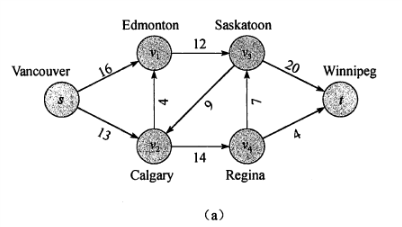
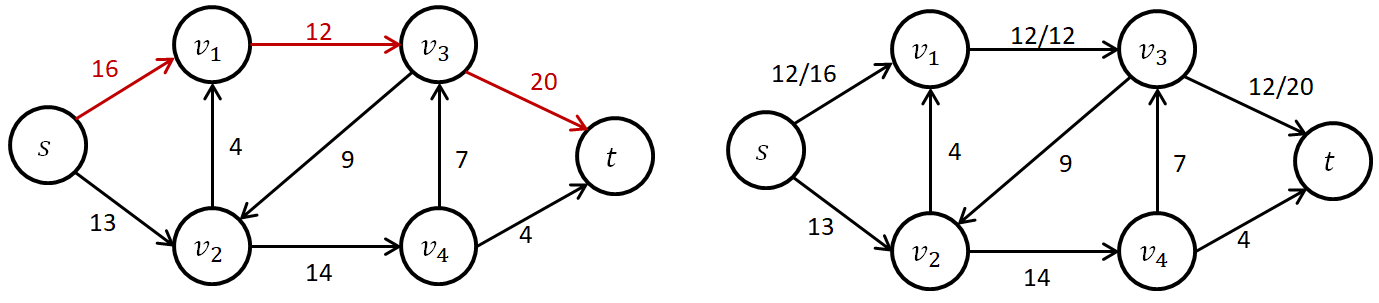
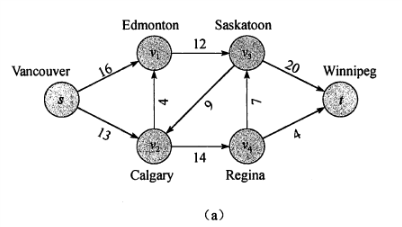
20174179 杨小川 计科卓越班

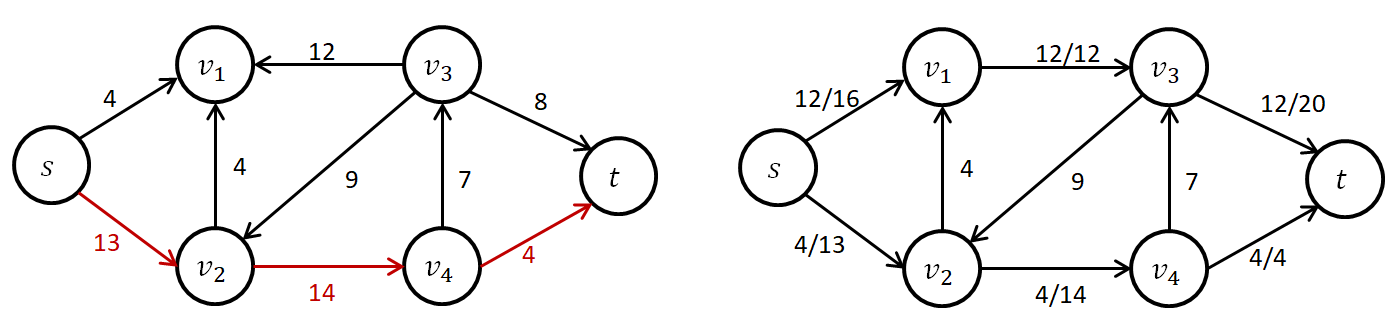
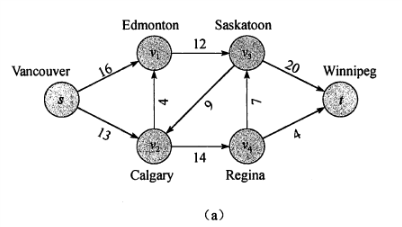
1. 26.2-3



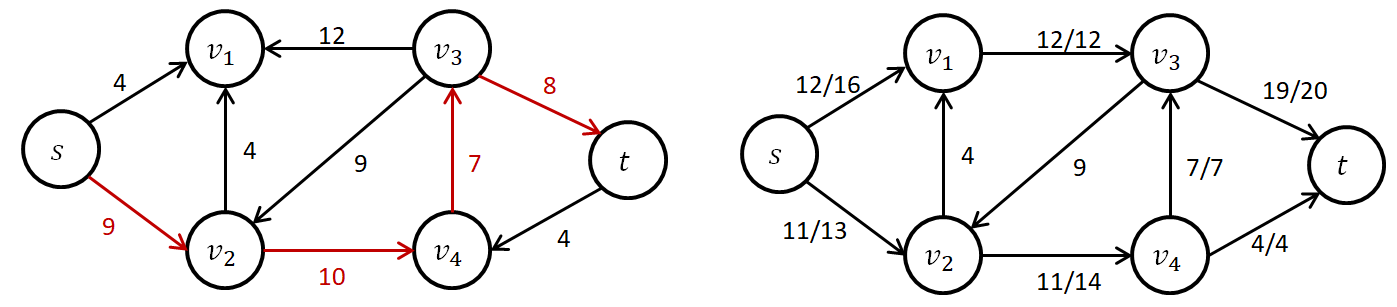
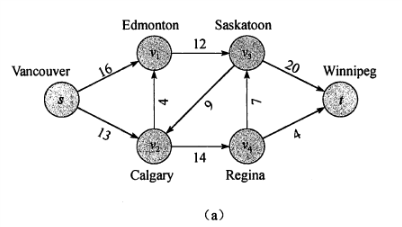
1：flow = 12



2：flow = 4

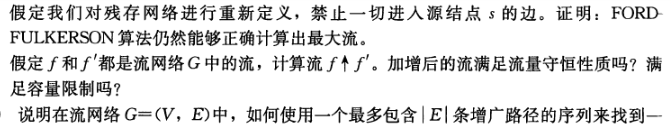


3：flow = 7



Maxflow = 23

1. 26.2-8



Ford-Fulkerson算法中循环选择的s->t都是简单路径，因为容量都是非负的，因此增广路径不会出现从s出发再回到s的情况，所以就算残存网络中有进入源节点s的边，也不会被选入增广路径中，删掉它们不会对算法的结果产生影响。因此如果残存网络中禁止一切进入源节点s的边，那么Ford-Fulkerson算法仍然能够正确计算出最大流。

(3) 选课：

一周7天，每天可排12节课。开设几百门课程供学生选课，每门课可以在一周内安排几次，但讲的内容一样，所以选了课的学生只需要一周上该门课程一次就行。编写程序帮助学生选课，使其能够选修尽量多的课程。

输入格式：

5 //课程总数(<= 7x12)

1 1 1 //第一门课程一周只安排了一次，安排在周1第1节

2 1 1 2 2 //第二门课程一周安排了2次，分别在周1第1节 和 周2第2节

1 2 2

2 3 2 3 3

时间

课程

1 3 3

c

c

s

t

c

c

c

c

#include<stdio.h>

#include<cstring>

#include<iostream>

#include<vector>

#include<queue>

#define MAX 12\*7\*2 + 1

#define S 0

using namespace std;

int n; *//* *class*

int maxflow = 0;

vector<int> Vertex;

int G[MAX][MAX] = {0}; *//* *Residual* *Graph*

int f[MAX] = {0}; *//* *flow*

int pre[MAX]; *//* *prevoius*

*/\**

*vertex:* *{S,* *1-n(class),* *times,* *T}*

*G:* *{S* *->* *class}*

*{class1* *->* *times}*

*......*

*{classn* *->* *times}*

*{times* *->* *T}*

*\*/*

int time(int day, int num) *//* *calculate* *time*

{

return n+12\*(day-1)+num;

}

void Edmonds\_Karp(int T) *//* *bfs* *using* *queue*

{

while(true)

{

memset(pre, -1, sizeof(pre));

memset(f, 0, sizeof(f));

f[S] = 9999;

queue<int> qVertex;

qVertex.push(S);

while(!qVertex.empty()) *//* *bfs* *find* *each* *Augmentation* *path*

{

int temp = qVertex.front();

qVertex.pop();

for(int i = 0; i < Vertex.size(); i++)

{

if(Vertex[i] != temp && G[temp][Vertex[i]] > 0

&& pre[Vertex[i]] == -1) *//* *p* *in* *Gf*

{

qVertex.push(Vertex[i]);

pre[Vertex[i]] = temp;

f[Vertex[i]] = min(f[temp],

G[temp][Vertex[i]]);

}

}

if(f[T] != 0) *//* *Augmentation* *exists*

break;

}

if(f[T]!=0) *//* *Augmentation* *exists*

{

for(int i = T;i != S;i = pre[i]) *//* *reset* *Gf*

{

G[pre[i]][i] -= f[T];

G[i][pre[i]] += f[T];

}

maxflow += f[T];

}

else

break;

}

}

void Build\_Graph(int T, int \*visited)

{

for(int Class=1;Class<=n;Class++)

{

int k,day,num;

cin >> k;

for(int j=0;j<k;j++)

{

cin >> day >> num;

G[Class][time(day,num)] = 1; *//* *each* *class* *must* *have* *class*

G[time(day,num)][T] = 1; *//* *each* *time* *have* *one* *class*

if(visited[time(day,num)] == 0)

{

Vertex.push\_back(time(day,num));

visited[time(day,num)]++;

}

}

}

Vertex.push\_back(T);

}

int main()

{

cin >> n;

const int T = n + 12\*7 + 1; *//* *T*

int visited[MAX] = {0};

for(int i=1;i<=n;i++)

{

G[S][i]=1;

Vertex.push\_back(i);

visited[i]++;

}

Build\_Graph(T, visited);

Edmonds\_Karp(T);

cout << maxflow << endl;

return 0;

}

