NORTHWESTERN UNIVERSITY

EECS 336 – Homework 7

Weihan Chu

May 24, 2016

1 QUESTION 1

1.1 PART A

case1: if $a_{i-1} > 1/2$ and $a_i >= 1/2$,we can get that:

$$\begin{aligned}\hat{c_i} &= c_i + \phi_i - \phi_{i-1}\\ &= 1 + (2num_i - size_i) - (2num_{i-1} - size_{i-1})\\ &= 1 + (2(num_{i-1} - 1) - size_{i-1}) - (2num_{i-1} - size_{i-1})\\ &= -1\end{aligned}$$

case2: if $a_{i-1} = 1/2$ and $a_i < 1/2$,we can get that:

$$\begin{aligned}\hat{c_i} &= c_i + \phi_i - \phi_{i-1}\\ &= 1 + (size_i/2 - num_i) - (2num_{i-1} - size_{i-1})\\ &= 1 + (size_{i-1}/2 - (num_{i-1} - 1)) - (2num_{i-1} - size_{i-1})\\ &= 2 + 3/2size_{i-1} - 3num_{i-1}\\ &= 2\end{aligned}$$

So the TABLE-DELETE's amortized cost is bounded above by a constant.

1.2 PART B

We can get the new potential function is:

$$\phi(i) = \begin{cases} 8 \times num_i - 4size_i & if\ \alpha(i) \geq 1/2 \\ 5size_i - 10num_i, & if\ \alpha(i) < 1/2 \end{cases}$$

case1: if $a_{i-1} > 1/2$ and $a_i >= 1/2$,we can get that:

$$\begin{aligned} \hat{c_i} &= c_i + \phi_i - \phi_{i-1} \\ &= 1 + (8num_i - 4size_i) - (8num_{i-1} - 4size_{i-1}) \\ &= 1 + (8(num_{i-1} - 1) - 4size_{i-1}) - (8num_{i-1} - 4size_{i-1}) \\ &= -7 \end{aligned}$$

case2: if $a_{i-1} = 1/2$ and $a_i < 1/2$,we can get that:

$$\begin{aligned} \hat{c_i} &= c_i + \phi_i - \phi_{i-1} \\ &= 1 + (5size_i - 10num_i) - (8num_{i-1} - 4size_{i-1}) \\ &= 1 + (5size_{i-1} - 10(num_{i-1} - 1)) - (8num_{i-1} - 4size_{i-1}) \\ &= 11 + 9size_{i-1} - 18num_{i-1} \\ &= 11 \end{aligned}$$

So the TABLE-DELETE's amortized cost is bounded above by a constant.

2 QUESTION 2

2.1 PART A

case1: if $a_{i-1} > 1/2$ and $a_i >= 1/2$,we can get that:

$$\begin{aligned} \hat{c_i} &= c_i + \phi_i - \phi_{i-1} \\ &= 1 + (2num_i - size_i) - (2num_{i-1} - size_{i-1}) \\ &= 1 + (2(num_{i-1} - 1) - size_{i-1}) - (2num_{i-1} - size_{i-1}) \\ &= -1 \end{aligned}$$

case2: if $a_{i-1} = 1/2$ and $a_i < 1/2$,we can get that:

$$\begin{aligned}\hat{c_i} &= c_i + \phi_i - \phi_{i-1}\\ &= 1 + (size_i - 2num_i) - (2num_{i-1} - size_{i-1})\\ &= 1 + (size_{i-1} - 2(num_{i-1} - 1)) - (2num_{i-1} - size_{i-1})\\ &= 3 + 2size_{i-1} - 4num_{i-1}\\ &= 3\end{aligned}$$

case3: if $1/3 < a_{i-1} <= 1/2$ and $1/3 <= a_i <= 1/2$,we can get that:

$$\begin{aligned}\hat{c_i} &= c_i + \phi_i - \phi_{i-1}\\ &= 1 + (size_i - 2num_i) - (size_{i-1} - 2num_{i-1})\\ &= 1 + (size_i - 2(num_i - 1)) - (size_{i-1} - 2num_{i-1})\\ &= 3\end{aligned}$$

case4: if $a_{i-1} = 1/3$,we can get that:

$$\begin{aligned}\hat{c_i} &= c_i + \phi_i - \phi_{i-1}\\ &= num_i + 1 + (size_i - 2num_i) - (size_{i-1} - 2num_{i-1})\\ &= num_{i-1} - 1 + 1 + (2/3size_{i-1} - 2(num_{i-1} - 1)) - (size_{i-1} - 2num_{i-1})\\ &= 2 - 1/3size_{i-1} + num_{i-1}\\ &= 2\end{aligned}$$

So the TABLE-DELETE's amortized cost is bounded above by a constant.

2.2 PART B

The potential function I use is $|1/2T.size - 2T.num|$
case1: if $a_{i-1} > 1/4$ and $a_i >= 1/4$,we can get that:

$$\begin{aligned}\hat{c_i} &= c_i + \phi_i - \phi_{i-1}\\ &= 1 + (2num_i - 1/2size_i) - (2num_{i-1} - 1/2size_{i-1})\\ &= 1 + (2(num_{i-1} - 1) - 1/2size_{i-1}) - (2num_{i-1} - 1/2size_{i-1})\\ &= -1\end{aligned}$$

case2: if $a_{i-1} = 1/4$ and $a_i < 1/4$,we can get that:

$$\begin{aligned}\hat{c_i} &= c_i + \phi_i - \phi_{i-1}\\ &= 1 + (2num_i - 1/2size_i) - (1/2size_{i-1} - 2num_{i-1})\\ &= 1 + (2(num_{i-1} - 1) - 1/2size_{i-1}) - (1/2size_{i-1} - 2num_{i-1})\\ &= 4num_{i-1} - size_{i-1} - 1\\ &= -1\end{aligned}$$

case3: if $1/6 < a_{i-1} <= 1/4$ and $1/6 <= a_i <= 1/4$,we can get that:

$$\begin{aligned}\hat{c_i} &= c_i + \phi_i - \phi_{i-1}\\ &= 1 + (1/2size_i - 2num_i) - (1/2size_{i-1} - 2num_{i-1})\\ &= 1 + (1/2size_i - 2(num_i - 1)) - (1/2size_{i-1} - 2num_{i-1})\\ &= 3\end{aligned}$$

case4: if $a_{i-1} = 1/6$,we can get that:

$$\begin{aligned}\hat{c_i} &= c_i + \phi_i - \phi_{i-1}\\ &= num_i + 1 + (1/2size_i - 2num_i) - (1/2size_{i-1} - 2num_{i-1})\\ &= num_{i-1} - 1 + 1 + (1/2 * 2/3size_{i-1} - 2(num_{i-1} - 1)) - (1/2size_{i-1} - 2num_{i-1})\\ &= 2 - 1/6size_{i-1} + num_{i-1}\\ &= 2\end{aligned}$$

So the TABLE-DELETE's amortized cost is bounded above by a constant.

3 QUESTION 3

We need to prove this problem in two aspects.
First,we assume that there are a solution A_1 for LONGEST-PATH-LENGTH problem and this solution is in polynomial time. And there is also a algorithm A_2 for deciding LONGEST-PATH. So we can get that:

```
Input:<G,u,v,k>
Output:<True or False>

Algorithm A2(G,u,v,k){
    Max= A1(G,u,v);
    if(k <= Max)
        return True;
    else
        return False;
}
```

So if we have the solution of A_1, we can get the solution of A_2 in constant time. Since A_1 runs in polynomial time, so A_2 also runs in polynomial time.

Second,we assume that there are a solution B_1 for LONGEST-PATH problem and this solution is in polynomial time. And there is also a algorithm B_2 for LONGEST-PATH-LENGTH problem. So we can get that:

```
Input:<G,u,v>
Output:The size of the longest path between u and v

Algorithm B2(G,u,v,k){
    k=n;
    while((k>=0) and B1(G,u,v,k)==False) do
        k--;
    return k;
}
```

So if we have the solution of B_1, we can get the solution of B_2 in n time.Since B_1 runs in polynomial time, So B_2 also runs in polynomial time
Therefore, we can prove that the optimization problem LONGEST-PATH-LENGTH can be solved in polynomial time IF AND ONLY IF LONGEST-PATH in P.

4 Question 4

First,we give a formal encoding of directed graphs as binary strings using an adjacency matrix representation:
Directed graph $G(V,E)$ is the graph that in which all edges are represented by an arrow from u to v which means the edge(u,v).
The adjacency-matrix $a(U,V)$ is a matrix using 1 is there is a arrow from one vertex to another, or using 0 is there is not a arrow from one vertex to another.
As the pic1 I attached in the jpg file, which is an example of a directed graph. From the graph, we can get the adjacency matrix representation by binary strings as follows:

$$\begin{array}{ccccc} 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{array}$$

This matrix representation of graph can be encoded as :

0101010110000010100100010

This representation uses V^2 bits, which is in polynomial form. And we know the matrix is square, we can create it by counting the total bits of input and taking the square root of the input and then writing the bits in the same order.

Then we give a formal encoding of directed graph as binary string using adjacency-list representation.
We just use the n_{th} line being the vertex number and followed by numbers of all the numbers of all vertices connected to this vertex.

```
1 2 4
2 1 3 4
3 5
4 2 5
5 4
```

To represent the adjacency list, the memory will be $O(|V|+|E|)$. Since each vertex number will be between 1 and $|V|$, so the total number of bits needed will be $O((|V|+|E|)lg|V|)$,which is also in the polynomial form.
So now we get that the adjacency matrix representation and adjacency list representation both takes the polynomial type of time for representation. We will focus on to prove they are polynomially related.
First, if we want to covert adjacency matrix representation to adjacency list representation:

```
For each row do
    Scan the row from left to right
    Add the column number to the list
```

So this is polynomially related.
Then, if we want to covert adjacency list representation to adjacency matrix representation:

```
Start with a matrix M of all 0.
Let i be a counter of the line you are at. In that line:
    if hit the number j
        then put a 1 in M[i,j]
```

So this is polynomially related.
So the two representations are polynomially related.

5 Question 5

To solve this problem by dynamic programming, we define $c[i,w]$ to be the solution for items $1,2,...,i$ and maximum weight , then we can get that:

case1: $c[i,w] = 0$, if $i = 0$.
case2: $c[i,w] = c[i-1,w]$, if $w_i > w$, which means the new item is more than the current weight limit.
case3: $c[i,w] = max(v_i + c[i-1, w-w_i], c[i-1,w])$, if $i > 0$ and $w \geq w_i$.

```
for j from 0 to W do:
     m[0, j] = 0

for i from 1 to n do:
     for j from 0 to W do:
        if w[i-1] > j then:
             m[i, j] = m[i-1, j]
        else:
             m[i, j] = max(m[i-1, j], m[i-1, j-w[i-1]] + v[i-1])
```

This solution will therefore run in $O(nW)$ time, where n is the number of items and W is the maximum weight that the knapsack can hold. This is not a polynomial running time for any reasonable representation of the input.
Because a computer uses binary, so we can get that:

$$\begin{aligned} T(n) &= O(nW) \\ &= (n2^{lengthofW}) \end{aligned}$$

So if we add one bit in the W, the run time will be doubled. So this problem is not a polynomial time problem.