Contents

1 Sync 1
1.1 Sync 1

2 Data Structure 1
2.1 Binary Search 1
2.2 BIT 1
2.3 BWT 1

3 Divide and Conquer 1
3.1 count inversions 1

4 DP 2
4.1 Doubling 2
4.2 LCS 2
4.3 LIS 2
4.4 LIS 2 2
4.5 Minimum Edit Distance 2

5 Enumerate 3
5.1 Halfcut Enumerate 3

6 Graph 3
6.1 SPFA 3
6.2 Dijkstra 3
6.3 Floyd Warshall 3
6.4 Disjoint set Kruskal 4
6.5 Disjoint set Kruskal 2 4
6.6 Bipatirate 4
6.7 Hungarian algorithm 5
6.8 LCA 5
6.9 Trie 5

7 Math 6
7.1 Hash 6

8 Function 6
8.1 CHAR 6
8.2 string 6
8.3 setprecision 6
8.4 GCD LCM 6
8.5 reverse 6
8.6 sort 6
8.7 map 6
8.8 set 7

9 Other 7
9.1 Ants Colony 7
9.2 Binary codes 7
9.3 Disk Tree 8

10 Greedy 8
10.1 Sticks 8

11 DP 8
11.1 Crested Ibis vs Monster 8
11.2 dpd Knapsack 1 9

12 Math 9
12.1 Big Mod 9
12.2 How Many Os 9
12.3 ORXOR 9

13 Segement Tree 9
13.1 Frequent values 9

14 Dijkstra 10
14.1 Walk Through the Forest 10

15 Kruskal 11
15.1 Qin Shi Huang Road System 11

16 Bipartite Graph 11
16.1 SAM I AM 11

1 Sync

1.1 Sync

```cpp
int main(){
    std::ios::sync_with_stdio(false);
    // 開始寫程式
}
```

2 Data Structure

2.1 Binary Search

```cpp
int binary_search(int arr[maxn], int lef, int rig, int target){
    if(lef > rig) return 0x3f3f3f3f;
    int mid = (lef + rig) >> 1;
    if(arr[mid] == target) return mid;
    else if(arr[mid] > target){
        return binary_search(arr, lef, mid - 1, target);
    }
    else{
        return binary_search(arr, mid + 1, rig, target);
    }
}
```

2.2 BIT

```cpp
/* BIT Binary Index Tree */
#define lowbit(k) (k & -k)
void add(vector<int> &tr, int id, int val) {
  for (; id <= n; id += lowbit(id)) {
    tr[id] += val;
  }
}
int sum(vector<int> &tr, int id) {
  int ret = 0;
  for (; id >= 1; id -= lowbit(id)) {
    ret += tr[id];
  }
  return ret;
}
```

2.3 BWT

```cpp
/* BWT 資料轉換演算法 */
void BWT(){
    for(int i = 0; i < n; ++i){
        if(back[i] == 0)
            mini[zero++] = i;
    for(int i = 0; i < n; ++i)
        if(back[i] == 1)
            mini[zero++] = i;
    int ptr = mini[0];
    for(int i = 0; i < n; ++i){
        cout << back[ptr] << " ";
        ptr = mini[ptr];
    }
    cout << endl;
}
```

3 Divide and Conquer

3.1 count inversions

```cpp
/*逆序數對*/
int arr[maxn], buf[maxn];
int count_inversions(int lef, int rig){
    if(rig - lef <= 1) return 0;
    int mid = (lef + rig)/2;
    int ans = count_inversions(lef, mid) + count_inversions(mid, rig);
    int i = lef, j = mid, k = lef;
    while(i < mid || j < rig){
        if(i >= mid) buf[k] = arr[j++];
        else if(j >= rig) buf[k] = arr[i++];
        else{
            if(arr[i] <= arr[j]) buf[k] = arr[i++];
            else{
                buf[k] = arr[j++];
                ans += mid - i;
            }
        }
        k++;
    }
    for(int k = lef; k < rig; ++k) arr[k] = buf[k];
    return ans;
}
```

4 DP

4.1 Doubling

```
/* 倍增 */
int LOG = sqrt(N); // 2^LOG >= N
vector<int> arr(N);
vector<vector<int>> dp(N, vector<int>(LOG));
for(int i = 0; i < N; ++i) cin >> arr[i];
int L, Q, a, b;
cin >> L >> Q;
for(int i = 0; i < N; ++i){
    dp[i][0] = lower_bound(arr.begin(), arr.end(),
        arr[i] + L) - arr.begin();
    if(dp[i][0] == N || arr[i] + L < arr[dp[i][0]])
        dp[i][0] -= 1;
}
for(int i = 1; i < LOG; ++i)
    for(int j = 0; j < N; ++j)
        dp[j][i] = dp[dp[j][i - 1]][i - 1];
for(int i = 0; i < Q; ++i){
    cin >> a >> b;
    a--; // 要減減是因為arr的index從0開始但題目從1開始
    b--;
    if(a > b) swap(a, b);
    int ans = 0;
    for(int i = LOG - 1; i >= 0; --i){ // 從後往回推
        if(dp[a][i] < b){
            ans += (1 << i);
            a = dp[a][i];
        }
    }
    cout << ans + 1 << endl;
}
```

4.2 LCS

```
/* Longest Common Subsequence */
int LCS(string s1, string s2) {
  int n1 = s1.size(), n2 = s2.size();
  int dp[n1+1][n2+1] = {0};
  // dp[i][j] = s1的前i個字元和s2的前j個字元
  for (int i = 1; i <= n1; i++) {
    for (int j = 1; j <= n2; j++) {
      if (s1[i - 1] == s2[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[n1][n2];
}
```

4.3 LIS

```
/* Longest Increasing Subsequence */
int LIS(vector<int> &a) {
  vector<int> s;
  for (int i = 0; i < a.size(); i++) {
    if (s.empty() || s.back() < a[i]) {
      s.push_back(a[i]);
    } else {
      *lower_bound(s.begin(), s.end(), a[i],
        [](int x, int y) {return x < y;}) = a[i];
    }
  }
  return s.size();
}
```

4.4 LIS 2

```
int LIS(vector<int> &a){
    int len[a.size()];
    for(int i = 0; i < a.size(); ++i) len[i] = 1;
    int maxi = -1;
    for(int i = 0; i < a.size(); ++i)
        for(int j = i + 1; j < a.size(); ++j)
            if(a[i] <= a[j]) len[j] = max(len[j],
                len[i] + 1);

    for(int i = 0; i < a.size(); ++i)
        maxi = max(maxi, len[i]);
    return maxi;
}
```

4.5 Minimum Edit Distance

```
// 利用 dfs 輸出替換字串的步驟
void backtracking(int i, int j){
    if(i == 0 || j == 0){
        while(i > 0){
            cout << cnt++ << " Delete " << i << endl;
            i--;
        }
        while(j > 0){
            cout << cnt++ << " Insert " << i + 1 <<
                "," << strB[j-1] << endl;
            j--;
        }
        return;
    }
    if(strA[i-1] == strB[j-1]){
        backtracking(i-1, j-1);
    }
    else{
        if(dis[i][j] == dis[i-1][j-1] + 1){
            cout << cnt++ << " Replace " << i << ","
                << strB[j-1] << endl;
            backtracking(i-1, j-1);
        }
        else if(dis[i][j] == dis[i-1][j] + 1){
            cout << cnt++ << " Delete " << i << endl;
            backtracking(i-1, j);
        }
        else if(dis[i][j] == dis[i][j-1] + 1){
            cout << cnt++ << " Insert " << i + 1 <<
                "," << strB[j-1] << endl;
            backtracking(i, j-1);
        }
    }
}
void MED(){
    // 由於 B 是 0 ，所以 A 轉換成 B
        時每個字元都要被刪除
    for(int i = 0; i <= strA.size(); ++i) dis[i][0] =
        i;
    // 由於 A 是 0 ，所以 A 轉換成 B
        時每個字元都需要插入
    for(int j = 0; j <= strB.size(); ++j) dis[0][j] = j;
    for(int i = 1; i <= strA.size(); ++i){
        for(int j = 1; j <= strB.size(); ++j){
            // 字元相同代表不需修改，修改距離直接延續
            if(strA[i-1] == strB[j-1]) dis[i][j] = dis[i-1][j-1];
            else{
                // 取 replace , delete , insert 最小，選其 +1 為最少編輯距離
                dis[i][j] = min(dis[i-1][j-1], min(dis[i-1][j], dis[i][j-1])) + 1;
            }
        }
    }
}
```

5 Enumerate

5.1 Halfcut Enumerate

```
/* 折半枚舉 */
void dfs(set<long long int> &s, int depth, int T, long long int sum){
    if(depth >= T){
        s.insert(sum);
        return;
    }
    dfs(s, depth + 1, T, sum); // 取或不取的概念
    dfs(s, depth + 1, T, sum + A[depth]);
}
int main(){
    int N, T;
    set<long long int> s1, s2;
    cin >> N >> T;
    for(int i = 0; i < N; ++i) cin >> A[i];
    dfs(s1, 0, N/2, 0); // 折半枚舉
    dfs(s2, N/2, N, 0);
    long long int ans = 0;
    // 題目:枚舉集合 Sx 的數字 Sxi，找出 Sy 集合內小於等於 T-Sxi 中最大的數 Syj
    for(auto &x : s1){
        auto it = s2.upper_bound(T - x);
        long long int y = *(--it);
        if(x + y <= T) ans = max(ans, x + y);
    }
    cout << ans << endl;
}
```

6 Graph

6.1 SPFA

```
bool SPFA(int s){
    // 記得初始化這些陣列
    int cnt[1000+5], dis[1000+5];
    bool inqueue[1000+5];
    queue<int> q;

    q.push(s);
    dis[s] = 0;
    inqueue[s] = true;
    cnt[s] = 1;
    while(!q.empty()){
        int now = q.front();
        q.pop();
        inqueue[now] = false;

        for(auto &e : G[now]){
            if(dis[e.t] > dis[now] + e.w){
                dis[e.t] = dis[now] + e.w;
                if(!inqueue[e.t]){
                    cnt[e.t]++;
                    if(cnt[e.t] > m){
                        return false;
                    }
                    inqueue[e.t] = true;
                    q.push(e.t);
                }
            }
        }
    }
    return true;
}
```

6.2 Dijkstra

```
/* Dijkstra 最短路徑 */
struct Edge{
    int v, w;
};
struct Item{
    int u, dis;
    // 取路徑最短
    bool operator < (const Item &other) const{
        return dis > other.dis;
    }
};
int dis[maxn];
vector<Edge> G[maxn];
void dijkstra(int s){
    for(int i = 0; i <= m; i++){
        dis[i] = inf;
    }
    dis[s] = 0;
    priority_queue<Item> pq;
    pq.push({s, 0});
    while(!pq.empty()){
        // 取路徑最短的點
        Item now = pq.top();
        pq.pop();
        if(now.dis > dis[now.u]){
            continue;
        }
        // 把與 now.u 相連的點都跑一遍
        for(Edge e : G[now.u]){
            if(dis[e.v] > now.dis + e.w){
                dis[e.v] = now.dis + e.w;
                pq.push({e.v, dis[e.v]});
            }
        }
    }
}
```

6.3 Floyd Warshall

```
void floyd_warshall(){
    for(int i = 0; i < n; i++){
        for(int j = 0; j < n; j++){
            G[i][j] = INF;
        }
        G[i][i] = 0;
    }
    for (int k = 0; k < n; k++){     // 嘗試每一個中繼點
        for (int i = 0; i < n; i++){ // 計算每一個i點與每一個j點
            for (int j = 0; j < n; j++){
                G[i][j] = min(G[i][j], G[i][k] + G[k][j]);
            }
        }
    }
}
```

6.4 Disjoint set Kruskal

```
struct Edge{
    int u, v, w;
    // 用權重排序 由大到小
    bool operator < (const Edge &other) const{
        return w > other.w;
    }
}edge[maxn];
// disjoint set
int find(int x){
  if(parent[x] < 0){
    return x;
  }
  else{
    return parent[x] = find(parent[x]);
  }
}
void unite(int a, int b){
  a = find(a);
  b = find(b);
  if(a != b){
    if(parent[a] < parent[b]){
      parent[a] += parent[b];
      parent[b] = a;
    }
    else{
      parent[b] += parent[a];
      parent[a] = b;
    }
  }
}
void kruskal(){
    memset(parent, -1, sizeof(parent));
    sort(edge, edge + m);
    int i, j;
    for(i = 0, j = 0; i < n - 1 && j < m; i++){
        // 如果 u 和 v 的祖先相同, 則 j++ (祖先相同代表會產生環 所以不要)
        while(find(edge[j].u) == find(edge[j].v)) j++;
        // 若部會產生環 則讓兩點之間產生橋 (連接兩顆子生成樹)
        unite(edge[j].u, edge[j].v);
        j++;
    }
}
```

6.5 Disjoint set Kruskal 2

```
struct Edge{
    int u, v;
    double w;
    bool operator < (const Edge &rhs) const{
        return w < rhs.w;
    }
}edge[maxn * maxn];
vector<Edge> G[maxn]; // 紀錄有哪些邊在 MST 上
int parent[maxn];
// disjoint set
int find(int x){
    return x == parent[x] ? x : parent[x] = find(parent[x]);
}
bool unite(int a, int b){
    int x = find(a);
    int y = find(b);
    if(x == y) return false;
    parent[x] = y;
    return true;
}
double kruskal(){
    m = 0; // m: 邊的數量
    for(int i = 0; i < n; ++i)
        for(int j = i + 1; j < n; ++j)
            edge[m++] = (Edge){i, j, dist(i, j)};
    sort(edge, edge + m);
    for(int i = 0; i < n; ++i){
        parent[i] = i;
        G[i].clear();
    }
    double total = 0.0;
    int edge_cnt = 0;
    for(int i = 0; i < m; ++i){
        int u = edge[i].u, v = edge[i].v;
        double cnt = edge[i].w;
        if(unite(u, v)){
            G[u].push_back((Edge){u, v, cnt});
            G[v].push_back((Edge){v, u, cnt});
            total += cnt;
            if(++edge_cnt == n-1) break;
        }
    }
    return total;
}
```

6.6 Bipatirate

```
/* 二分圖 */
const int maxn = 300 + 5;
int n, color[maxn];
vector<vector<int>> v(maxn);
bool dfs(int s){
    for(auto it : v[s]){
        if(color[it] == -1){
            color[it] = 3 - color[s];
            if(!dfs(it)){
                return false;
            }
        }
        if(color[s] == color[it]){
            return false;
        }
    }
    return true;
}
void isBipatirate(){
    bool flag = true;
    for(int i = 1; i <= n; ++i){
        if(color[i] == -1){
            color[i] = 1;
            flag &= dfs(i);
        }
    }
    if(flag){
        cout << "YES" << endl;
    }
    else{
        cout << "NO" << endl;
    }
}
int main(){
    while(cin >> n && n){
        for(int i = 1; i <= n; ++i) v[i].clear();
        memset(color, -1, sizeof(color));
        int a, b;
        while(cin >> a >> b && (a || b)){
            v[a].emplace_back(b);
            v[b].emplace_back(a);
        }
        isBipatirate();
    }
}
```

6.7 Hungarian algorithm

```
/* 匈牙利演算法 */
const int maxn = 500+5;
int t, N, bn, gn, match[maxn];
bool visited[maxn];
vector<vector<int>> G(maxn);
struct People{
    int h;
    string music, sport;
    People(){}
    People(int h, string music, string sport){
        this->h = h;
        this->music = music;
        this->sport = sport;
    }
}lef[maxn], rig[maxn];
bool check(People boy, People girl){
    if(abs(boy.h - girl.h) <= 40 && boy.music ==
        girl.music && boy.sport != girl.sport) return
        true;
    return false;
}
bool dfs(int s){
    for(int i = 0; i < G[s].size(); ++i){
        int v = G[s][i];
        if(visited[v]) continue;
        visited[v] = true;
        if(match[v] == -1 || dfs(match[v])){
            match[v] = s;
            return true;
        }
    }
    return false;
}
int Hungarian(){
    int cnt = 0;
    memset(match, -1, sizeof(match));
    for(int i = 0; i < bn; ++i){
        memset(visited, false, sizeof(visited));
        if(dfs(i)) cnt++;
    }
    return cnt;
}
int main(){
    cin >> t;
    while(t--){
        cin >> N;
        bn = 0, gn = 0;
        for(int i = 0; i <= N; ++i) G[i].clear();
        int h;
        string sex, music, sport;
        for(int i = 0; i < N; ++i){
            cin >> h >> sex >> music >> sport;
            if(sex == "M") lef[bn++] = People(h,
                music, sport);
            else rig[gn++] = People(h, music, sport);
        }
        for(int i = 0; i < bn; ++i){
            for(int j = 0; j < gn; ++j)
                if(check(lef[i], rig[j]))
                    G[i].emplace_back(j);
        }
        cout << N - Hungarian() << endl;
    }
}
```

6.8 LCA

```
/*最低共同祖先*/
// 此 node 下有幾顆 node
int dfs(int node, int dep){
    depth[node] = dep + 1;
    if(G[node].empty()){
        siz[node] = 1;
        return 1;
    }
    int total = 1;
    for(auto i : G[node])
        total += dfs(i.v, dep + 1);
    siz[node] = total;
    return siz[node];
}
// 找出每個節點的 2^i 倍祖先
// 2^20 = 1e6 > 200000
void find_parent(){
    for(int i = 1; i < 20; i++)
        for (int j = 0; j < N; j++)
            parent[j][i] =
                parent[parent[j][i-1]][i-1];
}
// 求兩點的LCA（利用倍增法）
int LCA(int a, int b){
    if (depth[b] < depth[a]) swap(a, b);
    if (depth[a] != depth[b]){
        int dif = depth[b] - depth[a];
        for (int i = 0; i < 20; i++){
            if (dif & 1) b = parent[b][i];
            dif >>= 1;
        }
    }
    if (a == b) return a;
    for (int i = 19; i >= 0; i--){
        if (parent[a][i] != parent[b][i]){
            a = parent[a][i];
            b = parent[b][i];
        }
    }
    return parent[a][0];
}
```

6.9 Trie

```
/* Trie 字典樹 */
struct Tire{
    int path;
    map<string, int> G[maxn];
    void init(){
        path = 1;
        G[0].clear();
    }
    void insert(string str){
        int u = 0;
        string word = "";
        for(int i = 0; i < str.size(); ++i){
            if(str[i] == '\\'){
                if(!G[u].count(word)){
                    G[path].clear();
                    G[u][word] = path++;
                }
                u = G[u][word];
                word = "";
            }
            else word += str[i];
        }
    }
    void put(int u, int space){
        for(auto i = G[u].begin(); i != G[u].end();
            ++i){
            for(int j = 0; j < space; ++j){
                cout << " ";
            }
            cout << i->first << endl;
            put(i->second, space + 1);
        }
    }
}tree;
```

7 Math

7.1 Hash

```
/* 建議搭配 Other - Stammering_Aliens 食用*/
#define ull unsigned long long int
const int maxn = 40000+5;
const ull seed = 131;
ull pw[maxn], hhash[maxn], hhash2[maxn];
char str[maxn];
void init(){
    hhash[0] = 0;
    for(int i = len-1; i >= 0; --i)
        hhash[i] = (hhash[i+1] * seed + str[i]);
}
```

8 Function

8.1 CHAR

```
isdigit()
isalnum() // 判斷字母 || 數字
isalpha()
islower()
isupper()
isblank() // 判斷 即 space 和 \t
toupper()
tolower()
```

8.2 string

```
int main(){
    string str;
    while(cin >> str){
        // substr 取 str idx 2~4 的值
        cout << str.substr(2, 4) << endl;
        // substr 取 str idx 2 以後的所有值
        cout << str.substr(2) << endl;

        string subst;
        cin >> subst;
        // str.append 連接字串
        cout << str.append(subst) << endl;

        char s[100], ss[100];
        cin >> s >> ss;

        char *p;
        // strstr 回傳在s裡找到ss後的整個字串(從 ss
            idx 0 到結束)
        p = strstr(s, ss);
        cout << p << endl;
        // strstr 也可以單純用來找字串
        if(p != NULL) cout << "yes" << endl;
        else cout << "no" << enld;
    }
}
```

8.3 setprecision

```
double cnt = 3.5555;
cout << fixed << setprecision(3) << cnt ;
```

8.4 GCD LCM

```
int gcd(int a, int b){
    return (b == 0 ? a : gcd(b, a % b));
}
int lcm(int a, int b){
    return a * b / gcd(a, b);
}

/* 輾轉相除法 - 求兩數是否互質
如果兩數互質 最終結果其中一方為0時 另一方必為1
若兩數有公因數 最終結果其中一方為0時 另一方必不為1 */
while ( ( num1 %= num2 ) != 0 && ( num2 %= num1 ) !=
    0 );
```

8.5 reverse

```
int a[10] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
reverse(a, a + 5);

vector<int> v;
reverse(v.begin(), v.end());

string str = "123";
reverse(str.begin(), str.end());
cout << str << endl; //321
```

8.6 sort

```
priority_queue<int, vector<int>, less<int>> // 大到小
priority_queue<int, vector<int>, greater<int>> //
    小到大

int arr[] = {4, 5, 8, 3, 7, 1, 2, 6, 10, 9};
sort(arr, arr+10);

vector<int> v;
sort(v.begin(), v.end()); //小到大

int cmp(int a, int b){
    return a > b;
}
sort(v.begin(), v.end(), cmp); //大到小
```

8.7 map

```
int main(){
    map<string, string> mp;
    map<string, string>::iterator iter;
    map<string, string>::reverse_iterator iter_r;

    mp.insert(pair<string, string>("r000", "zero"));

    mp["r123"] = "first";

    for(iter = mp.begin(); iter != mp.end(); iter++)
        cout<<iter->first<<" "<<iter->second<<endl;
    for(iter_r = mp.rbegin(); iter_r != mp.rend();
        iter_r++)
        cout<<iter_r->first<<"
            "<<iter_r->second<<endl;

    iter = mp.find("r123");
    mp.erase(iter);

    iter = mp.find("r123");
    if(iter != mp.end())
       cout<<"Find, the value is
           "<<iter->second<<endl;
    else
```

```
        cout<<"Do not Find"<<endl;

    mp.clear();
    mp.erase(mp.begin(), mp.end());
}
```

8.8 set

```
int main(){
    set<int> st {1, 6, 8}; // 直接初始化的寫法
    st.insert(1); // 也可以這樣寫就好
    set<int>::iterator iter;

    // 如果有找到，就會傳回正確的 iterator，否則傳回 st.end()
    if (iter != st.end()) {
        cout << "Found: " << *iter << endl;
    } else {
        cout << "Not found." << endl;
    }
    // cout: Found: 6

    // 取值：使用iterator
    x = *st.begin(); // set 中的第一個元素(最小的元素)
    x = *st.rbegin(); // set 中的最後一個元素(最大的元素)

    // search
    iter = st.find(6);
    auto it = st.find(x); // binary search, O(log(N))
    auto it = st.lower_bound(x); // binary search, O(log(N))
    auto it = st.upper_bound(x); // binary search, O(log(N))

    st.clear();
}
```

9 Other

9.1 Ants Colony

```
/* LCA 最低共同祖先 */
const int maxn = 1e5 + 5;
struct Edge{
    int v;
    int w;
};
int N;
vector<Edge> G[maxn];
int parent[maxn][20+5];
int depth[maxn], siz[maxn];
// 此 node 下有機顆 node
int dfs(int node, int dep){
    depth[node] = dep + 1;
    if(G[node].empty()){
        siz[node] = 1;
        return 1;
    }
    int total = 1;
    for(auto i : G[node])
        total += dfs(i.v, dep + 1);
    siz[node] = total;
    return siz[node];
}
// 找出每個節點的 2^i 倍祖先
// 2^20 = 1e6 > 200000
void find_parent(){
    for(int i = 1; i < 20; i++)
        for (int j = 0; j < N; j++)
            parent[j][i] = parent[parent[j][i-1]][i-1];
}
// 求兩點的LCA（利用倍增法）
int LCA(int a, int b){
    if (depth[b] < depth[a]) swap(a, b);
    if (depth[a] != depth[b]){
        int dif = depth[b] - depth[a];
        for (int i = 0; i < 20; i++){
            if (dif & 1) b = parent[b][i];
            dif >>= 1;
        }
    }
    if (a == b) return a;
    for (int i = 19; i >= 0; i--){
        if (parent[a][i] != parent[b][i]){
            a = parent[a][i];
            b = parent[b][i];
        }
    }
    return parent[a][0];
}
long long int dist[maxn];
// 從 0 開始到每個點的距離
void distance(){
    for (int u = 0; u < N; ++u){
        for(int i = 0; i < G[u].size(); ++i){
            dist[G[u][i].v] = dist[u] + G[u][i].w;
}
int main(){
    while(cin >> N && N){
        memset(dist, 0, sizeof(dist));
        memset(parent, 0, sizeof(parent));
        memset(depth, 0, sizeof(depth));
        memset(siz, 0, sizeof(siz));
        for(int i = 0; i <= N; ++i){
            G[i].clear();
        }
        for(int i = 1; i < N; ++i){
            int u, w;
            cin >> u >> w;
            G[u].push_back({i, w});
            parent[i][0] = u;
        }
        find_parent();
        dfs(0, 0);
        distance();
        int s; cin >> s;
        bool space = false;
        for(int i = 0; i < s; ++i){
            int a, b;
            cin >> a >> b;
            int lca = LCA(a, b);
            if(space) cout << " ";
            space = true;
            cout << (dist[a] + dist[b]) - (dist[lca] * 2);
        }
        cout << endl;
    }
}
```

9.2 Binary codes

```
/* BWT 資料轉換演算法 */
void BWT(){
    for(int i = 0; i < n; ++i){
        if(back[i] == 0){
            mini[zero++] = i;
    for(int i = 0; i < n; ++i){
        if(back[i] == 1){
            mini[zero++] = i;
    int ptr = mini[0];
    for(int i = 0; i < n; ++i){
        cout << back[ptr] << " ";
        ptr = mini[ptr];
    }
    cout << endl;
}
int main(){
    cin >> n;
    for(int i = 0; i < n; ++i){
        cin >> back[i];
    zero = 0;
    BWT();
}
```

9.3 Disk Tree

```
/* Trie 字典樹 */
const int maxn = 50000+5;
struct Tire{
    int path;
    map<string, int> G[maxn];
    void init(){
        path = 1;
        G[0].clear();
    }
    void insert(string str){
        int u = 0;
        string word = "";
        for(int i = 0; i < str.size(); ++i){
            if(str[i] == '\\'){
                if(!G[u].count(word)){
                    G[path].clear();
                    G[u][word] = path++;
                }
                u = G[u][word];
                word = "";
            }
            else word += str[i];
        }
    }
    void put(int u, int space){
        for(auto i = G[u].begin(); i != G[u].end();
            ++i){
            for(int j = 0; j < space; ++j)
                cout << " ";
            cout << i->first << endl;
            put(i->second, space + 1);
        }
    }
}tree;
int main(){
    int n;
    string str;
    while(cin >> n && n){
        tree.init();
        for(int i = 0; i < n; ++i){
            cin >> str;
            str += '\\';
            tree.insert(str);
        }
        tree.put(0, 0);
        cout << endl;
    }
}
```

10 Greedy

10.1 Sticks

```
/* Greedy + dfs */
const int maxn = 100+5;
int n, stickLengthSum, ans, stick[maxn];
bool visited[maxn];
bool dfs(int length, int idx, int stickTotal){
    if(length == ans){
        if(stickTotal == n) return true;
        length = 0;
    }
    if(length == 0){
        for(idx = 0; visited[idx]; idx++);
        visited[idx] = true;
        if(dfs(length + stick[idx], idx+1,
            stickTotal+1)) return true;
        visited[idx] = false;
    }
    else{
        for(int j = idx; j < n; ++j){
            if(visited[j] || (j && stick[j] ==
                stick[j-1] && !visited[j-1]))
                continue;
            if(stick[j] + length > ans) continue;
            visited[j] = true;
            if(dfs(length + stick[j], j+1,
                stickTotal+1)) return true;
            visited[j] = false;
            if(length + stick[j] == ans) return false;
        }
    }
    return false;
}
int main(){
    while(scanf("%d", &n) && n){
        stickLengthSum = 0;
        for(int i = 0; i < n; ++i){
            scanf("%d", &stick[i]);
            stickLengthSum += stick[i];
        }
        sort(stick, stick + n, greater<int>());
        for(ans = stick[0]; ans <= stickLengthSum;
            ans++){
            memset(visited, false, sizeof(visited));
            if(stickLengthSum % ans != 0) continue;
            if(dfs(0, 0, 0)) break;
        }
        printf("%d\n", ans);
    }
}
```

11 DP

11.1 Crested Ibis vs Monster

```
/* dp 背包 - 重量/價值/可重複使用
9 3
8 3
4 2
2 1
0 3 3 3 3 3 3 3 3 6
0 2 2 2 2 3 3 3 3 5
0 1 1 2 2 3 3 3 3 4
因為這題可以重複使用同一條魔法
所以可以這樣dp */
int a[10000+5], b[10000+5];
int dp[10000+5][10000+5];
int main(){
    int h, n;
    cin >> h >> n;
    for(int i = 1; i <= n; i++)
        cin >> a[i] >> b[i];
    memset(dp, 0x3f3f3f3f, sizeof(dp));
    dp[0][0] = 0;
    for(int i = 1; i <= n; i++)
        for(int j = 0; j <= h; j++)
            dp[i][j] = min(dp[i-1][j], dp[i][max(0, j
                - a[i])] + b[i]);
    cout << dp[n][h] << endl;
}
```

11.2 dpd Knapsack 1

```cpp
/* dp 背包 - 時間/數量/價值 - 第幾分鐘符合
w[i]: 3
陣列每一格代表的意義是最大上限為 index 時可以放入的最大 value
0 0 0 30 30 30 30 30 30
w[i]: 4
0 0 0 30 50 50 50 80 80
w[i]: 5
0 0 0 30 50 60 60 80 90  */
int main(){
    int N, W;
    cin >> N >> W;
    int w[100000+5], v[100000+5];
    for(int i = 0; i < N; i++)
        cin >> w[i] >> v[i];
    long long int dp[100000+5];
    memset(dp, 0, sizeof(dp));
    for(int i = 0; i < N; i++)
        for(int j = W; j >= w[i]; j--)
            dp[j] = max(dp[j], dp[j - w[i]] + v[i]);
    cout << dp[W] << endl;
}
```

12 Math

12.1 Big Mod

```python
'''
Mod
pow(x, y, z) = x^y % z
'''
# python 如何讀取直到 EOF 用 try except
try:
    while True:
        # input().split() 用空格切開讀取一整行
        # map (型態, input().split()) 才能把值全讀成 int
        B, P, M = map(int, input().split())
        print(pow(B, P, M))
except EOFError:
    exit
```

12.2 How Many Os

```cpp
/* 數論 */
int main(){
    long long int n, m;
    while(cin >> n >> m && (n >= 0) && (m >= 0)){
        long long int total1 = 0, total2 = 0;
        long long int ten = 1, tmp = n-1;
        while(tmp >= 10){
            if(tmp % 10 == 0){
                tmp /= 10;
                total1 += (tmp - 1) * ten + ((n-1) % ten) + 1;
            }
            else{
                tmp /= 10;
                total1 += tmp * ten;
            }
            ten *= 10;
        }
        ten = 1; tmp = m;
        while(tmp >= 10){
            if(tmp % 10 == 0){
                tmp /= 10;
                total2 += (tmp - 1) * ten + (m % ten) + 1;
            }
            else{
                tmp /= 10;
                total2 += tmp * ten;
            }
            ten *= 10;
        }
        if(n == 0) total1--;
        cout << total2 - total1 << endl;
    }
}
```

12.3 ORXOR

```cpp
/* bitwise operator 二進位制數論
如何切區段，之所以要1<<n是為了可以跑000~111
i = 0，binary i = 000
0 : 1 5 7
i = 1，binary i = 001
1 : 1 5 7
i = 2，binary i = 010，看得出來切了一刀
2 : 1 | 5 7
i = 3，binary i = 011
3 : 1 | 5 7
i = 4，binary i = 100，為了要切在index=2，所以才要1<<j
4 : 1 5 | 7
i = 5，binary i = 101
5 : 1 5 | 7
i = 6，binary i = 110
6 : 1 | 5 | 7
i = 7，binary i = 111
7 : 1 | 5 | 7
可以觀察出來，前兩位 bit 是 1 時代表的意義是切在哪裡 */
int main(){
    int n; cin >> n;
    int num[20+7];
    memset(num, 0, sizeof(num));
    for(int i = 1; i <= n; i++)
        cin >> num[i];
    // 不知道為甚麼只有 2147483647 給過
    int mini = 2147483647;
    // 1 << n = n * 2
    for(int i = 0; i < (1 << n); i++){
        int XOR = 0, OR = 0;
        for(int j = 1; j <= n; j++){
            OR |= num[j];
            if((i & (1 << j))){
                XOR ^= OR;
                OR = 0;
            }
        }
        XOR ^= OR;
        mini = min(mini, XOR);
    }
    cout << mini << endl;
}
```

13 Segement Tree

13.1 Frequent values

```cpp
/* Segement Tree & RMQ (Range Sum Query)
idx:  1   2   3   4   5   6   7   8   9   10
num: -1  -1   1   1   1   1   3  10  10  10
fre:  2   2   4   4   4   4   1   3   3   3
border
left: 1   1   3   3   3   3   7   8   8   8
right:2   2   6   6   6   6   7  10  10  10 */
# define Lson(x) x << 1
# define Rson(x) (x << 1) + 1
const int maxn = 1e5+5;
struct Tree{
    int lef, rig, value;
}tree[4 * maxn];
struct Num{
    int lef, rig, value, fre;
}num[maxn];
// 建立 segement tree
void build(int lef, int rig, int x){
    tree[x].lef = lef;
    tree[x].rig = rig;
    // 區塊有多長，題目詢問的重點
    if(lef == rig){
        tree[x].value = num[lef].fre;
        return;
    }
    int mid = (lef + rig) >> 1;
    build(lef, mid, Lson(x));
    build(mid + 1, rig, Rson(x));
    tree[x].value = max(tree[Lson(x)].value, tree[Rson(x)].value);
}
// 查詢 segement tree
int query(int lef, int rig, int x){
    // 題目所查詢的區間剛好在同個區塊上，num[lef].v == num[rig].v
    if(num[lef].value == num[rig].value) return rig - lef + 1;
    int ans = 0;
    // 查詢的左區間邊界切到區塊，且此區間有數個區塊
    if(lef > num[lef].lef){
        // 計算切到的區間大小
        ans = num[lef].rig - lef + 1;
        // 更新左邊界至被切區塊的右邊界加一，就不會切到區
        lef = num[lef].rig + 1;
    }
    // 查詢的右區間邊界切到區塊，且此區間有數個區塊
    if(rig < num[rig].rig){
        // 計算切到的區間大小，並找出最大
        ans = max(ans, rig - num[rig].lef + 1);
        // 更新右邊界
        rig = num[rig].lef - 1;
    }
    // 如果左邊界大於右邊界，表示不需要再進行查詢直接回傳
    if(lef > rig) return ans;
    if(tree[x].lef >= lef && tree[x].rig <= rig) return tree[x].value;
    int mid = (tree[x].lef + tree[x].rig) >> 1;
    if(lef <= mid) ans = max(ans, query(lef, rig, Lson(x)));
    if(mid < rig) ans = max(ans, query(lef, rig, Rson(x)));
    return ans;
}
int main(){
    int n, q;
    while(cin >> n && n){
        cin >> q;
        int start = 1;
        for(int i = 1; i <= n; ++i){
            cin >> num[i].value;
            if(num[i].value != num[i-1].value){
                for(int j = start; j < i; ++j){
                    num[j].rig = i - 1;
                    num[j].fre = i - start;
                }
                start = num[i].lef = i;
            }
            else num[i].lef = start;
        }
        // 最後一段 [start, n]
        for(int j = start; j <= n; ++j){
            num[j].rig = n;
            num[j].fre = n - start + 1;
        }
        build(1, n, 1);
        int lef, rig;
        for(int i = 0; i < q; ++i){
            cin >> lef >> rig;
            cout << query(lef, rig, 1) << endl;
        }
    }
}
```

14 Dijkstra

14.1 Walk Through the Forest

```
/* Dijkstra + 路徑最優化 DP */
const int inf = 0x3f3f3f3f;
const int maxn = 1000+5;
int n, m;
struct Edge{
    int v, w;
};
struct Item{
    int u, dis;
    bool operator < (const Item &other) const{
        return dis > other.dis;
    }
};
int dis[maxn];
long long int dp[maxn];
vector<Edge> G[maxn];
vector<int> path[maxn];
void dijkstra(int s){
    for(int i = 0; i <= n; ++i){
        dis[i] = inf;
    }
    dis[s] = 0;
    priority_queue<Item> pq;
    pq.push({s, 0});
    while(!pq.empty()){
        Item now = pq.top();
        pq.pop();

        if(now.dis > dis[now.u]){
            continue;
        }

        for(Edge e: G[now.u]){
            if(dis[e.v] > now.dis + e.w){
                dis[e.v] = now.dis + e.w;
                pq.push({e.v, dis[e.v]});
            }
        }
    }
}
long long int dfs(int u){
    // ans 是 pointer，指向 dp[u] 的記憶體位址
    // 對於 ans 的 value 改變會記錄在 dp[u]
    long long int& ans = dp[u];
    if(ans != -1) return ans;
    if(u == 2) return ans = 1;
    ans = 0;
    for(int i = 0; i < path[u].size(); ++i)
        ans += dfs(path[u][i]);
    return ans;
}
int main(){
    while(cin >> n && n){
        cin >> m;
        for(int i = 0; i <= n; ++i) G[i].clear();
        int u, v, w;
        for(int i = 0; i < m; ++i){
            cin >> u >> v >> w;
            G[u].push_back({v, w});
            G[v].push_back({u, w});
        }
        dijkstra(2); // dijkstra 紀錄從終點到每個點的距離
        memset(dp, -1, sizeof(dp));
        for(int i = 1; i <= n; ++i){
            path[i].clear();
            for(int j = 0; j < G[i].size(); ++j){
                int v = G[i][j].v;
                // 如果到 v 的距離比到 i 遠，代表從起點經過 i 再到 v
                if(dis[i] > dis[v])
                    path[i].push_back(v);
            }
        }
        cout << dfs(1) << endl;
    }
}
```

15 Kruskal

15.1 Qin Shi Huang Road System

```
/* kruskal disjoint set dfs */
const int maxn = 1000 + 5;
int n, m;
int x[maxn], y[maxn], p[maxn];
struct Edge{
    int u, v;
    double w;
    bool operator < (const Edge &rhs) const{
        return w < rhs.w;
    }
}edge[maxn * maxn];
vector<Edge> G[maxn];
int parent[maxn];
// 計算兩點之間的距離
double dist(int a, int b){
    double x2 = (x[a] - x[b]) * (x[a] - x[b]);
    double y2 = (y[a] - y[b]) * (y[a] - y[b]);
    return sqrt(x2 + y2);
}
// disjoint set
int find(int x){
    return x == parent[x] ? x : parent[x] = find(parent[x]);
}
bool unite(int a, int b){
    int x = find(a);
    int y = find(b);
    if(x == y) return false;
    parent[x] = y;
    return true;
}
double kruskal(){
    m = 0; // m: 邊的數量
    for(int i = 0; i < n; ++i)
        for(int j = i + 1; j < n; ++j)
            edge[m++] = (Edge){i, j, dist(i, j)};
    sort(edge, edge + m);
    for(int i = 0; i < n; ++i){
        parent[i] = i;
        G[i].clear();
    }
    double total = 0.0;
    int edge_cnt = 0;
    for(int i = 0; i < m; ++i){
        int u = edge[i].u, v = edge[i].v;
        double cnt = edge[i].w;
        if(unite(u, v)){
            G[u].push_back((Edge){u, v, cnt});
            G[v].push_back((Edge){v, u, cnt});
            total += cnt;
            if(++edge_cnt == n-1) break;
        }
    }
    return total;
}
double maxcost[maxn][maxn];
bool visited[maxn];
void dfs(int u){
    visited[u] = true;
    for(int i = 0; i < G[u].size(); ++i){
        int v = G[u][i].v;
        if(visited[v]) continue;
        double cost = G[u][i].w;
        maxcost[u][v] = maxcost[v][u] = cost;
        // 更新 MST 樹上的點到 v 點的距離
        for(int j = 0; j < n; ++j)
            if(visited[j])
                maxcost[j][v] = maxcost[v][j] = max(maxcost[j][u], cost);
        dfs(v);
    }
}
void solve(){
    double total = kruskal();
    memset(maxcost, 0, sizeof(maxcost));
    memset(visited, false, sizeof(visited));
    dfs(0);
    double ans = -1;
    // 把所有點都遍歷一次
    for(int i = 0; i < n; ++i)
        for(int j = i + 1; j < n; ++j)
            ans = max(ans, (p[i] + p[j]) / (total - maxcost[i][j]));
    printf("%.2lf\n", ans);
}
int main(){
    int t;
    scanf("%d", &t);
    while(t--){
        scanf("%d", &n);
        for(int i = 0; i < n; ++i)
            scanf("%d%d%d", &x[i], &y[i], &p[i]);
        solve();
    }
    return 0;
}
```

16 Bipartite Graph

16.1 SAM I AM

```
/* 二分圖匹配 + 最小點覆蓋 */
const int maxn = 1000+5;
int R, C, N;
bool arr[maxn][maxn], visitX[maxn], visitY[maxn];
int matchX[maxn], matchY[maxn];
int dfs(int x){
    visitX[x] = true;
    for(int y = 1; y <= C; ++y){
        if(arr[x][y] && !visitY[y]){
            visitY[y] = true;
            if(matchY[y] == 0 || dfs(matchY[y])){
                matchX[x] = y;
                matchY[y] = x;
                return 1;
            }
        }
    }
    return 0;
}
int Match(){
    int sum = 0;
    memset(matchX, 0, sizeof(matchX));
    memset(matchY, 0, sizeof(matchY));
    for(int i = 1; i <= R; ++i){
        memset(visitX, false, sizeof(visitX));
        memset(visitY, false, sizeof(visitY));
        sum += dfs(i);
    }
    return sum;
}
int main(){
    while(cin >> R >> C >> N && R && C && N){
        memset(arr, false, sizeof(arr));
        memset(visitX, false, sizeof(visitX));
        memset(visitY, false, sizeof(visitY));
        int row, col;
        for(int i = 0; i < N; ++i){
            cin >> row >> col;
            arr[row][col] = true;
        }
        int cnt = Match();
        cout << cnt;
        memset(visitX, 0, sizeof(visitX));
        memset(visitY, 0, sizeof(visitY));
        for(int i = 1; i <= R; ++i){
            if(matchX[i] == 0) dfs(i);
        for(int i = 1; i <= R; ++i)
            if(!visitX[i]) cout << " r" << i;
        for(int i = 1; i <= C; ++i)
            if(visitY[i]) cout << " c" << i;
        cout << endl;
    }
}
```