Algoritmos (André)

Sumário

Recursividade

8 rainhas

```
//INF492 : André Gustavo dos Santos

//Conta o número de maneiras de se colocar n rainhas
// num tabuleiro de xadrez n x n sem que nenhuma ataque a outra
//
//Usa o método de gerar todas as permutações, pois cada rainha
// estará numa linha e numa coluna diferente. Assim, uma permutação
// [2 4 1 3] indica que as rainhas nas colunas 1 2 3 4 estão
// respectivamente nas linhas 2 4 1 3, conforme ilustrado abaixo.
//
//  --R-
//  R---
//  ---R
//  -R--
//
// além de testar se um valor (no caso a linha) não apareceu na permutação,
// também testa se não estão na mesma diagonal (diferença abs de coordenadas)


#include <iostream>
using namespace std;

int cont = 0;

void imprimeperm(int a[], int n)
{
   int i;

   cout << "[ ";
   for(i=0;i<n;i++)
      cout << a[i] << " ";
   cout << "]\n";
}

bool posicaovalida(int a[], int k, int i)
{
   for(int j=0;j<k;j++) {
      if (a[j] == i)
         return false;
      if (abs(j-k) == abs(a[j]-i))
        return false;
   }
   return true;
}
```

```
void geraperm(int a[], int k, int n)
{
   if (k==n) {
      cont++;
      return;
   }

   for(int i=1;i<=n;i++)
      if (posicaovalida(a,k,i)) {
         a[k] = i;
         geraperm(a, k+1, n);
      }
}

int main()
{
   int a[100];
   int n;

   cin >> n;

   geraperm(a, 0, n);
   cout << cont << " solucoes para n = " << n << endl;

   return 0;
}
```

Labirinto

```
//INF492 : André Gustavo dos Santos

//Procura a saída de um labirinto, sendo '.' espaço livre e '#' as paredes
//
/*
Exemplo:
Tamanho do mapa: 7 8
########
#..#...#
##...#..
#.#.##.#
#....#.#
##.#..##
########
Posição inicial: 5 3
*/


#include <iostream>
#define MAX 100
using namespace std;
```

```
char mapa[MAX][MAX];
bool achei;

//imprime a situacao atual
void imprime(int nl, int nc)
{
   for(int i=0;i<nl;i++) {
      for(int j=0;j<nc;j++)
         cout << mapa[i][j];
      cout << endl;
   }
   cin.get();
}

//caminha no labirinto (por backtracking)
void caminha(int i, int j, int nl, int nc)
{
   if (achei)  //permite sair das chamadas recursivas empilhadas
      return;

   if (mapa[i][j]!='.')    //nao posso andar por aqui
      return;

   if (i==0 || j==0 || i==nl-1 || j==nc-1) {
      mapa[i][j] = 's';
      achei = true;   //consegui achar a saida
      return;
   }

   //se nao retornou ainda, e' porque posso continuar caminhando

   mapa[i][j] = 'o';   //marca que ja passei por aqui

   imprime(nl,nc);

   caminha(i+1,j,nl,nc);   //tenta caminhar para baixo
   caminha(i-1,j,nl,nc);   //tenta caminhar para cima
   caminha(i,j-1,nl,nc);   //tenta caminhar para esquerda
   caminha(i,j+1,nl,nc);   //tenta caminhar para direita

   if (!achei)
      mapa[i][j] = '.';  //desmarca posicao
}

int main()
{
   int i, j, nl, nc;
   int inil, inic;

   cout << "Tamanho do mapa: ";
   cin >> nl >> nc;
```

```
   cout << "Mapa:\n";
   for(i=0;i<nl;i++)
      for(j=0;j<nc;j++)
         cin >> mapa[i][j];

   cout << "Posicao inicial: ";
   cin >> inil >> inic;

   cin.get();

   inil--; //porque a matriz comeca de 0, assim 1 sera a primeira linha
   inic--;

   achei = false;
   caminha(inil,inic,nl,nc);

   if (!achei)
      cout << "Sem saida!\n";
   else {
      cout << "--Solucao:--\n\n";
      imprime(nl,nc);
   }

   return 0;
}
```

Regioes

```
//INF492 : André Gustavo dos Santos

//Semelhante ao do labrinto, mas caminha nos # marcando toda uma
//região cercada por "água", simbolizada por '.'
//
//Diferenças principais:
// - só para o backtracking depois de marcar todos espaçoes contíguos de uma região
// - a função principal marca e conta todas as regiões
/*
Exemplo:
Tamanho do mapa: 7 8
..#...#.
..#####.
#...#...
##......
###...#.
.....###
..#...#.
Número de regiões: 4
*/


#include <iostream>
#define MAX 102
```

```
using namespace std;

char mapa[MAX][MAX];

int contregioes;


//imprime a situacao atual
void imprime(int nl, int nc)
{
   for(int i=1;i<=nl;i++) {
      for(int j=1;j<=nc;j++)
         cout << mapa[i][j];
      cout << endl;
   }
   cin.get();
}

//caminha em uma regiao (por backtracking)
void caminha(int i, int j, int nl, int nc)
{
   if (mapa[i][j]!='#')    //nao posso andar por aqui
      return;

   //se nao retornou ainda, e' porque posso continuar caminhando

   mapa[i][j] = 'o';   //marca que ja passei por aqui

   imprime(nl,nc);

   caminha(i+1,j,nl,nc);   //tenta caminhar para baixo
   caminha(i-1,j,nl,nc);   //tenta caminhar para cima
   caminha(i,j-1,nl,nc);   //tenta caminhar para esquerda
   caminha(i,j+1,nl,nc);   //tenta caminhar para direita
}

int main()
{
   int i, j, nl, nc;
   int inil, inic;

   cout << "Tamanho do mapa: ";
   cin >> nl >> nc;

   //Cerca o mapa com "água", para não precisar testar limites do mapa no backtracking
   for(i=0;i<nl+2;i++)
      mapa[i][0] = mapa[i][nc+1] = '.';
   for(j=0;j<nc+2;j++)
      mapa[0][j] = mapa[nl+1][j] = '.';


   cout << "Mapa:\n";
```

```
   for(i=1;i<=nl;i++)
      for(j=1;j<=nc;j++)
         cin >> mapa[i][j];

   cin.get();

   contregioes = 0;
   for(i=1;i<=nl;i++)         //Procura região ainda não vista
      for(j=1;j<=nc;j++)
         if (mapa[i][j] == '#') {    //Se achou uma região ainda não vista
            contregioes++;
            cout << "Encontrada regiao " << contregioes << endl;
            caminha(i,j,nl,nc);     //Visita a região toda, por backtracking
            cout << "Regiao " << contregioes << " marcada\n";
         }

   cout << "\nNumero de regioes: " << contregioes << endl;

   system("pause");
   return 0;
}
```

Next_permutation 1

```
//INF492 : André Gustavo dos Santos

//Gera todas as permutações usando next_permutation da algorithm

//Tem a vantagem de não precisar implementar nada, gera automaticamente as permutações
//Tem a desvantagem de não ser possível usar cortes como no backtracking

#include <iostream>
#include <algorithm>

using namespace std;

void imprimeperm(int a[], int n)
{
   int i;

   cout << "[ ";
   for(i=0;i<n;i++)
      cout << a[i] << " ";
   cout << "]\n";
}

int main()
{
   int a[100];
   int n;

   cin >> n;
```

```
   for(int i=0;i<n;i++)
      a[i] = i+1;

   sort(a,a+n);    //A primeira permutação é o array ordenado
   //se não ordenar, ele gera as próximas a partir da atual

   do {
      imprimeperm(a,n);
   } while (next_permutation(a,a+n));  //next_permutation gera a próxima, lexicograficamente
   //quando estiver na última, next_permutation gera a primeira novamente, mas retorna false

   //existe também o prev_permutation, que gera a anterior, lexicograficamente

   return 0;
}
```

Next_permutation 2

```
//INF492: André Gustavo dos Santos

//Gera todos os anagramas (permutações de letras de uma palavra

#include <iostream>
#include <algorithm>
#include <cstring>

using namespace std;

int main()
{
   char a[100];
   int n;

   cin >> a;
   n = strlen(a);

   sort(a,a+n);    //A primeira permutação é o array ordenado
   //se não ordenar, ele gera as próximas a partir da atual

   do {
      cout << a << endl;
   } while (next_permutation(a,a+n));  //gera o próximo anagrama

   //tem a vantagem de já desconsiderar os repetidos, algo que
   //o backtrack-permutacoes não fazia

   return 0;
}
```

Permutações 1

```
//INF492 : André Gustavo dos Santos

//Gera todas as permutações de [1, 2, 3, ..., n]

#include <iostream>
using namespace std;

void imprimeperm(int a[], int n)
{
   int i;

   cout << "[ ";
   for(i=0;i<n;i++)
      cout << a[i] << " ";
   cout << "]\n";
}

bool pertence(int a[], int k, int x)
{
   for(int i=0;i<k;i++)
      if (a[i]==x)
         return true;
   return false;
}

void geraperm(int a[], int k, int n)
{
   if (k == n) {        // se terminou de gerar a permutação
      imprimeperm(a, n);
      return;
   }

   for(int i=1;i<=n;i++)        // para todo elemento i
      if (!pertence(a,k,i)) {   // se i não está na permutação parcial
         a[k] = i;            // inclui o elemento na permutação
         geraperm(a, k+1, n);  // gera o restante da permutação
      }
}

int main()
{
   int a[100];          // permutação parcial
   int n;               // tamanho do conjunto

   cin >> n;
   geraperm(a, 0, n);    // permutações de [1, 2, 3, ..., n]

   return 0;
}
```

Permutações 2

```
//INF492 : André Gustavo dos Santos

//Diferença do backtrack-permutacoes.cpp:
//
// Este não gera permutacoes de [1, 2, 3, ..., n],
// mas permite ler os valores dos elementos
//
// As diferenças estão marcadas no código

#include <iostream>
using namespace std;

int conj[100];  //DIFERENTE: mais uma variável, para guardar os elementos

void imprimeperm(int a[], int n)
{
   int i;

   cout << "[ ";
   for(i=0;i<n;i++)
      cout << conj[a[i]] << " "; //DIFERENTE: imprime conj[a[i]] em vez de a[i]
   cout << "]\n";
}

bool pertence(int a[], int k, int x)
{
   for(int i=0;i<k;i++)
      if (a[i]==x)
         return true;
   return false;
}

void geraperm(int a[], int k, int n)
{
   if (k == n) {        // se terminou de gerar a permutação
      imprimeperm(a, n);
      return;
   }

   for(int i=0;i<n;i++)         // DIFERENTE: para todo elemento i, mas a partir de 0
      if (!pertence(a,k,i)) {   // se i não está na permutação parcial
         a[k] = i;           // inclui o elemento na permutação
         geraperm(a, k+1, n);  // gera o restante da permutação
      }
}

int main()
{
   int a[100];          // permutação parcial
   int n;              // tamanho do conjunto
```

```
   cin >> n;

   for(int i=0;i<n;i++)    //DIFERENTE: le os elementos do conjunto
      cin >> conj[i];

   geraperm(a, 0, n);

   return 0;
}
```

Permutações 3

```
//INF492 : André Gustavo dos Santos

//Diferença do backtrack-permutacoes.cpp:
//
// Este não gera todas permutacoes de [1, 2, 3, ..., n],
// mas apenas aquelas em que 2 e 3 não aparecem seguidos
//
// As diferenças estão marcadas no código

#include <iostream>
using namespace std;

void imprimeperm(int a[], int n)
{
   int i;

   cout << "[ ";
   for(i=0;i<n;i++)
      cout << a[i] << " ";
   cout << "]\n";
}

bool pertence(int a[], int k, int x)
{
   for(int i=0;i<k;i++)
      if (a[i]==x)
         return true;
   return false;
}

bool seguidos(int a[], int n)
{
   for(int i=0;i<n-1;i++)
      if (a[i]==2 && a[i+1]==3)
         return true;
   return false;
}

void geraperm(int a[], int k, int n)
```

```
{
   if (k == n) {        // se terminou de gerar a permutação
      if (!seguidos(a, n))  // DIFERENTE: so imprime se 2 e 3 não estão seguidos
         imprimeperm(a, n);
      return;
   }

   for(int i=1;i<=n;i++)        // para todo elemento i
      if (!pertence(a,k,i)) {   // se i não está na permutação parcial
         a[k] = i;           // inclui o elemento na permutação
         geraperm(a, k+1, n);  // gera o restante da permutação
      }
}

int main()
{
   int a[100];          // permutação parcial
   int n;              // tamanho do conjunto

   cin >> n;
   geraperm(a, 0, n);    // permutações de [1, 2, 3, ..., n]

   return 0;
}
```

Permutações 4

```
//INF492 : André Gustavo dos Santos

//Diferença do backtrack-permutacoes-3.cpp:
//
// Também não gera todas permutacoes de [1, 2, 3, ..., n],
// somentes aquelas em que 2 e 3 não aparecem seguidos
//
// Mas corta o backtracking tão logo 2 e 3 apareçam seguidos
//
// As diferenças estão marcadas no código

#include <iostream>
using namespace std;

void imprimeperm(int a[], int n)
{
   int i;

   cout << "[ ";
   for(i=0;i<n;i++)
      cout << a[i] << " ";
   cout << "]\n";
}

bool pertence(int a[], int k, int x)
```

```
{
   for(int i=0;i<k;i++)
      if (a[i]==x)
         return true;
   return false;
}

bool seguidos(int a[], int n)
{
   for(int i=0;i<n-1;i++)
      if (a[i]==2 && a[i+1]==3)
         return true;
   return false;
}

void geraperm(int a[], int k, int n)
{
   if (seguidos(a,k))  //DIFERENTE: corta o backtracking assim que 2 e 3 aparecem seguidos
      return;
   // outra opção seria cortar na chamada da recursividade,
   // não chamar geraperm se k>0 && a[k-1]==2 && i==3

   if (k == n) {        // se terminou de gerar a permutação
      imprimeperm(a, n);
      return;
   }

   for(int i=1;i<=n;i++)        // para todo elemento i
      if (!pertence(a,k,i)) {   // se i não está na permutação parcial
         a[k] = i;           // inclui o elemento na permutação
         geraperm(a, k+1, n);  // gera o restante da permutação
      }
}

int main()
{
   int a[100];         // permutação parcial
   int n;              // tamanho do conjunto

   cin >> n;
   geraperm(a, 0, n);    // permutações de [1, 2, 3, ..., n]

   return 0;
}
```

Subconjunto 1

```
//INF492 : André Gustavo dos Santos

//Gera todos os subconjuntos de {1, 2, 3, ..., n}

using namespace std;
```

```
int c[100];
int lim;

void imprimesub(bool a[], int n)
{
   int i;

   cout << "{ ";
   for(i=0;i<n;i++)
      if (a[i])
         cout << i+1 << " ";
   cout << "}\n";
}

void gerasub(bool a[], int k, int n)
{
   if (k == n) {        // se terminou de gerar um subconjunto
      imprimesub(a, n);
      return;          // imprime e retorna
   }

   a[k] = true;         // subconjuntos com o elemento k
   gerasub(a, k+1, n);
   a[k] = false;        // subconjuntos sem o elemento k
   gerasub(a, k+1, n);
}

int main()
{
   bool a[100];        // subconjunto parcial
   int n;              // tamanho do conjunto

   cin >> n;
   gerasub(a, 0, n);    // subconjuntos de {1, 2, 3, ..., n}

   return 0;
}
```

Subconjuntos 2

```
//INF492 : André Gustavo dos Santos

//Diferença do backtrack-subconjuntos.cpp:
//
// Este não gera subconjuntos de {1, 2, 3, ..., n},
// mas permite ler os valores dos elementos do conjunto
//
// As diferenças estão marcadas no código

#include <iostream>
using namespace std;
```

```
int conj[100];  //DIFERENTE: mais uma variável, para guardar os elementos

void imprimesub(bool a[], int n)
{
   int i;

   cout << "{ ";
   for(i=0;i<n;i++)
      if (a[i])
         cout << conj[i] << " "; //DIFERENTE: imprime conj[i] em vez de i+1
   cout << "}\n";
}

void gerasub(bool a[], int k, int n)
{
   if (k == n) {        // se terminou de gerar um subconjunto
      imprimesub(a, n);
      return;          // imprime e retorna
   }

   a[k] = true;         // subconjuntos com o k-ésimo elemento
   gerasub(a, k+1, n);
   a[k] = false;        // subconjuntos sem o k-ésimo elemento
   gerasub(a, k+1, n);
}

int main()
{
   bool a[100];
   int n;

   cin >> n;
   for(int i=0;i<n;i++)    //DIFERENTE: le os elementos do conjunto
      cin >> conj[i];

   gerasub(a, 0, n);

   return 0;
}
```

Subconjuntos 3

```
//INF492 : André Gustavo dos Santos

//Diferença do backtrack-subconjuntos-2.cpp:
//
// Este não gera todos os subconjuntos,
// mas apenas aqueles cuja soma seja <= um limite
//
// As diferenças estão marcadas no código

#include <iostream>
```

```
using namespace std;

int conj[100];

void imprimesub(bool a[], int n)
{
   int i;

   cout << "{ ";
   for(i=0;i<n;i++)
      if (a[i])
         cout << conj[i] << " ";
   cout << "}\n";
}

int soma(bool a[], int n)
{
   int s = 0;
   for(int i=0;i<n;i++)
      if (a[i])
         s += conj[i];
   return s;
}

void gerasub(bool a[], int k, int n, int limite)
{
   if (k == n) {        // se terminou de gerar um subconjunto
      if (soma(a,n) <= limite)    //DIFERENTE: so imprime os que satisfazem
         imprimesub(a, n);
      return;          // imprime e retorna
   }

   a[k] = true;         // subconjuntos com o k-ésimo elemento
   gerasub(a, k+1, n, limite);
   a[k] = false;        // subconjuntos sem o k-ésimo elemento
   gerasub(a, k+1, n, limite);
}

int main()
{
   bool a[100];
   int n;

   cin >> n;
   for(int i=0;i<n;i++)
      cin >> conj[i];

   int lim;
   cin >> lim;     //DIFERENTE: le o limite da soma

   gerasub(a, 0, n, lim);  //DIFERENTE: passa o limite da soma
```

```
   return 0;
}
```

Subconjuntos 4

```
//INF492 : André Gustavo dos Santos

//Diferença do backtrack-subconjuntos-3.cpp:
//
// Também só imprime os subconjuntos cuja soma <= limite
//
// Este corta o backtracking assim que chega
// num subconjunto cuja soma seja > limite
// Não espera terminar de gerar o subconjunto
// para testar a soma, por isso é bem mais rápido
//
// As diferenças estão marcadas no código

#include <iostream>
using namespace std;

int conj[100];

void imprimesub(bool a[], int n)
{
   int i;

   cout << "{ ";
   for(i=0;i<n;i++)
      if (a[i])
         cout << conj[i] << " ";
   cout << "}\n";
}

int soma(bool a[], int n)
{
   int s = 0;
   for(int i=0;i<n;i++)
      if (a[i])
         s += conj[i];
   return s;
}

void gerasub(bool a[], int k, int n, int limite)
{
   if (soma(a,k) > limite) //DIFERENTE: corta o backtracking tão logo seja possível
      return;

   if (k == n) {        // se terminou de gerar um subconjunto
      imprimesub(a, n);
      return;          // imprime e retorna
   }
```

```
   a[k] = true;         // subconjuntos com o k-ésimo elemento
   gerasub(a, k+1, n, limite);
   a[k] = false;        // subconjuntos sem o k-ésimo elemento
   gerasub(a, k+1, n, limite);
}

int main()
{
   bool a[100];
   int n;

   cin >> n;
   for(int i=0;i<n;i++)
      cin >> conj[i];

   int lim;
   cin >> lim;

   gerasub(a, 0, n, lim);

   return 0;
}
```

Subconjuntos 5

```
//INF492 : André Gustavo dos Santos

//Diferença do backtrack-subconjuntos-4.cpp:
//
// Também só imprime os subconjuntos cuja soma <= limite
//
// Também corta o backtracking assim que chega
// num subconjunto cuja soma seja > limite
//
// Porém não soma os elementos novamente a cada chamada,
// a soma parcial é calculada à medida que os elementos
// são colocados no subconjunto e é passada na chamada
//
// As diferenças estão marcadas no código

#include <iostream>
using namespace std;

int conj[100];

void imprimesub(bool a[], int n)
{
   int i;

   cout << "{ ";
   for(i=0;i<n;i++)
```

```
      if (a[i])
         cout << conj[i] << " ";
   cout << "}\n";
}

//DIFERENTE: a soma parcial também é um parâmetro
void gerasub(bool a[], int k, int n, int limite, int soma)
{
   if (soma > limite) //corta o backtracking tão logo seja possível
      return;

   if (k == n) {        // se terminou de gerar um subconjunto
      imprimesub(a, n);
      return;          // imprime e retorna
   }

   a[k] = true;         // subconjuntos com o k-ésimo elemento
   gerasub(a, k+1, n, limite, soma+conj[k]);   //DIFERENTE: passa a soma parcial
   a[k] = false;        // subconjuntos sem o k-ésimo elemento
   gerasub(a, k+1, n, limite, soma);          //DIFERENTE: passa a soma parcial
}

int main()
{
   bool a[100];
   int n;

   cin >> n;
   for(int i=0;i<n;i++)
      cin >> conj[i];

   int lim;
   cin >> lim;

   gerasub(a, 0, n, lim, 0);  //DIFERENTE: a soma (inicialmente 0) é parâmetro

   return 0;
}
```

Subconjuntos debug

```
//INF492 : André Gustavo dos Santos

//Gera todos os subconjuntos de {1, 2, 3, ..., n}

#include <iostream>
using namespace std;
int c[100];
int lim;

void imprimesub(bool a[], int n)
{
```

```
   int i;

   cout << "{ ";
   for(i=0;i<n;i++)
      if (a[i])
         cout << i+1 << " ";
   cout << "}\n";
}

void gerasub(bool a[], int k, int n)
{
   if (k == n) {        // se terminou de gerar um subconjunto
      imprimesub(a, n);
      return;          // imprime e retorna
   }

   a[k] = true;         // subconjuntos com o elemento k
   cout << "Coloca elemento " << k+1 << endl;
   cin.get();
   gerasub(a, k+1, n);
   a[k] = false;        // subconjuntos sem o elemento k
   cout << "Retira elemento " << k+1 << endl;
   cin.get();
   gerasub(a, k+1, n);
}

int main()
{
   bool a[100];        // subconjunto parcial
   int n;              // tamanho do conjunto

   cin >> n;
   gerasub(a, 0, n);    // subconjuntos de {1, 2, 3, ..., n}

   return 0;
}
```

Grafos

Estrutura grafos

```
#include <iostream>
using namespace std;

#define MAXV    100
#define MAXGRAU  50

typedef struct grafo {
   int aresta[MAXV+1][MAXGRAU];    //vértices adjacentes a cada vértice
```

```
   int grau[MAXV+1];             //grau do vértice i
   int numvertices;              //número de vértices do grafo
};


void legrafo(grafo *g, bool direcionado);
void inicializagrafo(grafo *g);
void inserearesta(grafo *g, int x, int y);
void imprimegrafo(grafo *g);


void legrafo(grafo *g, bool direcionado)
{
   int i;
   int m;
   int x, y;

   inicializagrafo(g);

   cin >> g->numvertices >> m;
   for(i=0;i<m;i++) {
      cin >> x >> y;
      inserearesta(g,x,y);
      if (!direcionado)
         inserearesta(g,y,x);
   }
}

void inicializagrafo(grafo *g)
{
   int i;

   g->numvertices = 0;
   for(i=1; i<=MAXV; i++)
      g->grau[i] = 0;
}

void inserearesta(grafo *g, int x, int y)
{
   if (g->grau[x] >= MAXGRAU) {
      cout << "CUIDADO! insercao da aresta ("<< x << "," << y << ") excede grau maximo " <<
MAXGRAU << endl;
      //return;
   }

   g->aresta[x][g->grau[x]] = y;
   g->grau[x]++;
}

void imprimegrafo(grafo *g)
{
   int i,j;
```

```
   for(i=1; i<=g->numvertices; i++) {
      cout << i << ": ";
      for(j=0; j<g->grau[i]; j++)
         cout << g->aresta[i][j] << " ";
      cout << endl;
   }
}

int main()
{
   grafo g;

   legrafo(&g,false);
   imprimegrafo(&g);

   return 0;
}
```

Busca Largura

```
#include <iostream>
#include <queue>
using namespace std;

#define MAXV    100
#define MAXGRAU  50

typedef struct grafo {
   int aresta[MAXV+1][MAXGRAU];
   int grau[MAXV+1];
   int numvertices;
};

bool processado[MAXV];  //quais vertices ja foram processados
bool encontrado[MAXV];  //quais vertices ja foram encontrados

int pai[MAXV];         //qual o pai de cada um na arvore de busca

void buscalargura(grafo *g, int vi);
void processavertice(int v);
void processaaresta(int x, int y);
void legrafo(grafo *g, bool direcionado);
void inicializagrafo(grafo *g);
void inserearesta(grafo *g, int x, int y);
void imprimegrafo(grafo *g);
void inicializabusca(grafo *g);
void imprimecaminho(int v);
```

```
void buscalargura(grafo *g, int vi)
{
   queue <int> fila;   //fila de vertices a visitar
   int v;              //vertice atual
   int w;              //vertice vizinho
   int i;             //contador

   fila.push(vi);
   encontrado[vi] = true;

   while(!fila.empty()) {

      v = fila.front();
      fila.pop();
      processavertice(v);
      processado[v] = true;

      for(i=0; i<g->grau[v]; i++) {

         w = g->aresta[v][i];

         if (!encontrado[w]) {
            fila.push(w);
            encontrado[w] = true;
            pai[w] = v;
         }

         if (!processado[w])
            processaaresta(v,w);
      }
   }
}

void imprimecaminho(int v)
{
   if (pai[v]!=-1)
      imprimecaminho(pai[v]);
   cout << " " << v;
}


void inicializabusca(grafo *g)
{
   int i;

   for(i=1; i<=g->numvertices; i++) {
      processado[i] = false;
      encontrado[i] = false;
      pai[i] = -1;
   }
}
```

```
void processavertice(int v)
{
   cout << "Processado vertice " << v << endl;
}

void processaaresta(int x, int y)
{
   cout << "Processada aresta (" << x << "," << y << ")" << endl;
}


void legrafo(grafo *g, bool direcionado)
{
   int i;
   int m;
   int x, y;

   inicializagrafo(g);

   cin >> g->numvertices >> m;
   for(i=0;i<m;i++) {
      cin >> x >> y;
      inserearesta(g,x,y);
      if (!direcionado)
         inserearesta(g,y,x);
   }
}

void inicializagrafo(grafo *g)
{
   int i;

   g->numvertices = 0;
   for(i=1; i<=MAXV; i++)
      g->grau[i] = 0;
}

void inserearesta(grafo *g, int x, int y)
{
   if (g->grau[x] >= MAXGRAU)
      cout << "CUIDADO! insercao da aresta ("<< x << "," << y << ") excede grau maximo " <<
MAXGRAU << endl;

   g->aresta[x][g->grau[x]] = y;
   g->grau[x]++;
}

void imprimegrafo(grafo *g)
{
   int i,j;
```

```
   for(i=1; i<=g->numvertices; i++) {
      cout << i << ": ";
      for(j=0; j<g->grau[i]; j++)
         cout << g->aresta[i][j] << " ";
      cout << endl;
   }
}

int main()
{
   grafo g;
   legrafo(&g,false);
   imprimegrafo(&g);

   inicializabusca(&g);
   buscalargura(&g,1);

   return 0;
}
```

Busca Profundidade

```
#include <iostream>
#include <stack>
using namespace std;

#define MAXV    100
#define MAXGRAU  50

typedef struct grafo {
   int aresta[MAXV+1][MAXGRAU];
   int grau[MAXV+1];
   int numvertices;
};

bool processado[MAXV];  //quais vertices ja foram processados
bool encontrado[MAXV];  //quais vertices ja foram encontrados

int pai[MAXV];         //qual o pai de cada um na arvore de busca

void buscaprofundidade(grafo *g, int vi);
void processavertice(int v);
void processaaresta(int x, int y);
void legrafo(grafo *g, bool direcionado);
void inicializagrafo(grafo *g);
void inserearesta(grafo *g, int x, int y);
void imprimegrafo(grafo *g);
void inicializabusca(grafo *g);
```

```
void buscaprofundidade(grafo *g, int vi)
{
   stack <int> pilha;  //pilha de vertices a visitar
   int v;              //vertice atual
   int w;              //vertice vizinho
   int i;             //contador

   pilha.push(vi);
   encontrado[vi] = true;

   while(!pilha.empty()) {

      v = pilha.top();
      pilha.pop();
      processavertice(v);
      processado[v] = true;

      for(i=0; i<g->grau[v]; i++) {

         w = g->aresta[v][i];

         if (!encontrado[w]) {
            pilha.push(w);
            encontrado[w] = true;
            pai[w] = v;
         }

         if (!processado[w])
            processaaresta(v,w);
      }
   }
}

void inicializabusca(grafo *g)
{
   int i;

   for(i=1; i<=g->numvertices; i++) {
      processado[i] = false;
      encontrado[i] = false;
      pai[i] = -1;
   }
}


void processavertice(int v)
{
   cout << "Processado vertice " << v << endl;
}

void processaaresta(int x, int y)
{
```

```
   cout << "Processada aresta (" << x << "," << y << ")" << endl;
}

void legrafo(grafo *g, bool direcionado)
{
   int i;
   int m;
   int x, y;

   inicializagrafo(g);

   cin >> g->numvertices >> m;
   for(i=0;i<m;i++) {
      cin >> x >> y;
      inserearesta(g,x,y);
      if (!direcionado)
         inserearesta(g,y,x);
   }
}

void inicializagrafo(grafo *g)
{
   int i;

   g->numvertices = 0;
   for(i=1; i<=MAXV; i++)
      g->grau[i] = 0;
}


void inserearesta(grafo *g, int x, int y)
{
   if (g->grau[x] >= MAXGRAU)
      cout << "CUIDADO! insercao da aresta ("<< x << "," << y << ") excede grau maximo " <<
MAXGRAU << endl;

   g->aresta[x][g->grau[x]] = y;
   g->grau[x]++;
}

void imprimegrafo(grafo *g)
{
   int i,j;

   for(i=1; i<=g->numvertices; i++) {
      cout << i << ": ";
      for(j=0; j<g->grau[i]; j++)
         cout << g->aresta[i][j] << " ";
      cout << endl;
   }
}
```

```
int main()
{
   grafo g;
   legrafo(&g,false);
   imprimegrafo(&g);

   inicializabusca(&g);
   buscaprofundidade(&g,1);

   return 0;
}
```

Busca Profundidade backtracking

```
#include <iostream>
using namespace std;

#define MAXV    100
#define MAXGRAU  50

typedef struct grafo {
   int aresta[MAXV+1][MAXGRAU];
   int grau[MAXV+1];
   int numvertices;
};

bool processado[MAXV];  //quais vertices ja foram processados
bool encontrado[MAXV];  //quais vertices ja foram encontrados

int pai[MAXV];         //qual o pai de cada um na arvore de busca

void buscaprofundidade(grafo *g, int vi);
void processavertice(int v);
void processaaresta(int x, int y);
void legrafo(grafo *g, bool direcionado);
void inicializagrafo(grafo *g);
void inserearesta(grafo *g, int x, int y);
void imprimegrafo(grafo *g);
void inicializabusca(grafo *g);



void buscaprofundidade(grafo *g, int v)
{
   int w;            //vertice vizinho
   int i;           //contador

   encontrado[v] = true;
   processavertice(v);

   for(i=0; i<g->grau[v]; i++) {
      w = g->aresta[v][i];
```

```
      if (!encontrado[w] && !processado[w])
         processaaresta(v,w);


      if (!encontrado[w]) {
         pai[w] = v;
         buscaprofundidade(g,w);
      }
   }
   processado[v] = true;
}

void inicializabusca(grafo *g)
{
   int i;

   for(i=1; i<=g->numvertices; i++) {
      processado[i] = false;
      encontrado[i] = false;
      pai[i] = -1;
   }
}


void processavertice(int v)
{
   cout << "Processado vertice " << v << endl;
}

void processaaresta(int x, int y)
{
   cout << "Processada aresta (" << x << "," << y << ")" << endl;
}

void legrafo(grafo *g, bool direcionado)
{
   int i;
   int m;
   int x, y;

   inicializagrafo(g);

   cin >> g->numvertices >> m;
   for(i=0;i<m;i++) {
      cin >> x >> y;
      inserearesta(g,x,y);
      if (!direcionado)
         inserearesta(g,y,x);
   }
}
```

```
void inicializagrafo(grafo *g)
{
   int i;

   g->numvertices = 0;
   for(i=1; i<=MAXV; i++)
      g->grau[i] = 0;
}

void inserearesta(grafo *g, int x, int y)
{
   if (g->grau[x] >= MAXGRAU)
      cout << "CUIDADO! insercao da aresta ("<< x << "," << y << ") excede grau maximo " <<
MAXGRAU << endl;

   g->aresta[x][g->grau[x]] = y;
   g->grau[x]++;
}

void imprimegrafo(grafo *g)
{
   int i,j;

   for(i=1; i<=g->numvertices; i++) {
      cout << i << ": ";
      for(j=0; j<g->grau[i]; j++)
         cout << g->aresta[i][j] << " ";
      cout << endl;
   }
}

int main()
{
   grafo g;
   legrafo(&g,false);
   imprimegrafo(&g);

   inicializabusca(&g);
   buscaprofundidade(&g,1);

   return 0;
}
```

Componentes conectados

```
#include <iostream>
#include <stack>
using namespace std;

#define MAXV    100
#define MAXGRAU  50
```

```
typedef struct grafo {
   int aresta[MAXV+1][MAXGRAU];
   int grau[MAXV+1];
   int numvertices;
};

bool processado[MAXV];  //quais vertices ja foram processados
bool encontrado[MAXV];  //quais vertices ja foram encontrados

int pai[MAXV];         //qual o pai de cada um na arvore de busca

void buscaprofundidade(grafo *g, int vi);
void processavertice(int v);
void processaaresta(int x, int y);
void legrafo(grafo *g, bool direcionado);
void inicializagrafo(grafo *g);
void inserearesta(grafo *g, int x, int y);
void imprimegrafo(grafo *g);
void inicializabusca(grafo *g);



void buscaprofundidade(grafo *g, int vi)
{
   stack <int> fila;   //fila de vertices a visitar
   int v;             //vertice atual
   int w;              //vertice vizinho
   int i;             //contador

   fila.push(vi);
   encontrado[vi] = true;

   while(!fila.empty()) {

      v = fila.top();
      fila.pop();
      processavertice(v);
      processado[v] = true;

      for(i=0; i<g->grau[v]; i++) {

         w = g->aresta[v][i];

         if (!encontrado[w]) {
            fila.push(w);
            encontrado[w] = true;
            pai[w] = v;
         }

         if (!processado[w])
            processaaresta(v,w);
      }
```

```
   }
}

void inicializabusca(grafo *g)
{
   int i;

   for(i=1; i<=g->numvertices; i++) {
      processado[i] = false;
      encontrado[i] = false;
      pai[i] = -1;
   }
}


void processavertice(int v)
{
//    cout << "Processado vertice " << v << endl;
   cout << " " << v;
}

void processaaresta(int x, int y)
{
//    cout << "Processada aresta (" << x << "," << y << ")" << endl;
}

void legrafo(grafo *g, bool direcionado)
{
   int i;
   int m;
   int x, y;

   inicializagrafo(g);

   cin >> g->numvertices >> m;
   for(i=0;i<m;i++) {
      cin >> x >> y;
      inserearesta(g,x,y);
      if (!direcionado)
         inserearesta(g,y,x);
   }
}

void inicializagrafo(grafo *g)
{
   int i;

   g->numvertices = 0;
   for(i=1; i<=MAXV; i++)
      g->grau[i] = 0;
}
```

```
void inserearesta(grafo *g, int x, int y)
{
   if (g->grau[x] >= MAXGRAU)
      cout << "CUIDADO! insercao da aresta ("<< x << "," << y << ") excede grau maximo " <<
MAXGRAU << endl;

   g->aresta[x][g->grau[x]] = y;
   g->grau[x]++;
}

void imprimegrafo(grafo *g)
{
   int i,j;

   for(i=1; i<=g->numvertices; i++) {
      cout << i << ": ";
      for(j=0; j<g->grau[i]; j++)
         cout << g->aresta[i][j] << " ";
      cout << endl;
   }
}

void encontracomponentes(grafo *g)
{
   int c;  //contador de componentes
   int i;

   inicializabusca(g);

   c = 0;
   for(i=1; i<=g->numvertices; i++)
      if (!encontrado[i]) {
         c++;
         cout << "Componente " << c << ":";
         buscaprofundidade(g,i);
         cout << endl;
      }
}

int main()
{
   grafo g;
   legrafo(&g,false);
   imprimegrafo(&g);

   encontracomponentes(&g);

   return 0;
}
```

Prim

```
//INF492 - Topicos Especiais: desafios de programacao
//
//Metodo de Prim para arvore geradora minima (com lista de adjacencia)
//  escolhe arestas para conectar todos os vertices com custo total minimo
//  formando uma arvore (grafo conectado sem ciclo)
//
//  Obs: substituindo "peso" por "peso + distancia[v]" das duas linhas
//       marcadas com ***** vira algoritmo de Dijkstra para caminho mais curto
//
//Andre Gustavo dos Santos (14/05/08)
//

#include <iostream>

#define INF 1000000000
#define MAXVERT 100
#define MAXGRAU  50

using namespace std;

typedef struct {
   int vertice, peso;
} tipoaresta;

typedef struct {
   tipoaresta adj[MAXVERT+1][MAXGRAU];    //adjacentes a cada vertice
   int grau[MAXVERT+1];                   //grau de cada vertice
   int nvert, narest;
} tipografo;

int main()
{
   tipografo g;
   int x,y,z;             //auxiliar para leitura de dados

   int inicial;           //vertice inicial
   bool marcado[MAXVERT+1];    //ja marcados pelo algoritmo
   int distancia[MAXVERT+1];   //distancia do vertice a arvore
   int pai[MAXVERT+1];         //vertice pai (a ligacao desse a arvore)
   int i,v,w,min,peso;         //contadores e auxiliares
   int disttotal;          //peso total da arvore

   cin >> g.nvert >> g.narest;

   for(i=1;i<=g.nvert;i++)
     g.grau[i] = 0;

   for(i=0;i<g.narest;i++) {   //le distancias entre vertices
     cin >> x >> y >> z;
     g.adj[x][g.grau[x]].vertice = y;  //inclui y na lista de adjacentes de x
```

```
    g.adj[x][g.grau[x]].peso = z;    //guarda o peso da aresta x->y
    g.grau[x]++;

    g.adj[y][g.grau[y]].vertice = x;  //inclui x na lista de adjacentes de y
    g.adj[y][g.grau[y]].peso = z;    //guarda o peso da aresta y->x
    g.grau[y]++;
   }

/*
   cout << "--Lista de adjacencia--\n";
   for(v=1;v<=g.nvert;v++) {
      cout << v << ": ";
      for(i=0;i<g.grau[v];i++)
        cout << g.adj[v][i].vertice << " (" << g.adj[v][i].peso << "), ";
      cout << endl;
   }
   cout << "------------------------\n";
*/

   inicial = 1;          //funciona com qualquer vertice inicial

   for(i=1;i<=g.nvert;i++) {   //inicializacoes
    distancia[i] = INF;
    marcado[i] = false;
    pai[i] = -1;
   }

   v = inicial;
   distancia[v] = 0;

   while(!marcado[v]) {    //enquanto tem vertice para marcar
    marcado[v] = true;
    for(i=0;i<g.grau[v];i++) { //atualiza melhor distancia dos vizinhos de v
      //se ligar w em v para conectar w na arvore
      //for melhor que o a melhor forma ja conhecida
      w =    g.adj[v][i].vertice;
      peso = g.adj[v][i].peso;
      if (!marcado[w] && distancia[w] > peso) {  //*****diferente do Dijkstra
         distancia[w] = peso;                //*****diferente do Dijkstra
         pai[w] = v;
      }
    }
    min = INF;
    v = 1;
    for(i=1;i<=g.nvert;i++)
      if (!marcado[i] && distancia[i]<min) {   //busca o mais proximo nao marcado
         v = i;
         min = distancia[i];
      }
   }

   for(i=1;i<=g.nvert;i++)    //imprime as arestas que fazem parte da arvore
```

```
    if (pai[i]!=-1)
      cout << "(" << i << "," << pai[i] << ")\n";

  disttotal = 0;
  for(i=1;i<=g.nvert;i++)
    disttotal += distancia[i];
  cout << "Distancia total = " << disttotal << endl;

  return 0;
}
```

Dijkstra

```
//INF492 - Topicos Especiais: desafios de programacao
//
//Metodo de Dijkstra para caminho mais curto (com lista de adjacencia)
//  encontra o menor caminho de um vertice a todos os demais
//
//  Obs: substituindo "peso + distancia[v]" por "peso" das duas linhas
//       marcadas com *** vira algoritmo de Prim para arvore geradora minima
//
//Andre Gustavo dos Santos (14/05/08)
//

#include <iostream>

#define INF 1000000000
#define MAXVERT 100
#define MAXGRAU  50

using namespace std;

typedef struct {
  int vertice, peso;
} tipoaresta;

typedef struct {
  tipoaresta adj[MAXVERT+1][MAXGRAU];    //adjacentes a cada vertice
  int grau[MAXVERT+1];                   //grau de cada vertice
  int nvert, narest;
} tipografo;

int main()
{
  tipografo g;
  int x,y,z;             //auxiliar para leitura de dados

  int inicial;           //vertice inicial do caminho
  bool marcado[MAXVERT+1];    //ja marcados pelo algoritmo
  int distancia[MAXVERT+1];   //distancia do caminho mais curto conhecido
  int pai[MAXVERT+1];         //vertice pai (o anterior no caminho mais curto)
```

```
  int i,v,w,min,peso;        //contadores e auxiliares

  cin >> g.nvert >> g.narest;

  for(i=1;i<=g.nvert;i++)
   g.grau[i] = 0;

  for(i=0;i<g.narest;i++) {   //le distancias entre vertices
   cin >> x >> y >> z;
   g.adj[x][g.grau[x]].vertice = y;  //inclui y na lista de adjacentes de x
   g.adj[x][g.grau[x]].peso = z;     //guarda o peso da aresta x->y
   g.grau[x]++;

   g.adj[y][g.grau[y]].vertice = x;  //inclui x na lista de adjacentes de y
   g.adj[y][g.grau[y]].peso = z;     //guarda o peso da aresta y->x
   g.grau[y]++;
  }

/*
  cout << "--Lista de adjacencia--\n";
  for(v=1;v<=g.nvert;v++) {
     cout << v << ": ";
     for(i=0;i<g.grau[v];i++)
       cout << g.adj[v][i].vertice << " (" << g.adj[v][i].peso << "), ";
     cout << endl;
  }
  cout << "------------------------\n";
*/

  cin >> inicial;          //le vertice inicial do caminho

  for(i=1;i<=g.nvert;i++) {   //inicializacoes
   distancia[i] = INF;
   marcado[i] = false;
   pai[i] = -1;
  }

  v = inicial;
  distancia[v] = 0;

  while(!marcado[v]) {    //enquanto tem vertice para marcar
   marcado[v] = true;
   for(i=0;i<g.grau[v];i++) { //atualiza melhor caminho dos vizinhos de v
     //se caminhar ate v, e depois seguir direto v->w,
     //for melhor que o melhor caminho ja conhecido ate w
     w =    g.adj[v][i].vertice;
     peso = g.adj[v][i].peso;
     if (!marcado[w] && distancia[w] > distancia[v] + peso) {    //*** diferente do Prim
        distancia[w] = distancia[v] + peso;                  //*** diferente do Prim
        pai[w] = v;
     }
   }
```

```
    min = INF;
    v = 1;
    for(i=1;i<=g.nvert;i++)
      if (!marcado[i] && distancia[i]<min) {   //busca o mais proximo nao marcado
         v = i;
         min = distancia[i];
      }
   }

   for(i=1;i<=g.nvert;i++) {     //imprime caminho mais curto (de tras pra frente)
    cout << i;
    v = pai[i];
    while(v!=-1) {
      cout << " <- " << v;
      v = pai[v];
    }
    cout << ": Distancia total = " << distancia[i] << endl;
   }

   return 0;
}
```

Dijkstra Matriz

```
//INF492 - Topicos Especiais: desafios de programacao
//
//Metodo de Dijkstra para caminho mais curto (com matriz de adjacencia)
//  encontra o menor caminho de um vertice a todos os demais
//
//Andre Gustavo dos Santos (14/05/08)
//
#include <iostream>
#define INF 1000000000
#define MAXV 100

using namespace std;

int main()
{
   int d[MAXV+1][MAXV+1];   //matriz com as distancias
   int nvert, narest;      //numero de vertices e arestas do grafo
   int x,y,z;              //auxiliar para leitura de dados

   int inicial;           //vertice inicial do caminho
   bool marcado[MAXV+1];    //ja marcados pelo algoritmo
   int distancia[MAXV+1];   //distancia do caminho mais curto conhecido
   int pai[MAXV+1];        //vertice pai (o anterior no caminho mais curto)
   int i,j,v,w,min;        //contadores e auxiliares

   cin >> nvert >> narest;

   for(i=1;i<=nvert;i++)
```

```
    for(j=1;j<=nvert;j++)
      d[i][j] = INF;       //Inicializa com "infinito"

  for(i=1;i<=nvert;i++)
    d[i][i] = 0;          //distancia do vertice a ele mesmo = 0

  for(i=0;i<narest;i++) {   //le distancias entre vertices
    cin >> x >> y >> z;
    d[x][y] = z;
    d[y][x] = z;          //caso seja grafo nao dirigido
  }
  cin >> inicial;         //le vertice inicial do caminho
  for(i=1;i<=nvert;i++) {   //inicializacoes
    distancia[i] = INF;
    marcado[i] = false;
    pai[i] = -1;
  }
  v = inicial;
  distancia[v] = 0;
  while(!marcado[v]) {    //enquanto tem vertice para marcar
    marcado[v] = true;
    for(w=1;w<=nvert;w++) //atualiza melhor caminho dos vizinhos de v
      //se caminhar ate v, e depois seguir direto v->w,
      //for melhor que o melhor caminho ja conhecido ate w
      if (!marcado[w] && distancia[w] > distancia[v] + d[v][w]) {
         distancia[w] = distancia[v] + d[v][w];
         pai[w] = v;
      }
    min = INF;
    v = 1;
    for(i=1;i<=nvert;i++)
      if (!marcado[i] && distancia[i]<min) {   //busca o mais proximo nao marcado
         v = i;
         min = distancia[i];
      }
  }

  for(i=1;i<=nvert;i++) {     //imprime caminho mais curto (de tras pra frente)
    cout << i;
    v = pai[i];
    while(v!=-1) {
      cout << " <- " << v;
      v = pai[v];
    }
    cout << ": Distancia total = " << distancia[i] << endl;
  }

  return 0;
}
```