HW 6 Yuqi Jin

(1)

(a.) False, Fractional knapsack could have the same maximum value as the integer knapsack problem. When the optimal solution for the fractional knapsack problem, which allows taking fractions of items. coincides with the solution obtained from the integer knapsack problem, where only whole items can be selected, with high value of tasks, and fractional knapsack exhaust the sun time.

(b) True. KMP algorithm efficiently in finding occurances of a specific pattern.

(c) True, by using the 5 colums trick. and quick sort pivot by choosing the exact medium point.

(d) True, by the definition of BST, right children always bigger than root and left children. Use recursive search to the right can find largest element.

(e) True, the minimum spanning tree must include the smallest-weight edge incident to each node to ensure that meets the criteria of minimum spanning tree. If edge is the lightest edge connected to node i, it is important to check minimum spanning tree, any other edge create a cycle and increase total weight of tree.

(f) False, for integer knapsack problem, items can not be divided and whole items can be selected. The greedy algorithm often doesn't guarantee an optimal solution. Dynamic Programing is commonly used for integer knapsack problem. and solved in pseudo-polynomical, if input size is doubled, the running time will not always scale linearly.

(g) False, Both $d(i,k)$ and $d(k,j)$ could be minimum distances. or part of different shortest paths in the graph. leading to $d(i,j)$ being the sum of shortest paths.

(h) False. it can not find in $O(M+N)$, According to the $\log^*(N)$ inverse Ackermann function. $i = \log^*(N)$ = min number times you take $\log_2$ to get equal or smaller than below 1.

(i) False, It doesn't aim to remove negative cycles. The potential function is used to reweie the edges. To make the graph suitable for subsequent use of Dijkstra's algorithm

(j) False, the maximum Procrastination method is not related to the amortized analysis. This method is to assign tasks to time slots based on their deadlines

2.

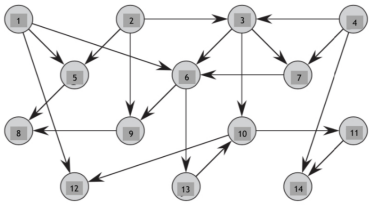


Figure 1:

2. (15 pts) Do a topological sort on this directed graph. (YES this is exactly the same graph done in class. Still a good exercise.)

   (a) Starting from node 1 do a topological sort using a stack to enumerate the nodes. Alway prefer the smaller node number when pushing into the stack. When the stack is empty start with the lowest node number not yet visited. You should show enough of the individual stack operatinos to trace the enumeration (crossing out element is ok for pop operations).

   (b) Show on the graph that order of execution that has no deadlocks. What is the graphs property that guarantees this lack of deadlocks?

   (c) Give the number and size of the directed connected components in this directed graph. How many connected components in the graph if all arcs are allowed to be undirected?

   (d) If we did the topological sort after permuting the labels and still using the low label first rule for the new lables, would the result be the same order of execution and the same number of directed connected components?

Visited List

1	2	3	4	5	6	7	8	9	10	11	12	13	14
T	T	T	T	T	T	T	T	T	T	T	T	T	T

(a)

1	{5, 6, 12}
2	{3, 5, 8}
3	{6, 7, 10}
4	{3, 7, 14}
5	{8}
6	{9, 13}
7	{6}
8	{ }
9	{8}
10	{11, 12}
11	{16}
12	{ }
13	{10}
14	{ }

Stack

4
2
3
7
1
6
13
10
12
11
14
9
5
8

pop all the eles from stack, and print it in that order

4, 2, 3, 7, 1, 6, 13, 10, 12, 11, 14, 9, 5, 8

Start from node 1, Topological sort (1), visited[1] = True

↓

Topological sort (5) visited(5) = True

↓

Topological sort (8) visited(8) = True

↓

List is empty, no more recursion call

push (8)

↓

5 → 8, 8 is visited, no more recursion call

push (5)

↓

Turns to 1, smaller node is 6

Topological sort (6), visited(6) = True

↓

Topological sort (9), visited (9) = True

↓

8 is already visited, No more recursion call

push (9)

↓

Topological Sort (13), visited(13) = True

↓

Topological sort (10), visited (10) = True

↓

Topological Sort (11), visited (11) = True

↓

Topological sort (16), visited (16) = True

↓

List is empty, no more recursion call.

push (14)

11 has visited 14, no more curison call.

push (11)

Turns to 10, smaller node is 12

Topological sort (12) visited (12) = True

↓

List is empty, no more recursion call

push (12)

push (10)

10 has been visited, no more recursal call

push (13)

9, 13 have been visited, no more recursal call

push (6)

12, 6, 5 all visited, list is empty, no more recursion call.

push (1)

4 is the smaller node now.

Topological Sort (4), visited (4) = True

↓

7, 7, 10 are visited. no more recursal call

push (4)

2 is the smaller node now

Topological sort (2). visited (2) = True

↓

Topological sort (3), visited (3) = True.

↓

Topological sort (7), visited (7) = True.

↓

6 is visited, no more recursion call

push (7)

↓

6, 10 are visited, no more recursal call

↓

push (3)

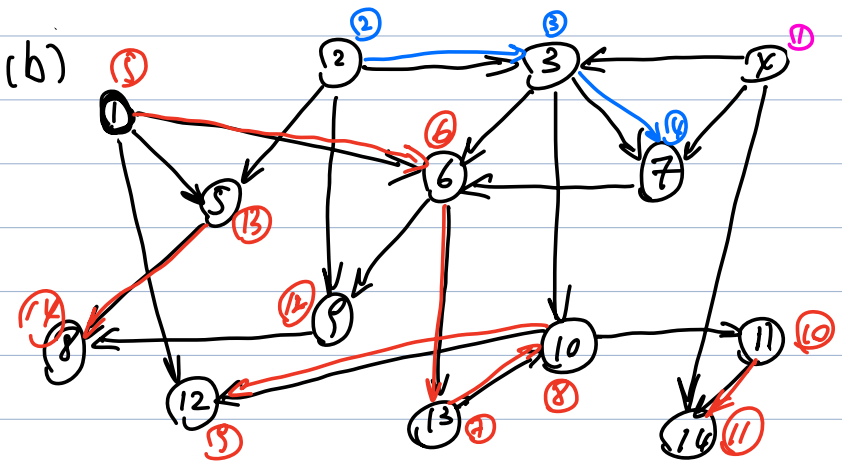
↓

5, 9 are visited, no more recursal call

↓

push (2)

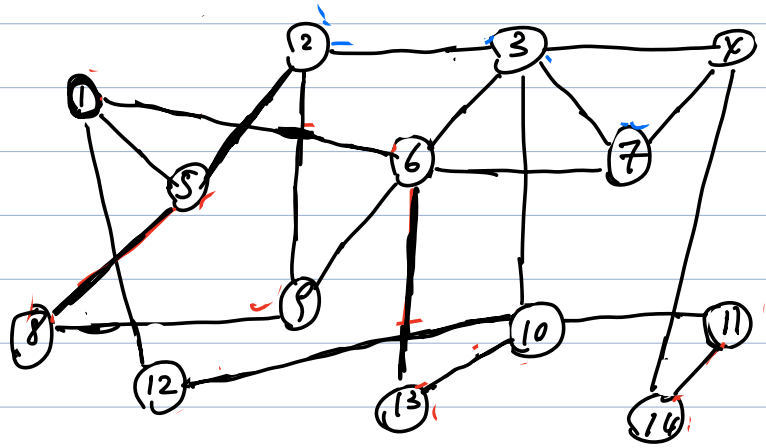
(b)



After drawing the order of execution on Graph, there are no deadlocks. The properties for Graphs guarantee Lack of dead lock, is circular wait. for example, node 1 waiting for node 2, 2 waiting 3, and N waiting for 1, It is likely a circular Graph.

(c) Number of Directed Connected Components: 3
Sizes of Connected Components:
Component: [1, 5, 8, 6, 9, 13, 10, 11, 14, 12], Size = 10
Component: [2, 3, 7], size = 3
Component: [4], size = 1

if all arcs allowed to be undirected.
The number is 1.



[1, 5, 2, 3, 4, 7, 6, 9, 8, 13, 10, 12, 11, 14], size 14

(d) After permuting the labels, and follow low label first Rule.
Order of Execution, might lead to a different order of execution in the topological sort. Nodes with different labels would be considered into different sequences. potentially resulting in a rearranged order of tasks or Process in the execution sequence.

Number of Directed Connected Components: might change, new node labels could potentially change the graph's connectivity.

3. 3. (10 pts) Consider Floyd-Warshall algorithm for a directed graph with 4 nodes. The first value for $D^{(0)}(i,j) = W(i,j)$ is the following:

```
D(0)(i,j)    j=1   j=2   j=3   j=4   j=5
    i=1        0     5     9     inf   3
    i=2       inf    0     8     inf   inf
    i=3       inf    1     0     7     4
    i=4       inf   inf   inf    0     inf
    i=5       11    inf   inf    6     0
```

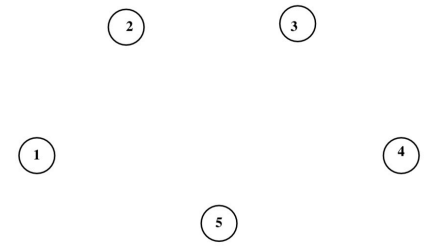


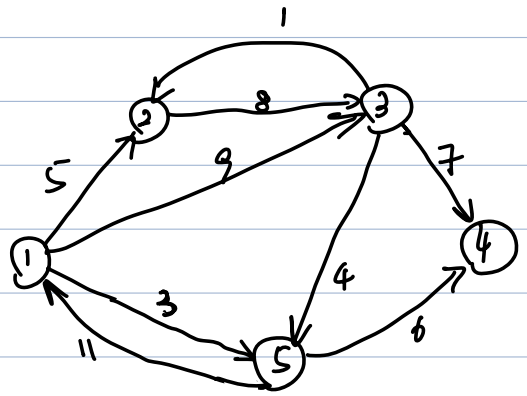
Figure 2:

(a) Draw the directed graphs on Fig. 3 with arcs labeled by their weights. Give the iterations k = 1, 2, ... for the Floyd-Warshall algorithm. (Recall that the k = 1 iteration considers paths from i to j thru node 1 and k = 2 paths thru node 2, etc.) Show your work as a sequence of two 5 by 5 tables.

(b) Solve the all pairs shortest path by two iterations of "repeated squaring", i.e. for $i, j, k = 1, 2, 3, 4, 5$

$$D^{(2)}(i,j) = MIN[\, D^{(0)}(i,k) + D^{(0)}(k,j)\,],$$
$$D^{(4)}(i,j) = MIN[\, D^{(2)}(i,k) + D^{(2)}(k,j)\,].$$

(c) Show your work as a sequence of two 5 by 5 tables.



(a)

D(0)(i,j)	j=1	j=2	j=3	j=4	j=5
i=1	0	5	9	inf	3
i=2	inf	0	8	inf	inf
i=3	inf	1	0	7	4
i=4	inf	inf	inf	0	inf
i=5	11	inf	inf	6	0

k=1

	j=1	j=2	j=3	j=4	j=5
i=1	0	5	9	inf	3
i=2	inf	0	8	inf	inf
i=3	inf	1	0	7	4
i=4	inf	inf	inf	0	inf
i=5	11	16	20	6	0

k=2

	j=1	j=2	j=3	j=4	j=5
i=1	0	5	9	inf	3
i=2	inf	0	8	inf	inf
i=3	inf	1	0	7	4
i=4	inf	inf	inf	0	inf
i=5	11	16	20	6	0

k=3

	j=1	j=2	j=3	j=6	j=5
i=1	0	5	9	16	3
i=2	inf	0	8	15	12
i=3	inf	1	0	7	4
i=6	inf	inf	inf	0	inf
i=5	11	16	20	6	0

k=6

	j=1	j=2	j=3	j=6	j=5
i=1	0	5	9	16	3
i=2	inf	0	8	15	12
i=3	inf	1	0	7	K
i=6	inf	inf	inf	0	inf
i=5	11	16	20	6	0

k=5

	j=1	j=2	j=3	j=6	j=5
i=1	0	5	9	9	3
i=2	23	0	8	15	12
i=3	15	1	0	7	4
i=6	inf	inf	inf	0	inf
i=5	11	16	20	6	0

final output

	j=1	j=2	j=3	j=6	j=5
i=1	0	5	9	9	3
i=2	23	0	8	15	12
i=3	15	1	0	7	4
i=6	inf	inf	inf	0	inf
i=5	11	16	20	6	0

(b) $D^{(2)}(i,j) = MIN[D^{(0)}(i,k) + D^{(0)}(k,j)]$

$D^{(4)}(i,j) = MIN[D^{(2)}(i,k) + D^{(2)}(k,j)]$

$i,j,k = 1,2,3,4,5$

D[2](1,1)=0 D[2](1,2)=5 D[2](1,3)=9 D[2](1,4)=INF D[2](1,5)=3

D[2](2,1)=INF D[2](2,2)=0 D[2](2,3)

D[2] ·— - -

D[2] · - — - -

D[2] · — — -

plug in all numbers, compare Minimum

D[4](1,1) - - - -

D[4](1,2) - — — ·

D[4](1,3) — — —

D[4](1,4) ·— — —

D[4](1,5) · — —

(c)

D(2)	j=1	j=2	j=3	j=4	j=5
i=1	0	5	9	INF	3
i=2	INF	0	8	INF	INF
i=3	INF	1	0	7	4
i=4	INF	INF	INF	0	INF
i=5	11	16	20	6	0

D(4)	j=1	j=2	j=3	j=4	j=5
i=1	0	5	9	16	3
i=2	INF	0	8	15	12
i=3	INF	1	0	7	4
i=4	INF	INF	INF	0	INF
i=5	11	16	20	6	0

4.

4. (15 pts) This is a problem in the arithmetic of **Binomial queues** – also called a **Binomial Heap** when enforcing min-Heap like order. This is nice collectoi of trees *e.g. forest* that are have either zero nodes (just a null pointer) or 2^k nodes for each tree.

   (a) Draw two data structures for a binomial queues with a number of nodes give by $N = n_0+2n_1+4n_2+8n_3+16n_4$ and binary numbers $\{n_4n_3n_2n_1, n_1\} = \{1010\}$ and $\{n_4n_3n_2n_1, n_1\} = \{1111\}$.

   (b) If we demand that the value at the root each tree or subtree is a minimum for that tree each of the separate queues, describe rule for addition that maintains this property in the sum. For binomial queue with this property and $O(N)$ nodes what is the complexity of bound on finding the minimum value?

   (c) In general adding two binomial queues with $O(N_1)$ and $O(N_2)$ nodes bound on the complexity of the complexity of this add operation?

(a) $N = n_0 + 2n_1 + 4n_2 + 8n_3 + 16n_4$

corresponding to different powers of 2 $\{1, 2, 4, 8, 16\}$

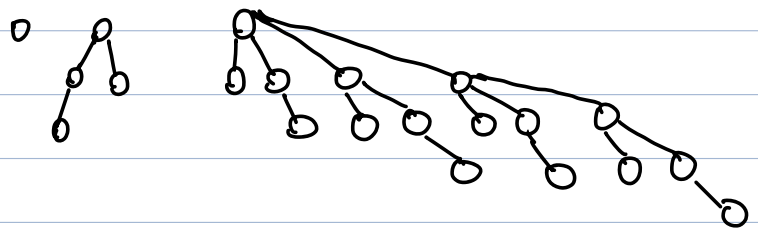
Binary numbers $\{n_4 n_3 n_2 n_1, n_1\} = \{1010\}$

$n_4 = 1$, $n_3 = 0$, $n_2 = 1$, $n_1 = 0$, $n_0 = 1$

First one.

$N = 1 + 0 + 4 \cdot 1 + 0 + 16 \cdot 1 = 1 + 20$

$N = 21$

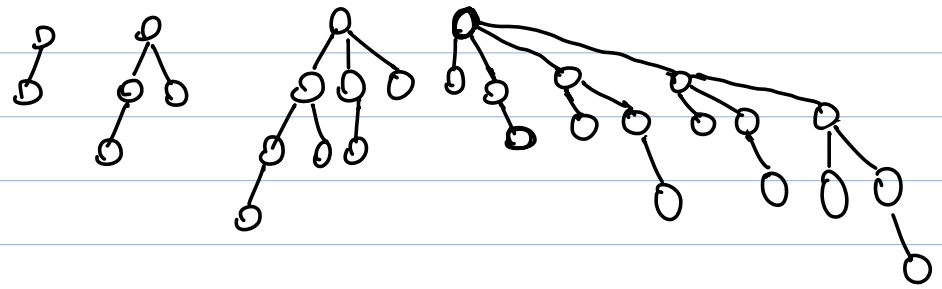


Second one.

$\{n_4 n_3 n_2 n_1, n_1\} = \{1111\}$

$n_4 = 1$ $n_3 = 1$ $n_2 = 1$ $n_1 = 1$, $n_0 = 0$

$N = 0 + 2 + 4 + 8 + 16 = 30$



(b). Rule for addition, when merging two binomial queues, compare and merge trees of the same order. Then maintain Min-Heap Property, to ensure root of the merged tree holds minimum value among roots of trees being merged. Combine trees of equal order, making trees with smaller root value, child of the tree with Larger root value.

There is O(N) nodes, the complexity for finding minimum element is O(logN). Search for the minimum value by comparing the roots of these trees only need to examine the roots of each tree to find overall min value.

(c) In other words, merging two binomial queues with size N_1 and N_2 is bounded by $O(\log(N_1+N_2))$. The actual merge operation only touches the root nodes of a few trees. There're $\log N$ trees in each Binomial Queue., the merge is $O(\log(N))$

5.

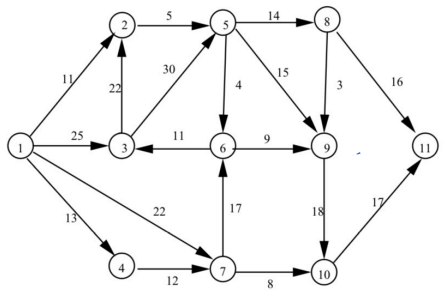


Figure 3:

5. (20 pts) Consider the directed weighted graph in Figure 3. (There are copies of this graph on the next page.[1]

   (a) Use Dijkstra's algorithm to find the shortest paths from node 1 to all of the nodes. (Dijkstra's, like Prim's minimum spanning tree algorithm, adds one node at a time.)
   (i) Make a series of tables showing the values of the distance array $d(i)$ and the predecessor array $\pi(i)$ after each update
   (ii.) Draw the final tree on the figure with the final values of $d(i)$ and $p(i)$ at each node.

   (b) Repeat the exercise above except now use using Belllman-Ford this time (Bellman-Ford, like Kruskal's minimum spanning tree algorithm, adds one arc at a time.)
   (i.) Make a series of tables showing the values of the distance array $d(i)$ and the predecessor array $\pi(i)$ after each update.
   (ii.) Draw the final tree on the figure with the final values of $d(i)$ and $p(i)$ at each node.

   (c) Computed the minimum spanning tree considering all arcs to be bi-directional (in spite of the figure!) using Prim's algorithm starting form node 1, listing in order the total weight and predecessors as they are modified at each step.

   (d) Are the final trees in all these cases the same? Explain

[1] **Note in part a and b below you start with $d(1) = 0$, $d(i > 1) = \infty$ and $\pi(i) = -1$. The table at each step only needs to show the values of $d(i)$ and $\pi(i)$ that change!**

(a)

(i) Use Dijkstra's Algorithm from node 1 to all of the nodes.

d(i) Table

Node	1	2	3	4	5	6	7	8	9	10	11
dis	0	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞

π(i) table

Node	1	2	3	4	5	6	7	8	9	10	11
pre	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1

Start from node 1, calculate dis from node 1 to node 2, 3, 7, 4

d(i) Table

Node	1	2	3	4	5	6	7	8	9	10	11
dis	0	11	25	13	∞	∞	22	∞	∞	∞	∞

π(i) table

Node	1	2	3	4	5	6	7	8	9	10	11
pre	-1	1	1	1	-1	-1	1	-1	-1	-1	-1

The dis from Node 1 to Node 2 is smaller than others, so Mark 1 visited, and Explore Node 2. Node 2 can only go to Node 5, calculate dis $11+5=16$

Mark 2 visited.

d(i) Table

Node	1	2	3	4	5	6	7	8	9	10	11
dis	0	11	25	13	16	∞	22	∞	∞	∞	∞

π(i) table

Node	1	2	3	4	5	6	7	8	9	10	11
pre	-1	1	1	1	2	-1	1	-1	-1	-1	-1

Next the smaller dis is Node 4, update dis $4 \to 7$ is $13+12=25$

Mark 4 visited $25 > 22$ for 7, so Node 7 no need to update

d(i) Table

Node	1	2	3	4	5	6	7	8	9	10	11
dis	0	11	25	13	16	∞	22	∞	∞	∞	∞

π(i) table

Node	1	2	3	4	5	6	7	8	9	10	11
pre	-1	1	1	1	2	-1	1	-1	-1	-1	-1

Start from Node 5, calculate dis from Node $5 \to 6, 8, 9$

d(i) Table

Node	1	2	3	4	5	6	7	8	9	10	11
dis	0	11	25	13	16	20	22	30	31	∞	∞

π(i) table

Node	1	2	3	4	5	6	7	8	9	10	11
pre	-1	1	1	1	2	5	1	5	5	-1	-1

Explore from Node 6, calculate dis from $6 \to 9, 3$, no need update for Node 3 because $25 < 31$

d(i) Table

Node	1	2	3	4	5	6	7	8	9	10	11
dis	0	11	25	13	16	20	22	30	29	∞	∞

π(i) table

Node	1	2	3	4	5	6	7	8	9	10	11
pre	-1	1	1	1	2	5	1	5	6	-1	-1

Explore from node 7, calculate dis from 7 → 6, 10. No need to change for 6 20<38.

d(i) Table

Node	1	2	3	4	5	6	7	8	9	10	11
dis	0	11	25	13	16	20	22	30	29	30	∞

π(i) table

Node	1	2	3	4	5	6	7	8	9	10	11
pre	-1	1	1	1	2	5	1	5	6	7	-1

Explore Node 3, no need to change dis and pre for 2, 5
Explore Node 9, from 9 → 10, no need to update
Explore Node 8, from 8 → 11, dis is 30 + 16 = 46

d(i) Table

Node	1	2	3	4	5	6	7	8	9	10	11
dis	0	11	25	13	16	20	22	30	29	30	46

π(i) table

Node	1	2	3	4	5	6	7	8	9	10	11
pre	-1	1	1	1	2	5	1	5	6	7	8

Explore Node 10 → 11, dis is 30+17 = 47 > 46, no need update
Node 11 is Last node, finished.

Final Tables:

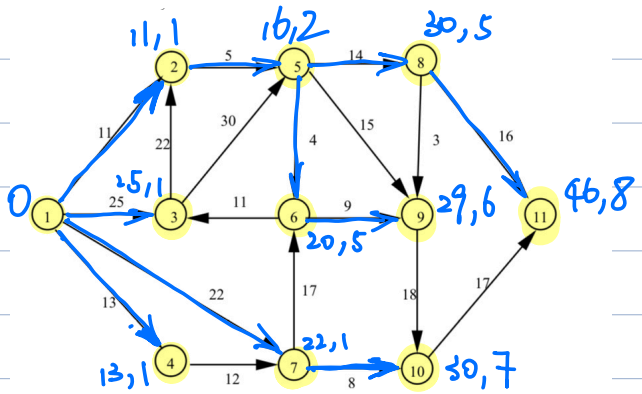
d(i) Table

Node	1	2	3	4	5	6	7	8	9	10	11
dis	0	11	25	13	16	20	22	30	29	30	46

π(i) table

Node	1	2	3	4	5	6	7	8	9	10	11
pre	-1	1	1	1	2	5	1	5	6	7	8

(ii) final tree. using Blue color



(b) Bellman-ford Algorithm.

(i)

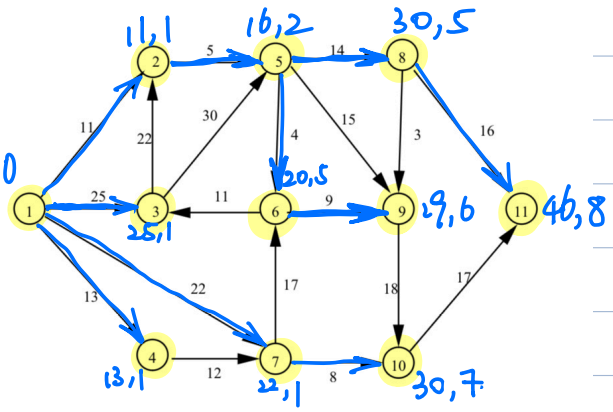
d(i) Table

Node	1	2	3	4	5	6	7	8	9	10	11
dis	0	11	25	13	16	20	22	30	29	30	46

π(i) table

Node	1	2	3	4	5	6	7	8	9	10	11
pre	-1	1	1	1	2	5	1	5	6	7	8

(ii) Final tree Blue color



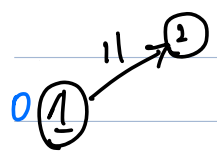
Arcs

1 2 –
1 3 –
1 7 –
1 4 –
2 5 –
3 2 –
3 5 –
4 7 –
5 8 - 5 6 - 5 9 –
6 3 –
6 9 –
7 6 –
7 10 –
8 9 - 8 11 –
9 10 –
10 11 –

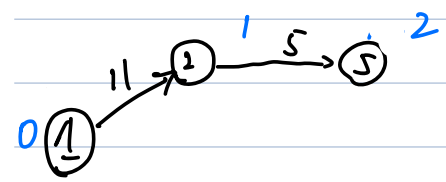
(c)

Minimum spanning tree. Prim Algorithm. Select Min Edge which be connected

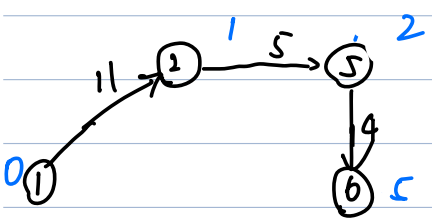
Start from node 1. {1,2}, {1,3}, {1,7}, {1,4}, choose {1,2},



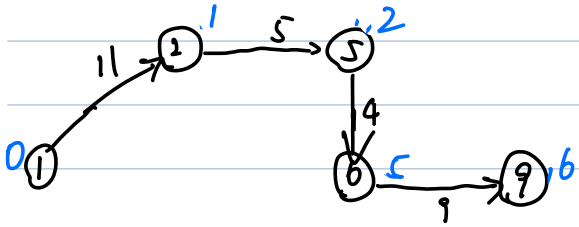
from {2,5}, {1,3}, {1,7}, {1,4}, choose {2,5}



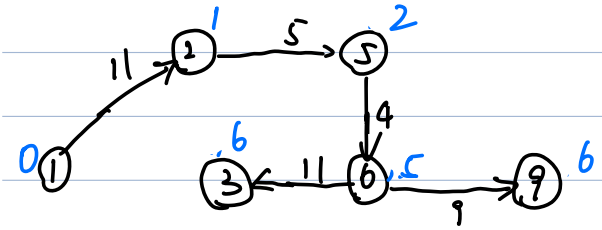
from {5,6}, {5,9}, {5,8}, {1,3}, {1,7} {1,4} choose {5,6}



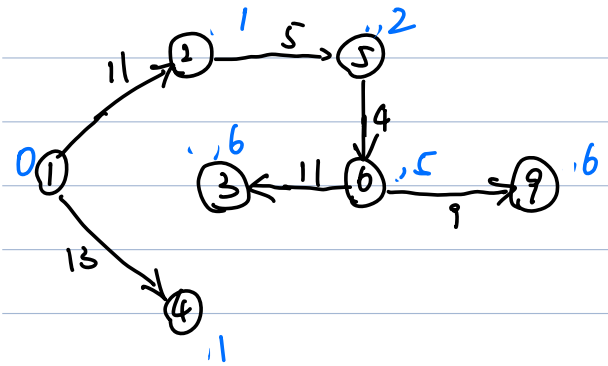
from {6,9} {6,3}, {5,9}, {5,8}, {1,3}, {1,7}, {1,6} choose {6,9}



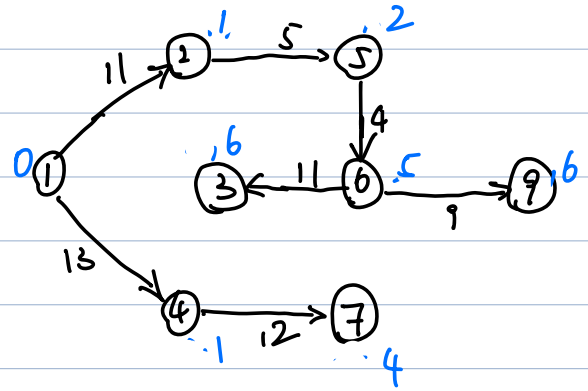
Choose {6,3}



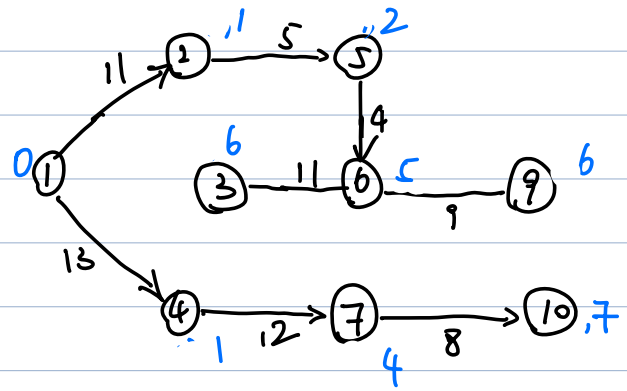
choose {1,6}



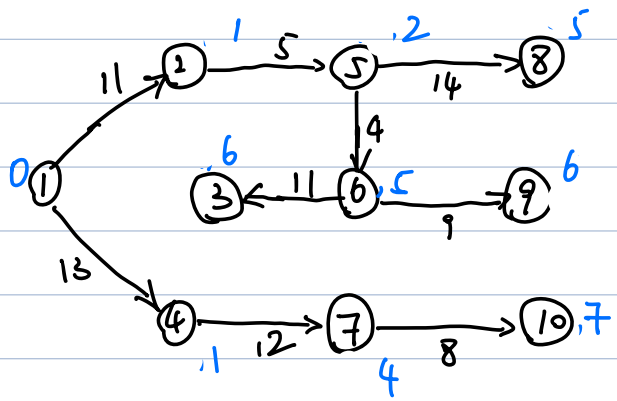
Choose {4,7}



Choose {7,10}



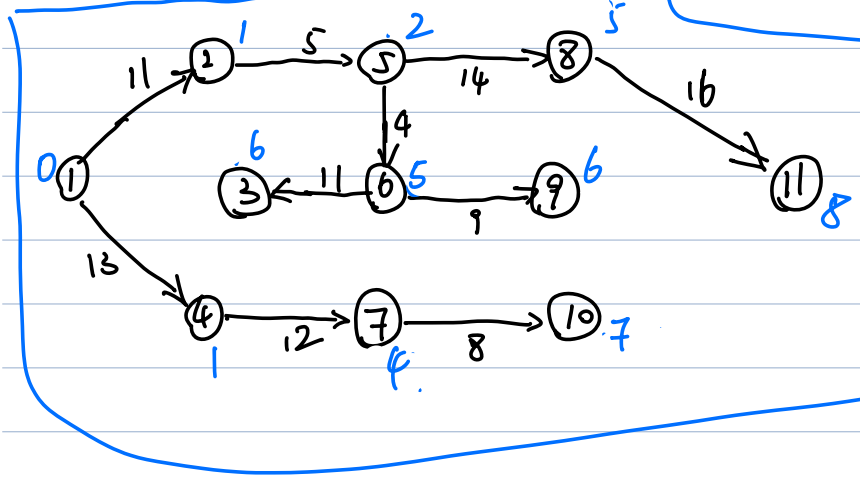
Choose {5, 8}



Choose {8, 11}

Get All vertices, finish.

Total cost = 103



(d) These final trees are not the same. The final tree for Prim's Algorithm is different from Dijkstra's algorithm and Bellman-Ford Algorithm. Prim's Algorithm is greedy algorithm, as shown above, even though each time, choose min edge from all adjacent edges, it will still miss the correct shortest path. In this example, from node 0 → node 3 is 11 cost, but from node 1 to node 3 is 25 cost. In Prim's Algorithm choose node 0 → node 3 looks correct but from overall shortest path, it will miss correct answer, because it doesn't count the previous node's cost distance.

6.

Object number	1	2	3	4	5	6
Value	10	11	12	13	14	15
Size	1	1	2	3	4	3

6. (10 pts) Consider the following knapsack problem. We have six objects with different values and Vi and sizes wi. The object values and sizes are listed in the table below:

(a) Suppose we are given a knapsack with total size 11, and you are allowed to use fractional assignments. What is the maximum value that fits in the knapsack for the above tasks?

(b) What is the maximum value of the tasks that fit entirely in the knapsack with size 11 — i.e. the integer knapsack.

(a). use the fractional assignments.

take the value / size , 10/1 , 11/1, 12/2, 13/3, 14/4, 15/3

= 10 , 11 , 6 , 4.3, 3.5, 5

Then sort the array based on the ratio. (value/ size)

ratio:	11	10	6	5	4.3	3.5
value	11	10	12	15	13	14
size	1	1	2	3	3	4

Iterate :

i = 0, size = 1 < total_size: , add value. 11 , remain 11-1 = 10

i = 1, size = 1 < t-size , add value 10 , remain 10-1 = 9

i = 2, size = 2 < t-size, add value 12, remain 9-2 = 7

i = 3, size = 3 < t-size , add value 15 , remain = 7-3 = 4

i = 4, size = 3 < t-size. add value 13 , remain = 4-3 = 1

i = 5, size = 4 > t-size , add (1/4) · 14 = 3.5 , remain = 0

So, final value with maximum is 11+10+12+15+13+3.5 = 64.5

(b) when fit entirely in the knapsack with size 11, in other words solve 0/1 knapsack problem.

total_size = 11,

V	10	11	12	13	14	15
S	1	1	2	3	4	3

V	S	I	total size 0	1	2	3	4	5	6	7	8	9	10	11
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10	1	1	0	10	10	10	10	10	10	10	10	10	10	10
11	1	2	0	11	21	21	21	21	21	21	21	21	21	21
12	2	3	0	11	21	23	33	33	33	33	33	33	33	33
13	3	4	0	11	21	23	33	34	36	46	46	46	46	46
14	4	5	0	11	21	23	33	34	36	46	47	47	47	60
15	3	6	0	11	21	23	33	36	38	48	49	51	61	62

The max value of the tasks for entirely in knapsack is 62

7.

7. (15 pts) Consider searching in the "text" $T(1:N)$ of length $N = 25$ using the KMP algorithm.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
b	a	c	b	b	a	c	b	a	b	a	c	b	a	b	a	c	b	a	c	b	a	b	a	c

for the following string of length $M = 8$ as an array $P(1:M)$ (i.e. $P(i), i = 1, 2, \cdots M$)

b	a	c	b	a	b	a	c

(a) Give the prefix function for above string: That is the value of $\pi(i)$ for $i = 1, \cdots 8$.[2] It is useful to copy the pattern in $P(1:M)$ to slide it against $P(1:M)$ After a failure at $q+1$ you can safely shift by $\pi(q)$ before starting to search again. Note overlapping finds are ok!)

(b) Find all the instances of this pattern in the text. That is give the start index in the text for the aligned pattern. (You can do this even if you have not got the right prefix function. Slide $P(1:M)$ against $T(1:N)$.)

(c) Specify all the non-trivial shift (i.e grater than 1 unit) that occurred in the scan using the KMP algorithm.

[2] Recall that the prefix function looks at the first q values of $P(1:q)$ and ask how far you can shift to match to have largest prefix match the suffix in these q terms.

$$\pi(q) = MAX\{k < q \text{ such that } P(1:k) = P(1+q-k:q)\}.$$

8

(a)

prefix function π.

P: | b | a | c | b | a | b | a | c |

π | 0 | 0 | 0 | 1 | 2 | 1 | 2 | 3 |

1 2 3 4 5 6 7 8

$\pi(1) = 0$ $\pi(5) = 2$

$\pi(2) = 0$ $\pi(6) = 1$

$\pi(3) = 0$ $\pi(7) = 2$

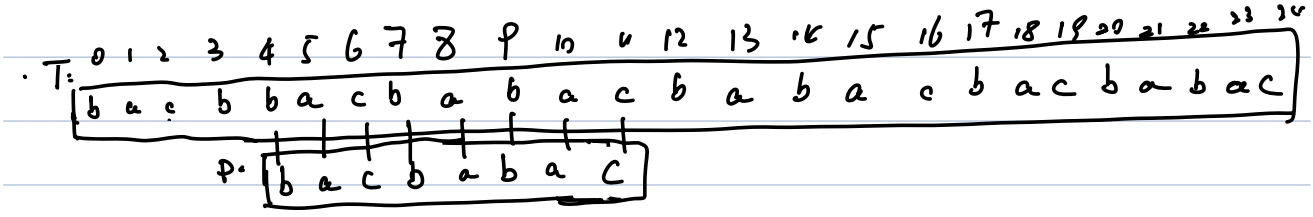
$\pi(6) = 1$ $\pi(8) = 3$

(b).

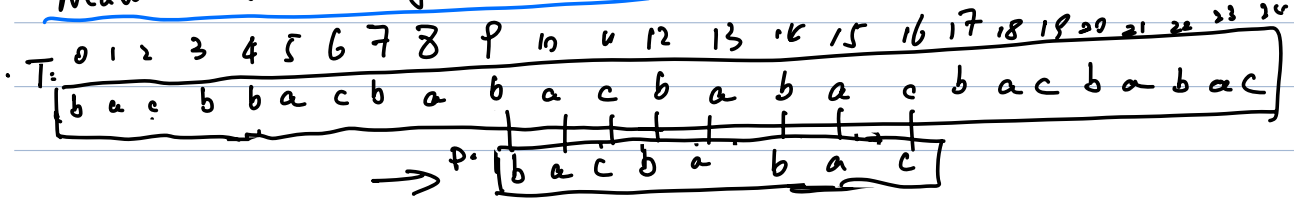
T: b a c b b a c b a b a c b a b a c b a c b a b a c

P: b a c b a b a c

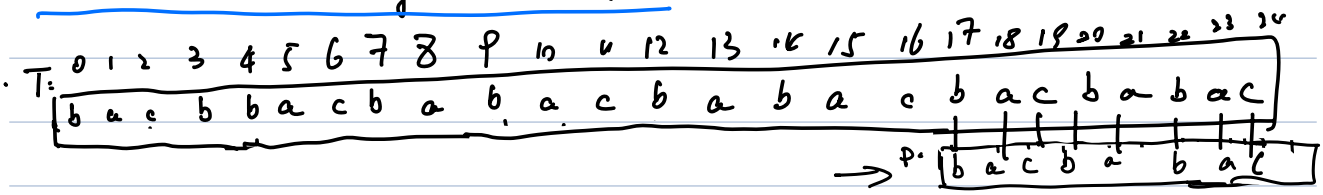
→



Matches, the starting index at 4.

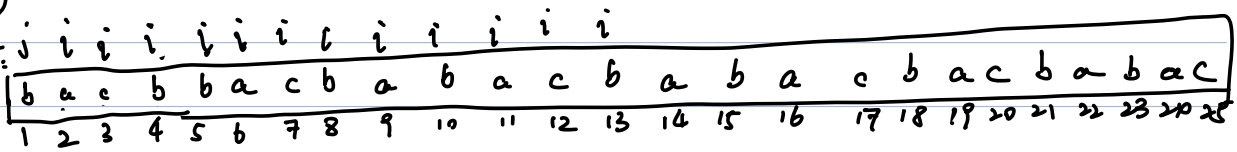


Matches, the starting index at 9



Matches, the starting index at 17

(C)



	0	1	2	3	4	5	6	7
P:	b	a	c	b	a	b	a	c
π	0	0	0	1	2	1	2	3

use two pointers i, j.

i → Text j → pattern

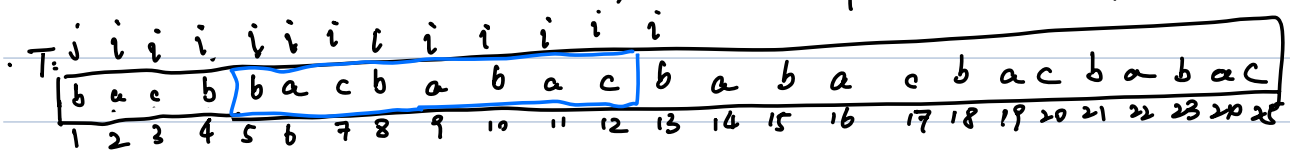
* if P[j+1] = T[i]
  j+1;
  i++;

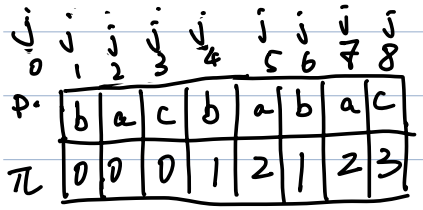
Due to KMP, we also have π function. found that bac are the longest prefix. saving time directly find bac

* if mismatch, move j to the index (shows on π[i])
  case: if j is already on index 0, and still not matching, then move i

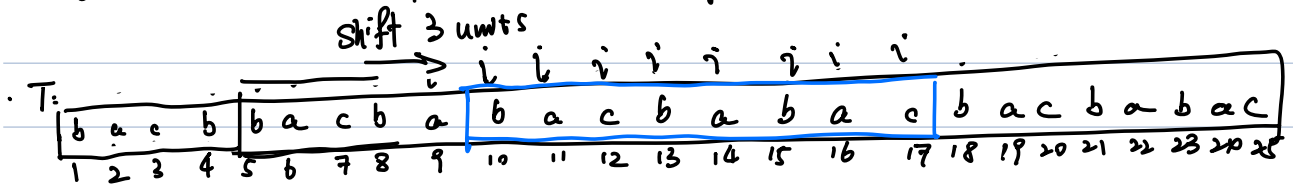
now, j ends the pattern, find one of the string match

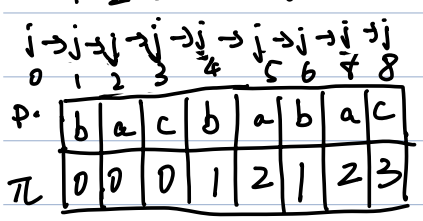
j has reached P[8], from Text, previous 8 shifts of i

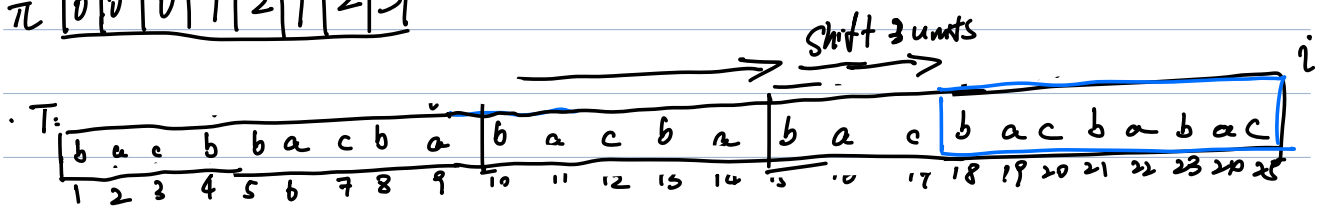


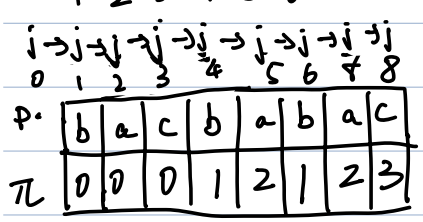


Then because not finishing iterating all characters from Text. So index j back to 0. Also from π[j], find bac are longest prefix.





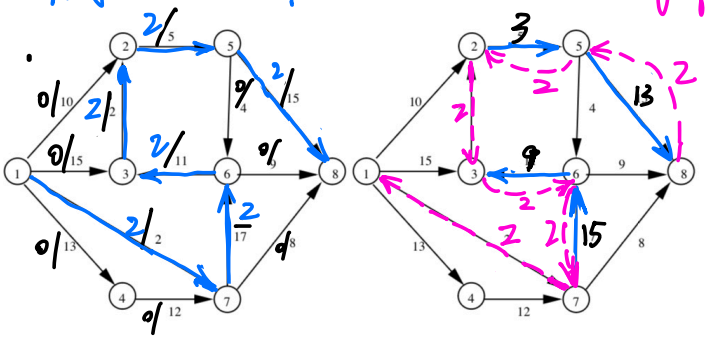




Same for third shift, look for bac.

Once i reaches end of Text away, the iteration ends

8.



(a)

Augmenting paths
$1 \to 7 \to 6 \to 3 \to 2 \to 5 \to 8$.

Bottle neck capacity.

flow Edge 1-7 and 3-2 have smallest edge, which the capacity equals 2

Figure 4:

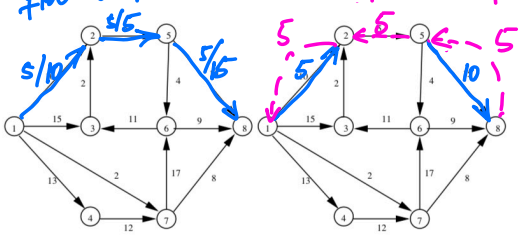
8. (15 pts) Consider Fig. 4 as a directed, capacitated graph, where the numbers on an arc now indicate an arc's capacity to carry flow from node 1 to node 8. In the max-flow algorithm of Ford and Fulkerson, the key step is, once a path has been found, to augment the flow and construct the residual graph for the next iteration.

(a) Suppose your program picks the first augmentation path to be $1 \to 7 \to 6 \to 3 \to 2 \to 5 \to 8$. Draw the augmented flow path on the left graph above and the residual graph above in the right side. What is the capacity of this path?

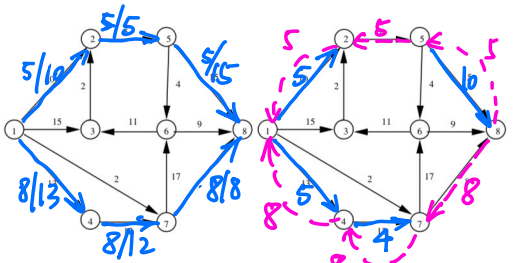
(b) Now be smarter and beginning with a min h op $1 \to 2 \to 5 \to 8$ for the first path enumerate the additional paths that bring you to maximun flow. Draw **all** flow graphs and **all** the residual and after **each** augmentation. What is the maximum flow? Draw the minimum cut to show this right.

(b)

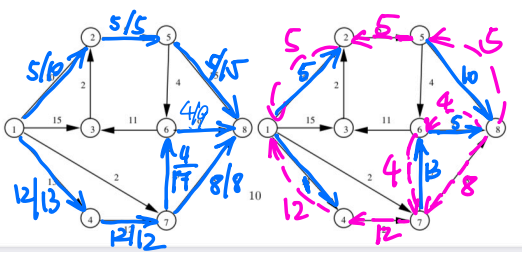
flow graphs Residual



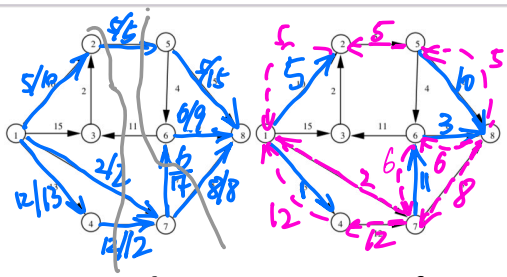
path $1 \to 4 \to 7 \to 8$ Bottle neck capacity 8



Path $1 \to 4 \to 7 \to 6 \to 8$ bottle neck capacity 4



path $1 \to 7 \to 6 \to 8$, bottle neck capacity 2



Maximum flow $5+8+6=19$