Contents

1 Sync 1
1.1 Sync . . . 1

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

3 Divide and Conquer 2
3.1 count inversions . . . 2

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 . . . 4
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 Other 6
8.1 Ants Colony . . . 6
8.2 Binary codes . . . 6
8.3 Fire Fire Fire . . . 7
8.4 Disk Tree . . . 7
8.5 Stammering Aliens . . . 7
8.6 Fabled Rooks . . . 8
8.7 Rails . . . 8
8.8 String Distance and Transform Process . . . 9

9 Greedy 9
9.1 Sticks . . . 9

10 DP 10
10.1 Crested Ibis vs Monster . . . 10
10.2 dpd Knapsack 1 . . . 10
10.3 Homer Simpson . . . 10
10.4 Let Me Count The Ways . . . 10
10.5 Luggage . . . 10
10.6 Partitioning by Palindromes . . . 11
10.7 SuperSale . . . 11
10.8 Walking on the Safe Side . . . 11
10.9 Cutting Sticks . . . 12
10.10 Race to 1 . . . 12
10.11 Apple . . . 12
10.12 Stamps . . . 12
10.13 Evacuation Plan . . . 13
10.14 Ladies Choice . . . 13

11 LIS 14
11.1 Wavio Sequence . . . 14
11.2 Robots II . . . 14

12 Math 15
12.1 Big Mod . . . 15
12.2 Bubble Sort Expect Value . . . 15
12.3 Fraction Floor Sum . . . 15
12.4 How Many Os . . . 15
12.5 Number of Pairs . . . 15
12.6 ORXOR . . . 16
12.7 X drawing . . . 16
12.8 Playing With Stones . . . 16
12.9 And Then There Was One . . . 16

13 Binary Search 16
13.1 Fill the Containers . . . 16
13.2 Where is the marble . . . 17

14 Graph 17
14.1 Maximum sum on a torus . . . 17

15 Segement Tree 17
15.1 Frequent values . . . 17

16 Dijkstra 18
16.1 Airport Express . . . 18
16.2 Walk Through the Forest . . . 19

17 Kruskal 19
17.1 Qin Shi Huang Road System . . . 19

18 Bipartite Graph 20
18.1 Claw Decomposition . . . 20
18.2 Guardian of Decency . . . 20
18.3 Taxi Cab Scheme . . . 21
18.4 SAM I AM . . . 21

19 Function 22
19.1 CHAR . . . 22
19.2 string . . . 22
19.3 setprecision . . . 22
19.4 GCD LCM . . . 22
19.5 reverse . . . 22
19.6 sort . . . 22
19.7 map . . . 22
19.8 set . . . 22

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

```cpp
/* 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

```cpp
/* 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

```cpp
/* 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

```cpp
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

```cpp
// 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];
}
```

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 Other

8.1 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;
    }
}
```

8.2 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();
}
```

8.3 Fire Fire Fire

```
/* dfs
只要我有一個小孩不是防火牆，我就必須是防火牆 */
const int maxn = 1000+5;
int cnt = 0;
vector<int> G[maxn];
bool exi[maxn], visited[maxn];
void dfs(int node, int parent){
    if(G[node].size() == 1 && G[node][0] == parent)
        return;
    for(int i = 0; i < G[node].size(); ++i){
        int now = G[node][i];
        if(visited[now]) continue;
        visited[now] = true;
        dfs(G[node][i], node);
    }
    bool flag = false;
    for(int j = 0; j < G[node].size(); ++j){
        if(exi[G[node][j]] != true && G[node][j] !=
            parent){
            flag = true;
            break;
        }
    }
    if(flag && exi[node] != true){
        exi[node] = true;
        cnt++;
    }
    return;
}
int main(){
    int n;
    while(cin >> n && n){
        for(int i = 1; i <= n; ++i) G[i].clear();
        memset(exi, false, sizeof(exi));
        memset(visited, false, sizeof(visited));
        for(int i = 1; i <= n; ++i){
            int siz; cin >> siz;
            for(int j = 0; j < siz; ++j){
                int num; cin >> num;
                G[i].emplace_back(num);
            }
        }
        cnt = 0;
        dfs(1, 1);
        if(n == 1) cnt++;
        cout << cnt << endl;
    }
}
```

8.4 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;
    }
}
```

8.5 Stammering Aliens

```
/* hash 字串 + 二分搜尋 */
#define ull unsigned long long int
const int maxn = 40000+5;
const ull seed = 131;
ull pw[maxn], hhash[maxn], hhash2[maxn];
int m, len;
char str[maxn];
map<ull, int> mp;
void init(){
    hhash[0] = 0;
    for(int i = len-1; i >= 0; --i){
        hhash[i] = (hhash[i+1] * seed + str[i]);
    }
}
int check(int x){
    for(int i = 0; i + x - 1 < len; ++i){
        ull tmp = hhash[i] - (hhash[i + x] * pw[x]);
        hhash2[i] = tmp;
    }
    sort(hhash2, hhash2 + len - x + 1);
    int cnt = 0;
    for(int i = 0; i < len - x + 1; ++i){
```

```
        if(i && hhash2[i] == hhash2[i-1])
            cnt++;
        else{
            if(cnt >= m) return 1;
            cnt = 1;
        }
    }
    if(cnt >= m) return 1;
    return 0;
}
int main(){
    pw[0] = 1;
    for(int i = 1; i < maxn; ++i)
        pw[i] = (pw[i-1] * seed);
    while(scanf("%d", &m) && m){
        scanf("%s", str);
        len = strlen(str);
        init();
        int lef = 1, rig = len + 1;
        while(lef < rig){
            int mid = (lef + rig) >> 1;
            if(check(mid))
                lef = mid + 1;
            else rig = mid;
        }
        int ans = rig - 1;
        if(!ans){
            puts("none");
            continue;
        }
        int pos;
        mp.clear();
        for(int i = 0; i + ans - 1 < len; ++i){
            ull tmp = hhash[i] - hhash[i + ans] *
                pw[ans];
            mp[tmp]++;
            if(mp[tmp] >= m) pos = i;
        }
        printf("%d %d\n", ans, pos);
    }
    return 0;
}
```

8.6 Fabled Rooks

```
/* 特定排序後放入格子
以右邊大小排序 要從左邊開始放
以左邊大小排序 要從右邊開始放 */
int n;
const int maxn = 5000+5;
struct Edge{
    int lef, rig, pos, idx;
    bool operator < (const Edge &rhs) const{
        if(rig != rhs.rig)
            return rig < rhs.rig;
        else
            return lef < rhs.lef;
    }
}x[maxn], y[maxn];
bool used[maxn];
bool solve_x(){
    memset(used, false, sizeof(used));
    for(int i = 0; i < n; ++i){
        x[i].pos = 0;
        for(int j = x[i].lef; j <= x[i].rig; ++j){
            if(!used[j]){
                x[i].pos = j;
                used[j] = true;
                break;
            }
        }
        if(x[i].pos == 0) return false;
    }
    return true;
}
bool solve_y(){
    memset(used, false, sizeof(used));
    for(int i = 0; i < n; ++i){
        y[i].pos = 0;
        for(int j = y[i].lef; j <= y[i].rig; ++j){
            if(!used[j]){
                y[i].pos = j;
                used[j] = true;
                break;
            }
        }
        if(y[i].pos == 0) return false;
    }
    return true;
}
int main(){
    while(cin >> n && n){
        int x1, y1, x2, y2;
        for(int i = 0; i < n; ++i){
            cin >> x1 >> y1 >> x2 >> y2;
            x[i].lef = min(x1, x2);
            x[i].rig = max(x1, x2);
            y[i].lef = min(y1, y2);
            y[i].rig = max(y2, y2);
            x[i].idx = y[i].idx = i;
            x[i].pos = y[i].pos = 0;
        }
        sort(x, x + n);
        sort(y, y + n);
        if(!solve_x() || !solve_y()) cout <<
            "IMPOSSIBLE" << endl;
        else{
            int ans_x[maxn], ans_y[maxn];
            for(int i = 0; i < n; ++i){
                ans_x[x[i].idx] = x[i].pos;
                ans_y[y[i].idx] = y[i].pos;
            }
            for(int i = 0; i < n; ++i)
                cout << ans_x[i] << " " << ans_y[i]
                    << endl;
        }
    }
}
```

8.7 Rails

```
/* deque 火車
倒退逆推法 能怎樣進來就能怎樣出去
lis: 1 2 3 4 5
dq:  3 2 4 1 5
1. 如果 lis front = dq front，dq pop
2. 反之 ans.push dq front，每次檢查 ans top 是否 =
    lis front
*/
int main(){
    int n;
    while(cin >> n && n){
        int train;
        deque<int> dq;
        while(cin >> train && train){
            dq.emplace_back(train);
            deque<int> lis, ans;
            for(int i = 2; i <= n; ++i){
                cin >> train;
                dq.emplace_back(train);
            }
            for(int i = 1; i <= n; ++i)
                lis.emplace_back(i);
            for(int i = 0, j = 0; j < n, i < n; ++i){
                if(lis[i] == dq[j]) ++j;
                else
                    ans.emplace_back(lis[i]);
                while(!ans.empty()){
                    if(dq[j] != ans.back()) break;
                    ans.pop_back();
```

```
                    ++j;
                }
            }
            if(!ans.empty())
                cout << "No" << endl;
            else
                cout << "Yes" << endl;
            dq.clear();
        }
        cout << endl;
    }
}
```

8.8 String Distance and Transform Process

```
/* MED - Minimum Edit Distance
增加刪除修改 使得字串A 以最小步驟數替換成 字串B
abcac
bcd
  j 0 b c d
i +--------
0 | 0 1 2 3
a | 1 1 2 3
b | 2 1 2 3
c | 3 2 1 2
a | 4 3 2 2
c | 5 4 3 3
1 Delete 1
2 Replace 3,d
3 Delete 4 */
const int maxn = 80+5;
string strA, strB;
int dis[maxn][maxn];
int cnt;
// 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;
            }
        }
    }
}
int main(){
    bool space = false;
    while(getline(cin, strA) && getline(cin, strB)){
        cnt = 1;
        MED();
        if(space) cout << endl;
        space = true;
        cout << dis[strA.size()][strB.size()] << endl;
        backtracking(strA.size(), strB.size());
    }
}
```

9 Greedy

9.1 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);
    }
}
```

10 DP

10.1 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;
}
```

10.2 dpd Knapsack 1

```
/* 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;
}
```

10.3 Homer Simpson

```
/* dp 背包 - 時間/數量 - 漢堡
3 5 54
吃 3 分鐘漢堡時
0 -1 -1 1 -1 -1 2 -1 -1 3 -1 -1 4 -1 -1 5 -1 -1 6 -1
    -1 7 -1 -1 8 -1 -1 9 -1 -1 10 -1 -1 11 -1 -1 12
    -1 -1 13 -1 -1 14 -1 -1 15 -1 -1 16 -1 -1 17 -1
    -1 18
吃 5 分鐘漢堡時 (更新)
0 -1 -1 1 -1 1 2 -1 2 3 2 3 4 3 4 5 4 5 6 5 6 7 6 7 8
    7 8 9 8 9 10 9 10 11 10 11 12 11 12 13 12 13 14
    13 14 15 14 15 16 15 16 17 16 17 18
只有當該時間可剛好吃滿漢堡時會更新
全部初始設 -1，用以判斷 譬如當 1 分鐘時
    吃不了任何漢堡*/
int main(){
    int m, n, t;
    while(cin >> m >> n >> t){
        int dp[10000+5];
        memset(dp, -1, sizeof(dp));
        dp[0] = 0;
        for(int i = m; i <= t; i++)
            if(dp[i - m] != -1)
                dp[i] = max(dp[i], dp[i - m] + 1);
        for(int i = n; i <= t; i++)
            if(dp[i - n] != -1)
                dp[i] = max(dp[i], dp[i - n] + 1);
        // 時間無法剛好吃滿的時候
        if(dp[t] == -1){
            for(int i = t; i >= 0; i--)
                if(dp[i] != -1){
                    cout << dp[i] << " " << t - i <<
                        endl;
                    break;
                }
        }
        else cout << dp[t] << endl;
    }
}
```

10.4 Let Me Count The Ways

```
/* dp - 時間/數量 - 硬幣排序
要湊出 17
1 1 1 1 1 2 2 2 2 2 4 4 4 4 4 6 6 */
int main(){
    long long int n;
    long long int dp[30000+5];
    int coin[] = {1, 5, 10, 25, 50};
    memset(dp, 0, sizeof(dp));
    // 直接把 dp 做好
    dp[0] = 1;
    for(int i = 0; i < 5; i++)
        for(int j = coin[i]; j < 30000+5; j++)
            if(dp[j - coin[i]] != -1)
                dp[j] += dp[j - coin[i]];
    while(cin >> n){
        if(dp[n] == 1)
            cout << "There is only " << dp[n] << "
                way to produce " << n << " cents
                change." << endl;
        else
            cout << "There are " << dp[n] << " ways
                to produce " << n << " cents change."
                << endl;
    }
}
```

10.5 Luggage

```
/* dp 背包 - 重量/是否成立
7 7 13 1
1 1 0 0 0 0 0 1 1 0 0 0 0 0 1 1 1 0 0 0 0 1 1 0 0 0 0 0
    1
Note: dp[0] = true */
int main(){
    int t;
```

```
    cin >> t;
    cin.ignore();
    while(t--){
        string str;
        getline(cin , str);
        vector<int> v;
        stringstream ss;
        int num, cnt = 0, sum = 0;;
        bool dp[4000+5];
        memset(dp, false, sizeof(dp));
        ss << str;
        while(ss >> num){
            cnt++;
            sum += num;
            v.emplace_back(num);
        }
        if(sum & 1){
            cout << "NO" << endl;
            continue;
        }
        dp[0] = true;
        for(int i = 0; i < v.size(); i++)
            for(int j = sum; j >= v[i]; j--)
                if(dp[j - v[i]])
                    dp[j] = true;
        cout << (dp[sum/2] ? "YES" : "NO") << endl;
    }
}
```

10.6 Partitioning by Palindromes

```
/* string & dp - 字串長度判斷迴文
r a c e c a r
i = 0, j = 0
-> r = r ，dp[1] = dp[0] + 1 = 1
i = 1, j = 0
-> 因 a != r，dp[2] = 0x3f3f3f3f
i = 1, j = 1
-> 因 a = a, dp[2] = dp[1] + 1 = 2 */
bool check_palindromes(int lef, int rig){
    // 比較字串兩端都是迴文
    while(lef < rig){
        if(str[lef] != str[rig]) return 0;
        lef++;
        rig--;
    }
    return 1;
}
int main(){
    int t;
    cin >> t;
    while(t--){
        cin >> str;
        memset(dp, 0x3f3f3f3f, sizeof(dp));
        dp[0] = 0;
        for(int i = 0; i < str.size(); ++i)
            for(int j = 0; j <= i; ++j)
                if(str[i] == str[j])
                    if(check_palindromes(j, i))
                        if(dp[i+1] > dp[j] + 1)
                            dp[i+1] = dp[j] + 1;
        cout << dp[str.size()] << endl;
    }
}
```

10.7 SuperSale

```
/* dp 背包 - 重量/價值/不可重複使用
第一個人的負重: 23
0 0 0 0 52 52 52 52 52 54 54 54 54 106 106 106 106
    106 106 106 106 106 151 151
第二個人的負重: 20
0 0 0 0 52 52 52 52 52 54 54 54 54 106 106 106 106
    106 106 106 106
第三個人的負重: 20
0 0 0 0 52 52 52 52 52 54 54 54 54 106 106 106 106
    106 106 106 106
第四個人的負重: 26
0 0 0 0 52 52 52 52 52 54 54 54 54 106 106 106 106
    106 106 106 106 106 151 151 151 151 151 */
struct Edge{
    int p;
    int w;
}edge[1000+5];
int main(){
    int t;
    cin >> t;
    while(t--){
        int n; cin >> n;
        for(int i = 0; i < n; i++)
            cin >> edge[i].p >> edge[i].w;
        int g, total = 0;
        cin >> g;
        for(int i = 0; i < g; i++){
            int pw; in >> pw;
            int dp[30+5];
            memset(dp, 0, sizeof(dp));
            for(int j = 0; j < n; j++)
                for(int k = pw; k >= edge[j].w; k--)
                    dp[k] = max(dp[k], dp[k -
                        edge[j].w] + edge[j].p);
            total += dp[pw];
        }
        cout << total << endl;
    }
}
```

10.8 Walking on the Safe Side

```
/* dp - 地圖更新
更新地圖
一張如下的地圖 其 dp 更新方法為加上和加左的路
0 0 0 0 0
0 1 0 0 0
0 0 1 0 1
0 0 0 0 0
1 1 1 1 1
1 0 1 2 3
1 1 0 2 0
1 2 2 4 4 */
bool mp[100+5][100+5];
long long int dp[100+5][100+5];
int main(){
    int t; cin >> t;
    bool space = false;
    while(t--){
        if(space) cout << endl;
        else space = true;
        int r, c; cin >> r >> c;
        cin.ignore();
        memset(mp, false, sizeof(mp));
        memset(dp, 0, sizeof(dp));
        string str;
        for(int i = 0; i < r; i++){
            getline(cin, str);
            int n, num;
            stringstream ss(str);
            ss >> n;
            while(ss >> num)
                mp[n][num] = true;
        }
        dp[1][1] = 1;
        for(int i = 1; i <= r; i++){
            for(int j = 1; j <= c; j++){
                if(mp[i][j]) continue;
                if(i > 1)
                    dp[i][j] += dp[i-1][j];
```

```
                if(j > 1)
                    dp[i][j] += dp[i][j-1];
            }
        }
        cout << dp[r][c] << endl;
    }
}
```

10.9 Cutting Sticks

```
/* dp - 動態切割取最小
100
3
25 50 75
dp:
0 0 50 125 200
0 0 0 50 125
0 0 0 0 50
0 0 0 0 0
0 0 0 0 0 */
int main(){
    int l;
    while(cin >> l && l){
        int n;
        cin >> n;
        vector<int> s(n+2);
        s[0] = 0;
        for(int i = 1; i <= n; ++i)
            cin >> s[i];
        // 從現在開始 n 的數量變為 n + 1
        s[++n] = l;
        int dp[n+5][n+5];
        memset(dp, 0, sizeof(dp));
        // r: 切幾段 b: 起點 c: 中間點 e: 終點
        for(int r = 2; r <= n; ++r){
            for(int b = 0; b < n; ++b){
                // 如果從 b 開始切 r 刀會超出長度就
                    break
                if(b + r > n) break;
                // e: 從 b 開始切 r 刀
                int e = b + r;
                dp[b][e] = 0x3f3f3f3f;
                // c: 遍歷所有從 b 開始到 e
                    結束的中間點
                for(int c = b + 1; c < e; ++c){
                    // dp[b][c] 從 b 到 c 最少 cost +
                        dp[c][e] 從 c 到 e 最少 cost
                    // s[e] - s[b] 兩段之間的 cost
                    dp[b][e] = min(dp[b][e], dp[b][c]
                        + dp[c][e] + s[e] - s[b]);
                }
            }
        }
        cout << "The minimum cutting is " << dp[0][n]
            << "." << endl;
    }
}
```

10.10 Race to 1

```
/* dp - 數量
期望值、質數、dfs */
const int N = 1000000;
bool sieve[N+5];
vector<int> pri;
double dp[N+5];
// 線性篩
void Linear_Sieve(){
    for (int i = 2; i < N; i++){
        if (!sieve[i])
            pri.push_back(i);
        for (int p: pri){
            if (i * p >= N) break;
            sieve[i * p] = true;
            if (i % p == 0) break;
        }
    }
}
double dfs(int n){
    if(dp[n] != -1) return dp[n];
    dp[n] = 0;
    if(n == 1) return dp[n];
    int total = 0, prime = 0;
    for(int i = 0; i < pri.size() && pri[i] <= n;
        i++){
        total++;
        if(n % pri[i]) continue;
        prime++;
        dp[n] += dfs(n/pri[i]);
    }
    // 算期望值
    dp[n] = (dp[n] + total)/prime;
    return dp[n];
}
int main(){
    int t, num, ca = 1;
    for(int i = 0; i <= N; i++)
        dp[i] = -1;
    Linear_Sieve();
    cin >> t;
    while(t--){
        cin >> num;
        cout << "Case " << ca++ << ": " << fixed <<
            setprecision(10) << dfs(num) << endl;
    }
}
```

10.11 Apple

```
/* dp - 數量
col = 蘋果 n
row = 盤子 m
* 0 1 2 3 4
1 1 1 1 1 1
2 1 1 2 2 3
3 1 1 2 3 4 */
int dp[10+5];
int main(){
  int t; cin >> t;
  while(t--){
    int n, m;
    cin >> m >> n;
    memset(dp, 0, sizeof(dp));
    dp[0] = 1;
    for(int i = 1; i <= n; ++i)
      for(int j = i; j <= m; ++j)
        dp[j] += dp[j - i];
    cout << dp[m] << endl;
  }
}
```

10.12 Stamps

```
/* dp - dfs/分配可能性並更新 */
const int maxn = 100+5;
int h, k, r, maxi = 0;
int x[maxn], y[maxn];
int ans[maxn]; // 存可貼出最大郵票值的面額
void dfs(int i){
    // 若 x[i] 的 i 多於可貼的郵票數量
    if(i >= k){
        if(r > maxi){
            maxi = max(maxi, r);
            for(int i = 0; i < k; ++i)
                ans[i] = x[i];
```

```
        }
        return;
    }
    // 存此層尚未更新前的 r、y 值，因為 dfs 完要回去上一層
    int r_before_this_layer = r;
    int y_before_this_layer[maxn];
    for(int j = 0; j < maxn; ++j)
        y_before_this_layer[j] = y[j];
    // next: 下一可考慮的郵票面額
    // postage: 貼完郵票的總面額(y的idx)
    // num: 要貼幾張
    // x[i-1] 要 -1 是因為 x 從 0 開始存第一種面額
    for(int next = x[i-1] + 1; next <= r + 1; ++next){
        x[i] = next;
        for(int postage = 0; postage < x[i-1] * h;
            ++postage){
            if(y[postage] >= h) continue;
            for(int num = 1; num <= h - y[postage];
                ++num)
                if(y[postage] + num < y[postage + num
                    * next] && (postage + num * next
                    < maxn))
                    y[postage + num * next] =
                        y[postage] + num;
        }
        // 更新現在連續最大值到多少
        while(y[r+1] < 0x3f3f3f) r++;
        // x 可貼面額種類多 1
        dfs(i+1);
        // 還原 r、y 值
        r = r_before_this_layer;
        for(int j = 0; j < maxn; ++j)
            y[j] = y_before_this_layer[j];
    }
}
int main(){
    while(cin >> h >> k && h && k){
        memset(x, 0, sizeof(x));
        memset(y, 0x3f3f3f3f, sizeof(y));
        x[0] = 1;
        r = h;
        // x[0] = 1，1 張郵票可貼到的最大值
        for(int i = 0; i <= r; ++i)
            y[i] = i;
        maxi = 0;
        dfs(1);
        for(int i = 0; i < k; ++i)
            printf("%3d", ans[i]);
        printf(" ->%3d\n", maxi);
    }
}
```

10.13 Evacuation Plan

```
/* dp - 路徑/隊伍分配救難所 */
const int maxn = 4000+5;
int path[maxn][maxn];
long long int dp[maxn][maxn];
struct Edge{
    int idx, position;
    bool operator < (const Edge &rhs) const{
        return position < rhs.position;
    }
}team[maxn], shelter[maxn];
int main(){
    int n;
    while(cin >> n){
        for(int i = 1; i <= n; ++i){
            cin >> team[i].position;
            team[i].idx = i;
        }
        sort(team + 1, team + n + 1);
        int m; cin >> m;
        for(int i = 1; i <= m; ++i){
            cin >> shelter[i].position;
            shelter[i].idx = i;
        }
        sort(shelter + 1, shelter + m + 1);
        memset(dp, 0x3f3f3f3f, sizeof(dp));
        dp[1][0] = 0;
        for(int i = 1; i <= m; ++i){
            for(int j = i; j <= n; ++j){
                // dp[i][j] = min(dp[i][j-1],
                    dp[i-1][j-1]) +
                    abs(team[j].position -
                    shelter[i].position);
                if(dp[i][j-1] <= dp[i-1][j-1]){
                    dp[i][j] = min(dp[i][j-1],
                        dp[i-1][j-1]) +
                        abs(team[j].position -
                        shelter[i].position);
                    path[i][j] = 0; //
                        從左邊來，前面的 teams 有人來
                        j shelter
                }
                else{
                    dp[i][j] = min(dp[i][j-1],
                        dp[i-1][j-1]) +
                        abs(team[j].position -
                        shelter[i].position);
                    path[i][j] = 1; //
                        從左上來，前面的 teams 不會來
                        j shelter
                }
            }
        }
        int now_shelter = m;
        int ans[maxn];
        //
            紀錄路徑，若從左邊來，上一隊也來此；若從右邊來，上隊
        for(int i = n; i > 0; --i){
            ans[team[i].idx] =
                shelter[now_shelter].idx;
            now_shelter -= path[now_shelter][i];
        }
        cout << dp[m][n] << endl;
        for(int i = 1; i < n; ++i)
            cout << ans[i] << " ";
        cout << ans[n] << endl;
    }
}
```

10.14 Ladies Choice

```
/* dp - ladies & men */
const int maxn = 1000+5;
int n;
int man[maxn][maxn], manidx[maxn], lady[maxn][maxn],
    ladyidx[maxn];
int dp[maxn];
deque<int> dq;
void dp_func(){
    while(!dq.empty()){
        int man_now = dq.front();
        dq.pop_front();
        // manidx 現在指著的 lady
        int lady1 = manidx[man_now];
        // man 目前最想要的 lady
        int lady_first = man[man_now][lady1];
        // ladyidx 現在指著的 man
        int man1 = ladyidx[lady_first];
        if(man1 == 0){
            dp[man_now] = lady_first;
            ladyidx[lady_first] = man_now;
        }
        else if(lady[lady_first][man1] >
            lady[lady_first][man_now]){
            dp[man_now] = lady_first;
```

```
            manidx[man1]++;
            dq.emplace_back(man1);
            ladyidx[lady_first] = man_now;
        }
        else{
            dq.emplace_back(man_now);
            manidx[man_now]++;
        }
    }
}
int main(){
    int t; cin >> t;
    bool space = false;
    while(t--){
        cin >> n;
        if(space) cout << endl;
        space = true;
        memset(man, 0, sizeof(man));
        memset(lady, 0, sizeof(lady));
        memset(manidx, 0, sizeof(manidx));
        memset(ladyidx, 0, sizeof(ladyidx));
        dq.clear();
        for(int i = 1; i <= n; ++i){
            for(int j = 1; j <= n; ++j)
                cin >> man[i][j];
            dq.emplace_back(i);
            manidx[i] = 1;
        }
        for(int i = 1; i <= n; ++i){
            for(int j = 1; j <= n; ++j){
                int man_lady;
                cin >> man_lady;
                lady[i][man_lady] = j;
            }
        }
        dp_func();
        for(int i = 1; i <= n; ++i)
            cout << dp[i] << endl;
    }
}
```

11 LIS

11.1 Wavio Sequence

```
/* LIS、LDS */
int N;
const int maxn = 10000 + 5;
int length[maxn];
int seq[maxn], revseq[maxn];
void LIS(vector<int> &s){
    if(s.size() == 0) return;
    vector<int> v;
    v.emplace_back(s[0]);
    seq[0] = 1;
    for(int i = 1; i < s.size(); ++i){
        int n = s[i];
        if(n > v.back())
            v.push_back(n);
        else
            *lower_bound(v.begin(), v.end(), n) = n;
        seq[i] = v.size();
    }
    return;
}
void LDS(vector<int> &s){
    if(s.size() == 0) return;
    vector<int> v;
    v.emplace_back(s[0]);
    revseq[0] = 1;
    for(int i = 1; i < s.size(); ++i){
        int n = s[i];
        if(n > v.back())
            v.push_back(n);
        else
            *lower_bound(v.begin(), v.end(), n) = n;
        revseq[i] = v.size();
    }
    return;
}
int main(){
    while(cin >> N){
        vector<int> s(N), revs(N);
        for(int i = 0; i < N; i++){
            cin >> s[i];
            revs[i] = s[i];
        }
        reverse(revs.begin(), revs.end());
        LIS(s);
        LDS(revs);
        reverse(revseq, revseq + N);
        int maxi = -1;
        for(int i = 0; i < N; i++)
            if(min(seq[i], revseq[i]) > maxi)
                maxi = min(seq[i], revseq[i]);
        cout << maxi * 2 - 1 << endl;
    }
}
```

11.2 Robots II

```
/* LIS
No.:   2  4 11 13 25 28 41 42
LIS:   1  2  3  4  4  5  5  5
num:   1  1  1  1  1  2  2  4
path: -1  0  1  2  2  3  3  5 */
const int maxn = 100+5;
int r, c;
vector<int> G;
int LIS[maxn * maxn], num[maxn * maxn], path[maxn *
    maxn];
bool garbage[maxn][maxn];
void show_path(int n){
    if(path[n] != -1) show_path(path[n]);
    if((n != G.size() - 1) || garbage[r][c]) cout <<
        " " << G[n];
}
int main(){
    int ca = 1;
    while(cin >> r >> c && (r != -1) && (c != -1)){
        memset(garbage, false, sizeof(garbage));
        G.clear();
        int x, y;
        while(cin >> x >> y && x && y){
            garbage[x][y] = true;
        }
        // 紀錄有垃圾的點的編號
        for(int i = 1; i <= r; ++i){
            for(int j = 1; j <= c; ++j){
                if(garbage[i][j]) G.emplace_back((i -
                    1) * c + j);
            }
        }
        // 如果終點沒有垃圾，假設他有
        if(!garbage[r][c]) G.emplace_back(r * c);
        G.emplace_back(0);
        // i 和 j
            是按照編號大小順序由小排到大的垃圾編號
        for(int i = 0; i < G.size(); ++i){
            LIS[i] = 1;
            num[i] = 1;
            path[i] = -1;
            for(int j = 0; j < i; ++j){
                // 判斷垃圾的 col 前後
                if(((G[j] - 1) % c) <= ((G[i] - 1) %
                    c)){
                    // num 是經過的路徑數量。path
                        是從誰來
                    if(LIS[i] == LIS[j] + 1){
```

```
                        num[i] += num[j];
                    }
                    else if(LIS[i] < LIS[j] + 1){
                        LIS[i] = LIS[j] + 1;
                        num[i] = num[j];
                        path[i] = j;
                    }
                }
            }
        }
        G.pop_back();
        // 要把假設還回去
        if(!garbage[r][c]) LIS[G.size() - 1]--;
        cout << "CASE#" << ca++ << ": " <<
            LIS[G.size() - 1] << " " << num[G.size()
            - 1];
        show_path(G.size() - 1);
        cout << endl;
    }
}
```

12 Math

12.1 Big Mod

```
'''
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 Bubble Sort Expect Value

```
/* 數論 期望值算法:
擲一枚公平的六面骰子，其每次「點數」的期望值是 3.5
E(x) = 1 * 1/6 + 2 * 1/6 + 3 * 1/6 + 4 * 1/6 + 5 *
    1/6 + 6 * 1/6
= (1 + 2 + 3 + 4 + 5 + 6)/6 = 3.5
bubble sort 每兩兩之間交換機率是 1/2
總共會做 C(n, 2) 次
E(x) = C(n, 2) * 1/2 = (n * (n - 1))/2 * 1/2 */
int t, ca = 1;
cin >> t;
while(t--){
    long long int n;
    cin >> n;
    cout << "Case " << ca++ << ": ";
    // 如果 (n * (n - 1)) 可以被 4 整除
        代表最後答案會是整數，否則會是分數
    if((n * (n - 1)) % 4){
        cout << ( (n * (n - 1)) / 2 ) << "/2"<< endl;
    }
    else{
        cout << ( (n * (n - 1)) / 2 ) / 2 << endl;
    }
}
```

12.3 Fraction Floor Sum

```
/* 數論
[N/i] == M
-> M <= N/i < M + 1
-> N/(M+1) < i <= N/M */
int main(){
    long long int N;
    cin >> N;
    long long int ans = 0;
    for(long long int i = 1; i <= N; i++){
        long long int M = N / i, n = N / M;
        // 總共會有 n - i 個的 [N/i] 值都是 M
        ans += (n - i + 1) * M;
        // 更新跳過 以免重複計算
        i = n;
    }
    cout << ans << endl;
}
```

12.4 How Many Os

```
/* 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;
    }
}
```

12.5 Number of Pairs

```
/* 數論
uper_bound ex:
10 20 30 30 40 50
upper_bound for element 30 is at index 4
lower_bound ex:
10 20 30 40 50
lower_bound for element 30 at index 2 */
int main(){
    int t;
    cin >> t;
    while(t--){
        int n, l, r;
        vector<int> v;
        cin >> n >> l >> r;
        int num;
```

```
        for(int i = 0; i < n; i++){
            cin >> num;
            v.emplace_back(num);
        }
        sort(v.begin(), v.end());
        long long int ans = 0;
        for(int i = 0; i < n; i++)
            ans += (upper_bound(v.begin() + i + 1,
                v.end(), r - v[i]) -
                lower_bound(v.begin() + i + 1,
                v.end(), l - v[i]));
        cout << ans << endl;
    }
}
```

12.6 ORXOR

```
/* 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;
}
```

12.7 X drawing

```
/* 數論畫圖 */
int main(){
    long long int n;
    long long int a, b;
    long long int p, q, r, s;
    cin >> n >> a >> b;
    cin >> p >> q >> r >> s;
    for(long long int i = p; i <= q; i++){
        for(long long int j = r; j <= s; j++)
            if(abs(i - a) == abs(j - b)) cout << '#';
            else cout << '.';
        cout << endl;
    }
}
```

12.8 Playing With Stones

```
/* Nim Game - SG 函數 */
long long int SG(long long int n){
    return n % 2 == 0 ? n/2 : SG(n/2);
}
int main(){
    int t;
    cin >> t;
    while(t--){
        int n;
        cin >> n;
        long long int a, v = 0;
        for(int i = 0; i < n; ++i){
            cin >> a;
            v ^= SG(a);
        }
        if(v) cout << "YES" << endl;
        else cout << "NO" << endl;
    }
}
```

12.9 And Then There Was One

```
/* 環狀取石頭更新
f(1)=0
f(i)=(f(i-1)+k)%i
f(n)=(f(n-1)+m)%n
最後石頭編號: f(n)+1=1 */
const int maxn = 10000+5;
int f[maxn];
int main(){
    int n, k, m;
    while(cin >> n >> k >> m && n && k && m){
        f[1] = 0;
        // i 是剩下的石頭數量
        for(int i = 2; i < n; ++i){
            f[i] = (f[i-1] + k) % i;
        }
        f[n] = (f[n-1] + m) % n;
        cout << f[n] + 1 << endl;
    }
}
```

13 Binary Search

13.1 Fill the Containers

```
/*binary search 變形*/
int binary_search(int arr[maxn], int lef, int rig,
    int mini){
    if(lef > rig) return mini;
    int amount = 1, fill = 0;
    int mid = (lef + rig) >> 1;
    for(int i = 0; i < n; ++i){
        if(amount > m) break;
        fill += arr[i];
        if(fill > mid){
            fill = arr[i];
            amount++;
        }
    }
    if(!flag && amount <= m) mini = mid;
    if(flag && amount == m) mini = mid;
    if(amount == m){
```

```
        flag = true;
        return binary_search(arr, lef, mid - 1, mid);
    }
    else if(amount < m){
        return binary_search(arr, lef, mid - 1, mini);
    }
    else{
        return binary_search(arr, mid + 1, rig, mini);
    }
}
int main(){
    int ca = 1;
    while(cin >> n >> m){
        flag = false;
        int arr[maxn];
        int maxi = 0, sum = 0;
        for(int i = 0; i < n; ++i){
            cin >> arr[i];
            sum += arr[i];
            maxi = max(maxi, arr[i]);
        }
        cout << binary_search(arr, maxi, sum, maxi)
            << endl;
    }
}
```

13.2 Where is the marble

```
/*upper_bound & lower_bound*/
int main(){
    int N, Q;
    int ca = 1;
    while(cin >> N >> Q && N && Q){
        vector<int> v(N);
        for(int i = 0; i < N; ++i) cin >> v[i];
        sort(v.begin(), v.end());
        cout << "CASE# " << ca++ << ":" << endl;
        int marble;
        for(int i = 0; i < Q; ++i){
            cin >> marble;
            int lef = lower_bound(v.begin(), v.end(),
                marble) - v.begin();
            int rig = upper_bound(v.begin(), v.end(),
                marble) - v.begin();
            if(lef == rig) cout << marble << " not
                found" << endl;
            else{
                cout << marble << " found at " << lef
                    + 1 << endl;
            }
        }
    }
}
```

14 Graph

14.1 Maximum sum on a torus

```
/* Prefix sum in Graph*/
const int maxn = 80;
const int inf = 0x3f3f3f3f;
int arr[maxn*2 + 5][maxn*2 + 5];
int prefix_sum[maxn*2 + 5][maxn*2 + 5];
int ans[maxn*2];
int n;
int maxSub(int start){
    int maxi, dp;
    maxi = dp = ans[start];
    for(int i = start + 1; i < start + n; ++i){
        dp += ans[i];
        maxi = max(maxi, dp);
    }
    return maxi;
}
int main(){
    int t;
    cin >> t;
    while(t--){
        memset(arr, 0, sizeof(arr));
        cin >> n;
        for(int i = 0; i < n; ++i)
            for(int j = 0; j < n; ++j){
                cin >> arr[i][j];
                arr[n+i][j] = arr[i][n+j] =
                    arr[n+i][n+j] = arr[i][j];
            }
        int len = 2*n;
        memset(prefix_sum, 0, sizeof(prefix_sum));
        for(int i = 0; i < len; ++i)
            for(int j = 0; j < len; ++j){
                if(i == 0) prefix_sum[i][j] =
                    arr[i][j];
                else prefix_sum[i][j] =
                    prefix_sum[i-1][j] + arr[i][j];
            }
        int maxi = -inf;
        for(int i = 0; i < len; ++i){
            for(int j = i; j < i + n && j < len; ++j){
                for(int k = 0; k < len; ++k){
                    if(i == 0) ans[k] =
                        prefix_sum[j][k];
                    else ans[k] = prefix_sum[j][k] -
                        prefix_sum[i-1][k];
                }
                for(int k = 0; k < n; ++k){
                    int answer = maxSub(k);
                    maxi = max(maxi, answer);
                }
            }
        }
        cout << maxi << endl;
    }
}
```

15 Segement Tree

15.1 Frequent values

```
/* 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;
        }
    }
}
```

16 Dijkstra

16.1 Airport Express

```
/* Dijkstar 捷徑票 */
int n, m, S, T;
const int inf = 1e9;
const int maxn = 20000 + 5;
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], from[maxn];
vector<Edge> G[maxn];
void dijkstra(int s){
    for(int i = 0; i <= n; i++)
        dis[i] = inf;
    dis[s] = 0;
    for(int i = 0; i <= n; i++)
        from[i] = i;
    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;
        // 鬆弛 更新
        // 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;
                from[e.v] = now.u;
                pq.push({e.v, dis[e.v]});
            }
        }
    }
}
deque<int> ans;
void dfs(int T){
    ans.emplace_back(T);
    if(from[T] != T) dfs(from[T]);
}
int main(){
    bool space = false;
    while(cin >> n >> S >> T){
        if(!space) space = true;
        else cout << endl;
        for(int i = 0; i <= n; i++)
            G[i].clear();
        ans.clear();
        cin >> m;
        int u, v, w;
        for(int i = 0; i < m; i++){
            cin >> u >> v >> w;
            // 無向圖
            G[u].push_back({v, w});
            G[v].push_back({u, w});
        }
        dijkstra(S);
        dfs(T);
        int ori = dis[T];
        int mini = dis[T], state = 0;
        int ticket;
        cin >> ticket;
        for(int i = 0; i < ticket; ++i){
            cin >> u >> v >> w;
            G[u].push_back({v, w});
            dijkstra(S);
            if(dis[T] < mini){
                mini = min(mini, dis[T]);
                state = u;
                ans.clear();
                dfs(T);
            }
            G[u].pop_back();
```

```
            G[v].push_back({u, w});
            dijkstra(S);
            if(dis[T] < mini){
                mini = min(mini, dis[T]);
                state = v;
                ans.clear();
                dfs(T);
            }
            G[v].pop_back();
        }
        for(int i = ans.size()-1; i > 0; i--)
            cout << ans[i] << " ";
        cout << ans[0];
        cout << endl;
        if(mini == ori)
            cout << "Ticket Not Used" << endl;
        else
            cout << state << endl;
        cout << mini << endl;
    }
}
```

16.2 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;
    }
}
```

17 Kruskal

17.1 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;
}
```

18 Bipartite Graph

18.1 Claw Decomposition

```
/*二分圖 Bipatirate*/
const int maxn = 300+5;
int n;
int color[maxn];
vector<vector<int>> v(maxn);
bool dfs(int s){
    for(auto it : v[s]){
        if(color[it] == -1){
            //
                如果與點相連又還未填色，填塞成與原點不同的
            color[it] = 3 - color[s];
            // 同樣對此點去判定與此點相連的點的填色
            if(!dfs(it)) return false;
        }
        if(color[s] == color[it]){
            // 如果相鄰兩點同色，回傳 false
            return false;
        }
    }
    return true;
}
void isBipatirate(){
    bool flag = true;
    for(int i = 1; i <= n; ++i){
        if(color[i] == -1){
            // 如果還未填色過，就先填色成
                1，並對與此點相連的點都 dfs 判定填色
            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();
    }
}
```

18.2 Guardian of Decency

```
/* 二分圖最大匹配
匈牙利演算法 Hungarian algorithm*/
const int maxn = 500+5;
int bn, gn;
int match[maxn];
bool visited[maxn];
vector<vector<int>> G(maxn);
struct People{
    int h;
    string music, sport;
    // constructor
    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;
        // 如果這個女生還沒被配對過，直接匹配
        // 如果已經被配對，則根據這個女生所配對的對象
            dfs 重新匹配所有人的對象
        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(){
    int t;
    cin >> t;
    while(t--){
        int N;
        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;
    }
}
```

18.3 Taxi Cab Scheme

```
/* 二分圖最大匹配
匈牙利演算法 Hungarian algorithm */
const int maxn = 500+5;
int n;
int match[maxn];
bool visited[maxn];
vector<int> G[maxn];
struct People{
    int s, x1, y1, x2, y2;
    bool operator < (const People & rhs) const {
        return s < rhs.s;
    }
}p[maxn];
bool check(People boy, People girl){
    int tmp = boy.s + abs(boy.x2 - boy.x1) +
        abs(boy.y2 - boy.y1) + abs(boy.x2 - girl.x1)
        + abs(boy.y2 - girl.y1);
    if(tmp < girl.s) 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;
    meset(match, -1, sizeof(match));
    for(int i = 0; i < n; ++i){
        memset(visited, false, sizeof(visited));
        if(dfs(i)) cnt++;
    }
    return cnt;
}
int main(){
    int t;
    scanf("%d", &t);
    while(t--){
        scanf("%d", &n);
        for(int i = 0; i < n; ++i) G[i].clear();
        for(int i = 0; i < n; ++i){
            int h, m;
            scanf("%d:%d", &h, &m);
            p[i].s = h * 60 + m;
            scanf("%d%d%d%d", &p[i].x1, &p[i].y1,
                &p[i].x2, &p[i].y2);
        }
        sort(p, p + n);
        for(int i = 0; i < n; ++i)
            for(int j = i + 1; j < n; ++j)
                if(check(p[i], p[j]))
                    G[i].push_back(j);
        printf("%d\n", n - Hungarian());
    }
}
```

18.4 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;
    }
}
```

19 Function

19.1 CHAR

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

19.2 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;
    }
}
```

19.3 setprecision

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

19.4 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 );
```

19.5 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
```

19.6 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
```

19.7 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());
}
```

19.8 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 {
```

```
10          cout << "Not found." << endl;
11      }
12      // cout: Found: 6
13
14      // 取值：使用iterator
15      x = *st.begin(); // set 中的第一個元素(最小的元素)
16      x = *st.rbegin(); // set
            中的最後一個元素(最大的元素)
17
18      // search
19      iter = st.find(6);
20      auto it = st.find(x); // binary search, O(log(N))
21      auto it = st.lower_bound(x); // binary search,
            O(log(N))
22      auto it = st.upper_bound(x); // binary search,
            O(log(N))
23
24      st.clear();
25  }
```