

华中科技大学计算机科学与技术学院

算法分析与设计报告



专 业： 计算机科学与技术

班 级： 计算机 ACM1801 班

学 号： U201814757

姓 名： 夏 媛

成 绩：

指导教师： 何 琨

完成日期： 2020 年 12 月 15 日

目录

实验一	1
1.1 畅通工程	1
1.2 Jungle Roads	3
1.3 还是畅通工程	6
1.4 畅通工程再续	8
1.5 Truck History	11
实验二	14
2.1 My Huge Bag	14
2.2 Beat That Monster!!!	16
2.3 Merge Slimes!!!	18
2.4 Happy TSP	20
2.5 Crush Proper Gems	23
实验感想	26

实验一

1.1 畅通工程

1.1.1 题目

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？

输入：测试输入包含若干个测试用例，每个测试用例第 1 行给出两个正整数，分别是城镇数目 N 和道路数目 M ，随后 M 行对应 M 条道路，每行给出一对正整数，比如：A B 表示城镇 A 和城镇 B 有路。城镇从 1 到 N 编号。

输出：对于每个测试用例，输出最少还需要建设的道路数目。

1.1.2 算法思想

初始化时每个城镇都没有路，所以总共要连接的路的数量为 $N-1$ 个，在输入路的过程中，先用并查集判断，如果两个城镇确实不在一个集合，这条路就有用，两个城镇归为一个集合中，要连接的路减一。如果两个城镇之前就已经在一个集合里，增加的这条路就无用，要连接的路不变。

并查集的基本操作有 find（查询祖先节点）、union（合并）：

```
int find(int x){
    return father[x]==x?x:find(x);
}
void union(int x,int y){
    int fx = find(x);
    int fy = find(y);
    if(fx!=fy){
        father[fx] = fy;
    }
}
```

1.1.3 结果

Username	Prob	Result	Time (ms)	Mem (MB)	Lang
xiayuan233	A -	All			All
xiayuan2333	A	Accepted	140	1.8	C++

图 1.1 题 1.1 运行结果

1.1.4 代码

```
#include <iostream>
#include <stdio.h>
using namespace std;

int p[1008];
int ranknum[1008];

int find(int x){
    int r = x;
    while(p[r]!=r){
        r = p[r];
    }
    int i=x,j;
    //路径压缩
    while(i!=r){
        j=p[i];
        p[i]=r;
        i=j;
    }
    return r;
}

int join(int x,int y){
    int fx = find(x),fy=find(y);
    if(fx!=fy){
        if(ranknum[fx]>ranknum[fy]){
            p[fy]=fx;
            ranknum[fx]+=ranknum[fy];
        }else{
            p[fx]=fy;
            ranknum[fy]+=ranknum[fx];
        }
    }
    return 1;
}

else{
    return 0;
}
```

```

    }
}
int main(){
    int num_city,num_road;
    int a,b;
    int ans;
    while(scanf("%d",&num_city),num_city){
        scanf("%d",&num_road);
        //init
        for(int i=1;i<num_city+1;i++){
            p[i]=i;
            ranknum[i]=1;
        }
        ans = num_city-1;
        while(num_road){
            cin>>a>>b;
            if(join(a,b)){
                ans--;
            }
            num_road--;
        }
        cout<<ans<<endl;
    }
    return 0;
}

```

1.2 Jungle Roads

1.2.1 题目

给出一个村庄的连通道路网，每条道路有一份维护费用。要求找到维护费用之和最小的维持连接所有村庄的道路。

输入：测试输入包含若干个测试用例，每个测试用例第一行 n ($1 < n < 27$) 为村庄数目，村庄用字母 A~Z 表示。接下来 $n-1$ 行输入村庄 i 与字母序号比 i 大的村庄的连通关系。如：B 3 C 10 H 40 I 8，表示字母序号大于 B 与 B 连通的村庄有 3 个，村庄 B 和村庄之间道路维护费用是 10，B、H 间是 40，B、I 间是 8。

输出：确保村庄连接的最小维护费用。

1.2.2 算法思想

道路网即为带权无向图，维护费用视为边的权值。问题即找出图的最小生成树的。本题使用邻接矩阵存储图，使用 Prim 算法求最小生成树。

Prim 算法的基本思想为：

1. 选择一个点作为树的根节点，初始时树只有这一个节点
2. 选择图中距离这棵树最近但没有被树收录的一个顶点，将其加入树中，并且保证不构成回路。
3. 重复 2，将图中所有顶点一一收录到树中。

Prim 算法的伪代码为：

```
void Prim(int s){
    {s} -> MST;
    while(1){
        不属于 MST 且距离 MST 最小的顶点 -> v;
        if(v 不存在){
            算法结束;
        }
        MST+{v} -> MST;
        for(v 的每个邻接点 u){
            更新 u 到 MST 的最小距离;
        }
    }
}
```

1.2.3 结果

Username	Prob	Result	Time (ms)	Mem (MB)	Lang
xiayuan233	B -	All			All
xiayuan2333	B	Accepted	0	0.2	C++

图 1.2 题 1.2 运行结果

1.2.4 代码

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

using namespace std;
#define IFN 999999
//存储结构：邻接矩阵
int adj[30][30];
struct addedge{
    int adj_v;
    int edgenum;
}closeedge[30];
int mini_adjedge(int num_village){
    int tmp=IFN;
    int vertex=-1;
```

```

    for(int i=1;i<num_village;i++){
        if((closeedge[i].edgenum<tmp)&&(closeedge[i].edgenum!=0)){
            tmp = closeedge[i].edgenum;
            vertex=i;
        }
    }
    return vertex;
}

void Prim(int num_village){
    //init closeedge,s=1
    int ans = 0;
    closeedge[1].adj_v=0;
    closeedge[1].edgenum=0;
    for(int i=2;i<num_village;i++){
        closeedge[i].adj_v=1;
        closeedge[i].edgenum=adj[i][1];
    }
    //
    for(int k=2;k<num_village;k++){
        int vertex = mini_adjedge(num_village);
        if(vertex==-1){
            cout<<"not exit a MST";
            exit(1);
        }
        ans = ans + closeedge[vertex].edgenum;
        closeedge[vertex].adj_v=0;
        closeedge[vertex].edgenum=0;
        //init again
        for(int i=2;i<num_village;i++){
            if((closeedge[i].adj_v!=0)&&(closeedge[i].edgenum>adj[i][vertex])){
                closeedge[i].adj_v=vertex;
                closeedge[i].edgenum=adj[i][vertex];
            }
        }
    }
    cout<<ans<<endl;
    return;
}

int main(){
    int num_village;
    int weight,num_edge;
    char c;
    while(cin>>num_village,num_village){    //1<n<27

```

```

        for(int i=1;i<num_village+1;i++){
            for(int j=1;j<num_village+1;j++){
                adj[i][j]=IFN;
            }
        }
        int tmp = num_village;
        num_village--;
        while(num_village--){
            cin>>c>>num_edge;
            int y = (int)(c)-64;
            while(num_edge--){

                cin>>c>>weight;
                int z = (int)(c)-64;
                adj[z][y]=adj[y][z]=weight;
            }
        }
        //init over
        Prim(tmp+1);
    }
    return 0;
}

```

1.3 还是畅通工程

1.3.1 题目

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。

输入：测试输入包含若干测试用例。每个测试用例的第 1 行给出村庄数目 N （ < 100 ）；随后的 $N(N-1)/2$ 行对应村庄间的距离，每行给出一对正整数，分别是两个村庄的编号，以及此两村庄间的距离。村庄从 1 到 N 编号。

输出：使得所有村庄连通的最小的公路总长度。

1.3.2 算法思想

将道路长度视为边的权值，问题即求出图的最小生成树。算法同 1.2.2。

1.3.3 结果

Username	Prob	Result	Time (ms)	Mem (MB)	Lang
xiayuan233	C -	All			All
xiayuan2333	C	Accepted	998	1.8	C++

图 1.3 题 1.3 运行结果

1.3.4 代码

```

#include<iostream>
#include<stdio.h>

using namespace std;
#define IFN 999999
//存储结构：邻接矩阵
int adj[105][105];
struct addedge{
    int adj_v;
    int edgenum;
}closeedge[105];
int mini_adjedge(int num_village){
    int tmp=IFN;
    int vertex=-1;
    for(int i=1;i<num_village;i++){
        if((closeedge[i].edgenum<tmp)&&(closeedge[i].edgenum!=0)){
            tmp = closeedge[i].edgenum;
            vertex=i;
        }
    }
    return vertex;
}

void Prim(int num_village){
    //init closeedge,s=1
    int ans = 0;
    closeedge[1].adj_v=0;
    closeedge[1].edgenum=0;
    for(int i=2;i<num_village;i++){
        closeedge[i].adj_v=1;
        closeedge[i].edgenum=adj[i][1];
    }
    //
    for(int k=2;k<num_village;k++){
        int vertex = mini_adjedge(num_village);
        if(vertex==-1){

```

```

        cout<<"not exit a MST";
        exit(1);
    }
    ans = ans + closeedge[vertex].edgenum;
    closeedge[vertex].adj_v=0;
    closeedge[vertex].edgenum=0;
    //init again
    for(int i=2;i<num_village;i++){
        if((closeedge[i].adj_v!=0)&&(closeedge[i].edgenum>adj[i][vertex])){
            closeedge[i].adj_v=vertex;
            closeedge[i].edgenum=adj[i][vertex];
        }
    }
}
cout<<ans<<endl;
return;
}

int main(){
    int num_village;
    int weight,a,b;
    while(cin>>num_village,num_village){    //1<n<27
        for(int i=1;i<num_village+1;i++){
            for(int j=1;j<num_village+1;j++){
                adj[i][j]=IFN;
            }
        }
        int num_dis = (num_village-1)*(num_village)/2;
        while(num_dis--){
            cin>>a>>b>>weight;
            adj[a][b]=adj[b][a]=weight;
        }
        //init over
        Prim(num_village+1);
    }
    return 0;
}

```

1.4 畅通工程再续

1.4.1 题目

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。政府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。

输入：输入包括多组数据。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。

输出：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!”。

1.4.2 算法思想

建模为图，对于 2 个小岛之间的距离小于 10 米，或大于 1000 米的情况，将两个点之间的距离设置为足够大，确保不会被选中。然后可以使用 Prim 算法求最小生成树，算法同 1.2.2。

1.4.3 结果

Username	Prob	Result	Time (ms)	Mem (MB)	Lang
xiayuan233	D -	All			All
xiayuan2333	D	Accepted	46	1.9	C++

图 1.4 题 1.4 运行结果

1.4.4 代码

```
#include<iostream>
#include<stdio.h>
#include<math.h>
using namespace std;
#define IFN 999999.0

//存储结构：邻接矩阵
double adj[105][105];
struct addedge{
    int adj_v;
    float edgenum;
}closeedge[105];
struct position{
    int x;
    int y;
}ver_position[105];
int mini_adjedge(int num_village){
    float tmp=IFN;
    int vertex=-1;
    for(int i=2;i<num_village;i++){
```

```

        if((closeedge[i].edgenum<tmp)&&(closeedge[i].edgenum!=0)){
            tmp = closeedge[i].edgenum;
            vertex=i;
        }
    }
    return vertex;
}

void Prim(int num_village){
    //init closeedge,s=1
    double ans = 0;
    closeedge[1].adj_v=0;
    closeedge[1].edgenum=0;
    for(int i=2;i<num_village;i++){
        closeedge[i].adj_v=1;
        closeedge[i].edgenum=adj[i][1];
    }
    //
    int count =1;
    for(int k=2;k<num_village;k++){
        double min = IFN;
        int vertex = 0;
        for(int i=1;i<num_village;i++){
            if(min>closeedge[i].edgenum&&closeedge[i].edgenum){
                min = closeedge[i].edgenum;
                vertex = i;
            }
        }
        if(vertex){
            ans = ans + closeedge[vertex].edgenum;
            closeedge[vertex].adj_v=0;
            closeedge[vertex].edgenum = 0;
            count++;
        }
        //init again
        for(int i=2;i<num_village;i++){
            if((closeedge[i].edgenum>adj[i][vertex])&&closeedge[i].edgenum){
                closeedge[i].adj_v=vertex;
                closeedge[i].edgenum=adj[i][vertex];
            }
        }
    }
    if(count==num_village-1){
        printf("%.1lf\n",ans*100);
    }else{

```

```

        cout<<"oh!"<<endl;
    }
    return;
}

int main(){
    int num_island,num_case;
    int a,b;
    cin>>num_case;
    while(num_case--){
        cin>>num_island;
        //init
        for(int i=1;i<num_island+1;i++){
            cin>>ver_position[i].x>>ver_position[i].y;
        }
        for(int i=1;i<num_island+1;i++){
            for(int j=1;j<num_island+1;j++){
                if(i==j){
                    adj[i][j]=0;
                }else{
                    int x1 = ver_position[i].x,x2 = ver_position[j].x;
                    int y1 = ver_position[i].y,y2 = ver_position[j].y;
                    double dist = sqrt(1.0*(x1-x2)*(x1-x2)+(y1-y2)*(y1-y2));
                    if(dist<10.0||dist>1000.0){
                        adj[i][j]=adj[j][i]=IFN;
                    }else{
                        adj[i][j]=adj[j][i]=dist;
                    }
                }
            }
        }
        Prim(num_island+1);
    }
    return 0;
}

```

1.5 Truck History

1.5.1 题目

给出一系列卡车类型代码，每个代码是由 7 个小写字母组成的字符串。卡车类型的距离定义为卡车类型代码中带有不同字母的位置数，假定每种卡车类型都恰好衍生自其他卡车类型（第一个卡车类型并非衍生自任何其他类型）。将派生

计划的质量定义为 $\frac{1}{\sum_{(t_o,t_d)} d(t_o,t_d)}$ ，其中 t_o 是原始类型， t_d 是从其派生的类型。

输入：输入包含几个测试用例。输入第一行为卡车类型的数量 N ($2 \leq N \leq 2000$)，接下来 N 行每行包含一个开车类型代码（这 N 个代码中没有重复的）

输出：最小的派生计划。

1.5.2 算法思想

使用邻接矩阵记录任意两个字符串之间的距离，建模为图，每个顶点对应一种卡车类型代码，任意两个顶点间都有边相连，边权为卡车类型的距离。使用 Prim 算法求最小生成树，算法思想同 1.2.2

1.5.3 结果

Username	Prob	Result	Time (ms)	Mem (MB)	Lang
<input type="text" value="xiayuan233"/>	<input type="text" value="E -"/>	<input type="text" value="All"/>			<input type="text" value="All"/>
xiayuan2333	E	Accepted	688	15.6	C++

图 1.5 题 1.5 运算结果

1.5.4 代码

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

#define INF 999999
using namespace std;
char derive[2005][8];
int adjMAT[2005][2005];
int close_edge[2005];
void Prim(int num_type){
    int Q = 0;
    //init,s = 0
    close_edge[0]=0;
    for(int i=1;i<num_type;i++){
        close_edge[i]=adjMAT[i][0];
    }
    int count=1;
    for(int i=0;i<num_type-1;i++){
        int min = INF,vertex = -1;
        //find a min closeedge
        for(int j=0;j<num_type;j++){
            if((min>close_edge[j])&&close_edge[j]){
```

```

        min = close_edge[j];
        vertex = j;
    }
}
if(vertex!=-1){
    count++;
    Q = Q + close_edge[vertex];
    close_edge[vertex]=0;
    //update close_edge
    for(int j=0;j<num_type;j++){
        if(adjMAT[j][vertex]<close_edge[j]&&close_edge[j]){
            close_edge[j]=adjMAT[j][vertex];
        }
    }
}
}
cout<<"The highest possible quality is 1/"<<Q<<'. '<<endl;
}
int dist(char a[],char b[]){
    int res=0;
    for(int i=0;i<7;i++){
        if(a[i]!=b[i]){
            res++;
        }
    }
    return res;
}
int main(){
    int num_type;
    while(cin>>num_type,num_type){
        for(int i=0;i<num_type;i++){
            cin>>derive[i];
        }
        //init adjMAT:
        for(int i=0;i<num_type;i++){
            for(int j=i;j<num_type;j++){
                if(i==j){
                    adjMAT[i][j]=0;
                }else{
                    adjMAT[i][j]=adjMAT[j][i]=dist(derive[i],derive[j]);
                }
            }
        }
    }
    Prim(num_type);
}

```

```

    }
    return 0;

}

```

实验二

2.1 My Huge Bag

2.1.1 题目

有 $N(1 \leq N \leq 100)$ 个物品，每个物品 i 的重量为 $w_i(1 \leq w_i \leq W)$ ，价值为 $v_i(1 \leq v_i \leq 1000)$ 。背包的最大承重是 $W(1 \leq W \leq 10^9)$ ，在这 N 个物品中选择装入背包，使得总的价值最大。

输入：第一行输入 N 和 W ，接下来 N 行，每行一输入对整数： w_i 和 v_i 。

输出：背包能装下的最大价值之和。

2.1.2 算法思想

典型的 0-1 背包问题，需要用动态规划来求解。简单的背包问题的数组 $dp[i][j]$ 表示，最大承重 j 的背包选择前 $1 \sim i$ 个物品装入的最大价值总和，但这样需要开数组 $dp[N][W]$ ，显然内存会爆炸。所有需要使用一种新的动态规划的数组。

定义 $dp[i][j]$ 表示：选择前 $1 \sim i$ 个物品获取总价值为 j 所需要的最小背包容量，这样开辟的数组 $dp[N][\sum v_i]$ ，是可以满足的，状态转移方程为：

$$dp[i][j] = \min(dp[i-1][j], dp[i-1][j-v_i] + w_i)$$

根据状态转移方程即可求出 $dp[N][\sum v_i]$ ，即结果。主要代码为：

```

for(int i=1;i<=N;i++){
    for(int j=1;j<=sum_value;j++){ //从 1 到所有物品的价值总和遍历 j
        if(j<v[i]){ //j 小于当前物品的价值时
            dp[i][j] = dp[i-1][j];
        }else{
            dp[i][j] = min(dp[i-1][j],dp[i-1][j-v[i]]+w[i]);
        }
    }
}
}

```


空间优化：可以将 i 维度去掉，只留下 $dp[x]$ ， $dp[x]$ 表示获取总价值为 x 的物品所需要的最小背包容量。转移方程为：

$$dp[x] = \min\{dp[x - v_i] + w_i\}$$

主要代码为：

```
for(int i=1;i<=N;i++){
    for(int j=sum_value;j>=v[i];j--){ //从所有物品的价值总和到 1 遍历 j
        dp[j]=min(dp[j],dp[j-v[i]]+w[i]);
    }
}
```

2.1.3 结果

Username	Prob	Result	Time (ms)	Mem (MB)	Lang
<input type="text"/>	<input type="text" value="All"/>	<input type="text" value="All"/>			<input type="text" value="All"/>
xiayuan2333	A	Accepted	67	82.7	C++

图 2.1 题 2.1 运算结果

2.1.4 代码

```
#include<iostream>
#define MIN(A,B) ((A<B)?(A):(B))
#define MAX(A,B) ((A>B)?(A):(B))

using namespace std;
typedef long long ll;

#define INF 999999999 //WA 排除：INF>1e9
ll minw[105][100030];

int main(){
    ll sum_Weight,num_items,w,v; //过大
    scanf("%lld %lld",&num_items,&sum_Weight);
    ll i=1;
    ll tmp = num_items;
    for(int j=0;j<1e5+30;j++){
        minw[0][j]=INF;
    }
    minw[0][0]=0;
    //minw[i][j] 1~i 个背包达到 j 的价值所需的最小重量
    while(num_items--){
        scanf("%lld %lld",&w,&v);
        //空间优化为一维 minw[j]
        /*
```

```

for i=1->NUM:
    for j=max_value->0:
        minw[j]=MIN(minw[j-1],minw[j-1]+w[i])
    */
for(int j=0;j<1e5+30;j++){
    if(j<v){
        minw[i][j]=minw[i-1][j];
    }else{
        minw[i][j]=MIN(minw[i-1][j],minw[i-1][j-v]+w);
    }
}
i++;
}
if(sum_Weight==0||tmp==0){
    cout<<0;
    return 0;
}
for(int i=1e5+20;i>0;i--){
    if(minw[tmp][i]<=sum_Weight){
        printf("%d",i);
        return 0;
    }
}
printf("%d",0);
return 0;
}

```

2.2 Beat That Monster!!!

2.2.1 题目

Ibis 在打怪兽，怪兽的生命值是 H ($1 \leq H \leq 10^4$)。Ibis 能够施放 N ($1 \leq N \leq 10^3$) 种诅咒，第 i 种诅咒能减少怪物 A_i ($1 \leq A_i \leq 10^4$) 的生命值，并消耗 B_i ($1 \leq B_i \leq 10^4$) 的魔法。同一诅咒可以被释放多次，当怪物生命值小于等于 0 时 Ibis 获胜，求打败怪兽需要的最少的魔法值。

输入：输入的第一行是 H 和 N ，接下来的 N 行，每行输入一对整数 A 和 B 。

输出：打败怪兽所需最少的魔法值

2.2.2 算法思想

由于每种咒语可以使用多次，所以是一个完全背包问题，设 $dp[i][j]$ 表示：使

用 $1 \sim i$ 的咒语对怪物造成 i 点伤害需要的最小的魔法值。状态转移方程为：

$$dp[i][j] = \min\{dp[i-1][j-k*A[i]] + k*B[i]\}, 0 \leq k \leq \frac{j}{A[i]}$$

然后使用三重循环计算，结果即 $dp[N][H]$

时间和空间优化，去掉 $dp[i][j]$ 的 i 维， $dp[x]$ 表示对怪物造成 x 点伤害所需的最小魔法值。和朴素的 0-1 背包不同，题目的“物品”可以使用多次，因此 dp 数组的上限 $Uppre = H + \max A[i]$ ，状态转移方程为：

$$dp[x + A[i]] = \min(dp[x] + B[i], dp[x])$$

问题的解即为 $dp[H]$ ，主要代码为：

```
for(int i=1;i<=N;i++){
    for(int v=1;v<=H+maxA;v++){
        if(v<=A[i]){
            f[v]=min(f[v],B[i]);
        }else{
            f[v]=min(f[v],f[v-A[i]]+B[i]);
        }
    }
}
```

2.2.3 结果

Username	Prob	Result	Time (ms)	Mem (MB)	Lang
xiayuan233	B -	All			All
xiayuan2333	B	Accepted	30	1.7	C++

图 2.2 题 2.2 计算结果

2.2.4 代码

```
#include<stdio.h>

#define NUM 10010
#define MIN(A,B) ((A)<(B)?(A):(B))
#define INF 999999999

int f[NUM];

int main(){
    int H,N;
    scanf("%d %d",&H,&N);
    f[0]=0;
```

```

for(int i=1;i<NUM;i++){
    f[i]=INF;
}
int A,B;
while(N--){
    scanf("%d %d",&A,&B);
    for(int v=1;v<=H+3;v++){
        if(v<=A){
            f[v]=MIN(f[v],B);
        }else{
            f[v]=MIN(f[v],f[v-A]+B);
        }
    }
}
printf("%d",f[H]);

return 0;
}

```

2.3 Merge Slimes!!!

2.3.1 题目

有 N ($2 \leq N \leq 400$) 只史莱姆, 第 i 只史莱姆的体积为 a_i ($1 \leq a_i \leq 10^9$)。Taro 试图将所有史莱姆合并为一只更大的史莱姆, 他将重复执行以下操作, 直到只有一个史莱姆: 选择两只相邻的史莱姆, 体积分别是 x 和 y , 则新合成的史莱姆的体积是 $x+y$, 费用也是 $x+y$ 。合并史莱姆时, 史莱姆之间的位置信息不会改变。

求将所有史莱姆合并的最小费用。

输入: 输入的第一行为 N , 接下来的 N 行输入 a_i 。

输出: 输入最小花费

2.3.2 算法思想

可以使用矩阵链乘法的思想来解决该问题, 设 $dp(l, r)$ 表示从第 l 个史莱姆合并到第 r 个史莱姆需要的最小花费, 状态转移方程为:

$$dp(l, r) = \begin{cases} 0 & l = r \\ \min_{l \leq k < r} \{ dp(l, k) + dp(k+1, r) + \sum_{i=l}^r a_i \} & l \neq r \end{cases}$$

问题的解即为 $dp(1, N)$ 。

2.3.3 结果

Username	Prob	Result	Time (ms)	Mem (MB)	Lang
xiayuan233	C -	All			All
xiayuan2333	C	Accepted	21	3	C++

图 2.3 题 2.3 运算结果

2.3.4 代码

```
#include<stdio.h>
//类似矩阵链乘法
//状态转移方向: dp[1][1]->dp[1][n]
#define N 400
#define INF 0x7fffffffffffffff
typedef long long ll;
ll dp[N+1][N+1];
ll A[N+1];
int main(){
    int num;    //num>=2
    scanf("%d",&num);
    int i=1;
    ll sum1i=0,sum1j=0;;
    for(int i=1;i<N+1;i++){
        dp[i][i]=0;
    }
    int tmp = num;
    while(num--){
        scanf("%lld",&A[i]);
        i++;
    }
    ll minbuf=INF;
    for(int j=1;j<=tmp;j++){
        sum1j = sum1j + A[j];
        for(int i=j;i>=1;i--){
            sum1i = sum1i + A[i];
            for(int k=i;k<j;k++){
                if((dp[i][k]+dp[k+1][j]+sum1i)<minbuf){
                    minbuf = dp[i][k]+dp[k+1][j]+sum1i;
                }
            }
        }
    }
}
```

```

        if(minbuf!=INF){
            dp[i][j]=minbuf;
            minbuf=INF;
        }
    }
    sum1i = 0;
}
printf("%lld",dp[1][tmp]);
return 0;
}

```

2.4 Happy TSP

2.4.1 题目

三维空间中有 $N(2 \leq N \leq 17)$ 个城市，第 i 个城市的坐标为 (X_i, Y_i, Z_i) ，其中 $-10^6 \leq X_i, Y_i, Z_i \leq 10^6$ 。城市编号从 1 到 N 。从坐标 (a, b, c) 的城市到坐标 (p, q, r) 的城市的费用为 $|p - a| + |q - b| + \max(0, r - c)$ 。从城市 1 出发，访问其他所有城市，最后回到城市 1，求所需要的最小花费。

输入：输入第一行为 N ，接下来的 N 行每行输入 3 个整数 X 、 Y 、 Z 。

输出：最小花费。

2.4.2 算法思想

使用邻接矩阵作为图的存储结构，对于任意两个城市，记录它们之间的距离。为了记录经过的城市，注意城市数小于 18，所以可以使用一个 `int` 型整数 s 表示， s 的二进制表示中第 i 位为 1 表示经过了第 i 个城市，方便起见，这里从 0 开始给城市编号。 $dp[i][S]$ 表示从 i 点出发，经过 S 回到源点 0 的最小花费。转移方程为：

$$dp[i][S] = \min\{dp[j][S - \{j\}] + dist[i][j] \mid j \in S\}$$

其中 $S - \{j\}$ 表示从 S 的城市记录中删掉 j ，具体可由位操作实现。算法首先要对 dp 数组初始化，当 $S=0$ 时，从任一点 i 出发回到源点 0 的花费为 $dist[i][0]$ 。伪代码如下：

```

for i=1 to N: //初始化
    dp[i][0] = dist[i][0];
for S=1 to 1<<(S-1): //罗列 S 的所有可能
    for i=0 to N-1: //遍历所有点
        if(i 在 S 内) continue;
        for k=1 to N-1:

```

```

if(k 在 S 内)
    dp[i][S] = min(dp[i][S], dp[k][S-{k}] + dist[i][k]);

```

2.4.3 结果

Username	Prob	Result	Time (ms)	Mem (MB)	Lang
xiayuan233	D -	All			All
xiayuan2333	D	Accepted	54	9.1	C++

图 2.4 题 2.4 运算结果

2.4.4 代码

```

#include<stdio.h>
#include<math.h>
#define MIN(A,B) ((A)>(B)?(B):(A))
#define MAX(A,B) ((A)>(B)?(A):(B))
#define INF 0x7fffffffffffffff
typedef long long ll;
struct point{
    ll X,Y,Z;
}POINT[18];
int N;
ll adjMAT[18][18];
bool mask[18]; //min_dst 需要, 标记当前经过的点, 包括 s=0
int destination; //终点
ll cur_min;
int V; //min_dst 需要, 当前节点能用来构造最短路的, 这里面不包括 s=0
void min_dst(int cur, ll dis) //DFS 确定原点 to 终点的最短路
{
    if(dis >= cur_min){
        return;
    }
    if(cur == destination){
        cur_min = MIN(dis, cur_min);
        return;
    }
    for(int i=0; i<N; i++){
        if(!mask[i] && ((i==0) || ((V >> (i-1)) & 1))) {
            mask[i] = true;
            min_dst(i, dis + adjMAT[cur][i]);
            mask[i] = false;
        }
    }
}

```

```

    }
}
//不需要求最短路，测试集没有相关案例，所有点只需经过一次就好
int main(){
    scanf("%d",&N); //2<=N<=17
    ll dp[N][1<=(N-1)]; //dp 数组,s=0
    for(int i=0;i<N;i++){
        scanf("%lld %lld %lld",&(POINT[i].X),&(POINT[i].Y),&(POINT[i].Z));
    }
    //init adjmat
    for(int i=0;i<N;i++){
        for(int j=0;j<N;j++){
            adjMAT[i][j]=fabs(POINT[j].X-POINT[i].X)+fabs(POINT[j].Y-
POINT[i].Y)+MAX(0,POINT[j].Z-POINT[i].Z);
        }
    }
    //初始化第一列
    dp[0][0]=0;
    V = ~0;
    /*for(int i=1;i<N;i++){
        mask[i]=true;
        cur_min=INF;
        destination=0;
        min_dst(i,0);
        dp[i][0]=cur_min; //全图中，i 点到达 s=0 的最短路径
        mask[i]=false;
    }
    */
    for(int i=1;i<N;i++){
        dp[i][0]=adjMAT[i][0];
    }
    //dp[i][S] 从 i 出发，经过 S 回到 s=0 的最短路径
    for(int j=1;j<(1<=(N-1));j++){
        for(int i=0;i<N;i++){
            if((j>>i-1)&1){ //i is in j
                continue;
            }
            dp[i][j]=INF;
            for(int k=1;k<N;k++){
                if((j>>(k-1))&1){ //k is in j
                    if(dp[i][j]>dp[k][j^(1<=(k-1))]+adjMAT[i][k]){
                        dp[i][j]=dp[k][j^(1<=(k-1))]+adjMAT[i][k];
                    }
                }
            }
        }
    }
}

```



```

    }
}
}
printf("%lld", dp[0][(1 << (N-1)) - 1]);
return 0;
}

```

2.5 Crush Proper Gems

2.5.1 题目

有 N ($1 \leq N \leq 100$) 个宝石，标号从 1 到 N 。可以执行多次以下操作（也可以为 0 次）：选择整数 x ，然后将所有标号为 x 的倍数的宝石粉碎。最后，对于宝石 i ，如果没有被粉碎，将收到 a_i 的奖励（ a_i 可能是负数， $|a_i| \leq 10^9$ ）。问题：执行操作，最多能得到多少奖励？

输入：输入第一行为 N ，接下来输入 N 的 a 。

输出：能得到的最大奖励。

2.5.2 算法思想

使用最大权闭合子图的思想建模，若 $k \times x$ 号宝石未被粉碎，则 x 号宝石一定未被粉碎，即 $k \times x$ 选中则 x 必须也被选中。所以建模如下：

图共有 $N+2$ 个顶点，除了 N 个宝石顶点外添加源点 s 和汇点 t ，若宝石顶点 i 对应宝石的奖励为 a_i ，若 a_i 大于 0，则从源点 s 连向顶点 i ，权值为 a_i ；若 a_i 小于 0，则从顶点 i 连向汇点 t ，权值为 $-a_i$ 。对于任意连个宝石顶点 i 、 j ，若 i 是 j 的倍数，则从顶点 i 连向顶点 j ，权值为无限大。

接下来使用网络流算法求出最小割，问题的答案即 $\sum_{a_i \geq 0} a_i - \text{最大流}$

2.5.3 结果

Username	Prob	Result	Time (ms)	Mem (MB)	Lang
xiayuan233	E -	All			All
xiayuan2333	E	Accepted	7	3.3	C++

图 1.5 题 1.5 运算结果

2.5.4 代码

```

#include<stdio.h>
#include<math.h>

```

```

#define MIN(A,B) ((A)>(B)?(B):(A))
#define MAX(A,B) ((A)>(B)?(A):(B))
#define INF 0x7fffffff
typedef long long ll;
struct point{
    ll X,Y,Z;
}POINT[18];
int N;
ll adjMAT[18][18];
bool mask[18]; //min_dst 需要, 标记当前经过的点,包括 s=0
int destination; //终点
ll cur_min;
int V; //min_dst 需要, 当前节点能用来构造最短路的,这里面不包括 s=0
void min_dst(int cur,ll dis) //DFS 确定原点 to 终点的最短路
{
    if(dis>=cur_min){
        return;
    }
    if(cur==destination){
        cur_min=MIN(dis,cur_min);
        return;
    }
    for(int i=0;i<N;i++){
        if(!mask[i]&&((i==0)||((V>=(i-1)&1)))){
            mask[i]=true;
            min_dst(i,dis+adjMAT[cur][i]);
            mask[i]=false;
        }
    }
}

//不需要求最短路, 测试集没有相关案例, 所有点只需经过一次就好
int main(){
    scanf("%d",&N); //2<=N<=17
    ll dp[N][1<<(N-1)]; //dp 数组,s=0
    for(int i=0;i<N;i++){
        scanf("%lld %lld %lld",&(POINT[i].X),&(POINT[i].Y),&(POINT[i].Z));
    }
    //init adjmat
    for(int i=0;i<N;i++){
        for(int j=0;j<N;j++){
            adjMAT[i][j]=fabs(POINT[j].X-POINT[i].X)+fabs(POINT[j].Y-
            POINT[i].Y)+MAX(0,POINT[j].Z-POINT[i].Z);
        }
    }
}

```

```

//初始化第一列
dp[0][0]=0;
V = ~0;
/*for(int i=1;i<N;i++){
    mask[i]=true;
    cur_min=INF;
    destination=0;
    min_dst(i,0);
    dp[i][0]=cur_min;    //全图中，i 点到达 s=0 的最短路径
    mask[i]=false;
}
*/
for(int i=1;i<N;i++){
    dp[i][0]=adjMAT[i][0];
}
//dp[i][S] 从 i 出发，经过 S 回到 s=0 的最短路径
for(int j=1;j<(1<<(N-1));j++){
    for(int i=0;i<N;i++){
        if((j>>i-1)&1){ //i is in j
            continue;
        }
        dp[i][j]=INF;
        for(int k=1;k<N;k++){
            if((j>>(k-1))&1){ //k is in j
                if(dp[i][j]>dp[k][j^(1<<(k-1))]+adjMAT[i][k]){
                    dp[i][j]=dp[k][j^(1<<(k-1))]+adjMAT[i][k];
                }
            }
        }
    }
}
}
printf("%lld",dp[0][(1<<(N-1))-1]);
return 0;
}

```

实验感想

通过本次实验，让我认识到课堂上光光学习理论知识是不足的，使用程序设

计语言编写出来、对实际问题建模并选择算法同样也是关键。在本次实验中，我感受到了算法的魅力，也认识到了自己的不足并获得了一定的提高。

对于算法课，我觉得注重于学生们沟通讨论的开放式课堂非常好。但也希望老师能适当控制讲课速度，突出知识重点，如果能够使用 PPT 就更好了。