期末板子总结

排序

利用归并排序计算逆序对

```
long long merge(int l, int mid, int r) {
    int p1 = l, p2 = mid + 1, ind = p1;
    long long move = 0;
    while(p1 <= mid && p2 <= r) {
        if(arr[p1] <= arr[p2])
            tmp[ind++] = arr[p1++];
        else {
            move += p2 - ind;
            tmp[ind++] = arr[p2++];
        }
    }
    while(p1 <= mid)
        tmp[ind++] = arr[p1++];
    while(p2 <= r)
        tmp[ind++] = arr[p2++];
    for(int i = l; i < ind; i++)
        arr[i] = tmp[i];
    return move;
}

long long mergeSort(int l, int r) {
    long long move = 0;
    if(l < r) {
        int mid = l + (r - l) / 2;
        move += mergeSort(l, mid);
        move += mergeSort(mid + 1, r);
        move += merge(l, mid, r);
    }
    return move;
}
```

高精度、快速运算

高精度乘法

```
int mult(char a[], char b[], int cnum[]) {
    int alen, blen, clen, anum[2005], bnum[2005];
    alen = strlen(a);
    blen = strlen(b);
    for(int i = 0; i < alen; i++)
        anum[i] = a[alen - i - 1] - '0';
    for(int i = 0; i < blen; i++)
        bnum[i] = b[blen - i - 1] - '0';
    clen = alen + blen;
    for(int i = 0; i <= clen; i++)
        cnum[i] = 0;
    for(int i = 0; i < alen; i++)
    {
        for(int j = 0; j < blen; j++)
        {
            cnum[i + j] += anum[i] * bnum[j];
            cnum[i + j + 1] += (cnum[i + j]) / 10;
            cnum[i + j] %= 10;
        }
    }
    while(cnum[clen] == 0 && clen > 0)
        clen--;
    return clen;
}
```

矩阵快速幂

```
#include<stdio.h>
#define MOD 1000000007
int t, n, cnt, bindaryDecomposition[10], ind, num;
long long base[130][130], pow2[130][130], tmp[130][130];
int main() {
    scanf("%d", &t);
    while(t--) {
        scanf("%d", &n);
        ind = 0, num = n;
        while(num > 0) {
            bindaryDecomposition[ind++] = num % 2;
            num /= 2;
        }
        for(int i = 0; i < n; i++) {
            for(int j = 0; j < n; j++) {
                scanf("%lld", &pow2[i][j]);
                base[i][j] = 0;
            }
        }
        for(int i = 0; i < n; i++)
            base[i][i] = 1;
        for(int i = 0; i <= ind; i++) {
            if(bindaryDecomposition[i] == 1) {
                for(int j = 0; j < n; j++) {
                    for(int k = 0; k < n; k++) {
                        tmp[j][k] = 0;
                        for(int l = 0; l < n; l++)
                            tmp[j][k] = (tmp[j][k] + (base[j][l] * pow2[l][k])
% MOD) % MOD;
                    }
                    for(int k = 0; k < n; k++)
                        base[j][k] = tmp[j][k];
                }
            }
            for(int j = 0; j < n; j++) {
                for(int k = 0; k < n; k++) {
                    tmp[j][k] = 0;
                    for(int l = 0; l < n; l++)
                        tmp[j][k] = (tmp[j][k] + (pow2[j][l] * pow2[l][k]) %
MOD) % MOD;
                }
            }
            for(int j = 0; j < n; j++)
                for(int k = 0; k < n; k++)
                    pow2[j][k] = tmp[j][k];
        }
        for(int i = 0; i < n; i++) {
            for(int j = 0; j < n; j++)
                printf("%lld ", base[i][j]);
            printf("\n");
        }
    }
    return 0;
}
```

动态规划

钢管切割

```
#include<stdio.h>
#include<stdlib.h>
int n, sInd, flag, num, ind;
long long p[10005], dp[10005], s[10005], nums[10005];
int main() {
    scanf("%d", &n);
    for(int i = 1; i <= n; i++)
        scanf("%d", &(p[i]));
    dp[1] = p[1];
    s[1] = nums[1] = 1;
    sInd = 2;
    for(int i = 2; i <= n; i++) {
        flag = 0;
        dp[i] = p[i];
        s[i] = i;
        nums[i] = 1;
        for(int j = i - 1; j >= 1; j--) {
            if(dp[j] + p[i - j] > dp[i]) {
                dp[i] = dp[j] + p[i - j];
                nums[i] = nums[j] + 1;
                s[i] = i - j;
            }
        }
    }
    printf("%lld\n%lld\n", dp[n], nums[n]);
    ind = n;
    while(ind != s[ind]) {
        printf("%lld ", s[ind]);
        ind = ind - s[ind];
    }
    printf("%lld ", s[ind]);
    return 0;
}
```

卡特兰数

```
dp[1] = 1;
for(int i = 2; i <= n; i++) {
    dp[i] = 0;
    for(int j = i - 1; j >= 1; j--)
```

```
        dp[i] += dp[i - j] * dp[j];
}
```

矩阵链乘法

```
#include<stdio.h>
#include<stdlib.h>
#include<limits.h>
int n;
long long length[305], dpMax[305][305], dpMin[305][305];
int main() {
    scanf("%d", &n);
    for(int i = 0; i <= n; i++)
        scanf("%lld", &length[i]);
    for(int i = 0; i < n; i++)
        dpMax[i][i] = dpMin[i][i] = 0;
    for(int i = 1; i < n; i++) {
        for(int j = 0; j < n - i; j++) // [j,i]
        {
            dpMax[j][j + i] = 0;
            dpMin[j][j + i] = LLONG_MAX;
            for(int k = j; k < j + i; k++) {
                if((length[j] * length[k + 1] * length[j + i + 1] + dpMax[j][k]
+ dpMax[k + 1][j + i]) > dpMax[j][j + i])
                    dpMax[j][j + i] = (length[j] * length[k + 1] * length[j + i
+ 1] + dpMax[j][k] + dpMax[k + 1][j + i]);
                if((length[j] * length[k + 1] * length[j + i + 1] + dpMin[j][k]
+ dpMin[k + 1][j + i]) < dpMin[j][j + i])
                    dpMin[j][j + i] = (length[j] * length[k + 1] * length[j + i
+ 1] + dpMin[j][k] + dpMin[k + 1][j + i]);
            }
        }
    }
    printf("%.4lf\n", ((double)dpMax[0][n - 1]) / ((double)dpMin[0][n - 1]));
    return 0;
}
```

流水线调度

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

```
#include<limits.h>
typedef struct s {
    int father;
    long long time;
}station;
int T, m, n, f1, f2, preAns, depth, routeNum, route[25];
long long p[11][25], t[11][11];
station dp[11][25];
int main() {
    scanf("%d", &T);
    while(T--) {
        scanf("%d%d", &m, &n);
        for(int i = 0; i < n; i++)
            for(int j = 0; j < m; j++)
                scanf("%lld", &(p[i][j]));
        for(int i = 0; i < n; i++)
            for(int j = 0; j < n; j++)
                scanf("%lld", &(t[i][j]));
        for(int i = 0; i < n; i++)
            dp[i][0].time = p[i][0], dp[i][0].father = -1;
        for(int i = 1; i < m; i++) {
            for(int j = 0; j < n; j++) {
                dp[j][i].time = LLONG_MAX;
                for(int k = 0; k < n; k++) {
                    if((t[k][j] + dp[k][i - 1].time) < dp[j][i].time)
                        dp[j][i].time = t[k][j] + dp[k][i - 1].time, dp[j]
[i].father = k;
                    else if((t[k][j] + dp[k][i - 1].time) == dp[j][i].time) {
                        f1 = dp[j][i].father, f2 = k, depth = i, preAns = 0;
                        while(f1 != f2 && depth > 0) {
                            if(f1 > f2)
                                preAns = 1;
                            else if(f1 < f2)
                                preAns = -1;
                            depth--, f1 = dp[f1][depth].father, f2 = dp[f2]
[depth].father;
                        }
                        if(preAns == 1)
                            dp[j][i].father = k;
                    }
                }
                dp[j][i].time += p[j][i];
            }
        }
        routeNum = 0;
        for(int i = 1; i < n; i++) {
            if(dp[routeNum][m - 1].time > dp[i][m - 1].time)
                routeNum = i;
            else if(dp[routeNum][m - 1].time == dp[i][m - 1].time) {
                depth = m - 1;
                f1 = dp[routeNum][depth].father;
                f2 = dp[i][depth].father;
                preAns = 0;
                while(f1 != f2 && depth > 0) {
                    if(f1 > f2)
                        preAns = 1;
                    else if(f1 < f2)
                        preAns = -1;
                    depth--;
                    f1 = dp[f1][depth].father;
                    f2 = dp[f2][depth].father;
                }
                if(preAns == 1)
                    routeNum = i;
            }
        }
        printf("%lld\n", dp[routeNum][m - 1].time);
        depth = m - 1, route[depth + 1] = routeNum + 1;
        routeNum = dp[routeNum][depth].father;
        while(depth > 0)
            depth--, route[depth + 1] = routeNum + 1, routeNum = dp[routeNum]
[depth].father;
        for(int i = 1; i <= m; i++)
            printf("Station%d: Line%d\n", i, route[i]);
    }
    return 0;
}
```

OBST

```
#include<stdio.h>
#include<stdlib.h>
#include<limits.h>
int n, lson[505], rson[505], head[505][505];
long long p, sum[505], dp[505][505];
```

```
void saveSon(int l, int r) {
    if(l == r)
        return;
    if(l <= head[l][r] - 1)
        lson[head[l][r]] = head[l][head[l][r] - 1], saveSon(l, head[l][r] - 1);
    if(r >= head[l][r] + 1)
        rson[head[l][r]] = head[head[l][r] + 1][r], saveSon(head[l][r] + 1, r);
}
int main() {
    sum[0] = 0;
    scanf("%d", &n);
    for(int i = 1; i <= n; i++) {
        scanf("%lld", &p);
        sum[i] = sum[i - 1] + p;
    }
    for(int i = 1; i <= n; i++) {
        dp[i][i] = sum[i] - sum[i - 1];
        head[i][i] = i;
    }
    for(int i = 1; i <= n - 1; i++) {
        for(int j = 1; j <= n - i; j++) {
            dp[j][i + j] = LLONG_MAX;
            if((dp[j][j] + dp[j + 1][i + j] + sum[i + j] - sum[j]) < dp[j][i +
j]) {
                dp[j][i + j] = (dp[j][j] + dp[j + 1][i + j] + sum[i + j] -
sum[j]);
                head[j][i + j] = j;
            }
            if((dp[i + j][i + j] + dp[j][i + j - 1] + sum[i + j - 1] - sum[j -
1]) < dp[j][i + j]) {
                dp[j][i + j] = (dp[i + j][i + j] + dp[j][i + j - 1] + sum[i + j
- 1] - sum[j - 1]);
                head[j][i + j] = i + j;
            }
            for(int k = j + 1; k <= i + j - 1; k++) {
                if((dp[k][k] + dp[j][k - 1] + dp[k + 1][i + j] + sum[i + j] -
sum[k] + sum[k - 1] - sum[j - 1]) < dp[j][i + j]) {
                    dp[j][i + j] = (dp[k][k] + dp[j][k - 1] + dp[k + 1][i + j]
+ sum[i + j] - sum[k] + sum[k - 1] - sum[j - 1]);
                    head[j][i + j] = k;
                }
            }
        }
    }
    printf("%lld\n", dp[1][n]);
    for(int i = 1; i <= n; i++)
        lson[i] = rson[i] = -1;
    saveSon(1, n);
    for(int i = 1; i <= n; i++)
        printf("%d %d\n", lson[i], rson[i]);
    return 0;
}
```

字符串相关

最长公共子串

```
maxLen = 0;
for(int i = 0; i < len1; i++) {
    for(int j = 0; j < len2; j++) {
        if(!visited[i][j]) {
            visited[i][j] = 1;
            p1 = i;
            p2 = j;
            cnt = 0;
            while(p1 < len1 && p2 < len2 && s1[p1] == s2[p2]) {
                visited[p1][p2] = 1;
                cnt++;
                p1++;
                p2++;
            } maxLen = max(maxLen, cnt);
        }
    }
}
```

最长公共子序列

```
for(int i = 0; i < len1; i++) {
    for(int j = 0; j < len2; j++) {
        if(s1[i] == s2[j])
            dp[i + 1][j + 1] = 1 + dp[i][j];
        else
            dp[i + 1][j + 1] = max(dp[i][j + 1], dp[i + 1][j]);
    }
}
```

背包问题

01背包

优化后的一维算法

```
#include<bits/stdc++.h>
#define ll long long
using namespace std;
int t, n;
ll v0[1005], t0[1005], dp[1005];
int main() {
    scanf("%d%d", &t, &n);
    for(int i = 0; i < n; i++)
        scanf("%lld%lld", &t0[i], &v0[i]);
    for(int i = 0; i <= t; i++)
        dp[i] = 0;
    for(int i = 0; i < n; i++) {
        for(int j = t; j >= t0[i]; j--)
            dp[j] = max(dp[j], dp[j - t0[i]] + v0[i]);
    }
    printf("%lld\n", dp[t]);
    return 0;
}
```

完全背包

```
#include<bits/stdc++.h>
using namespace std;
int t, m;
long long  t0[10005], w0[10005], dp[10000005];
int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr), cout.tie(nullptr);
    cin >> t >> m;
    for (int i = 0; i < m; i++)
        cin >> t0[i] >> w0[i];
    for (int i = 0; i <= t; i++)
```

```
        dp[i] = 0;
    for(int i = 0; i < m; i++){
        for(int j = t0[i]; j <= t; j++)
            dp[j] = max(dp[j], dp[j - t0[i]] + w0[i]);
    }
    cout << dp[t];
    return 0;
}
```

分组背包

```
#include<bits/stdc++.h>
#define pii pair<int, int>
using namespace std;
int n, m, a, b, c, dp[1005];
vector<vector<pii>> group(101, vector<pii>());
int main() {
    scanf("%d%d", &m, &n);
    for (int i = 0; i < n; i++) {
        scanf("%d%d%d", &a, &b, &c);
        group[c].emplace_back(a, b);
    }
    for (int i = 0; i <= m; i++)
        dp[i] = 0;
    for (int i = 1; i <= 100; i++) { // 这里的100指的是最大的组号
        if (group[i].size() == 0)
            continue;
        for (int j = m; j >= 0; j--) {
            for(auto k: group[i]){
                if (j >= k.first)
                    dp[j] = max(dp[j], dp[j - k.first] + k.second);
            }
        }
    }
    printf("%d", dp[m]);
    return 0;
}
```

图算法及相关的数据结构

堆优化的dijkstra

下面这段代码以无向图为例

```
vector<vector<pii>> graph(n + 1);
vector<long long> gpath(n + 1, INF);
for (int i = 0; i < m; i++) {
    // input xi, yi, ti
    graph[xi].push_back({yi, ti});
    // 如果是有向图就删除下面这一句
    graph[yi].push_back({xi, ti});
} gpath[s] = 0;
queue<int> q;
q.push(s);
while(!q.empty()) {
    node = q.front();
    q.pop();
    for(auto edge : graph[node]) {
        if(gpath[node] + edge.time < gpath[edge.to]) {
            gpath[edge.to] = gpath[node] + edge.time;
            q.push(edge.to);
        }
    }
}
```

floyd-warshall 算法

```
#include<bits/stdc++.h>
#define INF LLONG_MAX
using namespace std;
typedef long long ll;
typedef pair<int, ll> pil;
int main() {
    int n, m, q, u, v;
    scanf("%d%d", &n, &m);
    vector<vector<ll>> graph(m, vector<ll>(3));
    ll dis[305][305];
    for (int i = 1; i <= n; i++)
        for (int j = 1; j <= n; j++)
            dis[i][j] = (i == j) ? 0 : INF;
    for(int i = 0; i < m; i++)
        scanf("%lld%lld%lld", &(graph[i][0]), &(graph[i][1]), &(graph[i][2]));
    for (const auto& edge : graph) {
        long long u = edge[0], v = edge[1], w = edge[2];
        dis[u][v] = min(dis[u][v], w);
    } for (int k = 1; k <= n; ++k) {
        for (int i = 1; i <= n; ++i) {
            for (int j = 1; j <= n; ++j) {
                if (dis[i][k] != INF && dis[k][j] != INF)
                    dis[i][j] = min(dis[i][j], dis[i][k] + dis[k][j]);
            }
        }
    } scanf("%d", &q);
    while(q--) {
        scanf("%d%d", &u, &v);
        printf("%lld\n", dis[u][v] == INF ? -1 : dis[u][v]);
    } return 0;
}
```

相信我吧，这真的是SPFA

```
vector<vector<pil>> graph(n + 1);
vector<bool> visited(n + 1, false);
vector<ll> distance(n + 1, 1000000000);
visited[1] = true, distance[1] = 0, count[1] = 1, q.push(1);
while (!q.empty()) {
    node = q.front(), q.pop(), visited[node] = false;
    for (auto edge: graph[node]) {
        if (distance[edge.first] > distance[node] + edge.second) {
            distance[edge.first] = distance[node] + edge.second;
            if (!visited[edge.first]) {
                q.push(edge.first);
                visited[edge.first] = true;
            }
        }
    }
}
```

最小生成树

Prim算法

prim算法计算最小生成树权重

```
#include<bits/stdc++.h>
using namespace std;
int t, n, m, u, v, currentIndex;
long long w, cost, currentWeight;
vector<vector<pair<int, long long>>> graph;
vector<bool> visited;
int main() {
    scanf("%d", &t);
    while(t--) {
        scanf("%d%d", &n, &m);
        graph.assign(n + 1, vector<pair<int, long long>>());
        visited.assign(n + 1, false);
        for(int _ = 0; _ < m; _++) {
            scanf("%d%d%lld", &u, &v, &w);
            graph[u].emplace_back(make_pair(v, w));
            graph[v].emplace_back(make_pair(u, w));
        } cost = 0;
        priority_queue<pair<long long, int>, vector<pair<long long, int>>,
greater<>> pQueue;
        pQueue.emplace(0, 1);
        while(!pQueue.empty()) {
            currentIndex = pQueue.top().second, currentWeight =
pQueue.top().first;
            pQueue.pop();
            if(visited[currentIndex])
                continue;
            cost += currentWeight, visited[currentIndex] = true;
            for(pair<int, long long> edge: graph[currentIndex]) {
                if(!visited[edge.first])
                    pQueue.emplace(edge.second, edge.first);
            }
        } printf("%lld\n", cost);
    } return 0;
}
```

Kruskal

这里提供并查集版本的Kruskal算法，但是注意哟，这里求的生成树不全（只连接k个节点）

```
#include<bits/stdc++.h>
#define pii pair<int, int>
using namespace std;
```

```
struct dsu {
    vector<size_t> pa, size;
    explicit dsu(size_t size_) : pa(size_), size(size_, 1) {
        iota(pa.begin(), pa.end(), 0);
    }
    size_t find(size_t x) {
        return pa[x] == x ? x : pa[x] = find(pa[x]);
    }
    void unite(size_t x, size_t y) {
        x = find(x), y = find(y);
        if (x == y) return;
        if (size[x] < size[y]) swap(x, y);
        pa[y] = x;
        size[x] += size[y];
    }
};
struct edge {
    int u, v, w;
}edges[40000];
int n, m, k, num, ans;
int main () {
    ios::sync_with_stdio(false);
    cin.tie(nullptr), cout.tie(nullptr);
    cin >> n >> m >> k;
    dsu set0(n);
    for (int i = 0; i < m; i++)
        cin >> edges[i].u >> edges[i].v >> edges[i].w;
    qsort(edges, m, sizeof(edge), [](const void *p1, const void *p2){
        return ((edge *)p1) -> w - ((edge *)p2) -> w;
    });
    num = n - k, ans = 0;
    for (int i = 0; i < m; i++) {
        if (num == 0)
            break;
        int a = set0.find(edges[i].u), b = set0.find(edges[i].v);
        if (a != b)
            set0.unite(a, b), ans += edges[i].w, num--;
    }
    if (num != 0)
        cout << "No Answer";
    else
        cout << ans;
    return 0;
}
```

最大流问题

dinic算法

下面的这个范例提供了最大流模板的使用，以及遇到无法抵达终点的时候的解决方案。

```
#include<bits/stdc++.h>
#define ll long long
#define N 10000
using namespace std;
struct MF {
    struct edge {
        int v, nxt;
        ll cap, flow;
    } e[N];
    int fir[N], dep[N], cur[N];
    int cnt = 0, n, S, T;
    ll maxFlow = 0;
    void init(int n0, int s0, int t0) {
        memset(fir, -1, sizeof fir);
        cnt = 0, maxFlow = 0;
        n = n0, S = s0, T = t0;
    }
    void addEdge(int u, int v, ll w) {
        e[cnt] = {v, fir[u], w, 0};
        fir[u] = cnt++;
        e[cnt] = {u, fir[v], 0, 0};
        fir[v] = cnt++;
    }
    bool bfs() {
        queue<int> q;
        memset(dep, 0, sizeof(int) * (n + 1));
        dep[S] = 1;
        q.push(S);
        while (q.size()) {
            int u = q.front();
            q.pop();
            for (int i = fir[u]; ~i; i = e[i].nxt) {
                int v = e[i].v;
                if ((!dep[v]) && (e[i].cap > e[i].flow)) {
                    dep[v] = dep[u] + 1;
                    q.push(v);
                }
            }
        }
        return dep[T];
    }
    ll dfs(int u, ll flow) {
        if ((u == T) || (!flow)) return flow;
        ll ret = 0;
        for (int& i = cur[u]; ~i; i = e[i].nxt) {
            int v = e[i].v, d;
            if ((dep[v] == dep[u] + 1) &&
                (d = dfs(v, min(flow - ret, e[i].cap - e[i].flow)))) {
                ret += d;
                e[i].flow += d;
                e[i ^ 1].flow -= d;
                if (ret == flow) return ret;
            }
        }
        return ret;
    }
    void dinic() {
        while (bfs()) {
            memcpy(cur, fir, sizeof(int) * (n + 1));
            maxFlow += dfs(S, LLONG_MAX);
        }
    }
} mf;
int n, m, a, b;
ll x, c;
int main () {
    ios::sync_with_stdio(false);
    cin.tie(nullptr), cout.tie(nullptr);
    cin >> n >> m >> x;
    mf.init(n, 1, n);
    while (m--) {
        cin >> a >> b >> c;
        mf.addEdge(a, b, c);
    }
    mf.dinic();
    if (mf.maxFlow == 0)
```

```
        cout << "Orz Ni Jinan Saint Cow!";
    else
        cout << mf.maxFlow << " " << ((mf.maxFlow + x - 1) / mf.maxFlow);
    return 0;
}
```

二分图匹配

不带权

```
#include<bits/stdc++.h>
using namespace std;
struct augment_path {
    vector<vector<int> > g;
    vector<int> p, vis;
    int n, m, dfn, res, cnt;

    augment_path(int _n, int _m) : n(_n), m(_m) {
        p = vector<int>(n + m + 1, -1), vis = vector<int>(n + m + 1);
        g.resize(n + m + 1), res = 0, dfn = 0, cnt = 0;
    }

    void add(int u, int v) {
        g[u].emplace_back(v + n);
        g[v + n].emplace_back(u);
    }

    bool dfs(int v) {
        vis[v] = dfn;
        for (int u : g[v]) {
            if (p[u] == -1) {
                p[u] = v, p[v] = u;
                return true;
            }
        }
        for (int u : g[v]) {
            if (vis[p[u]] != dfn && dfs(p[u])) {
                p[v] = u, p[u] = v;
                return true;
            }
        }
        return false;
    }

    int solve() {
        while (true) {
            dfn++, cnt = 0;
            for (int i = 1; i <= n; i++) {
                if (p[i] == -1 && dfs(i))
                    cnt++;
            }
            if (cnt == 0)
                break;
            res += cnt;
        }
        return res;
    }
};
int main () {
    ios::sync_with_stdio(false);
    cin.tie(nullptr), cout.tie(nullptr);
    int n, m, k, b;
    cin >> n >> m;
    augment_path ap(n, m);
    for (int a = 1; a <= n; a++) {
        cin >> k;
        while (k--) {
            cin >> b;
            ap.add(a, b);
        }
    }
    cout << ap.solve();
    return 0;
}
```

带权

这里提供的是最小权匹配，也就是把边的值设置为负数的最大边匹配，最大边也可以同理理解。

```
#include<bits/stdc++.h>
#define ll long long
using namespace std;
struct point {
    ll x, y;
}sPoints[245], tPoints[245];
int t, n, u;
vector<vector<ll>> graph;
vector<ll> lx, ly, slack;
vector<bool> visX, visY;
vector<int> pre, matchX, matchY;
queue<int> q;
bool check(int v) {
    visY[v] = true;
    if (matchY[v] != -1) {
        q.push(matchY[v]);
        visX[matchY[v]] = true;
        return false;
    } while (v != -1) {
        matchY[v] = pre[v];
        swap(v, matchX[pre[v]]);
    } return true;
}
void bfs(int i) {
    while (!q.empty()) q.pop();
    q.push(i), visX[i] = true;
    while (true) {
        while (!q.empty()) {
            u = q.front(), q.pop();
            for (int v = 0; v < n; v++) {
                if (!visY[v]) {
                    ll delta = lx[u] + ly[v] - graph[u][v];
                    if (slack[v] >= delta) {
                        pre[v] = u;
                        if (delta)
                            slack[v] = delta;
                        else if (check(v))
                            return;
                    }
                }
            }
        } ll a = LLONG_MAX;
        for (int j = 0; j < n; j++) {
            if (!visY[j])
                a = min(a, slack[j]);
        } for (int j = 0; j < n; j++) {
            if (visX[j])
```

```
                lx[j] -= a;
            if (visY[j])
                ly[j] += a;
            else
                slack[j] -= a;
        }
        for (int j = 0; j < n; j++) {
            if (!visY[j] && slack[j] == 0 && check(j)) {
                return;
            }
        }
    }
}
int main() {
    scanf("%d", &t);
    while (t--) {
        scanf("%d", &n);
        for (int i = 0; i < n; i++)
            scanf("%lld%lld", &(sPoints[i].x), &(sPoints[i].y));
        for (int i = 0; i < n; i++)
            scanf("%lld%lld", &(tPoints[i].x), &(tPoints[i].y));
        graph.assign(n, vector<ll>(n));
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++)
                graph[i][j] = -abs(sPoints[i].x - tPoints[j].x) -
abs(sPoints[i].y - tPoints[j].y);
        } lx.assign(n, LLONG_MIN), ly.assign(n, 0);
        for (int i = 0; i < n; i ++) {
            for (int j = 0; j < n; j++)
                lx[i] = max(lx[i], graph[i][j]);
        } pre.assign(n, -1), matchX.assign(n, -1), matchY.assign(n, -1);
        for (int i = 0; i < n; i++) {
            slack.assign(n, LLONG_MAX);
            visX.assign(n, false);
            visY.assign(n, false);
            bfs(i);
        } ll cost = 0;
        for (int i = 0; i < n; i++)
            cost += graph[i][matchX[i]];
        cout << -cost << endl;
    } return 0;
}
```

不带权

拓扑排序的应用

分层拓扑排序

```
#include<bits/stdc++.h>
using namespace std;
int t, n, m, u, v, hours, k, flag, size0;
int main() {
    scanf("%d", &t);
    while (t--) {
        hours = 0;
        scanf("%d%d", &n, &m);
        vector<vector<int>> graph(n + 1);
        vector<int> indegree(n + 1, 0);
        while(m--) {
            scanf("%d%d", &u, &v);
            flag = 0;
            for(int u0: graph[v]) {
                if(u0 == u) {
                    flag = 0;
                    break;
                }
            } if(flag == 1) break;
            graph[u].push_back(v);
            indegree[v]++;
        } queue<int> q;
        for(int i = 1; i <= n; i++)
            if (indegree[i] == 0)
                q.push(i);
        hours = 0;
        while (!q.empty()) {
            size0 = q.size();
            hours++;
            while(size0--) {
                k = q.front();
                q.pop();
                for(int node : graph[k]) {
                    indegree[node]--;
                    if(indegree[node] == 0)
                        q.push(node);
                }
            }
        }
        printf("%d\n", hours);
    }
    return 0;
}
```

DAG（半连通图）的判断

```
#include<bits/stdc++.h>
using namespace std;
int t, n, m, u, v, node;
bool flag;
int main() {
    scanf("%d", &t);
    while(t--) {
        scanf("%d%d", &n, &m);
        vector<vector<int>> graph(n + 1);
        vector<int> indegree(n + 1, 0);
        while(m--) {
            scanf("%d%d", &u, &v);
            graph[u].push_back(v);
            indegree[v]++;
        } queue<int> q;
        for(int i = 1; i <= n; i++)
            if (indegree[i] == 0)
                q.push(i);
        flag = true;
        while(!q.empty()) {
            if(q.size() != 1) {
                flag = false;
                break;
            } node = q.front();
            q.pop();
            for(int edge: graph[node])
                if(--indegree[edge] == 0)
                    q.push(edge);
        }
        printf("%s\n", flag ? "YES" : "NO");
    } return 0;
}
```

负环的判断

BF算法判负环

```
#include<bits/stdc++.h>
using namespace std;
class Edge {
public:
    int s, e;
    long long v;
};
Edge edges[6005];
const long long INF = 1000000000;
int t, n, m, flag;
long long dis[2003];
int main() {
    scanf("%d", &t);
    while(t--) {
        scanf("%d%d", &n, &m);
        for(int i = 1; i <= m; i++)
            scanf("%d%d%lld", &(edges[i].s), &(edges[i].e), &(edges[i].v));
        dis[1] = 0;
        for(int i = 2; i <= n; i++)
            dis[i] = INF;
        for(int i = 1; i <= n - 1; i++) {
            for(int j = 1; j <= m; j++) {
                if(dis[edges[j].e] > dis[edges[j].s] + edges[j].v)
                    dis[edges[j].e] = dis[edges[j].s] + edges[j].v;
            }
        } flag = 1;
        for(int i = 1; i <= m; i++) {
            if(dis[edges[i].e] > dis[edges[i].s] + edges[i].v) {
                printf("boo how");
                flag = 0;
                break;
            }
        } if(flag) for(int i = 1; i <= n; i++) printf("%lld ", dis[i]);
        printf("\n");
    } return 0;
}
```

SPFA判断负环

```
#include<bits/stdc++.h>
#define ll long long
#define pil pair<int, ll>
using namespace std;
int t, n, m, u, v, node;
ll w;
bool flag;
int main () {
    ios::sync_with_stdio(false);
    cin.tie(nullptr), cout.tie(nullptr);
    cin >> t;
    while (t--) {
        cin >> n >> m;
        vector<vector<pil>> graph(n + 1);
        vector<ll> dis(n + 1, INT32_MAX);
        vector<int> inCnt(n + 1, 0);
        vector<bool> inQueue(n + 1, false);
        queue<int> q;
        while (m--) {
            cin >> u >> v >> w;
            if (w >= 0)
                graph[u].emplace_back(v, w), graph[v].emplace_back(u, w);
            else
                graph[u].emplace_back(v, w);
        }
        q.push(1), inCnt[1] = 1, dis[1] = 0, flag = false;
        while (!q.empty()) {
            node = q.front(), q.pop(), inQueue[node] = false;
            if (inCnt[node] >= n) {
                flag = true;
                break;
            }
            for (auto edge: graph[node]) {
                if (dis[node] + edge.second < dis[edge.first]) {
                    dis[edge.first]= dis[node] + edge.second;
                    if (!inQueue[edge.first])
                        q.push(edge.first), inQueue[edge.first] = true,
inCnt[edge.first]++;
                }
            }
        }
        cout << (flag ? "YES\n" : "NO\n");
    }
    return 0;
}
```

计算几何

wa的时候记得开`long double`，说不定就过了（）。

点的数据结构

```
struct point {
    double x;
    double y;
    point operator+(const point& other) const {
        return {x + other.x, y + other.y};
    }
    point operator-(const point& other) const {
        return {x - other.x, y - other.y};
    }
    point operator*(const double scalar) const {
        return {x * scalar, y * scalar};
    }
    point operator/(const double scalar) const {
        return {x / scalar, y / scalar};
    }
};
```

点到线段的距离

```
typedef struct Point {
    double x, y;
}point;
struct LineSegment {
    Point p1, p2;
};
double distance(Point p1, Point p2) {
    return sqrt(pow(p2.x - p1.x, 2) + pow(p2.y - p1.y, 2));
}
double distance(Point p, LineSegment l) {
    double length = distance(l.p1, l.p2);
    if (length == 0)
        return distance(p, l.p1);
    double r = ((p.x - l.p1.x) * (l.p2.x - l.p1.x) + (p.y - l.p1.y) * (l.p2.y -
l.p1.y)) / pow(length, 2);
    if (r <= 0)
        return distance(p, l.p1);
    else if (r >= 1)
        return distance(p, l.p2);
    else {
        Point foot = { l.p1.x + r * (l.p2.x - l.p1.x), l.p1.y + r * (l.p2.y -
l.p1.y) };
        return distance(p, foot);
    }
}
```

线段的相交、平行

```
typedef struct p {
    double x, y;
}point;
int t;
double direction(p i, p j, p k) {
    return (k.x - i.x) * (j.y - i.y) - (k.y - i.y) * (j.x - i.x);
}
bool onSegment(p i, p j, p k) {
    return (min(i.x, j.x) <= k.x && k.x <= max(i.x, j.x)) && (min(i.y, j.y) <=
k.y && k.y <= max(i.y, j.y));
}
bool intersect(p p1, p p2, p p3, p p4) {
    double d1 = direction(p3, p4, p1), d2 = direction(p3, p4, p2), d3 =
direction(p1, p2, p3), d4 = direction(p1, p2, p4);
    if(d1 * d2 < 0 && d3 * d4 < 0)
        return true;
    else if(d1 == 0 && onSegment(p3, p4, p1))
        return true;
    else if(d2 == 0 && onSegment(p3, p4, p2))
        return true;
    else if(d3 == 0 && onSegment(p1, p2, p3))
        return true;
    else if(d4 == 0 && onSegment(p1, p2, p4))
        return true;
    return false;
}
bool parallel(point p1, point q1, point p2, point q2) {
    return (p1.x - q1.x) * (p2.y - q2.y) == (p1.y - q1.y) * (p2.x - q2.x);
}
```

凸包的面积计算

```
#include<bits/stdc++.h>
using namespace std;
typedef struct P {
    long long x, y;
}point;
int n, top, idx, t, start;
long long area;
point p[100005], convex[100005];
long long cross(const point &a, const point &b, const point &c, const point &d)
{
    return (b.x - a.x) * (d.y - c.y) - (b.y - a.y) * (d.x - c.x);
}
bool cmp(const point &a, const point &b) {
    long long temp = cross(p[0], a, p[0], b);
    if (temp == 0) return a.x < b.x;
    return temp > 0;
}
int main() {
    scanf("%d", &t);
    while (t--) {
        scanf("%d", &n);
        start = 0;
        for (int i = 0; i < n; i++) {
            scanf("%lld%lld", &(p[i].x), &(p[i].y));
            if (p[i].y < p[start].y || (p[i].y == p[start].y && p[i].x <
p[start].x))
                start = i;
        } swap(p[0], p[start]);
        sort(p + 1, p + n, cmp);
        top = 0, idx = 0;
        while (idx < n) {
            while (top >= 2 && cross(convex[top - 2], convex[top - 1],
convex[top - 1], p[idx]) <= 0)
                --top;
            convex[top++] = p[idx++];
        } area = 0;
        for(int i = 2; i < top; i++)
```

```
            area += abs(cross(convex[0], convex[i - 1], convex[0], convex[i]));
        if(area % 2 == 0)
            printf("%lld.0\n", area / 2);
        else
            printf("%lld.5\n", area / 2);
    } return 0;
}
```

快速傅里叶变换 FFT

```
const double PI = acos(-1.0);
struct Complex {
    long double x, y;
    Complex(long double _x = 0.0, long double _y = 0.0) {
        x = _x;
        y = _y;
    }
    Complex operator-(const Complex &b) const {
        return Complex(x - b.x, y - b.y);
    }
    Complex operator+(const Complex &b) const {
        return Complex(x + b.x, y + b.y);
    }
    Complex operator*(const Complex &b) const {
        return Complex(x * b.x - y * b.y, x * b.y + y * b.x);
    }
};
void change(Complex y[], int len) {
    int k;
    for (int i = 1, j = len / 2; i < len - 1; i++) {
        if (i < j) swap(y[i], y[j]);
        k = len / 2;
        while (j >= k)
            j = j - k, k = k / 2;
        if (j < k) j += k;
    }
}
void dft(Complex y[], int len, int on) {
    change(y, len);
    for (int h = 2; h <= len; h <<= 1) {
        Complex wn(cos(2 * PI / h), sin(on * 2 * PI / h));
        for (int j = 0; j < len; j += h) {
            Complex w(1, 0);
            for (int k = j; k < j + h / 2; k++) {
                Complex u = y[k];
                Complex t = w * y[k + h / 2];
                y[k] = u + t, y[k + h / 2] = u - t, w = w * wn;
            }
        }
    }
    if (on == -1) {
        for (int i = 0; i < len; i++)
            y[i].x /= len, y[i].y /= len;
    }
}
```

这里附上使用方法

```
int main() {
    while (scanf("%s%s", str1, str2) == 2) {
        int len1 = strlen(str1);
        int len2 = strlen(str2);
        int len = 1;
        while (len < len1 * 2 || len < len2 * 2) len <<= 1;
        for (int i = 0; i < len1; i++) x1[i] = Complex(str1[len1 - 1 - i] -
'0', 0);
        for (int i = len1; i < len; i++) x1[i] = Complex(0, 0);
        for (int i = 0; i < len2; i++) x2[i] = Complex(str2[len2 - 1 - i] -
'0', 0);
        for (int i = len2; i < len; i++) x2[i] = Complex(0, 0);
        fft(x1, len, 1);
        fft(x2, len, 1);
        for (int i = 0; i < len; i++) x1[i] = x1[i] * x2[i];
        fft(x1, len, -1);
        for (int i = 0; i < len; i++) sum[i] = int(x1[i].x + 0.5);
        for (int i = 0; i < len; i++) {
            sum[i + 1] += sum[i] / 10;
            sum[i] %= 10;
        }
        len = len1 + len2 - 1;
        while (sum[len] == 0 && len > 0) len--;
        for (int i = len; i >= 0; i--) printf("%c", sum[i] + '0');
        printf("\n");
    }
    return 0;
}
```

字符串算法

字符串哈希

注意，下面这个模板的字符串哈希的坐标是1到n而不是0到n-1

```
const ull mod1 = 1e9 + 7, mod2 = 2147483647;
const int N = 10e6 + 10;
const int p = 131;
ull a1[N], a2[N], h1[N], h2[N];
void hash1(string s) { //预处理hash函数前缀和
    a1[0] = 1;
    int n = s.size();
    for (int i = 1; i <= n; i++)
    {
        a1[i] = a1[i - 1] * p % mod1;
        h1[i] = (h1[i - 1] * p % mod1 + (s[i - 1] - '0' + 1)) % mod1;
    }
}
void hash2(string s) { //预处理hash函数前缀和
    a2[0] = 1;
    int n = s.size();
    for (int i = 1; i <= n; i++)
    {
        a2[i] = a2[i - 1] * p % mod2;
        h2[i] = (h2[i - 1] * p % mod2 + (s[i - 1] - '0' + 1)) % mod2;
    }
}
ull get1(int l, int r) { //计算s[l--r]的hash值
    return (h1[r] - h1[l - 1] * a1[r - l + 1] % mod1 + mod1) % mod1;
}
ull get2(int l, int r) { //计算s[l--r]的hash值
    return (h2[r] - h2[l - 1] * a2[r - l + 1] % mod2 + mod2) % mod2;
}
```

KMP算法

这里我们提供四个函数，两种计算next数组（一种为定义值-1，另一种为正常定义值），两种实现kmp（思路不同，功能不同）。

```
void cal_next(char *str, int *next, int length) {
    next[0] = -1;
    int k = -1;
    for (int q = 1; q <= length - 1; q++) {
        while (k > -1 && str[k + 1] != str[q])
            k = next[k];
        if (str[k + 1] == str[q])
            k = k + 1;
        next[q] = k;
    }
}

vector<int> prefix(string s) {
  int n = (int)s.length();
  vector<int> pi(n);
  for (int i = 1; i < n; i++) {
    int j = pi[i - 1];
    while (j > 0 && s[i] != s[j]) j = pi[j - 1];
    if (s[i] == s[j]) j++;
    pi[i] = j;
  }
  return pi;
}

int KMP(char *str, int slen, char *ptr, int plen) {
    int *next = new int[plen];
    cal_next(ptr, next, plen);
    int k = -1;
    for (int i = 0; i < slen; i++) {
        while (k > -1 && ptr[k + 1] != str[i])
            k = next[k];
        if (ptr[k + 1] == str[i])
            k = k + 1;
        if (k == plen - 1)
            return i - plen + 1;
    }
    return -1;
}

vector<int> find_occurrences(string text, string pattern) {
  string cur = pattern + '#' + text;
  int sz1 = text.size(), sz2 = pattern.size();
  vector<int> v;
  vector<int> lps = prefix(cur);
  for (int i = sz2 + 1; i <= sz1 + sz2; i++) {
    if (lps[i] == sz2)
        v.push_back(i - 2 * sz2);
  }
  return v;

}
```

c++模板的使用

重载运算符

结构体

```
struct Point {
    int x, y;
    // 小于号运算符重载
    bool operator<(const Point& other) const {
        return (x < other.x) || (x == other.x && y < other.y);
    }
    // 大于号运算符重载
    bool operator>(const Point& other) const {
        return (x > other.x) || (x == other.x && y > other.y);
    }
};
```

类

```
class MyClass {
public:
    int value;
    // 加法运算符重载
    MyClass operator+(const MyClass& other) const {
        MyClass result;
        result.value = this->value + other.value;
        return result;
    }
    // 除法运算符重载
    MyClass operator/(const MyClass& other) const {
        if (other.value == 0) {
            // 处理除零错误或抛出异常
        }
        MyClass result;
        result.value = this->value / other.value;
        return result;
    }
};
```

优先队列

```
priority_queue<Type, Container, Functional>
```

其中Type为数据类型， Container为保存数据的容器，Functional为元素比较方式。

- 在STL中，默认情况下（不加后面两个参数）是以vector为容器，以 operator< 为比较方式，所以在只使用第一个参数时，优先队列默认是一个最大堆，每次输出的堆顶元素是此时堆中的最大元素。
- 用到最小堆，则一般要把模板的三个参数都带进去。STL里面定义了一个仿函数greater<>，对于基本类型可以用这个仿函数声明最小堆(升序)

如果需要自定义比较方式，对于结构体来说重载运算符是一个好选择。除此之外，还可以自定义比较函数。

自定义比较器

自定义比较器可以用到很多方面，比如c++的`sort`、`priority_queue`等等。

结构体

```
struct MyType {
    int priority;
    // 其他成员...

    MyType(int p) : priority(p) {}// 构造器
};
priority_queue<MyType, vector<MyType>, MyComparator> pq;
struct MyComparator {
    bool operator()(const MyType& a, const MyType& b) const {
        return a.priority > b.priority; // 优先级逻辑，这里可以根据需求修改
    }
};
```

类

```
class MyClass {
public:
    int value;
    // 构造函数
    MyClass(int v) : value(v) {}
    // 输出函数（方便展示结果）
    void display() const {
        std::cout << "Value: " << value << std::endl;
    }
};
bool myComparator(const MyClass& a, const MyClass& b) {
    return a.value < b.value; // 根据需要定义比较逻辑
}
priority_queue<MyType, vector<MyType>, MyComparator> pq;
```

unordered_map与map的使用

`std::unordered_map` 是 C++ 中用于存储键-值对的数据结构，基于哈希表实现，提供了快速的查找、插入和删除操作。`std::map` 按照键的值升序排列，如果需要有序的哈希表可以使用它。下面是关于 `std::unordered_map` 的基本使用方法：

导入头文件

```
#include <unordered_map>
```

定义和初始化

```
std::unordered_map<KeyType, ValueType> myMap;

std::unordered_map<std::string, int> ages = {
    {"Alice", 30},
    {"Bob", 25},
    {"Charlie", 35}
};
```

插入和访问元素

```
// 插入元素
myMap[key] = value; // 如果键 key 不存在，则插入；如果存在，则更新值为 value

// 访问元素
value = myMap[key]; // 根据键获取值，如果键不存在
// 则返回默认值（0 或空值，具体取决于值的类型）
```

检查键是否存在

```
if (myMap.find(key) != myMap.end()) {
    // 键存在于 unordered_map 中
} else {
    // 键不存在于 unordered_map 中
}
```

删除元素

```
myMap.erase(key); // 删除指定键的元素
遍历 unordered_map
cpp
Copy code
for (const auto& pair : myMap) {
    // pair.first 是键，pair.second 是对应的值
    // 可以在循环中对键值对进行操作
}
```

遍历方法

如果你想用迭代器遍历也可以。

```
for (const auto& pair : myMap) {
    std::cout << "Key: " << pair.first << ", Value: " << pair.second <<
std::endl;
}
```

unordered_set与set的使用

和上文中的另外两个数据结构一样，集合也是分为有序的和无序的。两种数据结构操作的方式大致心相通，这里仅仅记录区别部分。

插入、删除

```
mySet.insert(5); // 插入元素 5
mySet.insert(10); // 插入元素 10 // 重复插入相同元素不会改变 set 中的内容
mySet.insert(5); // 仍然只有一个元素 5
mySet.erase(10); // 删除元素 10
```

构造函数

这里提供结构体、类的构造函数书写方法：

结构体

```
struct Point {
    int x, y;
    // 结构体的构造函数
    Point(int x_val, int y_val) : x(x_val), y(y_val) {}
};
```

类

```
class MyClass {
public:
    int value;
    // 类的构造函数
    MyClass(int v) : value(v) {}
};
```

复杂度分析

主定理

$$T(n) = aT(\frac{n}{b}) + f(n)$$

在上述表达式中，a表示被分解成a个子问题，b表示每次分治后，问题规模都变为了$\frac{1}{b}$。

$$T(n) = \begin{cases} \Theta(n^{\log_b a}) & \text{if } \Theta(n^{\log_b a}) > f(n), \\ \Theta(n^{\log_b a} \log n) & \text{if } \Theta(n^{\log_b a}) = f(n), \\ \Theta(f(n)) & \text{if } \Theta(n^{\log_b a}) < f(n) \end{cases}$$

摊还分析

聚集法

会计法

势能法