

## 7.3 课堂笔记 17

### 本周作业（7.7 之前提交）

1. 邻接表的存储和遍历，自己根据参考代码练习两次
2. 洛谷 P1341（欧拉路）
3. 洛谷 P3916（邻接表写法）

### 上周作业答案

1. P7771

```
1  #include <iostream>
2  using namespace std;
3  int g[1010][1010];
4  int ru[100010];
5  int chu[100010];
6  int path[1000000];
7  int n, m, sta, en;
8  void dfs(int x){
9      for(int i = 1; i <= n; i++){
10         if(g[x][i] > 0){
11             g[x][i]--;
12             dfs(i);
13         }
14     }
15     en++;
16     path[en] = x;
17 }
18 int main(){
19     cin >> n >> m;
20     int x, y, cnt1 = 0, cnt2 = 0;
21     for(int i = 1; i <= m; i++){
22         cin >> x >> y;
23         g[x][y]++;
24         ru[y]++;
25         chu[x]++;
26     }
27     bool flag = false;
28     for(int i = 1; i <= n; i++){//判断是否是回路
29         if(chu[i] != ru[i]){
30             flag = true;
31         }
32     }
```

```

33  for(int i = 1; i <= n; i++){
34      if(chu[i] == ru[i] + 1){
35          sta = i;
36          cnt1++; //统计起点
37      }
38      if(ru[i] == chu[i] + 1){
39          cnt2++; //统计起点
40      }
41      if(chu[i] - ru[i] >= 2 || ru[i] - chu[i] >= 2){
42          cout << "No";
43          return 0;
44      }
45  }
46  if(!flag){
47      dfs(1);
48      for(int i = en; i >= 1; i--){
49          cout << path[i] << " ";
50      }
51      return 0;
52  }
53  if(cnt1 == 1 && cnt2 == 1){
54      dfs(sta);
55      for(int i = en; i >= 1; i--){
56          cout << path[i] << " ";
57      }
58      return 0;
59  }
60  cout << "No";
61  return 0;
62  }

```

## 2. P3916 (邻接矩阵写法, 60 分)

```

1  #include <iostream>
2  #include <string.h>
3  using namespace std;
4  int g[1010][1010];
5  int n, m;
6  bool visit[1010];
7  int maxi;
8  void dfs(int i){
9      visit[i] = true;
10     for(int j = 1; j <= n; j++){
11         if(g[i][j] == 1 && !visit[j]){
12             maxi = max(maxi, j);
13             dfs(j);
14         }
15     }
16 }

```

```

17
18 int main(){
19     cin >> n >> m;
20     int i = 0;
21     int j = 0;
22     int w = 0;
23     for(int k = 1; k <= m; k++){
24         cin >> i >> j;
25         g[i][j] = 1;
26     }
27     for(int i = 1; i <= n; i++){
28         maxi = i;
29         memset(visit, 0, sizeof(visit));
30         dfs(i);
31         cout << maxi << " ";
32     }
33     return 0;
34 }

```

## 课堂内容

### 1. 邻接矩阵:

string 是 C++ 的 STL 提供的字符串

#### 1) 邻接矩阵的优势:

- 直观、简单、好理解
- 方便检查任意一对顶点间是否存在边
- 方便找任一顶点的所有“邻接点”（有边直接相连的顶点）
- 方便计算任一顶点的度

#### 2) 邻接矩阵的局限性: 时间复杂度 $O(n^2)$ , 空间复杂度 $O(n^2)$

1. 浪费空间。对于稠密图还是很合算的。但是对于稀疏图（点很多而边很少）有大量无效元素。 $a[10000][10000]$  直接爆空间:
2. 浪费时间。要确定图中有多少条边, 则必须按行、按列对每个元素进行检测, 所花费的时间代价很大。

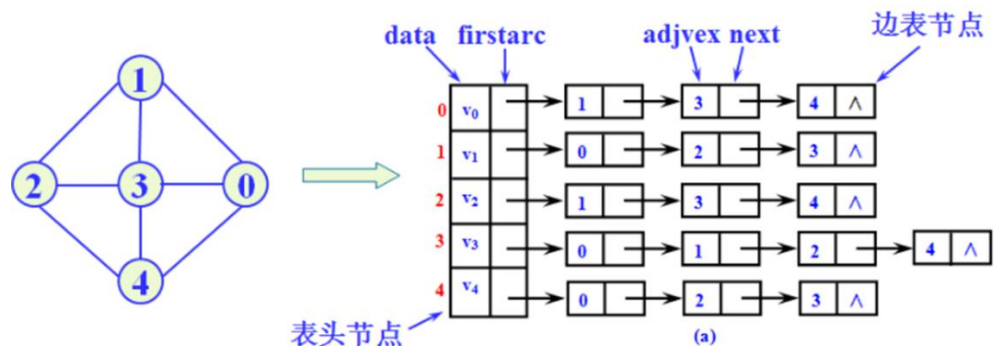
### 2. 邻接表的存储和访问:

图的邻接表存储方法是一种顺序分配与链式分配相结合的存储方法

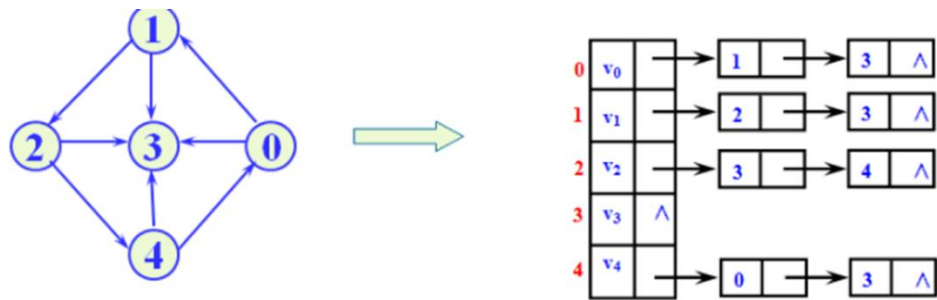
一般用数组模拟即可

#### 1) 无向图与有向图

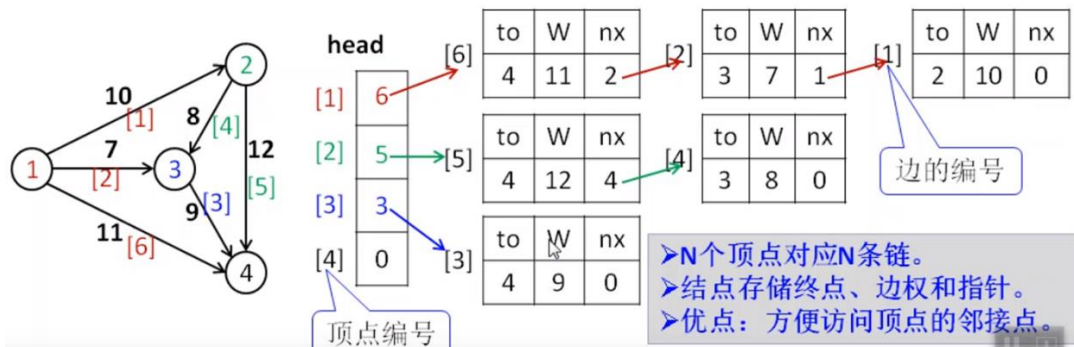
- 无向图



➤ 有向图

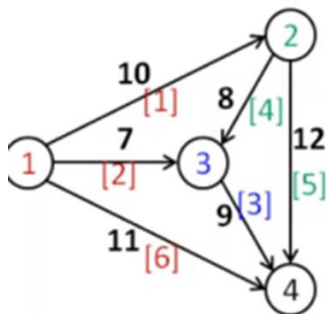


2) 数据结构



- head[i] 表头数组,存储当前顶点 i 的出边编号
- num 编号计数器,记录出边编号,也是下面三个数组的下标。
- to[num] 终点数组,存储 num 号边的终点,即顶点 i 的邻接点
- w[num] 边权数组,存储 num 号边的权值。
- next[num] 指针数组,存储 num 号边的下一条边

3. 邻接表的代码实现



{1,2,10}, {1,3,7},  
{3,4,9}, {2,3,8},  
{2,4,12}, {1,4,11}

		next	to	W
[1]	6	0	2	10
[2]	5	1	3	7
[3]	3	0	4	9
[4]	0	0	3	8
		4	4	12
		2	4	11

顶点编号      边的编号

### ➤ 初始化

```

6 struct edge{
7     int next; // 指针, 存储numEdge号边的下一条边
8     int to;   // 存储numEdge号边的终点, 即顶点i的邻接点
9     int w;    // 存储num号边的权值
10 } _e[110]; // 存边的数组, 下标为numEdge

```

### ➤ 添加边

```

12 // 添加一条从 from到to的权值为w的边
13 void AddEdge(int from, int to, int w) {
14     _numEdge++; // 边计数器++
15     _e[_numEdge].next = _head[from];
16     _e[_numEdge].to = to;
17     _e[_numEdge].w = w;
18     _head[from] = _numEdge; // 保存当前边的编号
19 }

```

### ➤ 遍历

```

28 for(int i = 1; i <= _n; i++) {
29     for(int j = _head[i]; j != 0; j = _e[j].next) {
30         cout << "顶点" << i << " -> 顶点" << _e[j].to << ", 长度:" << _e[j].w << endl;
31     }
32     cout << endl;
33 }
34 }

```

### ➤ 完整代码



```

4   int _numEdge = 0; // 初始化边的编号计算器
5   int _head[20]; // 以i点出边的编号数组
6   struct edge {
7       int next; // 指针, 存储numEdge号边的下一条边
8       int to; // 存储numEdge号边的终点, 即顶点i的邻接点
9       int w; // 存储numEdge号边的权值
10  } _e[110]; // 存边的数组, 下标为numEdge
11  // 添加一条从 from 到 to 的权值为 w 的边
12  void AddEdge(int from, int to, int w) {
13      _numEdge++; // 边计数器++
14      _e[_numEdge].next = _head[from];
15      _e[_numEdge].to = to;
16      _e[_numEdge].w = w;
17      _head[from] = _numEdge; // 保存当前边的编号
18  }
19  int main() {
20      int v1 = 0, v2 = 0, dis = 0;
21      cin >> _n >> _m;
22      for(int i = 1; i <= _m; i++) {
23          cin >> v1 >> v2 >> dis;
24          AddEdge(v1, v2, dis);
25      }
26      for(int i = 1; i <= _n; i++) {
27          for(int j = _head[i]; j != 0; j = _e[j].next) {
28              cout << "顶点" << i << " -> 顶点" << _e[j].to << ", 长度:" << _e[j].w << endl;
29          }
30          cout << endl;
31      }
32      return 0;

```

➤ 图的 DFS 遍历

```

1   void Dfs(int x) {
2       _visit[x] = true; // 记录走过
3       for(int i = _head[x]; i != 0; i = _e[i].next) {
4           int y = _e[i].to;
5           if(!_visit[y]) {
6               Dfs(y);
7           }
8       }
9   }

```

#### 4. 初赛试题测试

答案:

**几门级 CSP-J 第 15 套初赛模拟试题答案及解析**

一、单项选择题

题号	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
答案	C	D	C	B	C	C	A	A	D	B	A	C	B	B	D

**【解析】**

- 考查的是八进制的表示方法, 011 是八进制, 表示十进制的 9。
- 考查的是变量类型,  $2e-3$  是 double 类型的数, 占 8 个字节。
- 考查的是存储单位的换算, 网速的单位是 Mb/s, 1Byte = 8bit, 然后考虑到下载的时候, 不总是维持在最快的速度, 所以选择最接近的 C 答案。
- 考查的是与运算, 及网络号的计算, ipv4 中 ip 地址为 32 位, 分为 4 段。如果子网掩码的段是 255, 求与 & 运算后, 得到的还是原来的数值,  $10 \& 255 = 10$ ,  $20 \& 255 = 20$ , 最后一域  $220 \& 192 = 192$ , 因此结果选 B。

5. 考查的是 ASCII 码的规律, 空格的 ASCII 码为 32, 也可根据“A”的 ASCII 码为 65, “a”的 ASCII 码为 97, 从而推得空格的 ASCII 码为 32。
6. 考查的是 STL 中几个排序用法的区别, `stable_sort()` 是稳定的排序, 而 `sort()` 和 `qsort()` 都是不稳定的排序。`partial_sort` 部分排序也是不稳定的。
7. 考查常用数据结构的特点, `set` 用到的数据结构是平衡二叉树, 不是线性结构。
8. 考查 C++ 语言基础, `cin` 和 `cout` 是一种类 `class`。
9. 考查对计算机存储浮点数的特点, `double` 和 `float` 只是存储浮点数的精度不同, 换句话说, 当一个小数不能用二进制精确表示时, 存在 `double` 类型的变量里要比存在 `float` 变量里更接近原来真实的数, 和数的大小没有关系。
10. 考查内存的分类, 程序员自己开辟的内存空间是在“堆内存”中, 需要程序员自行进行回收。
11. 考查时间复杂度的相关内容, 以及对各个排序的认识。
12. 考查过河问题, 贪心算法, 1 和 2 过河, 花费 2, 1 回来, 花费 1; 10 和 5 过河, 花费 10, 2 回来, 花费 2; 1 和 2 一起过河, 花费 2, 总共为  $2+1+10+2+2=17$ 。
13. 考查康托展开公式, 52413 该序列展开后为:  $4 * 4! + 1 * 3! + 2 * 2! + 0 * 1! + 0 * 0!$ , 计算结果是 106。
14. 考查中国剩余定理, 3, 5, 7 是互质的, 它们的最小公倍数是 105, 第一个被 3, 5, 7 除, 余数为 2, 3, 2 的数是 23, 因此第 2 个满足条件的数为  $23+105$ , 第三个数满足条件的数位  $23+105 * 2$ , 以此类推, 第  $k$  个满足条件的数位  $23+105 * (k-1)$ 。
15. 考查学生知识面, 1993 年, 姚期智最先提出量子通信复杂性, 基本上完成了量子计算机的理论基础。1995 年, 提出分布式量子计算模式, 后来成为分布式量子算法和量子通讯协议安全性的基础。因为对计算理论包括伪随机数生成、密码学与通信复杂度的突出贡献, 美国计算机协会 (ACM) 也把 2000 年度的图灵奖授予他。

## 二、阅读程序

1.

题号	(1)	(2)	(3)	(4)	(5)	(6)
答案	✓	×	×	×	C	C

### 【解析】

- (1) `sscanf` 是从字符数组中进行读入。换句话说, 把字符数组作为输入缓存。
- (2) `scanf("%d\n", &n);` 改成 `scanf("%d", &n);` 后的结果是不一样的, 如果换成后者, 第 15 行将读取一个回车, 接下来程序执行的过程中, 最开始会输出一个回车, 并且少计算一行。
- (3) `gets(a)` 是针对字符数组的函数; `getline(cin, a)` 是针对字符串的函数, 如果用 `getline`, 程序将编译错误。
- (4) `puts(b)` 会自动加上回车, 应该改成 `cout<<b<<endl;`。
- (5) 输入 `d`, 运算符没法与之匹配, 最后将输出 0, 长度是 1。
- (6) 如果输入 `a`, 则做加法, 如果紧接着没有输入运算符, 则继续做加法。

2.

题号	(1)	(2)	(3)	(4)	(5)	(6)	(7)
答案	✓	×	×	×	✓	×	A

### 【解析】

这是一道高精度加法算法, 输入的数中可能含有前导零, 输出的数中去掉前导零。

- (1) C++ 语言中, `struct` 是一种特殊的类, 成员函数和成员属性都是 `public` 的。
- (2) `memset` 是逐个字节进行设置的。
- (3) 数据 0, 的长度是 1, 所以不能把 `len` 设置成 0。
- (4) `int i` 如果写在 `for` 里面, 那么当 `for` 结束的时候, `i` 变量就被系统回收了, 导致下面



for(int j=i;j>=1;j--)中i不能使用,而出现错误。

(5)在做加法的过程中,加数是不会发生变化的,所以用const去修饰加法结果不会发生变化;用引用也是同样的道理,在做加法的过程中,不需要重新开辟空间去存储加数,而使用原来变量中存储的加数。

(6)使用ans.a[i]+=a[i]+x.a[i];能将进位保留下来,而使用ans.a[i]=a[i]+x.a[i];如果有进位的情况,就会发生错误。

(7)程序第15行,能把前导0去除,因此,只留下了有效的数位。

3.

题号	(1)	(2)	(3)	(4)	(5)	(6)
答案	√	×	B	C	C	D

【解析】

这是一题01背包问题,用递归算法实现的程序。

(1)main函数和f函数都在各种的栈内存中,所以它们定义的n和m对应应在内存中的空间是不同的。

(2)当背包的容量为0时,是不会导致错误的。

(3)数据的意思是有2大小的背包,2个物品,花费和价值分别是1 2和3 4问最大价值是多少,刚好可以装下第一个物品,而不能装第2个,所以最大价值是2。

(4)数据的意思是有10大小的背包,2个物品,花费和价值分别是1 2和3 4问最大价值是多少,则可以装下全部2个物品,所以最大价值是6。

(5)数据的意思是有10大小的背包,4个物品,花费和价值分别是2 1,3 3,4 5,7 9问最大价值是多少,需要择优第2个和第4个物品组合最优秀,所以答案是12。

(6)思考方式,同上。

### 三、完善程序

1.

题号	(1)	(2)	(3)	(4)	(5)
答案	B	A	C	B	D

【解析】

(1)用int\*a做参数,函数可以接受传进来的数组的起始地址。

(2)进制转换的倒取余算法,先取得余数,然后再求商。

(3)cntaa表示:数平方后转成b进制后的长度,因为序号从0开始,最后一个有效数位的序号应该是cntaa-1。

(4)B选项里的\*cnt里面只能存地址,所以是错误的。

(5)把超过10的数,转变成字符。

2.

题号	(1)	(2)	(3)	(4)	(5)
答案	D	A	B	A	C

【解析】

这是用状态压缩dp来求解旅行商问题。

(1)用整数的二进制表示状态,根据数据规模,需要准备 $1 < 16$ 这个大的整数。0表示老鼠没有到达过这个点,1则相反。

(2)用i表示寻找小老鼠走过的点的编号。

(3)用j表示寻找小老鼠除了走过i以外,还走过的点的编号。

(4)从集合s(是一个整数,用到的是它的二进制)中去掉i这个点,表示规模更小的问题。

(5)当集合x的所有点都被走过了,才是最后的答案,所以x的二进制应该是全1的状态。