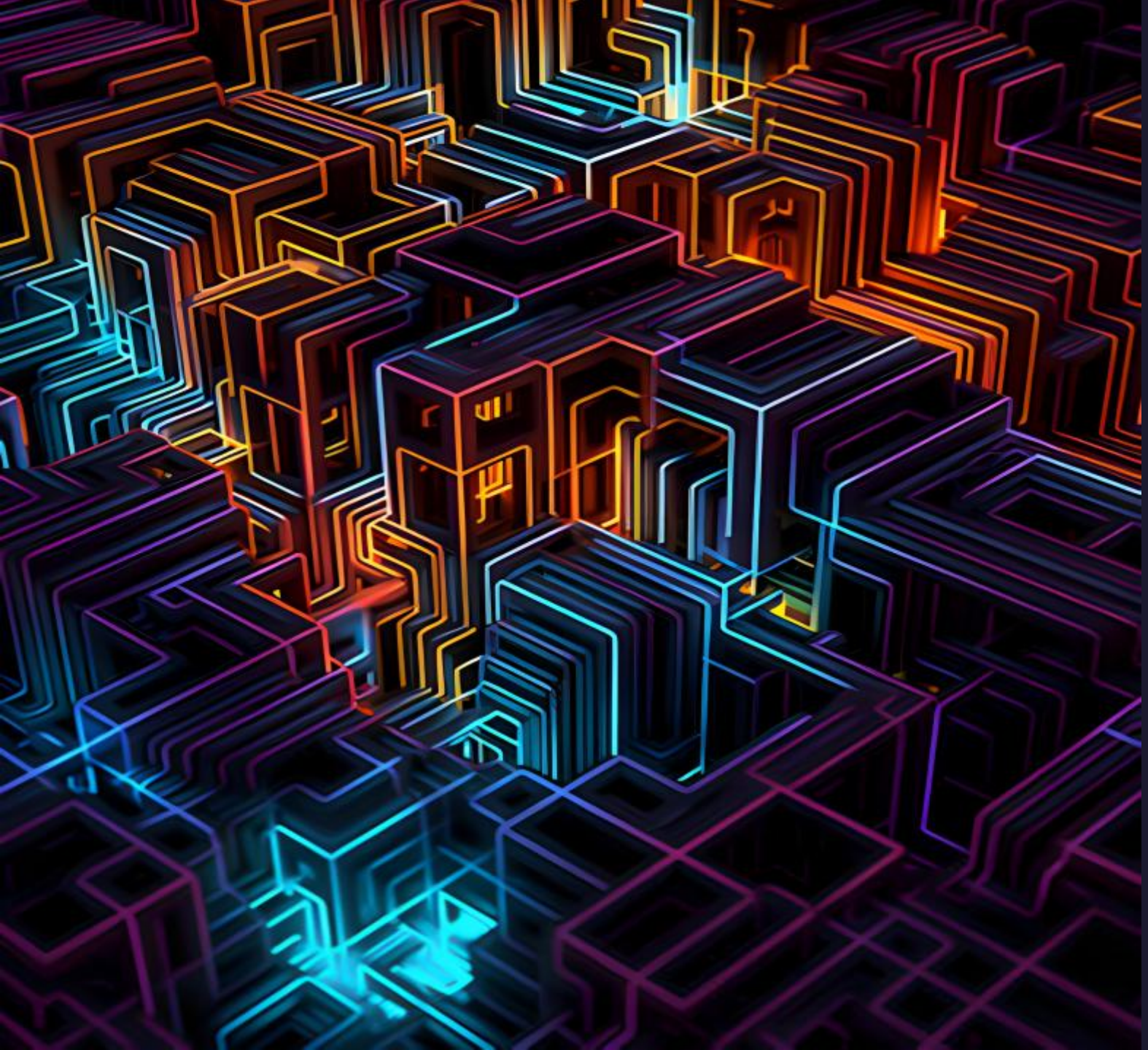


Técnicas de diseño de algoritmos: algoritmos greedy

Análisis y diseño de algoritmos avanzados

Dra. Valentina Narváez Terán

Tecnológico de Monterrey

Técnicas de diseño: algoritmos greedy

- Para problemas de optimización
- La idea: construir soluciones paso a paso, tomando la mejor decisión disponible en cada paso
    - En algunos problemas, esto puede producir óptimos. En otros no.
- Así que son un tipo de método aproximado
    - Métodos exactos vs aproximados
      Los métodos exactos producen óptimos asegurados, los aproximados no necesariamente.

Otros nombres: algoritmos avaros, codiciosos, voraces, glotones

¿Problemas?
Caminos mas cortos en grafos (Alg. de Dijkstra)
Spanning trees (Alg de Prim y Kruskal)

Knapsack, TSP, etc…

Técnicas de diseño: algoritmos greedy

- Los algoritmos greedy se basan en ideas intuitivas sobre como tomar buenas decisiones en cada etapa del problema

Cada decisión es:

- Factible: no rompe las restricciones del problema
- Localmente óptima: es la mejor elección posible en el paso actual, pero no necesariamente global
- Irrevocable: las decisiones tomadas en pasos anteriores no se altera en los próximos pasos (generalmente)

Tenemos la esperanza de que una secuencia de elecciones localmente optimas lleve al optimo global. Pero puede que no pase.

La perspectiva (mas o menos) completa

Algoritmos

Cuando tratamos con problemas de optimización, hay dos grandes ramas de algoritmos

Exactos
Óptimos asegurados

- Búsqueda exhaustiva
- B&B Backtraking
- Solvers declarativos (como CPLEX)

Aproximados
Óptimos no asegurados

Heurísticas

Intuiciones educadas sobre el problema, para construir soluciones

- Algs. Greedy
- Programación dinámica
- Randomized algs

Metaheurísticas

Heurísticas sobre heurísticas: Mejorar soluciones existentes, vía modificación y/o combinación

- Monosolución
- Poblacionales

Hiperheurísticas

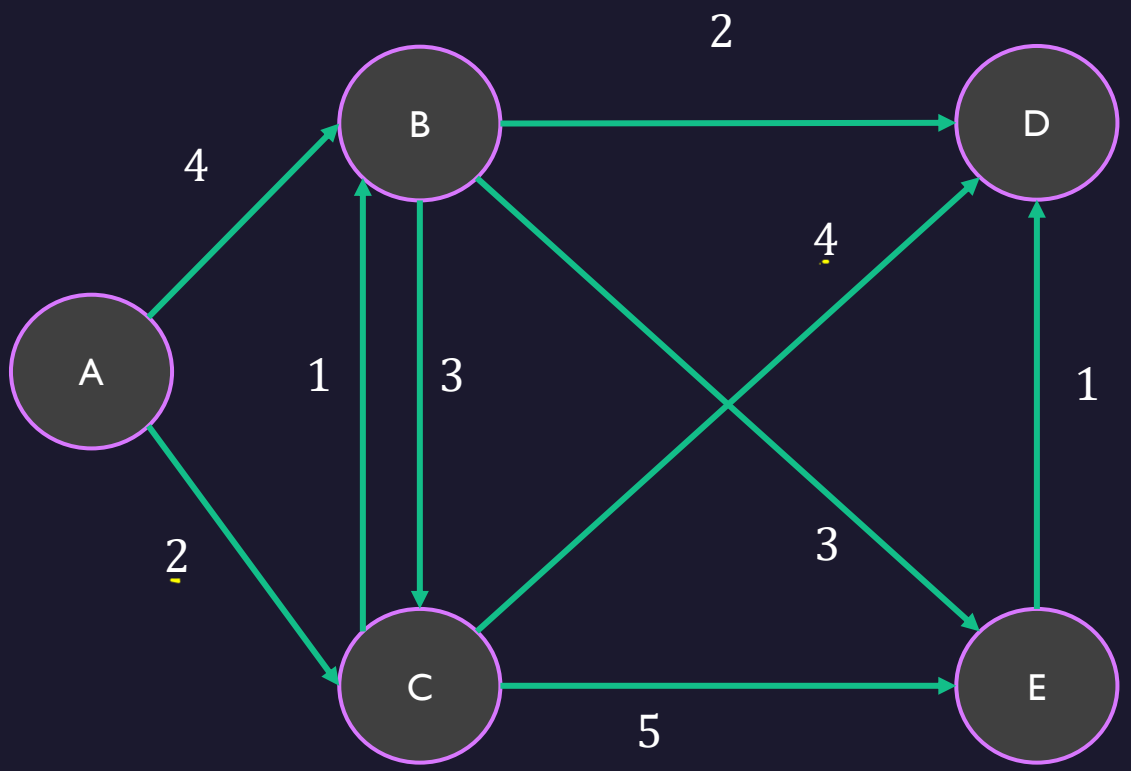
Optimización de metaheurísticas y sus componentes

En estas ramas, la investigación se centra en: algoritmos para problemas nuevos, algoritmos mejores que los existentes, estudios empíricos de dificultad… etc., etc., etc.

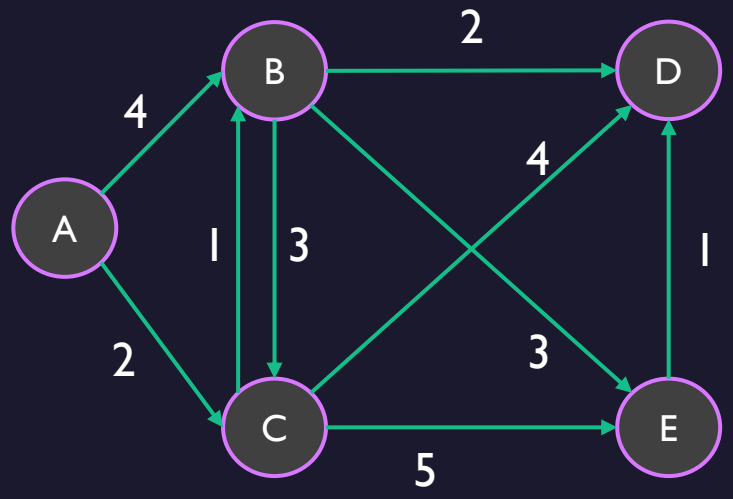
…

Algoritmo de Dijkstra

Es un algoritmo para obtener el camino mas corto (menos costoso) desde un vértice de origen hacia todos los demás, en grafos con aristas ponderadas



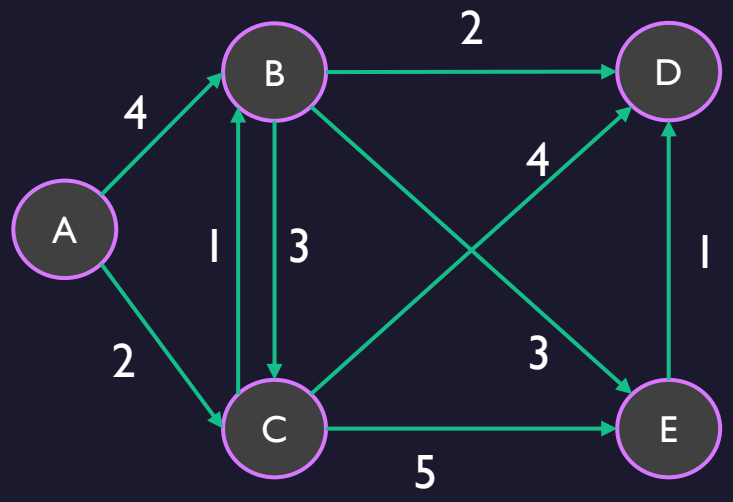
Algoritmo de Dijkstra



	A	B	C	D	E
distancia					
procesado					
desde					

```
origen = el nodo inicial

distancia[0 a n-1] = Inf
distancia[origen] = 0

procesado[0 a n-1] = 0
desde[0 a n-1] = 0

for i desde 0 a n-1
    u = nodo sin procesar con valor minimo en distancia
    procesado[u] = 1          #  u se marca en procesado

    for cada j vecino de u
        nueva_distancia = distancia[u] + peso de arista [u, j]

        if nueva_distancia < distancia[j]
            distancia[j] = nueva_distancia
            desde[j] = u
```

Algoritmo de Dijkstra



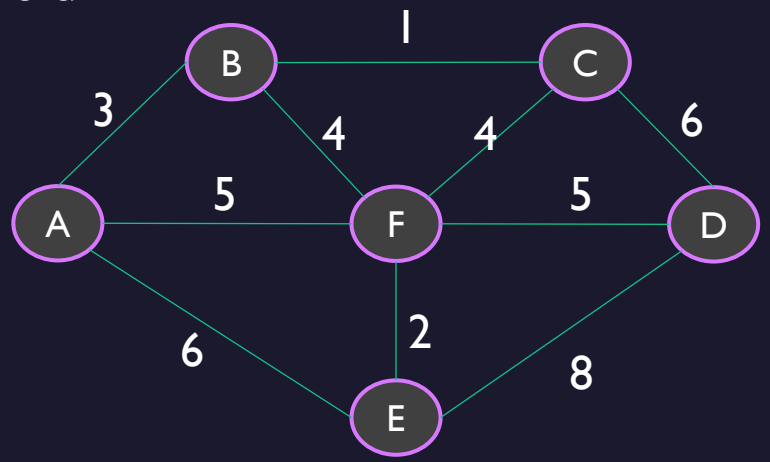
	A	B	C	D	E
distancia	0	3	2	5	6
procesado	1	1	1	1	1
desde	A	C	A	B	B

```
origen = el nodo inicial

distancia[0 a n-1] = Inf
distancia[origen] = 0

procesado[0 a n-1] = 0
desde[0 a n-1] = 0

for i desde 0 a n-1
    u = nodo sin procesar con valor minimo en distancia
    procesado[u] = 1          #  u se marca en procesado

    for cada j vecino de u
        nueva_distancia = distancia[u] + peso de arista [u, j]

        if nueva_distancia < distancia[j]
            distancia[j] = nueva_distancia
            desde[j] = u
```

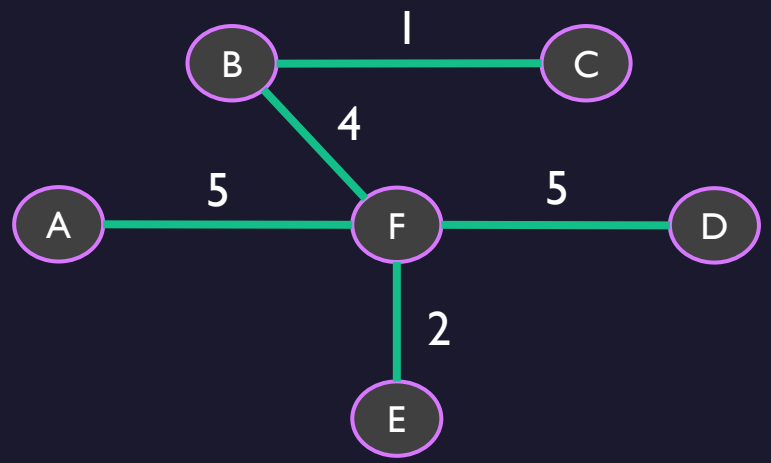
¿Por qué es greedy? Porque en cada paso del proceso, elije la arista que contribuye menos a la distancia total

Árbol de cobertura mínimo (min spanning tree)

Grafo G



Ejemplo de spanning tree de G ¿será mínimo?



Spanning tree: un grafo con los mismos vértices que G, pero sin ciclos

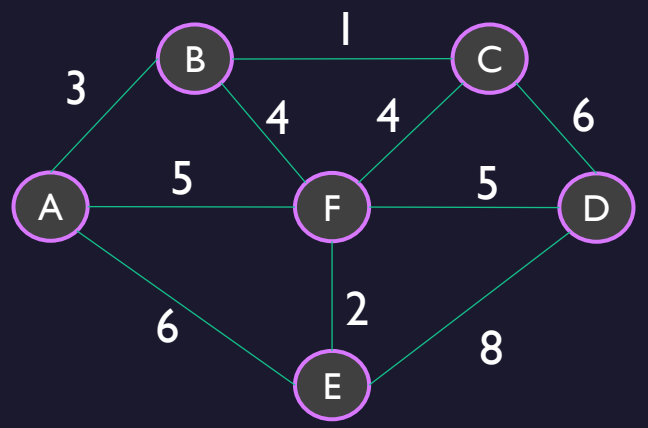
El spanning tree mínimo de un grafo ponderado tiene además la menor suma de costos en todas las aristas

¿Y sirve para…?

Planeación de infraestructura (redes, transporte, circuitos), ej. si necesitamos conectar varios lugares, reduciendo costos/tiempos totales

Algoritmo de Prim

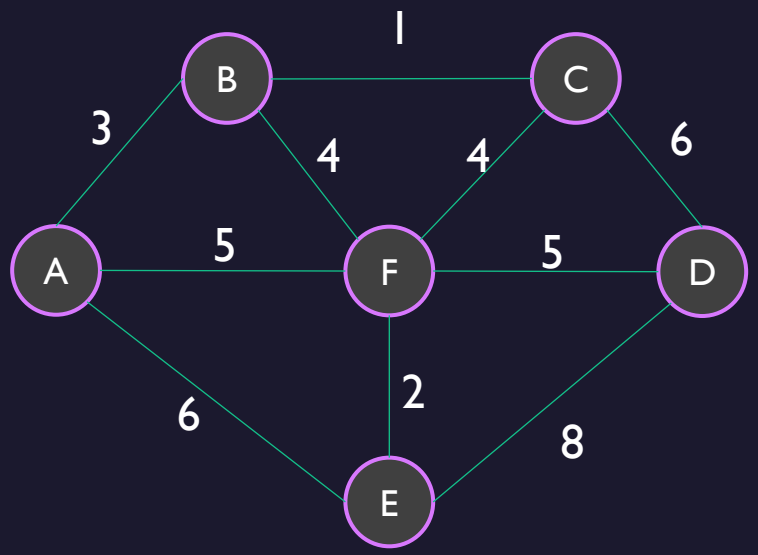
Es una modificación menor con respecto a Dijkstra



	A	B	C	D	E	F
distancia						
procesado						
desde						

```
distancia[0] = 0
distancia[1 a n-1] = Inf

desde[0] = -1

while falten nodos de procesar:
    u = nodo sin procesar con valor minimo en distancia
    procesado[u] = 1 #  u se marca en procesado

    for cada j vecino de u
        nueva_distancia = peso de arista [u, j]

        if nueva_distancia < distancia[j]
            distancia[j] = nueva_distancia
            desde[j] = u
```

Algoritmo de Kruskal



Otra alternativa para obtener el minimum spanning tree

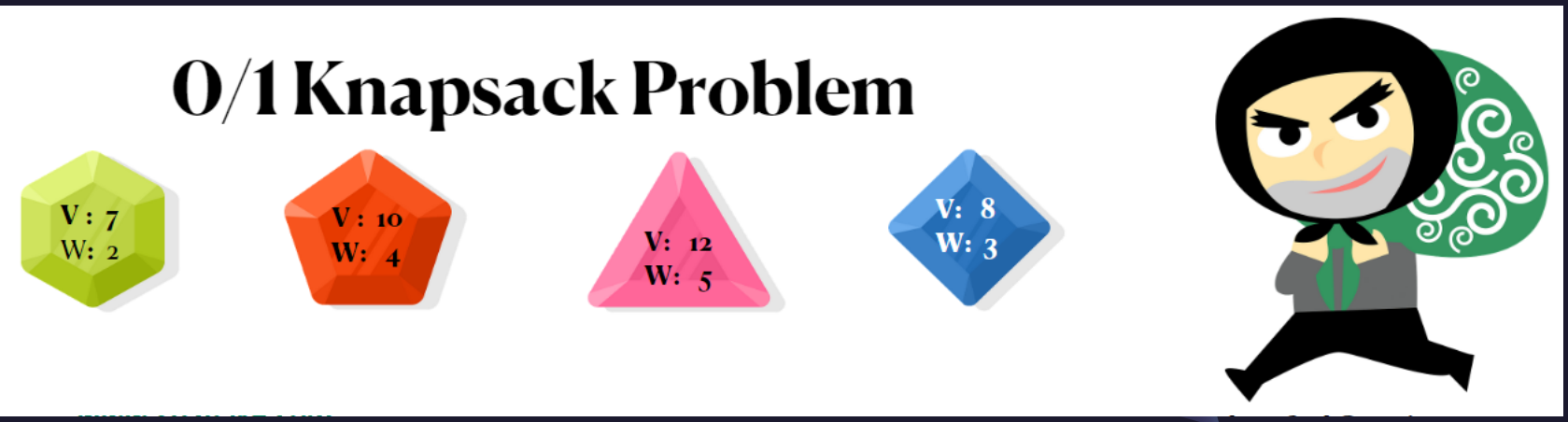
¿Por qué es greedy?

```
arista = aristas ordenadas por costo, ascendentemente

k = 0        //  aristas procesadas
T = grafo vacio
nSelected = 0
while nSelected < N-1
    // Recorre las aristas en orden
    agrega arista[k] a T
    if G tiene un ciclo:
        remueve arista[k] de T
    else
        nSelected++
    k++
```

Revisitando knapsack

Knapsack problem

Tienes un bolso y puedes cargar máximo $W = 10$
¿Cuáles objetos te llevas para maximizar su valor acumulado total V?



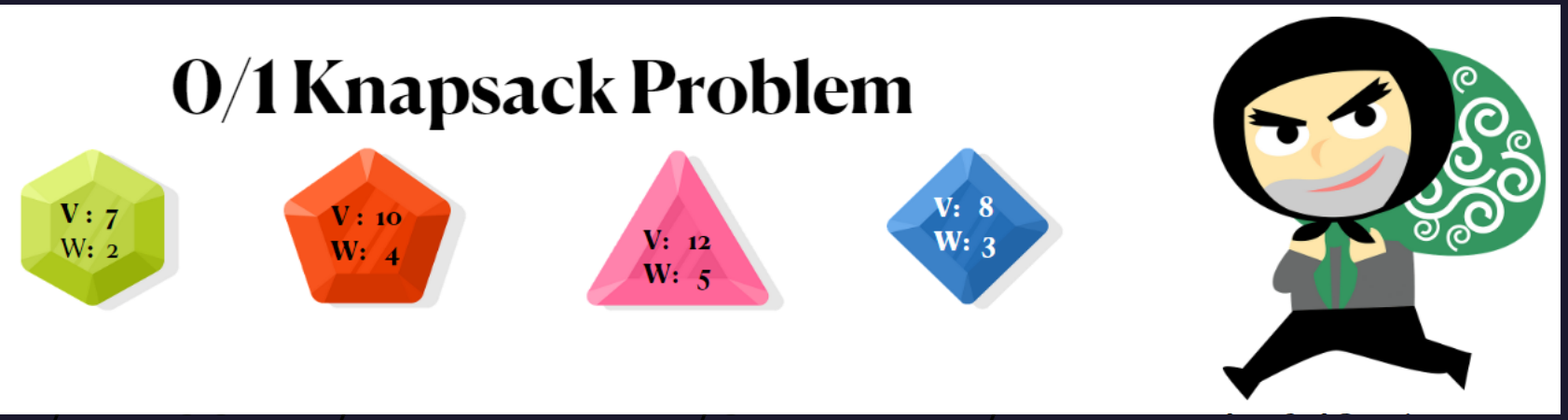
¿Cómo seria un algoritmo greedy para knapsack?

Revisitando knapsack

Idea:

Ordenar los objetos de acuerdo al ratio de su valor por peso, mas valiosos primero

Elegir los primeros i objetos que no sobrepasan la capacidad W



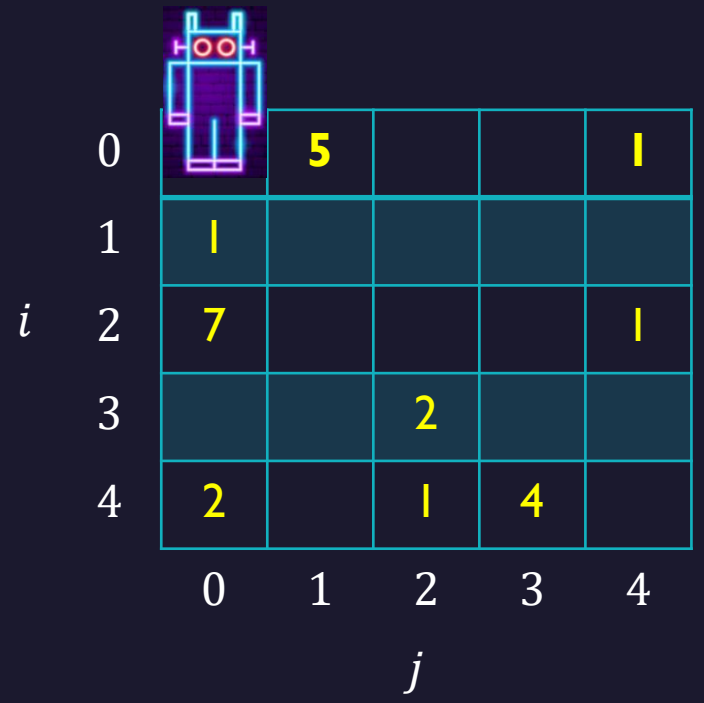
	V	W	V/W
verde	7	2	3.5
azul	8	3	2.6
rojo	10	4	2.5
rosa	12	5	2.4

Elegidos
Valor = 9

Ejemplo: coin-collecting (o gold mining)

Tienes una cuadricula de tamaño n. En las celdas marcadas hay monedas de oro

Un robot minero, que inicia en $[0,0]$, debe recolectarlas… pero esta averiado!
El robot solo se puede mover a la derecha o hacia abajo, nunca a izquierda o arriba.



¿Cuál es la máxima cantidad de monedas que puede recolectar?

¿Qué tal si las monedas tuvieran diferentes valores? Ej. de $1, 2\ y\ 5$

¿Cómo seria un algoritmo greedy?

En su salida muestra las monedas recolectadas y el camino elegido.

Actividad 1.6 Algoritmos greedy

- Algoritmos greedy para:
  - Knapsack (trata de pensar e implementar alguna mejora)
  - Coin collecting

  Puedes encontrar instancias en Teams (clase 08)

Individual