1. 数据结构
   1. 线段树

1 int sum[MAXN \* 3], add[MAXN \* 3];`

2 void pushup(int t){

3 sum[t] = sum[t << 1] + sum[t << 1 | 1];

4 }

5 void pushdown(int t, int x){

6 if (add[t]){

7 add[t << 1] += add[t];

8 add[t << 1 | 1] += add[t];

9 sum[t << 1] += ((x + 1) >> 1)\* add[t];

10 sum[t << 1 | 1] += (x >> 1) \* add[t];

11 add[t] = 0;

12 }

13 }

14 void update(int L, int R, int t, int p, int q, int x){

15 if (p <= L && q >= R){

16 sum[t] += (R - L + 1) \* x;

17 add[t] += x;

18 return;

19 }

20

21 pushdown(t, R - L + 1);

22 int mid = (L + R) >> 1;

23 if (p <= mid){

24 update(L, mid, t << 1, p, q, x);

25 }

26 if (q > mid){

27 update(mid + 1, R, t << 1 | 1, p, q, x);

28 }

29 pushup(t);

30 }

31 int query(int L, int R, int t, int p, int q){

32 if (p <= L && q >= R){

33 return sum[t];

34 }

35 pushdown(t, R - L + 1);

36 int mid = (L + R) >> 1;

37 int res = 0;

38 if (p <= mid)

39 res += query(L, mid, t << 1, p, q);

40 }

41 if (q > mid){

42 res += query(mid + 1, R, t << 1 | 1, p, q);

43 }

44 return res;

45 }

46 void build(int L, int R, int t){

47 if (L == R){

48 sum[t] = 1;

49 return;

50 }

51 int mid = (L + R) >> 1;

52 build(L, mid, t << 1);

53 build(mid + 1, R, t << 1 | 1);

54 sum[t] = sum[t << 1] + sum[t << 1 | 1];

55 }

* 1. 树状数组

1 int lowbit(int x)

2 {

3 return x & (-x);

4 }

5 void modify(int x,int add)//一维

6 {

7 while(x<=MAXN)

8 {

9 a[x]+=add;

10 x+=lowbit(x);

11 }

12 }

13 int get\_sum(int x)

14 {

15 int ret=0;

16 while(x!=0)

17 {

18 ret+=a[x];

19 x-=lowbit(x);

20 }

21 return ret;

22 }

23 void modify(int x,int y,int data)//二维

24 {

25 for(int i=x;i<MAXN;i+=lowbit(i))

26 for(int j=y;j<MAXN;j+=lowbit(j))

27 a[i][j]+=data;

28 }

29 int get\_sum(int x,int y)

30 {

31 int res=0;

32 for(int i=x;i>0;i-=lowbit(i))

33 for(int j=y;j>0;j-=lowbit(j))

34 res+=a[i][j];

35 return res;

36 }

* 1. 二叉查找树

1 struct Node{

2 int x;

3 Node \*left, \*right;

4 Node(int x = 0) :x(x){

5 left = NULL;

6 right = NULL;

7 }

8 };

9 Node \*root;

10 int bst\_insert(int x){

11 if (root == NULL){

12 root = new Node(x);

13 return 1;

14 }

15 Node \*p = root;

16 int t = 1;

17 while (p != NULL){

18 if (x > p->x){

19 t = t << 1 | 1;

20 if (p->right == NULL){

21 p->right = new Node(x);

22 break;

23 }

24 else{

25 p = p->right;

26 }

27 }

28 else{

29 t = t << 1;

30 if (p->left == NULL){

31 p->left = new Node(x);

32 break;

33 }

34 else{

35 p = p->left;

36 }

37 }

38 }

39 return t;

40 }

* 1. RMQ-ST

1 void rmq(int n)

2 {

3 for (int i = 1; i <= n; i++)

4 f[0][i] = deep[i];

5 for (int j = 1; j <= (int)(log((double)n) / log(2.0)); j++){

6 for (int i = 1; i <= n - (1 << j) + 1; i++){

7 f[j][i] = min(f[j - 1][i], f[j - 1][i + (1 << (j - 1))]);

8 }

9 }

10 }

* 1. 树链剖分
  2. 函数式线段树

1. 字符串
   1. KMP

1 int n, last[MAXN], j, m = 0;

2 char s[MAXN];

3 int main()

4 {

5 while (~scanf("%d", &n)){

6 if (n == 0) break;

7 memset(last, 0, sizeof(last));

8 scanf("%s", s + 1);

9

10 int k = 0;

11 last[1] = 0;

12 for (int i = 2; i <= n; i++){

13 while (s[k + 1] != s[i] && k > 0){

14 k = last[k];

15 }

16 if (s[k + 1] == s[i]){

17 k++;

18 }

19 last[i] = k;

20 }

21 printf("Test case #%d\n", ++m);

22 for (int i = 2; i <= n; i++){

23 j = i - last[i];

24 if (i % j == 0 && i > j){

25 printf("%d %d\n", i, i / j);

26 }

27 }

28 printf("\n");

29 }

30 }

* 1. 后缀数组

1 char str[MAXN];

2 int m, dn;

3 int s[MAXN], sa[MAXN], height[MAXN], p[MAXN], q[MAXN], c[MAXN];

4 int ranK[MAXN];

5 int f[MAXN \* 2][45];

6

7 void getSa(){

8 for (int i = 0; i < dn; i++){

9 p[i] = s[i];

10 c[s[i]]++;

11 }

12 for (int i = 1; i < m; i++){

13 c[i] += c[i - 1];

14 }

15 for (int i = dn - 1; i >= 0; i--){

16 sa[--c[p[i]]] = i;

17 }

18 for (int k = 1; k <= dn; k \*= 2){

19 int t = 0;

20 for (int i = dn - k; i < dn; i++){

21 q[t++] = i;

22 }

23 for (int i = 0; i < dn; i++){

24 if (sa[i] >= k){

25 q[t++] = sa[i] - k;

26 }

27 }

28 for (int i = 0; i < m; i++){

29 c[i] = 0;

30 }

31 for (int i = 0; i < dn; i++){

32 c[p[q[i]]]++;

33 }

34 for (int i = 0; i < m; i++){

35 c[i] += c[i - 1];

36 }

37 for (int i = dn - 1; i >= 0; i--){

38 sa[--c[p[q[i]]]] = q[i];

39 }

40

41 swap(p, q);

42 t = 1;

43 p[sa[0]] = 0;

44 for (int i = 1; i < dn; i++){

45 if (q[sa[i - 1]] == q[sa[i]] && q[sa[i - 1] + k] == q[sa[i] + k]){

46 p[sa[i]] = t - 1;

47 }

48 else{

49 p[sa[i]] = t++;

50 }

51 }

52 if (t >= dn) break;

53 m = t;

54 }

55

56 }

57

58 void getHeight(){

59 for (int i = 0; i < dn; i++){

60 ranK[sa[i]] = i;

61 }

62 int k = 0;

63 for (int i = 0; i < dn; i++){

64 if (k != 0) k--;

65 int j = sa[ranK[i] - 1];

66 while (s[i + k] == s[j + k]){

67 k++;

68 }

69 height[ranK[i]] = k;

70 }

71 }

* 1. AC自动机

1 int cnt[200];

2 char s[MAXM];

3 char words[160][100];

4 int n, ans;

5

6 struct Node

7 {

8 int count, id;

9 struct Node \*next[26];

10 struct Node \*fail;

11 void init(){

12 int i;

13 for (int i = 0; i < 26; i++){

14 next[i] = NULL;

15 }

16 count = -1;

17 fail = NULL;

18 id = -1;

19 }

20 };

21 Node \*root, \*d[MAXM];

22

23

24 void insert(char \*s, int id){

25 int len, k;

26 Node \*p = root;

27 len = strlen(s);

28 for (k = 0; k < len; k++){

29 int pos = s[k] - 'a';

30 if (p->next[pos] == NULL){

31 p->next[pos] = new Node;

32 p->next[pos]->init();

33 p = p->next[pos];

34 }

35 else

36 p = p->next[pos];

37 }

38 p->count = id;

39 }

40

41 void build(Node \*root){

42 int head, tail, i;

43 Node \*p, \*temp;

44 head = 0;

45 tail = 0;

46 root->fail = NULL;

47 d[head] = root;

48 while (head <= tail){

49 temp = d[head++];

50 for (int i = 0; i < 26; i++){

51 if (temp->next[i] == NULL) continue;

52 if (temp == root){

53 temp->next[i]->fail = root;

54 }

55 else{

56 p = temp->fail;

57 while (p != NULL){

58 if (p->next[i] != NULL){

59 temp->next[i]->fail = p->next[i];

60 break;

61 }

62 p = p->fail;

63 }

64 if (p == NULL){

65 temp->next[i]->fail = root;

66 }

67 }

68 d[++tail] = temp->next[i];

69 }

70 }

71 }

72

73 void query(){

74 int len = strlen(s);

75 Node \*p, \*temp;

76 p = root;

77 for (int i = 0; i < len; i++){

78 int pos = s[i] - 'a';

79 while (!p->next[pos] && p != root) p = p->fail;

80 p = p->next[pos];

81 if (!p) p = root;

82 temp = p;

83 while (temp != root){

84 if (temp->count >= 0){

85 cnt[temp->count]++;

86 }

87 temp = temp->fail;

88 }

89 }

90 }

91

92

93 int main()

94 {

95 while (~scanf("%d", &n)){

96 if (n == 0) break;

97 memset(cnt, 0, sizeof(cnt));

98 root = new Node;

99 root->init();

100 for (int i = 0; i < n; i++){

101 scanf("%s", &words[i]);

102 insert(words[i], i);

103 }

104 build(root);

105 scanf("%s", s);

106 query();

107 ans = -1;

108 for (int i = 0; i < n; i++){

109 if (cnt[i] >ans){

110 ans = cnt[i];

111 }

112 }

113 printf("%d\n", ans);

114 for (int i = 0; i < n; i++){

115 if (cnt[i] == ans){

116 printf("%s\n", words[i]);

117 }

118 }

119 }

120 }

* 1. 后缀自动机
  2. Manacher算法

1 char s[MAXN << 1], ss[MAXN];

2 int p[MAXN << 1];

3

4 int manacher(int m){

5 s[0] = '@';

6 s[1] = '#';

7 for (int i = 1, j = 2; i <= m; i++, j += 2){

8 s[j] = ss[i];

9 s[j + 1] = '#';

10 }

11 m = (m << 1) + 1;

12 int res = 0, id = 0;

13 int ans = 0;

14 for (int i = 1; i <= m; i++) {

15 if (res > i){

16 p[i] = min(p[2 \* id - i], res - i);

17 }

18 else{

19 p[i] = 1;

20 }

21 while (s[i + p[i]] == s[i - p[i]]){

22 p[i]++;

23 }

24 if (i + p[i] > res) {

25 res = i + p[i];

26 id = i;

27 }

28 ans = max(p[i], ans);

29 }

30 return ans - 1;

31 }

* 1. 字符串最小表示

1 int min\_max\_express(char s[], int m, bool flag){

2 //s从0开始, flag=true时最小表示

3 int i = 0, j = 1, k = 0;

4 while (i < m && j < m && k < m){

5 int p = s[(j + k) % m] - s[(i + k) % m];

6 if (p == 0){

7 k++;

8 }

9 else{

10 if (flag){

11 if (p > 0){

12 j += k + 1;

13 }

14 else{

15 i += k + 1;

16 }

17 }

18 else{

19 if (p > 0){

20 i += k + 1;

21 }

22 else{

23 j += k + 1;

24 }

25 }

26 if (i == j){

27 j++;

28 }

29 k = 0;

30 }

31 }

32 return min(i, j);

33 }

1. 图论

3.1 Dijstra

1 struct Edge

2 {

3 int from, to, dist;

4 Edge(int from, int to, int dist):from(from), to(to), dist(dist){};

5 };

6 struct HeapNode

7 {

8 int d, u;

9 HeapNode(int d, int u):d(d), u(u){};

10 bool operator <(const HeapNode& rhs) const{

11 return d > rhs.d;

12 }

13 };

14 struct Dijstra

15 {

16 int n, m;

17 vector<Edge> edges;

18 vector<int> G[MAXN];

19 bool done[MAXN];

20 int d[MAXN];

21 int p[MAXN];

22

23 void init(int n){

24 this->n = n;

25 for(int i = 0; i <= n; i++){

26 G[i].clear();

27 road[i].clear();

28 }

29 edges.clear();

30 }

31

32 void AddEdge(int from, int to, int dist){

33 edges.push\_back(Edge(from, to, dist));

34 m = edges.size();

35 G[from].push\_back(m - 1);

36 }

37

38 void dijstra(int s){

39 priority\_queue<HeapNode> Q;

40 for(int i = 0; i <= n; i++){

41 d[i] = INF;

42 }

43 d[s] = 0;

44 memset(done, 0, sizeof(done));

45 Q.push(HeapNode(0, s));

46 while(!Q.empty()){

47 HeapNode x = Q.top();

48 Q.pop();

49 int u = x.u;

50 if(done[u]) continue;

51 done[u] = true;

52 for(int i = 0; i < G[u].size(); i++){

53 Edge& e = edges[G[u][i]];

54 if(d[e.to] > d[u] + e.dist){

55 d[e.to] = d[u] + e.dist;

56 p[e.to] = G[u][i];

57 Q.push(HeapNode(d[e.to], e.to));

58 }

59 }

60 }

61 }

62 };

3.2 Prim

1 int prim() {

2 memset(vis,0,sizeof(vis));

3 int i;

4 int maxedge=0;

5 for (i = 1; i <= n; i++) {

6 dis[i]= value[1][i];

7 }

8 dis[1] = 0;

9 vis[1] = true;

10 for (i = 2; i <= n; i++) {

11 int temp = inf;

12 int mark;

13 for (int j = 1; j <= n; j++) {

14 if (!vis[j] && dis[j] < temp) {

15 temp = dis[j];

16 mark = j;

17 }

18 }

19 if(dis[mark]>maxedge)

20 maxedge=dis[mark];

21 vis[mark]=true;

22 for (int j = 1; j <= n; j++) {

23 if (!vis[j]&&dis[j]>value[mark][j])

24 dis[j] = value[mark][j];

25 }

26 }

27 return maxedge;

28 }

3.3 Spfa

1 int spfa(int s)

2 {

3 queue <int> q;

4 memset(d, INF, sizeof(d));

5 d[s] = 0;

6 memset(cnt, 0, sizeof(cnt));

7 memset(vis, 0, sizeof(vis));

8 q.push(s);

9 vis[s] = 1;

10 while (!q.empty())

11 {

12 int x;

13 x = q.front();

14 q.pop();

15 while (no[x]){

16 x = q.front();

17 q.pop();

18 }

19 vis[x] = 0;

20 for (int i = 0; i < G[x].size(); i++)

21 {

22 int y = G[x][i].v;

23 if (d[x] + G[x][i].w < d[y])

24 {

25 d[y] = d[x] + G[x][i].w;

26 if (!vis[y])

27 {

28 vis[y] = 1;

29 q.push(y);

30 }

31 }

32 }

33 }

34 }

3.4 无向图的割顶和桥

割顶

桥

1 int pre[MAXN], isbridge[MAXM], low[MAXN];

2 vector<Edge> G[MAXN];

3 int dfs\_clock;

4 int dfs(int u, int father){

5 int lowu = pre[u] = ++dfs\_clock;

6 //int child = 0;

7 for (int i = 0; i < G[u].size(); i++){

8 int v = G[u][i].to;

9 if (!pre[v]){

10 //child++;

11 int lowv = dfs(v, G[u][i].pos);

12 lowu = min(lowu, lowv);

13 if (lowv > pre[u]){

14 isbridge[G[u][i].pos] = true;

15 }

16 }

17 else if (pre[v] < pre[u] && G[u][i].pos != father){

18 lowu = min(lowu, pre[v]);

19 }

20 }

21 low[u] = lowu;

22 return lowu;

23 }

3.5 Dinic最大流

1 struct Edge{

2 int from, to, cap, flow;

3 //Edge(int u, int v, int c, int f) :from(u), to(v), cap(c), flow(f){};

4 };

5 bool comp(const Edge& a, const Edge& b){

6 return (a.from < b.from || (a.from == b.from && a.to < b.to));

7 }

8 struct Dinic{

9 int n, m, i, s, t;

10 Edge e;

11 vector<Edge> edges;

12 vector<int> G[MAXN];

13 int d[MAXN], cur[MAXN];

14 bool vis[MAXN];

15 void init(int n){

16 this->n = n;

17 for (i = 0; i <= n; i++){

18 G[i].clear();

19 }

20 edges.clear();

21 }

22 void AddEdge(int from, int to, int cap){

23 edges.push\_back(Edge{ from, to, cap, 0 });

24 edges.push\_back(Edge{ to, from, 0, 0 });

25 m = edges.size();

26 G[from].push\_back(m - 2);

27 G[to].push\_back(m - 1);

28 }

29 bool BFS(){

30 memset(vis, 0, sizeof(vis));

31 queue<int> Q;

32 Q.push(s);

33 d[s] = 0;

34 vis[s] = 1;

35 while (!Q.empty()){

36 int x = Q.front();

37 Q.pop();

38 for (i = 0; i < G[x].size(); i++){

39 Edge& e = edges[G[x][i]];

40 if (!vis[e.to] && e.cap > e.flow){

41 vis[e.to] = true;

42 d[e.to] = d[x] + 1;

43 Q.push(e.to);

44 }

45 }

46 }

47 return vis[t];

48 }

49 int DFS(int x, int a){

50 if (x == t || a == 0) return a;

51 int flow = 0, f;

52 for (int& i = cur[x]; i < G[x].size(); i++){

53 Edge& e = edges[G[x][i]];

54 if (d[x] + 1 == d[e.to] && (f = DFS(e.to, min(a, e.cap - e.flow))) > 0){

55 e.flow += f;

56 edges[G[x][i] ^ 1].flow -= f;

57 flow += f;

58 a -= f;

59 if (a == 0) break;

60 }

61 }

62 return flow;

63 }

64 int MaxFlow(int s, int t, int need){

65 int flow = 0;

66 this->s = s;

67 this->t = t;

68 while (BFS()){

69 memset(cur, 0, sizeof(cur));

70 flow += DFS(s, INF);

71 if (flow > need) return flow;

72 }

73 return flow;

74 }

75 bool checkFull(int s){

76 for (int i = 0; i < G[s].size(); i++){

77 if (edges[G[s][i]].flow != edges[G[s][i]].cap){

78 return false;

79 }

80 }

81 return true;

82 }

83 };

3.6 最小费用最大流

1 #include <iostream>

2 #include <string.h>

3 #include <stdio.h>

4 #include <algorithm>

5 #include <queue>

6 #define V 10100

7 #define E 1000100

8 #define inf 99999999

9 using namespace std;

10 int vis[V];

11 int dist[V];

12 int pre[V];

13

14 struct Edge{

15 int u,v,c,cost,next;

16 }edge[E];

17 int head[V],cnt;

18

19 void init(){

20 cnt=0;

21 memset(head,-1,sizeof(head));

22 }

23 void addedge(int u,int v,int c,int cost)

24 {

25 edge[cnt].u=u;edge[cnt].v=v;edge[cnt].cost=cost;

26 edge[cnt].c=c;edge[cnt].next=head[u];head[u]=cnt++;

27

28 edge[cnt].u=v;edge[cnt].v=u;edge[cnt].cost=-cost;

29 edge[cnt].c=0;edge[cnt].next=head[v];head[v]=cnt++;

30 }

31

32 bool spfa(int begin,int end){

33 int u,v;

34 queue<int> q;

35 for(int i=0;i<=end+2;i++){

36 pre[i]=-1;

37 vis[i]=0;

38 dist[i]=inf;

39 }

40 vis[begin]=1;

41 dist[begin]=0;

42 q.push(begin);

43 while(!q.empty()){

44 u=q.front();

45 q.pop();

46 vis[u]=0;

47 for(int i=head[u];i!=-1;i=edge[i].next){

48 if(edge[i].c>0){

49 v=edge[i].v;

50 if(dist[v]>dist[u]+edge[i].cost){

51 dist[v]=dist[u]+edge[i].cost;

52 pre[v]=i;

53 if(!vis[v]){

54 vis[v]=true;

55 q.push(v);

56 }

57 }

58 }

59 }

60 }

61 return dist[end]!=inf;

62 }

63

64 int MCMF(int begin,int end){

65 int ans=0,flow;

66 int flow\_sum=0;

67 while(spfa(begin,end)){

68 flow=inf;

69 for(int i=pre[end];i!=-1;i=pre[edge[i].u])

70 if(edge[i].c<flow)

71 flow=edge[i].c;

72 for(int i=pre[end];i!=-1;i=pre[edge[i].u]){

73 edge[i].c-=flow;

74 edge[i^1].c+=flow;

75 }

76 ans+=dist[end];

77 flow\_sum += flow;

78 }

79 //cout << flow\_sum << endl;

80 return ans;

81 }

82

83 int main()

84 {

85 //freopen("in.txt","r",stdin);

86 int n,m,a,b,c;

87 while(scanf("%d%d",&n,&m)!=EOF){

88 init();

89 addedge(0,1,2,0);

90 addedge(n,n+1,2,0);

91 for(int i=1;i<=m;i++){

92 scanf("%d%d%d",&a,&b,&c);

93 addedge(a,b,1,c);

94 addedge(b,a,1,c);

95 }

96 printf("%d\n",MCMF(0,n+1));

97 }

98 return 0;

99 }

3.7 KM算法

3.8 Two-SAT

1 struct TwoSat{

2 int n;

3 vector<int> G[MAXN\*2];

4 bool mark[MAXN\*2];

5 int S[MAXN\*2], c;

6

7 bool dfs(int x){

8 if(mark[x^1]) return false;

9 if(mark[x]) return true;

10 mark[x] = true;

11 S[c++] = x;

12 for(int i = 0; i < G[x].size(); i++){

13 if(!dfs(G[x][i])) return false;

14 }

15 return true;

16 }

17

18 void init(int n){

19 this->n = n;

20 for(int i = 0; i < n \* 2; i++){

21 G[i].clear();

22 }

23 memset(mark, 0, sizeof(mark));

24 }

25

26 void add\_clause(int x, int xval, int y, int yval){

27 x = x \* 2 + xval;

28 y = y \* 2 + yval;

29 G[x^1].push\_back(y);

30 G[y^1].push\_back(x);

31 }

32

33 bool solve(){

34 for(int i = 0; i < n \* 2; i += 2){

35 if(!mark[i] && !mark[i + 1]){

36 c = 0;

37 if(!dfs(i)){

38 while(c > 0){

39 mark[S[--c]] = false;

40 }

41 if(!dfs(i + 1)){

42 return false;

43 }

44 }

45 }

46 }

47 return true;

48 }

49 };

3.9 Splay树

3.11 有向图强连通分量

1 int pre[MAXN], lowlink[MAXN], sccno[MAXN];

2 vector<int> G[MAXN];

3 stack<int> S;

4 int dfs\_clock, scc\_cnt;

5 void dfs(int u){

6 pre[u] = lowlink[u] = ++dfs\_clock;

7 S.push(u);

8 for (int i = 0; i < G[u].size(); i++){

9 int v = G[u][i];

10 if (!pre[v]){

11 dfs(v);

12 lowlink[u] = min(lowlink[u], lowlink[v]);

13 }

14 else{

15 if (!sccno[v]){

16 lowlink[u] = min(lowlink[u], pre[v]);

17 }

18 }

19 }

20 if (lowlink[u] == pre[u]){

21 scc\_cnt++;

22 for (;;){

23 int x = S.top();

24 S.pop();

25 sccno[x] = scc\_cnt;

26 if (x == u) break;

27 }

28 }

29 }

30 void find\_scc(int n){

31 dfs\_clock = scc\_cnt = 0;

32 memset(sccno, 0, sizeof(sccno));

33 memset(pre, 0, sizeof(pre));

34 for (int i = 1; i <= n; i++){

35 if (!pre[i]){

36 dfs(i);

37 }

38 }

39 }

3.12 无向图点双连通分量

1 struct Edge{

2 int from, to;

3 Edge(int from = 0, int to = 0) :from(from), to(to){};

4 };

5 vector<int> G[MAXN], bcc[MAXN];

6 int pre[MAXN], iscut[MAXN], bccno[MAXN], pre[MAXN];

7 stack<Edge> S;

8 int dfs\_clock, bcc\_cnt, bridge\_cnt;

9 int dfs(int u, int fa)

10 {

11 int lowu = pre[u] = ++dfs\_clock;

12 int child = 0;

13 for (int i = 0; i < G[u].size(); i++){

14 int v = G[u][i];

15 Edge e = Edge(u, v);

16 if (!pre[v]){

17 S.push(e);

18 child++;

19 int lowv = dfs(v, u);

20 lowu = min(lowu, lowv);

21 if (lowv >= pre[u]){

22 iscut[u] = true;

23 bcc\_cnt++;

24 bcc[bcc\_cnt].clear();

25 while (true){

26 Edge x = S.top(); S.pop();

27 if (bccno[x.from] != bcc\_cnt){

28 bcc[bcc\_cnt].push\_back(x.from);

29 bccno[x.from] = bcc\_cnt;

30 }

31 if (bccno[x.to] != bcc\_cnt){

32 bcc[bcc\_cnt].push\_back(x.to);

33 bccno[x.to] = bcc\_cnt;

34 }

35 if (x.from == u && x.to == v) break;

36 }

37 }

38 }

39 else if (pre[v] < pre[u] && v != fa){

40 S.push(e);

41 lowu = min(lowu, pre[v]);

42 }

43 }

44 if (fa < 0 && child == 1) iscut[u] = false;

45 return lowu;

46 }

47 void find\_bcc(int n)

48 {

49 memset(pre, 0, sizeof(pre));

50 memset(iscut, 0, sizeof(iscut));

51 memset(bccno, 0, sizeof(bccno));

52 dfs\_clock = bcc\_cnt = 0;

53 bridge\_cnt = 0;

54 for (int i = 0; i < n; i++){

55 if (!pre[i]) dfs(i, -1);

56 }

57 }

3.13 无向图边双连通分量

1 struct Edge{

2 int v, next;

3 }edge[MAXN \* 4];

4

5 int n, m, cnt, NE, bridge\_cnt;

6 int head[MAXN];

7 int parent[MAXN];

8 int isbridge[MAXN], low[MAXN], pre[MAXN];

9 bool mark[MAXN];

10

11 void add\_edge(int u, int v){

12 edge[NE].v = v;

13 edge[NE].next = head[u];

14 head[u] = NE++;

15 }

16

17 void dfs(int u, int father){

18 int flag = 0;

19 low[u] = pre[u] = ++cnt;

20 mark[u] = true;

21 for (int i = head[u]; i != -1; i = edge[i].next){

22 int v = edge[i].v;

23 if (v == father&&!flag){ flag = 1; continue; }

24 if (pre[v] == 0){

25 parent[v] = u;

26 dfs(v, u);

27 low[u] = min(low[u], low[v]);

28 if (low[v] > pre[u]){

29 isbridge[v] = 1;

30 bridge\_cnt++;

31 }

32 }

33 else if (mark[v]){

34 low[u] = min(low[u], pre[v]);

35 }

36 }

37 }

38 void find\_bcc(){

39 memset(isbridge, 0, sizeof(isbridge));

40 memset(pre, 0, sizeof(pre));

41 memset(mark, false, sizeof(mark));

42 for (int i = 1; i <= n; i++){

43 if (!pre[i])dfs(1, -1);

44 }

45 }

3.14 无向图全局最小割

1 int a[MAXN][MAXN], v[MAXN], dis[MAXN];

2 bool vis[MAXN];

3 int stoer\_wagner(int n){

4 int res = INF;

5 for (int i = 0; i < n; i++){

6 v[i] = i + 1;

7 }

8 while (n > 1){

9 int maxp = 1, pre = 0;

10 for (int i = 1; i < n; i++){

11 dis[v[i]] = a[v[0]][v[i]];

12 if (dis[v[i]] > dis[v[maxp]]){

13 maxp = i;

14 }

15 }

16 memset(vis, 0, sizeof(vis));

17 vis[v[0]] = true;

18 for (int i = 1; i < n; i++){

19 if (i == n - 1){

20 res = min(res, dis[v[maxp]]);

21 for (int j = 0; j < n; j++){

22 a[v[pre]][v[j]] += a[v[j]][v[maxp]];

23 a[v[j]][v[pre]] = a[v[pre]][v[j]];

24 }

25 v[maxp] = v[--n];

26 }

27 vis[v[maxp]] = true;

28 pre = maxp;

29 maxp = -1;

30 for (int j = 1; j < n; j++){

31 if (!vis[v[j]]){

32 dis[v[j]] += a[v[pre]][v[j]];

33 if (maxp == -1 || dis[v[maxp]] < dis[v[j]]){

34 maxp = j;

35 }

36 }

37 }

38 }

39 }

40 return res;

41 }