Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación

IIC2133 — Estructuras de Datos y Algoritmos
Segundo semestre 2017

Tarea 1

Fecha de entrega: Jueves 7 de Septiembre

Objetivos

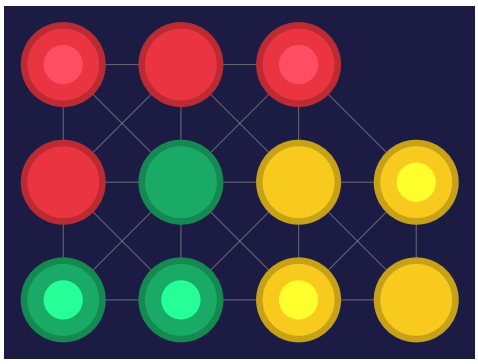
- Modelar un problema de satisfacción de restricciones usando estructuras de datos.
- Implementar un algoritmo eficiente de backtracking para resolver un problema complejo.

Introducción

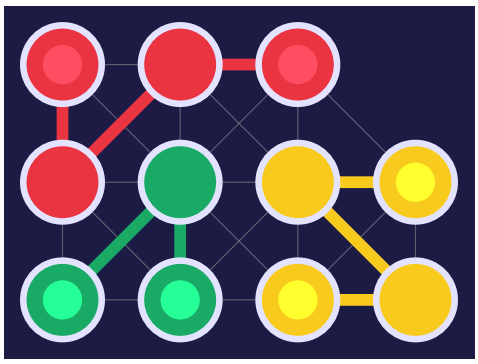
¿Has visto alguna vez dentro de un computador que algunos cables son de colores específicos, y que hay que conectarlos en los puertos del mismo color? Dentro de los distintas placas del computador se requiere alimentar distintos componentes con distintos tipos de energía, los cuales se representan con distintos colores. Decidir cómo disponer los cables no siempre es una tarea trivial, por lo que hay un ejército de duendecillos encargados de conectar los cables correctamente cada vez que quieres hacer algo con tu computador. Después de años de trabajo duro, los duendecillos han decidido modernizarse, y automatizar la disposición de los cables, por lo que te han contactado para que implementes un algoritmo capaz de resolver sus problemas.

Problema: Cableado

Una placa consiste en una cuadrícula de **n** componentes a ser conectadas, alineadas en **h** filas y **w** columnas, como se muestra en la siguiente imagen:



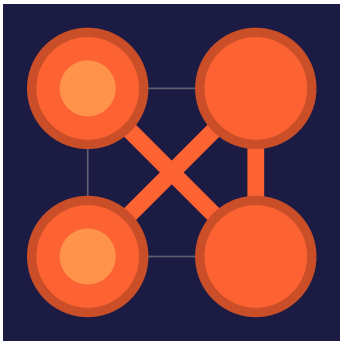
Componentes a conectar



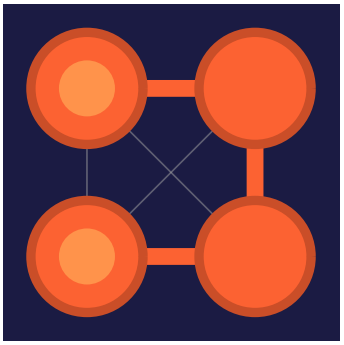
Componentes conectadas

Los distintos colores representan los distintos tipos de energía que necesitan los componentes. Los componentes terminales representan los extremos de los cables para un tipo de energía, y se dibujan con un círculo en su interior. No necesariamente existe un componente en cada coordenada de la placa.

Para no hacer corto-circuito, está terminantemente prohibido cruzar un cable por sobre otro, sean del mismo tipo o de tipos distintos, como se muestra en la siguiente configuración:

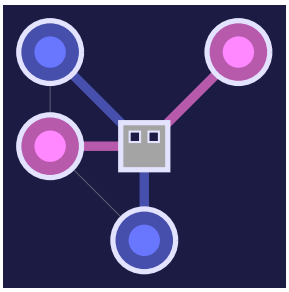


Configuración inválida: cruce de cables.

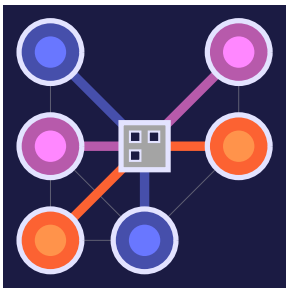


Configuración válida.

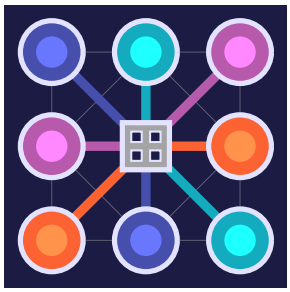
Existe un componente especial exclusivamente para este propósito, llamado *hub*, el cual no tiene color y viene con distintas capacidades:



Hub de capacidad 2



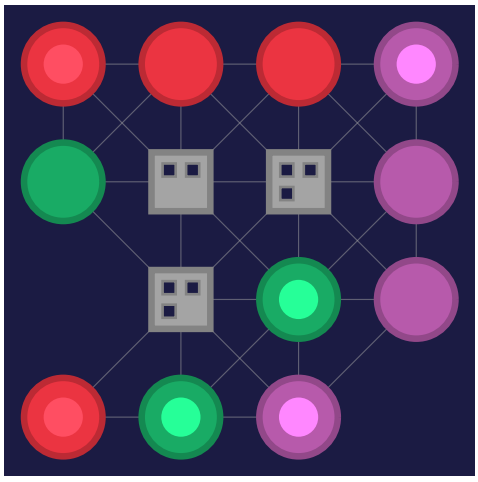
Hub de capacidad 3



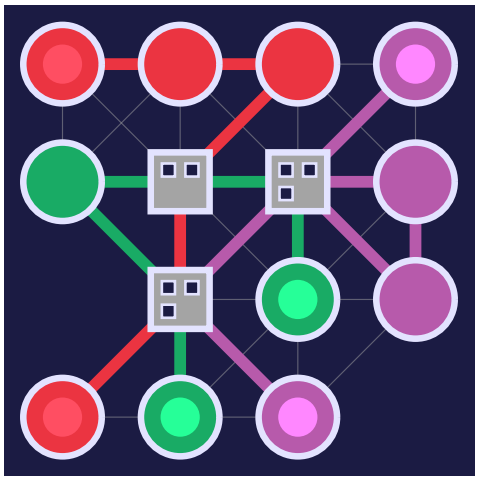
Hub de capacidad 4

La capacidad de un hub indica cuantos cables distintos pueden pasar por él, sean o no del mismo tipo. Esta puede saberse fácilmente viendo la cantidad de agujeros que presenta. Para que un hub funcione correctamente debe funcionar a máxima capacidad, es decir, deben pasar por él la cantidad máxima de cables posibles.

Tu objetivo es, dada una placa con la disposición de las distintas componentes, encontrar el cableado de manera de alimentar todas las componentes con el tipo de energía correcto. Esto significa también ocupar todos los hubs a máxima capacidad. A continuación se muestra un ejemplo:



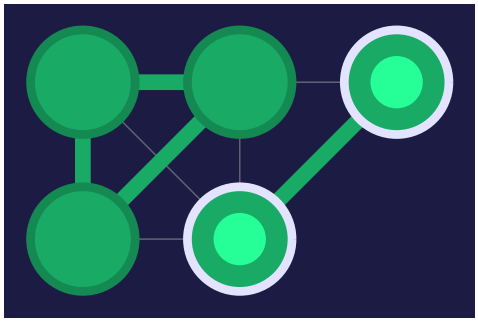
Componentes a conectar



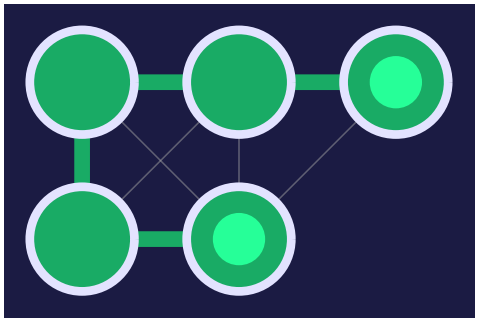
Componentes conectadas, procurando incluir a los hubs

Al final del proceso deberás asegurar, para cada componente, que se cumplan las siguientes restricciones:

1. Si tiene color, cualquier conexión con sus vecinos debe ser mediante un cable de ese color.
2. Si es terminal, debe estar conectado con exactamente un vecino.
3. Si es un hub de capacidad k, deberá estar conectado con exactamente $2k$ vecinos. Los colores de los cables que hacen estas conexiones deberán estar de a pares.
4. Si es una componente común, deberá estar conectado a exactamente dos de sus vecinos.
5. A cada componente debe llegarle energía, por lo que ambos extremos del cable deben estar conectados a componentes terminales. A continuación se muestra un ejemplo de este caso:



Cumplen las restricciones, pero el cable de la izquierda no tiene energía.



Todas las componentes están alimentadas por el tipo correcto de energía.

Deberás escribir un programa en C que reciba la configuración de una placa y entregue las conexiones necesarias para cumplir las condiciones anteriores. Se espera que lo resuelvas usando **Backtracking**, y para que este sea suficientemente rápido, se espera que mejores su rendimiento usando las siguientes herramientas a tu criterio:

- Modelar el problema usando estructuras de datos de modo que cada operación de Backtracking sea $\mathcal{O}(1)$
- Utilizar estructuras de datos para poder hacer podas que sean $\mathcal{O}(1)$ de calcular en cada paso.
- Utilizar estructuras de datos para poder hacer heurísticas que sean $\mathcal{O}(log(n))$ de calcular en cada paso[1].

Modelación

Deberás entregar un informe donde respondas lo siguiente, independiente de que lo hayas implementado o no:

- Como modelar el problema para que cada operación de Backtracking sea $\mathcal{O}(1)$.
- Propón una poda y como modelarla de manera de que sea $\mathcal{O}(1)$ de calcular en cada paso.
- Propón una heurística y como modelarla de manera que sea $\mathcal{O}(log(n))$ de calcular en cada paso.

Se recomienda encarecidamente que escribas este informe **ANTES** de ponerte a programar la solución.

[1] Siendo n la cantidad de componentes.

Interfaz Gráfica

Tus ayudantes han preparado una interfaz gráfica para poder visualizar el problema y su resolución. Los comandos son los siguientes:

- `watcher_open(height, width)`: Abre una ventana con una placa vacía de `height` filas y `width` columnas.
- `watcher_snapshot(filename)`: Imprime un archivo de nombre `filename.pdf` con el contenido actual de la ventana.
- `watcher_set_component(row, col, color, end)`: Dibuja en la fila `row` y la columna `col` un componente de color `color`. `end` indica si el componente es terminal o no.
- `watcher_set_hub(row, col, capacity)`: Dibuja en la fila `row` y la columna `col` hay un hub de capacidad `capacity`.
- `watcher_set_cord(row_i, col_i, row_f, col_f, color)`: Dibuja que entre el componente de la posición `(row_i, col_i)` y el de la posición `(row_f, col_f)` un cable de color `color`.
- `watcher_remove_cord(row_i, col_i, row_f, col_f)`: Borra el cable entre el componente de la posición `(row_i, col_i)` y el de la posición `(row_f, col_f)`.
- `watcher_cell_highlight(row, col, status)`: Ilumina o apaga, según `status`, el componente en la posición `(row, col)`. Útil para debugear.
- `watcher_close()`: Libera los recursos utilizados por la ventana

Input

Tu tarea esta debe recibir el siguiente input:

1. La ruta del archivo `.txt` que contiene la la información de la placa y sus componentes.

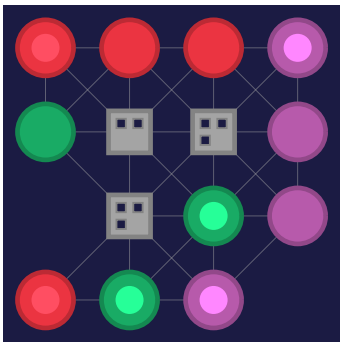
El archivo posee la siguiente estructura:

- La primera línea contiene los números **h** y **w** que indican la cantidad de filas y columnas que tendrá la placa, respectivamente.
- Las siguientes **h** líneas contienen **w** pares de números **x y** separados por espacios de la siguiente forma:
  - **x** representa el color del componente. Si es **0**, es un hub.
  - **y** indica si el componente es terminal **(1)** o si es común **(0)**. En el caso de los hubs, indica la capacidad.

  En caso de ser **0 0**, se habla de una celda vacía de la placa.

A continuación un archivo de ejemplo, con su respectiva representación gráfica:

```
4 4
1 1 1 0 1 0 5 1
3 0 0 2 0 3 5 0
0 0 0 3 3 1 5 0
1 1 3 1 5 1 0 0
```



Output

El **output** de tu programa debe ser la visualización de la placa con el cableado correcto, por lo cual debes asegurarte que el resultado final sea correcto y se muestre correctamente en la ventana.

Evaluación

La nota de tu tarea se descompone como se detalla a continuación:

- 60% la nota de tu código.
- 40% la nota de tu informe.

Tu código será evaluado con distintos tests de dificultad creciente. Se leerá solo el estado final de la ventana, en el momento que llames a `watcher_close()`

Cada test tendrá un límite de 10 segundos, si no respondes en ese margen de tiempo se considerará que no lo resolviste y tendrás 0 puntos en ese test. Se recomienda que para entregar, no pierdas tiempo imprimiendo estados intermedios de tu solución, y no uses sleeps dentro de tu código.

Entrega

- **Lugar:** GIT - Repositorio asignado (asegúrate de seguir la estructura inicial de éste).
  - En la carpeta *Programa* debe encontrarse el código.
  - En la carpeta *Informe* debe encontrarse un archivo `Informe.pdf` con la modelación de la tarea.
  - Se recogerá el estado en la rama *master*.
- **Hora:** 1 minuto pasadas las 23:59 del día de la entrega.
- Se espera que el código compile con `make` dentro de la carpeta *Programa* y genere un ejecutable de nombre `planner` en esa misma carpeta.
- No se permitirán entregas atrasadas.

Bonus

- **Carrera de duendecillos (+20% a la nota de *Código*)** :
  Las 10 tareas más rápidas competirán entre ellas en un set especial de tests superdifíciles. La mejor de ellas obtendrá un +20% a su nota de código, y la peor un +11% a su nota de código.
- **Manejo de memoria perfecto (+5% a la nota de *Código*)**:
  Se aplicará este bonus si **valgrind** reporta en tu código 0 leaks y 0 errores de memoria, considerando que tu programa haga lo que tiene que hacer.
- **Buen uso de espacio y del formato (+5% a la nota del *Informe*)** :
  La nota de tu informe aumentará un 5% si tu informe está bien presentado y usa el espacio y formato de manera de transmitir mejor la información. A juicio del corrector.