“Hashing” externo
Exemplo - resultados

```
progs$ hashe
Word (end to finish)? jupiter
Word (end to finish)? urano
Word (end to finish)? saturno
Word (end to finish)? lua
Word (end to finish)? end
  *---*
  | \ |
   ---
  |  ----> saturno \
  *---*
  | \ |
   ---
  |  ----> urano   \
  *---*
  |  ----> jupiter -->   lua \
  *---*
```


Algoritmos em grafos

Alguns conceitos que devem estar presentes...

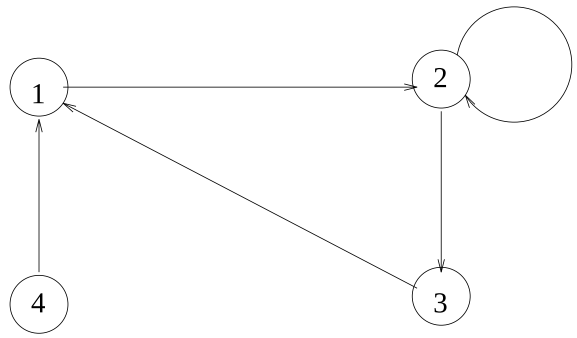
- Grafo dirigido: $G = (V, E)$, V é um conjunto (normalmente finito) de vértices, $E \subseteq V \times V$
- Grafo não dirigido.
- Subgrafo.
- Árvore.
- Sucessores (imediatos), descendentes.
- Caminhos e ciclos. Caminhos e ciclos simples.

Aplicações de grafos

- Representação de estruturas como termos, etc.
- Análise estática de programas; por exemplo, num programa em "assembly", as instruções posem ser os vértices existindo um ramo (I, J) se, a seguir à instrução I pode ser executada a instrução J.
- Problemas diversos de investigação operacional.
- Representação de redes de Comp., redes de electricidade, etc.
- Representação de sistemas de equações.
- etc.

Representações de grafos

$$\{\{1, 2, 3, 4\}, \{(1, 2), (2, 2), (2, 3), (3, 1), (4, 1)\}\}$$



$n = 4 \;\; g =$

	1	2	3	4
1	0	1	0	0
2	0	1	1	0
3	1	0	0	0
4	1	0	0	0

A matriz de adjacências do grafo
Admitimos $V = \{1, 2, \cdots, n\}$

Representações de grafos

$$n = 4 \;\; lr = \boxed{\begin{array}{c}(1,2)\\(2,2)\\(2,3)\\(3,1)\\(4,1)\end{array}}$$

Lista de ramos do grafo

$$n = 4 \;\; ls = \boxed{\begin{array}{l}1: \; 2\\2: \; 2,3\\3: \; 1\\4: \; 1\end{array}}$$

Lista de sucessores do grafo

Exercício 52 *Invente uma caracterização diferente das indicadas para os grafos dirigidos. Discuta as suas vantagens e inconvenientes.* $\diamond$

Representações de grafos em linguagem C

Pode ser algo como

– Matriz de incidências

```
struct grafo{
  int n;
  boolean m[VMAX][VMAX];
} g;
```

– Lista de ramos

```
struct grafo{
  int n;
  struct ramo{
    int origem, destino;
  } ramos[EMAX];
} g;
```

Grafos em linguagem C - continuação

– Lista de sucessores

```
struct grafo{
  int n;
  struct suc * lista[VMAX];
} g;
struct suc{
  int vertice;
  struct suc *seguinte;
};
```

onde

- "VMAX" é o número máximo de vértices.
- "EMAX" é o número máximo de ramos.
- "boolean" = "char" (via "typedef").

Eficiências das representações de grafos

Eficiências das representações de grafos

Qual a melhor representação? Depende do problema, do espaço de memória disponível...

Seja $n = |V|$, $m = |E|$.

	Mat. Inc.	**L. ramos**	**L. sucessores**
Espaço	$\approx n^2$ bits	$\approx 2m$ ints	$\approx n + 2m$ ints
$(a, b) \in E$?	$O(1)$	$O(\log m)$	$O(m)$
Primeiro suc.	$O(n)$	$O(\log m)$	$O(1)$
Próximo suc.	$O(n)$	$O(1)$	$O(1)$

Supõe-se que a lista de ramos está ordenada.

Outras questões poderiam ser consideradas; por exemplo, a facilidade de "editing" do grafo,

Visita de grafos Listagem dos nós por uma certa ordem.

Visita em profundidade
(recursiva) Marca todos os nós como não visitados.

Função $vp(n)$:

1. Visita n (seja o que isso for; por exemplo, lista n). Marca o nó n como visitado.

2. Para cada sucessor imediato s de n
   Se s ainda não foi visitado
   chama recursivamente $vp(s)$

Note-se que a visita é efectuada como se os descendentes fossem tratados como uma pilha (LIFO).

Exemplo do grafo dado

$$\begin{array}{llll} vp(1): & 1,2,3 & vp(2): & 2,3,1 \\ vp(3): & 3,1,2 & vp(4): & 4,1,2,3 \end{array}$$

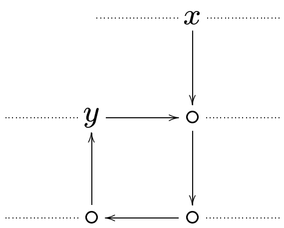
Visita em largura
Por níveis, sucessores tratados como uma fila...

Exemplo do grafo dado com o ramo $(4,3)$ **adicionado**

$$vl(4):\ 4,1,3,2$$

Aplicações da visita em profundidade
O grafo é cíclico?

Teorema O grafo $G = (V, E)$ é cíclico (isto é, tem pelo menos um ciclo) sse existe pelo menos um vértice x tal que na visita $vp(x)$ em profundidade se encontra algum vértice y com algum sucessor (já visitado) que está no caminho actual entre x e y.



Algoritmo Ver páginas seguintes. Note-se que é necessário (ou pode ser) chamar $vp(x)$ para todos os $x \in V$.

Exemplo do grafo dado É cíclico porque, por exemplo em $vp(4)$ temos

$$1, 2, \to 2(2 \text{ já visitado}) \cdots$$

Eficiência $O(\max(|V|+|E|)) = O(|V|+|E|)$ se usarmos a lista de sucessores como representação do grafo.

Aplicações da visita em profundidade
O grafo é cíclico? - programa I

```
/* O grafo tem ciclos? Se sim imprima um deles
   O grafo: n, g[1..n] onde g[i] e' a lista dos
   no's "sucessores" do no' i (lista de sucessores). */
#define NMAX 50           /* Maximo numero de no's   */
#define NAO  0
#define SIM  1
typedef char boolean;

int  sp,
     stack[NMAX];          /* para guardar os no's    */
char marca[NMAX];
/* marca[x]==SIM sse a chamada ciclo(x) existe
"no stack": ciclo(-) --> ... --> ciclo(x) --> ...
  (redundante...)
  NOTE-SE que esta utilização de marca[] não é
  idêntica à efectuada na pesquisa em profundidade
  (onde se marcavam simplesmente os nós já visitados) */
int  n;                    /*  numero de no's         */
struct no{                 /*  lista de sucessores    */
    int numero;
    struct no *seguinte;
    } *(g[NMAX]);

boolean ciclo(int x);
void constroi_grafo(void);
void display_ciclo(void);
```

Aplicações da visita em profundidade
O grafo é cíclico? - programa II

```
main(){
   int i,k;
   constroi_grafo();
   for(i=1;i<=n;i++){
     for(k=1;k<=n;k++) marca[k]=NAO;
     sp=0;
     if(ciclo(i)){ display_ciclo(); exit(0);  }
   }
   printf("Nao ha' ciclos.\n");
}

boolean ciclo(int x){
   struct no *t;
   int num;
   stack[sp++]=x;
   if(marca[x])     return(SIM);
   marca[x]=SIM;
   t=g[x];
   while(t!=NULL){
     num=t->numero;
     if(ciclo(num)) return(SIM);
     else t=t->seguinte; }
   sp--;
   marca[x]=NAO;
   return(0);
}
```


Aplicações da visita em profundidade
O grafo é cíclico? - programa III

```
void display_ciclo(void)
/* por ordem inversa... */
{ int i=sp-1;
  do{printf("%3d\n",stack[i]); i--;}
  while(stack[i]!=stack[sp-1]);
printf("\n");
}

void next(int a, int b);
void constroi_grafo(void){
  int i;
  n=6;
  for(i=1;i<=n;i++) g[i]=NULL;
  next(1,5); next(1,2); next(2,3);
  next(4,6); next(4,3);
  next(5,6); next(5,3); next(5,2);
  next(6,5); next(6,4);
}

void next(int a, int b){
  struct no *t;
  t=g[a];
  g[a]=(struct no *) malloc(sizeof(struct no));
  g[a]->numero  =b;
  g[a]->seguinte=t;
}
```

Aplicações da visita em profundidade
Existe caminho entre 2 vértices?

Teorema No grafo $G = (V, E)$ existe caminho entre os vértices x e y sse y for encontrado na visita em profundidade a partir de x.

Exercício 53 *Escrever a função booleana $cam(x, y)$ que diz se existe caminho entre os vértices x e y.* $\diamond$

Exercício 54 *Adaptar a função do exercício anterior por forma a indicar um caminho entre x e y caso exista.* $\diamond$

Exercício 55 *Escrever um programa que determina o fecho reflexivo e transitivo da relação binária $E \subseteq V \times V$. Note-se que $(x, y) \in E^\star$ sse existe em G um caminho de x para y. Analise a eficiência do programa que escreveu.* $\diamond$

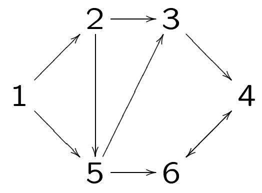
Eficiência $O(\min(|V|, |E|))$ se usarmos a lista de sucessores como representação do grafo. Isto porque cada ramo só pode ser "utilizado" uma vez.

Aplicações da visita em profundidade
Caminhos simples entre 2 vértices

Caminhos simples entre x e y = todos os modos de, partindo de x em pesquisa em profundidade, chegar a y. Note-se em particular que todos os caminhos obtidos por pesquisa em profundidade serem simples.

Exercício 56 *Alterar o programa de modo da imprimir o caminho mais curto entre os vértices dados. Supõe-se que a matriz das incidências é agora uma matriz das distâncias (positivas); se não existe ramo, a distancia é infinita (em linguagem C um valor muito grande). Usar um esquema de "branch and bound": se a distância parcial já é maior que a menor das distâncias já encontradas, os caminhos a partir daí são ignorados.* $\diamond$

Exemplo para o programa apresentado em seguida



Aplicações da visita em profundidade
Caminhos entre 2 vértices - programa I

```
/* Imprime todos os caminhos simples                */
#define NMAX 50      /* Maximo numero de no's       */

int origem,          /*  no' de partida             */
    destino,         /*  no' de chegada             */
    passou[NMAX],    /*  se ja' passou pelo no'     */
    caminho[NMAX],   /*  da origem para o destino   */
    c=0;             /*  num. de no's no caminho[]  */
int n=6,             /*  numero de no's             */
char g[NMAX][NMAX]={ /*  matriz das incidencias     */
    {0, 0,0,0,0,0,0},/* linha nao usada, no's 1..6  */
    {0, 0,1,0,0,1,0},/* coluna 0 nao usada          */
    {0, 0,0,1,0,1,0},
    {0, 0,0,0,1,0,0},
    {0, 0,0,0,0,0,1},
    {0, 0,0,1,0,0,1},
    {0, 0,0,0,1,0,0}};
```


Aplicações da visita em profundidade
Caminhos entre 2 vértices - programa I

```
void progride(int x);
void imprime(void);
main(){
   int i;
   printf("No' de partida? ");
   scanf("%d",&origem);
   printf("No' de chegada? ");
   scanf("%d",&destino);
   for(i=1;i<=n;i++) passou[i]=0;
   progride(origem);
}

void progride(int x){
   int i, ns;
   passou[x]=1;
   caminho[++c]=x;
   if(x==destino)
      imprime();
   else{
      for(i=1;i<=n;i++)
      if(g[x][i] && !passou[i])
        progride(i);
     }
   passou[x]=0;
   c--;
}
```

Aplicações da visita em profundidade
Caminhos entre 2 vértices - programa III

```
void imprime(void)
{ int i;
  for(i=1;i<=c;i++)
        printf("%d  ",caminho[i]);
  printf("\n");
}
```

Correndo o programa...

```
$ caminhos
No' de partida? 1
No' de chegada? 4
1  2  3  4
1  2  5  3  4
1  2  5  6  4
1  5  3  4
1  5  6  4
```


Grafos - Aplicações da pesquisa em profundidade

Algumas aplicações...

- Ciclos
- Existência de caminhos
- Fecho reflexivo e transitivo de relações binárias
- Distâncias mínimas
- O grafo é conexo?
- Componentes conexos.

Problema Dado um grafo com distâncias nos ramos, $d: E \to \mathbf{N}$, determinar a distancia mínima *entre todos os pares de vértices.*

Já vimos um algoritmo que poderia ser adaptado a este fim: a pesquisa em profundidade. A sua eficiência é

$$O(n^2(n+m)) = O(n^4)$$

onde $n = |V|$, $m = |E|$.

Vamos ver um algoritmo $O(n^3)$, o *algoritmo de Floyd*.

Princípio Para determinar as distâncias mínimas entre todos os pares $i, j \in V$ faz-se um ciclo no máximo valor k do vértice permitido como vértice intermédio dos caminhos entre i e j.

```
------------------------------------------------
DADOS: distâncias directas d[i][j] entre vértices
       1 <= i,j <= n
RESULTADO: todas as distâncias mínimas, dm[i][j]
------------------------------------------------
/* inicialmente distâncias directas  */
for i=1 to n
   for j=1 to n
      dm[i][j] = d[i][j];
/* k: max. vert. intermedio permitido */
for k=1 to n
   for i=1 to n
      for j=1 to n
         dm[i][j]=min(dm[i][j],dm[i][k]+dm[k][j]);
```

Nota Ficava mais claro mas mais longo se usassemos um temporário `t[ ][ ]` cujos valores no final de cada iteração fossem colocados em `dm[ ][ ]`.

Problema Dado um grafo com distâncias nos ramos, $d: E \to \mathbf{N}$, determinar para cada par de vértices se existe caminho entre eles.

A pesquisa em profundidade permite resolver este problema em tempo $O(n^2(n+m)) = O(n^4)$ onde $n = |V|$, $m = |E|$. Uma variação do algoritmo de Floyd tem eficiência $O(n^3)$.

Corresponde a uma álgebra diferente mas é semelhante...

Problema	**Caminho**	**Distância**
Dados	E	Distâncias dos ramos
Problema	Fecho $E^\star$	Distâncias mínimas
Operador 1	"$\vee$"	min
Operador 2	"$\wedge$"	"$+$"
Expressão	$A \vee (B \wedge C)$	min$(A, (B+C))$

Grafos com distâncias nos ramos
Algoritmo de Floyd - fecho transitivo de E

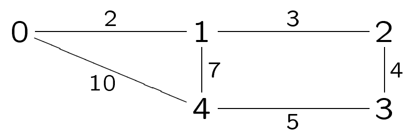
O algoritmo fica

```
DADOS: matriz das incidências m[i][j]
       1 <= i,j <= n
RESULTADO: fecho reflexivo e transitivo mt[i][j]

for i=1 to n     /* inicialmente mt = ramos     */
   for j=1 to n
      mt[i][j] = m[i][j] || (i==j);
for k=1 to n     /* k: max. v. interm. permitido*/
   for i=1 to n
      for j=1 to n
         mt[i][j]=dm[i][j] || (mt[i][k]&&mt[k][j]);
```

Caminhos correspondentes Podem ser obtidos usando uma matriz de vértices intermédios, "`intermedio[ ][ ]` no programa seguinte.

Exemplo do grafo no programa seguinte



Algoritmo de Floyd - Programa I

```
/*--------------------------------------------------
 Algoritmo de Floyd para determinar a distancia minima
 e correspondente caminho  em G=(V,E) entre todos os
 pares de vertices de um grafo.
----------------------------------------------------*/
#define INF    1.0e30
#define NENHUM (-1)
#define N      5
/* Matriz de adjacencias do grafo */
float f[N][N] = {{0,  2,  INF,INF,10 },
                 {2  ,0,  3,  INF,7  },
                 {INF,3  ,0,  4,  INF},
                 {INF,INF,4  ,0,  5  },
                 {10 ,7  ,INF,5  ,0  } };
float d[N][N];   /* Distancias minimas */
int intermedio[N][N];
/* intermedio[i][j] e' um vertice intermedio de um
caminho optimo entre i e j =0 se for directo          */
void resultados(void);
/*--------------------------------------------------*/
```


Algoritmo de Floyd - Programa II

```
main()
{ int i,j,k;

  /* todos os caminhos directos no inicio */
  for(i=0;i<N;i++)
    for(j=0;j<N;j++){
      d[i][j]=f[i][j];
      intermedio[i][j]=NENHUM;
    }
  /* permitindo a vertice intermedio k... */
  for(k=0;k<N;k++)
    for(i=0;i<N;i++)
      for(j=0;j<N;j++){
        float dn=d[i][k]+d[k][j];
        if(dn<d[i][j]){
          d[i][j]=dn;
          intermedio[i][j]=k;
        }
      }
  resultados();
}
```

Algoritmo de Floyd - Programa III

```
void caminho(int,int);

void resultados(void)
{
  int i,j;
  for(i=0;i<N;i++)
    for(j=0;j<N;j++) {
      printf("De %2d a %2d distancia minima= %3f\n",
             i,j,d[i][j]);
      printf("Caminho\n");
      caminho(i,j);
      printf("\n");
    }
}

void caminho(int i,int j)
{
  if(intermedio[i][j]==NENHUM){
    printf(" %d->%d",i,j);
    return;
  }
  caminho(i,intermedio[i][j]);
  caminho(intermedio[i][j],j);
```


Algoritmo de Floyd - Programa IV

```
---------------------------------------------

                        RESULTADOS

De  0 a  0 distancia minima= 0.000000
Caminho  0->0

De  0 a  1 distancia minima= 2.000000
Caminho  0->1

De  0 a  2 distancia minima= 5.000000
Caminho  0->1 1->2

De  0 a  3 distancia minima= 9.000000
Caminho  0->1 1->2 2->3

   ...etc...
```

Algoritmo de Dijkstra

Trata-se de um algoritmo que determina as distâncias mínimas entre um vértice dado e todos os outros. Não o explicaremos neste curso.

Resumo da eficiência dos algoritmos de determinação de distâncias mínimas.
Onde $n = |V|$, $m = |E|$.

Algoritmo	**Problema**	**N. result.**	**Eficiência**
P. profund.	x a y	1	$O(\text{min}(n, m))$
Dijkstra	x a todos	n	$O(n^2)$
Floyd	todos a todos	$n(n-1)/2$	$O(n^3)$

Árvore mínima de suporte

Definição Árvore de suporte de um grafo não dirigido é um subgrafo que é uma árvore e que inclui todos os vértices (só existe se o grafo for conexo).

Problema DADOS: Um grafo não dirigido $G = (V, E)$ com custos (=distâncias) associados aos ramos.
PRETENDE-SE: Uma árvore de suporte $A = (V, E')$ de G cujo custo (=soma dos custos dos ramos) seja mínimo.

Exemplo Ligar as povoações de uma certa região por estradas (ou telefones, etc.) de forma que o custo total seja mínimo.

Árvore mínima de suporte
Um algoritmo

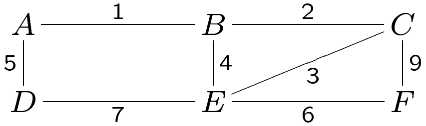
Algoritmo

```
DADOS: matriz dos custos c[i][j]
       definida para os ramos (i,j)
RESULTADO: Ramos EA da árvore mínima de suporte

EA = { };  /* inicialmente vazia              */
init(fp);  /* fp: fila de prioridade
              relativamente aos custos [ ][ ]*/
for all (i,j) in E
  fp=insere((i,j,c[i][j]), fp);
for k=1 to |E|
  (a,b,c_ab)=extrai_min(fp);
  if 'a' e 'b' em componentes conexos distintos
     EA=EA U {(a,b)}  /* uniao */
     custo = custo+c_ab;
```


Árvore mínima de suporte
Um algoritmo - exemplo

Grafo exemplo

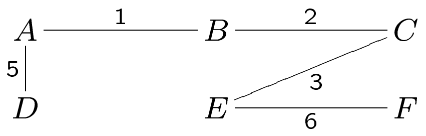


Evolução do algoritmo

Ramo escolhido	**Ext. conexas?**	EA
		$\{\}$
$(A,B)-1$	NÃO	$\{(A,B)\}$
$(B,C)-2$	NÃO	$\{(A,B),(B,C)\}$
$(C,E)-3$	NÃO	$\{(A,B),(B,C),$ $(C,E)\}$
$(B,E)-4$	SIM	o mesmo
$(A,D)-5$	NÃO	$\{(A,B),(B,C),$ $(C,E),(A,D)\}$
$(E,F)-6$	NÃO	$\{(A,B),(B,C),$ $(C,E),(A,D),$ $(E,F)\}$
$(D,E)-7$	SIM	o mesmo
$(C,F)-9$	SIM	o mesmo

Árvore mínima de suporte
Um algoritmo - exemplo

Grafo exemplo



Custo da (de uma) árvore mínima; $1{+}2{+}3{+}5{+}6 = 17$.

Árvore mínima de suporte
Algoritmo - porque funciona

Teorema O algoritmo está correcto.

Dem. Suponhamos por absurdo que não estava (a árvore não seria óptima) e seja (A, B) o primeiro ramo incluido que não pertence a uma árvore óptima.

A e B estão desligados entre si na árvore parcialmente construída. Seja A' uma árvore óptima que inclua a árvore já construída. A' tem que ter outro ramo (pelo menos) ainda não incluido pelo algoritmo, seja (X, Y) de custo maior ou igual ao de (A, B) e que liga (ou faz parte da ligação entre) vértices presentemente desligados entre si.

Se nessa árvore óptima substituirmos (X, Y) por (A, B) temos uma árvore com um custo total não inferior. Mas isso é uma contradição porque por hipótese (A, B) não pertence a nenhuma árvore óptima.