

Contents

1	Basic	
1.1	BIT	
1.2	Black Magic	
1.3	DJS	
1.4	DFS	
1.5	BFS	
1.6	Segment Tree	
1.7	Template	
2	Data Structure	
2.1	Range Sum Query	
2.2	Splay Tree	
3	DP	
3.1	LCS	
3.2	LIS	
3.3	迴文	
4	Geometry	
4.1	Convex hull	
5	Graph	
5.1	Bellman Ford	
5.2	Dijk	
5.3	Edges	
5.4	Floyd	
5.5	KM	
5.6	Global Minimum Cut	
5.7	Kruskal	
5.8	K-th Shortest Path Length	
5.9	SPFA	
6	Math	
6.1	GCDhjackh	
6.2	Prime	
6.3	Gauss Elimination	
6.4	Matrix	
7	String	
7.1	KMP	
7.2	Trie	

1 Basic

1.1 BIT

```

1 #define lowbit(k) (k & -k)
2
3 int n;
4 vector<int> B1, B2;
5
6 void add(vector<int> &tr, int id, int val) {
7     for (; id <= n; id += lowbit(id)) {
8         tr[id] += val;
9     }
10 }
11 void range_add(int l, int r, int val) {
12     add(B1, l, val);
13     add(B1, r + 1, -val);
14     add(B2, l, val * (1 - 1));
15     add(B2, r + 1, -val * r);
16 }
17 int sum(vector<int> &tr, int id) {
18     int ret = 0;
19     for (; id >= 1; id -= lowbit(id)) {
20         ret += tr[id];
21     }
22     return ret;
23 }
24 int prefix_sum(int id) {
25     return sum(B1, id) * id - sum(B2, id);
26 }
27 int range_sum(int l, int r) {
28     return prefix_sum(r) - prefix_sum(l - 1);
29 }

```

1.2 Black Magic

```

1 #include <bits/stdc++.h>
2 // #include <ext/pb_ds/assoc_container.hpp>
3 // #include <ext/pb_ds/tree_policy.hpp>
4 // #include <ext/pb_ds/priority_queue.hpp>
5 using namespace std;
6 using namespace __gnu_pbds;
7 using set_t =
8     tree<int, null_type, less<int>, rb_tree_tag,
9         tree_order_statistics_node_update>; //紅黑樹(set)
10 using map_t =
11     tree<int, int, less<int>, rb_tree_tag,
12         tree_order_statistics_node_update>; //紅黑樹(map)
13 using heap_t =
14     __gnu_pbds::priority_queue<int>;
15 using ht_t =
16     gp_hash_table<int, int>;
17 int main() {
18     //set-----
19     set_t st;
20     st.insert(5); st.insert(6);
21     st.insert(3); st.insert(1);
22
23     // the smallest is (0), biggest is (n-1), kth small
24     // is (k-1)
25     int num = *st.find_by_order(0);
26     cout << num << '\n'; // print 1
27
28     num = *st.find_by_order(st.size() - 1);
29     cout << num << '\n'; // print 6
30
31     // find the index
32     int index = st.order_of_key(6); //在裡面第幾大
33     cout << index << '\n'; // print 3
34
35     // check if there exists x
36     int x = 5;
37     int check = st.erase(x);
38     if (check == 0) printf("st not contain 5\n");
39     else if (check == 1) printf("st contain 5\n");
40
41     //tree policy like set
42     st.insert(5); st.insert(5);
43     cout << st.size() << '\n'; // print 4
44
45     //map-----
46     map_t mp;
47     mp[1] = 2;
48     cout << mp[1] << '\n';
49     auto tmp = *mp.find_by_order(0); // pair
50     cout << tmp.first << " " << tmp.second << '\n';
51
52     //heap-----
53     heap_t h1, h2;
54     h1.push(1); h1.push(3);
55     h2.push(2); h2.push(4);
56     h1.join(h2);
57     cout << h1.size() << h2.size() << h1.top() << '\n';
58     // 支援合併
59     // 404
60
61     //hash-table-----
62     ht_t ht;
63     ht[85] = 5;
64     ht[89975] = 234;
65     for (auto i : ht) {
66         cout << i.first << " " << i.second << '\n';
67     }
68     //比較強的unordered map
69 }

```

1.3 DJS

```

1 const int MAXN = 1000;
2 int boss[MAXN];
3 void init(int n) {
4     for (int i = 0; i < n; i++) {
5         boss[i] = -1;
6     }
7 }
8 int find(int x) {
9     if (boss[x] < 0) {
10        return x;
11    }
12    return boss[x] = find(boss[x]);
13 }
14 bool uni(int a, int b) {
15     a = find(a);
16     b = find(b);
17     if (a == b) {
18         return false;
19     }
20     if (boss[a] > boss[b]) {
21         swap(a, b);
22     }
23     boss[a] += boss[b];
24     boss[b] = a;
25     return true;
26 }

```

1.4 DFS

```

1 struct Edge {
2     int bi,color; //a連接到的bi,通道顏色
3     bool operator < (const Edge &other) const{
4         return color < other.color;
5     }
6 };
7 vector<Edge> G[maxn];
8
9 void DFS(int me,int mydad,int distance){
10     if(dist[me] < distance) return;
11     dist[me] = distance;
12     for(int i = 0;i<G[me].size();i++){
13         int v = G[me][i].bi;
14         DFS(v,me,distance+1);
15     }
16 }

```

1.5 BFS

```

1 bool visit[maxn]; //訪問過的
2 void BFS(int point){
3     queue<int> q;
4     q.push(point);
5     while(!q.empty()){
6         int u = q.front();
7         if(visit[u]) continue; //訪問過就下一個
8         visit[u] = true;
9         for(int i =
10             0;i<edge[u][i];i++){ //連出去的線丟到queue
11             q.push(edge[u][i]);
12         }
13     }
14 }

```

1.6 Segment Tree

```

1 #include <./basic/Template.h>
2 const int INF = 1e9;
3 const int MAXN = ;
4
5 int n;
6 int a[MAXN], tr[MAXN << 1];

```

```

7 // !!! remember to call this function
8 void build() {
9     for (int i = 0; i < n; i++) {
10         tr[i + n] = a[i];
11     }
12     for (int i = n - 1; i > 0; i--) {
13         tr[i] = max(tr[i << 1], tr[i << 1 | 1]);
14     }
15 }
16 void update(int id, int val) {
17     for (tr[id += n] = val; id > 1; id >>= 1) {
18         tr[id >> 1] = max(tr[id], tr[id ^ 1]);
19     }
20 }
21 int query(int l, int r) { // [l, r)
22     int ret = -INF;
23     for (l += n, r += n; l < r; l >>= 1, r >>= 1) {
24         if (l & 1) {
25             ret = max(ret, tr[l++]);
26         }
27         if (r & 1) {
28             ret = max(ret, tr[--r]);
29         }
30     }
31     return ret;
32 }
33 }

```

1.7 Template

```

1 #pragma GCC optimize("O2")
2 #include <bits/stdc++.h>
3 using namespace std;
4 using LL = long long;
5 using ULL = unsigned long long;
6 using PII = pair<int, int>;
7 using PLL = pair<LL, LL>;
8 using VI = vector<int>;
9 using VVI = vector<vector<int>>>;
10 using dvt = double;
11 const int INF = 1e9;
12 const int MXN = 0;
13 const int MXV = 0;
14 const double EPS = 1e-9;
15 const int MOD = 1e9 + 7;
16 typedef long long ll;
17 typedef vector<int> vi;
18 typedef vector<string> vs;
19 typedef pair<int, int> pii;
20 typedef vector<pii> vpii;
21 #define MP make_pair
22 #define SORT(a) sort(a.begin(), a.end())
23 #define REVERSE(a) reverse(a.begin(), a.end())
24 #define ALL(a) a.begin(), a.end()
25 #define PI acos(-1)
26 #define ms(x, y) memset(x, y, sizeof(x))
27 #define inf 1e9
28 #define INF 1e16
29 #define pb push_back
30 #define MAX 100005
31 #define debug(a, b) cout << a << ": " << b << endl
32 #define Debug cout << "Reached here" << endl
33 #define prnt(a) cout << a << "\n"
34 #define mod 1000000007LL
35 #define FOR(i, a, b) for (int i = (a); i < (b); i++)
36 #define FORr(i, a, b) for (int i = (a); i >= (b); i--)
37 #define itrALL(c, itr) for (__typeof((c).begin()) itr = (c).begin(); itr != (c).end(); itr++)
38 #define lc ((node) << 1)
39 #define rc ((node) << 1 | 1)
40 #define VecPrnt(v) \
41     FOR(J, 0, v.size()) \
42     cout << v[J] << " "; \
43     cout << endl
44 #define endl "\n"

```

```

45 #define PrintPair(x) cout << x.first << " " <<
    x.second << endl
46 #define EPS 1e-9
47 #define ArrPrint(a, st, en) \
48     for (int J = st; J <= en; J++) \
49         cout << a[J] << " "; \
50     cout << endl;
51 #define MP make_pair
52 #define PB push_back
53 #define Fi first
54 #define Se second
55 #define FOR(i, L, R) for (int i = L; i < (int)R; ++i)
56 #define FORD(i, L, R) for (int i = L; i > (int)R; --i)
57 #define IOS \
58     cin.tie(nullptr); \
59     cout.tie(nullptr); \
60     ios_base::sync_with_stdio(false);
61
62 int main()
63 {
64     // ios_base::sync_with_stdio(0);
65     // cin.tie(NULL); cout.tie(NULL);
66     // freopen("in.txt", "r", stdin);
67     IOS;
68 }
69
70 /* Direction Array */
71
72 // int fx[]={1,-1,0,0};
73 // int fy[]={0,0,1,-1};
74 // int fx[]={0,0,1,-1,-1,1,-1,1};
75 // int fy[]={-1,1,0,0,1,1,-1,-1};
76
77 /***** END OF HEADER *****/

```

2 Data Structure

2.1 Range Sum Query

```

1 #include <./basic/Template.h>
2 int a[MAX + 7], tree[4 * MAX + 7], lazy[4 * MAX + 7];
3 void build(int node, int l, int r)
4 {
5     if (l == r)
6     {
7         tree[node] = a[l];
8         return;
9     }
10    if (l >= r)
11        return;
12    int mid = (l + r) / 2;
13    build(node * 2, l, mid);
14    build(node * 2 + 1, mid + 1, r);
15    tree[node] = tree[node * 2] + tree[node * 2 + 1];
16 }
17 void upd(int node, int l, int r, int v)
18 {
19     lazy[node] += v;
20     tree[node] += (r - l + 1) * x;
21 }
22 void pushDown(int node, int l, int r) //passing
    update information to the children
23 {
24     int mid = (l + r) / 2;
25     upd(node * 2, l, mid, lazy[node]);
26     upd(node * 2 + 1, mid + 1, r, lazy[node]);
27     lazy[node] = 0;
28 }
29 void update(int node, int l, int r, int x, int y, int
    v)
30 {
31     if (x > r || y < l)
32         return;
33     if (x >= l && r <= y)

```

```

34 {
35     upd(node, l, r, v);
36     return;
37 }
38 pushDown(node, l, r);
39 int mid = (l + r) / 2;
40 update(node * 2, l, mid, x, y, v);
41 update(node * 2 + 1, mid + 1, r, x, y, v);
42 tree[node] = tree[node * 2] + tree[node * 2 + 1];
43 }

```

2.2 Splay Tree

```

1 #include <./basic/Template.h>
2 /**
3  Splay Tree :
4  Node:
5      void addIt(int ad) : adding an integer in a
        range
6      void revIt() : reversing flag
7      void upd() : push_up( gather from child)
8      void pushdown() : pass values to the child(
        like lazy propagation)
9  Splay:
10     Node* newNode(int v, Node* f) :Returns Pointer
        of a node whose parent is f, and value v
11     Node* build(int l, int r, Node* f) : building
        [L,R] which parent is f
12     void rotate(Node* t, int d) : Rotation of
        Splay Tree
13     void splay(Node* t, Node* f) : Splaying , t
        resides just below the f
14     void select(int k, Node* f) : Select k th
        element in the tree ,splay it to the just
        below f
15     Node*&get(int l, int r) : Getting The node
        for segment [L,R]
16     void reverse(int l, int r) : Reverse a segment
17     void del(int p) : deletes entry a[p]
18     void split(int l, int r, Node*&s1) : Split the
        array and s1 stores the [L,R] segment
19     void cut(int l, int r) : Cut the segment [L,R]
        and insert in at the end
20     void insert(int p, int v) : Insert after p, ( 0
        means before the array) an element whose
        value is v
21     void insertRange(int pos, Node* s) : Insert
        after pos, an segment denoted by s
22     int query(int l, int r) : Output desired result
        for [L,R]
23     void addRange(int l, int r, int v) : Add v to
        all the element in segment [L,R]
24     void output(int l, int r) : Output the segment
        [L,R]
25 */
26
27 /*
28 The following code answers the following queries
29 1 L R Output Maximum value in range [L,R]
30 2 L R Reverse the array [L,R]
31 3 L R v add v in range [L,R]
32 4 pos removes entry from pos
33 5 pos v - insert an element after position v
34
35 We assumes the initial array stored in
    ar[]={1,2,3,4... n}
36 */
37
38 typedef int T;
39
40 const int N = 2e5+50; // >= Node + Query
41 T ar[N]; // Initial Array
42 struct Node{
43     Node *ch[2], *pre; // child and parent
44     T val; // Value stored in each node

```

```

45     int size; //size of the subtree rooted at this
46     node
47     T mx; // additional info stored to solve
48     problems, here maximum value
49     T sum;
50     T add; //lazy updates
51     bool rev; // reverse flag
52     Node(){size=0; val=mx=-1e9; add=0;}
53     void addIt(T ad){
54         add+=ad;
55         mx+=ad;
56         sum += size*ad;
57         val+=ad;
58     }
59     void revIt(){
60         rev^=1;
61     }
62     void upd(){
63         size=ch[0]->size+ch[1]->size+1;
64         mx=max(val, max(ch[0]->mx, ch[1]->mx));
65         sum= ch[0]->sum + ch[1]->sum + val;
66     }
67     void pushdown();
68 }Tnull,*null=&Tnull;
69 void Node::pushdown(){
70     if (add!=0){
71         for (int i=0; i<2; ++i)
72             if (ch[i]!=null) ch[i]->addIt(add);
73         add = 0;
74     }
75     if (rev){
76         swap(ch[0], ch[1]);
77         for (int i=0; i<2; i++)
78             if (ch[i]!=null) ch[i]->revIt();
79         rev = 0;
80     }
81 }
82 struct Splay{
83     Node nodePool[N], *cur; // Static Memory and cur
84     pointer
85     Node* root; // root of the splay tree
86     Splay(){
87         cur=nodePool;
88         root=null;
89     }
90     void clear(){
91         cur=nodePool;
92         root=null;
93     }
94     Node* newNode(T v, Node* f){
95         cur->ch[0]=cur->ch[1]=null;
96         cur->size=1;
97         cur->val=v;
98         cur->mx=v; cur->sum = 0;
99         cur->add=0;
100         cur->rev=0;
101         cur->pre=f;
102         return cur++;
103     }
104     Node* build(int l, int r, Node* f){
105         if(l>r) return null;
106         int m=(l+r)>>1;
107         Node* t=newNode(ar[m], f);
108         t->ch[0]=build(l, m-1, t);
109         t->ch[1]=build(m+1, r, t);
110         t->upd();
111         return t;
112     }
113     void rotate(Node* x, int c){
114         Node* y=x->pre;
115         y->pushdown();
116         x->pushdown();
117         y->ch[!c]=x->ch[c];
118         if (x->ch[c]!=null) x->ch[c]->pre=y;
119         x->pre=y->pre;
120         if (y->pre!=null)
121             if (y->pre->ch[0]==y) y->pre->ch[0]=x;
122             else y->pre->ch[1]=x;
123         }
124         x->ch[c]=y;
125         y->pre=x;
126         y->upd();
127         if (y==root) root=x;
128     }
129 void splay(Node* x, Node* f){
130     x->pushdown();
131     while (x->pre!=f){
132         if (x->pre->pre==f){
133             if (x->pre->ch[0]==x) rotate(x, 1);
134             else rotate(x, 0);
135         }else{
136             Node *y=x->pre, *z=y->pre;
137             if (z->ch[0]==y){
138                 if (y->ch[0]==x)
139                     rotate(y, 1), rotate(x, 1);
140                 else rotate(x, 0), rotate(x, 1);
141             }else{
142                 if (y->ch[1]==x)
143                     rotate(y, 0), rotate(x, 0);
144                 else rotate(x, 1), rotate(x, 0);
145             }
146         }
147     }
148     x->upd();
149 }
150 void select(int k, Node* f){
151     int tmp;
152     Node* x=root;
153     x->pushdown();
154     k++;
155     for(;;){
156         x->pushdown();
157         tmp=x->ch[0]->size;
158         if (k==tmp+1) break;
159         if (k<=tmp) x=x->ch[0];
160         else{
161             k-=tmp+1;
162             x=x->ch[1];
163         }
164     }
165     splay(x, f);
166 }
167 Node*&get(int l, int r){
168     select(l-1, null);
169     select(r+1, root);
170     return root->ch[1]->ch[0];
171 }
172 void reverse(int l, int r){
173     Node* o=get(l, r);
174     o->rev^=1;
175     splay(o, null);
176 }
177 void del(int p)
178 {
179     select(p-1, null);
180     select(p+1, root);
181     root->ch[1]->ch[0] = null;
182     splay(root->ch[1], null);
183 }
184 void split(int l, int r, Node*&s1)
185 {
186     Node* tmp=get(l, r);
187     root->ch[1]->ch[0]=null;
188     root->ch[1]->upd();
189     root->upd();
190     s1=tmp;
191 }

```

```

194 }
195 void cut(int l,int r)
196 {
197     Node* tmp;
198     split(l,r,tmp);
199     select(root->size-2,null);
200     root->ch[1]->ch[0]=tmp;
201     tmp->pre=root->ch[1];
202     root->ch[1]->upd();
203     root->upd();
204 }
205
206 void init(int n){
207     clear();
208     root=newNode(0,null);
209     root->ch[1]=newNode(n+1,root);
210     root->ch[1]->ch[0]=build(1,n,root->ch[1]);
211     splay(root->ch[1]->ch[0],null);
212 }
213
214 void insertPos(int pos,T v)
215 {
216     select(pos,null);
217     select(pos+1,root);
218     root->ch[1]->ch[0] =
219         newNode(v,root->ch[1]);
220     splay(root->ch[1]->ch[0],null);
221 }
222 void insertRange(int pos,Node *s)
223 {
224     select(pos,null);
225     select(pos+1,root);
226     root->ch[1]->ch[0] = s;
227     s->pre = root->ch[1];
228     root->ch[1]->upd();
229     root->upd();
230 }
231 T query(int l,int r)
232 {
233     Node *o = get(l,r);
234     return o->mx;
235 }
236
237 void addRange(int l,int r,T v)
238 {
239     Node *o = get(l,r);
240     o->add += v;
241     o->val += v;
242     o->sum += o->size * v;
243     splay(o,null);
244 }
245
246 void output(int l,int r){
247     for (int i=l;i<=r;i++){
248         select(i,null);
249         cout<<root->val<<endl;
250     };
251 }
252 }St;
253
254
255
256 int main()
257 {
258     int n,m,a,b,c;
259
260     scanf("%d%d", &n, &m);
261
262     for(int i=1;i <= n;i ++ ) ar[i] = i;
263     St.init(n);
264
265     FOR(i,1,m+1)
266     {
267         scanf("%d%d", &a, &b);
268         St.cut(a,b);
269     }

```

```

270
271     St.output(1,n);
272
273
274     return 0;
275 }

```

3 DP

3.1 LCS

```

1 int LCS(string s1, string s2) {
2     int n1 = s1.size(), n2 = s2.size();
3     vector<vector<int>> dp(n1 + 1, vector<int>(n2 + 1,
4         0));
5     for (int i = 1; i <= n1; i++) {
6         for (int j = 1; j <= n2; j++) {
7             if (s1[i - 1] == s2[j - 1]) {
8                 dp[i][j] = dp[i - 1][j - 1] + 1;
9             } else {
10                 dp[i][j] = max(dp[i - 1][j], dp[i][j - 1]);
11             }
12         }
13     }
14     return dp[n1][n2];
15 }

```

3.2 LIS

```

1 int LIS(vector<int> &a) {
2     vector<int> s;
3     for (int i = 0; i < a.size(); i++) {
4         if (s.empty() || s.back() < a[i]) {
5             s.push_back(a[i]);
6         } else {
7             *lower_bound(s.begin(), s.end(), a[i],
8                 [](int x, int y) {return x < y;}) = a[i];
9         }
10    }
11    return s.size();
12 }

```

3.3 迴文

```

1 bool isPalindrome[100][100];
2 // Find the palindromes of a string in O(n^2)
3
4 int main()
5 {
6     ios_base::sync_with_stdio(0);
7     // freopen("in.txt", "r", stdin);
8     string s;
9     cin>>s;
10    int len=s.size();
11    for(int i=0; i<len; i++)
12        isPalindrome[i][i]=true;
13
14    for(int k=1; k<len; k++){
15        for(int i=0; i+k<len; i++){
16            int j=i+k;
17            isPalindrome[i][j]=(s[i]==s[j]) &&
18                (isPalindrome[i+1][j-1] || i+1==j-1);
19        }
20    }
21    return 0;
22 }

```

4 Geometry

4.1 Convex hull

```

1 #include <./basic/Template.h>
2 struct PT
3 {
4     int x, y;
5     PT() {}
6     PT(int x, int y) : x(x), y(y) {}
7     bool operator<(const PT &P) const
8     {
9         return x < P.x || (x == P.x && y < P.y);
10    }
11 };
12 ll cross(const PT p, const PT q, const PT r)
13 {
14     return (ll)(q.x - p.x) * (ll)(r.y - p.y) -
15            (ll)(q.y - p.y) * (ll)(r.x - p.x);
16 }
17
18 vector<PT> Points, Hull;
19
20 void findConvexHull()
21 {
22     int n = Points.size(), k = 0;
23
24     SORT(Points);
25
26     // Build lower hull
27
28     FOR(i, 0, n)
29     {
30         while (Hull.size() >= 2 &&
31                cross(Hull[Hull.size() - 2], Hull.back(),
32                     Points[i]) <= 0)
33         {
34             Hull.pop_back();
35             k--;
36         }
37         Hull.pb(Points[i]);
38         k++;
39     }
40
41     // Build upper hull
42
43     for (int i = n - 2, t = k + 1; i >= 0; i--)
44     {
45         while (Hull.size() >= t &&
46                cross(Hull[Hull.size() - 2], Hull.back(),
47                     Points[i]) <= 0)
48         {
49             Hull.pop_back();
50             k--;
51         }
52         Hull.pb(Points[i]);
53         k++;
54     }
55     Hull.resize(k);
56 }

```

5 Graph

5.1 Bellman Ford

```

1 #include <./basic/Template.h>
2 bool bellman(int src)
3 {
4     // Nodes are indexed from 1
5     for (int i = 1; i <= n; i++)
6         dist[i] = INF;

```

```

7     dist[src] = 0;
8     for(int i = 2; i <= n; i++)
9     {
10         for (int j = 0; j < edges.size(); j++)
11         {
12             int u = edges[j].first;
13             int v = edges[j].second;
14             ll weight = adj[u][v];
15             if (dist[u] != INF && dist[u] + weight <
16                 dist[v])
17                 dist[v] = dist[u] + weight;
18         }
19     }
20     for (int i = 0; i < edges.size(); i++)
21     {
22         int u = edges[i].first;
23         int v = edges[i].second;
24         ll weight = adj[u][v];
25         // True if neg-cycle exists
26         if (dist[u] != INF && dist[u] + weight < dist[v])
27             return true;
28     }
29     return false;
30 }

```

5.2 Dijk

```

1 #include <./basic/Template.h>
2 const long long int INF = 1e18;
3 const int MAXN = 1000000;
4 struct Edge {
5     int to;
6     long long int cost;
7     Edge(int v, long long int c) : to(v), cost(c) {}
8     bool operator < (const Edge &other) const {
9         return cost > other.cost;
10    };
11 };
12
13 int n;
14 long long int dis[MAXN];
15 vector<Edge> G[MAXN];
16
17 void init() {
18     for (int i = 0; i < n; i++) {
19         G[i].clear();
20         dis[i] = INF;
21     }
22 }
23
24 void Dijkstra(int st, int ed = -1) {
25     priority_queue<Edge> pq;
26     pq.emplace(st, 0);
27     dis[st] = 0;
28     while (!pq.empty()) {
29         auto now = pq.top();
30         pq.pop();
31         if (now.to == ed) {
32             return;
33         }
34         if (now.cost > dis[now.to]) {
35             continue;
36         }
37         for (auto &e : G[now.to]) {
38             if (dis[e.to] > now.cost + e.cost) {
39                 dis[e.to] = now.cost + e.cost;
40                 pq.emplace(e.to, dis[e.to]);
41             }
42         }
43     }
44 }

```

5.3 Edges

```

1 struct Edge
2 {
3     int from, to, w;
4     bool operator<(const Edge& rhs) // optional
5     {
6         return w < rhs.w;
7     }
8 };

```

5.4 Floyd

```

1 const LL INF = 1e18;
2 const int MAXN = ;
3
4 int n;
5 LL G[MAXN][MAXN];
6
7 void init() {
8     for (int i = 0; i < n; i++) {
9         for (int j = 0; j < n; j++) {
10             G[i][j] = INF;
11         }
12         G[i][i] = 0;
13     }
14 }
15 void floyd() {
16     for (int k = 0; k < n; k++) {
17         for (int i = 0; i < n; i++) {
18             for (int j = 0; j < n; j++) {
19                 if (G[i][k] != INF && G[k][j] != INF) {
20                     G[i][j] = min(G[i][j], G[i][k] + G[k][j]);
21                 }
22             }
23         }
24     }
25 }

```

5.5 KM

```

1 const int INF = 1e9;
2 const int MAXN = ;
3 struct KM { //1-base
4     int n, G[MAXN][MAXN];
5     int lx[MAXN], ly[MAXN], my[MAXN];
6     bool vx[MAXN], vy[MAXN];
7     void init(int _n) {
8         n = _n;
9         for (int i = 1; i <= n; i++) {
10             for (int j = 1; j <= n; j++) {
11                 G[i][j] = 0;
12             }
13         }
14     }
15     bool match(int i) {
16         vx[i] = true;
17         for (int j = 1; j <= n; j++) {
18             if (lx[i] + ly[j] == G[i][j] && !vy[j]) {
19                 vy[j] = true;
20                 if (!my[j] || match(my[j])) {
21                     my[j] = i;
22                     return true;
23                 }
24             }
25         }
26         return false;
27     }
28     void update() {
29         int delta = INF;
30         for (int i = 1; i <= n; i++) {
31             if (vx[i]) {
32                 for (int j = 1; j <= n; j++) {
33                     if (!vy[j]) {
34                         delta = min(delta, lx[i] + ly[j] -

```

```

35                     }
36                 }
37             }
38         }
39         for (int i = 1; i <= n; i++) {
40             if (vx[i]) {
41                 lx[i] -= delta;
42             }
43             if (vy[i]) {
44                 ly[i] += delta;
45             }
46         }
47     }
48     int run() {
49         for (int i = 1; i <= n; i++) {
50             lx[i] = ly[i] = my[i] = 0;
51             for (int j = 1; j <= n; j++) {
52                 lx[i] = max(lx[i], G[i][j]);
53             }
54         }
55         for (int i = 1; i <= n; i++) {
56             while (true) {
57                 for (int i = 1; i <= n; i++) {
58                     vx[i] = vy[i] = 0;
59                 }
60                 if (match(i)) {
61                     break;
62                 } else {
63                     update();
64                 }
65             }
66         }
67         int ans = 0;
68         for (int i = 1; i <= n; i++) {
69             ans += lx[i] + ly[i];
70         }
71         return ans;
72     }
73 };

```

5.6 Global Minimum Cut

```

1 #include <./basic/Template.h>
2 /*Given an undirected graph  $G = (V, E)$ , we define a
3 cut of  $G$  to be a partition
4 of  $V$  into two non-empty sets  $A$  and  $B$ . Earlier, when
5 we looked at network
6 flows, we worked with the closely related definition
7 of an s-t cut: there, given
8 a directed graph  $G = (V, E)$  with distinguished source
9 and sink nodes  $s$  and  $t$ ,
10 an s-t cut was defined to be a partition of  $V$  into
11 sets  $A$  and  $B$  such that  $s \in A$ 
12 and  $t \in B$ . Our definition now is slightly different,
13 since the underlying graph
14 is now undirected and there is no source or sink.
15 This problem can be solved by max-flow. First we
16 remove undirected edges and replace
17 them by two opposite directed edge. Now we fix a node
18  $s$ . Then we consider each of
19 the  $n$  nodes as  $t$  and run max-flow. The minimum of
20 those values is the answer.
21 This is  $O(n^3)$ .
22 */
23
24 struct Stoer_Wagner
25 {
26     vector<vl> weights;
27     Stoer_Wagner(ll N)
28     {
29         weights.resize(N, vl(N, 0));
30     }
31     void AddEdge(ll from, ll to, ll cap)
32     {
33         weights[from][to] += cap;
34         weights[to][from] += cap;
35     }
36 }

```



```

26 }
27 pair<ll, vl> GetMinCut()
28 {
29     ll N = weights.size();
30     vl used(N), cut, best_cut;
31     ll best_weight = -1;
32
33     for (ll phase = N - 1; phase >= 0; phase--)
34     {
35         vl w = weights[0];
36         vl added = used;
37         ll prev, last = 0;
38         for (ll i = 0; i < phase; i++)
39         {
40             prev = last;
41             last = -1;
42             for (ll j = 1; j < N; j++)
43                 if (!added[j] && (last == -1 ||
44                     w[j] > w[last]))
45                     last = j;
46             if (i == phase - 1)
47             {
48                 for (ll j = 0; j < N; j++)
49                     weights[prev][j] +=
50                     weights[last][j];
51                 for (ll j = 0; j < N; j++)
52                     weights[j][prev] =
53                     weights[j][last];
54                 used[last] = true;
55                 cut.push_back(last);
56                 if (best_weight == -1 || w[last]
57                     < best_weight)
58                 {
59                     best_cut = cut;
60                     best_weight = w[last];
61                 }
62             }
63             else
64             {
65                 for (ll j = 0; j < N; j++)
66                     w[j] += weights[last][j];
67                 added[last] = true;
68             }
69         }
70     }
71     return make_pair(best_weight, best_cut);
72 }
73
74 int main()
75 {
76     ll T;
77     sl(T);
78     f(t, 1, T + 1)
79     {
80         ll N, M;
81         sll(N, M);
82         Stoer_Wagner SW(N);
83         f(i, 0, M)
84         {
85             ll a, b, c;
86             slll(a, b, c);
87             SW.AddEdge(a - 1, b - 1, c);
88         }
89     }

```

5.7 Krushal

```

1 #include <./basic/Template.h>
2 struct edge
3 {
4     int u, v, w;
5     bool operator<(const edge &p) const

```

```

6 {
7     return w < p.w;
8 }
9 };
10 edge get;
11 int parent[100];
12 vector<edge> e;
13 int find(int r)
14 {
15     if (parent[r] == r)
16         return r;
17     return parent[r] = find(parent[r]);
18 }
19 int mst(int n)
20 {
21     sort(e.begin(), e.end());
22     for (int i = 1; i <= n; i++)
23         parent[i] = i;
24     int cnt = 0, s = 0;
25     for (int i = 0; i < (int)e.size(); i++)
26     {
27         int u = find(e[i].u);
28         int v = find(e[i].v);
29         if (u != v)
30         {
31             parent[u] = v;
32             cnt++;
33             s += e[i].w;
34             if (cnt == n - 1)
35                 break;
36         }
37     }
38 }

```

5.8 K-th Shortest Path Length

```

1 #include <./basic/Template.h>
2 int n, m, x, y, k, a, b, c;
3 vi Graph[103], Cost[103];
4 vector<priority_queue<int>> d(103);
5 priority_queue<pii> Q;
6
7 void goDijkstra()
8 {
9
10     // Here, elements are sorted in decreasing order
11     // of the first elements
12     // of the pairs and then the second elements if
13     // equal first element.
14     // d[i] is the priority_queue of the node i where
15     // the best k path length
16     // will be stored in decreasing order. So,
17     // d[i].top() has the longest of the
18     // first k shortest path.
19     d[x].push(0);
20     Q.push(MP(x, 0));
21     // Q contains the nodes in the increasing order
22     // of their cost
23     // Since the priority_queue sorts the pairs in
24     // decreasing order of their
25     // first element and then second element, to sort
26     // it in increasing order
27     // we will negate the cost and push it.
28
29     while (!Q.empty())
30     {
31         pii t = Q.top();
32         Q.pop();
33         int u = t.first, costU = -t.second;
34         // Since the actual cost was negated.
35
36         FOR(j, 0, Graph[u].size())
37         {
38             int v = Graph[u][j];
39
40             // prnt(v); prnt(d[v].size());

```



```

34
35 // Have we already got k shortest paths?
36 // Or is the longest path can be made
37 // better?
38 if (d[v].size() < k || d[v].top() > costU
39 + Cost[u][j])
40 {
41     int temp = costU + Cost[u][j];
42     d[v].push(temp);
43     Q.push(MP(v, -temp));
44 }
45 if (d[v].size() > k)
46     d[v].pop();
47 // If we have more than k shortest path
48 // for the current node, we can pop
49 // the worst ones.
50 }
51 if (d[y].size() < k)
52     prnt(-1);
53 // We have not found k shortest path for our
54 // destination.
55 else
56     prnt(d[y].top());
57 }
58
59 int main()
60 {
61     // ios_base::sync_with_stdio(0);
62     // cin.tie(NULL); cout.tie(NULL);
63     // freopen("in.txt", "r", stdin);
64
65     while (scanf("%d%d", &n, &m) && n + m)
66     {
67         scanf("%d%d%d", &x, &y, &k);
68
69         FOR(i, 0, m)
70         {
71             scanf("%d%d%d", &a, &b, &c);
72
73             Graph[a].pb(b);
74             Cost[a].pb(c);
75         }
76
77         goDijkstra();
78
79         FOR(i, 0, 103)
80             Graph[i].clear(),
81             Cost[i].clear();
82         FOR(i, 0, 103)
83         {
84             while (!d[i].empty())
85                 d[i].pop();
86         }
87
88         while (!Q.empty())
89             Q.pop();
90
91     }
92
93     return 0;
94 }

```

5.9 SPFA

```

1
2 #include <./basic/Template.h>
3 #define MAXN 1000000
4 struct Edge
5 {
6     int at;
7     long long cost;
8 };
9 int n;
10 long long dis[MAXN];
11 vector<Edge> G[MAXN];

```

```

12 void init()
13 {
14     for (int i = 0; i < n; i++)
15     {
16         G[i].clear();
17         dis[i] = INF;
18     }
19 }
20 bool SPFA(int st)
21 {
22     vector<int> cnt(n, 0);
23     vector<bool> inq(n, false);
24     queue<int> q;
25
26     q.push(st);
27     dis[st] = 0;
28     inq[st] = true;
29     while (!q.empty())
30     {
31         int now = q.front();
32         q.pop();
33         inq[now] = false;
34         for (auto &e : G[now])
35         {
36             if (dis[e.at] > dis[now] + e.cost)
37             {
38                 dis[e.at] = dis[now] + e.cost;
39                 if (!inq[e.at])
40                 {
41                     cnt[e.at]++;
42                     if (cnt[e.at] > n)
43                     {
44                         // negative cycle
45                         return false;
46                     }
47                     inq[e.at] = true;
48                     q.push(e.at);
49                 }
50             }
51         }
52     }
53     return true;
54 }

```

6 Math

6.1 GCDhjackh

```

1 int extgcd(int a, int b, int c, int &x, int &y) {
2     if (b == 0) {
3         x = c / a;
4         y = 0;
5         return a;
6     }
7     int d = extgcd(b, a % b, c, x, y);
8     int tmp = x;
9     x = y;
10    y = tmp - (a / b) * y;
11    return d;
12 }

```

6.2 Prime

```

1 const int maxn = ;
2 int arr[maxn];
3 int prime[maxn];
4 void init(){
5     for (int i = 0; i < maxn; ++i){
6         arr[i] = 0;
7     }
8 }
9 void find(){

```

```

10 int num = 0;
11 for(int i = 2; i < maxn; i++){
12     if(arr[i] == 0){
13         prime[num] = 0;
14         num++;
15         for(int j = i*i; j < maxn; j+=i){
16             arr[j] = 1;
17         }
18     }
19 }
20 }

```

6.3 Gauss Elimination

```

1 #include <./basic/Template.h>
2 const int MAXN = 300;
3 const double EPS = 1e-8;
4 int n;
5 double A[MAXN][MAXN];
6 void Gauss()
7 {
8     for (int i = 0; i < n; i++)
9     {
10         bool ok = 0;
11         for (int j = i; j < n; j++)
12         {
13             if (fabs(A[j][i]) > EPS)
14             {
15                 swap(A[j], A[i]);
16                 ok = 1;
17                 break;
18             }
19         }
20         if (!ok)
21             continue;
22         double fs = A[i][i];
23         for (int j = i + 1; j < n; j++)
24         {
25             double r = A[j][i] / fs;
26             for (int k = i; k < n; k++)
27             {
28                 A[j][k] -= A[i][k] * r;
29             }
30         }
31     }
32 }

```

6.4 Matrix

```

1 template <typename T, int N = 2>
2 struct Mat
3 { // Matrix
4     unsigned long long v[N][N];
5     Mat operator*(Mat b) const
6     {
7         Mat val;
8         for (int i = 0; i < N; i++)
9         {
10             for (int j = 0; j < N; j++)
11             {
12                 val.v[i][j] = 0;
13                 for (int k = 0; k < N; k++)
14                 {
15                     val.v[i][j] += v[i][k] *
16                         b.v[k][j];
17                 }
18             }
19         }
20         return val;
21 };

```

7 String

7.1 KMP

```

1 void failure(string s, int len, int *f)
2 {
3     f[0] = -1;
4     for(int i = 1; i < len; i++)
5     {
6         int k = f[i-1];
7
8         while(s[i] != s[k+1] && k >= 0)
9             k = f[k];
10
11         if(s[i] == s[k+1])f[i] = k+1;
12         else f[i] = -1;
13     }
14 }
15
16 int compare(string big, string little, int *f)
17 {
18     int Blen = big.length(), Llen = little.length();
19     int i = 0, j = 0;
20
21     while(i < Blen && j < Llen)
22     {
23         if(big[i] == little[j])
24         {
25             i++;
26             j++;
27         }
28         else if(j == 0)i++;
29         else j = f[j-1] + 1;
30     }
31
32     if(j == Llen)return 1;
33     else return 0;
34 }

```

7.2 Trie

```

1 #include <./basic/Template.h>
2 struct Node
3 {
4     char ch;
5     int v;
6     Node *next[26];
7     Node()
8     {
9         v = 0;
10         FOR(i, 0, 26)
11             next[i] = NULL;
12     }
13 };
14
15 void insert(Node *root, string s)
16 {
17     FOR(i, 0, s.size())
18     {
19         int v = s[i] - 'a';
20         if (root->next[v] == NULL)
21         {
22             root->next[v] = new Node();
23         }
24         root = root->next[v];
25         ++root->v;
26         root->ch = s[i];
27     }
28     return;
29 }
30
31 void search(Node *root, string s)
32 {
33     FOR(i, 0, s.size())
34     {

```

```
34     int v = s[i] - 'a';
35     root = root->next[v];
36     if (root->v == 1)
37     {
38         cout << s << ' ' << s.substr(0, i + 1) <<
39             '\n';
40         return;
41     }
42     cout << s << ' ' << s << '\n';
43 }
44
45 int main()
46 {
47     vector<string> v;
48     string s;
49     Node *root = new Node();
50     while (cin >> s)
51     {
52         insert(root, s);
53         v.push_back(s);
54     }
55     FOR(i, 0, v.size()) { search(root, v[i]); }
56 }
```

