Homework #4 Solution

Contact TAs: `ada@csie.ntu.edu.tw`

Problem 1 (18%)

List[0:n] represents the range of [0, n] and invalid list and empty list return NULL.

Algorithm 1 Reconstruction from pre-order and in-order list

(1)
```
procedure AddNodeToCurrPos(PreList, PostList, currentPos)
    if $PreList = NULL$ then
        $currentPos.value \leftarrow NULL$
        return
    $currentPos.value \leftarrow PreList[0]$
    $preListCut \leftarrow PreList.getPos(PostList[0])$
    $postListCut \leftarrow PostList.getPos(PreList[1])$
    $leftPreList \leftarrow PreList[1 : preListCut + 1]$
    $rightPreList \leftarrow PreList[preListCut + 1 : Length(PreList)]$
    $leftPostList \leftarrow PostList[0 : postListCut + 1]$
    $rightPostList \leftarrow PostList[postListCut + 1 : Length(PostList) - 1]$
    AddNodeToCurrPos($leftPreList, leftPostList, currentPos.leftChild$)
    AddNodeToCurrPos($rightPreList, rightPostList, currentPos.rightChild$)
```

Algorithm 2 Reconstruction from pre-order and in-order list

(2)
```
procedure AddNodeToCurrPos(PreList, InList, currentPos)
    if $PreList = NULL$ then
        $currentPos.value \leftarrow NULL$
        return
    $currentPos.value \leftarrow PreList[0]$
    $InListCut \leftarrow InList.getPos(PreList[0])$
    $leftInList \leftarrow InList[0 : InListCut]$
    $rightInList \leftarrow InList[InListCut + 1 : Length(InList)]$
    $PreListCut \leftarrow 1$
    while $PreListCut < Length(PreList)$ do
        if $PreList[PreListCut] isnotinleftInList$ then
            break
        $PreListCut \leftarrow PreListCut + 1$
    $leftPreList \leftarrow PreList[1 : PreListCut]$
    $rightPreList \leftarrow PreList[PreListCut : Length(PreList)]$
    AddNodeToCurrPos($leftPreList, leftInList, currentPos.leftChild$)
    AddNodeToCurrPos($rightPreList, rightInList, currentPos.rightChild$)
```

Algorithm 3 Reconstruction from post-order and in-order list

(3)

```
procedure AddNodeToCurrPos(PostList, InList, currentPos)
    if $PostList = NULL$ then
        $currentPos.value \leftarrow NULL$
        return
    $currentPos.value \leftarrow PostList[-1]$
    $InListCut \leftarrow InList.getPos(PostList[-1])$
    $leftInList \leftarrow InList[0 : InListCut]$
    $rightInList \leftarrow InList[InListCut + 1 : Length(InList)]$
    $PostListCut \leftarrow 0$
    while $PostListCut < Length(PostList)$ do
        if $PostList[PostListCut] isnotinleftInList$ then
            break
        $PostListCut \leftarrow PostListCut + 1$
    $leftPostList \leftarrow PostList[0 : PostListCut]$
    $rightPostList \leftarrow PostList[PostListCut : Length(PostList) - 1]$
    AddNodeToCurrPos($leftPostList, leftInList, currentPos.leftChild$)
    AddNodeToCurrPos($rightPostList, rightInList, currentPos.rightChild$)
```

Problem 2 (22%)

(1) Seperate the numbers according to whether the number of ones of the number in binary representation is odd. For example, $(10)_2, (1110)_2, (111011)_2$ will be in one group and $(110)_2, (0)_2, (1110111)_2$ will be in another.

(2) Please read the textbook as reference.

(3) If one of the two groups defined above is an empty set, then the answer will be the number of elements.

Otherwise, we will have to solve the problem using flow. Consider each number as a node and connect them with an edge if there is exactly one different bit between the two numbers in binary representation. Create a source node and connect all number who has odd number of ones in binary representation. Create a sink node and connect all number who has even number of ones in binary representation. Set all the capacity of the edge to 1. Use Edmonds-Karp to calculate the maximum flow *f*. The answer will be N - *f*.

Problem 3 (30%)

(1) (1%)

If G contains a negative cycle c, then its mean weight $\mu(c) < 0$, contradicts with the definition of G to be *con-word*.

(2) (4%)

(a) (2%)

This comes directly from correctness of Bellman-Ford algorithm.

Because every vertex can be reached from s, at least one path from s to v exists.

Let p be the shortest paths from s to v. If many, pick any with smallest number of edges. If p has at least N edges, then it passes at least $N + 1$ vertices. There're only N vertices in G, so p must pass some vertices at least twice, which means there's a cycle in p. Removing this cycle leads to shorter path, or a shortest path with even smaller number of edges, both caused contradiction. So there must exist a shortest path from s to v using at most $N - 1$ edges.

(b) (2%)

First observe that

$$d_N(v) \geq d(v) = \min_{0 \leq k \leq N-1} d_k(v)$$

so

$$\max_{0 \leq k \leq N-1} d_N(v) - d_k(v) \geq 0$$

This means exist $0 \leq k^* \leq N - 1$ such that

$$d_N(v) - d_{k^*}(v) \geq 0$$

and because $N - k^* > 0$,

$$\frac{d_N(v) - d_{k^*}(v)}{N - k^*} \geq 0$$

hence

$$\max_{0 \leq k \leq N-1} \frac{d_N(v) - d_k(v)}{N - k} \geq 0$$

Notice that it also holds if there doesn't exist any path with exactly N edges and $d_N(v) = \infty$. The $\infty - \infty = 0$ case isn't important, because if $d_k(v) = \infty$, it will never be taken by the max function.

(3) (8%)

(a) (2%)

Because we can go from u to v through e, so

$$d(v) \leq d(u) + w(e)$$

Consider the path p from v to u using the remaining edges on cycle c, because c has total weight 0, so p has total weight $0 - w(e) = -w(e)$. So we have

$$d(u) \leq d(v) - w(e)$$

or just

$$d(v) \geq d(u) + w(e)$$

Combining inequalities with two directions, we get

$$d(v) = d(u) + w(e)$$

(b) (5%)

Let v_0 be an arbitrary vertex on c. Because

$$d(v_0) = \min 0 \leq k \leq N - 1 d_k(v_0)$$

so there exists $0 \leq k \leq N - 1$ that

$$d(v_0) = d_k(v_0)$$

Let v_1 be the next vertex of v_0, that is, we can go from v_0 to v_1 by some edge e_1 in c.
By (3)(a),

$$d(v_1) = d(v_0) + w(e_1) = d_k(v_0) + w(e_1)$$

So we can go from s to v_0 using any shortest path with exactly k edges, then go to v_1 using e_1, resulting in a path from s to v_1 with exactly $k + 1$ edges, which means

$$d_{k+1}(v_1) = d(v_1)$$

Similarly, we can find $v_2, v_3, \cdots$, and by induction, we can finally obtain

$$d_{k+i}(v_i) = d(v_i)$$

and

$$d_N(v_{N-k}) = d(v_{N-k})$$

Clearly v_{N-k} is also on c, so we're done.

(c) (1%)

By (2)(b), for all $v \in V$,

$$\max_{0 \leq k \leq N-1} \frac{d_N(v) - d_k(v)}{N - k} \geq 0$$

And by (3)(b), there exists v that

$$\max_{0 \leq k \leq N-1} \frac{d_N(v) - d_k(v)}{N - k} = 0$$

So

$$\min_{v \in V} \max_{0 \leq k \leq N-1} \frac{d_N(v) - d_k(v)}{N - k} = 0$$

(4) (7%)

(a) (3%)

Let p contains N edges $e_1, e_2, e_3, \cdot, e_N$. Let $e_i = (v_{i-1}, v_i)$, then p goes from $v_0, v_1, v_2, \cdots, v_N$ (surely $v_0 = s, v_N = v$).
Because $v_0, v_1, \cdots, v_N$ has $N + 1$ numbers, and there are only N vertices, at least two of them must repeat.
Assume $v_i = v_j, 0 \leq i < j \leq N$, then $e_{i+1}, \cdots, e_j$ is a cycle with length $l = j - i$, contained in p. Its total weight is $W = \sum_{k=i+1}^{j} w(e_k)$.
Removing this cycle we get another valid path $p' = e_1, \cdots, e_i, e_j + 1, \cdots, e_N$, with $N - l$ edges and total weight $w(p') = w(p) - W = d_N(v) - W$.
So

$$d_{N-l}(v) \leq w(p') = d_N(v) - W$$

$$d_N(v) - d_{N-l}(v) \geq W$$

(b) (4%)

Let p be any shortest path with N edges.

If p contains any positive-weight cycle $e_{i+1}, \cdots, e_j$ with total weight $W > 0$, then by (4)(a),

$$d_N(v) - d_{N-l}(v) \geq W > 0$$

So

$$\max_{0 \leq k \leq N-1} \frac{d_N(v) - d_k(v)}{N-k} \geq \frac{d_N(v) - d_{N-l}(v)}{N-(N-l)} \geq \frac{W}{l} > 0$$

which contradicts to the problem assumption that such value equals to 0.

So p can't contain any *positive-weight* cycle. But by (4)(a), p must contain some cycle. Therefore, p contains at least one *con-weight* cycle.

(5) (10%)

(a) (3%)

If G doesn't contain any cycle, $d_N(v) = \infty$ for all $v \in V$, and the formula correctly computes $\mu^*(G) = \infty$.

If G contains at least one cycle, $\mu^*(G) = m$ is a finite value (It can't be $-\infty$ because the mean value can't be smaller than the weight of smallest edge).

Let $G' = (V, E')$ be the graph obtained from G by subtracting m from weights of all edges. That is, for any $e \in E$, there exists a corresponding edge $e' \in E'$ with $w(e') = w(e) - m$.

For **any** cycle c in G and corresponding cycle c' in G', new mean weight is subtracted by m:

$$\mu(c') = \frac{1}{|c'|} \sum_{e' \in c'} w(e') = \frac{1}{|c|} \left(\sum_{e \in c} w(e) - m \right) = \left(\frac{1}{|c|} \sum_{e \in c} w(e) \right) - m = \mu(c) - m$$

The new graph has minimum mean cycle weight

$$\mu^*(G') = \mu^*(G) - m = 0$$

so G' is a *con-weight* graph!

Denote $d'(v)$ and $d'_k(v)$ be the shortest path lengths in G'. We can now use the result of (3)(c):

$$\min_{v \in V} \max_{0 \leq k \leq N-1} \frac{d'_N(v) - d'_k(v)}{N-k} = 0$$

But similar to cycles, for **any** path p from s to v with k edges, and corresponding p', we have

$$w(p') = \sum_{e' \in p'} w(e') = \sum_{e \in p} (w(e) - m) = w(p) - km$$

So for the paths with exactly k edges, their total weight are all subtracted the same amount km. Hence

$$d'_k(v) = d_k(v) - km$$

Also, if $d_k(v) = \infty$, it means there's no such path, so $d'_k(v) = \infty$. Formula above still holds if we define $\infty - km = \infty$.

For any $v \in V, 0 \leq k \leq N-1$,

$$\frac{d'_N(v) - d'_k(v)}{N-k} = \frac{d_N(v) - d_k(v) - (N-k)m}{N-k} = \frac{d_N(v) - d_k(v)}{N-k} - m$$

so for original graph G,

$$\min_{v \in V} \max_{0 \leq k \leq N-1} \frac{d_N(v) - d_k(v)}{N-k} = \min_{v \in V} \max_{0 \leq k \leq N-1} \frac{d'_N(v) - d'_k(v)}{N-k} + m = m = \mu^*(G)$$

(b) (7%)

It's very similar to Bellmen-Ford algorithm, but this time we need to store $d_k(v)$ separately for every k.

Initially,

$$d_0(v) = \begin{cases} 0 & , v = s \\ \infty & , \text{otherwise} \end{cases}$$

This step takes $O(N)$.

For $1 \le k \le N$, take minimum of all incident edge,

$$d_k(v) = \min_{e=(u,v)\in E} d_{k-1}(u) + w(e)$$

Take $d_k(v) = \infty$ if there's no incident edges.

If $d_k(v) < \infty$, also store which edge give the shortest length,

$$prev_k(v) = \operatorname*{arg\,min}_{e=(u,v)\in E} d_{k-1}(u) + w(e)$$

Calculate the values from $k = 1$ upto $k = N$. For each k, each edge will be considered exactly once, so complexity of this step is $O(N^2 + NM)$.

Next, calculate the minimum mean cycle weight:

$$\mu^*(G) = \min_{v\in V} \max_{0\le k\le N-1} \frac{d_N(v) - d_k(v)}{N-k}$$

This step is straightforward $O(N^2)$.

If $\mu^*(G) < \infty$, we need to actually find a cycle with minimum mean weight. First we find a vertex which takes minimum of the above formula:

$$v^* = \operatorname*{arg\,min}_{v\in V} \max_{0\le k\le N-1} \frac{d_N(v) - d_k(v)}{N-k}$$

Then find an arbitrary shortest path from s to v^* with exactly N edges. This can be done by backtracing from v^* through *prev* link until reach s. Call this path $p = v_0, v_1, \cdots, v_N$ (surely $v_0 = s, v_N = v^*$).

In fact, v^* corresponds to a vertex satisfying the formula in (4)(b) in G', so p contains at least one cycle with minimum mean weight (*con-weight* cycles in G' correspond to *minimum mean weight* cycles in G), but no cycle with strictly greater mean weight.

So what we need to do is find an arbitrary cycle in p, then we're done. And this is achieved by finding two different index i, j such that $v_i = v_j$ in p, and take the edges between them.

To obtain a *simple* cycle, we can modify the backtracing method above (to find p) to stop immediately once we reach a repeated vertex u, and the cycle is just the edges between the last appearance of u and this u (See codes below).

The procedure of finding the cycle runs in $O(N)$.

The overall time complexity is $O(N^2 + NM)$.

Below is the sample code. Here we used a trick: initialize $d_0(v) = 0$ for all $v \in V$, as there's another vertex s that connects to every v for cost 0. For simplicity, we used `double` for storing mean value. To avoid floating point errors, you can write a fraction class instead.

```cpp
typedef pair<int, int> pii;
const int INF = 1000000000;
int N, M;
vector<pii> edges[N]; // edge[v] contains pairs (u, c) :
   edge (u -> v) cost c
int dis[N+1][N], prv[N+1][N], visit[N];

double min_mean_cycle(vector<int> &ans)
{
  ans.clear();
  for(int i=0; i<=N; i++)
    for(int j=0; j<N; j++)
      dis[i][j] = INF;
  for(int i=0; i<N; i++)
    dis[0][i] = 0;

  for(int k=1; k<=N; k++)
  {
    for(int v=0; v<N; v++)
    {
      for(size_t i=0; i<edges[v].size(); i++)
      {
        int u = edges[v][i].first, c = edges[v][i].second;
        int newdis = dis[k-1][u] + c;
        if(newdis < dis[k][v])
        {
          dis[k][v] = newdis;
          prv[k][v] = u;
        }
      }
    }
  }

  int vs = -1;
  double mu = INF;
  for(int v=0; v<N; v++)
  {
    if(dis[N][v] == INF) continue;
    double cur_max = -INF;
    for(int k=0; k<=N-1; k++)
    {
      cur_max = max(cur_max, (double)(dis[N][v] - dis[k][v])
          / (N-k));
    }
    if(cur_max < mu)
```

```
      {
        mu = cur_max;
        vs = v;
      }
    }

    if(vs != -1)
    {
      for(int i=0; i<N; i++)
        visit[i] = false;
      vector<int> vlist;
      int v = vs, u = -1, k = N;
      while(1)
      {
        if(visit[v])
        {
          u = v;
          break;
        }
        vlist.push_back(v);
        visit[v] = true;
        v = prv[k][v];
      }

      ans.push_back(u);
      while(1)
      {
        v = vlist.back();
        ans.push_back(v);
        if(v == u) break;
        vlist.pop_back();
      }
    }

    return mu;
  }
```

Problem 4 - MUST (Programming, 30+6%)

See the sample solution attached on the website.