Team notebook

Universidad Tecnologica de Pereira

March 30, 2018

Contents

1 Dynamic Programming

1.1 convex hull trick

```
struct line {
  long long m, b;
  line (long long a, long long c) : m(a), b(c) {}
  long long eval(long long x) {
    return m * x + b;
  }
};

long double inter(line a, line b) {
  long double den = a.m - b.m;
  long double num = b.b - a.b;
  return num / den;
}

/**
 *  min m_i * x_j + b_i, for all i.
 *     x_j <= x_{j + 1}
 *     m_i >= m_{j + 1}
 * */
struct ordered_cht {
  vector<line> ch;
  int idx; // id of last "best" in query
  ordered_cht() {
    idx = 0;
  }

  void insert_line(long long m, long long b) {
    line cur(m, b);
    // new line's slope is less than all the previous
    while (ch.size() > 1 &&
       (inter(cur, ch[ch.size() - 2]) >= inter(cur, ch[ch.size() - 1]))) {
       // f(x) is better in interval [inter(ch.back(), cur), inf)
       ch.pop_back();
    }

    ch.push_back(cur);
  }

  long long eval(long long x) { // minimum
    // current x is greater than all the previous x,
    // if that is not the case we can make binary search.
    idx = min<int>(idx, ch.size() - 1);
    while (idx + 1 < (int)ch.size() && ch[idx + 1].eval(x) <=
        ch[idx].eval(x))
      idx++;

    return ch[idx].eval(x);
  }
};


/**
 *  Dynammic convex hull trick
 * */
```

```
typedef long long int64;
typedef long double float128;

const int64 is_query = -(1LL<<62), inf = 1e18;

struct Line {
  int64 m, b;
  mutable function<const Line*()> succ;
  bool operator<(const Line& rhs) const {
    if (rhs.b != is_query) return m < rhs.m;
    const Line* s = succ();
    if (!s) return 0;
    int64 x = rhs.m;
    return b - s->b < (s->m - m) * x;
  }
};

struct HullDynamic : public multiset<Line> { // will maintain upper hull
    for maximum
  bool bad(iterator y) {
    auto z = next(y);
    if (y == begin()) {
      if (z == end()) return 0;
      return y->m == z->m && y->b <= z->b;
    }
    auto x = prev(y);
    if (z == end()) return y->m == x->m && y->b <= x->b;
    return (float128)(x->b - y->b)*(z->m - y->m) >= (float128)(y->b -
        z->b)*(y->m - x->m);
  }
  void insert_line(int64 m, int64 b) {
    auto y = insert({ m, b });
    y->succ = [=] { return next(y) == end() ? 0 : &*next(y); };
    if (bad(y)) { erase(y); return; }
    while (next(y) != end() && bad(next(y))) erase(next(y));
    while (y != begin() && bad(prev(y))) erase(prev(y));
  }

  int64 eval(int64 x) {
    auto l = *lower_bound((Line) { x, is_query });
    return l.m * x + l.b;
  }
};
```

2 Graphs

2.1 dinic

```
// taken from
    https://github.com/jaehyunp/stanfordacm/blob/master/code/MinCostMaxFlow.cc
typedef long long LL;

struct edge {
  int u, v;
  LL cap, flow;
  edge() {}
  edge(int u, int v, LL cap): u(u), v(v), cap(cap), flow(0) {}
};

struct dinic {
  int N;
  vector<edge> E;
  vector<vector<int>> g;
  vector<int> d, pt;

  dinic(int N): N(N), E(0), g(N), d(N), pt(N) {}

  void add_edge(int u, int v, LL cap) {
    if (u != v) {
      E.emplace_back(edge(u, v, cap));
      g[u].emplace_back(E.size() - 1);
      E.emplace_back(edge(v, u, 0));
      g[v].emplace_back(E.size() - 1);
    }
  }

  bool bfs(int S, int T) {
    queue<int> q({S});
    fill(d.begin(), d.end(), N + 1);
    d[S] = 0;
    while(!q.empty()) {
      int u = q.front(); q.pop();
      if (u == T) break;
      for (int k: g[u]) {
        edge &e = E[k];
        if (e.flow < e.cap && d[e.v] > d[e.u] + 1) {
          d[e.v] = d[e.u] + 1;
          q.emplace(e.v);
```

```
      }
    }
  }
  return d[T] != N + 1;
}

LL dfs(int u, int T, LL flow = -1) {
  if (u == T || flow == 0) return flow;
  for (int &i = pt[u]; i < int(g[u].size()); ++i) {
    edge &e = E[g[u][i]];
    edge &oe = E[g[u][i]^1];
    if (d[e.v] == d[e.u] + 1) {
      LL amt = e.cap - e.flow;
      if (flow != -1 && amt > flow) amt = flow;
      if (LL pushed = dfs(e.v, T, amt)) {
        e.flow += pushed;
        oe.flow -= pushed;
        return pushed;
      }
    }
  }
  return 0;
}

LL max_flow(int S, int T) {
  LL total = 0;
  while (bfs(S, T)) {
    fill(pt.begin(), pt.end(), 0);
    while (LL flow = dfs(S, T))
      total += flow;
  }
  return total;
}
};
```