Grafos - Aplicação

Uma estratégia para a solução é a seguinte: crie um grafo com as cidades como nós e as estradas como arcos. Para achar um caminho de comprimento *nr* do nó *A* ao nó *B,* procure um nó *C* de modo que exista um arco de *A* até *C* e um caminho de comprimento *nr - 1* de *C* até *B.* Se essas condições forem atendidas para um nó *C*, o caminho desejado existirá; se elas não forem atendidas para qualquer nó *C*, o caminho não existirá.

O algoritmo usará uma função recursiva auxiliar, *procurarCaminho(k,a,b)*. Essa função retornará *true* se existir um segmento de comprimento *k* de *A* até *B,* e *false*, caso contrário.

Grafos - Aplicação

Implemente na linguagem C a função procurarCaminho(...).

```
procurarCaminho(int k, int a, int b)
{
   int c;
   if (k == 1)
      return (adjacente (a, b));
   for (c = 0; c < n; ++c)
      if (adjacente(a,c) && procurarCaminho (k - 1,
c, b))
         return(1);
   return(0);
}
```

Grafos - Aplicação

Visando possibilitar o teste da função apresentada implemente, na linguagem C, uma função *main()*, que receba uma linha de entrada contendo quatro inteiros seguidos por um número qualquer de linhas de entrada com dois inteiros cada uma. O primeiro inteiro na primeira linha, *n,* representa um número de cidades, que, para simplificar, serão numeradas de 0 a *n* - 1. *O* segundo e o terceiro inteiro nessa linha estão entre 0 e *n* - 1 e representam duas cidades. Queremos sair da primeira cidade para a segunda usando exatamente *nr* estradas, onde *nr* é o quarto inteiro na primeira linha de entrada. Cada linha de entrada subsequente contém dois inteiros representando duas cidades, indicando que existe uma estrada da primeira cidade até a segunda. A última linha na sequência conterá dois valores inteiros negativos. O problema é determinar se existe um percurso do tamanho solicitado pelo qual se possa viajar da primeira cidade para a segunda.

```
int main()  {
   int **adj, a, b, nr, city1, city2, i, j;
   scanf ("%d", &n);
   adj = (int **) malloc (n*sizeof(int *));
   for (i=0; i<n; i++)
      adj[i] = (int *) malloc (n*sizeof(int));
   for (i=0; i< n; i++)
      for (j=0; j< n; j++)
         adj[i][j]=0;
   scanf ("%d %d", &a, &b) ;
   scanf ("%d", &nr);
   do {
      scanf("%d %d\n", &city1, &city2);
      if (city1>-1)
         ligar(city1,city2);
   } while(city1>-1);
   if (procurarCaminho(nr,a,b))
      printf("existe um caminho de %d ateh %d em %d passos", a, b, nr);
   else
      printf("nao existe caminho de %d ateh %d em %d passos", a, b, nr);
   return(0);
}
```

Grafos - Aplicação

Embora o algoritmo anterior seja uma solução para o problema, ele apresenta algumas deficiências. Identifique-as.

Vários caminhos são investigados diversas vezes durante o processo recursivo.

Além disso, embora o algoritmo precise realmente verificar todo caminho possível, o resultado final apenas confirma se existe o caminho desejado; ele não indica qual é o caminho. Seria preferível achar os arcos do caminho além de saber apenas se o caminho existe ou não.

Finalmente, o algoritmo não verifica a existência de um caminho independentemente do comprimento; ele só verifica há presença de um caminho de comprimento específico.

UNIVASF

Grafos - Representação

A representação em matriz de adjacência de um grafo é frequentemente inadequada:

- requer o conhecimento prévio do número de nós;
  - se for necessária a construção de um grafo no decorrer da solução de um problema?
  - se ele precisar ser atualizado dinamicamente durante a execução do programa?

Criar uma nova matriz a cada inclusão ou eliminação de um nó é proibitivamente ineficiente, em especial numa situação do mundo real em que um grafo pode ter uma centena de nós ou mais.

UNIVASF

Grafos - Representação

Além disso, mesmo que um grafo tenha muito poucos arcos tal que a matriz de adjacência (e a matriz ponderada para um grafo ponderado) seja esparsa, será necessário reservar espaço para todo possível arco entre dois nós, quer este arco exista, quer não. Se o grafo contiver **n** nós, precisará ser usado um total de $\mathbf{n^2}$ alocações.

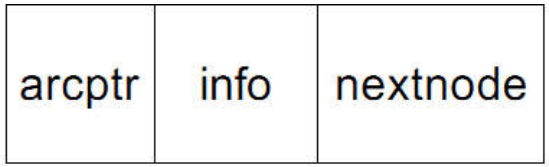
Um aluno atento já visualizou que a solução é usar…

uma estrutura ligada (encadeada), alocando e liberando nós a partir de uma lista disponível.

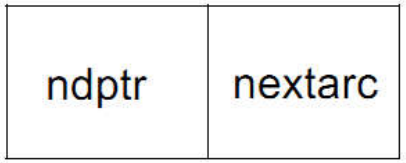
Proponha uma estrutura ligada capaz de representar adequadamente um grafo.

UNIVASF

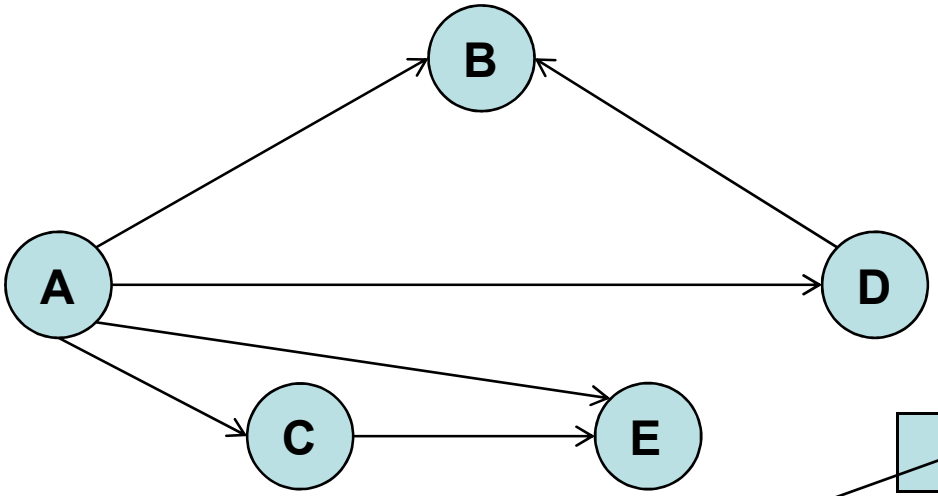
Grafos - Representação

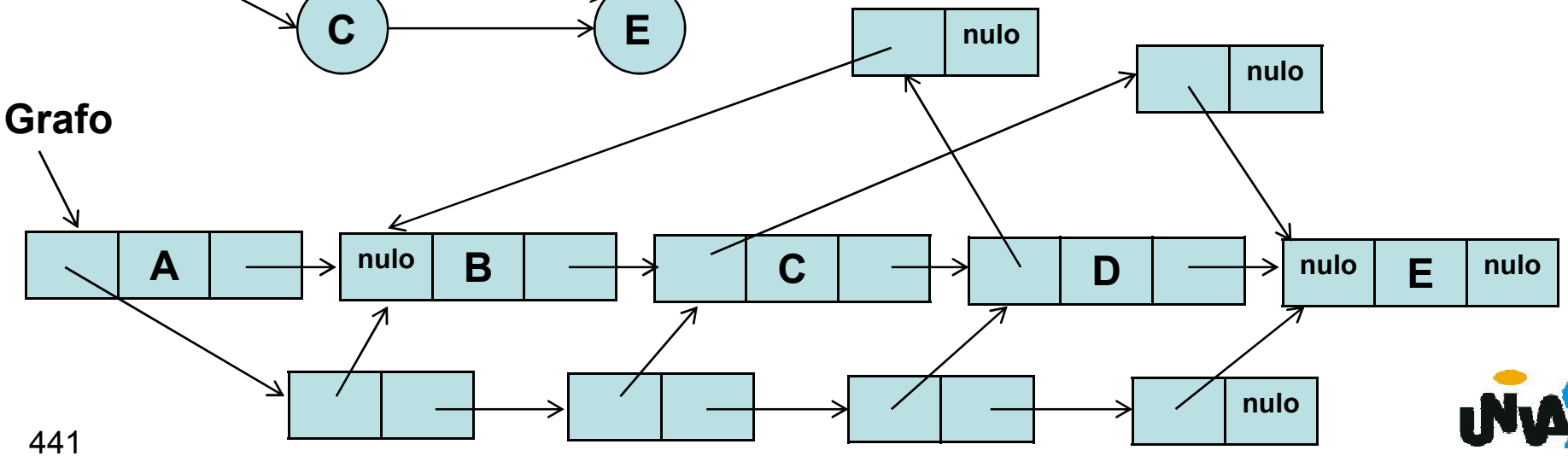


Um exemplo de um nó de cabeçalho representativo de um nó de grupo.



Um exemplo de um nó de lista representando um arco.





Grafos - Representação

Observe que os nós de cabeçalho e os de lista têm diferentes formatos e precisam ser representados por estruturas diferentes.

Isto requer a manutenção de duas listas disponíveis distintas ou a definição de uma união.

Mesmo no caso de um grafo ponderado no qual cada nó de lista contém um campo ***info*** para armazenar o peso de um arco, talvez sejam necessárias duas estruturas diferentes se a informação nos nós de cabeçalho não for um inteiro.

Entretanto, para simplificar, presumiremos que ambos os nós, de cabeçalho e de lista, têm o mesmo formato e incluem dois ponteiros e um só campo de informação inteiro.

UNIVASF

Grafos - Representação

Esses nós são declarados usando a implementação em vetor, como:

```
#define MAXNODES 500

typedef struct nodetype {
   int info;
   int point ;
   int next;
}tipoNodo;

typedef tipoNodo listaDeNodos[MAXNODES];
```


UNIVASF

Grafos - Representação

No caso de um nó de cabeçalho, **node[p]** representa um nó de grafo **A**, **node[p].info** representa a informação associada ao nó de grafo **A**, **node[p].next** aponta para o próximo nó de grafo, e **node[p].point** aponta para o primeiro nó da lista representando um arco emanando de **A**. No caso de um nó de lista, **node[p]** representa um arco **<A,B>**, **node[p].info** representa o peso do arco, **node[p].next** aponta para o próximo arco emanando de **A**, e **node[p].point** aponta para o nó de cabeçalho representando o nó de grafo **B**.

UNIVASF

Grafos - Representação

Como alternativa, podemos usar a implementação dinâmica declarando os nós como segue:

```
struct nodetype {
   int info;
   struct nodetype *point;
   struct nodetype *next;
};
struct nodetype *nodeptr;
```

Por hora usaremos a implementação em vetor e presumiremos, inicialmente, a existência das rotinas ***getnode*** e ***freenode***.

UNIVASF

Grafos - Representação

Implementaremos agora as operações primitivas de grafos usando a representação ligada em vetor.

Implemente a operação **joinwt(node, p,q,wt)** a qual recebe uma lista de nós, dois ponteiros, **p** e **q**, para dois nós de cabeçalho e cria um arco entre eles com peso **wt**. Se já existir um arco entre eles, o peso desse arco será definido com o valor de **wt**.

Observação: Lembre-se da posssibilidade de presumir a existência da rotina ***getnode***.

UNIVASF