Algoritmos

Exploración de grafos

Alberto Valderruten

Dept. de Computación, Universidade da Coruña

alberto.valderruten@udc.es

Contenido

Índice

1. Recorridos sobre grafos
2. Juegos de estrategia
3. Retroceso

Recorrido en profundidad (1)

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

```
procedimiento rp (v) {v no ha sido visitado anteriormente}
    marca[v] := visitado;
    para cada w adyacente a v hacer
        si marca[w] != visitado entonces rp(w)
fin procedimiento
```

- El recorrido en profundidad asocia a un grafo conexo un *árbol de recubrimiento*
- **Análisis**: n nodos, m aristas
  - se visitan todos los nodos (n llamadas recursivas) $\Rightarrow \Theta(n)$
  - en cada visita se inspeccionan todos los adyacentes $\Rightarrow \Theta(m)$

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

Recorrido en profundidad (2)

- Ejemplo: **numerar en preorden** → Añadir al principio de rp:

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

- Para recorrer completamente el grafo (incluso no conexo):

```
procedimiento recorridop (G=(N,A))
    para cada nodo v hacer marca[v] := no visitado;
    para cada nodo v hacer
        si marca[v] != visitado entonces rp(v)
fin procedimiento
```

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

  árbol → “bosque de recubrimiento”

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

  → Añadir al final de rp:

```
escribir(v);
```

  e invertir el resultado

Recorrido en profundidad (3)

- Versión no recursiva: usa una Pila

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

Recorrido en anchura (1)

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

```
procedimiento ra (v)
    Crear Cola (C); marca[v] := visitado; Insertar Cola (v,C);
    mientras no Cola Vacía (C) hacer
        u := Eliminar Cola (C);
        para cada nodo w adyacente a u hacer
            si marca[w] != visitado entonces
                marca[w] := visitado; Insertar Cola (w,C)
            fin si
        fin para
    fin mientras
fin procedimiento
```

- Grafo conexo $\rightarrow$ árbol de recubrimiento
- $\boxed{T(n) = \Theta(n+m)}$
- Exploración parcial de grafos, camino más corto entre 2 nodos...

Índice

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 $[1..2 *$ 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**

```
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? $\rightarrow$ **Función con memoria**
  Recordar los nodos visitados: matriz conocido$[0..n, 0..n]$
  $\rightarrow$ **Función Nim**
  Problema: inicialización $\rightarrow$ 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
        fin si;
    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 (fib)

Índice

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 otras soluciones
- *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
```

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
```