Exploración en Grafos

Objetivos

- Recorrido en profundidad de grafos. Aplicaciones
- Recorrido en anchura de grafos. Aplicaciones
- Recorrido con vuelta atrás (*Backtracking*)
- Aplicación del *Backtracking*
- Métodos "Ramifica y poda" (*Branch and Bound*)
- Aplicación del *Branch and Bound*

Grafos

Sea V un conjunto de puntos: $V^2 = V \times V$.

Un **grafo** es un par $G = (V, E)$ con $E \subset V^2$. Los elementos de V se llaman *nodos* o *vértices*. Los elementos de E se llaman *lados*. Notación: $V = \{1, \ldots, n\}$, y $|E| = a$.

- Un grafo se dice *dirigido* cuando el orden de los componentes de los pares de E es relevante: $(x, y) \neq (y, x)$. Sus lados se denominan *arcos*.

- Un grafo se dice *no dirigido* cuando el orden de los compomentes de los pares de E no es relevante. Sus lados se denominan *aristas*.

- Dados $u, v \in V$ se dice que existe un *camino* entre u y v si existe una secuencia de pares de E:

$$(u, v_1), (v_1, v_2), \ldots, (v_n, v)$$

- Se llama *longitud del camino* al número de pares.

Grafos (II)

- Un camino de u a v es un *ciclo* cuando $u = v$.
- Un camino se dice *simple* cuando en él no hay ciclos.
- Dados $u, v \in V$ se dice que v es *adyacente* a u si $(u, v) \in E$.
- Representación natural de objetos y relaciones entre éstos.
- Grafos etiquetados y ponderados.

Casos particulares importantes de grafos:

- Grafos dirigidos acíclicos (GDA).
- Árboles.

Algunas aplicaciones de los grafos

- Representación del conocimiento
- Problemas de búsqueda
- Análisis y verificación de programas
- Redes eléctricas
- Redes de ordenadores y comunicaciones
- Redes de distribución
- Planificación de vuelos
- Planificación de exámenes
- Encontrar secuencias óptimas de procesos

Recorridos de árboles

- Preorden
- Inorden
- Postorden
- Por niveles

Todos los recorridos en un árbol son $\Theta(n)$ (suponiendo $\mathrm{O}(1)$ para visitar cada nodo)

Recorridos en Grafos Dirigidos

La resolución eficiente de muchos problemas relacionados con grafos dirigidos requiere realizar la visita sistemática de los vértices y/o arcos. Existen dos recorridos principales, que son base de muchos algoritmos:

- recorrido en profundidad
- recorrido en anchura

Recorrido en profundidad

Es una generalización del recorrido en preorden de un árbol.

```
typedef enum{NO_VISITADO, VISITADO} EstadoVertice;
EstadoVertice marca[n];

void RecorridoProfundidad()
{
  para (vertice v = 0; v < n; v++)
    marca[v] = NO_VISITADO;
  para (vertice v = 0; v < n; v++)
    si (marca[v] == NO_VISITADO)
      bpp(v);
}

void bpp (vertice v)
{
  marca[v] = VISITADO;
  para cada w adyacente a v
    si (marca[w] == NO_VISITADO)
      bpp(w);
}
```

La mejor representación es Lista de adyacencia. Eficiencia: $\mathrm{O}(|E|)$.

Recorrido en profundidad

El recorrido en profundidad produce una clasificación de los arcos del grafo en cuatro categorías:

1. *Arcos de árbol* (o de bosque). Son arcos que llevan a vértices aún no visitados. Su conjunto foma el *bosque de expansión en profundidad.*

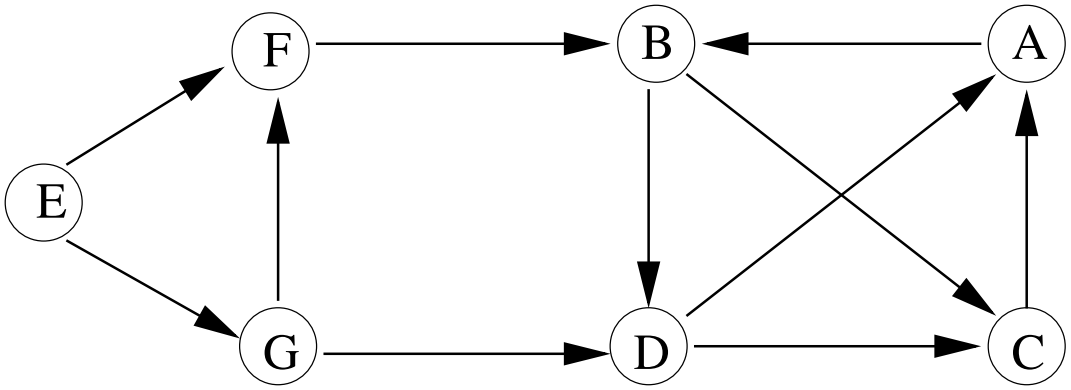
2. *Arcos de rectroceso*. Van desde un nodo a un antecesor en el bosque de expansión.

3. *Arcos de avance*. Van de un nodo a un descendiente en el bosque de expansión.

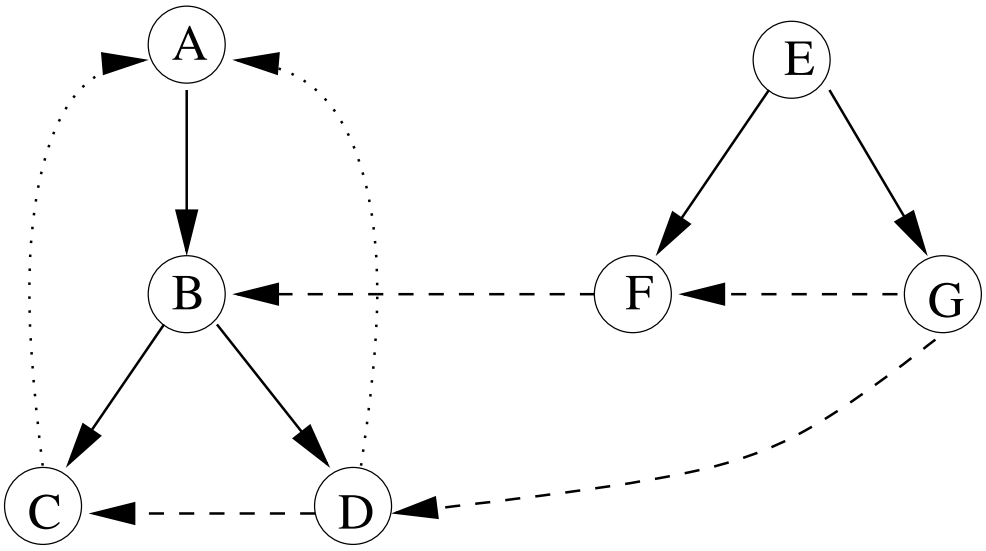
4. *Arcos cruzados*. Van desde un nodo a otro con el que no tienen relación de parentesco. Siempre van de derecha a izquierda.

El recorrido en profundidad de un grafo se puede registar en un vector `bpa`, tal que `bpa[v]` indica el orden en que se visita el nodo `v`.

Ejemplo de recorrido en profundidad



Llamada	Nodos visitados
`bpf(A)`	`A`
`  bpf(B)`	`A,B`
`    bpf(C)`	`A,B,C`
`    bpf(D)`	`A,B,C,D`
`bpf(E)`	`A,B,C,D,E`
`  bpf(F)`	`A,B,C,D,E,F`
`  bpf(G)`	`A,B,C,D,E,F,G`



Ordenación Topológica

- Dependencias temporales entre las tareas que componen un proyecto (grafos PERT).
- Dependencias entre las asignaturas de un plan de estudios.

Ordenación Topológica: ordenación lineal de los nodos de un grafo dirigido acíclico tal que si $(i, j) \in E$ entonces i precede a j. Se resuelve con una simple modificación del recorrido en profundidad del grafo G.

```
OrdenaciónTopológica()
{
  para (vertice v = 0; v < n; v++)
    marca[v] = NO_VISITADO;
  para (vertice v = 0; v < n; v++)
    si (marca[v] == NO_VISITADO)
      OrdTopol(v);
}
```

```
OrdTopol(vertice v)
{
  marca[v] = VISITADO;
  para cada w adyacente a v
    si (marca[w] == NO_VISITADO)
      OrdTopol(w);
  escribe(v);
}
```

Este algoritmo produce una ordenación topológica inversa. Para obtener una ordenación en sentido adecuado basta usar una pila.

Recorrido en Anchura

- Visita un nodo, después intenta visitar un vecino de éste, luego un vecino de este vecino y así, sucesivamente.

- Se usa principalmente para explorar parcialmente grafos infinitos o para encontrar el camino más corto entre dos puntos.

```
void RecorridoAnchura()
{
  para (vertice v = 0; v < n; v++)
    marca[v] = NO_VISITADO;
  para (vertice v = 0; v < n; v++)
    si (marca[v] == NO_VISITADO)
      bpa(v);
}
```

```
void bpa (vertice v)
{
  Cola<vertice> c;
  marca[v] = VISITADO;
  c.poner(v);
  mientras (!c.vacia())
    u = c.cabecera();
    c.borrar();
    para cada vértice w adyacente a u
      si (marca[v] == NO_VISITADO)
        marca[w] = VISITADO;
        c.insertar(w);
}
```

Backtracking

Backtracking: Método de búsqueda exhaustiva sobre grafos dirigidos acíclicos (y árboles), que es acelerado mediante poda.

Método de resolución general de problemas. Requisitos: representación de soluciones mediante vectores de dimensión n $(x_1, x_2, \ldots, x_n)$, con $x_i \in S_i$ (finito), y función objetivo $P(x_1, x_2, \ldots, x_n)$ a maximizar (minimizar o satisfacer).

Unas veces se busca una solución y otras, *todas* las soluciones.

Asociadas al problema existen **restricciones**:

- *Explícitas*: restringen los valores que pueden tomar las componentes x_i.

  Dominio de soluciones: compuesto por todos los vectores que verifican las restricciones explícitas.

- *Implícitas*: indican qué soluciones del dominio de soluciones verifican la función objetivo. (Relaciones entre las x_i).

El método backtracking resuelve el problema realizando una búsqueda en el dominio de soluciones. La búsqueda se facilita organizándolo en forma de GDA o árbol.

La búsqueda avanza determinando una nueva componente del vector solución en cada paso. Usa funciones objetivo modificadas $P(x_1, \ldots, x_i)$ para determinar si el nodo es *prometedor*. En caso contrario, se pueden ignorar todos sus descendientes.

Terminología

- *Estado*: nodo del árbol del dominio de las soluciones.
- *Espacio de estados*: conjunto de caminos desde la raíz a otros nodos.
- *Estado solución*: Estados del problema para el que el camino desde la raíz hasta ese nodo es un vector del dominio de las soluciones.
- *Estado respuesta*: Estado solución para el que el camino de la raíz hasta ese nodo es un vector del conjunto de soluciones (verifica las restricciones implícitas).
- *Arbol espacio de estados*: el árbol en que se organiza el espacio de estados.
  - *árbol estático*: No depende de la instancia concreta del problema.
  - *árbol dinámico*: Sí depende de la instancia concreta del problema.

Exploración en el *Backtracking*

A partir del árbol de espacio de estados, el problema se resuelve generando sistemáticamente todos los nodos del árbol, buscando los que corresponden a nodos respuesta.

El recorrido comienza en la raíz y sigue una filosofía de recorrido en profundidad.

En cada momento, el nodo en exploración actual v es evaluado mediante una *función de poda*. Si la función de poda es falsa, se detiene la exploración del nodo actual y la de *cualquier* descendiente. Se continúa por el nodo antecesor inmediato a v, w generando posibles nodos adyacentes a v. Si no los hubiera se retrocede al antecesor inmediato de w. Y así hasta finalizar el recorrido de todo el árbol.

Esquema genérico de *Backtracking*

Procedimiento *Backtrack*(X, n)
X: soluciones $X(1:n)$, que se escriben en cuanto se encuentran. $T(X(1),\ldots,X(k-1))$: valores para $X(k)$. $B_k(X(1),\ldots,X(k))$: cierto si $X(1:k)$ cumple las restricciones implícitas

$k \leftarrow 1$
mientras $k > 0$ **hacer**
  si hay valores no probados de $X(k)$ tales que
    $X(k) \in T(X(1),\ldots T(k-1))$ y
    $B_k(X(1),\ldots,X(k))$ es cierto
  entonces
    si $(X(1),\ldots,X(k))$ es un camino a un nodo respuesta
    entonces escribe la solución $(X(1),\ldots,X(k))$
    finsi
    $k \leftarrow k+1$
  en otro caso
    $k \leftarrow k-1$
  finsi
finmientras

Esquema recursivo de *Backtracking*

Procedimiento *RBacktrack*(k)

para cada valor de $X(k)$ tal que
$X(k) \in T(X(1), \ldots T(k-1))$ y
$B_k(X(1), \ldots, X(k))$ es cierto **hacer**
si $(X(1), \ldots, X(k))$ es un camino a un nodo respuesta
entonces escribe la solución $(X(1), \ldots, X(k))$
finsi
RBacktrack($k+1$)

Resolución de un problema mediante *Backtr*

1. Buscar una representación en forma de n-uplas, con componentes finitos.

2. Identificar las restricciones explícitas e implícitas.

3. Establecer la organización del árbol espacio de estados.

4. Definir la función de poda.

5. Aplicar la estructura genérica Backtracking.