Welcome to
Algorithms and Data Structures! -
CS2100

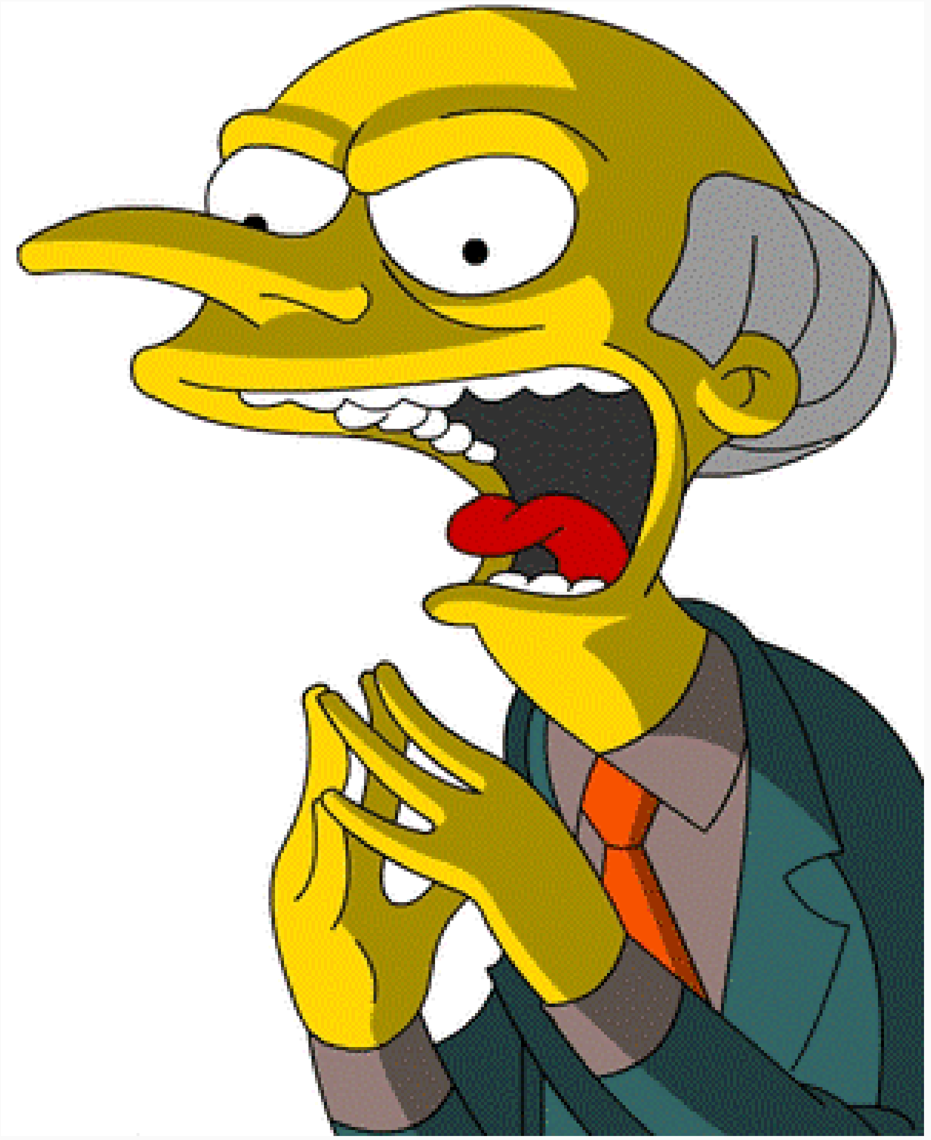
Algoritmos voraces, golosos, codiciosos (Greedy algorithms)

1. **Qué es?**
   Siempre busca la mejor solución local, esperando tener la mejor solución global

2. **Cuál es el problema de esto?**
   Se ignora el efecto a futuro.
   No necesariamente llega a la solución optima global

3. **Algoritmos voraces tienen dos propiedades**
   a. Subestructuras óptimas
   b. Elección codiciosa

Para muchos problemas, utilizar un algoritmo voraz puede fallar. **Entonces en qué casos se podría utilizar?**



Algoritmos voraces (esquema genérico)

```
función voraz(C:conjunto)
              devuelve conjunto
{C es el conjunto de todos los candidatos}
principio
  S:=Ø; {S es el conjunto en el que se
          construye la solución}
  mq ¬solución(S) ∧ C≠Ø hacer
    x:=elemento de C que
         maximiza seleccionar(x);
    C:=C-{x};
    si completable(S∪{x})
      entonces S:=S∪{x}
    fsi
  fmq;
  si solución(S)
    entonces devuelve S
    sino devuelve no hay solución
  fsi
fin
```

Algoritmos voraces, golosos, codiciosos (Greedy algorithms)

Como implementamos PRIM y KRUSKAL con algoritmos voraces?

Kruskal

```
función voraz(C:conjunto)
                devuelve conjunto
{C es el conjunto de todos los candidatos}
principio
  S:=Ø; {S es el conjunto en el que se
           construye la solución}
  mq ¬solución(S) ∧ C≠Ø hacer
    x:=elemento de C que
           maximiza seleccionar(x);
    C:=C-{x};
    si completable(S∪{x})
      entonces S:=S∪{x}
    fsi
  fmq;
  si solución(S)
    entonces devuelve S
    sino devuelve no hay solución
  fsi
fin
```

Funcion kruskal(Graph(V,E))

Solución(S): árbol que cubre todos los vértices
Seleccionar(C): extraer arista con peso mínimo de C
Completable(X, S): verificar que X no forme ciclo en S

Kruskal

```
función voraz(C:conjunto)
              devuelve conjunto
{C es el conjunto de todos los candidatos}
principio
  S:=Ø; {S es el conjunto en el que se
         construye la solución}
  mq ¬solución(S) ∧ C≠Ø hacer
    x:=elemento de C que
         maximiza seleccionar(x);
    C:=C-{x};
    si completable(S∪{x})
      entonces S:=S∪{x}
    fsi
  fmq;
  si solución(S)
    entonces devuelve S
    sino devuelve no hay solución
  fsi
fin
```

Funcion kruskal(Graph(V,E))

- C = MinHeap(E)
- S = { }: //árbol
- While ¬Solución(S,V) and |C| != 0
  - X = C.extractMin()
  - C = C − X
  - If Completable(X, S):
    - S = S + X
- Devolver S

Solución(S): |S| == |V| árbol que cubre todos los vértices
Seleccionar(C):   extraer arista con peso mínimo de C
Completable(X, S): verificar que X no forme ciclo en S

PRIM

```
función voraz(C:conjunto)
                    devuelve conjunto
{C es el conjunto de todos los candidatos}
principio
  S:=∅; {S es el conjunto en el que se
            construye la solución}
  mq ¬solución(S) ∧ C≠∅ hacer
    x:=elemento de C que
            maximiza seleccionar(x);
    C:=C-{x};
    si completable(S∪{x})
      entonces S:=S∪{x}
    fsi
  fmq;
  si solución(S)
    entonces devuelve S
    sino devuelve no hay solución
  fsi
fin
```

- Funcion PRIM(Graph(V,E))
  - C = G(V, E) //
  - S =   // vértices visitados
  - T = // Arbol solucion
  - While ¬Solución(S,V) and |C| != 0
    - (u, v) = Seleccionar(C)
    - S = S U v
    - T = T U (u, v)

Solución(S): |S| == |V|
Seleccionar(C):   devolver un arista adyacente a u de S con el menor peso
Completable(X, S): No se aplica

Welcome to
Algorithms and Data Structures! - CS2100