**七种qsort排序方法**

<本文中排序都是采用的从小到大排序>   
  
一、对int类型数组排序   
  
int num[100];   
  
Sample:   
  
int cmp ( const void \*a , const void \*b )   
{   
return \*(int \*)a - \*(int \*)b;   
}   
  
qsort(num,100,sizeof(num[0]),cmp);   
  
二、对char类型数组排序（同int类型）   
  
char word[100];   
  
Sample:   
  
int cmp( const void \*a , const void \*b )   
{   
return \*(char \*)a - \*(int \*)b;   
}   
  
qsort(word,100,sizeof(word[0]),cmp);   
  
三、对double类型数组排序（特别要注意）   
  
double in[100];   
  
int cmp( const void \*a , const void \*b )   
{   
return \*(double \*)a > \*(double \*)b ? 1 : -1;   
}   
  
qsort(in,100,sizeof(in[0]),cmp)；   
  
四、对结构体一级排序   
  
struct In   
{   
double data;   
int other;   
}s[100]   
  
//按照data的值从小到大将结构体排序,关于结构体内的排序关键数据data的类型可以很多种，参考上面的例子写   
  
int cmp( const void \*a ,const void \*b)   
{   
return (\*(In \*)a).data > (\*(In \*)b).data ? 1 : -1;   
}   
  
qsort(s,100,sizeof(s[0]),cmp);   
  
五、对结构体二级排序   
  
struct In   
{   
int x;   
int y;   
}s[100];   
  
//按照x从小到大排序，当x相等时按照y从大到小排序   
  
int cmp( const void \*a , const void \*b )   
{   
struct In \*c = (In \*)a;   
struct In \*d = (In \*)b;   
if(c->x != d->x) return c->x - d->x;   
else return d->y - c->y;   
}   
  
qsort(s,100,sizeof(s[0]),cmp);   
  
六、对字符串进行排序   
  
struct In   
{   
int data;   
char str[100];   
}s[100];   
  
//按照结构体中字符串str的字典顺序排序   
  
int cmp ( const void \*a , const void \*b )   
{   
return strcmp( (\*(In \*)a)->str , (\*(In \*)b)->str );   
}   
  
qsort(s,100,sizeof(s[0]),cmp);   
  
七、计算几何中求凸包的cmp   
  
int cmp(const void \*a,const void \*b) //重点cmp函数，把除了1点外的所有点，旋转角度排序   
{   
struct point \*c=(point \*)a;   
struct point \*d=(point \*)b;   
if( calc(\*c,\*d,p[1]) < 0) return 1;   
else if( !calc(\*c,\*d,p[1]) && dis(c->x,c->y,p[1].x,p[1].y) < dis(d->x,d->y,p[1].x,p[1].y)) //如果在一条直线上，  
则把远的放在前面   
return 1;   
else return -1;   
}   
PS:   
  
其中的qsort函数包含在<stdlib.h>的头文件里，strcmp包含在<string.h>的头文件里

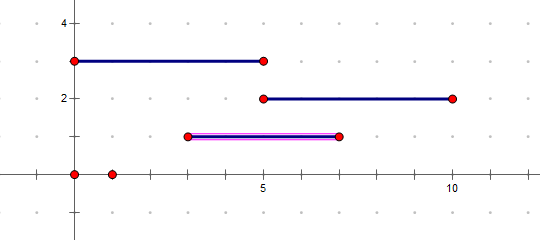
**贪心算法**

**活动安排问题之一**

有若干个活动，第i个开始时间和结束时间是[Si,fi)，只有一个教室，活动之间不能交叠，求最多安排多少个活动？  
  
分析： 我们就是想提高教室地利用率，尽可能多地安排活动。  
考虑容易想到的几种贪心策略：

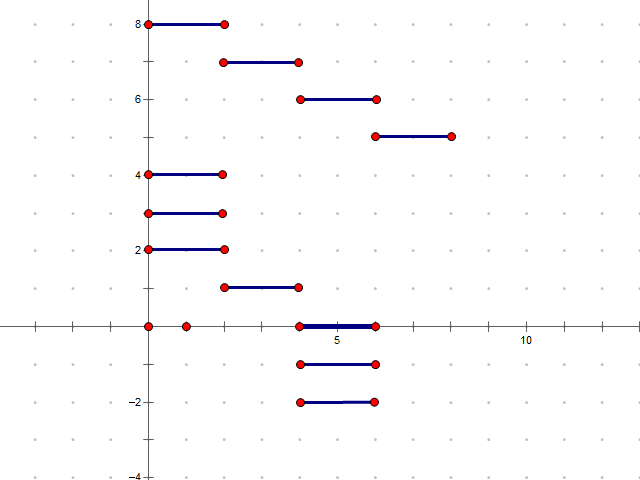
（1） 开始最早的活动优先，目标是想尽早结束活动，让出教室。  
然而， 这个显然不行，因为最早的活动可能很长，影响我们进行后面的活动。例如活动开始和结束时间分别为[0, 100), [1,2) ,[2, 3), [3, 4),[4,5]，安排［0，100)的这个活动之后，其他活动无法安排，可是最优解是安排除它外的4个活动。

（2） 短活动优先， 目标也是尽量空出教室。但是不难构造如下反例： [0,5) [5,10) [3, 7), 这里[3,7)最短，但如果我们安排了[3,7)，其它两个无法安排了。但是最优解显然是安排其它两个，而放弃[3,7)，可见这个贪心策略也是不行的。



（3） 最少冲突的活动优先， 既然上面安排活动是想减少冲突，那么如果我们优先安排冲突最少的活动可以么？至少从（1）和（2）看来，这个策略是有效的。真是对的么？ 尝试这个例子：

[0,2) [2,4) [4,6) [6,8)  
[1,3) [1,3) [1,3) [3,5) [5,7) [5,7) [5,7)



看一下[0,2) 和3个活动冲突——3个[1,3)

[2,4)和4个活动冲突3个[1,3)和一个[3,5)  
[4,6)和也和4个活动冲突3个[5,7)和一个[3,5)  
[6,8)和3个活动冲突——3个[5,7)

下面[1,3)和[5,7)每个都和5个活动冲突，  
而[3,5)只和两个活动冲突——[2,4)和[4,6)。

那按照我们的策略应该先安排[3,5), 可是一旦选择了[3,5)，我们最多只可能安排3个活动。  
但明显第一行的4个活动都可以安排下来，所以这种策略也是不对的。

（4） 看似最不对的策略——结束时间越早的活动优先。这个策略是有效的，我们可以证明。假设最优解OPT中安排了m个活动，我们把这些活动也按照结束时间由小到大排序，显然是不冲突的。假设排好顺序后，这些活动是a(1) , a(2), a(3)….am

假设按照我们的贪心策略，选出的活动自然是按照结束时间排好顺序的，并且也都是不冲突的，这些活动是b(1), b(2) …b(n)  
  
问题关键是，假设a(1) = b(1), a(2) = b(2)…. a(k) = b(k)，但是a(k+1) != b(k+1)，回答几个问题：

(1)b(k+1)会在a(k+2), a(k+3), …. a(m)中出现么？  
不会。因为b(k+1)的结束时间是最早的，即f(b(k+1)) <= f(a(k+1)),而a(k+2), a(k+3), …. a(m)的开始时间和结束时间都在f(a(k+1))之后，所以b(k+1)不在其中。

(2)b(k+1)和a(1), a(2), …. a(k) 冲突么？  
不冲突，因为a(1), a(2), …. a(k)就是b(1), b(2), …. b(k)

(3)b(k+1)和a(k+2), a(k+3), …. a(m)冲突么？  
不冲突，因为f(b(k+1)) <= f(a(k+1))，而a(k+2), a(k+3), …. a(m)的开始时间都在f(a(k+1))之后，更在f(b(k+1))之后。  
  
因此我们可以把a(k+1) 换成b(k+1)， 从而最优解和我们贪心得到的解多了一个相同的，经过一个一个替换，我们可以把最优解完全替换成我们贪心策略得到的解。 从而证明了这个贪心策略的最优性。

最后，我们来提供输入输出数据，由你来写一段程序，实现这个算法，只有写出了正确的程序，才能继续后面的课程。

**输入**

第1行：1个数N，线段的数量(2 <= N <= 10000)

第2 - N + 1行：每行2个数，线段的起点和终点(-10^9 <= S,E <= 10^9)

**输出**

输出最多可以选择的线段数量。

**输入示例**

3

1 5

2 3

3 6

**输出示例**

2

**活动安排问题之二**

有若干个活动，第i个开始时间和结束时间是[Si,fi)，活动之间不能交叠，要把活动都安排完，至少需要几个教室？

分析：能否按照之一问题的解法，每个教室安排尽可能多的活动，即按结束时间排序，再贪心选择不冲突的活动，安排一个教室之后，剩余的活动再分配一个教室，继续贪心选择……  
  
反例： A：[1,2)  B：[1,4) C：[5,6) D：[3，7)  
  
已经按结束时间排好顺序，我们会选择  
教室1： A C  
教室2:  B  
教室3:  D  
需要3个教室。  
但是如果换一种安排方法，我们可以安排AD在一个教室，而BC在另外一个教室，两个教室就够了。

所以之前的贪心策略解决不了这个问题。

怎么办？之前的策略是用一个教室找所有它能安排下的活动，即用教室找活动，我们能不能用活动找教室呢？

策略： 按照开始时间排序优先安排活动，如果冲突，则加一个教室。  
简单地理解一下，策略是这样，我们把活动按照开始时间有小到大的顺序排序。假设目前已经分配了k个教室（显然k初始等于0），对于当前这个活动，  
（1） 如果它能安排在k个教室里的某一个，则把它安排在其中的任何一个教室里，k不变。  
（2） 否则它和每个教室里的活动都冲突，则增加一个教室，安排这个活动。

这个策略是最优么？

我们想像一下k增加1的过程： 因为我们是按照开始时间排序的，意味着当前考虑的这个活动开始的时候，k个教室里都有活动没结束（因为如果有一个教室的活动结束了，我们就可以安排这个活动进入那个教室而不冲突，从而不用增加k)。这就意味着在这个活动开始的时间点，算上目前考虑的这个活动，有(k + 1)个活动正在进行，同一时刻有(k + 1)个活动在进行，无论我们如何安排教室，都至少需要(k + 1)个教室。因为每个教室里不能同时进行两个活动。而我们的策略恰好需要(k + 1)个教室，所以是最优的。

这个策略也告诉我们，如果从时间轴上“宏观”考虑这个问题。考虑每个时间点同时进行的活动个数，作为这个时间点的厚度（把活动开始和结束时间想像成线段，那么每个时间点有多少条线段覆盖它，可以简单理解为“厚度”），我们至少需要最大厚度那么多个教室——因为那时恰好有最大厚度那么多个活动同时进行，而我们这个贪心策略恰好给了我们一个用最大厚度那么多个教室安排全部活动的一个方案。

如果只需要教室的个数，我们可以把所有开始时间和结束时间排序，遇到开始时间就把厚度加1，遇到结束时间就把厚度减1，显然最初始和最后结束时的厚度是0，在一系列厚度变化的过程中，峰值（最大值）就是最多同时进行的活动数，也是我们至少需要的教室数。

**输入**

第一行一个正整数n (n <= 10000)代表活动的个数。

第二行到第(n + 1)行包含n个开始时间和结束时间。

开始时间严格小于结束时间，并且时间都是非负整数，小于1000000000

**输出**

一行包含一个整数表示最少教室的个数。

**输入示例**

3

1 2

3 4

2 9

**输出示例**

2

#include<cstdio>

#include<iostream>

#include<algorithm>

typedef long long LL;

using namespace std;

LL startime[10001];

LL endtime[10001];

int main()

{

int n;

cin>>n;

for(int i=0; i<=n-1; i++)

{

cin>>startime[i];

cin>>endtime[i];

}

sort(startime,startime+n);

sort(endtime,endtime+n);

int sum=0;

int a=0;

int b=0;

int memory=0;

for(a=0; a<=n-1; a++)

if(startime[a]<endtime[b])

{

sum++;

if(sum>memory) memory=sum;

}

else b++;

cout<<memory;

return 0;

}

**独木舟问题**

n个人，已知每个人体重，独木舟承重固定，每只独木舟最多坐两个人，可以坐一个人或者两个人。显然要求总重量不超过独木舟承重，假设每个人体重也不超过独木舟承重，问最少需要几只独木舟？  
  
分析：

一个显然的策略是按照人的体重排序。

极端化贪心策略，最重的人要上船——如果最重的人和最轻的人体重总和不超过船的承重，则他们两个占用一条船。否则（因为假设最重的人的体重也不超过船的承重了），最重的人单独占一条船。转变为(n – 1)或者(n – 2)的问题了。

关键在于这种贪心策略是正确的。我们可以证明，最优解也可以变为这种策略。

（1） 假设最重的人和最轻的人的体重和超过了船的承重，那么最优解中，显然也是最重的人单独占一条船，所以这种情况下最优解和贪心策略是相同的。

（2） 假设最重的人和最轻的人的体重和没超过船的承重。

（2.1）如果最优解中，最重的人单独占用一条船，则可以把最轻的人也放上去，这样最优解用的船数不增加。如果最轻的人占用一条船，同样我们可以把最重的人放上去，最优解船数不增。  
  
（2.2） 如果最优解中最重的人x和x’占用一只船(x, x’)，而最轻的人y和y’占用一只船(y, y’)  
我们换成(x, y) (x’,y’)

(x, y)显然没超过船的承重——因为我们假设就是如此。关键看(x’, y’)。

x’ + y’<= x’ + x 因为(x’, x)没超重，所以(x’,y’)也合法。所以换一下，最优解船数也不增。**这样我们就证明了如果可能把最重的人和最轻的人放在一条船上，不会影响最优解。**

反复应用这个策略，就可以把n降低为(n – 1)或者(n – 2)个人的规模，从而解决这个问题。

**输入**

第一行包含两个正整数n (0<n<=10000)和m (0<m<=2000000000)，表示人数和独木舟的承重。

接下来n行，每行一个正整数，表示每个人的体重。体重不超过1000000000，并且每个人的体重不超过m。

**输出**

一行一个整数表示最少需要的独木舟数。

**输入示例**

3 6

1

2

3

**输出示例**

2

#include <iostream>

#include <cstdio>

#include <algorithm>

using namespace std;

int main()

{

long long i,j,k=0,n,m,a[10005];

scanf("%lld%lld",&m,&n);

for(i=0; i<m; i++)

scanf("%lld",&a[i]);

sort(a,a+m);

for(i=0,j=m-1; i<=j;)

if(a[i]+a[j]<=n)

{

i++;

j--;

k++;

}

else if(i==j)

k++;

else

{

j--;

k++;

}

printf("%lld\n",k);

return 0;

}

**任务执行顺序**

分析： 本题可以抽象成，从一个整数开始，每次减去a，再加上b (a,b都是正数)，要求每次操作都不产生负数。

针对本题a[i] = R[i], b[i] = R[i] – O[i]，注意O[i] < R[i],我们有0<b[i]<a[i]。 所以尽管每次有减有加，但是加的没有减的多，总数还是在不断见效的。关键我们是要“最有利”的一种执行顺序。大家可以尝试多种贪心策略。

我们给出标准答案——按照b[i]不增的顺序排序，是最“有利”的。

为了定义“有利”，我们这样证明我们的结论：

如果对于b[0]>=b[1] >=…>=b[x] < b[x + 1]   
(a[0],b[0])….(a[x], b[x]) (a[x + 1], b[x + 1])的组合可以不产生负数，则我们交换b[x]和b[x + 1]也可以不产生负数。

证明：

交换(a[x], b[x])和(a[x + 1], b[x + 1])对x + 1更有利了，因为每个括号实际上是一个负数，所以越早安排这个括号，被减数就越大，就越不容易形成负数。  
关键看(a[x],b[x])移动到后面会不会产生负数。

那其实是看之前的结果 -a[x + 1] + b[x + 1] – a[x]会不会产生负数，（注意-a[x + 1] + b[x + 1]不会产生负数，因为我们刚才已经证明了，对x + 1更有利）

而我们知道之前的结果-a[x] + b[x] – a[x + 1]不会产生负数（因为我们的假设就是这样），而b[x + 1] > b[x]，所以前者更大，所以-a[x + 1] + b[x + 1] – a[x]不会产生负数。

因此我们证明了交换之后仍然不产生负数，也就是原先不产生负数，我们交换后仍然不产生负数。  
  
而经过若干次这样的交换之后，我们肯定会把序列交换成按照b的不增顺序排序的。从而我们证明了，任何可行的方案都不好于按照b不增顺序排序的序列执行的方案，从而证明了我们的贪心策略是有效的。  
  
很奇怪的策略——我们只考虑了b，居然能得到最优策略。可见贪心算法还是需要感觉，大胆假设，小心求证。

**输入**

第1行：1个数N，表示任务的数量。（2 <= N <= 100000)

第2 - N + 1行：每行2个数R[i]和O[i]，分别为执行所需的空间和存储所需的空间。(1 <= O[i] < R[i] <= 10000)

**输出**

输出执行所有任务所需要的最少空间。

**输入示例**

20

14 1

2 1

11 3

20 4

7 5

6 5

20 7

19 8

9 4

20 10

18 11

12 6

13 12

14 9

15 2

16 15

17 15

19 13

20 2

20 1

**输出示例**

135

**#include <stdio.h>**

**#include <stdlib.h>**

**struct In**

**{**

**int x,y,z;**

**}a[100000];**

**int cmp( const void \*a ,const void \*b)**

**{**

**return (\*(struct In \*)a).z > (\*(struct In \*)b).z ? 1 : -1;**

**}**

**int main()**

**{**

**int i,t,temp,sum=0;**

**scanf("%d",&t);**

**for(i=0;i<t;i++)**

**{**

**scanf("%d%d",&a[i].x,&a[i].y);**

**a[i].z=a[i].x-a[i].y;**

**}**

**qsort(a,t,sizeof(a[0]),cmp);**

**for(i=t-1,temp=0;i>=0;i--)**

**if(temp>=a[i].x)temp-=a[i].y;**

**else**

**{**

**sum+=a[i].x-temp;**

**temp=a[i].x-a[i].y;**

**continue;**

**}**

**printf("%d\n",sum);**

**return 0;**

**}**

**经典贪心**

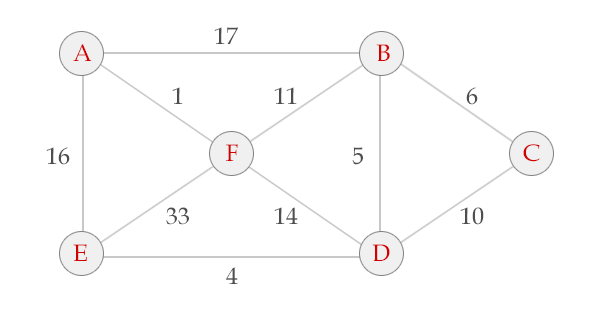
**Prim算法**

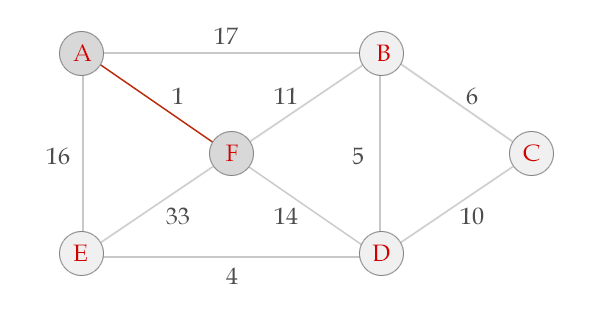
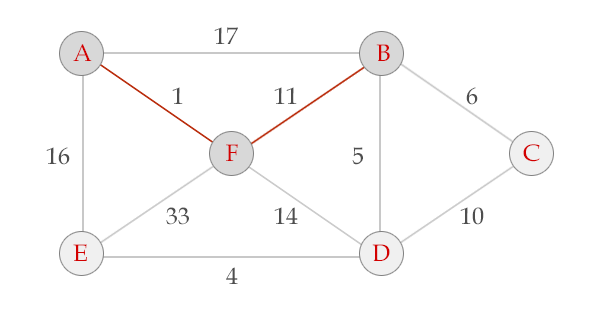
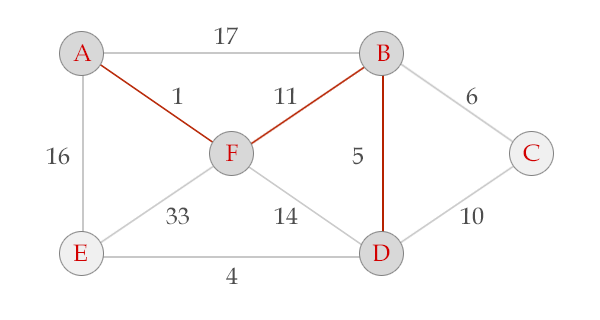
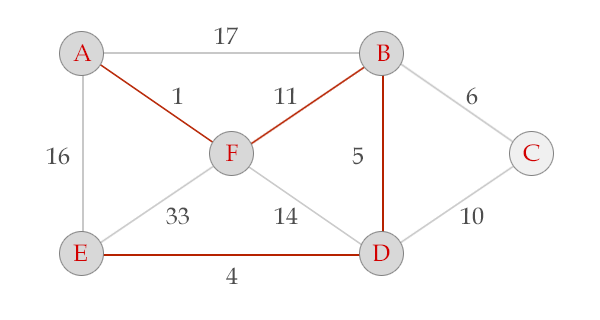
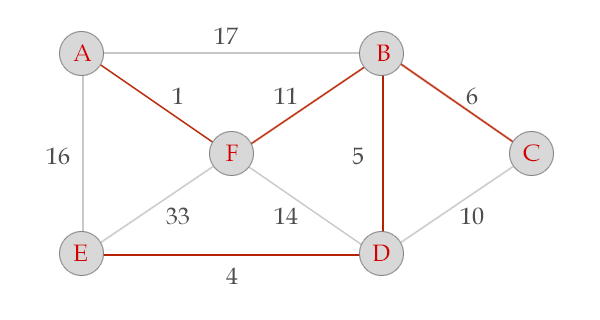
最小生成树的Prim算法也是贪心算法的一大经典应用。Prim算法的特点是时刻维护一棵树，算法不断加边，加的过程始终是一棵树。

Prim算法过程：  
  
一条边一条边地加， 维护一棵树。

初始 E ＝ ｛｝空集合， V = ｛任意节点｝  
  
循环（n – 1）次，每次选择一条边（v1,v2）， 满足：v1属于V , v2不属于V。且（v1,v2）权值最小。  
  
E = E + （v1,v2）  
V = V + v2  
  
最终E中的边是一棵最小生成树， V包含了全部节点。

以下图为例介绍Prim算法的执行过程。



Prim算法的过程从A开始 V = {A}, E = {}  
  
选中边AF , V = {A, F}, E = {(A,F)}   
  
选中边FB, V = {A, F, B}, E = {(A,F), (F,B)}  
  
选中边BD, V = {A, B, F, D},   E = {(A,F), (F,B), (B,D)}  
  
选中边DE, V = {A, B, F, D, E},   E = {(A,F), (F,B), (B,D), (D,E)}  
   
选中边BC, V = {A, B, F, D, E, c},   E = {(A,F), (F,B), (B,D), (D,E), (B,C)}, 算法结束。

Prim算法的证明：假设Prim算法得到一棵树P，有一棵最小生成树T。假设P和T不同，我们假设Prim算法进行到第(K – 1)步时选择的边都在T中，这时Prim算法的树是P’, 第K步时,Prim算法选择了一条边e = (u, v)不在T中。假设u在P’中，而v不在。  
  
因为T是树，所以T中必然有一条u到v的路径，我们考虑这条路径上第一个点u在P’中，最后一个点v不在P’中，则路径上一定有一条边f = (x,y)，x在P’中，而且y不在P’中。  
我们考虑f和e的边权w(f)与w(e)的关系：

若w(f) > w(e)，在T中用e换掉f （T中加上e去掉f)，得到一个权值和更小的生成树，与T是最小生成树矛盾。  
若w(f) < w(e), Prim算法在第K步时应该考虑加边f，而不是e,矛盾。  
  
因此只有w(f) = w(e),我们在T中用e换掉f，这样Prim算法在前K步选择的边在T中了，有限步之后把T变成P,而树权值和不变， 从而Prim算法是正确的。  
请仔细理解Prim算法——时刻维护一棵生成树。我们的证明构造性地证明了所有地最小生成树地边权（多重）集合都相同！

**输入**

第1行：2个数N,M中间用空格分隔，N为点的数量，M为边的数量。（2 <= N <= 1000, 1 <= M <= 50000)

第2 - M + 1行：每行3个数S E W，分别表示M条边的2个顶点及权值。(1 <= S, E <= N，1 <= W <= 10000)

**输出**

输出最小生成树的所有边的权值之和。

**输入示例**

9 14

1 2 4

2 3 8

3 4 7

4 5 9

5 6 10

6 7 2

7 8 1

8 9 7

2 8 11

3 9 2

7 9 6

3 6 4

4 6 14

1 8 8

**输出示例**

37

#include <iostream>

#include <algorithm>

#include <vector>

using namespace std;

#define MAX 1005

#define INF 0x7fffffff

int N,M,S,E,W;

vector<vector<int> > val(MAX,vector<int>(MAX,INF));

long long prim(){

long long res=0;

vector<bool> vis(MAX,false);

vector<int> min(MAX,INF);

min[1]=0;

for(int i=1;i<=N;i++){

int j,k;

for(k=-1,j=1;j<=N;j++)

if(!vis[j]&&(k==-1||min[j]<min[k]))

k=j;

vis[k]=1;

res+=min[k];

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

if(!vis[i]&&val[k][i]<min[i])

min[i]=val[k][i];

}

return res;

}

int main(){

cin>>N>>M;

for(int i=0;i<M;i++){

cin>>S>>E>>W;

val[S][E]=val[E][S]=W;

}

cout<<prim()<<endl;

return 0;

**｝**