目录

[Lab 1 3](#_Toc58219208)

[Lab1-A 畅通工程 3](#_Toc58219209)

[一　题目 3](#_Toc58219210)

[二　算法思想 4](#_Toc58219211)

[三　代码 5](#_Toc58219212)

[四　结果 7](#_Toc58219213)

[Lab1-B Jungle Roads 9](#_Toc58219214)

[一　题目 9](#_Toc58219215)

[二　算法思想 11](#_Toc58219216)

[三　代码 11](#_Toc58219217)

[四　结果 16](#_Toc58219218)

[Lab1-C 还是畅通工程 17](#_Toc58219219)

[一　题目 17](#_Toc58219220)

[二　算法思想 18](#_Toc58219221)

[三　代码 18](#_Toc58219222)

[四　结果 22](#_Toc58219223)

[Lab1-D 通畅工程再续 23](#_Toc58219224)

[一　题目 23](#_Toc58219225)

[二　算法思想 24](#_Toc58219226)

[三　代码 24](#_Toc58219227)

[四　结果 29](#_Toc58219228)

[Lab1-E Truck History 30](#_Toc58219229)

[一　题目 30](#_Toc58219230)

[二　算法思想 31](#_Toc58219231)

[三　代码 31](#_Toc58219232)

[四　结果 36](#_Toc58219233)

[Lab 2 37](#_Toc58219234)

[Lab2-A My Huge Bag 37](#_Toc58219235)

[一　题目 37](#_Toc58219236)

[二　算法思想 39](#_Toc58219237)

[三　代码 39](#_Toc58219238)

[四　结果 40](#_Toc58219239)

[Lab2-B Beat That Monster!!! 41](#_Toc58219240)

[一　题目 41](#_Toc58219241)

[二　算法思想 43](#_Toc58219242)

[三　代码 43](#_Toc58219243)

[四　结果 44](#_Toc58219244)

[Lab2-C Merge Slimes!!! 45](#_Toc58219245)

[一　题目 45](#_Toc58219246)

[二　算法思想 47](#_Toc58219247)

[三　代码 47](#_Toc58219248)

[四　结果 49](#_Toc58219249)

[Lab2-D Happy TSP 50](#_Toc58219250)

[一　题目 50](#_Toc58219251)

[二　算法思想 52](#_Toc58219252)

[三　代码 52](#_Toc58219253)

[四　结果 54](#_Toc58219254)

[Lab2-E Crush Proper Gems 55](#_Toc58219255)

[一　题目 55](#_Toc58219256)

[二　算法思想 56](#_Toc58219257)

[三　代码 57](#_Toc58219258)

[四　结果 60](#_Toc58219259)

Lab 1

Lab1-A 畅通工程

HDU – 1232

一　题目

Description

某省调查城镇交通状况，得到现有城镇道路统计表，表中列出了每条道路直接连通的城镇。省政府“畅通工程”的目标是使全省任何两个城镇间都可以实现交通（但不一定有直接的道路相连，只要互相间接通过道路可达即可）。问最少还需要建设多少条道路？

Input

测试输入包含若干测试用例。每个测试用例的第1行给出两个正整数，分别是城镇数目N ( < 1000 )和道路数目M；随后的M行对应M条道路，每行给出一对正整数，分别是该条道路直接连通的两个城镇的编号。为简单起见，城镇从1到N编号。  
注意:两个城市之间可以有多条道路相通,也就是说  
3 3  
1 2  
1 2  
2 1  
这种输入也是合法的  
当N为0时，输入结束，该用例不被处理。

Output

对每个测试用例，在1行里输出最少还需要建设的道路数目。

Sample Input

4 2

1 3

4 3

3 3

1 2

1 3

2 3

5 2

1 2

3 5

999 0

0

Sample Output

1

0

2

998

二　算法思想

数据结构：并查集。

typedef struct city

{

    int nth;//序号

    int parent;//父亲节点

    int size;//子集规模

} City;

getParent()函数返回叶子节点的根节点并进行路径压缩。

merge()函数将两叶子节点所在的树合并。

算法：Kruskal

对于图中的所有边，尝试将改变加入生成树中，如果加入该边不使树中出现环，则将该边连接的树合并；否则移除该边。重复以上步骤直至所有边都被尝试添加过或生成树满足条件边数=点数-1。

三　代码

#*include* <iostream>

#*include* <vector>

#*include* <algorithm>

using namespace std;

typedef struct city

{

    int nth;//*序号*

    int parent;//*父亲节点*

    int size;//*子集规模*

*city*(int nthInit)//*构造函数*

    {

        nth = nthInit;

        parent = nthInit;

        size = 1;

    };

} City;

//*储存城市信息的并查集vector，创建为全局变量，方便在函数中访问*

vector<City> cityV;

//*IN:城市序号;OUT:所属并查集根城市序号*

//*获取某城市root*

int *getParent*(int leaf)

{

    vector<int> toUpdate;

    int root = leaf;

*while* (cityV*[*root*]*.parent != root)

    {

        toUpdate.*push\_back*(root);

        root = cityV*[*root*]*.parent;

    }

*for* (int i : toUpdate)

        cityV*[*i*]*.parent = root;

*return* root;

}

//*IN:公路连接的两城市序号*

//*合并并查集*

int *merge*(int setA, int setB)

{

    setA = *getParent*(setA);

    setB = *getParent*(setB);

//*将较小的树合并到较大的树中*

*if* (cityV*[*setA*]*.size < cityV*[*setB*]*.size)

    {

        cityV*[*setA*]*.parent = setB;

        cityV*[*setB*]*.size += cityV*[*setA*]*.size;

    }

*else*

    {

        cityV*[*setB*]*.parent = setA;

        cityV*[*setA*]*.size += cityV*[*setB*]*.size;

    }

*return* 0;

}

int *main*(void)

{

*while* (true)

    {

        int cityNum;

        int roadNum;

        cityV.*clear*();

        cin *>>* cityNum;

//*结束条件，城市数为0*

*if* (cityNum == 0)

*break*;

        cin *>>* roadNum;

//*初始化vector，因为城市序号从1开始，所以第一个元素无意义*

*for* (int i = 0; i <= cityNum; i++)

        {

            auto tempcity = *city*(i);

            cityV.*push\_back*(tempcity);

        }

*for* (int i = 0; i < roadNum; i++)

        {

            int cityBeg;

            int cityEnd;

*scanf*("%d %d", &cityBeg, &cityEnd);

*if* (*getParent*(cityBeg) != *getParent*(cityEnd))

            {

*merge*(cityBeg, cityEnd);

            }

        }

//*第一个元素无意义，初值为-1*

        int count = -1;

*for* (City i : cityV)

        {

//*对根节点计数，结果减1即为需要路径数量*

*if* (*getParent*(i.nth) == i.nth)

                count++;

        }

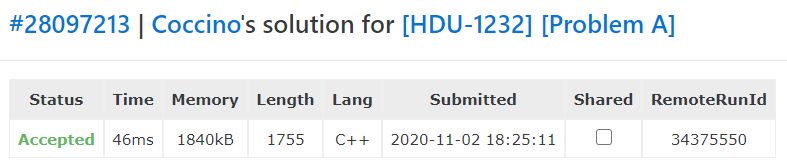
        cout *<<* count - 1 *<<* *endl*;

    }

*return* 0;

}

四　结果

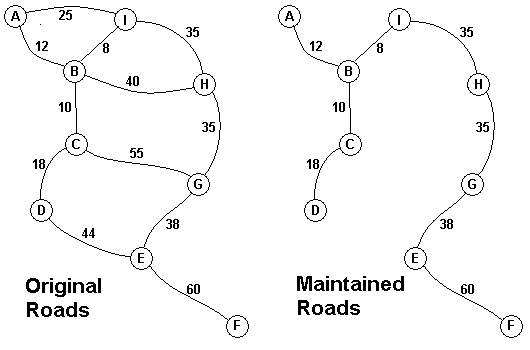


Lab1-B Jungle Roads

POJ - 1251

一　题目

Description



The Head Elder of the tropical island of Lagrishan has a problem. A burst of foreign aid money was spent on extra roads between villages some years ago. But the jungle overtakes roads relentlessly, so the large road network is too expensive to maintain. The Council of Elders must choose to stop maintaining some roads. The map above on the left shows all the roads in use now and the cost in aacms per month to maintain them. Of course there needs to be some way to get between all the villages on maintained roads, even if the route is not as short as before. The Chief Elder would like to tell the Council of Elders what would be the smallest amount they could spend in aacms per month to maintain roads that would connect all the villages. The villages are labeled A through I in the maps above. The map on the right shows the roads that could be maintained most cheaply, for 216 aacms per month. Your task is to write a program that will solve such problems.

Input

The input consists of one to 100 data sets, followed by a final line containing only 0. Each data set starts with a line containing only a number n, which is the number of villages, 1 < n < 27, and the villages are labeled with the first n letters of the alphabet, capitalized. Each data set is completed with n-1 lines that start with village labels in alphabetical order. There is no line for the last village. Each line for a village starts with the village label followed by a number, k, of roads from this village to villages with labels later in the alphabet. If k is greater than 0, the line continues with data for each of the k roads. The data for each road is the village label for the other end of the road followed by the monthly maintenance cost in aacms for the road. Maintenance costs will be positive integers less than 100. All data fields in the row are separated by single blanks. The road network will always allow travel between all the villages. The network will never have more than 75 roads. No village will have more than 15 roads going to other villages (before or after in the alphabet). In the sample input below, the first data set goes with the map above.

Output

The output is one integer per line for each data set: the minimum cost in aacms per month to maintain a road system that connect all the villages. Caution: A brute force solution that examines every possible set of roads will not finish within the one minute time limit.

Sample Input

9

A 2 B 12 I 25

B 3 C 10 H 40 I 8

C 2 D 18 G 55

D 1 E 44

E 2 F 60 G 38

F 0

G 1 H 35

H 1 I 35

3

A 2 B 10 C 40

B 1 C 20

0

Sample Output

216

30

二　算法思想

数据结构：并查集。

typedef struct village

{

    int parent; //父亲节点

    int size;   //子集规模

} Village;

getParent()函数返回叶子节点的根节点并进行路径压缩。

merge()函数将两叶子节点所在的树合并。

算法：Kruskal

将所有边按边权升序排序，每次尝试将最短的边加入生成树中。若加入该边不会使树中出现环，则将该边连接的树合并；否则移除该边。重复以上步骤直至所有边都被尝试添加过或生成树满足条件边数=点数-1。

三　代码

#*include* <iostream>

#*include* <vector>

#*include* <algorithm>

#*include* <stdio.h>

#*define* *villageMAX* 30

//*#define roadMAX 100*

using namespace std;

typedef struct village

{

//*int nth;    //序号*

    int parent;//*父亲节点*

    int size;//*子集规模*

*village*(){};

*village*(int nthInit)//*构造函数*

    {

//*nth = nthInit;*

        parent = nthInit;

        size = 1;

    };

} Village;

typedef struct road

{

    int beg;//*起始城市*

    int end;//*结束城市*

    int cost;//*花费*

*road*(){};

*road*(int a, int b, int c)//*构造函数*

    {

        beg = a;

        end = b;

        cost = c;

    };

} Road;

//*vector，创建为全局变量，方便在函数中访问*

vector<Village> *villageV*(*villageMAX*, *village*());

int villageNum;

//*vector<Road> roadV(roadMAX, road());*

vector<Road> roadV;

int roadNum;

vector<int> *toUpdate*(*villageMAX*);

//*IN:城市序号;OUT:所属并查集根城市序号*

//*获取某城市root*

int *getParent*(int leaf)

{

    int root = leaf;

//*int count=0;*

*while* (villageV*[*root*]*.parent != root)

    {

//*toUpdate[count] = root;*

//*count++;*

        root = villageV*[*root*]*.parent;

    }

//*for (int i = 0; i < count; i++)*

//*villageV[toUpdate[i]].parent = root;*

*return* root;

}

bool *roadCmp*(road a, road b)

{

//*return a.cost < b.cost;*

*return* a.cost > b.cost;

}

//*IN:公路连接的两城市序号*

//*合并并查集*

int *merge*(int setA, int setB)

{

//*将较小的树合并到较大的树中*

    villageV*[*setA*]*.parent = setB;

*return* 0;

}

bool *ifFinish*()

{

    int ref = *getParent*(0);

*for* (int j = 1; j < villageNum; j++)

    {

*if* (ref != *getParent*(j))

*return* false;

    }

*return* true;

}

int *main*(void)

{

*while* (true)

    {

        villageNum = 0;

        roadNum = 0;

*scanf*("%d", &villageNum);

*getchar*();

//*结束条件，城市数为0*

*if* (villageNum == 0)

*break*;

//*初始化城市vector*

*for* (int i = 0; i < villageNum; i++)

        {

            villageV*[*i*]*.parent = i;

            villageV*[*i*]*.size = 1;

        }

//*初始化道路vector*

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

        {

            int num;

            char villageBeg;

            cin *>>* villageBeg *>>* num;

//*scanf("%c %d", &villageBeg, &num);*

*getchar*();

//*villageBeg -= 'A';*

*for* (int j = 0; j < num; j++)

            {

                char villageEnd;

                int roadCost;

//*scanf("%c %d", &villageEnd, &roadCost);*

                cin *>>* villageEnd *>>* roadCost;

*getchar*();

//*roadV[roadNum].beg = i;*

//*roadV[roadNum].end = int(villageEnd - 'A');*

//*roadV[roadNum].cost = roadCost;*

                roadV.*push\_back*(*road*(i, int(villageEnd - 'A'), roadCost));

                roadNum++;

            }

        }

//*将道路按Cost从小到大排序*

*sort*(roadV.*begin*(), roadV.*end*(), *roadCmp*);

        int sum = 0;

//*从小到大以此尝试添加道路*

//*for (int i = 0; i < roadNum; i++)*

*while* (!roadV.*empty*())

        {

            Road roadAdd = roadV.*back*();

            roadV.*pop\_back*();

//*int rootA = getParent(roadAdd.beg);*

//*int rootB = getParent(roadAdd.end);*

            int rootA = roadAdd.beg;

            int rootB = roadAdd.end;

*while* (villageV*[*rootA*]*.parent != rootA)

                rootA = villageV*[*rootA*]*.parent;

*while* (villageV*[*rootB*]*.parent != rootB)

                rootB = villageV*[*rootB*]*.parent;

*if* (rootA != rootB)

            {

                villageV*[*rootA*]*.parent = rootB;

//*merge(rootA, rootB);*

                sum += roadAdd.cost;

            }

//*if (ifFinish())*

//*break;*

        }

//*printf("%d\n", sum);*

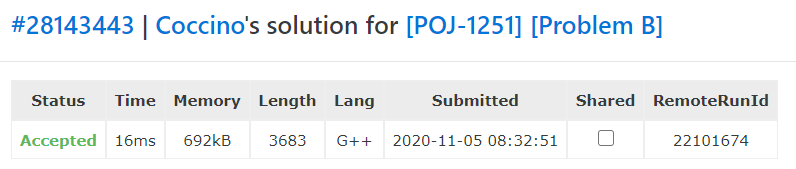
        cout *<<* sum *<<* *endl*;

    }

*return* 0;

}

四　结果



Lab1-C 还是畅通工程

HDU - 1233

一　题目

Description

某省调查乡村交通状况，得到的统计表中列出了任意两村庄间的距离。省政府“畅通工程”的目标是使全省任何两个村庄间都可以实现公路交通（但不一定有直接的公路相连，只要能间接通过公路可达即可），并要求铺设的公路总长度为最小。请计算最小的公路总长度。

Input

某省调查乡村交通状况，得到的统计表中列出了任意两村庄间的距离。省政府“畅通工程”的目标是使全省任何两个村庄间都可以实现公路交通（但不一定有直接的公路相连，只要能间接通过公路可达即可），并要求铺设的公路总长度为最小。请计算最小的公路总长度。

Output

对每个测试用例，在1行里输出最小的公路总长度。

Sample Input

3

1 2 1

1 3 2

2 3 4

4

1 2 1

1 3 4

1 4 1

2 3 3

2 4 2

3 4 5

0

Sample Output

3

5

二　算法思想

数据结构：并查集。

typedef struct city

{

    int parent;       //父亲节点

    int size;         //子集规模

} City;

getParent()函数返回叶子节点的根节点并进行路径压缩。

merge()函数将两叶子节点所在的树合并。

算法：Kruskal

对于图中的所有边，尝试将改变加入生成树中，如果加入该边不使树中出现环，则将该边连接的树合并；否则移除该边。重复以上步骤直至所有边都被尝试添加过或生成树满足条件边数=点数-1。

三　代码

#*include* <iostream>

#*include* <vector>

#*include* <algorithm>

#*include* <stdio.h>

using namespace std;

typedef struct city

{

//*int nth;          //序号*

    int parent;//*父亲节点*

    int size;//*子集规模*

*city*(int nthInit)//*构造函数*

    {

//*nth = nthInit;*

        parent = nthInit;

        size = 1;

    };

} City;

typedef struct road

{

    int beg;//*起始城市*

    int end;//*结束城市*

    int cost;//*花费*

*road*(int a, int b, int c)//*构造函数*

    {

        beg = a;

        end = b;

        cost = c;

    };

} Road;

//*储存城市信息的并查集vector，创建为全局变量，方便在函数中访问*

vector<City> cityV;

vector<Road> roadV;

//*IN:城市序号;OUT:所属并查集根城市序号*

//*获取某城市root*

int *getParent*(int leaf)

{

    vector<int> toUpdate;

    int root = leaf;

*while* (cityV*[*root*]*.parent != root)

    {

        toUpdate.*push\_back*(root);

        root = cityV*[*root*]*.parent;

    }

*for* (vector<int>::iterator i = toUpdate.*begin*(); i *<* toUpdate.*end*(); i*++*)

        cityV*[\**i*]*.parent = root;

    toUpdate.*clear*();

*return* root;

}

bool *roadCmp*(Road a, Road b)

{

*return* a.cost < b.cost;

}

//*IN:公路连接的两城市序号*

//*合并并查集*

int *merge*(int setA, int setB)

{

//*将较小的树合并到较大的树中*

*if* (cityV*[*setA*]*.size < cityV*[*setB*]*.size)

    {

        cityV*[*setA*]*.parent = setB;

        cityV*[*setB*]*.size += cityV*[*setA*]*.size;

    }

*else*

    {

        cityV*[*setB*]*.parent = setA;

        cityV*[*setA*]*.size += cityV*[*setB*]*.size;

    }

*return* 0;

}

bool *ifFinish*()

{

    int ref = *getParent*(0);

*for* (int i = 1; i < cityV.*size*(); i++)

    {

*if* (ref != *getParent*(i))

*return* false;

    }

*return* true;

}

int *main*(void)

{

*while* (true)

    {

        int cityNum;

        int roadNum;

        cityV.*clear*();

        roadV.*clear*();

*scanf*("%d", &cityNum);

*getchar*();

//*结束条件，城市数为0*

*if* (cityNum == 0)

*break*;

//*初始化城市vector*

*for* (int i = 0; i < cityNum; i++)

        {

            City tempcity = *city*(i);

            cityV.*push\_back*(tempcity);

        }

//*初始化道路vector*

*for* (int i = 0; i < cityNum \* (cityNum - 1) / 2; i++)

        {

            int cityBeg;

            int cityEnd;

            int roadCost;

*scanf*("%d %d %d", &cityBeg, &cityEnd, &roadCost);

*getchar*();

            roadV.*push\_back*(*road*(cityBeg - 1, cityEnd - 1, roadCost));

        }

//*将道路按Cost从小到大排序*

*sort*(roadV.*begin*(), roadV.*end*(), *roadCmp*);

        int sum = 0;

//*从小到大以此尝试添加道路*

*while* (!roadV.*empty*())

        {

            Road roadAdd = roadV.*front*();

            roadV.*erase*(roadV.*begin*());

            int rootA = *getParent*(roadAdd.beg);

            int rootB = *getParent*(roadAdd.end);

*if* (rootA != rootB)

            {

*merge*(rootA, rootB);

                sum += roadAdd.cost;

            }

*if* (*ifFinish*())

*break*;

        }

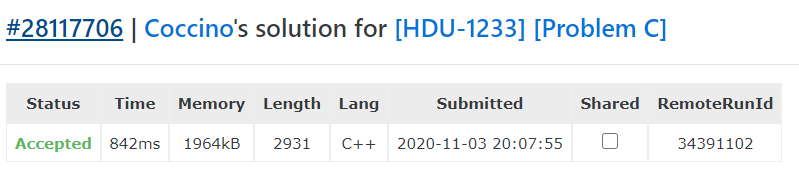
        cout *<<* sum *<<* *endl*;

    }

*return* 0;

}

四　结果



Lab1-D 通畅工程再续

HDU - 1875

一　题目

Description

相信大家都听说一个“百岛湖”的地方吧，百岛湖的居民生活在不同的小岛中，当他们想去其他的小岛时都要通过划小船来实现。现在政府决定大力发展百岛湖，发展首先要解决的问题当然是交通问题，政府决定实现百岛湖的全畅通！经过考察小组RPRush对百岛湖的情况充分了解后，决定在符合条件的小岛间建上桥，所谓符合条件，就是2个小岛之间的距离不能小于10米，也不能大于1000米。当然，为了节省资金，只要求实现任意2个小岛之间有路通即可。其中桥的价格为 100元/米。

Input

输入包括多组数据。输入首先包括一个整数T(T <= 200)，代表有T组数据。  
每组数据首先是一个整数C(C <= 100),代表小岛的个数，接下来是C组坐标，代表每个小岛的坐标，这些坐标都是 0 <= x, y <= 1000的整数。

Output

每组输入数据输出一行，代表建桥的最小花费，结果保留一位小数。如果无法实现工程以达到全部畅通，输出”oh!”.

Sample Input

2

2

10 10

20 20

3

1 1

2 2

1000 1000

Sample Output

1414.2

oh!

二　算法思想

数据结构：并查集。

typedef struct land

{

    int posX;

    int posY;

    int parent;//*父亲节点*

    int size;//*子集规模*

    };

} Land;

getParent()函数返回叶子节点的根节点并进行路径压缩。

merge()函数将两叶子节点所在的树合并。

算法：Kruskal

对于图中的所有边，尝试将改变加入生成树中，如果加入该边不使树中出现环，则将该边连接的树合并；否则移除该边。重复以上步骤直至所有边都被尝试添加过或生成树满足条件边数=点数-1。

三　代码

#*include* <iostream>

#*include* <vector>

#*include* <algorithm>

#*include* <stdio.h>

#*include* <math.h>

using namespace std;

typedef struct land

{

//*int nth;          //序号*

    int posX;

    int posY;

    int parent;//*父亲节点*

    int size;//*子集规模*

*land*(int nthInit, int posXInit, int posYInit)//*构造函数*

    {

//*nth = nthInit;*

        posX = posXInit;

        posY = posYInit;

        parent = nthInit;

        size = 1;

    };

} Land;

typedef struct bridge

{

    int beg;//*起始城市*

    int end;//*结束城市*

    double cost;//*花费*

*bridge*(int a, int b, double c)//*构造函数*

    {

        beg = a;

        end = b;

        cost = c;

    };

} Bridge;

//*储存城市信息的并查集vector，创建为全局变量，方便在函数中访问*

vector<Land> landV;

vector<Bridge> bridgeV;

//*IN:城市序号;OUT:所属并查集根城市序号*

//*获取某城市root*

int *getParent*(int leaf)

{

    vector<int> toUpdate;

    int root = leaf;

*while* (landV*[*root*]*.parent != root)

    {

        toUpdate.*push\_back*(root);

        root = landV*[*root*]*.parent;

    }

*for* (vector<int>::iterator i = toUpdate.*begin*(); i *<* toUpdate.*end*(); i*++*)

        landV*[\**i*]*.parent = root;

    toUpdate.*clear*();

*return* root;

}

bool *bridgeCmp*(Bridge a, Bridge b)

{

*return* a.cost < b.cost;

}

//*IN:公路连接的两城市序号*

//*合并并查集*

int *merge*(int setA, int setB)

{

//*将较小的树合并到较大的树中*

*if* (landV*[*setA*]*.size < landV*[*setB*]*.size)

    {

        landV*[*setA*]*.parent = setB;

        landV*[*setB*]*.size += landV*[*setA*]*.size;

    }

*else*

    {

        landV*[*setB*]*.parent = setA;

        landV*[*setA*]*.size += landV*[*setB*]*.size;

    }

*return* 0;

}

double *getDistance*(int i, int j)

{

*return* (*sqrt*(*pow*(landV*[*i*]*.posX - landV*[*j*]*.posX, 2) + *pow*(landV*[*i*]*.posY - landV*[*j*]*.posY, 2)));

}

bool *ifFinish*()

{

    int ref = *getParent*(0);

*for* (int i = 1; i < landV.*size*(); i++)

    {

*if* (ref != *getParent*(i))

*return* false;

    }

*return* true;

}

int *main*(void)

{

    int s;

*scanf*("%d", &s);

*getchar*();

*while* (s-- != 0)

    {

        int landNum;

        int bridgeNum;

        landV.*clear*();

        bridgeV.*clear*();

*scanf*("%d", &landNum);

*getchar*();

//*初始化城市vector*

*for* (int i = 0; i < landNum; i++)

        {

            int x, y;

*scanf*("%d %d", &x, &y);

*getchar*();

            Land templand = *land*(i, x, y);

            landV.*push\_back*(templand);

        }

//*初始化道路vector*

*for* (int i = 0; i < landNum; i++)

        {

*for* (int j = i + 1; j < landNum; j++)

            {

                double bridgeCost = *getDistance*(i, j);

*if* (bridgeCost < 10 || bridgeCost > 1000)

*continue*;

                bridgeV.*push\_back*(*bridge*(i, j, bridgeCost \* 100));

            }

        }

//*将道路按Cost从小到大排序*

*sort*(bridgeV.*begin*(), bridgeV.*end*(), *bridgeCmp*);

        double sum = 0;

        bool getAnswer = false;

//*从小到大以此尝试添加道路*

*while* (!bridgeV.*empty*())

        {

            Bridge bridgeAdd = bridgeV.*front*();

            bridgeV.*erase*(bridgeV.*begin*());

            int rootA = *getParent*(bridgeAdd.beg);

            int rootB = *getParent*(bridgeAdd.end);

*if* (rootA != rootB)

            {

*merge*(rootA, rootB);

                sum += bridgeAdd.cost;

            }

*if* (*ifFinish*())

            {

                getAnswer = true;

*break*;

            }

        }

*if* (getAnswer)

*printf*("%.1f\n", sum);

*else*

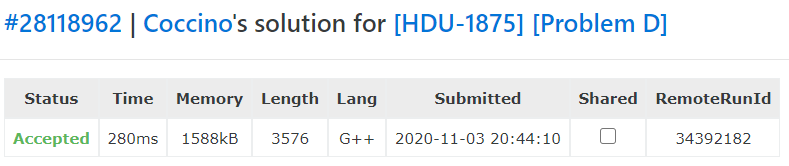
*printf*("oh!\n");

    }

*return* 0;

}

四　结果



Lab1-E Truck History

POJ - 1789

一　题目

Description

Advanced Cargo Movement, Ltd. uses trucks of different types. Some trucks are used for vegetable delivery, other for furniture, or for bricks. The company has its own code describing each type of a truck. The code is simply a string of exactly seven lowercase letters (each letter on each position has a very special meaning but that is unimportant for this task). At the beginning of company's history, just a single truck type was used but later other types were derived from it, then from the new types another types were derived, and so on.

Today, ACM is rich enough to pay historians to study its history. One thing historians tried to find out is so called derivation plan -- i.e. how the truck types were derived. They defined the distance of truck types as the number of positions with different letters in truck type codes. They also assumed that each truck type was derived from exactly one other truck type (except for the first truck type which was not derived from any other type). The quality of a derivation plan was then defined as

**1/Σ(to,td)d(to,td)**

where the sum goes over all pairs of types in the derivation plan such that to is the original type and td the type derived from it and d(to,td) is the distance of the types.

Since historians failed, you are to write a program to help them. Given the codes of truck types, your program should find the highest possible quality of a derivation plan.

Input

The input consists of several test cases. Each test case begins with a line containing the number of truck types, N, 2 <= N <= 2 000. Each of the following N lines of input contains one truck type code (a string of seven lowercase letters). You may assume that the codes uniquely describe the trucks, i.e., no two of these N lines are the same. The input is terminated with zero at the place of number of truck types.

Output

For each test case, your program `should output the text "The highest possible quality is 1/Q.", where 1/Q is the quality of the best derivation plan.

Sample Input

4

aaaaaaa

baaaaaa

abaaaaa

aabaaaa

0

Sample Output

The highest possible quality is 1/3.

二　算法思想

数据结构：并查集。

typedef struct truck

{

    char code[7];

    int parent;//*父亲节点*

    int size;//*子集规模*

} Truck;

getParent()函数返回叶子节点的根节点并进行路径压缩。

merge()函数将两叶子节点所在的树合并。

算法：Kruskal

对于图中的所有边，尝试将改变加入生成树中，如果加入该边不使树中出现环，则将该边连接的树合并；否则移除该边。重复以上步骤直至所有边都被尝试添加过或生成树满足条件边数=点数-1。

三　代码

#*include* <iostream>

#*include* <algorithm>

#*include* <stdio.h>

#*include* <fstream>

#*define* *MAX* 2010

using namespace std;

typedef struct truck

{

//*int nth;          //序号*

    char code[7];

    int parent;//*父亲节点*

    int size;//*子集规模*

*truck*(){};

*truck*(int nthInit)//*构造函数*

    {

//*nth = nthInit;*

        parent = nthInit;

        size = 1;

    };

} Truck;

typedef struct gap

{

    int beg;//*起始城市*

    int end;//*结束城市*

    int cost;//*花费*

*gap*(){};

*gap*(int a, int b, int c)//*构造函数*

    {

        beg = a;

        end = b;

        cost = c;

    };

} Gap;

//*储存城市信息的并查集vector，创建为全局变量，方便在函数中访问*

int truckNum;

Truck \*truckV = new Truck[*MAX*]();

Gap \*gapV = new Gap[*MAX* \* (*MAX* - 1) / 2]();

int \*toUpdate = new int[*MAX* \* (*MAX* - 1) / 2];

//*IN:城市序号;OUT:所属并查集根城市序号*

//*获取某城市root*

int *getParent*(int leaf)

{

    int root = leaf;

    int count = 0;

*while* (truckV[root].parent != root)

    {

        toUpdate[count] = root;

        count++;

        root = truckV[root].parent;

    }

*for* (int i = 0; i < count; i++)

        truckV[toUpdate[i]].parent = root;

*return* root;

}

bool *gapCmp*(Gap a, Gap b)

{

*return* a.cost < b.cost;

}

//*IN:公路连接的两城市序号*

//*合并并查集*

int *merge*(int setA, int setB)

{

//*将较小的树合并到较大的树中*

*if* (truckV[setA].size < truckV[setB].size)

    {

        truckV[setA].parent = setB;

        truckV[setB].size += truckV[setA].size;

    }

*else*

    {

        truckV[setB].parent = setA;

        truckV[setA].size += truckV[setB].size;

    }

*return* 0;

}

int *getGap*(int a, int b)

{

//*string aStr = truckV[a].code;*

//*string bStr = truckV[b].code;*

    int gap = 0;

*for* (int i = 0; i < 7; i++)

*if* (truckV[a].code[i] != truckV[b].code[i])

            gap++;

*return* gap;

}

bool *ifFinish*()

{

    int ref = *getParent*(0);

*for* (int i = 1; i < truckNum; i++)

    {

*if* (ref != *getParent*(i))

*return* false;

    }

*return* true;

}

int *main*(void)

{

//*freopen("E:\\algorithm\_EXP\\test-1-5.txt", "r", stdin);*

*while* (true)

    {

        int gapNum;

*scanf*("%d", &truckNum);

*getchar*();

*if* (truckNum == 0)

*break*;

//*初始化城市vector*

//*truckV = new truck[truckNum]();*

*for* (int i = 0; i < truckNum; i++)

        {

            truckV[i] *=* *truck*(i);

*scanf*("%s", truckV[i].code);

*getchar*();

        }

//*初始化道路数组*

//*gapV = new gap[truckNum \* (truckNum - 1) / 2]();*

        int count = 0;

*for* (int i = 0; i < truckNum; i++)

        {

*for* (int j = i + 1; j < truckNum; j++, count++)

            {

                int gapCost = *getGap*(i, j);

                gapV[count].beg = i;

                gapV[count].end = j;

                gapV[count].cost = gapCost;

            }

        }

//*将道路按Cost从小到大排序*

*sort*(gapV, gapV + truckNum \* (truckNum - 1) / 2, *gapCmp*);

//*从小到大以此尝试添加道路*

        int sum = 0;

*for* (int i = 0; i < truckNum \* (truckNum - 1) / 2; i++)

        {

            int rootA = gapV[i].beg;

*if* (rootA != truckV[rootA].parent)

                rootA = *getParent*(gapV[i].beg);

            int rootB = gapV[i].end;

*if* (rootB != truckV[rootB].parent)

                rootB = *getParent*(gapV[i].end);

*if* (rootA != rootB)

            {

*merge*(rootA, rootB);

                sum += gapV[i].cost;

            }

*if* (*ifFinish*())

*break*;

        }

*printf*("The highest possible quality is 1/%d.\n", sum);

//*delete[] gapV;*

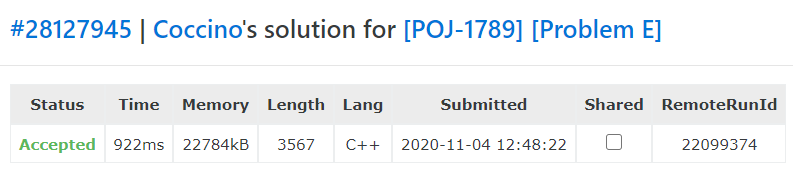
//*delete[] truckV;*

    }

*return* 0;

}

四　结果



Lab 2

Lab2-A My Huge Bag

AtCoder – dp\_e

一　题目

Description

There are NN items, numbered 1,2,…,N1,2,…,N. For each ii (1≤i≤N1≤i≤N), Item ii has a weight of wiwi and a value of vivi.

Taro has decided to choose some of the NN items and carry them home in a knapsack. The capacity of the knapsack is WW, which means that the sum of the weights of items taken must be at most WW.

Find the maximum possible sum of the values of items that Taro takes home.

Constraints

All values in input are integers.

1≤N≤1001≤N≤100

1≤W≤1091≤W≤109

1≤wi≤W1≤wi≤W

1≤vi≤103

Input

Input is given from Standard Input in the following format:

N W

w1w1 v1v1

w2w2 v2v2

: :

wNwN vNvN

Output

Print the maximum possible sum of the values of items that Taro takes home.

Sample Input 1

3 8

3 30

4 50

5 60

Sample Output 1

90

Items 11 and 33 should be taken. Then, the sum of the weights is 3+5=83+5=8, and the sum of the values is 30+60=9030+60=90.

Sample Input 2

1 1000000000

1000000000 10

Sample Output 2

10

Sample Input 3

6 15

6 5

5 6

6 4

6 6

3 5

7 2

Sample Output 3

17

Items 2,42,4 and 55 should be taken. Then, the sum of the weights is 5+6+3=145+6+3=14, and the sum of the values is 6+6+5=176+6+5=17.

二　算法思想

算法：动态规划

其中i≥0，j≥0，i为物品编号，dp[j]为背包中物品总价值为j时背包最小容量，dp[0] = 0。

遍历所有物品，每次更新所有dp[0]~dp[sum(value)]的值，最大的满足dp[j]<=W的j就是容量为W的背包能装下物品的最大总价值。

三　代码

#*include* <algorithm>

#*include* <iostream>

#*include* <vector>

#*define* *INF* 0x3fffffff

using namespace std;

int *main*()

{

    ios::*sync\_with\_stdio*(false);

    long long N, W;//*物品数量，背包容量*

    cin *>>* N *>>* W;

    vector<long long> *valueV*(N);

    vector<long long> *weightV*(N);

//*输入重量和价值信息*

    long long sumValue = 0;

*for* (long long i = 0; i < N; i++)

    {

        cin *>>* weightV*[*i*]* *>>* valueV*[*i*]*;

        sumValue += valueV*[*i*]*;

    }

    vector<long long> *dp*(sumValue + 1, *INF*);

    dp*[*0*]* = 0;

//*对于每个物品，更新能够使总value到达某值所需最小重量*

*for* (long long i = 0; i < N; i++)

    {

*for* (long long j = sumValue; j - valueV*[*i*]* >= 0; j--)

            dp*[*j*]* = *min*(dp*[*j*]*, dp*[*j - valueV*[*i*]]* + weightV*[*i*]*);

    }

    long long ans;

*for* (ans = sumValue; ans >= 0; ans--)

*if* (dp*[*ans*]* <= W)

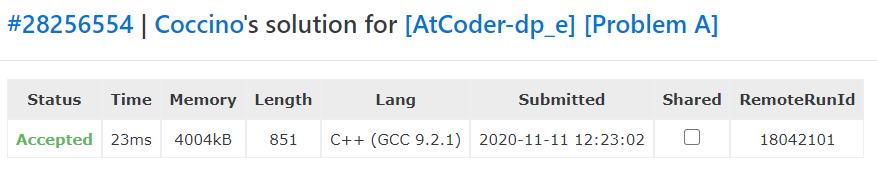
*break*;

    cout *<<* ans *<<* *endl*;

*return* 0;

}

四　结果



Lab2-B Beat That Monster!!!

AtCoder – abc153\_e

一　题目

Description

Ibis is fighting with a monster.

The *health* of the monster is HH.

Ibis can cast NN kinds of spells. Casting the ii-th spell decreases the monster's health by AiAi, at the cost of BiBi Magic Points.

The same spell can be cast multiple times. There is no way other than spells to decrease the monster's health.

Ibis wins when the health of the monster becomes 00 or below.

Find the minimum total Magic Points that have to be consumed before winning.

Constraints

1≤H≤1041≤H≤104

1≤N≤1031≤N≤103

1≤Ai≤1041≤Ai≤104

1≤Bi≤1041≤Bi≤104

All values in input are integers.

Input

Input is given from Standard Input in the following format:

HH NN

A1A1 B1B1

::

ANAN BNBN

Output

Print the minimum total Magic Points that have to be consumed before winning.

Sample Input 1

9 3

8 3

4 2

2 1

Sample Output 1

4

First, let us cast the first spell to decrease the monster's health by 88, at the cost of 33 Magic Points. The monster's health is now 11.

Then, cast the third spell to decrease the monster's health by 22, at the cost of 11 Magic Point. The monster's health is now −1−1.

In this way, we can win at the total cost of 44 Magic Points.

Sample Input 2

100 6

1 1

2 3

3 9

4 27

5 81

6 243

Sample Output 2

100

It is optimal to cast the first spell 100100 times.

Sample Input 3

9999 10

540 7550

691 9680

700 9790

510 7150

415 5818

551 7712

587 8227

619 8671

588 8228

176 2461

Sample Output 3

139815

二　算法思想

算法：动态规划

其中i≥0，j≥0，i为物品编号，dp[j]为对怪物造成伤害为j时最小消耗，dp[0] = 0。

遍历所有魔法，每次更新所有dp[0]~dp[HP+max(damage)]的值，dp[HP]~dp[HP+max(damage)]中最小的值就是能够使怪物死亡消耗的最小的魔力值。

三　代码

#*include* <algorithm>

#*include* <iostream>

#*include* <vector>

#*define* *INF* 0x3fffffff

using namespace std;

int *main*()

{

    ios::*sync\_with\_stdio*(false);

    long long H, N;//*怪物血量，魔法容量*

    cin *>>* H *>>* N;

    vector<long long> *damageV*(N);

    vector<long long> *costV*(N);

//*输入魔法伤害和消耗*

*for* (long long i = 0; i < N; i++)

        cin *>>* damageV*[*i*]* *>>* costV*[*i*]*;

    long long maxDPH = *\*max\_element*(damageV.*begin*(), damageV.*end*());

    vector<long long> *dp*(H + maxDPH + 1, *INF*);

    dp*[*0*]* = 0;

//*对于种魔法，更新使怪物死亡消耗的最小魔力值*

*for* (long long i = 0; i < N; i++)

    {

*for* (long long j = damageV*[*i*]*; j <= H + maxDPH; j++)

            dp*[*j*]* = *min*(dp*[*j*]*, dp*[*j - damageV*[*i*]]* + costV*[*i*]*);

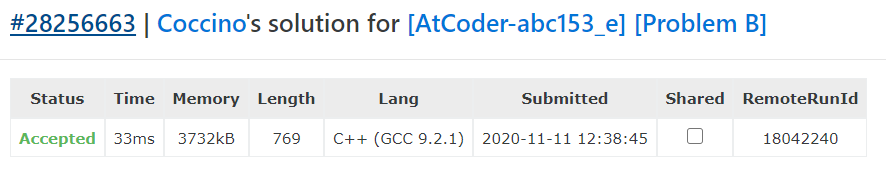
    }

    cout *<<* *\*min\_element*(dp.*begin*() *+* H, dp.*end*()) *<<* *endl*;

*return* 0;

}

四　结果



Lab2-C Merge Slimes!!!

AtCoder – dp\_n

一　题目

Description

There are NN slimes lining up in a row. Initially, the ii-th slime from the left has a size of aiai.

Taro is trying to combine all the slimes into a larger slime. He will perform the following operation repeatedly until there is only one slime:

Choose two adjacent slimes, and combine them into a new slime. The new slime has a size of x+yx+y, where xx and yy are the sizes of the slimes before combining them. Here, a cost of x+yx+y is incurred. The positional relationship of the slimes does not change while combining slimes.

Find the minimum possible total cost incurred.

Constraints

All values in input are integers.

2≤N≤4002≤N≤400

1≤ai≤109

Input

Input is given from Standard Input in the following format:

NN

a1a1 a2a2 …… aNaN

Output

Print the minimum possible total cost incurred.

Sample Input 1

4

10 20 30 40

Sample Output 1

190

Taro should do as follows (slimes being combined are shown in bold):

(10, 20, 30, 40) → (30, 30, 40)

(30, 30, 40) → (60, 40)

(60, 40) → (100)

Sample Input 2

5

10 10 10 10 10

Sample Output 2

120

Taro should do, for example, as follows:

(10, 10, 10, 10, 10) → (20, 10, 10, 10)

(20, 10, 10, 10) → (20, 20, 10)

(20, 20, 10) → (20, 30)

(20, 30) → (50)

Sample Input 3

3

1000000000 1000000000 1000000000

Sample Output 3

5000000000

The answer may not fit into a 32-bit integer type.

Sample Input 4

6

7 6 8 6 1 1

Sample Output 4

68

Taro should do, for example, as follows:

(7, 6, 8, 6, 1, 1) → (7, 6, 8, 6, 2)

(7, 6, 8, 6, 2) → (7, 6, 8, 8)

(7, 6, 8, 8) → (13, 8, 8)

(13, 8, 8) → (13, 16)

(13, 16) → (29)

二　算法思想

算法：动态规划

其中i≥0，k>j≥i，k≥0，i，j，k为史莱姆编号，dp[i][k]为合并第i到k个史莱姆的最小消耗，dp[i][i] = 0。

对于规模为n\*n的dp数组，采用由左到右、由下到上的顺序更新，即

for (k = 0; k < N; k++)

    for (i = k - 1; i >= 0; i--)

        dp[i][k] =

三　代码

#*include* <algorithm>

#*include* <iostream>

#*include* <vector>

#*define* *ll* long long

#*define* *INF* 0x7fffffffffffffff

using namespace std;

*ll* *sum*(vector<*ll*>::iterator beg, vector<*ll*>::iterator end)

{

*ll* ans = 0;

*while* (beg *<* end)

    {

        ans += *\**beg;

        beg*++*;

    }

*return* ans;

}

int *main*()

{

    ios::*sync\_with\_stdio*(false);

    int N;

    cin *>>* N;

    vector<*ll*> *slimes*(N);//*各史莱姆初始大小*

*for* (*ll* i = 0; i < N; i++)

        cin *>>* slimes*[*i*]*;

    vector<vector<*ll*>> *dp*(N, *vector*<*ll*>(N, *INF*));//*动态规划，dp[i][j]表示合并从[i,j]的史莱姆的最小消耗*

//*将对角线上的小号初始化为1*

*for* (*ll* i = 0; i < N; i++)

        dp*[*i*][*i*]* = 0;

*for* (*ll* k = 0; k < N; k++)

    {

*for* (*ll* i = k - 1; i >= 0; i--)

        {

*ll* minCost = *INF*;

*for* (*ll* j = i; j < k; j++)

            {

                minCost = *min*(minCost, dp*[*i*][*j*]* + dp*[*j + 1*][*k*]* + *sum*(slimes.*begin*() *+* i, slimes.*begin*() *+* k *+* 1));

            }

            dp*[*i*][*k*]* = minCost;

        }

    }

    cout *<<* dp*[*0*][*N - 1*]* *<<* *endl*;

*return* 0;

}

四　结果



Lab2-D Happy TSP

AtCoder – abc180\_e

一　题目

Description

In a three-dimensional space, there are NN cities: City 11 through City NN. City ii is at point (Xi,Yi,Zi)(Xi,Yi,Zi).

The cost it takes to travel from a city at point (a,b,c)(a,b,c) to a city at point (p,q,r)(p,q,r) is |p−a|+|q−b|+max(0,r−c)|p−a|+|q−b|+max(0,r−c).

Find the minimum total cost it takes to start at City 11, visit all other cities at least once, and return to City 11.

Constraints

2≤N≤172≤N≤17

−106≤Xi,Yi,Zi≤106−106≤Xi,Yi,Zi≤106

No two cities are at the same point.

All values in input are integers.

Input

Input is given from Standard Input in the following format:

NN

X1X1 Y1Y1 Z1Z1

⋮⋮

XNXN YNYN ZNZN

Output

Print the minimum total cost it takes to start at City 11, visit all other cities at least once, and return to City 11.

Sample Input 1

2

0 0 0

1 2 3

Sample Output 1

9

The cost it takes to travel from City 11 to City 22 is |1−0|+|2−0|+max(0,3−0)=6|1−0|+|2−0|+max(0,3−0)=6.

The cost it takes to travel from City 22 to City 11 is |0−1|+|0−2|+max(0,0−3)=3|0−1|+|0−2|+max(0,0−3)=3.

Thus, the total cost will be 99.

Sample Input 2

3

0 0 0

1 1 1

-1 -1 -1

Sample Output 2

10

For example, we can visit the cities in the order 11, 22, 11, 33, 11 to make the total cost 1010. Note that we can come back to City 11 on the way.

Sample Input 3

17

14142 13562 373095

-17320 508075 68877

223606 -79774 9979

-24494 -89742 783178

26457 513110 -64591

-282842 7124 -74619

31622 -77660 -168379

-33166 -24790 -3554

346410 16151 37755

-36055 51275 463989

37416 -573867 73941

-3872 -983346 207417

412310 56256 -17661

-42426 40687 -119285

43588 -989435 -40674

-447213 -59549 -99579

45825 7569 45584

Sample Output 3

6519344

二　算法思想

算法：动态规划

dp[i][j]表示由起点出发，到达状态i，目前处于城市j所需花费。

状态：长度为N的二进制序列，第k位为1表示城市k已访问，为0表示未访问。

按从左到右、从下到上的顺序更新规模为(1<<N)\*N的dp数组，即

    for (i = 1; i < (1 << N); i++)

            for (j = 0; j < N; j++)

                    if (((i >> j) & 1) == 1)

                            for (k = 0; k < N; k++)

                                    dp[i][j] = min(dp[i][j], dp[i - (1 << j)][k] + distance(k, j));

三　代码

#*include* <algorithm>

#*include* <iostream>

#*include* <vector>

#*define* *ll* long long

#*define* *INF* 0x7fffffffffffff

using namespace std;

struct site

{

*ll* x, y, z;

};

vector<site> siteV;//*各城市坐标*

*ll* *distance*(*ll* a, *ll* b)//*返回由a到达b的cost*

{

*return* *abs*(siteV*[*a*]*.x - siteV*[*b*]*.x) + *abs*(siteV*[*a*]*.y - siteV*[*b*]*.y) + *max*(siteV*[*b*]*.z - siteV*[*a*]*.z, (long long)0);

}

int *main*()

{

    int N;

    cin *>>* N;

    siteV.*insert*(siteV.*begin*(), N, *site*());

*for* (int i = 0; i < N; i++)

        cin *>>* siteV*[*i*]*.x *>>* siteV*[*i*]*.y *>>* siteV*[*i*]*.z;

//*dp[i][j]表示由起点出发，到达状态i，目前处于城市j所需花费*

//*状态：长度为N的二进制序列，第k位为1表示城市k已访问，为0表示未访问*

    vector<vector<*ll*>> *dp*(1 << N, *vector*<*ll*>(N, *INF*));

//*由未访问任何城市的状态访问起点花费为0*

    dp*[*0*][*0*]* = 0;

*for* (*ll* i = 1; i < (1 << N); i++)

    {

*for* (*ll* j = 0; j < N; j++)

        {

*if* (((i >> j) & 1) == 1)

            {

*for* (*ll* k = 0; k < N; k++)//*确定上一次访问的城市k，即从哪种状态转换到当前状态可以使总cost最小*

                {

                    dp*[*i*][*j*]* = *min*(dp*[*i*][*j*]*, dp*[*i - (1 << j)*][*k*]* + *distance*(k, j));

                }

            }

        }

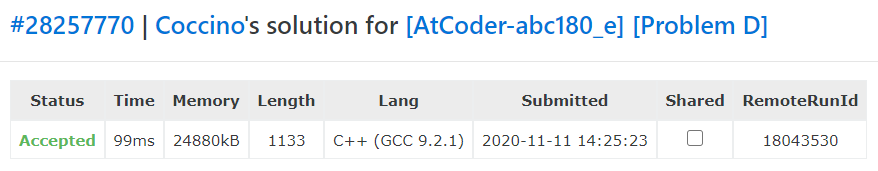
    }

    cout *<<* dp*[*(1 << N) - 1*][*0*]* *<<* *endl*;

*return* 0;

}

四　结果



Lab2-E Crush Proper Gems

AtCoder – arc085\_c

一　题目

Description

We have NN gemstones labeled 11 through NN.

You can perform the following operation any number of times (possibly zero).

Select a positive integer xx, and smash all the gems labeled with multiples of xx.

Then, for each ii, if the gem labeled ii remains without getting smashed, you will receive aiai yen (the currency of Japan). However, aiai may be negative, in which case you will be charged money.

By optimally performing the operation, how much yen can you earn?

Constraints

All input values are integers.

1≤N≤1001≤N≤100

|ai|≤109

Input

Input is given from Standard Input in the following format:

NN

a1a1 a2a2 ...... aNaN

Output

Print the maximum amount of money that can be earned.

Sample Input 1

6

1 2 -6 4 5 3

Sample Output 1

12

It is optimal to smash Gem 33 and 66.

Sample Input 2

6

100 -100 -100 -100 100 -100

Sample Output 2

200

Sample Input 3

5

-1 -2 -3 -4 -5

Sample Output 3

0

It is optimal to smash all the gems.

Sample Input 4

2

-1000 100000

Sample Output 4

99000

二　算法思想

建模：网络流

将源点与所有价值为负的宝石相连，将所有价值为正的宝石与汇点相连，边的容量为宝石价值绝对值；将所有价值为负的宝石和所有编号是该宝石整数倍的价值为正的宝石相连，边的容量为无穷。该网络流图的最小割可以划分应砸碎的宝石和应保留的宝石。

算法：FF算法

使用DFS寻找增广路径，根据增广路径的瓶颈容量更新增广路径上边的残存容量，重复寻找增广路径直至没有增广路径存在。此时残差图上所有可由源点到达的节点就是最小割。

三　代码

#*include* <algorithm>

#*include* <iostream>

#*include* <queue>

#*include* <vector>

#*define* *ll* long long

#*define* *INF* 0x7fffffffff

using namespace std;

//*用邻接矩阵*

struct pipe

{

*ll* posC;//*正向残存容量*

*ll* negC;//*反向残存容量*

};

*ll* N;

vector<vector<*ll*>> residualPlot;//*残差图*

vector<*ll*> augPath;//*增广路径*

vector<bool> visited;

int *DFS*(*ll* curNode)

{

    augPath.*push\_back*(curNode);

    visited*[*curNode*]* *=* true;

*if* (curNode == N + 1)

*return* 1;

*for* (*ll* i = 0; i < N + 2; i++)

    {

//*若i未被访问且存在由当前节点到n的残存边*

*if* (!visited*[*i*]* && residualPlot*[*curNode*][*i*]* > 0)

        {

*if* (*DFS*(i) != 0)//*若找到路径，则结束DFS*

*return* 1;

        }

    }

//*便利所有邻边，无增广路径，恢复现场，返回*

    augPath.*pop\_back*();

    visited*[*curNode*]* *=* false;

*return* 0;

}

int *main*()

{

    cin *>>* N;

//*输入宝石信息*

    vector<*ll*> *gems*(N + 1);

*for* (*ll* i = 1; i <= N; i++)

        cin *>>* gems*[*i*]*;

//*初始化残差图邻接矩阵，所有边容量置为0*

//*0为源点，1-N为宝石，N+1为汇点*

    residualPlot.*insert*(residualPlot.*begin*(), N + 2, *vector*<*ll*>(N + 2, 0));

*for* (*ll* i = 1; i <= N; i++)

    {

//*若宝石价值为负，则添加由源点到宝石的边，容量为宝石价值绝对值*

*if* (gems*[*i*]* < 0)

        {

            residualPlot*[*0*][*i*]* = -gems*[*i*]*;

//*添加由负宝石到相关正宝石的边，容量为INF*

*for* (*ll* j = 2 \* i; j <= N; j += i)

            {

*if* (gems*[*j*]* > 0)

                    residualPlot*[*i*][*j*]* = *INF*;

            }

        }

//*否则，添加宝石到汇点的边，容量为宝石价值绝对值*

*else*

            residualPlot*[*i*][*N + 1*]* = gems*[*i*]*;

    }

//*使用FF算法求最大流、最小割*

*while* (true)

    {

//*求增广路径，保存在augPath中*

        visited.*insert*(visited.*begin*(), N + 2, false);

*if* (*DFS*(0) == 0)

*break*;

*ll* bottleCapacity = *INF*;

*for* (*ll* i = 0; i < augPath.*size*() - 1; i++)

            bottleCapacity = *min*(bottleCapacity, residualPlot*[*augPath*[*i*]][*augPath*[*i + 1*]]*);

*for* (*ll* i = 0; i < augPath.*size*() - 1; i++)

        {

            residualPlot*[*augPath*[*i*]][*augPath*[*i + 1*]]* -= bottleCapacity;

            residualPlot*[*augPath*[*i + 1*]][*augPath*[*i*]]* += bottleCapacity;

        }

        augPath.*clear*();

        visited.*clear*();

    }

*ll* total = 0;//*总奖励*

*for* (*ll* i = 1; i <= N; i++)

        total += gems*[*i*]*;

//*使用BFS求最小割，在退出FF、算法前visited已初始化，直接使用*

    queue<*ll*> bfs;

    bfs.*push*(0);

    visited*[*0*]* *=* true;

*while* (!bfs.*empty*())

    {

*ll* cur = bfs.*front*();

        bfs.*pop*();

*for* (*ll* next = 1; next <= N; next++)

        {

*if* (!visited*[*next*]* && residualPlot*[*cur*][*next*]* > 0)

            {

                bfs.*push*(next);

                visited*[*next*]* *=* true;

                total -= gems*[*next*]*;

            }

        }

    }

    cout *<<* total *<<* *endl*;

*return* 0;

}

四　结果

