UNIVERSIDADE DA CORUÑA

Algoritmos:
Exploración de grafos

Alberto Valderruten

LFCIA - Departamento de Computación

Facultad de Informática

Universidad de A Coruña, España

www.lfcia.org/alg

www.fi.udc.es

Contenido

- Juegos de estrategia
  - Juego de Nim
- Recorridos sobre grafos
  - en profundidad
  - en anchura
- Retroceso
  - Problema de las 8 reinas

Juegos de estrategia

- **Exploración de grafos**:

  ¿ problema $\leftrightarrow$ recorrido sobre un grafo ?

- **Juegos de estrategia**

- Ejemplo: variante del **Juego de Nim**
  - Un montón de n palillos
  - 2 jugadores, alternativamente
  - 1ª jugada: coger $[1..n-1]$ palillos
  - jugadas siguientes: coger $[1.,2 * \text{jugada anterior}]$ palillos
  - Objetivo: coger el último palillo

- Grafo: $\begin{cases} \text{nodo} & \leftrightarrow \text{situación: } < \text{quedan } i \text{ palillos, se pueden coger } j > \\ \text{arista} & \leftrightarrow \text{jugada: nº de palillos} \end{cases}$

- Ejemplo: desde $i = 5, j = 4$ (i.e. jugada anterior $= 2$) y **juego yo**

Juego de Nim (1)

- Marcado de nodos:
  - 2 tipos de nodos: $\begin{cases} \text{situación de derrota} \\ \text{situación de victoria} \equiv \text{“victoria segura si juego bien”} \end{cases}$
  - Situación final (de derrota): $< 0, 0 >$
- Marcado de jugadas:
  Distinguir las jugadas de victoria
- Reglas de marcado: desde $< 0, 0 >$
  - marcar *situación de victoria* si $\exists$ sucesor *situación de derrota*
  - marcar *situación de derrota* si todos los sucesores son *situación de victoria*

Juego de Nim (2)

- **Función Ganadora**

```
función Ganadora (i,j)      {determina si la situación <i,j> es ganadora}
{hipótesis: 0<=j<=i}
    para k := 1 hasta j hacer
        si no Ganadora (i-k,min(2k,i-k)) entonces devolver verdadero;
    devolver falso
fin función
```

  $\rightarrow$ muy ineficiente

- Programación Dinámica:

  $V[i,j]$ = verdadero $\leftrightarrow < i,j >$ es situación de victoria

  $\rightarrow$ **Procedimiento Ganador**:

  calcula $V[r,s] \forall 1 \leq s \leq r < i$; $V[i,s] \forall 1 \leq s < j \Rightarrow V[i,j]$

  Problema: muchas entradas no se usan nunca

  (ejemplo: para $n = 248$, basta con explorar 1000 nodos;

  Ganador: más de 30 veces esa cantidad)

Juego de Nim (3)

```
procedimiento Ganador (n)
{1<=j<=i<=n y V[i,j]=verdadero <=> la situación <i,j> es ganadora}
    V[0,0]:=falso;
    para i:=1 hasta n hacer
        para j:=1 hasta i hacer
            k:=1;
            mientras k<j y V[i-k,min(2k,i-k)] hacer k:=k+1;
            V[i,j]:=no V[i-k,min(2k,i-k)]
fin procedimiento
```

- ¿Combinar técnicas? $\longrightarrow$ **Función con memoria**

  Recordar los nodos visitados: matriz conocido$[0..n, 0..n]$

  $\longrightarrow$ **Función Nim**

  Problema: inicialización

  $\longrightarrow$ técnica de inicialización virtual

  (una matriz auxiliar indica las posiciones ocupadas)

Juego de Nim (4)

```
{inicialización para Nim}
V[0,0]:=falso; conocido[0,0]:=verdadero;
para i:=1 hasta n hacer
    para j:=1 hasta i hacer conocido[i,j]:=falso

función Nim(i,j)      {determina si la situación <i,j> es ganadora}
    si conocido[i,j] entonces devolver V[i,j];
    conocido[i,j]:=verdadero;
    para k:=1 hasta j hacer
        si no Nim(i-k,min(2k,i-k)) entonces
            V[i,j]:=verdadero;
            devolver verdadero;
    V[i,j]:=falso;
    devolver falso
fin función
```

- La misma técnica se aplica a muchos juegos de estrategia
- Observación: Nim puede resolverse sin recorrer un grafo (con Fibonacci...)

Recorridos sobre grafos: en profundidad (1)

- Recursividad

- $G = (N, A)$: grafo no dirigido
  $\rightarrow$ recorrido a partir de cualquier $v \in N$:

```
procedimiento rp2(v)
    CrearPila(P);
    marca[v] := visitado;
    Apilar(v,P);
    mientras no Pila Vacía (P) hacer
        mientras exista w adyacente a Cima(P) tal que marca[w] != visitado hacer
            marca[w] := visitado;
            Apilar(w,P);
        Desapilar(P)
fin procedimiento
```

- Marca los nodos visitados hasta marcar todos los nodos

- El recorrido en profundidad asocia a un grafo conexo un árbol de recubrimiento

Recorrido en profundidad (2)

- **Análisis**:

  n nodos, m aristas

  se visitan todos los nodos (n llamadas recursivas) $\Rightarrow \Theta(n)$

  en cada visita se inspeccionan todos los adyacentes $\Rightarrow \Theta(m)$

  $$\boxed{T(n) = \Theta(n+m)}$$

- Ejemplo: **numerar en preorden**

  Añadir al principio:

  ```
  visita := visita+1;
  preorden[v] := visita;
  ```

- Grafo dirigido: la diferencia está en la definición de *adyacente*

  árbol $\longrightarrow$ "bosque de recubrimiento"

- Ejemplo: **ordenación topológica** (grafo dirigido acíclico)

  $\longrightarrow$ Añadir al final: `escribir(v);`

  e invertir el resultado

Recorrido en anchura

- Diferencia: al llegar a un nodo v, primero visita todos los nodos adyacentes
  $\longrightarrow$ no es "naturalmente recursivo"

```
procedimiento ra(v)
    CrearCola(C);
    marca[v] := visitado;
    InsertarCola(v,C);
    mientras no ColaVacía(C) hacer
        u := EliminarCola(C);
        para cada nodo w adyacente a u hacer
            si marca[w] != visitado entonces
                marca[w] := visitado;
                InsertarCola(w,C)
fin procedimiento
```

- Grafo conexo $\longrightarrow$ árbol de recubrimiento
- $\boxed{T(n) = \Theta(n+m)}$
- Exploración parcial de grafos grandes, camino más corto entre dos nodos...

Retroceso

- Mecanismo de vuelta atrás, o *backtracking*.
- Recorrido implícito $\equiv$ se va calculando el grafo a medida que se va avanzando en la búsqueda de una solución.
- Ante una solución encontrada: detenerse o buscar soluciones alternativas.
- *Fallo* $\equiv$ no se puede completar la solución que se está construyendo
  $\rightarrow$ retroceso hasta primer nodo con vecinos sin explorar
- Ejemplo: **Problema de las 8 reinas**
  Colocar 8 reinas en un tablero de ajedrez sin que se den jaque.

```
para i1:=1 hasta 8 hacer
    para i2:=1 hasta 8 hacer
      ...
        para i8:=1 hasta 8 hacer
            ensayo := [i1,i2,...,i8];
            si solución (ensayo) entonces devolver ensayo
escribir no hay solución
```

Problema de las 8 reinas

```
ensayo := permutación inicial;
mientras no solución (ensayo) y ensayo <> permutación final hacer
    ensayo := permutación siguiente
si solución (ensayo) entonces devolver ensayo
sino escribir no hay solución
```

- **Procedimiento test**:

```
procedimiento test (k,col,diag1,diag2)
{ensayo[1..k] es k-completable; col = {ensayo[i] : 1 <= i <= k};
diag1 = {ensayo[i]-i+1 : 1 <= i <= k}; diag2 = {ensayo[i]+i-1 : 1 <= i <= k}}
    si k=8 entonces escribir ensayo
    sino
        para j:=1 hasta 8 hacer
            si j no pertenece a col y j-k no pertenece a diag1
                                   y j+k no pertenece a diag2 entonces
                ensayo[k+1] := j ;                          {es (k+1)-completable}
                test ( k+1, col U {j}, diag1 U {j-k}, diag2 U {j+k}
fin procedimiento
```

UNIVERSIDADE DA CORUÑA

Algoritmos:
Exploración de grafos

Alberto Valderruten

LFCIA - Departamento de Computación

Facultad de Informática

Universidad de A Coruña, España

www.lfcia.org/alg

www.fi.udc.es