Teoría de Algoritmos
Capitulo 3: Algoritmos Greedy
Tema 9: Heurísticas greedy

- Algoritmos greedy como heurísticas
  - El Problema del Coloreo de un Grafo
  - El problema del Viajante de Comercio
  - El Problema de la Mochila

Heurísticas

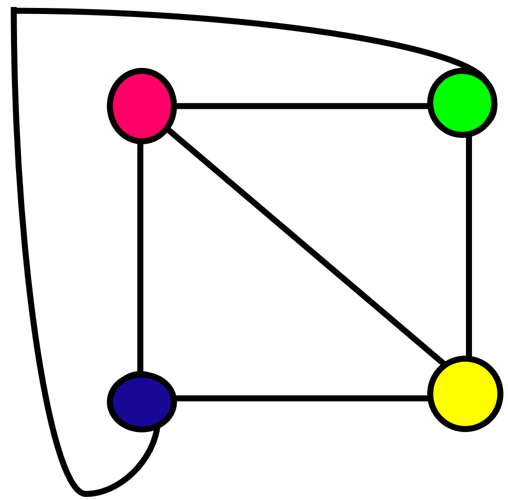
- Son procedimientos que, basados en la experiencia, proporcionan buenas soluciones a problemas concretos
  - Algoritmos Genéticos, Enfriamiento (Recocido) Simulado, Búsqueda Tabú,
  - Computación Evolutiva, GRASP (Greedy Randomized Adaptive Search Procedures), Búsqueda Dispersa, Colonias de Hormigas, Búsqueda por Entornos Variables, Búsqueda Local Guiada, Búsqueda Local Iterativa
  - Métodos Ruidosos, aceptación de umbrales, Algoritmos Miméticos, Redes de Neuronas, ...

Heurísticas Greedy

- Es mejor satisfacer que optimizar
- El tiempo efectivo que se tarda en resolver un problema es un factor clave
- Los algoritmos greedy son muy buenos como heurísticas
  - El problema del coloreo de un grafo
  - El problema del Viajante de Comercio
  - El problema de la Mochila
  - ...
- Suelen usarse también para encontrar una primera solución (optimo local)

El Problema del Coloreo de un Grafo

- Planteamiento
  - Dado un grafo plano G=(V, E), determinar el mínimo numero de colores que se necesitan para colorear todos sus vertices, y que no haya dos de ellos adyacentes pintados con el mismo color
- Si el grafo no es plano puede requerir tantos colores como vertices haya
- Las aplicaciones son muchas
  - Representación de mapas
  - Diseño de paginas webs
  - Diseño de carreteras



El Problema del Coloreo de un Grafo

- El problema es NP y por ello se necesitan heuristicas para resolverlo
- El problema reune todos los requisitos para ser resuelto con un algoritmo greedy
- Del esquema general greedy se deduce un algoritmo inmediato
- Teorema de Appel-Hanke (1976): Un grafo plano requiere a lo sumo 4 colores para pintar sus nodos de modo que no haya vertices adyacentes con el mismo color

El Problema del Coloreo de un Grafo

- Suponemos que tenemos una paleta de colores (con mas colores que vertices)
- Elegimos un vertice no coloreado y un color. Pintamos ese vertice de ese color
- Lazo greedy: Seleccionamos un vertice no coloreado v. Si no es adyacente (por medio de una arista) a un vertice ya coloreado con el nuevo color, entonces coloreamos v con el nuevo color
- Se itera hasta pintar todos los vertices

Implementacion del algoritmo

Funcion **COLOREO**

{COLOREO pone en NuevoColor los vertices de G que pueden tener el mismo color}

Begin

NuevoColor = ∅

Para cada vertice no coloreado v de G Hacer

Si v no es adyacente a ningun vertice en NuevoColor Entonces

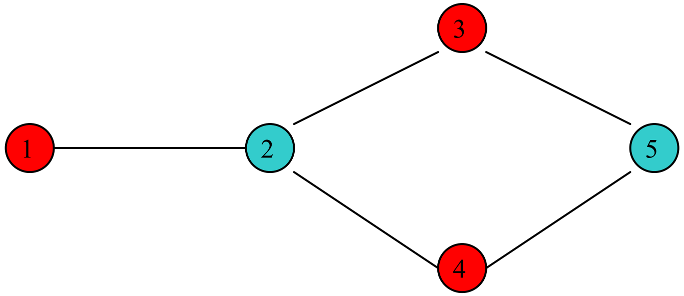
Marcar v como coloreado

Añadir v a NuevoColor

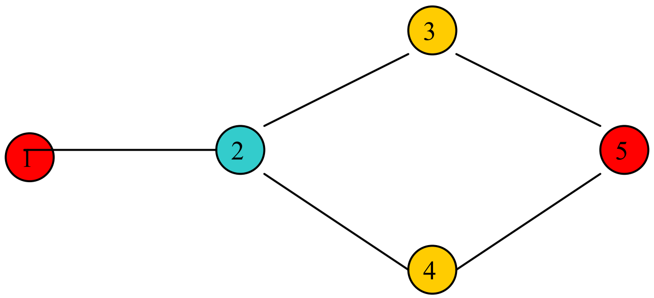
End

Se trata de un algoritmo que funciona en **O(n),** pero que no siempre da la solución optima

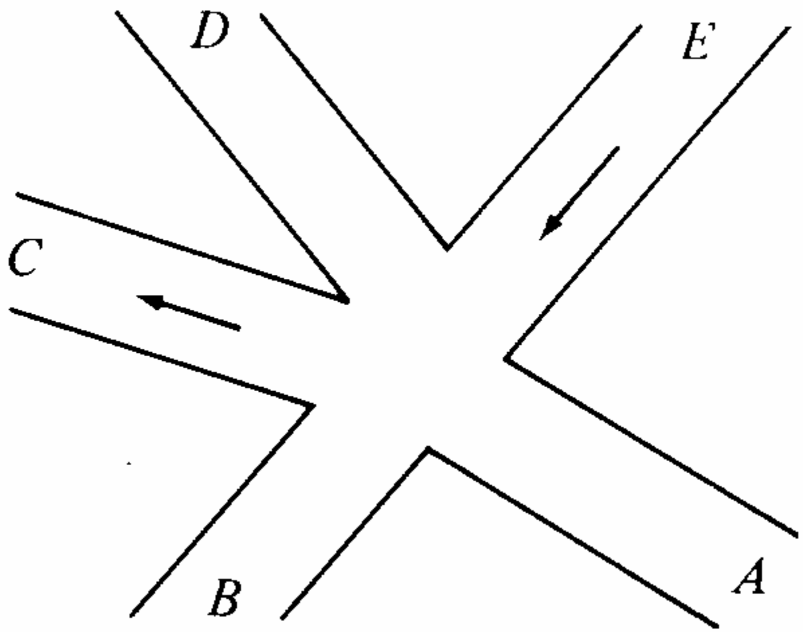
Ejemplo



El orden en el que se escogen los vertices para colorearlos puede ser decisivo: el algoritmo da la solucion optimal en el grafo de arriba, pero no en el de abajo



Ejemplo: Diseño de cruces de semaforos



- A la izquierda tenemos un cruce de calles
- Se señalan los sentidos de circulacion.
- La falta de flechas, significa que podamos ir en las dos direcciones.
- Queremos diseñar un patron de semaforos con el minimo numero de semaforos, lo que
- Ahorrara tiempo (de espera) y dinero
- Suponemos un grafo cuyos vertices representan turnos, y cuyas aristas unen esos turnos que no pueden realizarse simultaneamente sin que haya colisiones, y el problema del cruce con semaforos se convierte en un problema de coloreo de los vertices de un grafo

Un Refinamiento del Pseudocodigo

```
        procedure greedy ( var G: GRAPH; var newclr: SET );
            begin
(1)             newclr := ∅;
(2)             for each uncolored vertex v of G do begin
(3.1)               found := false;
(3.2)               for each vertex w in newclr do
(3.3)                   if there is an edge between v and w in G then
(3.4)                       found := true;
(3.5)               if found = false then begin
                        { v is adjacent to no vertex in newclr }
(4)                     mark v colored;
(5)                     add v to newclr
                    end
                end
            end; { greedy }
```

Segundo Refinamiento del Pseudocodigo

```
procedure greedy ( var G: GRAPH; var newclr: LIST );
    { greedy assigns to newclr those vertices that may be
        given the same color }
    var
        found: boolean;
        v, w:  integer;
    begin
        newclr := ∅;
        v := first uncolored vertex in G;
        while v <> null do begin
            found := false;
            w := first vertex in newclr;
            while w <> null do begin
                if there is an edge between v and w in G then
                    found := true;
                w := next vertex in newclr
            end;
            if found = false do begin
                mark v colored;
                add v to newclr
            end;
            v := next uncolored vertex in G
        end
    end; { greedy }
```

El Problema del Viajante de Comercio

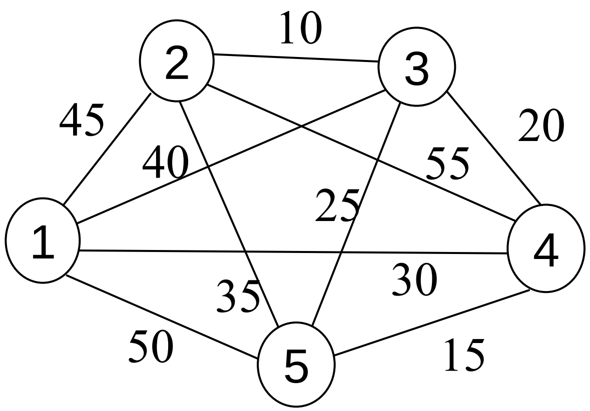
- Un viajante de comercio que reside en una ciudad, tiene que trazar una ruta que, partiendo de su ciudad, visite todas las ciudades a las que tiene que ir una y sólo una vez, volviendo al origen y con un recorrido mínimo
- Es un problema NP, no existen algoritmos en tiempo polinomial, aunque si los hay exactos que lo resuelven para grafos con 40 vértices aproximadamente.
- Para más de 40, es necesario utilizar heurísticas, ya que el problema se hace intratable en el tiempo.
- El PVC es uno de los mas importantes en Teoría de Algoritmos

El Problema del Viajante de Comercio

- Supongamos un grafo no dirigido y completo G = (N, A) y L una matriz de distancias no negativas referida a G. Se quiere encontrar un **Circuito Hamiltoniano Minimal**.
- Este es un problema Greedy típico, que presenta las 6 condiciones para poder ser enfocado con un algoritmo greedy
- Destaca de esas 6 características **la condición de factibilidad**:
  - que al seleccionar una arista no se formen ciclos,
  - que las aristas que se escojan cumplan la condición de no ser incidentes en tercera posición al nodo escogido

El Problema del Viajante de Comercio

- Consideremos el siguiente grafo



- Posibilidades:
  - Los nodos son los candidatos. Empezar en un nodo cualquiera y en cada paso moverse al nodo no visitado más próximo al último nodo seleccionado.
  - Las aristas son los candidatos. Hacer igual que en el Algoritmo de Kruskal, pero garantizando que se forme un ciclo.

El problema de la Mochila

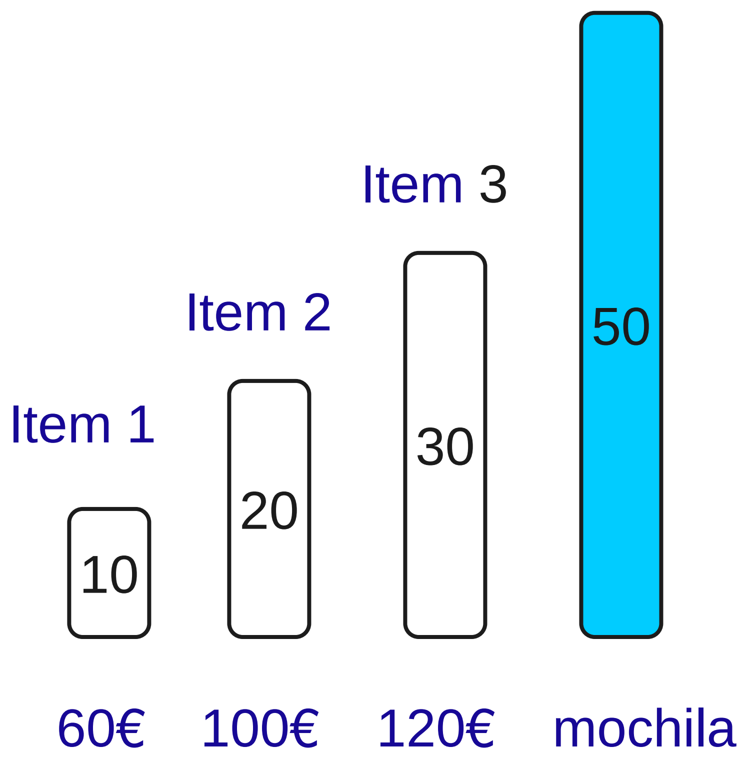
- Tenemos n objetos y una mochila. El objeto i tiene un peso w_i y la mochila tiene una capacidad M.
- Si metemos en la mochila la fraccion x_i, $0 \leq x_i \leq 1$, del objeto i, generamos un beneficio de valor $p_i x_i$
- El objetivo es rellenar la mochila de tal manera que se maximice el beneficio que produce el peso total de los objetos que se transportan, con la limitacion de la capacidad de valor M

$$\text{maximizar} \quad \sum_{1 \leq i \leq n} p_i x_i$$

$$\text{sujeto a} \quad \sum_{1 \leq i \leq n} w_i x_i \leq M$$

con $0 \leq xi \leq 1$, $1 \leq i \leq n$

Ejemplo



Es un claro problema de tipo greedy

Sus aplicaciones son inumerables

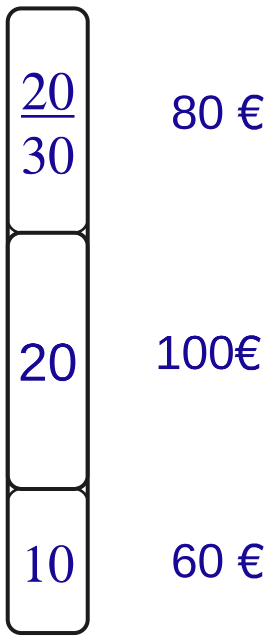
Es un banco de pruebas algoritmico

La tecnica greedy produce soluciones optimales para este tipo de problemas

Mochila fraccional

- Tomando los items en orden de mayor valor por unidad de peso, se obtiene una solucion optimal

{60/10, 100/20, 120/30}



Total = 240 €

Mochila fraccional

- Supongamos 5 objetos de peso y precios dados por la tabla, la capacidad de la mochila es 100.

Price ($US)	20	30	65	40	50
weight (Lbs.)	10	20	30	40	50

- Metodo 1 elegir primero el menos pesado
  - Peso total = 10 + 20 + 30 + 40 = 100
  - Costo total = 20 + 30 + 65 + 40 = 155
- Metodo 2 elegir primero el mas caro
  - Peso Total = 30 + 50 + 20 = 100
  - Costo Total = 65 + 50 + 20 = 135

Otro ejemplo de Mochila Fraccional

- Supongamos el siguiente caso de problema de la mochila: $n = 3$, $M = 20$, $(p_1,p_2,p_3) = 25,24,15$ y $(w_1,w_2,w_3) = (18,15,10)$

	(x1,x2,x3)	$\sum w_i x_i$	$\sum p_i x_i$
1)	(1/2,1/3,1/4)	16.5	24.25
2)	(1,2/15,0)	20	28.2
3)	(0,2/3,1)	20	31
4)	(0,1,1/2)	20	31.5

Solucion Greedy

- Definimos la densidad del objeto Ai por w_i/s_i.
- Se usan objetos de tan baja densidad como sea posible, es decir, los seleccionaremos en orden creciente de densidad.
- Si es posible se coge todo lo que se pueda de Ai, pero si no se rellena el espacio disponible de la mochila con una fraccion del objeto en curso, hasta completar la capacidad, y se desprecia el resto.
- Primero, se ordenan los objetos por densidad no decreciente, i.e.:

$$w_i/s_i \leq w_{i+1}/s_{i+1} \text{ for } 1 \leq i < n.$$

- Entonces se actúa de la siguiente manera

PseudoCodigo

```
Procedimiento MOCHILA_GREEDY(P,W,M,X,n)
//P(1:n) y W(1:n) contienen los costos y pesos respectivos de los
    n objetos ordenados como P(I)/W(I) ≥  P(I+1)/W(I+1). M es
    la capacidad de la mochila y X(1:n) es el vector solution//
real P(1:n), W(1:n), X(1:n), M, cr;
integer I,n;
    x = 0; //inicializa la solucion en cero //
    cr = M;  // cr = capacidad restante de la mochila //
    Para i = 1 hasta n Hacer
        Si W(i) > cr Entonces exit endif
        X(I) = 1;
        cr =  c - W(i);
    repetir
        Si I ≤ n Entonces X(I) = cr/W(I) endif
End MOCHILA_GREEDY
```

Demostración de la correccion

- Vamos a demostrar que el algoritmo siempre encuentra la solucion optimal del problema
- Sea $p_1/w_1 \geq p_2/w_2 \geq \ldots \geq p_n/w_n$
- Sea $X = (x_1, x_2, \ldots, x_n)$ la solucion generada por MOCHILA_GREEDY
- Sea $Y = (y_1, y_2, \ldots, y_n)$ una solucion factible cualquiera
- Queremos demostrar que

$$\sum_{i=1}^{n} (x_i - y_i) p_i \geq 0$$

- Esta demostración hay que estudiarla en el libro de Horowitz y Sahni.