XCPC Template For Contest

Ayers Zhang

2022 年 10 月 31 日

目录

1 杂项

1.1 快速读入和快速输出

```
template <typename T>
void read(T &x) {
  x = 0; char c = 0; int f = 1;
  for (; !isdigit(c); c = getchar()) if (c == '-') f = -1;
  for (; isdigit(c); c = getchar()) x = x * 10 + (c & 15);
  x *= f;
}

template <typename T>
void write(T x) {
  if (x < 0) putchar('-'), x = -x;
  if (x > 9) write(x / 10);
  putchar(x % 10 + '0');
}
```

1.2

2 图论

2.1 最短路

```
struct Node {
  int u;
  i64 dis;
  Node() {}
  Node(int a, i64 b) : u(a), dis(b) {}
  bool operator<(const Node &rhs) const {
    return dis > rhs.dis;
  }
};
std::priority_queue<Node> q;
i64 dis[N];
bool vis[N];
void dijkstra(int s) {
  while (!q.empty()) q.pop();
  for (int i = 1; i <= n; i++) {
    dis[i] = inf, vis[i] = 0;
  }
  q.push(Node(s, 0));
  dis[s] = 0;
  while (!q.empty()) {
    int u = q.top().u;
    q.pop();
    if (vis[u]) continue;
    vis[u] = 1;
    for (int e = head[u]; e; e = edge[e].nxt) {
      int v = edge[e].v;
      if (dis[v] > dis[u] + edge[e].w) {
        dis[v] = dis[u] + edge[e].w;
        q.push(Node(v, dis[v]));
      }
    }
  }
}
```

2.2 DAG 最长路

由于带环图理论上不存在最长路，所以只考虑 DAG 中的最长路，采用拓扑排序进行求解。

```
void findLongestPath() {
  std::queue<int> q;
  for (int i = 0; i < n; i++) {
    if (ind[i] == 0) {
      q.push(i);
    }
  }
  dis[0] = inf;
  while (!q.empty()) {
    int u = q.front();
    q.pop();
    for (int e = head[u]; e; e = ed[e].nxt) {
```

```
      int v = ed[e].to;
      ind[v]--;
      dis[v] = std::max(dis[v], dis[u] + ed[e].w);
      if (ind[v] == 0) {
        q.push(v);
      }
    }
  }
}
```

2.3 网络流相关

2.3.1 最小费用最大流

Dinic 写法，在普通情况下时间复杂度为 $O(V^2E)$，二分图的时间复杂度为 $O(VE)$。

每条边单位流量会花费价值，在跑出最大流的情况下要求费用最小

```
const int N = 805, M = 320005;

struct Edge {
  int v;
  i64 w, c;
  int nxt;

  Edge() {}
  Edge(int a, i64 ws, i64 cs, int b) : v(a), w(ws), c(cs), nxt(b) {}
} edge[M];

int edgeCnt = 1;
int head[N], arc[N];
bool vis[N];

int S, T;

i64 dis[N];
i64 ans, ret;

void addEdge(int u, int v, i64 w, i64 c) {
  edge[++edgeCnt] = Edge(v, w, c, head[u]);
  head[u] = edgeCnt;
}

void add(int u, int v, i64 w, i64 c) {
  addEdge(u, v, w, c);
  addEdge(v, u, 0, -c);
}

bool spfa() {
  for (int i = S; i <= T; i++) {
    dis[i] = inf;
    arc[i] = head[i];
  }
  std::queue<int> q;
  dis[S] = 0;
  vis[S] = 1;
  q.push(S);
  while (!q.empty()) {
    int u = q.front();
    q.pop();
    vis[u] = 0;
    for (int e = head[u]; e; e = edge[e].nxt) {
      int v = edge[e].v;
      if (edge[e].w && dis[v] > dis[u] + edge[e].c) {
        dis[v] = dis[u] + edge[e].c;
        if (!vis[v]) {
          q.push(v);
```

```
          vis[v] = 1;
        }
      }
    }
  }
  return dis[T] != inf;
}

i64 dfs(int u, i64 flow) {
  if (u == T) {
    return flow;
  }
  vis[u] = 1;
  i64 res = 0;
  for (int &e = arc[u]; e && res < flow; e = edge[e].nxt) {
    int v = edge[e].v;
    if (!vis[v] && edge[e].w && dis[v] == dis[u] + edge[e].c) {
      i64 x = dfs(v, std::min(edge[e].w, flow - res));
      if (x) {
        edge[e].w -= x;
        edge[e ^ 1].w += x;
        res += x;
        ret += x * edge[e].c;
      }
    }
  }
  vis[u] = 0;
  return res;
}

void solve() {
  // 注意要事先设定好源点和汇点，源点为第一个点，汇点为最后一个点
  // 其中的 ans 就是最大流，ret 为最小费用
  memset(vis, 0, sizeof vis);
  ans = 0;
  ret = 0;
  while (spfa()) {
    i64 x;
    while ((x = dfs(S, inf))) {
      ans += x;
    }
  }
}
```

2.3.2 带权二分图最大匹配 KM

```
const int N = 805;

int n, m;
int matchx[N], matchy[N];
int pre[N];
bool visx[N], visy[N];
i64 slack[N];
i64 lx[N], ly[N];
std::queue<int> q;
```

```cpp
i64 g[N][N];

bool check(int u) {
  visy[u] = 1;
  if (matchy[u] != -1) {
    q.push(matchy[u]);
    visx[matchy[u]] = 1;
    return 0;
  }
  while (u != -1) {
    matchy[u] = pre[u];
    std::swap(u, matchx[pre[u]]);
  }
  return 1;
}

void add(int u, int v, i64 w) {
  g[u][v] = std::max(0ll, w);
}

void bfs(int i) {
  while (!q.empty()) {
    q.pop();
  }
  q.push(i);
  visx[i] = true;
  while (true) {
    while (!q.empty()) {
      int u = q.front();
      q.pop();
      for (int v = 1; v <= n; v++) {
        if (!visy[v]) {
          i64 delta = lx[u] + ly[v] - g[u][v];
          if (slack[v] >= delta) {
            pre[v] = u;
            if (delta) {
              slack[v] = delta;
            } else if (check(v)) {  // delta=0 代表有机会加入相等子图 找增广路
                                    // 找到就return 重建交错树
              return;
            }
          }
        }
      }
    }
    // 没有增广路 修改顶标
    i64 a = inf;
    for (int j = 1; j <= n; j++) {
      if (!visy[j]) {
        a = std::min(a, slack[j]);
      }
    }
    for (int j = 1; j <= n; j++) {
      if (visx[j]) {  // S
        lx[j] -= a;
      }
```

```
      if (visy[j]) {  // T
        ly[j] += a;
      } else {  // T'
        slack[j] -= a;
      }
    }
    for (int j = 1; j <= n; j++) {
      if (!visy[j] && slack[j] == 0 && check(j)) {
        return;
      }
    }
  }
}

i64 res;

void solve() {
  n = std::max(n, m);
  for (int i = 1; i <= n; i++) {
    matchx[i] = matchy[i] = -1;
    lx[i] = -inf;
    ly[i] = slack[i] = 0;
    pre[i] = 0;
    visx[i] = visy[i] = 0;
  }
  res = 0;
  for (int i = 1; i <= n; i++) {
    for (int j = 1; j <= n; j++) {
      lx[i] = std::max(lx[i], g[i][j]);
    }
  }
  for (int i = 1; i <= n; i++) {
    std::fill(slack + 1, slack + 1 + n, inf);
    std::fill(visx + 1, visx + 1 + n, false);
    std::fill(visy + 1, visy + 1 + n, false);
    bfs(i);
  }
  for (int i = 1; i <= n; i++) {
    if (g[i][matchx[i]] > 0) {
      res += g[i][matchx[i]];
    }
  }
}
```

3 数学

3.1 数论

3.2 组合数学

3.2.1 球盒问题模型

n 个球	r 个盒子	是否允许有空盒	方案数
不相同	不相同	允许	r^n
相同	不相同	不允许	$\binom{n-1}{r-1}$
相同	不相同	允许	$\binom{n+r-1}{r-1}$
不相同	不相同	不允许	$r! \times S(n,r)$
不相同	相同	不允许	$S(n,r)$
不相同	相同	允许	$\Sigma_{k=1}^{r} S(n,k)$

4 数据结构

4.1 ST 表解决静态 RMQ 问题

```
template <typename T>
class sparseTable {
private:
  std::vector< std::vector<T> > st;

  static T func(const T &lhs, const T &rhs) {
    return std::max(lhs, rhs);
  }

  std::function<T(const T &, const T &)> judge;

public:
  sparseTable(const std::vector<T> &v, std::function<T(const T &, const T &)> _func = func) {
    judge = _func;
    int n = v.size(), k = ceil(log2(n)) + 1;
    st.assign(n, std::vector<T>(k, 0));
    for (int i = 0; i < n; i++) {
      st[i][0] = v[i];
    }
    for (int j = 1; j < k; j++) {
      for (int i = 0; i + (1 << j) - 1 < n; i++) {
        st[i][j] = judge(st[i][j - 1], st[i + (1 << (j - 1))][j - 1]);
      }
    }
  }

  T query(int l, int r) {
    int k = log2(r - l + 1);
    return judge(st[l][k], st[r - (1 << k) + 1][k]);
  }
};
```