Contents

1 Data Structure

1.1 BIT

```cpp
#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;
}
```

1.2 Segment tree

```cpp
int dfs(int lef, int rig){
    if(lef + 2 == rig){
        if(num[lef] > num[rig-1]){
            return lef;
        }
        else{
            return rig-1;
        }
    }
    int mid = (lef + rig)/2;
    int p1 = dfs(lef, mid);
    int p2 = dfs(mid, rig);
    if(num[p1] > num[p2]){
        return p1;
    }
    else{
        return p2;
    }
}
```

1.3 Trie

```cpp
const int MAXL = ; // 自己填
const int MAXC = ;
struct Trie {
  int nex[MAXL][MAXC];
  int len[MAXL];
  int sz;
  void init() {
    memset(nex, 0, sizeof(nex));
    memset(len, 0, sizeof(len));
    sz = 0;
  }
  void insert(const string &str) {
    int p = 0;
    for (char c : str) {
      int id = c - 'a';
      if (!nex[p][id]) {
        nex[p][id] = ++sz;
      }
      p = nex[p][id];
    }
    len[p] = str.length();
  }
  vector<int> find(const string &str, int i) {
    int p = 0;
    vector<int> ans;
    for (; i < str.length(); i++) {
      int id = str[i] - 'a';
      if (!nex[p][id]) {
        return ans;
      }
      p = nex[p][id];
      if (len[p]) {
        ans.pb(len[p]);
      }
    }
    return ans;
  }
};
```

2 DP

2.1 LCS

```cpp
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];
}
```

2.2 LIS

```cpp
int LIS(vector<int> &a) { // Longest Increasing
    Subsequence
  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();
}
```

3 Graph

3.1 SPFA

```cpp
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;
}
```

3.2 Dijkstra

```cpp
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 <= n; 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;
        }
        // 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;
                pq.push({e.v, dis[e.v]});
            }
        }
    }
}
```

3.3 Floyd Warshall

```cpp
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]);
            }
        }
    }
}
```

3.4 Disjoint set Kruskal

```cpp
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++;
    }
}
```

3.5 KM

```cpp
const int X = 50;    // X的點數，等於Y的點數
const int Y = 50;    // Y的點數
int adj[X][Y];       // 精簡過的adjacency matrix
int lx[X], ly[Y];    // vertex labeling
int mx[X], my[Y];    // X各點的配對對象、Y各點的配對對象
int q[X], *qf, *qb; // BFS queue
```

```
int p[X];           // BFS
    parent，交錯樹之偶點，指向上一個偶點
bool vx[X], vy[Y];  // 記錄是否在交錯樹上
int dy[Y], pdy[Y];  // 表格

void relax(int x){ // relaxation
    for (int y=0; y<Y; ++y)
        if (adj[x][y] != 1e9)
            if (lx[x] + ly[y] - adj[x][y] < dy[y]){
                dy[y] = lx[x] + ly[y] - adj[x][y];
                pdy[y] = x; //
                    記錄好是從哪個樹葉連出去的
            }
}
void reweight(){ // 調整權重、調整表格
    int d = 1e9;
    for (int y=0; y<Y; ++y) if (!vy[y]) d = min(d,
        dy[y]);
    for (int x=0; x<X; ++x) if ( vx[x]) lx[x] -= d;
    for (int y=0; y<Y; ++y) if ( vy[y]) ly[y] += d;
    for (int y=0; y<Y; ++y) if (!vy[y]) dy[y] -= d;
}
void augment(int x, int y){ // 擴充路徑
    for (int ty; x != -1; x = p[x], y = ty){
        ty = mx[x]; my[y] = x; mx[x] = y;
    }
}
bool branch1(){ // 延展交錯樹：使用既有的等邊
    while (qf < qb)
        for (int x=*qf++, y=0; y<Y; ++y)
            if (!vy[y] && lx[x] + ly[y] == adj[x][y]){
                vy[y] = true;
                if (my[y] == -1){
                    augment(x, y);
                    return true;
                }
                int z = my[y];
                *qb++ = z; p[z] = x; vx[z] = true;
                    relax(z);
            }
    return false;
}
bool branch2(){ // 延展交錯樹：使用新添的等邊
    for (int y=0; y<Y; ++y){
        if (!vy[y] && dy[y] == 0){
            vy[y] = true;
            if (my[y] == -1){
                augment(pdy[y], y);
                return true;
            }
            int z = my[y];
            *qb++ = z; p[z] = pdy[y]; vx[z] = true;
                relax(z);
        }
    }
    return false;
}
int Hungarian(){
    // 初始化vertex labeling
    //  memset(lx, 0, sizeof(lx));  // 任意值皆可
    memset(ly, 0, sizeof(ly));
    for (int x=0; x<X; ++x)
        for (int y=0; y<Y; ++y)
            lx[x] = max(lx[x], adj[x][y]);

    // X側每一個點，分別建立等邊交錯樹。
    memset(mx, -1, sizeof(mx));
    memset(my, -1, sizeof(my));
    for (int x=0; x<X; ++x){
        memset(vx, false, sizeof(vx));
        memset(vy, false, sizeof(vy));
        memset(dy, 0x7f, sizeof(dy));
        qf = qb = q;
        *qb++ = x; p[x] = -1; vx[x] = true; relax(x);
        while (true){
            if (branch1()) break;
            reweight();
            if (branch2()) break;
        }
    }
    // 計算最大權完美匹配的權重
    int weight = 0;
    for (int x=0; x<X; ++x)
        weight += adj[x][mx[x]];
    return weight;
}
```

3.6 Dinic

```
// Maximum Flow
const int V = 100, E = 1000;
int adj[V]; // adjacency lists，初始化為-1。
struct Element {int b, r, next;} e[E*2];
int en = 0;
void addedge(int a, int b, int c){
    e[en] = (Element){b, c, adj[a]}; adj[a] = en++;
    e[en] = (Element){a, 0, adj[b]}; adj[b] = en++;
}
int d[V];       // 最短距離
bool visit[V];  // BFS/DFS visit record
int q[V];       // queue
int BFS(int s, int t){ // 計算最短路徑，求出容許圖
    memset(d, 0x7f, sizeof(d));
    memset(visit, false, sizeof(visit));
    int qn = 0;
    d[s] = 0;
    visit[s] = true;
    q[qn++] = s;

    for (int qf=0; qf<qn; ++qf){
        int a = q[qf];
        for (int i = adj[a]; i != -1; i = e[i].next){
            int b = e[i].b;
            if (e[i].r > 0 && !visit[b]){
                d[b] = d[a] + 1;
                visit[b] = true;
                q[qn++] = b;
                if (b == t) return d[t];
            }
        }
    }
    return V;
}
int DFS(int a, int df, int s, int t){ //
    求出一條最短擴充路徑，並擴充流量
    if (a == t) return df;
    if (visit[a]) return 0;
    visit[a] = true;
    for (int i = adj[a]; i != -1; i = e[i].next){
        int b = e[i].b;
        if (e[i].r > 0 && d[a] + 1 == d[b]){
            int f = DFS(b, min(df, e[i].r), s, t);
            if (f){
                e[i].r -= f;
                e[i^1].r += f;
                return f;
            }
        }
    }
    return 0;
}
int dinitz(int s, int t){
    int flow = 0;
    while (BFS(s, t) < V)
        while (true){
            memset(visit, false, sizeof(visit));
            int f = DFS(s, 1e9, s, t);
            if (!f) break;
            flow += f;
        }
    return flow;
```

```
62 }
```