Grafos

February 4, 2021

Contents

1 Tipos de grafos

1.1 Grafos Orientados

Um grafo orientado é um par (V,E) :

- V : Um conjunto finito de vértices ou nós
- E : Relaçáo binária sobre V (o conjunto de arestas ou arcos do grafo)

1.1.1 Exemplo de um grafo

Exemplo de um grafo

- Se (i,j)$\in$ E, j diz-se adjacente a i i e j são respectivamente os vértices de origem e destino da aresta (i,j)
- O grau de entrada de um vértice é o número de arestas com destino nesse vértice
- O grau de saída de um vértice é o número de arestas com origem nesse vértice
- Uma aresta (i,i) designa-se por anel
- O número máximo de arestas de um grafo com conjunto de vértices V é V^2
- Um grafo diz-se esparso se o número de arestas for muito inferior a V^2, e denso em caso contrário

1.1.2 Representação em Computador de Grafos Orientados

Matrix Grafo

```
#define MAX 100
typedef char GraphM[MAX][MAX];
```

1. Tempo
   - O espaço de memória necessário para representar um grafo (V,E) é $\Theta(V^2)$, independente do número de arcos, e portanto da densidade do grafo;
   - é poss+ivel verificar em tempo constante se dois vértices são adjacentes
   - para conhecer os adjacentes a um determinado vértice u, é necessário percorrer todos os vértices v do grafo, consultando para cada um a posição (u,v) da matriz.

1.1.3 Representação por Listas de Adjacências

Uma representação alternativa consiste em associar a cada vértice do grafo uma lista contendo os vértices que lhe são adjacentes.

- um array de apontadores
- cujos índices correspondem aos vértices do grafo, e

- contendo em cada posição o (endereço do) primeiro elemento da lista de adjacência do vértice respectivo

Exemplo

```
#define MAX 100

struct edge {
  int dest;
  struct edge *next;
};
typedef struct edge *GraphL[MAX];
```

1. Tempo

   - O espaço de memória necessário para esta representação de um grafo (V,E) é Θ(V+E), o que a torna uma representação eficiente no caso de grafos esparsos;
   - o teste de adjacência de dois vértices obriga à travessia de uma lista ligada, executada no pior caso em tempo Θ(V);
   - a consulta dos adjacentes a um vértice u implica apenas percorrer a lista de adjacências de u, em vez de percorrer todos os vértices. Num grafo pouco denso, o impacto na implementação de um algoritmo que efectue esta operação repetidamente pode ser muito grande. É o caso por exemplo dos algoritmos de travessia de grafos.

1.2 Grafos com pesos

Em certos contextos é útil associar informação (pesos) às arestas de um grafo, em particular numérica.

1.2.1 Representação em Computador de Grafos com Pesos

1. Matriz Exemplo

   Em C:

   ```
   #define NE 0 // Nodo vazio
   #define MAX 100
   typedef int WEIGHT;
   typedef WEIGHT GraphM[MAX][MAX];
   ```

1.2.2 Representação por Listas de Adjacências

- As arestas do grafo são aqui representadas por nós das listas ligadas de adjacências. Sendo assim, basta criar um campo adicional nestas estruturas (nós das listas) para guardar o peso das arestas.

- Note-se que, uma vez que nesta representação o teste de existência de uma aresta não é feito pela consulta do valor numérico do peso (como era o caso com uma matriz de adjacências), mas sim pela existência ou não de um nó com um determinado destino na lista ligada, não é agora necessária a utilização de um valor especial NE para representar o peso de uma aresta inexistente

Em C:

```
#define NE 0 // Valor de uma aresta inexistente
#define MAX 100
typedef int WEIGHT;
struct edge {
  int dest;
  WEIGHT weight;
  struct edge *next;
};
typedef struct edge *GraphL[MAX];
```

1.3 Grafos não orientados

Num grafo não-orientado as arestas são conjuntos com dois vértices \u,v\ $\in$ E em vez de pares ordenados. Por outras palavras, as aresta são bi-direccionais, o que é adequado, por exemplo, para modelar redes em que todas as ligações entre pares de vértices funcionam nos dois sentidos. Exemplo

1.3.1 Representação em Computador de Grafos Não-orientados

A representação típica de um grafo não-orientado passa pela sua conversão para um grafo orientado simétrico, em que se \u,v\ $\in$ E então também \v,u\ $\in$ E.

Note-se que uma tal representação contém redundância:

- No caso da representação por uma matriz de adjacências poder-se-á eliminar esta redundância representando de forma eficiente apenas uma matriz triangular.

- No caso da representação por listas de adjacências a eliminação da redundância será quase de certeza uma má ideia. Se se representar a aresta (u,v) $\in$ E apenas por um nó, na lista de adjacência de u ou de v, então, para ter acesso a todos os vértices adjacentes a um qualquer nó não bastará percorrer a sua lista de adjacências; será necessário percorrer todas as listas de adjacências do grafo.

1.4 Codigo

```
#include <stdio.h>
#include <stdlib.h>

#define N 8

enum search { DFirst, BFirst };

typedef struct aresta {
  int destino, peso;
  struct aresta *prox;
} * LAdj;

typedef LAdj Grafo[N];

/* Adicionar uma aresta */
Ladj newA(int dest, int peso, Ladj t){
 Ladj new = malloc(sizeof(struct aresta));
 new->destino = dest ;
 new->peso = peso;
 new->prox = t;
 return new;
}

/* Construir um Grafo a partir de uma matriz */
void controiGrafo(int mat[N][N], Grafo g) {
  for (int i = 0; i < N; i++) {
    g[i] = NULL;
    for (int j = 0; j < N; j++)
      if (mat[i][j] != 0)
        g[i] = newA(j, mat[i][j], g[i]);
```

```
  }
}

/* Determinar quantas arestas tem um determinado grafo */
int quantasArestas(Grafo g) {
  int n = 0;
  LAdj a;
  for (int i = 0; i < N; i++)
    for (a = g[i]; a; a = a->prox, n++)
      ;
  return n;
}

/* Determinar a capacidade de um grafo */
int capacidade(Grafo g, int v) {
  int n = 0;
  LAdj a;
  for (int i = 0; i < N; i++)
    for (a = g[i]; a; a = a->prox) {
      if (i == v)
        n -= a->peso;
      if (a->destino == v)
        n += a->peso;
    }
  return n;
}
```

Figure 1: Some functions

2 Algoritmos de Travessia de Grafos