Name ______________________ Student No. ______________________

No aids allowed. Answer all questions on test paper. Use backs of sheets for scratch work. You may use any results presented in lecture. **Total marks: 50**

1. Let $G = (V, E)$ be a connected undirected graph with edges $E = \{e_1, \ldots, e_m\}$ and a cost $c(e_i) \in \mathbb{N}$ assigned to each edge e_i. Suppose that edge e_1 has a smaller cost than any of the other edges; that is $c(e_1) < c(e_i)$, for $i = 2, \ldots, m$.

[5] (a) Show that there is at least one minimum cost spanning tree for G that includes edge e_1.

Kruskal's algorithm produces a min cost spanning tree that includes the edge e_1. (Kruskal's algorithm orders the edges in non-decreasing order of costs, so e_1 is considered first and $T \cup \{e_1\} = \emptyset \cup \{e_1\} = \{e_1\}$, and $(V, \{e_1\})$ contains no cycles).

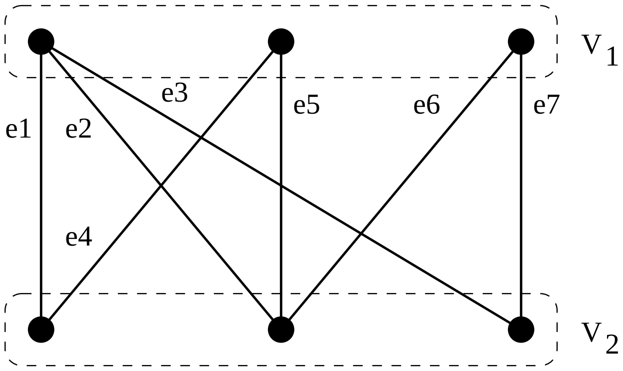
[10] (b) Use the following version of the Exchange Lemma: *if T_1 and T_2 are spanning trees for G, then for every edge $e \in T_2 - T_1$ there is an edge $e' \in T_1 - T_2$ such that $T_1 \cup \{e\} - \{e'\}$ is a spanning tree for G,* to show that every minimum cost spanning tree for G contains e_1.

Suppose that T_1 is a min cost spanning tree that does *not* contain e_1. Let T_2 be the result of running Kruskal's algorithm.

Since $e_1 \in T_2 - T_1$ ($e_1 \in T_2$ by (a)), by the above version of the Exchange Lemma, there exists an edge e_2 such that $e_2 \in T_1 - T_2$, and $T_3 = T_1 \cup \{e_1\} - \{e_2\}$ is a spanning tree. Note that $c(T_3) = c(T_1) + c(e_1) - c(e_2)$, and since $c(e_1) < c(e_2)$, it follows that $c(T_3) < c(T_1)$, contradicting the assumption that T_1 is a min cost spanning tree.

2. **Maximum Weight Matching:** Let $G = (V_1 \cup V_2, E)$ be a bipartite graph, with edge set $E \subseteq V_1 \times V_2$. Let $w : E \longrightarrow \mathbb{N}$ assign a weight $w(e) \in \mathbb{N}$ to each edge $e \in E = \{e_1, \ldots, e_m\}$. A *matching* for G is a subset $M \subseteq E$ such that no two edges in M share a common vertex. The weight of M is $w(M) = \sum_{e \in M} w(e)$.

   For example in the graph:



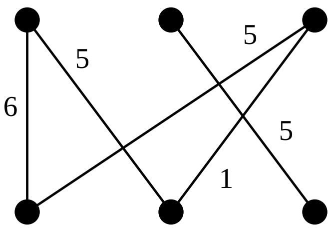
   where $w(e_1) = 8, w(e_2) = 2, w(e_3) = 16, w(e_4) = 4, w(e_5) = 1, w(e_6) = 64, w(e_7) = 32$, the matching $M = \{e_3, e_4, e_6\}$ has weight $16+4+64 = 84$. This matching is a *maximum weight matching* (i.e., M has the largest possible weight of all matchings in the graph).

[5] (a) Give at high level a simple greedy algorithm which, given a bipartite graph with edge weights, attempts to find a matching with the largest possible weight.

order the edges in non-increasing order of weights: $w(e_1) \geq w(e_2) \geq \ldots \geq w(e_m)$.

$M \longleftarrow \emptyset$
for $i : 1..m$
    if $M \cup \{e_i\}$ does *not* contain two edges with a common vertex then
        $M \longleftarrow M \cup \{e_i\}$
    end if
end for

[5] (b) Give an example of a bipartite graph with edge weights for which your algorithm in part (a) *fails* to find a matching with the largest possible weight.



max matching is $5+5+5=15$, and our greedy algorithm outputs $6+5+1=12$.

[10] (c) Now suppose all edge weights in the bipartite graph are distinct, and each is a power of 2. (Our original example satisfies these conditions.) Sketch a proof that your greedy algorithm always succeeds in finding a maximum weight matching in this case. (you will get most of the marks if you state an assertion which can be proved by induction and implies what you are to prove, as in the correctness proof of Kruskal's algorithm.)

Let M_{opt} be an optimal matching.

Define "M is promising" to mean that M can be extended to M_{opt} with edges that have not been considered yet. We show that "M is promising" is a loop invariant of our algorithm. The result will follow from this (it will also follows that there is a *unique* max matching).

Basis Case: $M=\emptyset$, so it is certainly promising.

Induction Step: Assume M is promising, and let M' be M after considering edge e_i. We show that:

$$e_i \in M' \text{ iff } e_i \in M_{\text{opt}}$$

"$\Longrightarrow$" $e_i \in M'$, since the weights are distinct, and powers of 2:

$$w(e_i) > \sum_{j=i+1}^{m} w(e_i)$$

so unless $e_i \in M_{\text{opt}}$, $w(M_{\text{opt}}) < w(\text{result of algorithm})$.

"$\Longleftarrow$" $e_i \in M_{\text{opt}}$, so $M \cup \{e_i\}$ has no conflict, so the algorithm would add it.

[15] 3. Recall the (General) Knapsack Problem:

Input: $w_1, \dots, w_d, v_1, \dots, v_d, C \in \mathbb{N}$

Output: $M = \max_{S \subseteq \{1,\dots,d\}} \{V(S) | K(S) \leq C\}$, $K(S) = \sum_{i \in S} w_i$, and $V(S) = \sum_{i \in S} v_i$.

To give a dynamic programming solution, we first define a boolean array $R(i,j)$ for $0 \leq i \leq d, 0 \leq j \leq C$, as follows:

$$R(i,j) = \begin{cases} \mathsf{True} & \text{if } \exists S \subseteq \{1,\dots,i\} \text{ such that } K(S) = j \\ \mathsf{False} & \text{otherwise} \end{cases}$$

Now we define the array $V(i,j)$ for $0 \leq i \leq d, 0 \leq j \leq C$ as follows:

$$V(i,j) = \max_{S \subseteq \{1,\dots,i\}} \{V(S) | K(S) = j\}$$

Give the recurrence (including initial conditions) for computing $V(i,j)$. Assume the array $R(i,j)$ has already been computed.

$V(i,j) = 0$ if $i = 0$ or $j = 0$.

$$V(i,j) = \begin{cases} V(i-1,j) & \text{if } j < w_i \text{ or } R(i-1, j-w_i) = \mathsf{F} \\ \max\{v_i + V(i-1, j-w_i), V(i-1,j)\} & \text{otherwise} \end{cases}$$

Briefly justify your recurrence:

Suppose that $j < w_i$. Then weight i cannot be included, so $V(i,j) = V(i-1,j)$. If $R(i-1, j-w_i) = \mathsf{F}$, then there is no subset $S \subseteq \{1,\dots,i\}$ such that $i \in S$ and $K(S) = j$, so again weight i is not included, and $V(i,j) = V(i-1,j)$.

Otherwise, if $j \geq w_i$ and $R(i-1, j-w_i) = \mathsf{T}$, then weight i may or may not be included in S. We take the case which offers more value: $\max\{v_i + V(i-1, j-w_i), V(i-1,j)\}$.

Name ______________________ Student No. ______________________

No aids allowed. Answer all questions on test paper. Use backs of sheets for scratch work. You may use any results presented in lecture. **Total Marks: 50**

[10] 1. The Simple Knapsack Problem (SKS) takes as input: $d, w_1, w_2, \ldots, w_d, C \in \mathbb{N}$ and requires as output a set $S_0 \subseteq \{1, \ldots, d\}$ such that $K(S_0) = M$, where:

$$M = \max_{S \subseteq \{1,\ldots,d\}} \{K(S) | K(S) \leq C\}$$

and $K(S) = \sum_{i \in S} w_i$. Recall that the greedy method for SKS produces an answer $\overline{M}$ such that $\overline{M} \leq C$ but possibly $\overline{M} < M$. **Prove the following lemma:**

Lemma: Either $\overline{M} = M$, or $\overline{M} > \frac{1}{2}C$.

Proof of Lemma: Assume that $\overline{M} \neq M$, and let S_0 be the result of greedy, i.e. a subset of $\{1, \ldots, d\}$ such that $K(S_0) = \overline{M}$. First show that there is at least one weight in S_0:

If $S_0 = \emptyset$, then $\overline{M} = 0$, and all weights must be larger than C, but then $M = 0$, so $\overline{M} = M$ which is not the case by assumption.

Now show that there is at least one weight not in S_0:

If all the weights are in S, then $\overline{M} = \sum_{i=1}^{d} w_i = M$.

Finally, show now that $\overline{M} > \frac{1}{2}C$ by considering the first weight, call it w_j, which has been rejected after at least one weight has been added:

If $w_j \leq \frac{1}{2}C$, then the sum of the weights which are already in is $> \frac{1}{2}C$, so $\overline{M} > \frac{1}{2}C$. If $w_j > \frac{1}{2}C$, then, since the objects are ordered by greedy in non-increasing order of weights, the weights that are already in are $> \frac{1}{2}C$, so again $\overline{M} > \frac{1}{2}C$.

2. Define the problem **Dispersed Knapsack** as follows:

Input: $w_1, \ldots, w_d, C$, all positive integers, such that the w_i's satisfy the following condition:

$$w_i \geq \sum_{j=i+1}^{d} w_j, \quad \text{for } i = 1, \ldots, d-1$$

Output: $S_{\max} \subseteq \{1, \ldots, d\}$ such that $K(S_{\max}) = \max_{S \subseteq \{1,\ldots,d\}} \{K(S) | K(S) \leq C\}$. Recall that $K(S) = \sum_{i \in S} w_i$.

[5] (a) Give a greedy algorithm which solves Dispersed Knapsack by filling in the following:

$S \longleftarrow \emptyset$

for $i : 1..d$

if $\underline{w_i + \sum_{j \in S} w_j \leq C}$ then

$\underline{S \longleftarrow S \cup \{i\}}$

end if

end for

[5] (b) Give a definition of what it means for an intermediate solution S in the above algorithm to be "promising". Show that the loop invariant "S is promising" implies that the greedy algorithm gives the optimal solution.

Define "S is promising" to mean that S can be extended, using weights which have *not* been considered yet, to an optimal solutions $S_{\max}$.

At the end, when no more weights have been left to consider, the loop invariant still holds true, so S itself must be optimal.

[15] (c) Show that "S is promising" (as defined in part (b)) is a loop invariant of the greedy algorithm in part (a).

We give a proof by induction on the number of iterations.

Basis Case: $S = \emptyset$, so S is clearly promising.

Induction Step: Suppose that S is promising (so S can be extended, using weights which have not been considered yet, to $S_{\max}$). Let S' be S after one more iteration.

Suppose $i \in S'$. Since $w_i \geq \sum_{j=i+1}^{d} w_j$, it follows that:

$$K(S') \geq K(S) + \sum_{j=i+1}^{d} w_j \geq K(S_{\max})$$

so S' is already maximal, and we are done.

Suppose $i \notin S'$. Then $w_i + \sum_{j \in S} w_j > C$, so $i \notin S_{\max}$, so S' can be extended (using weights which have not been considered yet!) to $S_{\max}$.

In either case, S' is promising.

3. Consider the **Consecutive Subsequence Sum Problem:**
Input: Real numbers $r_1, \dots, r_n$ (for some $n \in \mathbb{N}$).
Output: For each consecutive subsequence of the form $r_i, r_{i+1}, \dots, r_j$ let

$$S_{ij} = r_i + r_{i+1} + \cdots + r_j$$

where $S_{ii} = r_i$. Find $M = \max_{1 \le i \le j \le n} S_{ij}$.

For example, if $n = 7$ and the input sequence is $1, -5, 3, -1, 2, -8, 3$, then $M = S_{35} = 3 + (-1) + 2 = 4$.

This problem can be solved in time $O(n^2)$ by systematically computing all of the sums S_{ij} and finding the maximum. However, there is a more efficient dynamic programming solution which runs in time $O(n)$.

Define the array $M(1..n)$ by:

$$M(j) = \max\{S_{1j}, S_{2j}, \dots, S_{jj}\}$$

In the above example:

$$\begin{array}{lcrrrrrrr} j & = & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ r_j & = & 1 & -5 & 3 & -1 & 2 & -8 & 3 \\ M(j) & = & 1 & -4 & 3 & 2 & 4 & -4 & 3 \end{array}$$

[5] (a) Explain how to find the solution M from the array $M(1..n)$.

$M = \max_{1 \le j \le n} M(j)$

[10] (b) Complete the four lines indicated in the program below for computing the values of the array $M(1..n)$, given $r_1, r_2, \dots, r_n$.

M(1) $\longleftarrow$ $\underline{r_1 (= S_{11})}$
for j:2..n
    if $\underline{M(j-1) > 0}$
    then
        M(j) $\longleftarrow$ $\underline{M(j-1) + r_j}$
    else
        M(j) $\longleftarrow$ $\underline{r_j}$
    end if
end for