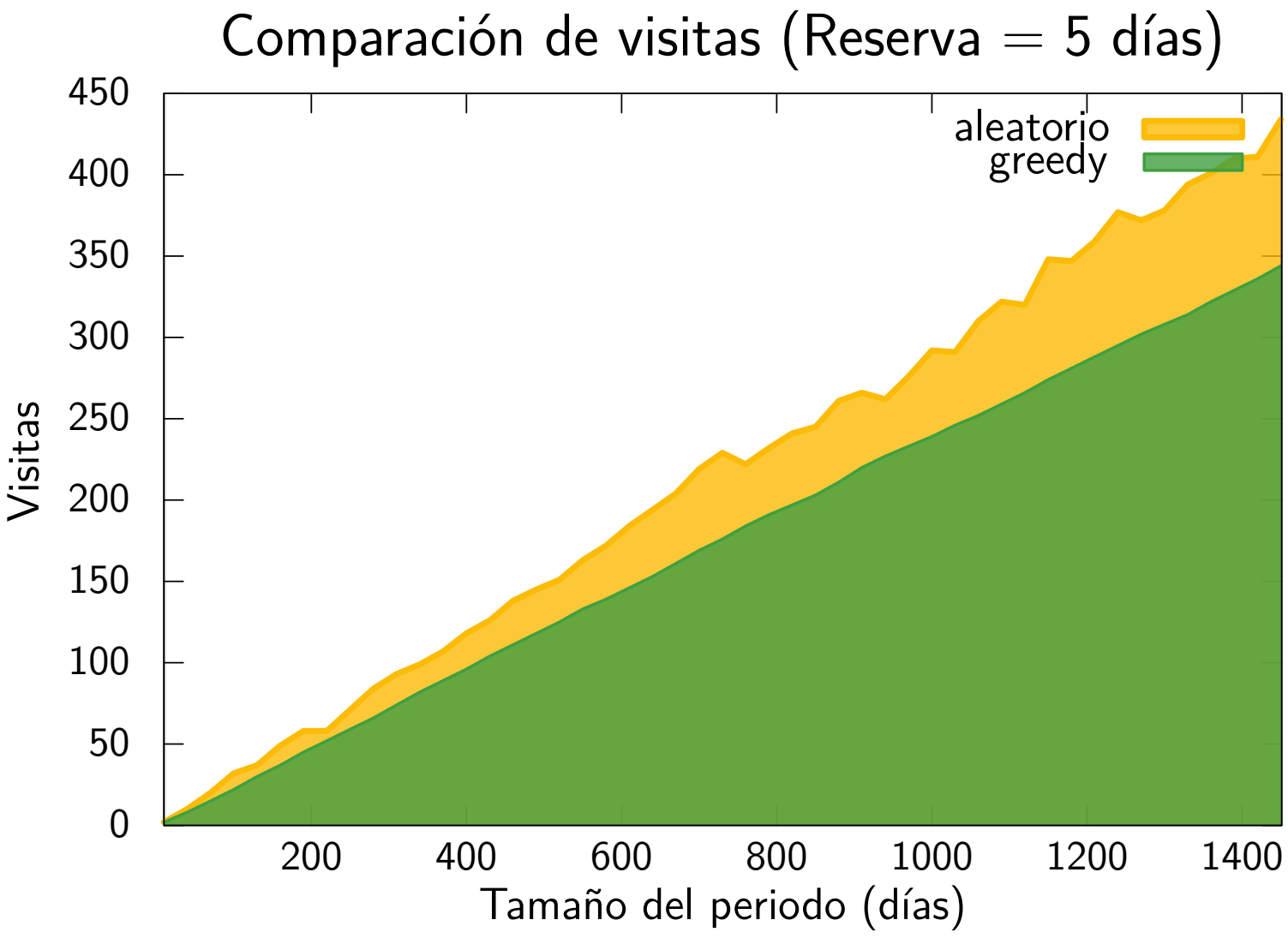
La eficiencia es $O(p)$.

En el caso de que el día abierto se encuentre en $\frac{r}{2}$ es $p - \frac{r}{2}$

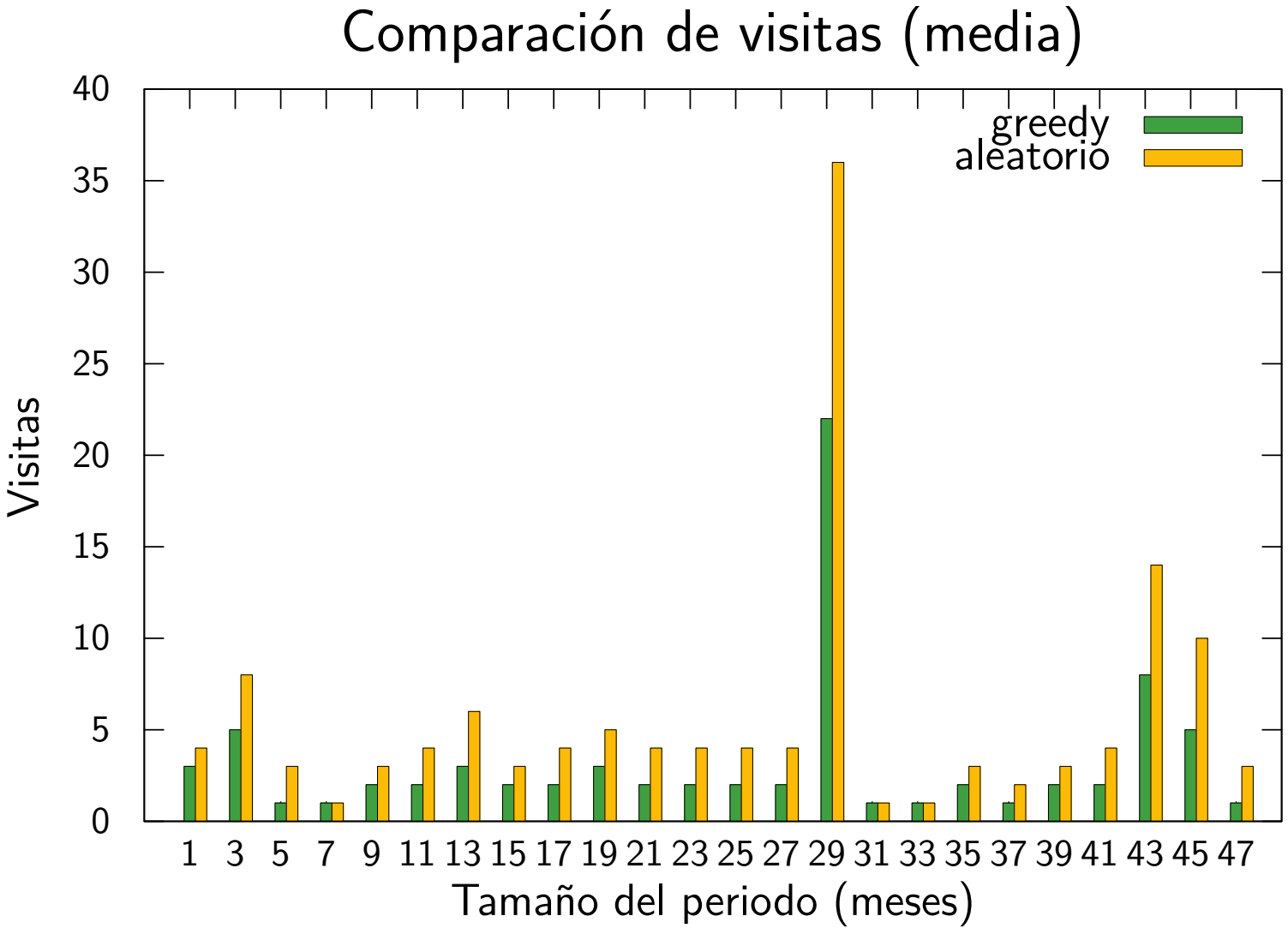
Visitas fijando el periodo



Visitas fijando la reserva



Media de visitas (r aleatorio)



Enunciado

Dado un conjunto de ciudades y una matriz con las distancias entre todas ellas, un viajante debe recorrer todas las ciudades exactamente una vez, regresando al punto de partida, de forma tal que la distancia recorrida sea mínima.

Solución

Permutación del conjunto de ciudades que indica el orden en que se deben recorrer

Algoritmo

Nos centraremos en una serie de algoritmos aproximados de tipo greedy y evaluaremos su rendimiento en un conjunto de instancias.

Greedy usados

- Vecino más cercano
- Inserción de vértices
- Inserción de aristas

Desarrollo del algoritmo

Primero escogeremos una ciudad de inicio. Después calculamos las distancias de esa ciudad al resto y escogemos la más cercana. A partir de esa ciudad volvemos a calcular la ciudad más cercana y procedemos hasta pasar por todos los vértices.

Aviso

No debemos olvidar incluir el camino del último vértice al inicial, ya que forma parte del problema y hay que sumar también esa distancia.

Paso 1

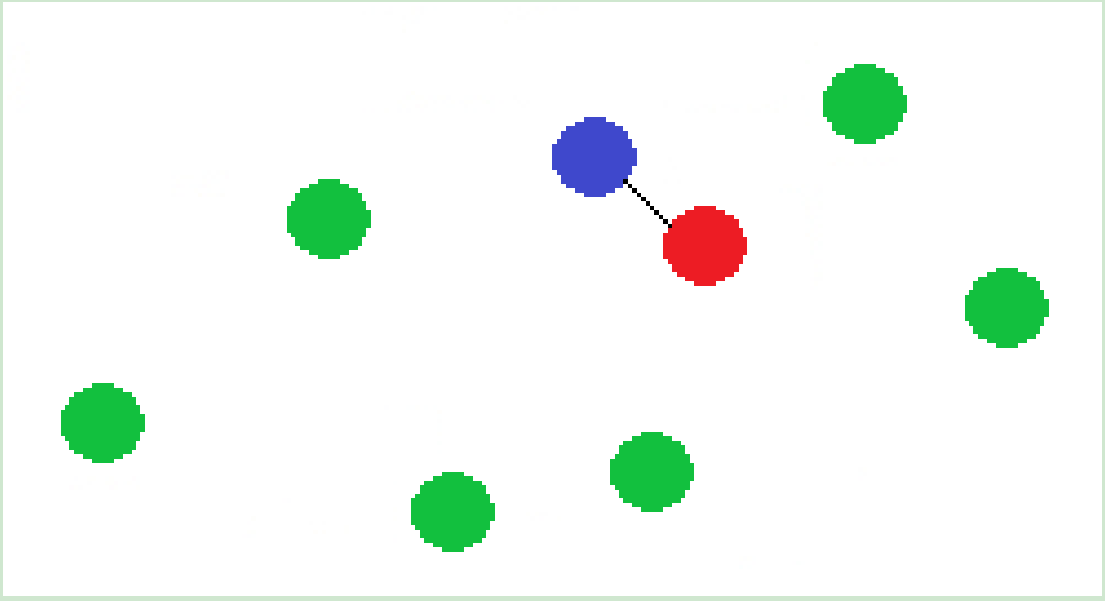


Figura : Tercer vértice

Paso 2

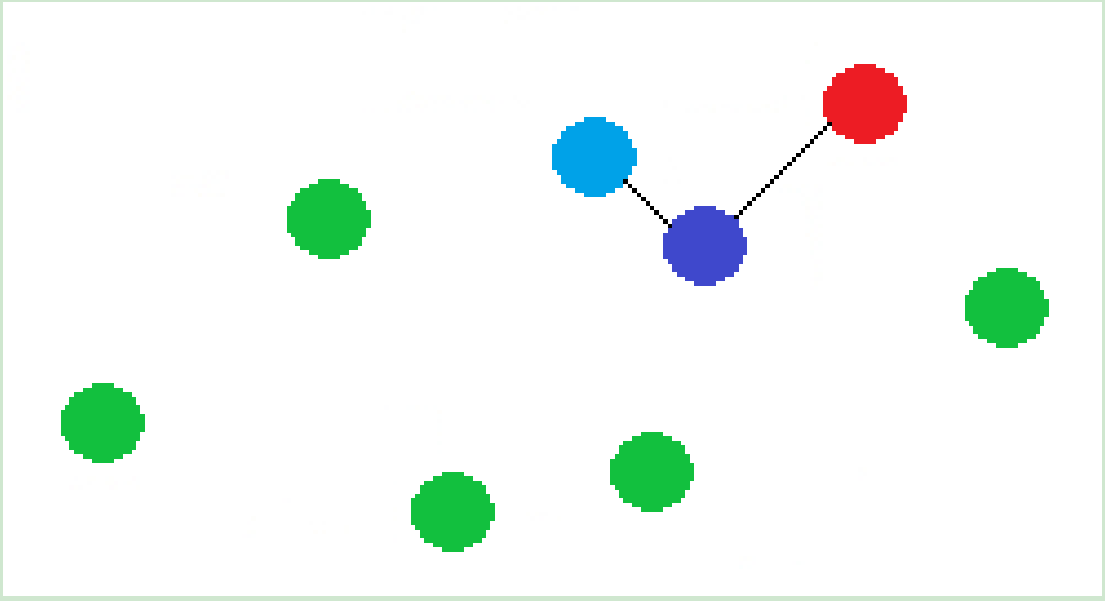


Figura : Tercer vértice

Paso 3

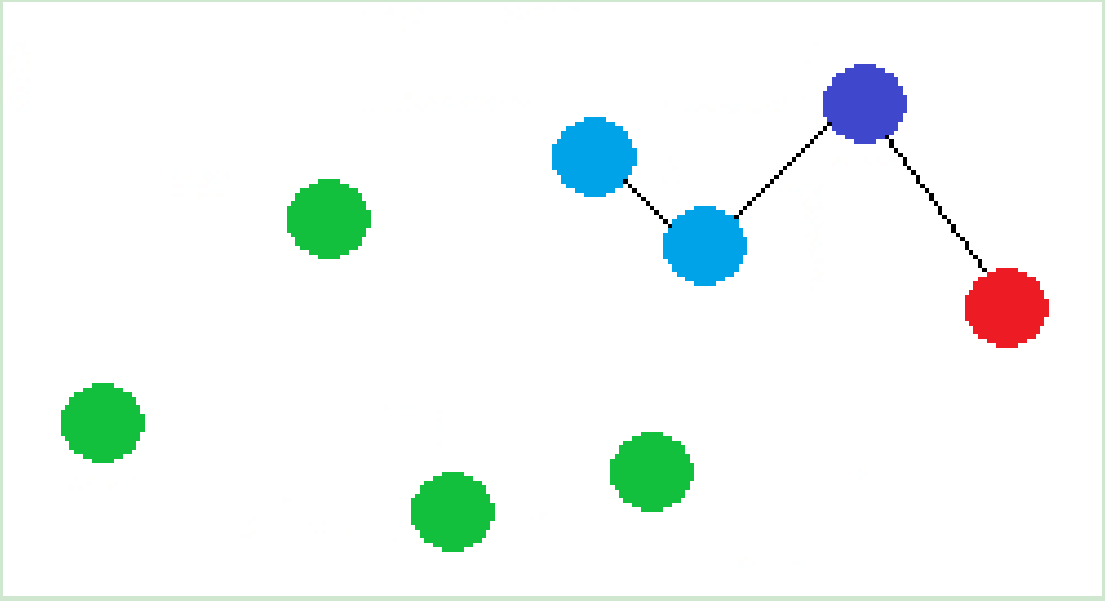


Figura : Cuarto vértice

Paso 4

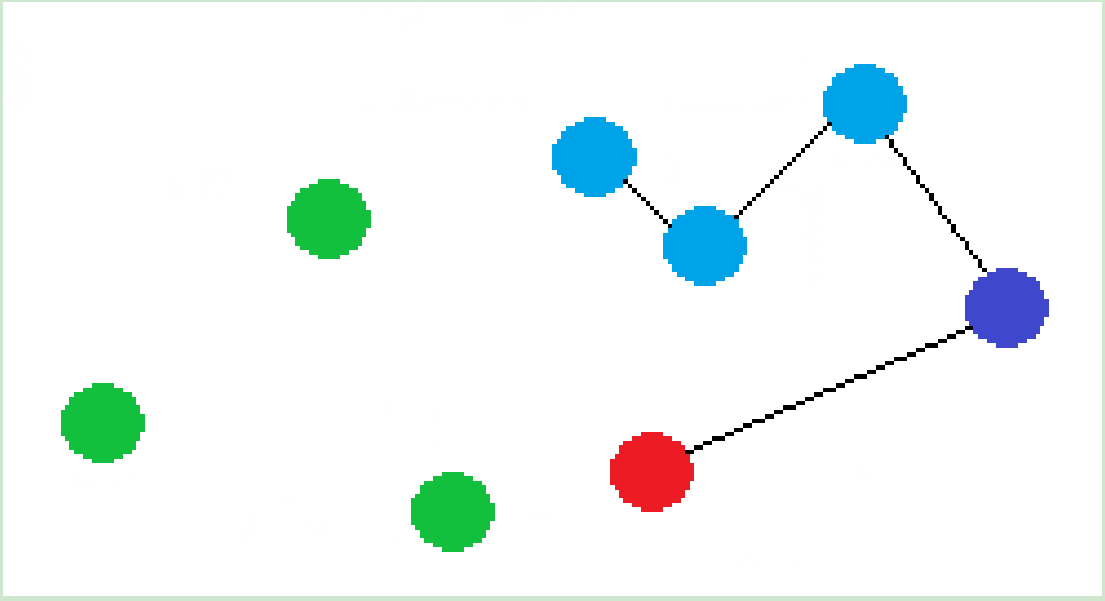


Figura : Quinto vértice

Paso 5

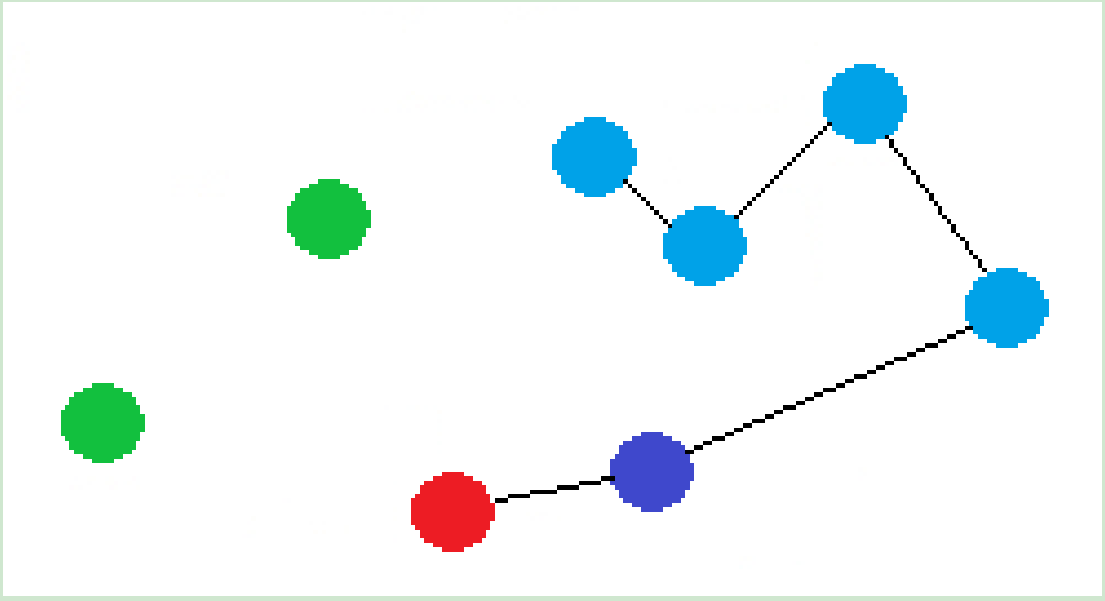


Figura : Sexto vértice

Paso 6

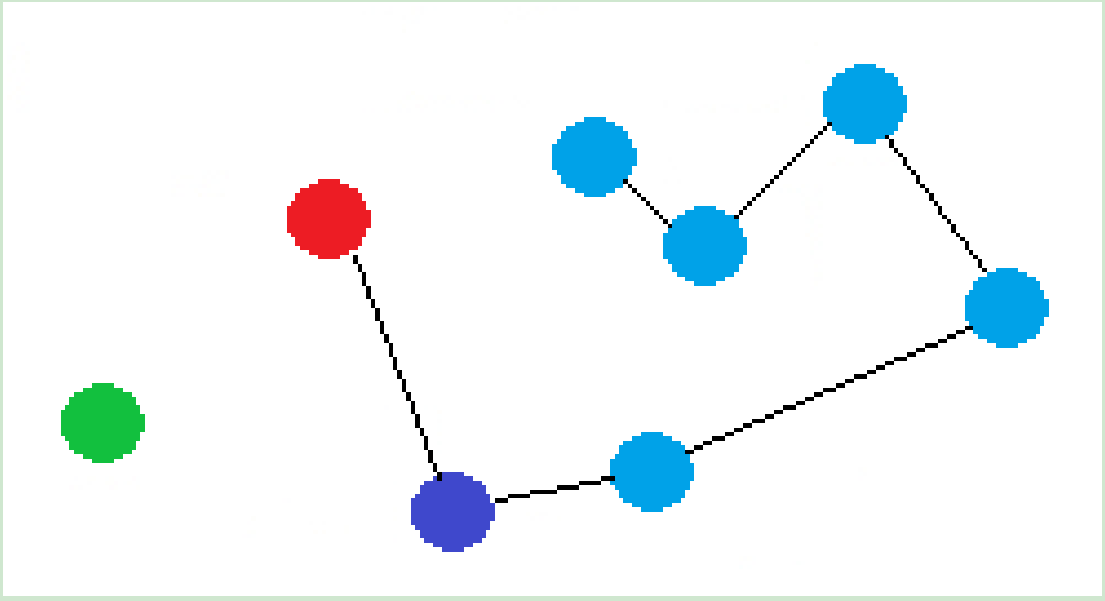


Figura : Séptimo vértice

Paso 7

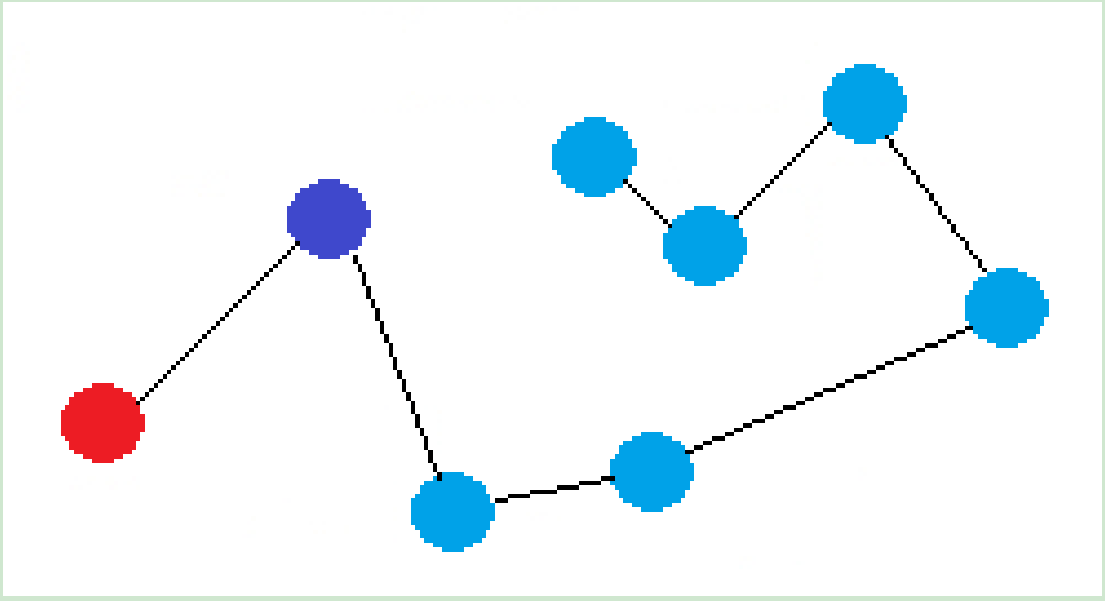


Figura : Octavo y último vértice

Paso 8

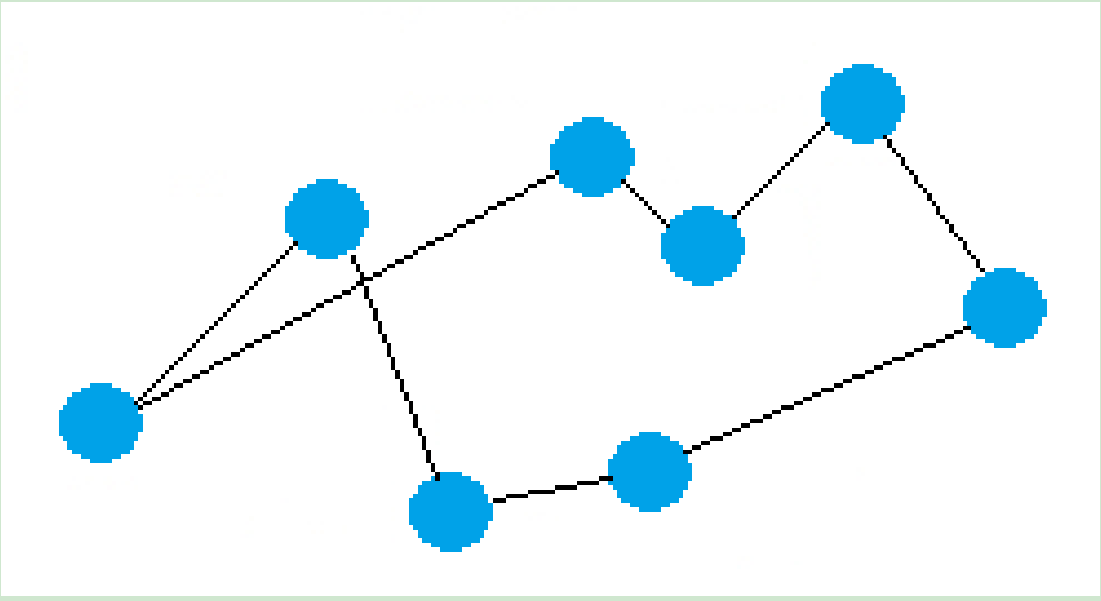


Figura : Recorrido vecino más cercano

Calculo de la eficiencia

Cada ciudad la denotamos como $x_i = (i_a, i_b), i \in \{0, 1, ...n-1\}$
Y definimos la distancia entre dos ciudades como
$dist(x_i, x_j) = \sqrt{(i_a - j_a)^2 + (i_b - j_b)^2}$

En cada paso el cálculo del mínimo tiene que comprobar que la nueva ciudad no forme ciclo (pasaríamos dos veces por la misma ciudad)
Definimos $\phi(x_i) = \{j \in A : min\{dist(x_i, x_j)\}\ y\ no\ forme\ ciclos\}$

Comprobar si una ciudad forma ciclos es lineal, por lo que tenemos que $\phi(x_i)$ es cuadrático. Debemos pasar por las n ciudades, por lo que su eficiencia es:

$$\sum_{k=0}^{n-1} \phi(x_k) \implies O(n) = n^3$$

Puntos

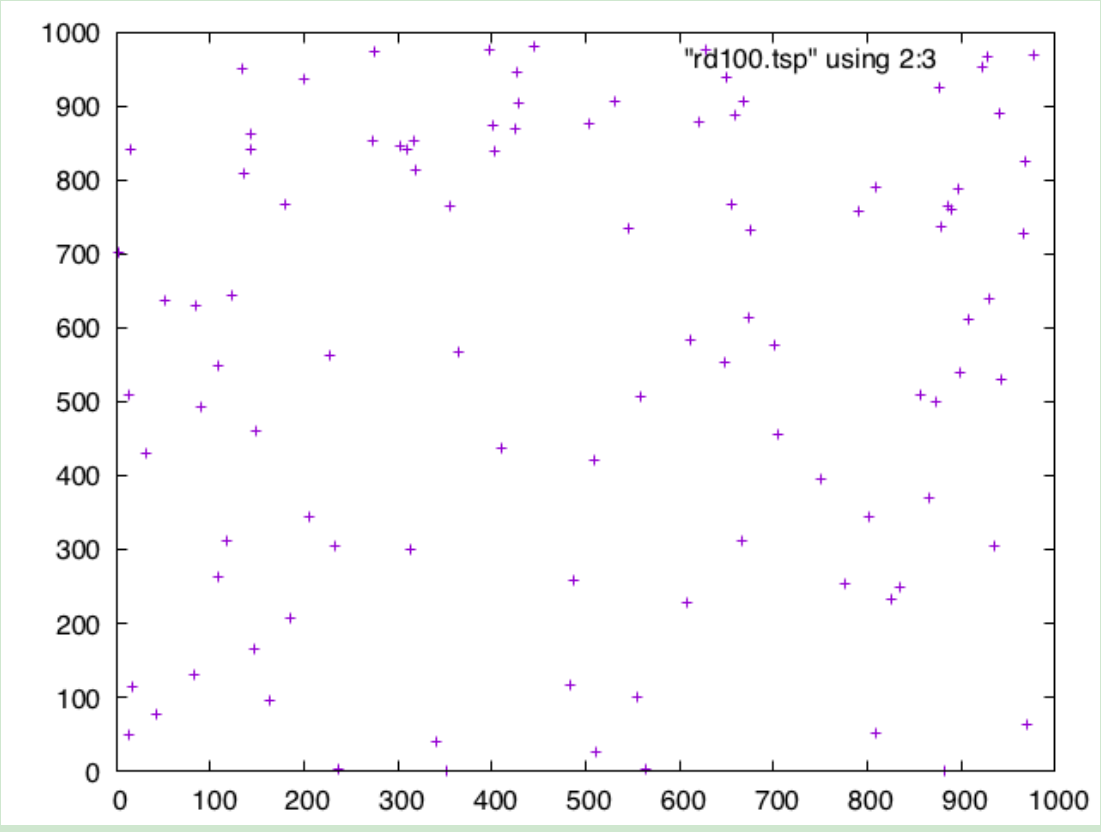


Figura : Quinto vértice

Recorrido óptimo

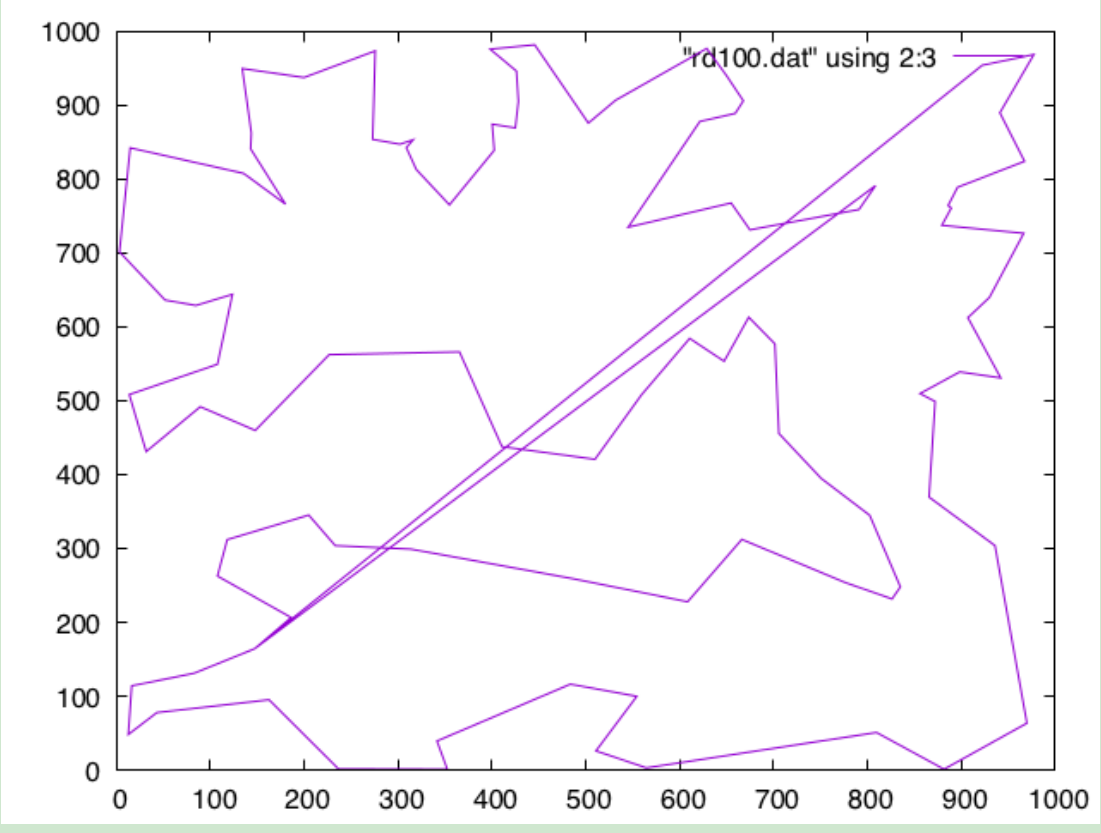


Figura : Quinto vértice

El recorrido parcial inicial lo determinan los 3 puntos que estén más al Este, Oeste y Norte respectivamente. Los 3 no deben ser los mismos, algo que nunca pasa, pues 2 de ellos pueden ser iguales (un punto puede ser a la vez el punto más al Norte y al Este), pero 3 a la vez no.

El nodo siguiente a insertar en el recorrido parcial será un nodo arbitrario, de lo que nos encargaremos para hacer funcionar el greedy es de insertarlo entre los dos vértices que más convengan, minimizar la distancia del circuito resultante.
Pasaremos por todos los vértices que nos quedan cada vez, y sobre cada uno calculamos la posición que más nos convenga dentro del circuito, y luego escogemos el vértice que nos da mejor resultado.

T

enemos que recorrer n vértices, y sobre cada uno de ellos aplicaremos una función $\phi(x)$ para cada vértice.

$$\sum_{i=1}^{n} \phi(x_i)$$

D

icha función $\phi(x)$ llama a "buscar punto"que es lineal, cada vez que buscas un punto llama n veces a "buscar posicion", que a su vez llama n veces a calcular longitud.
Por tanto $\phi(x)$ es cúbica, y la eficiencia que nos queda para el algoritmo de inserción es

$$O(n) = n^4$$

Puntos

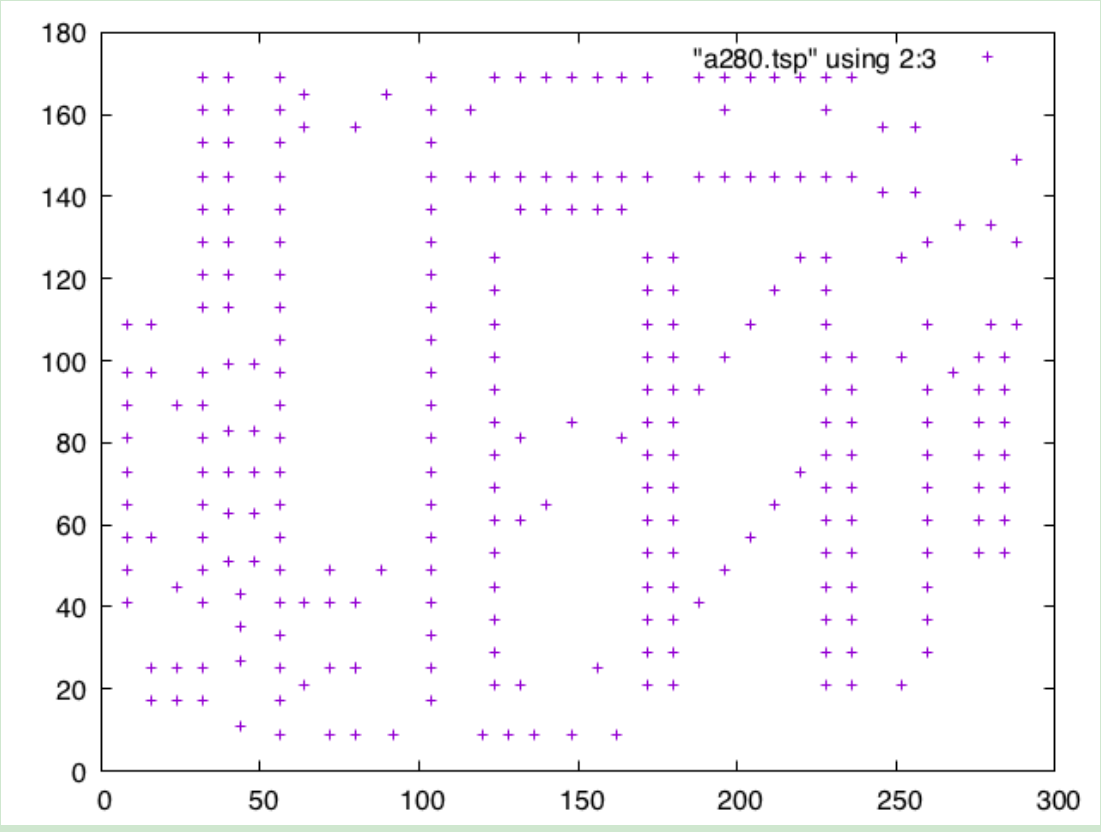


Figura : Quinto vértice

Recorrido óptimo

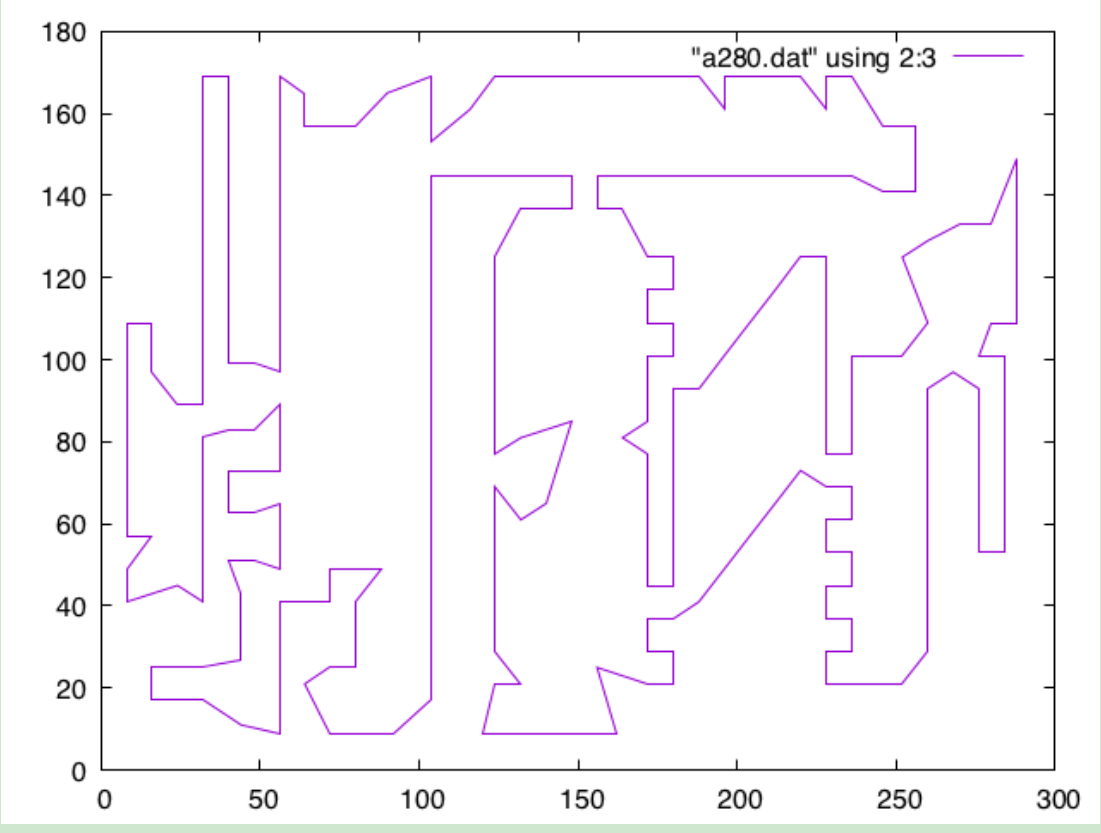


Figura : Quinto vértice

Estrategia que consiste en ir seleccionando las aristas de menor longitud.
Primero crearemos una clase Arista donde guardaremos los índices de los puntos dentro del grafo (índices de ambos vértices de la arista) y la distancia entre ellos, sacada de la matriz de adyacencia del grafo.
Posteriormente ordenaremos todas las aristas en un vector, e iremos seleccionando hasta pasar por todos los vértices una única vez, comprobando en cada paso que no forme nuevos ciclos.

D

efinimos una relación de orden entre aristas de la siguiente forma: Una arista es *menor que* otra si la distancia entre sus puntos es menor que la distancia entre los puntos de la segunda arista.

Hemos usado un vector de listas de aristas
$vector < list < Edges >>$.
Cada lista definirá un camino inconexo con el resto de listas del vector, de forma que su primer elemento y último sean los extremos. Al final del algoritmo nos quedará una única lista ordenada con la solución.

Para saber si una arista forma ciclo o no asignaremos un estado a cada vértice.

Inicialmente los vértices tienen estado -1, significa que no han sido usados aún. Si se inserta una nueva arista, su primer vértice (inicio del camino) pasará al estado [1+10*posicion], siendo *posicion* el lugar que ocupa dentro del vector de listas
El segundo vértice (último elemento) tendrá el estado [2+10*posicion]
De esta forma si (estado1/10 == estado2/10) sabemos que los vértices están en el mismo camino.

Si hubiese vértices entre el principio y el final (cuando tenemos más de una arista) los vértices intermedios tendrían estado [3+10*posicion]

D

enotando como *one* y *two* los códigos de los vértices de la arista que queremos insertar.

Caso 1

Ninguno está. En este caso se añadiría una nueva lista con una sola arista que nos indicaría que se ha creado un nuevo camino.

Código

```
((one == -1) & & (two == -1))
```

Caso 2

Está uno y el otro está libre.

Código

```
((one%10 == 1) & & (two == -1) || (one%10 == 2) & &
(two == -1)
(two%10 == 1) & & (one == -1) || (two%10 == 2) & &
(one == -1) )
```

Caso 3

La arista une caminos, están los dos vértices, pero en caminos diferentes. Meteremos los elementos de la segunda lista de aristas en la primera

Código

```
( ((one%10 == 1) & & (two%10 == 1) & & (one/10 != two/10)) ||
((one%10 == 2) && (two%10 == 2) & & (one/10 != two/10)) ||
((one%10 == 1) && (two%10 == 2) && (one/10 != two/10)) ||
((one%10 == 2) && (two%10 == 1) && (one/10 != two/10)) )
```

Siendo n el número de puntos del grafo, para ordenar las n^2 aristas usaremos el Heapsort, por tanto obtenemos una eficiencia de $n^2 log(n^2) \implies n^2 log(n)$.

En el peor caso recorreremos las n^2 aristas hasta encontrar las n aristas deseadas.
Como el algoritmo para ver si una arista forma ciclo es constante, si forma ciclo pasará a la siguiente iteración, si no forma añadirá dicha arista.

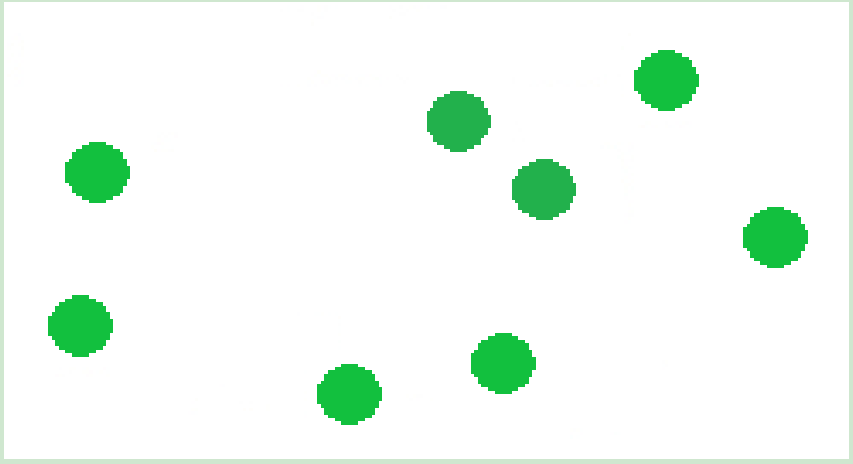


Figura : Eligiendo punto de inicio en azul

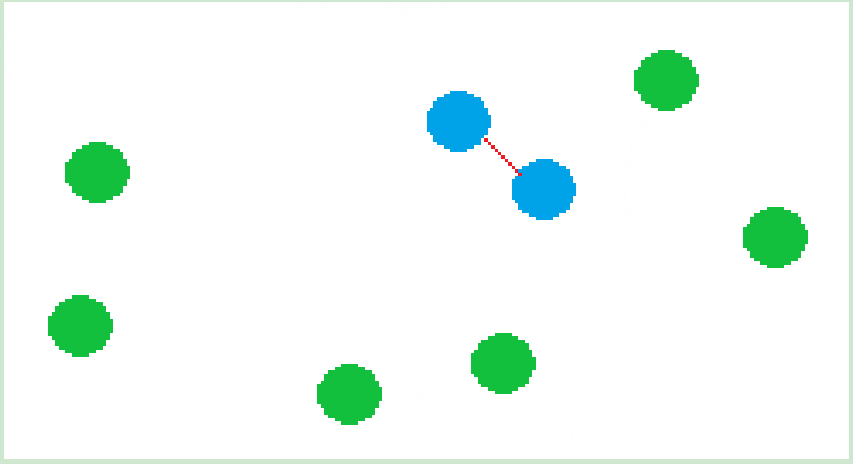


Figura : Tercer vértice

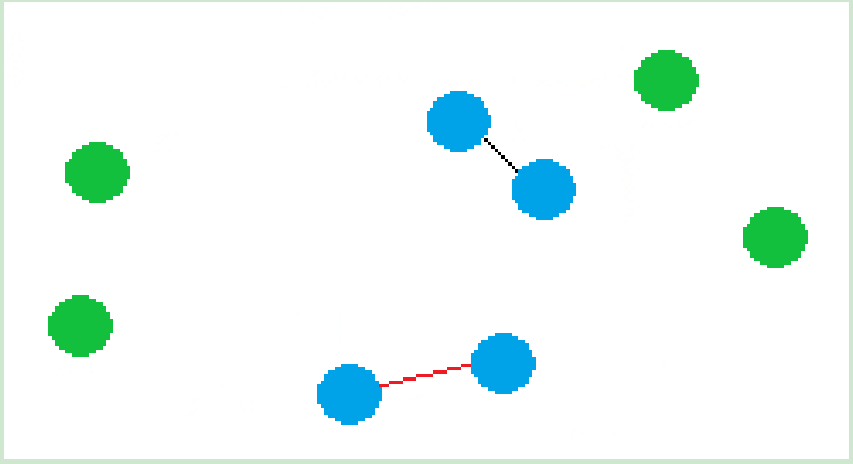


Figura : Cuarto vértice

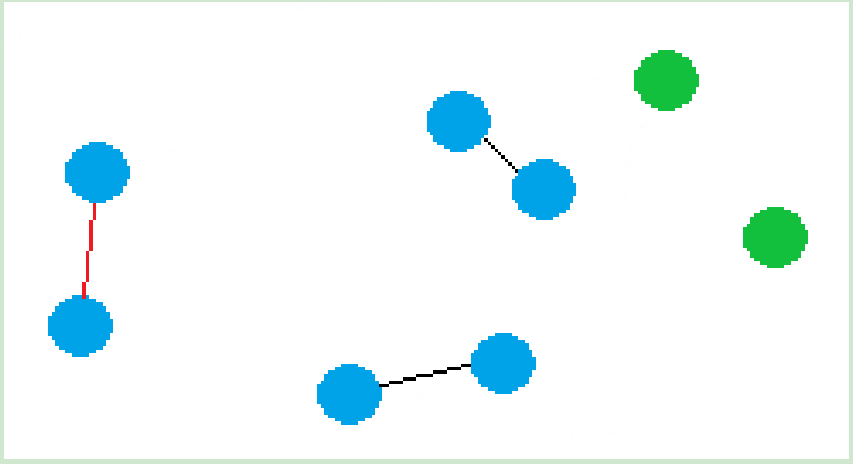


Figura : Quinto vértice

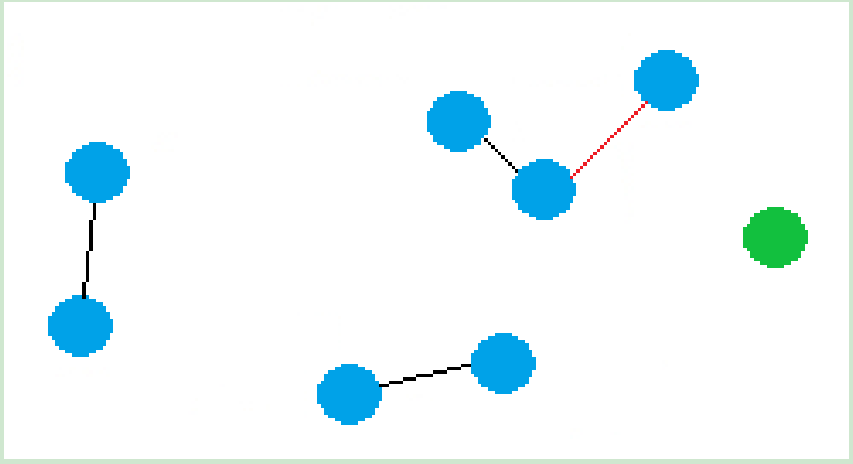


Figura : Sexto vértice

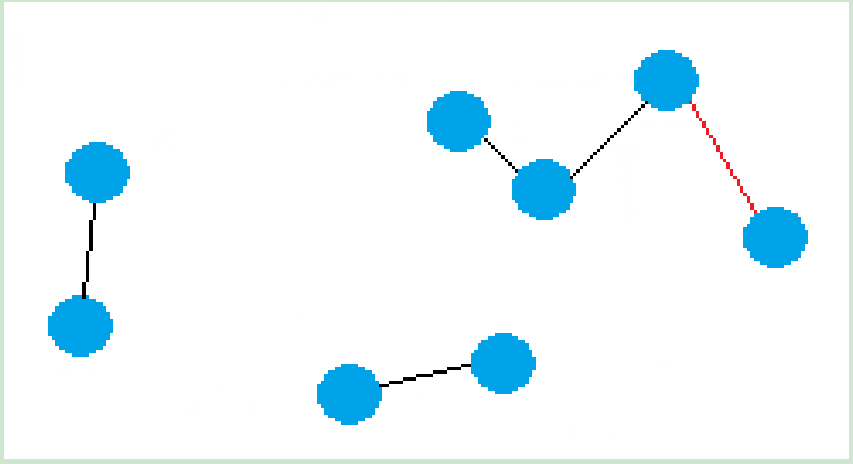


Figura : Séptimo vértice

Ojo

Debemos tener en cuenta que solamente podemos unir aristas que estén en dos caminos diferentes si sus puntos son extremos de caminos ya existentes. De esta forma aunque un vértice sea un extremo de un camino, si su otro extremo está en mitad de otro camino no nos vale, ya que llegaría un punto en el que pasaríamos dos veces por el mismo punto.

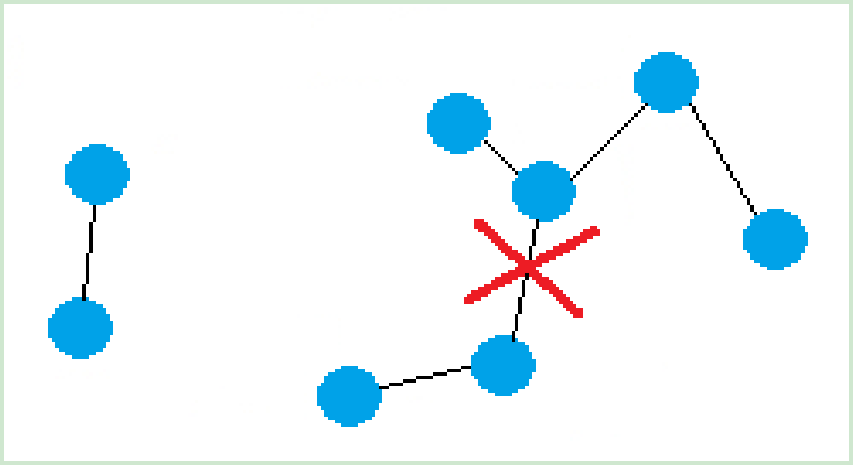


Figura : Octavo y último vértice

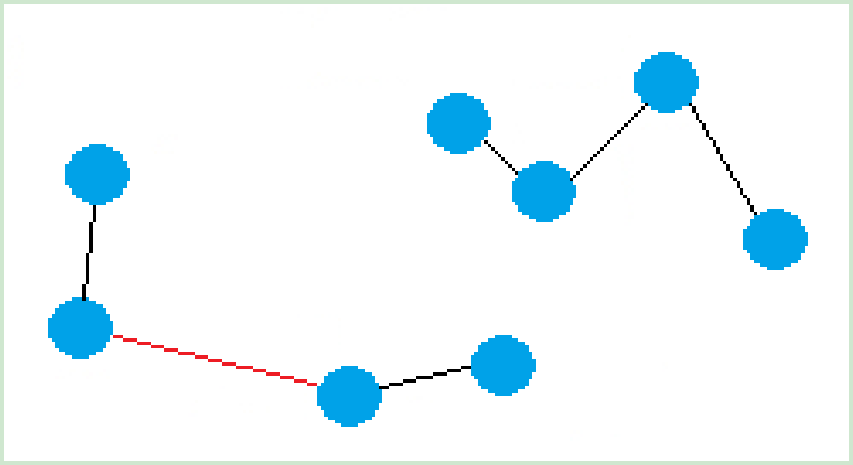


Figura : Octavo y último vértice

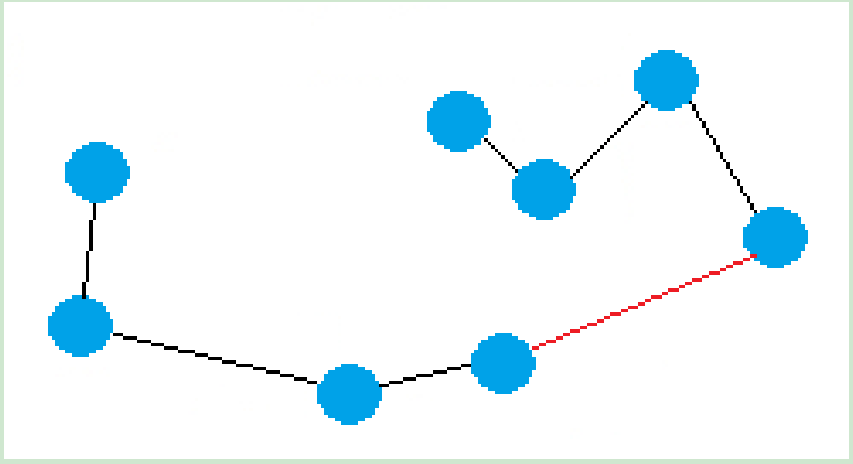


Figura : Octavo y último vértice

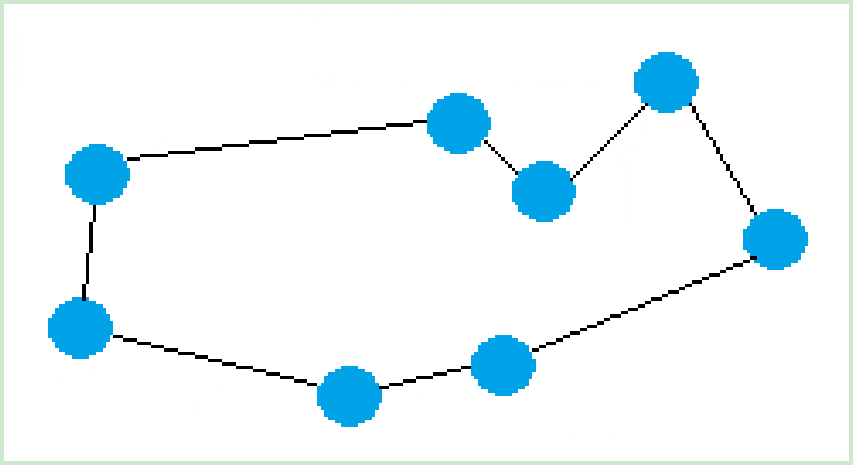


Figura : Recorrido vecino más cercano

Puntos

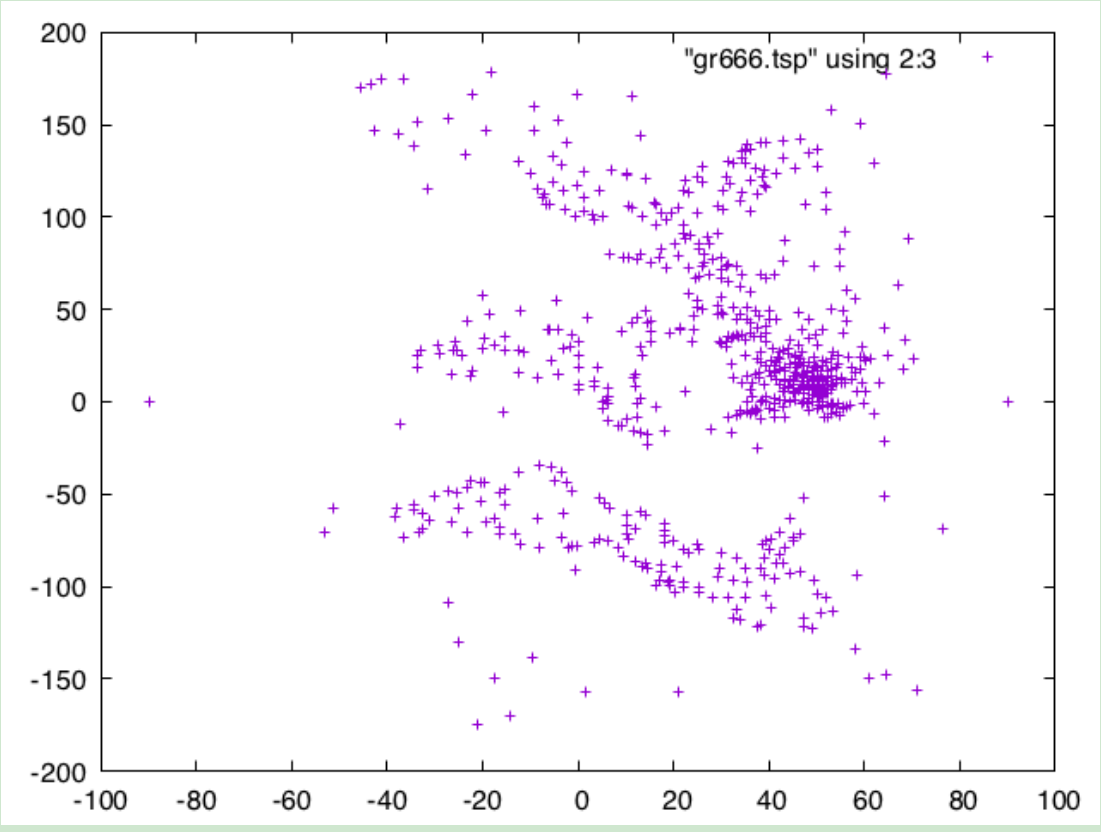


Figura : Quinto vértice

Recorrido óptimo

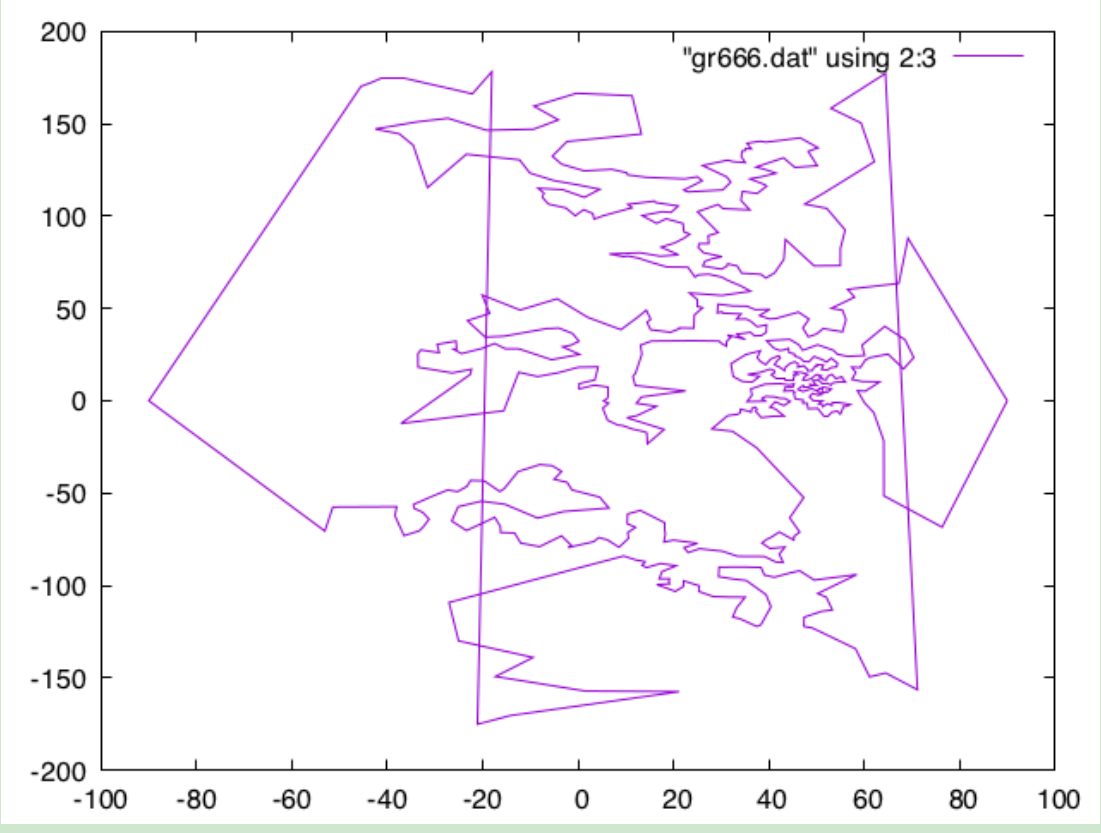


Figura : Quinto vértice