我的算法模板

动态规划部分

LCS 最长公共子序列

这里犯懒了，直接贴的是两个字符串的公共子序列的长度，其实道理是一样的。注意，这里两个序列的存储都是从0开始的，而dp数组的含义是前i个和前j个，也就是说，这里dp数组比序列往后错一个。

```
#include <iostream>
#include <cstring>
#include <algorithm>
#include <string>
using namespace std;
#define Max 505
unsigned long dp[Max][Max];

unsigned long max_sub_len(string a,string b){
    memset(dp,0, sizeof(dp));
    unsigned long len=a.size();
    for(int i=1;i<=len;i++)
        for(int j=1;j<=len;j++){
            if(a[i-1]==b[j-1])
                dp[i][j]=dp[i-1][j-1]+1;
            else
                dp[i][j]=max(dp[i-1][j],dp[i][j-1]);
        }
    return dp[len][len];
}
```

LIS 最长上升子序列

```
const int MAXX=100000+5;
const int INF=INT_MAX;

int a[MAXX],dp[MAXX]; // a数组为数据，dp[i]表示长度为i+1的LIS结尾元素的最小值

int main()
{
    int n;
    while(cin>>n)
    {
        for(int i=0; i<n; i++)
```

```
        {
            cin>>a[i];
            dp[i]=INF; // 初始化为无限大
        }
        int pos=0;     // 记录dp当前最后一位的下标
        dp[0]=a[0];    // dp[0]值显然为a[0]
        for(int i=1; i<n; i++)
        {
            if(a[i]>dp[pos])     // 若a[i]大于dp数组最大值，则直接添加
                dp[++pos] = a[i];
            else     // 否则找到dp中第一个大于等于a[i]的位置，用a[i]替换之。
                dp[lower_bound(dp,dp+pos+1,a[i])-dp]=a[i];  // 二分查找
        }
        cout<<pos+1<<endl;
    }
    return 0;
}
```

n个字符中插入m个乘号，求最大值

```
#include <iostream>
#include <cstring>
#include <algorithm>
#include <string>
#include <cstdio>
#include <climits>
#include <cstdlib>

using namespace std;
#define Max 20
unsigned long long nums[Max][Max];
unsigned long long dp[Max][Max];

//主要使用的字符串,其实输入的是tmp,father是经过简单处理之后的
string father;
unsigned long long my_atoull(int left,int right){
    unsigned long long ans=0;
    for(int i=left;i<=right;i++){
        ans *= 10;
        ans+=(father[i]-'0');
    }
    return ans;
}

int main(){
    freopen("in.txt","r",stdin);
    int m,n;
    while(cin>>m){
        string tmp;
        cin>>tmp;
```

```
        father = "0"+tmp;
        n=(int)tmp.length();

        //nums[i][j]表示从第i个字符到第j个字符组成的整数的大小
        for(int i=1;i<=n;i++)
            for(int j=i;j<=n;j++)
                nums[i][j]=my_atoull(i,j);


        //初始化dp数组
        for(int j=1;j<=n;j++){
            dp[0][j]=nums[1][j];
        }
        for(int i=1;i<=m;i++){
            for(int j=1;j<=i;j++)
                dp[i][j]=0;
        }

        //打表开始
        for(int i=1;i<=m;i++){
            for(int j=1;j<=n;j++){
                unsigned long long now=0;
                unsigned long long now_tmp;
                for(int k=i;k<=j-1;k++){
                    now_tmp = dp[i-1][k]*nums[k+1][j];
                    if(now_tmp>now)
                        now=now_tmp;
                }
                dp[i][j]=now;
            }
        }


        cout<<dp[m][n]<<endl;
    }
    return 0;
}
```

背包（全）

```
#include <iostream>
#include <cstring>
#include <algorithm>
using namespace std;
const int Max_volume=100005;
int dp[Max_volume];
int V;  //背包的最大容量


inline void ZeroOnePack(int value, int volume){
```

```
    for(int v=V;v>=volume;v--)
        if(v>=volume)
            dp[v]=dp[v]>dp[v-volume]+value?dp[v]:dp[v-volume]+value;
}

inline void CompletePack(int value, int volume){
    for(int v=volume;v<=V;v++)
            dp[v]=dp[v]>dp[v-volume]+value?dp[v]:dp[v-volume]+value;
}

inline void MultiplePack(int value,int volume, int number){
    if(volume*number>=V){
        CompletePack(value,volume);
        return;
    }
    int k=1;
    while(k<number){
        ZeroOnePack(value*k,volume*k);
        number-=k;
        k<<=1;
    }
    if(number)
        ZeroOnePack(value*number,volume*number);
}
```

股票系列

限制整个过程最多交易k次

```
int dp[2][1000005];
int maxProfit(int k, vector<int> &prices) {
    int len = (int) prices.size();
    if (len <= 0)
        return 0;
    k = min(k, len);
    if (k>len/2){
        int ans = 0;
        for (int i=1; i<len; ++i){
            ans += max(prices[i] - prices[i-1],0);
        }
        return ans;
    }
    memset(dp, 0, sizeof(dp));
    int cur = 0;
    for (int ii = 1; ii <= k; ii++) {
        cur = cur ^ 1;
        int min_prices = prices[0];
        for (int i = 1; i < len; i++) {
            min_prices = min(min_prices , prices[i] - dp[cur][i-1]);
            dp[cur^1][i] = max(dp[cur^1][i - 1] , prices[i] - min_prices);
```

```
            }
        }
        return dp[cur^1][len-1];
    }
```

限制一次交易后应该至少休息一次，解决方案是那个有趣的状态转移图

```
int buy[1000005];
int rest[1000005];
int sell[1000005];
int maxProfit(vector<int>& prices) {
    int len = (int) prices.size();
    if(len<2)
        return 0;
    rest[0]=0;
    buy[0]=0-prices[0];
    sell[0]=INT_MIN;
    for(int i=1;i<len;i++){
        rest[i] = max(rest[i-1],sell[i-1]);
        buy[i] = max(rest[i-1]-prices[i],buy[i-1]);
        sell[i] = buy[i-1]+prices[i];
    }
    return max(rest[len-1],sell[len-1]);
}
```

图论

DFS-邻接表实现

```
#include <cstdio>
#include <iostream>
#include <queue>
#include <functional>
#include <cstring>
#include <string>
#include <queue>
#include <algorithm>
using namespace std;
const int Max = 1005;
vector<int> g[Max];

bool used[Max];
int dist[Max];
//图的编号依旧是从1到n，很正常
void DFS(vector<int> g[],int start,int dist[],int n,int end){
    for(int i = 1;i<=n;i++) dist[i] = -1;
    for(int i = 1;i<=n;i++) used[i] = false;
    queue<int> que;
    while(!que.empty()) que.pop();
```

```
    que.push(start);
    used[start] = true;
    dist[start] = 0;
    while(!que.empty()){
        int u = que.front();
        que.pop();
        for(auto item : g[u]){
            if(!used[item]){
                used[item]=true;
                que.push(item);
                dist[item] = dist[u] + 1;
            }
            if(item == end) return;
        }
    }
}
```

二分图-匈牙利算法-邻接表实现

需要注意的是，这里的uN表示二分图左边的顶点个数，跟右边图定点个数以及编号方式随意。

```
#include <cstdio>
#include <iostream>
#include <cstring>
#include <algorithm>
#include <queue>
#include <vector>
using namespace std;

const int MaxN = 1005;
vector<int> g[MaxN];
int linker[MaxN];
bool used[MaxN];
int uN;
bool dfs(int u){
    for(auto v:g[u]){
        if(!used[v]){
            used[v] = true;
            if(linker[v] == -1 || dfs(linker[v])){
                linker[v] = u;
                return true;
            }
        }
    }
    return false;
}

int hungry(){
    int res = 0;
```

```
    memset(linker,-1,sizeof(linker));
    for(int  u = 0;u<uN;u++){
        memset(used,false,sizeof(used));
        if(dfs(u)) res++;
    }
    return res;
}
```

按字典序输出二分图左边已经配对的顶点，相关信息存储在数组ress中，按照以下代码输出的将是最小字典序，如若输出最大字典序的话，只需要将hungry中的第一个循环反向即可。

```
#define _CRT_SECURE_NO_WARNINGS
#include <cstdio>
#include <iostream>
#include <cstring>
#include <cmath>
#include <algorithm>
#include <vector>
#include <string>
#include <stack>
#include <cstdlib>
using namespace std;
const int mod = 100007;
const int MAXN = 1005;
typedef long long ll;
const int inf = 0x3f3f3f3f;
int uN, vN;
int Graph[MAXN][MAXN];
int linker[MAXN];
bool used[MAXN];
int ress[MAXN];
bool dfs(int u) {
    for (int v = 1; v <= 1000; v++) {
        if (Graph[u][v] && !used[v]) {
            used[v] = true;
            if (linker[v] == -1 || dfs(linker[v])) {
                linker[v] = u;
                ress[u] = 1;
                return true;
            }
        }
    }
    return false;
}
int hungary() {
    int res = 0;
    memset(linker, -1, sizeof(linker));
    for (int u = 1; u <= 1000; u++) {
        memset(used, false, sizeof(used));
```

```
        if (dfs(u)) res++;
    }
    for (int i = 1; i <= uN; i++) {
        int cnt = 0;
        if (ress[i] == 1) {
            cnt++;
            if (cnt == res) printf("%d\n", i);  //这里就是保证结尾没有多于空格，其实没什么实际意义，还费时间
            else printf("%d ", i);
        }
    }
    return res;
}
int main() {
    freopen("Text.txt", "r", stdin);
    while (~scanf("%d%d", &uN, &vN)) {
        memset(Graph, 0, sizeof(Graph));
        for (int i = 1; i <= vN; i++) {
            int num;
            scanf("%d", &num);
            for (int j = 1; j <= num; j++) {
                int tmp;
                scanf("%d", &tmp);
                Graph[i][tmp] = 1;
            }
        }
        printf("%d\n", hungary());
    }
    return 0;
}
```

最大流-EK算法-邻接表实现

```
#include <iostream>
#include <cstdio>
#include <cstring>
#include <vector>
#include <algorithm>
#include <queue>
#include <map>
#include <unordered_map>
using namespace std;
#define Max 10005
#define INF 0x7fffffff

int flow[Max],father[Max],vertex,E;
vector<pair<int,int>> g[Max];
inline void change_element_add(int a,int b,int c){
    int i=0;
    int len = (int) g[a].size();
```

```
    for(;i<len;i++)
        if(g[a][i].first == b)
            break;
    if(i<len)
        g[a][i].second += c;
    else
        g[a].emplace_back(b,c);
}

inline void change_element_sub(int a,int b,int c){
    int i=0;
    int len = (int) g[a].size();
    for(;i<len;i++)
        if(g[a][i].first == b){
            g[a][i].second -= c;
            break;
        }
}

inline int BFS(int s,int t){
    queue<int> q;
    while(!q.empty())
        q.pop();
    memset(father,-1, sizeof(int)*(vertex+5));
    flow[s] = INF;
    q.push(s);
    while(!q.empty()){
        int v = q.front();
        q.pop();
        for(auto item:g[v]){
            int i=item.first;
            if(father[i]==-1 && item.second>0){
                flow[i] = min(flow[v],item.second);
                father[i] = v;
                if(i==t)
                    return flow[t];
                q.push(i);
            }
        }
    }
    if(father[t] == -1)
        return -1;
    else
        return flow[t];
}

int EK(int s,int t){
    int ans = 0;
    int increase= BFS(s,t),k=t,last;
    while(increase!=-1){
        while(k!=s){
```

```
            last = father[k];
            change_element_sub(last,k,increase);
            change_element_add(k,last,increase);
            k=last;
        }
        ans += increase;
        k=t;
        increase= BFS(s,t);
    }
    return ans;
}
```

最小生成树-Kruskal算法-邻接表实现

```
#include <iostream>
#include <cstring>
#include <vector>
#include <cstdio>
#include <climits>
#include <cfloat>
#include <queue>
#include <functional>
#include <algorithm>
using namespace std;

struct edge{
public:
    int u;
    int v;
    int cost;
    edge(int u,int v,int cost):u(u),v(v),cost(cost){}
    bool operator < (const edge & b) const {
        return cost<b.cost;
    }
};

vector<edge> g;
const int Max = 10005;
int Find[Max];

int find(int x){
    if(Find[x] == -1)
        return x;
    else
        return Find[x] = find(Find[x]);
}


//返回最小生成树的最小费用
int Kruskal(int n){
```

```
    memset(Find,-1, sizeof(Find));
    sort(g.begin(),g.end());
    int cnt = 0;
    int ans = 0;
    for(auto item : g){
        int u = item.u;
        int v = item.v;
        int cost = item.cost;
        int t1 = find(u);
        int t2 = find(v);
        if(t1!=t2){
            ans += cost;
            Find[t1] = t2;
            cnt ++;
        }
        if(cnt == n-1)
            break;
    }
    if(cnt<n-1) return -1;
    else return ans;
}
```

并查集重要函数

```
int Find[Max];

//寻找根节点编号
int find(int x){
    if(Find[x] == -1)
        return x;
    else
        return Find[x] = find(Find[x]);
}
```

单源最短路-Dijkstra算法-邻接表实现

```
#include <cstdio>
#include <iostream>
#include <queue>
#include <functional>
#include <cstring>
#include <string>
#include <queue>
#include <algorithm>
using namespace std;
const int Max = 10005;
const int INF = 0x3f3f3f3f;

struct edge{
```

```
    int to;
    int cost;
    edge(int first,int to):to(first),cost(to){}
    edge(){}
    bool operator < (const edge & b) const {
        return cost>b.cost;
    }
};

vector<edge> g[Max];    //这个图本身
int dist[Max];  //最终每个点对应的距离
int Path[Max];  //记录路径
int vis[Max];   //记录是不是已经判断过了



//节点编号从1到n
void Dijkstra(int n,int start){
    memset(Path,-1, sizeof(Path));
    memset(vis,false,sizeof(vis));
    for(int i=1;i<=n;i++) dist[i] = INF;
    priority_queue<edge> que;
    while(!que.empty()) que.pop();
    dist[start] = 0;
    que.push(edge(start,0));
    edge tmp;
    while(!que.empty()){
        tmp = que.top();
        que.pop();
        int u = tmp.to;
        if(vis[u]) continue;
        vis[u] = true;
        for(auto item:g[u]){
            int v = item.to;
            int cost = item.cost;
            if(!vis[v] && dist[v]>dist[u]+cost){
                dist[v] = dist[u] + cost;
                que.push(edge(v,dist[v]));
                Path[v] = u;
            }
        }
    }
}
```

计算几何

凸包-Jarvis步进法

```
#include<cstdio>
```

```cpp
#include<vector>
#include<cmath>
#include<algorithm>
using namespace std;

//精度判断
const double eps = 1e-10;
double dcmp(double x) {
    if(fabs(x) < eps) return 0;
    else return x < 0 ? -1 : 1;
}

struct Point {
    double x, y;
    Point(double x=0, double y=0):x(x),y(y) {}
};
Point operator - (const Point& A, const Point& B) {
    return Point(A.x-B.x, A.y-B.y);
}
double Cross(const Point& A, const Point& B) {
    return A.x*B.y - A.y*B.x;
}
double Dot(const Point& A, const Point& B) {
    return A.x*B.x + A.y*B.y;
}
bool operator < (const Point& p1, const Point& p2) {
    return p1.x < p2.x || (p1.x == p2.x && p1.y < p2.y);
}
bool operator == (const Point& p1, const Point& p2) {
    return p1.x == p2.x && p1.y == p2.y;
}

//点集凸包，Jarvis步进法,注意，是逆时针方向，并且首尾不相连
vector<Point> ConvexHull(vector<Point> p) {
    //预处理，删除重复点
    sort(p.begin(), p.end());
    p.erase(unique(p.begin(), p.end()), p.end());

    int n = p.size();
    int m = 0;
    vector<Point> ch(n+1);
    for(int i = 0; i < n; i++) {
        while(m > 1 && Cross(ch[m-1]-ch[m-2], p[i]-ch[m-2]) <= 0) m--;
        ch[m++] = p[i];
    }
    int k = m;
    for(int i = n-2; i >= 0; i--) {
        while(m > k && Cross(ch[m-1]-ch[m-2], p[i]-ch[m-2]) <= 0) m--;
        ch[m++] = p[i];
    }
    if(n > 1) m--;
```

```
    ch.resize(m);
    return ch;
}
```

凸包-最后一次练习赛最后一题-包含判断两个凸包是否相交

```
#include<cstdio>
#include<vector>
#include<cmath>
#include<algorithm>
using namespace std;

//精度判断
const double eps = 1e-10;
double dcmp(double x) {
    if(fabs(x) < eps) return 0;
    else return x < 0 ? -1 : 1;
}

struct Point {
    double x, y;
    Point(double x=0, double y=0):x(x),y(y) {}
};
Point operator - (const Point& A, const Point& B) {
    return Point(A.x-B.x, A.y-B.y);
}
double Cross(const Point& A, const Point& B) {
    return A.x*B.y - A.y*B.x;
}
double Dot(const Point& A, const Point& B) {
    return A.x*B.x + A.y*B.y;
}
bool operator < (const Point& p1, const Point& p2) {
    return p1.x < p2.x || (p1.x == p2.x && p1.y < p2.y);
}
bool operator == (const Point& p1, const Point& p2) {
    return p1.x == p2.x && p1.y == p2.y;
}
//判断两条线段是否相离
bool SegmentProperIntersection(const Point& a1, const Point& a2, const Point&
b1, const Point& b2) {
    double c1 = Cross(a2-a1,b1-a1), c2 = Cross(a2-a1,b2-a1),
           c3 = Cross(b2-b1,a1-b1), c4=Cross(b2-b1,a2-b1);
    return dcmp(c1)*dcmp(c2)<0 && dcmp(c3)*dcmp(c4)<0;
}

bool OnSegment(const Point& p, const Point& a1, const Point& a2) {
    return dcmp(Cross(a1-p, a2-p)) == 0 && dcmp(Dot(a1-p, a2-p)) < 0;
}
```

```
//点集凸包，Jarvis步进法
vector<Point> ConvexHull(vector<Point> p) {
    //预处理，删除重复点
    sort(p.begin(), p.end());
    p.erase(unique(p.begin(), p.end()), p.end());

    int n = p.size();
    int m = 0;
    vector<Point> ch(n+1);
    for(int i = 0; i < n; i++) {
        while(m > 1 && Cross(ch[m-1]-ch[m-2], p[i]-ch[m-2]) <= 0) m--;
        ch[m++] = p[i];
    }
    int k = m;
    for(int i = n-2; i >= 0; i--) {
        while(m > k && Cross(ch[m-1]-ch[m-2], p[i]-ch[m-2]) <= 0) m--;
        ch[m++] = p[i];
    }
    if(n > 1) m--;
    ch.resize(m);
    return ch;
}
//判断点与凸多边形是否相离
int IsPointInPolygon(const Point& p, const vector<Point>& poly) {
    int wn = 0;
    int n = poly.size();
    for(int i=0; i<n; ++i) {
        const Point& p1 = poly[i];
        const Point& p2 = poly[(i+1)%n];
        if(p1 == p || p2 == p || OnSegment(p, p1, p2)) return -1;//在边界上
        int k = dcmp(Cross(p2-p1, p-p1));
        int d1 = dcmp(p1.y - p.y);
        int d2 = dcmp(p2.y - p.y);
        if(k > 0 && d1 <= 0 && d2 > 0) wn++;
        if(k < 0 && d2 <= 0 && d1 > 0) wn--;
    }
    if(wn != 0) return 1;//内部
    return 0;//外部
}
bool ConvexPolygonDisjoint(const vector<Point> ch1, const vector<Point> ch2) {
    int c1 = ch1.size();
    int c2 = ch2.size();
    for(int i=0; i<c1; ++i)
        if(IsPointInPolygon(ch1[i], ch2) != 0) return false;
    for(int i=0; i<c2; ++i)
        if(IsPointInPolygon(ch2[i], ch1) != 0) return false;
    for(int i=0; i<c1; ++i)
        for(int j=0; j<c2; ++j)
            if(SegmentProperIntersection(ch1[i], ch1[(i+1)%c1], ch2[j],
ch2[(j+1)%c2])) return false;
```

```
    return true;
}

int main()
{
    freopen("in.txt","r",stdin);
    int n, m;
    while(scanf("%d %d", &n, &m) == 2 && n > 0 && m > 0)
    {
        vector<Point> P1, P2;
        double x, y;
        for(int i = 0; i < n; i++) {
            scanf("%lf %lf", &x, &y);
            P1.push_back(Point(x, y));
        }
        for(int i = 0; i < m; i++) {
            scanf("%lf %lf", &x, &y);
            P2.push_back(Point(x, y));
        }
        if(ConvexPolygonDisjoint(ConvexHull(P1), ConvexHull(P2)))
            printf("YES\n");
        else
            printf("NO\n");
    }
    return 0;
}
```

VS2013 产品密钥 - 所有版本

Visual Studio Ultimate 2013 KEY（密钥）

BWG7X-J98B3-W34RT-33B3R-JVYW9

Visual Studio Premium 2013 KEY（密钥）

FBJVC-3CMTX-D8DVP-RTQCT-92494

Visual Studio Professional 2013 KEY（密钥）

XDM3T-W3T3V-MGJWK-8BFVD-GVPKY

Team Foundation Server 2013 KEY（密钥）

MHG9J-HHHX9-WWPQP-D8T7H-7KCQG